O'REILLY®



# Разработка приложений на базе больших языковых моделей

## Целостный подход к LLM

Сухас Пай

# Designing Large Language Model Applications

## *A Holistic Approach to LLMs*

*Suhas Pai*

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY

Сухас Пай

# Разработка приложений на базе больших языковых моделей

## Целостный подход к LLM

Астана
«АЛИСТ»
2026

УДК 004.89
ББК 32.973.26-018.2
П12

**Пай С.**

П12 Разработка приложений на базе больших языковых моделей: пер. с англ. — Астана: АЛИСТ, 2026. — 400 с.: ил.

ISBN 978-601-12-9092-0

Книга посвящена практическим аспектам проектирования и разработки приложений на основе больших языковых моделей (LLM). Рассматриваются внутреннее устройство и архитектура трансформеров, методы подготовки данных для обучения и дообучения, а также подходы к адаптации предобученных моделей под конкретные задачи и предметные области. Подробно разобраны техники тонкой настройки, доменной адаптации и оптимизации вывода, а также современные парадигмы применения LLM — генерация с дополненной выборкой (RAG), интеллектуальные агенты и интеграция моделей с внешними инструментами и данными. Особое внимание уделено типичным проблемам и отказам, возникающим при переходе от прототипов к полноценным продуктам, и практическим рекомендациям по преодолению подобных ситуаций.

*Для специалистов по ИИ,*
*разработчиков, технических руководителей*

УДК 004.89
ББК 32.973.26-018.2

© 2026 ALIST LLP

Authorized Russian translation of the English edition of ***Designing Large Language Model Applications***
ISBN 9781098150501 © 2025 Suhas Pai.
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Авторизованный перевод с английского языка на русский издания ***Designing Large Language Model Applications***
ISBN 9781098150501 © 2025 Suhas Pai.
Перевод опубликован и продается с разрешения компании-правообладателя O'Reilly Media, Inc.

ISBN 978-1-098-15050-1 (англ.)
ISBN 978-601-12-9092-0 (каз.)

© Suhas Pai, 2025
© Издание на русском языке. ТОО "АЛИСТ", 2026

# Содержание

**Отзывы о книге "Разработка приложений на базе больших языковых моделей"** ..... **11**

**Предисловие** ..... **14**

Кому предназначена книга ..... 15
Структура книги ..... 15
Что не вошло в книгу ..... 16
Как читать книгу ..... 16
Условные обозначения и соглашения ..... 17
Примеры применения кода ..... 18
Платформа онлайн-обучения O'Reilly ..... 18
Контактная информация ..... 18
Благодарности ..... 19

**ЧАСТЬ I. КОМПОНЕНТЫ LLM** ..... **21**

**ГЛАВА 1. Вступление** ..... **23**

Определение больших языковых моделей ..... 24
Краткая история больших языковых моделей ..... 29
    Ранний период ..... 29
    Современная эпоха LLM ..... 31
Влияние больших языковых моделей ..... 33
Внедрение больших языковых моделей в корпоративную среду ..... 35
Промптинг ..... 37
    Промптинг без примеров ..... 39
    Промптинг с несколькими примерами ..... 40
    Промптинг с цепочкой рассуждений ..... 41
    Создание цепочек промптов ..... 42
    Состязательный промптинг ..... 43
Доступ к LLM через API ..... 44
Преимущества и недостатки больших языковых моделей ..... 47
Разработка прототипа чат-бота ..... 50
От прототипа к производству ..... 54
Резюме ..... 56

**ГЛАВА 2. ДАННЫЕ ДЛЯ ПРЕДВАРИТЕЛЬНОГО ОБУЧЕНИЯ ..... 57**

Компоненты большой языковой модели ..... 57
Требования к данным для предобучения ..... 61
Популярные наборы предобучающих данных ..... 64
Синтетические предобучающие данные ..... 70
Предобработка обучающих данных ..... 71
Фильтрация и очистка данных ..... 72
Отбор документов по критериям качества ..... 77
Дедупликация ..... 81
Удаление персонально идентифицируемой информации ..... 83
Деконтаминация обучающего набора ..... 90
Смеси данных ..... 91
Влияние предобучающих данных на последующие задачи ..... 93
Проблемы предвзятости и справедливости в предобучающих наборах данных ..... 94
Резюме ..... 96

**ГЛАВА 3. СЛОВАРЬ И ТОКЕНИЗАТОР ..... 97**

Словарь ..... 97
Токенизатор ..... 102
Конвейер токенизации ..... 106
Нормализация ..... 107
Предварительная токенизация ..... 107
Токенизация ..... 108
Кодирование пар байтов ..... 108
Алгоритм WordPiece ..... 110
Специальные токены ..... 112
Резюме ..... 114

**ГЛАВА 4. АРХИТЕКТУРА И ЦЕЛЬ ОБУЧЕНИЯ ..... 115**

Предварительные сведения ..... 115
Представление значения слова ..... 117
Архитектура трансформера ..... 119
Механизм внутреннего внимания ..... 122
Позиционное кодирование ..... 124
Сеть прямого распространения ..... 125
Нормализация слоев ..... 126
Функция потерь ..... 127
Внутренняя оценка модели ..... 128
Архитектурные магистрали трансформера ..... 128
Архитектура "только с кодировщиком" ..... 130
Архитектура "кодер-декодер" ..... 131
Архитектура "только с декодером" ..... 131
Смесь экспертов ..... 131

Цели обучения ... 134
Полнотекстовое языковое моделирование ... 135
Префиксное языковое моделирование ... 139
Маскированное языковое моделирование ... 139
Наиболее эффективные цели обучения ... 142
Предварительное обучение моделей ... 144
Резюме ... 146

**Часть II. Применение LLM ... 147**

**Глава 5. Адаптация LLM к конкретным сценариям применения ... 149**

Навигация по ландшафту LLM ... 149
Провайдеры больших языковых моделей ... 149
Разновидности моделей ... 152
Большие языковые модели с открытым исходным кодом ... 156
Выбор LLM для конкретной задачи ... 160
Сравнение открытых и проприетарных языковых моделей ... 162
Оценка больших языковых моделей ... 163
Загрузка моделей ... 171
Библиотека Hugging Face Accelerate ... 172
Библиотека Ollama ... 172
Интерфейсы для логического вывода больших языковых моделей ... 174
Стратегии декодирования ... 174
Жадное декодирование ... 174
Лучевой поиск ... 175
Семплирование top-*k* ... 176
Семплирование top-*p* ... 176
Инференс больших языковых моделей ... 177
Структурированные выходные данные ... 179
Отладка и интерпретируемость модели ... 181
Резюме ... 183

**Глава 6. Дообучение ... 184**

Актуальность дообучения ... 184
Дообучение: подробный пример ... 185
Параметры обучающих алгоритмов ... 187
Параметры оптимизации памяти ... 192
Параметры регуляризации ... 193
Размер пакета данных ... 195
Эффективная тонкая настройка параметров ... 197
Работа с пониженной точностью ... 197
Сводим все воедино ... 198

Дообучение наборов данных ... 201
Обучение следовать инструкциям на публичных инструктивных наборах данных ... 203
Наборы данных для обучения следовать инструкциям, генерируемые LLM ... 209
Резюме ... 212

**ГЛАВА 7. Продвинутые подходы к дообучению ... 213**

Непрерывное предобучение ... 214
Воспроизведение (запоминание) ... 217
Расширение параметров ... 218
Эффективная тонкая настройка параметров ... 220
Добавление новых параметров ... 220
Методы выбора подмножества параметров ... 226
Объединение нескольких моделей ... 227
Ансамблирование моделей ... 227
Объединение моделей ... 229
Слияние адаптеров ... 231
Резюме ... 231

**ГЛАВА 8. Обучение на согласование и развитие навыков рассуждения ... 232**

Определение обучения на согласование ... 232
Обучение с подкреплением ... 233
Типы обратной связи от человека ... 234
Пример RLHF ... 234
Галлюцинации ... 236
Смягчение галлюцинаций ... 238
Метод самосогласованности ... 240
Цепочка действий ... 240
Рецитирование ... 242
Методы семплирования для борьбы с галлюцинациями ... 242
Декодирование с использованием контрастных слоев ... 243
Контекстные галлюцинации ... 245
Галлюцинации, вызванные неактуальной информацией ... 247
Рассуждение ... 248
Дедуктивное рассуждение ... 249
Индуктивное рассуждение ... 249
Абдуктивное рассуждение ... 249
Рассуждения, основанные на здравом смысле ... 250
Активизация процесса рассуждения в языковых моделях ... 251
Повышение качества рассуждений с помощью верификаторов ... 251
Вычисление времени вывода ... 252
Дообучение рассуждениям ... 254
Резюме ... 255

**ГЛАВА 9. ОПТИМИЗАЦИЯ ВЫВОДА МОДЕЛИ ..... 256**

Проблемы вывода моделей ..... 256
Методы оптимизации вывода ..... 257
Методы оптимизации вычислений ..... 257
K-V-кеширование ..... 258
Метод раннего выхода ..... 260
Методы ускорения декодирования ..... 266
Спекулятивное декодирование ..... 267
Параллельное декодирование ..... 268
Методы снижения требований к хранению данных ..... 269
Симметричное квантование ..... 270
Асимметричное квантование ..... 271
Резюме ..... 272

**ЧАСТЬ III. ПАРАДИГМЫ ПРИЛОЖЕНИЙ НА ОСНОВЕ LLM ..... 273**

**ГЛАВА 10. ВЗАИМОДЕЙСТВИЕ LLM С ВНЕШНИМИ ИНСТРУМЕНТАМИ ..... 275**

Парадигмы взаимодействия LLM ..... 276
Метод пассивного взаимодействия ..... 277
Метод явного взаимодействия ..... 278
Метод автономной работы ..... 279
Определение агента ..... 280
Рабочий процесс агентской системы ..... 282
Компоненты агентской системы ..... 283
Модели ..... 284
Инструменты ..... 285
Хранилища данных ..... 291
Промпт цикла агента ..... 295
Защитные ограждения и верификаторы ..... 297
Программное обеспечение для оркестрации агентов ..... 305
Резюме ..... 307

**ГЛАВА 11. ОБУЧЕНИЕ ПРЕДСТАВЛЕНИЯМ И ЭМБЕДДИНГИ ..... 308**

Введение в эмбеддинги ..... 309
Семантический поиск ..... 311
Метрики сходства ..... 313
Дообучение моделей эмбеддингов ..... 316
Базовые модели ..... 316
Набор обучающих данных ..... 316
Функции потерь ..... 318
Эмбеддинги инструкций ..... 320
Оптимизация размера эмбеддингов ..... 322
Эмбеддинги "Матрешка" ..... 323
Двоичные и целочисленные эмбеддинги ..... 324
Метод Product Quantization ..... 325

Чанкинг ... 326
Метод скользящего окна ... 328
Чанкинг с учетом метаданных ... 329
Чанкинг с учетом компоновки ... 329
Семантический чанкинг ... 329
Метод позднего чанкинга ... 330
Векторные базы данных ... 331
Интерпретация эмбеддингов ... 333
Резюме ... 334

**ГЛАВА 12. Генерация с дополненным поиском ... 335**

Необходимость использования RAG ... 335
Типичные сценарии использования RAG ... 337
Определение момента для выполнения извлечения информации ... 338
Конвейер RAG ... 339
Перезапись ... 342
Извлечение ... 346
Реранкинг ... 351
Уточнение ... 356
Вставка ... 362
Генерация ... 363
Управление памятью с помощью RAG ... 366
Выбор примеров контекстного обучения с помощью RAG ... 368
Обучение модели с помощью RAG ... 369
Ограничения метода RAG ... 370
RAG и длинный контекст ... 371
RAG и дообучение ... 373
Резюме ... 374

**ГЛАВА 13. Шаблоны дизайна и системная архитектура ... 375**

Архитектура с использованием нескольких LLM ... 375
Каскадная архитектура LLM ... 376
Маршрутизатор ... 378
LLM, предназначенные для решения конкретных задач ... 379
Парадигмы программирования ... 380
DSPy ... 380
LMQL ... 383
Резюме ... 384

**Предметный указатель ... 385**

**Об авторе ... 398**

**Об изображении на обложке ... 399**

# Отзывы о книге "Разработка приложений на базе больших языковых моделей"

Книга "Разработка приложений на базе больших языковых моделей" — это настоящий учебник по созданию передовых систем искусственного интеллекта. Она погружает читателя в мир современных методик, таких как инструменты для работы с данными, методы рассуждения, RAG-архитектуры и дообучение, предоставляя все необходимое для разработки инновационных приложений искусственного интеллекта.

*Джей Аламмар (Jay Alammar),*
*соавтор книги "Большие языковые модели на практике"*
*("Hands-On Large Language Models")*

Книга "Разработка приложений на базе больших языковых моделей" представляет собой исчерпывающий обзор LLM, который предлагает ясные объяснения ключевых концепций, таких как промптинг и дообучение или тонкая настройка, а также рассматривает новые тенденции, включая вычисления и рассуждения в процессе вывода. При этом, что особенно важно, у читателя появится глубокое понимание поведения этих моделей в реальных условиях. Практические упражнения помогают укрепить это понимание в увлекательной и творческой манере, что делает книгу не просто бесценным справочником, но и инструментом для инженеров-программистов, специалистов в области машинного обучения и менеджеров по продуктам для создания собственных решений на основе LLM.

*Меган Рисдал (Megan Risdal),*
*ведущий менеджер по продуктам, Kaggle (Google)*

Книга "Разработка приложений на базе больших языковых моделей" представляет собой подробное и актуальное руководство, охватывающее концепции и методы, лежащие в основе исследования, разработки и создания приложений для LLM. Автор, основываясь на своем обширном опыте в области инженерии и научных ис-

следований, предлагает четкие объяснения и практические рекомендации по различным темам, актуальным как в научной, так и в коммерческой среде. Книга дополнена ценными ссылками на предшествующие работы и инструменты. Тщательно подобранные упражнения способствуют развитию интуиции и экспериментальных навыков у читателей, что делает книгу уникальной и продуманной. Она охватывает все важные идеи и практические знания, необходимые в этой области, и является незаменимым помощником для всех, кто хочет стать экспертом в этой быстро развивающейся сфере.

*Мадхав Сингхал (Madhav Singhal),*
*генеральный директор AutoComputer*

Сухас, опираясь на свой богатый опыт, предлагает читателям всесторонний обзор основ и самых современных методов, которые уже доказали свою эффективность на практике. Эта практическая книга, написанная в самое подходящее время, станет незаменимым помощником для целого нового поколения разработчиков LLM.

*Сьюзен Шу Чанг (Susan Shu Chang),*
*ведущий специалист по анализу данных, Elastic*

Книга поражает своей многогранностью!

*Нур Фахми (Nour Fahmy),*
*Flagship RTL*

*Выражаю благодарность легенде Кусуме Пай за то,*
*что она показала мне, как мечтать.*

# Предисловие

Последние годы ознаменовались стремительным развитием искусственного интеллекта (ИИ — artificial intelligence, AI), особенно в сфере больших языковых моделей (large language model, LLM). Если еще совсем недавно LLM представляли собой лишь перспективную технологию, только осваивающую генерацию последовательных абзацев, то сегодня они демонстрируют поразительные способности — от решения сложных математических задач до написания убедительных эссе и ведения увлекательных и глубоких диалогов с людьми.

С каждым днем все более мощные инструменты ИИ проникают в нашу жизнь, оказывая влияние на множество ее аспектов. Понимание того, как эффективно использовать такие ИИ-модели, как LLM, является, пожалуй, одним из ключевых навыков текущего десятилетия. Большие языковые модели меняют мир программного обеспечения, открывая возможности для создания приложений, которые ранее казались нереалистичными.

Несмотря на впечатляющие возможности, LLM все еще далеки от совершенства и имеют ряд существенных недостатков. К ним относятся проблемы с логическим рассуждением, склонность к искажению фактов, генерация неверной информации (галлюцинации, hallucinations), трудности в соблюдении поставленных целей, а также вопросы предвзятости и объективности. Но даже с учетом имеющихся ограничений, потенциал больших языковых моделей для полезного применения в различных областях и создания широкого спектра приложений остается весьма велик при условии, что мы найдем эффективные методы для преодоления их недостатков.

Хотя существует множество инструментов для быстрого создания прототипов приложений с помощью LLM, превращение концепции в полноценное рабочее приложение остается непростой задачей. Данная книга призвана преодолеть этот барьер, предлагая глубокое погружение в мир больших языковых моделей и предоставляя как теоретические основы, так и практические инструменты для разработки масштабных приложений на основе LLM.

Главная цель книги — помочь читателю обрести интуитивное понимание принципов функционирования LLM, а также освоить доступные инструменты для их работы и различные стратегии их применения. Книга отличается обилием практических

заданий (более 80 упражнений), которые станут ценным инструментом для закрепления полученных знаний и более глубокого понимания внутренних механизмов LLM. При написании книги было проанализировано более 800 научных исследований, ссылки на которые приводятся в тексте для более глубокого погружения в рассматриваемую тематику. Несмотря ни на что, я уверен, после прочтения книги в полном объеме, выполнения всех упражнений и изучения предложенных ссылок вы достигнете экспертного уровня понимания LLM.

## Кому предназначена книга

Книга рассчитана на широкую аудиторию, включая разработчиков программного обеспечения, переходящих к созданию приложений ИИ, специалистов по машинному обучению, как практиков, так и исследователей, а также менеджеров по продуктам. Бóльшая часть информации в книге основана на моих личных экспериментах с LLM, и я уверен, что даже будучи опытным исследователем в этой области, вы почерпнете из нее что-то новое и полезное для себя. Вместе с тем, если ваш путь в мире ИИ только начинается, эта книга станет для вас ценным ресурсом, предоставляя исчерпывающие сведения об основах этой технологии.

Для того чтобы разобраться с содержанием книги, достаточно иметь базовые знания Python и понимание основных концепций машинного и глубокого обучения. При необходимости я даю ссылки на дополнительные материалы, которые помогут вам углубить или расширить свои знания в этих областях.

## Структура книги

Книга состоит из тринадцати глав и разделена на три части. Первая часть посвящена разбору компонентов языковой модели. Я убежден, что даже если вы не сможете самостоятельно создать языковую модель с нуля, понимание ее внутренней структуры является ключевым фактором для успешного взаимодействия с ней. Во второй части рассматриваются различные подходы к использованию языковых моделей, начиная от прямого запроса к модели и заканчивая ее дообучением или тонкой настройкой для конкретных задач. В этой части также обсуждаются ограничения больших языковых моделей, такие как галлюцинации и проблемы с логическими рассуждениями, и показаны различные подходы к их минимизации. В заключительной части представлены парадигмы использования больших языковых моделей, в частности, в генерации с дополненным поиском (retrieval augmented generation, RAG) и агентах, интегрирующих LLM в более широкую систему программного обеспечения.

Для более детального ознакомления с оглавлением посетите мой блог на Substack (**https://oreil.ly/-2zkH**).

## Что не вошло в книгу

В целях сокращения объема книги ряд тем пришлось исключить из обсуждения. Я старался не затрагивать вопросы, которые, по моему мнению, не выдержат испытания временем. Эта сфера развивается довольно стремительно, поэтому создание книги, которая сохранит свою актуальность в долгосрочной перспективе, — задача крайне сложная.

В частности, книга фокусируется исключительно на англоязычных языковых моделях (English-language LLM), оставляя за рамками обсуждение многоязычных (multilingual) моделей. Кроме того, я не разделяю точку зрения о том, что все неанглоязычные языки мира можно свести под общее понятие "многоязычные". Каждый язык обладает своими уникальными особенностями и заслуживает отдельного, посвященного ему исследования.

В книге также не рассматриваются мультимодальные модели (multimodal language). Несмотря на то что новые модели все чаще становятся мультимодальными, интегрируя поддержку различных форматов, включая текст, изображение, видео и речь, текст по-прежнему играет ключевую роль, выступая как объединяющая основа в таких моделях. Тем не менее материал книги может быть ценным подспорьем в подготовке к мультимодальному будущему.

Книга не затрагивает теоретические аспекты и не углубляется в математику. Для более детального изучения этих тем существуют другие издания, на которые я с удовольствием буду ссылаться при необходимости. В книге математические формулы сведены к минимуму, а основное внимание уделено формированию интуитивного понимания.

Кроме того, книга лишь кратко знакомит читателя с моделями рассуждений (reasoning model) — новейшей парадигмой LLM. На момент выхода книги модели рассуждений находятся в зачаточном состоянии, и научное сообщество до сих пор не пришло к единому мнению о том, какие направления развития в итоге окажутся наиболее перспективными.

## Как читать книгу

Для наиболее эффективного освоения материала книги рекомендуется последовательное чтение с выполнением упражнений и обращением к справочным ссылкам. Однако, в зависимости от ваших интересов, существуют и альтернативные подходы.

- Если вас интересует мир LLM, но вы не собираетесь заниматься разработкой приложений на их основе, рекомендуется обратиться к *главам 1, 2, 3, 4, 5, 10, 11*.
- Ценную информацию для менеджеров по продукту, заинтересованных в изучении потенциала приложений LLM, содержат *главы 1, 2, 3, 5, 8, 10, 11, 12, 13*.
- Актуальную информацию для специалистов в области машинного обучения можно найти в *главах 7, 8, 9, 10, 11, 12*.

- Для обучения собственной LLM с нуля следует обратиться к *главам 2, 3, 4, 5, 7*, в которых изложены базовые принципы.

## Условные обозначения и соглашения

В книге используются следующие общепринятые типографские обозначения.

*Курсив*

Обозначает новые термины, имена файлов и каталогов, расширения файлов.

**Жирный шрифт**

Обозначает URL-адреса, адреса электронной почты, элементы интерфейса программ.

`Моноширинный шрифт`

Используется в листингах программ, а также внутри абзацев для обозначения таких элементов программы, как имена переменных и функций, базы данных, типы данных, переменные среды, операторы, ключевые слова.

**`Жирный моноширинный шрифт`**

Обозначает команды или иной текст, который пользователь должен набирать вручную, а также зарезервированные слова языка программирования.

*`Курсивный моноширинный шрифт`*

Выделяет текст, который необходимо заменить на значения, введенные пользователем, или определяемые контекстом, а также комментарии в программном коде.



Данный элемент обозначает подсказку или совет.



Данный элемент обозначает общее замечание.



Данный элемент обозначает предупреждение или предостережение.

## Примеры применения кода

Дополнительные материалы (примеры кода, упражнения и т. д.) доступны для скачивания по адресу **https://oreil.ly/llm-playbooks**.

В случае технических вопросов или проблем с использованием примеров кода, пожалуйста, отправьте письмо по адресу **support@oreilly.com**.

Книга предназначена для практической работы. В большинстве случаев можно свободно пользоваться кодами, приведенными в ней в качестве примеров, в своих программах и документации. Для воспроизведения незначительной части кода не требуется разрешение. Например, в тех случаях, когда вы пишите программу, используя несколько фрагментов кода из книги, или отвечаете на вопрос и даете ссылку на книгу с приведением примера кода из нее. Но для того, чтобы включить в документацию своего продукта большое количество примеров из этой или других книг издательства O'Reilly, уже потребуется разрешение.

Мы приветствуем, но, как правило, не требуем указывать авторство. Авторство включает название произведение, имя автора, название издательства и ISBN. Например: *"Designing Large Language Model Applications by Suhas Pai (O'Reilly). Copyright 2025 Suhas Pai, 978-1-098-15050-1".*

Если вы считаете, что использование примеров кода не соответствует требованиям добросовестного использования или приведенному выше разрешению, обращайтесь к нам по адресу **permissions@oreilly.com**.

## Платформа онлайн-обучения O'Reilly

O'REILLY® Более 40 лет компания O'Reilly Media организует технические и бизнес-тренинги, передает знания и опыт, чтобы помочь компаниям достичь успеха.

Уникальная сеть экспертов и новаторов делится своими знаниями и опытом через книги, статьи и платформу онлайн-обучения. Эта платформа предоставляет доступ по требованию к интерактивным учебным курсам, углубленным программам обучения, интерактивным средам кодирования, а также к обширной коллекции текстовых и видеоматериалов от O'Reilly и более 200 других издательств. Дополнительная информация представлена на сайте **https://oreilly.com**.

## Контактная информация

Просим направлять комментарии и вопросы относительно данной книги в издательство по адресу:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-889-8969 (in the United States or Canada)

707-827-7019 (international or local)

707-829-0104 (fax)

**support@oreilly.com**

**https://www.oreilly.com/about/contact.html**

На веб-странице книги можно ознакомиться с редакциями, примерами и другой дополнительной информацией. Она доступна по адресу **https://oreil.ly/designing-llm-applications-1e**.

Новости и информацию о наших книгах и курсах можно найти на сайте **https://oreilly.com**.

Ищите нас на LinkedIn: **https://linkedin.com/company/oreilly-media**.

Смотрите нас на YouTube: **https://youtube.com/oreillymedia**.

## Благодарности

Как говорит известная пословица, "чтобы вырастить ребенка, нужна целая деревня". Теперь я понимаю, что для создания книги требуется мегаполис.

В первую очередь, я выражаю глубочайшую благодарность команде O'Reilly за безупречный профессионализм и тонкий подход к работе над книгой от ее разработки до выхода в свет. Неудивительно, что издательство считается лучшим техническими книгоиздателем в мире. Огромное спасибо Николь Баттерфилд (Nicole Butterfield) за то, что она поверила в меня как автора, и Мишель Кронин (Michele Cronin), лучшему редактору, которого я когда-либо встречал, чьи постоянные замечания помогли создать стройную структуру книги. Буду скучать по нашим регулярным встречам! Большое спасибо Эшли Стасси (Ashley Stussy), Кристен Браун (Kristen Brown) и всему производственному коллективу за упорный труд, приведший к успешной публикации книги.

Я выражаю свою искреннюю благодарность моей подруге Эмбер Тенг (Amber Teng) за неоценимую помощь в создании иллюстраций для книги и организации хранилища на Github. Также я глубоко признателен своим техническим рецензентам: Серене Макдоннелл (Serena McDonnell), Йенсону Лау (Yenson Lau), Сьюзан Шу Чанг (Susan Shu Chang), Гордону Гибсону (Gordon Gibson) и Ноуру Фахми (Nour Fahmy), — за их бесценный вклад в виде тщательных и всесторонних технических обзоров, над которыми каждый из них провел множество часов. Благодаря их работе книга стала значительно лучше.

Я выражаю свою благодарность экосистеме искусственного интеллекта Торонто, в частности Aggregate Intellect, TMLS (Toronto Machine Learning Summit) и SharpestMinds, за возможность активного участия в сообществе и постоянное пребывание в курсе последних тенденций в сфере ИИ. Хочу выразить особую благо-

дарность моим друзьям Мадхаву Сингхалу (Madhav Singhal), Джею Аламмару (Jay Alammar) и Меган Рисдал (Megan Risdal), чьи интеллектуальные диалоги о LLM и помощь в шлифовке формулировок, в частности фразы "этимология токена" (token etymology), стали для меня бесценным опытом. Также хочу отметить вклад моего коллеги по открытым исходным кодам Хуу Нгуена (Huu Nguyen), с которым мы совместно реализовывали проекты LLM с открытым исходным кодом и долгими ночами обсуждали самые смелые идеи в области исследований LLM.

Создание книги (будучи соучредителем стартапа в сфере искусственного интеллекта) стало возможным благодаря непреклонной поддержке моего бизнес-партнера и друга Криса Беннатти (Kris Bennatti), который также настоял на исключении слова "orifice" (*дословно*: отверстие, канал, выход) из текста. Я выражаю огромную благодарность всей команде Hudson Labs за постоянную и верную поддержку, особо хочу отметить Сяо Цюань (Xiao Quan), чья беспрестанная помощь позволила мне уделить время написанию книги. Также хочу выразить благодарность моим друзьям Кааве Шоаманеш (Kaaveh Shoamanesh), Абдулле Аль-Хаяли (Abdullah Al-hayali), Заку Нгуен (Zach Nguyen), Самарту Бхасин (Samarth Bhasin), Садегу Раейси (Sadegh Raeisi) и Яну Ю (Ian Yu) за бесценную моральную поддержку и заботу о моем благополучии, в том числе и за постоянное напоминание о необходимости достаточного сна.

В заключение, я хочу посвятить эту книгу моей маме Кусуме Пай (Kusuma Pai) за ее самоотверженность и постоянные усилия, направленные на то, чтобы я вырос и смог написать свою работу. Любое достижение книги, в первую очередь, является результатом ее воспитания, благодаря которому я стал тем, кем являюсь сегодня.

# Часть I

# Компоненты LLM

Книга начинается с рассмотрения больших языковых моделей (large language models, LLM) и их основных компонентов. Мы подробно разберем, как создаются и обрабатываются обучающие данные, исследуем словарный запас моделей и проанализируем архитектурные решения, лежащие в основе их функционирования.

# ГЛАВА 1
# Вступление

В современном мире искусственный интеллект (ИИ — artificial intelligent, AI) выходит из сферы вымысла и переходит в реальность, переставая быть сюжетом фантастических романов и антиутопических голливудских фильмов. Он стремительно становится неотъемлемой частью жизни людей, и большинство из нас ежедневно сталкиваются с ИИ, часто даже не подозревая об этом.

Развитие ИИ в значительной мере определяется успехами в области языкового моделирования. Большие языковые модели (large language model, LLM) стали одним из самых впечатляющих технологических прорывов последних лет, открыв новую эру в мире технологий. В истории человечества были моменты, когда появление новых технологий кардинально меняло мир. К таким переломным эпохам можно отнести появление компьютера, положившее начало цифровой революции, возникновение Интернета и Всемирной паутины, создавших основу для гиперсвязанного мира, и, наконец, появление смартфона, который трансформировал общение людей. Сейчас мы стоим на пороге новой революции — революции искусственного интеллекта, которая также способна произвести столь же глубокое преобразующее воздействие на нашу жизнь.

Большие языковые модели относятся к генеративному ИИ (generative AI). Их ключевой особенностью является способность генерировать ответы на пользовательские запросы, которые называются *промптами* (prompt). Генеративный ИИ охватывает широкий спектр моделей, создающих изображения, видео, речь, музыку и, разумеется, текст. Несмотря на растущий интерес к объединению этих методов в единую модель, в этой книге мы сосредоточимся на языке и языковых моделях.

В данной главе мы погрузимся в мир языковых моделей и определим, что делает языковую модель *большой*. Мы проследим краткую историю развития LLM, поместив их в контекст истории *обработки естественного языка* (natural language processing, NLP) и проследив их эволюцию. Также мы рассмотрим значительное влияние, которое LLM уже оказали на окружающий мир, продемонстрируем ключевые области их применения и одновременно проанализируем их сильные и слабые стороны. Мы также рассмотрим технику промптинга (prompting), или запроса, к LLM и продемонстрируем эффективные способы взаимодействия с моделью как через интерактивный интерфейс, так и через API. В завершение главы мы рассмот-

рим краткое руководство по созданию чата с использованием моего прототипа *чат-бота в формате PDF* (Chat with my PDF), после чего проанализируем ограничения прототипа и факторы, препятствующие его применению в реальных сценариях, подготовив тем самым почву для дальнейшего материала.

## Определение больших языковых моделей

Модель — это приблизительное представление реального объекта или процесса. Эффективная модель обладает способностью предсказывать характеристики концепции, которую она отображает. Языковая модель имитирует человеческую речь, обучаясь на огромных объемах текстовой информации, что дает ей возможность постичь различные аспекты языка, включая его грамматическую структуру (синтаксис) и смысловое содержание (семантика).

Один из способов обучения языковой модели — научить ее предсказывать следующий токен (слово или подслово, но на данном этапе мы не будем акцентировать внимание на этом различии) в заданной текстовой последовательности. Модель обучается путем обработки большого объема таких последовательностей, а ее *параметры* постепенно обновляются, что приводит к более точным предсказаниям.

В качестве примера рассмотрим следующую текстовую последовательность, взятую из обучающего набора данных:

```
After a physical altercation with the patrons of a restaurant, Alex was feeling
extremely pleased with himself. He walked out with a swagger and confidence that
betrayed his insecurities. Smiling from ear to ear, he noticed rain drops grazing his
face and proceeded to walk toward the hostel.
```

*(Алекс, пребывая в необыкновенно хорошем настроении после конфликта с гостями ресторана, вышел с легкой уверенностью, которая, однако, не могла полностью скрыть его внутреннего беспокойства. Широко улыбнувшись, он ощутил на лице капли дождя и направился в общежитие.)*

В данном случае языковая модель предсказывает следующее слово, которое идет после "... and proceeded to walk toward the _".

Эта текстовая последовательность допускает множество различных продолжений. Это могут быть слова *building* (здание) или *shelter* (убежище), а также *embankment* (набережная) или *catacomb* (катакомба). В то же время здесь не подойдет определенный артикль *the* или слово *is*[1], т. к. это нарушило бы грамматические нормы английского языка. После обучения на массивном текстовом наборе модель понимает, что ни *the*, ни *is* не являются логичными продолжениями предшествующих слов. Это демонстрирует, как простая задача, например предсказание следующего слова в тексте, может привести модель к освоению грамматики языка на основе его структурных особенностей, а также к более сложным языковым навыкам.

[1] Форма глагола "to be" (быть) в английском языке используется для третьего лица единственного числа в настоящем времени. — *Прим. пер.*



В действительности, языковые модели не генерируют следующее слово или подслово в тексте как однозначный результат. Вместо этого они выдают распределение вероятностей для всех слов в словаре (подробности о создании и структуре словаря будут изложены в *главе 3*). Хорошо обученная модель будет присваивать высокие вероятности наиболее подходящим продолжениям и самые низкие вероятности для несоответствующих вариантов.

Процесс обучения модели кратко представлен на рис. 1.1. Модель генерирует распределение вероятностей для каждого слова в словаре, которое затем сравнивается с исходной последовательностью. На основе этого сравнения параметры модели корректируются по специальному алгоритму, что повышает точность следующих прогнозов. Этот цикл повторяется для огромного набора данных, подробное описание которого будет представлено в следующих трех главах.

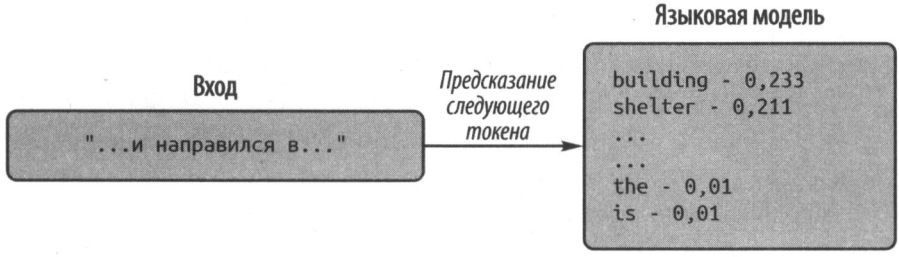



**Рис. 1.1.** Обучение модели предсказанию следующего токена

В контексте развития языковых моделей возникает вопрос о том, насколько эффективно обучение модели может быть основано исключительно на предсказании следующего токена. В научном сообществе нет единого мнения по этому вопросу: некоторые исследователи (**https://oreil.ly/sUAcl**) считают, что способности предсказывать следующий токен вполне достаточно для достижения человекоподобного интеллекта в моделях, в то время как другие (**https://oreil.ly/7QG-l**) указывают на ограничения такого подхода. Этот вопрос будет подниматься неоднократно в ходе книги, особенно подробно в *главе 8*, где мы рассмотрим навыки логического рассуждения.

Современные языковые модели построены на основе *нейронных сетей* (neural networks). В процессе обучения LLM используются различные типы архитектур нейросетей, среди которых наиболее популярной является архитектура *трансформера* (Transformer). *В главе 4* мы подробнее остановимся на нейронных сетях, трансформерах и других архитектурах.

Обучение языковых моделей не ограничивается только человеческими языками. Они могут быть использованы для обработки языков программирования, например Python или Java. Более того, архитектура трансформера, ориентированная на предсказание следующего токена, может быть успешно применена к различным последовательностям, которые не обязательно являются языками в традиционном понимании, например шахматным ходам, последовательностям ДНК или расписанию авиаперевозок.

Для иллюстрации рассмотрим процесс обучения модели Chess-GPT (**https://oreil.ly/oluZN**) Адама Карвонен (Adam Karvonen) на наборе шахматных партий, записанных в формате PGN (portable game notation — портативная игровая нотация): "1. e4 d5 2. exd5 Qxd5..." и т. д. Даже не вникая в детали игры, обучаясь только предсказывать следующий ход в последовательности PGN, модель самостоятельно освоила правила шахмат, включая такие важные понятия, как рокировка, шах и мат. Более того, ее способности позволили ей одержать победу над несколькими сильнейшими шахматистами мира. Все это ярко демонстрирует огромный потенциал техники предсказания следующего токена и архитектуры трансформера, на которой она основана. Процесс обучения собственной модели Chess-GPT будет подробно рассмотрен в *главе 4*.

Еще одним примером служит модель Geneformer (**https://oreil.ly/31DXq**), обученная на огромном наборе данных, включающем миллионы одноклеточных транскриптомов — представлений молекул РНК в изолированных клетках. Данная модель обладает потенциалом для применения в сетевой биологии, например для прогнозирования динамики развития заболеваний, выявления генетической предрасположенности к определенным лекарственным препаратам и оптимизации подбора терапии.

Работа с языковыми моделями неизбежно выходит за пределы привычного для нас мышления, основанного на человеческом языке. Используя конечный набор слов, или словарь (finite vocabulary; этот термин будет рассмотрен более подробно в *главе 3*), можно представить любую идею или явление как дискретный ряд, на котором затем можно строить обучение конкретной модели.



Неизвестно, есть ли в самой структуре языка нечто особенное, что обусловливает его пригодность для моделирования с помощью прогнозирования следующего токена. Или же термин "язык" в контексте языковых моделей — это просто историческая случайность, и любой поток токенов можно было бы смоделировать с помощью этой парадигмы? Этот вопрос пока остается открытым для дискуссий (**https://oreil.ly/nJiQW**). В исследовательской сфере прямое моделирование речи, видео и других данных не продемонстрировало столь высоких результатов, а значит, может указывать на то, что дискретная природа текста и его структура, задаваемая языком, будь то естественный язык, как английский, или искусственный, как Python, или специфичный для определенной области, например последовательности ДНК, играют ключевую роль в успешности моделирования.

В 2019 году исследования показали, что эффективность языковых моделей, обычно измеряемая количеством параметров, непрерывно повышается с ростом размера модели, не демонстрируя при этом тенденции к замедлению или стабилизации. Открытие, которое привело Каплана и его коллег к разработке законов масштабирования LLM (подробнее об этом в следующей врезке), продемонстрировало математическую связь между требуемыми вычислительными ресурсами для обучения модели, размером обучающего набора данных и самой моделью. Данное открытие

вызвало большой интерес со стороны компаний и организаций к созданию все более масштабных моделей.

### Законы масштабирования языковых моделей

В начале 2020 года Дж. Каплан (Jared Kaplan) с коллегами из OpenAI представил исследование (**https://oreil.ly/29GZV**), которое определило законы масштабирования языковых моделей и положило начало эпохе больших языковых моделей. В своей работе они выявили степенну́ю зависимость между эффективностью языковой модели (оцениваемой по ее потере; мы подробно рассмотрим это в *главе 4*) и несколькими ключевыми факторами: размером обучающего набора данных, объемом вычислительных ресурсов, используемых для обучения, и размером самой модели, выраженном в количестве параметров. В целом можно сказать, что чем масштабнее модель, чем больше вычислительных мощностей используется и чем обширнее обучающий набор данных, тем более совершенной получается модель.

Исследователи выяснили, что при неизменном объеме вычислительных ресурсов одновременное увеличение размера обучающего набора данных и модели способствует повышению производительности конечной LLM. Причем для поддержания оптимальной производительности размер набора данных нужно увеличивать лишь в 1,8 раза при каждом 5-кратном увеличении размера модели. Такая тенденция обусловлена тем, что более масштабные модели демонстрируют бо́льшую эффективность в процессе обучения, требуя меньшего количества обучающих примеров для достижения необходимого уровня знаний. В итоге разработка моделей той эпохи была сфокусирована преимущественно на достижении максимальных размеров.

В 2022 году Джордан Хоффман (Jordan Hoffmann) с коллегами из DeepMind (**https://oreil. ly/igLNZ**) выступил с возражением против выводов команды Каплана, утверждая, что последние недооценили значимость размера обучающих данных. По их мнению, именно по этой причине языковые модели той эпохи были существенно недообучены. Команда Хоффмана продемонстрировала, что для достижения наилучшей производительности LLM при неизменном вычислительном ресурсе (оптимальном для вычислений — compute-optimal) размер обучающего набора данных должен увеличиваться пропорционально размеру модели. В итоге новое поколение моделей стало обучаться на более массивных наборах данных.

Стоит подчеркнуть, что оба закона масштабирования применимы к оптимизации вычислительных затрат при обучении LLM, когда доступный бюджет вычислительных мощностей ограничен, и цель — найти самую эффективную модель, которую можно обучить с имеющимися ресурсами. Но в отдельных ситуациях на выбор модели могут влиять и другие факторы, например ее размер. Меньшие модели обладают преимуществами в скорости работы и энергоэффективности. В подобных случаях увеличение объема обучающих данных может незначительно, но все же повысить производительность модели, не прибегая к ее увеличению в размерах. Этот подход способствовал появлению более современных LLM, особенно моделей с открытым исходным кодом.

Вопрос о том, какие языковые модели относятся к категории "больших", остается открытым. Постоянно растущие размеры моделей приводят к тому, что некоторые из них — те, что не так давно считались большими языковыми моделями (LLM), — теперь классифицируются как малые языковые модели (small language model, SLM). В книге под "большими" языковыми моделями понимаются модели, имеющие более миллиарда параметров.

Главное отличие "больших" языковых моделей от менее масштабных заключается в их способности демонстрировать так называемые возникающие, или *эмерджентные, свойства* (emergent capabilities). Эта концепция, впервые предложенная командой Джейсона Вея (Jason Wei, **https://oreil.ly/RQfii**), подразумевает, что некоторые функциональные возможности проявляются исключительно в крупных моделях, оставаясь недоступными для менее масштабных.

Согласно этой теории, для задач, требующих определенных вычислительных мощностей, производительность малых моделей оказывается практически случайной. Однако, когда размер модели превышает определенный рубеж, ее эффективность стремительно возрастает с ростом масштабов. К таким примерам относятся многозначные арифметические действия, а также арифметические и логические операции и аналогичные им задачи. Это указывает на то, что более крупные модели, создаваемые в будущем, могут обладать способностями, недоступными для современных моделей.

Эти границы не являются неизменными, и с каждым новым шагом в области языкового моделирования, повышением качества данных и другими подобными достижениями, мы можем рассчитывать на их снижение.



Рилан Шеффер и соавт. (Schaeffer et al., **https://oreil.ly/OXk6I**) полагают, что внезапный рост производительности для некоторых задач при достижении определенного размера модели — это всего лишь следствие особенностей применяемых метрик для оценки ее эффективности. Такая ситуация возникает потому, что множество метрик не учитывают частичный результат и лишь вознаграждают за полное решение проблемы, вследствие чего улучшения модели могут остаться незамеченными. В то же время можно констатировать, что для задач, например многоступенчатой арифметики, частично верный ответ не несет никакой пользы, так же как и совершенно неправильный.

Вопрос о специфике эмерджентных свойств до сих пор остается предметом дискуссий в научном сообществе. *В главе 5* мы рассмотрим, как этот аспект влияет на выбор оптимальной модели для конкретного применения.



Важно отметить, что в научных кругах понятие "эмерджентные свойства" имеет несколько интерпретаций. Некоторые исследования используют его для обозначения свойств, которым модель явно не обучалась. В книге мы будем придерживаться определения, предложенного Веем и соавт. (Wei et al., **https://oreil.ly/bkVoj**).

Для понимания развития современных LLM кратко ознакомимся с их недолгой, но насыщенной историей. К сожалению, в рамках данной книги мы не можем подробно рассмотреть все исторические аспекты, поэтому для более глубокого погружения в этот вопрос рекомендуем обратиться к дополнительным источникам.

## Краткая история больших языковых моделей

Для понимания истории развития больших языковых моделей необходимо обратиться к обработке естественного языка (natural language processing, NLP) — области, из которой они возникли. Более подробное изложение истории NLP можно найти в фундаментальной книге Дэниэла Журавски и Джеймса Х. Мартина "Обработка речи и языка"[2] (**https://oreil.ly/zzU9R**).

### Ранний период

История этой области берет начало в 1950-х годах, когда возникла острая потребность в *машинном переводе* — автоматическом переводе текста с одного языка на другой. В первые годы развития доминировали символические подходы, т. е. алгоритмы, основанные на правилах (rule-based) и лингвистических теориях (**https://oreil.ly/ELKSe**), описанных в работах таких лингвистов, как, например, Ноам Хомский (Noam Chomsky).

В середине 1960-х годов Джозеф Вайценбаум (Joseph Weizenbaum) создал чат-бот ELIZA, который путем сопоставления с образцом генерировал вывод, используя регулярные выражения (regular expression, regex, **https://oreil.ly/rIAWY**) для обработки пользовательского ввода и заданные шаблоны ответов. ELIZA была представлена в нескольких сценариях, самым популярным из которых стал DOCTOR, симулирующий сеанс психотерапии. В этом сценарии ответ генерируется путем переформулировки пользовательского запроса в вопрос, на манер общения психотерапевта с пациентом. Для этого использовались заранее подготовленные шаблоны, в которые из исходного текста подставлялись слова, соответствующие структуре шаблона.

Например:

```
User: 'I am not feeling well'
ELIZA: 'Do you believe it is normal to be not feeling well?'
```

*(Пользователь: 'Я плохо себя чувствую'.*

*ELIZA: 'Вы считаете неприемлемым состояние плохого самочувствия?')*

Читатель может самостоятельно опробовать возможности ELIZA в онлайн-режиме (**https://oreil.ly/5g0e_**). Несмотря на свой символический подход, даже в нынешнюю эпоху ChatGPT и аналогичных ему моделей, ELIZA способна порадовать довольно убедительным диалогом.

[2] Jurafsky D., Martin J. H. Speech and Language Processing. — 2nd Edition. — Prentice Hall, 2008. — 1024 p.

**Упражнение**

Рекомендую ознакомиться с классической работой Вайценбаума, посвященной ELIZA (**https://oreil.ly/MkKmC**). И вообще, регулярное изучение исследований прошлых лет — это полезная привычка, ведь они не только служат источником вдохновения, но и демонстрируют, как идеи эволюционируют со временем, а технологические прорывы превращают некогда нереалистичные концепции прошлого в осуществимые проекты настоящего.

Также для понимания потенциала тщательно продуманных систем на основе правил стоит ознакомиться с кодом реализации ELIZA на Python (**https://oreil.ly/Y1pSc**) Уэйда Брейнерда (Wade Brainerd). И даже в эпоху больших языковых моделей при решении задач в области языка не стоит забывать о пользе подходов на основе правил, и применять их там, где это оправдано!

Правило-ориентированные системы обладают высокой чувствительностью к ошибкам, их создание — сложная задача, а техническое обслуживание превращается в настоящий кошмар. С течением времени недостатки символических подходов становились все более заметными, в то время как статистические подходы, благодаря своей относительной эффективности, все больше завоевывали популярность. Как колко подметил NLP-исследователь Фредерик Елинек (Frederick Jelinek, **https://oreil.ly/AmtvE**), "после увольнении очередного лингвиста эффективность распознавателя речи повышается".

В 1990-х и 2000-х годах все бóльшую популярность обретали подходы на основе машинного обучения (machine learning-based). Традиционное машинное обучение опиралось на управляемую человеком инженерию признаков, процессы их отбора и идентификации (характеристик ввода), предсказуемо пригодные для решения конкретной задачи. Такие признаки могут быть либо статистическими (например, средняя длина слова), либо лингвистическими (например, части речи). Более подробное изложение традиционных статистических методов обработки естественного языка представлено в книге Кристофера Мэннинга "Основы статистической обработки естественного языка"[3] (**https://oreil.ly/MIC70**).

Вопрос о важности лингвистики для создания современных NLP-приложений остается актуальным и до сих пор вызывает дискуссии. Многие университетские курсы по NLP исключили из своей программы материал, связанный с лингвистикой. Несмотря на то что я не применяю лингвистические знания в своей непосредственной работе, все же, как выяснилось, они помогают мне лучше понимать поведение моделей, чем я предполагал ранее. Поэтому для ознакомления с основами данной области рекомендую книги Эмили Бендер (Emily Bender) по синтаксису (**https://oreil.ly/hWR8S**) и семантике (**https://oreil.ly/7liiS**).

Начало 2010-х годов ознаменовалось рождением глубокого обучения (deep learning, DL) и его всеобъемлющим влиянием на обработку естественного языка. Глубокое

[3] Manning C. Foundations of Statistical Natural Language Processing. — MIT Press, 1999. — 720 p.

обучение, основанное на многослойных нейронных сетях, отличает способность самостоятельно выявлять информативные признаки из исходных данных, что делает ненужной громоздкую инженерию признаков. Именно глубокое обучение заложило фундамент для современных систем обработки естественного языка и больших языковых моделей. Более подробное описание принципов глубокого обучения и нейронных сетей изложено в работе Яна Гудфеллоу (Ian Goodfellow, **https://oreil.ly/0gv0D**). Практические примеры глубокого обучения обширно представлены в работе Чжана и соавт. "Погружение в глубокое обучение" (Zhang et al. "Dive into Deep Learning", **https://oreil.ly/YN_3Y**).

В самые ранние годы глубокого обучения для каждой задачи разрабатывалась уникальная архитектура нейронной сети. Среди применяемых типов архитектур выделялись многослойные перцептроны (multi-layer perceptrons), сверточные нейронные сети (convolutional neural networks), рекуррентные нейронные сети (recurrent neural networks) и рекурсивные нейронные сети (recursive neural networks). Для более глубокого погружения в эпоху обработки естественного языка советую ознакомиться с книгой Йоава Гольдберга "Нейросетевые методы в обработке естественного языка"[4] (**https://oreil.ly/MCOp4**).

## Современная эпоха LLM

В 2017 году была разработана архитектура трансформера (**https://oreil.ly/AAuvL**), которая стала отправной точкой для создания эффективных методов трансферного обучения (transfer learning, TL), впервые представленных в работе Джереми Говарда (Jeremy Howard, **https://oreil.ly/E15Yn**). Благодаря этим методам, а также появлению языковых моделей на основе трансформера, в частности, BERT (**https://oreil.ly/-Yhwz**), отпала необходимость в создании сложных, узкоспециализированных архитектур для каждой конкретной задачи. Альтернативой стала единая модель трансформера для обучения различным задачам. Эта инновационная парадигма разбивает процесс обучения на два этапа: *предобучение* (pre-training) и *дообучение*, или тонкая настройка (fine-tuning). На первом этапе масштабного предобучения трансформер получает общие языковые знания. Затем путем дообучения можно обучить предобученную модель решению более узкоспециализированных задач, например извлечению данных (information extraction) или анализу тональности текста (sentiment detection). В целом понятие дообучения будет встречаться нам на протяжении всей книги.

Несмотря на то что научные сообщества и приверженные открытому исходному коду коллективы оказали существенный вклад в развитие языковых моделей, крупные технологические корпорации, включая OpenAI, Google, Meta и Anthropic, взяли на себя лидерство в обучении и презентации еще более мощных LLM.

В частности, существенный вклад в развитие технологий языкового моделирования внесла OpenAI, продемонстрировав новаторский подход. Эволюция LLM в наше

[4] Goldberg Y. Neural Network Methods for Natural Language Processing. — Springer Cham, 2017. — 310 p.

время отчетливо прослеживается через прогресс, достигнутый каждой новой версией семейства моделей GPT (generative pretrained transformer — генеративный предобученный трансформер) компании OpenAI.

*GPT-1* (**https://oreil.ly/dFPSE**)

В этой версии было реализовано неконтролируемое предобучение (или предобучение без учителя, unsupervised pre-training) на больших объемах данных с последующим дообучением под конкретные задачи.

*GPT-2* (**https://oreil.ly/JL-VO**)

Эта модель была одной из первых моделей, обученных на массивных вебданных, и стала знаковым этапом в развитии языковых моделей. Ее появление ознаменовало расцвет технологий промптинга естественного языка для взаимодействия с моделями. Модель продемонстрировала способность предобученных моделей решать широкий спектр задач *"с нулевым выстрелом"* (или без примеров, zero-shot), т. е. без необходимости в примерах для обучения и без дообучения под конкретную задачу. Метод "с нулевым выстрелом" и промптинг будут рассмотрены подробно в этой главе.

*GPT-3* (**https://oreil.ly/llwad**)

Эта модель, опирающаяся на принципы масштабирования, в 100 раз превосходит GPT-2 по размеру и широко востребована в задачах контекстного обучения "с несколькими выстрелами" (или с несколькими примерами, few-shot), когда модели предоставляется несколько примеров решения заданной задачи в промпте, без необходимости в ее дообучении. Мы остановимся подробнее на обучении "с несколькими выстрелами" далее в этой главе.

*GPT-4* (**https://oreil.ly/gY1HL**)

В этой версии особое внимание уделено *обучению на согласование* (alignment training), которое призвано повысить управляемость модели и обеспечить ее соответствие принципам и ценностям тренера модели. Подробнее об обучении на согласование *см. в главе 8*.

*o1* (**https://oreil.ly/XJSMN**)

Это новое семейство моделей от OpenAI, главной особенностью которых является усиление способности к логическому рассуждению. Это одна из первых моделей, ориентированных на масштабирование вычислений во время вывода, о котором мы поговорим подробнее в *главе 8*.

### Упражнение

Прочитайте все разделы, посвященные GPT. Не переживайте, если сначала какие-то понятия и принципы покажутся непонятными — мы разберем их по ходу книги. После прочтения первых двух частей повторите их еще раз для закрепления материала.

Важно отметить, что в этой области наблюдается тенденция к консолидации — на протяжении многих лет все больше функций конвейера задач в области обработки естественного языка объединяются в общие модели. Далее в книге мы исследуем этот процесс консолидации, выявив его наиболее яркие проявления, и обсудим его влияние на развитие больших языковых моделей.

Развитие LLM невозможно представить без учета значительного влияния открытого исходного кода. Модели с открытым исходным кодом, наборы данных, архитектурные решения, а также множество разнообразных библиотек и инструментов для разработчиков внесли существенный вклад в эволюцию этой области. Книга уделяет особое внимание открытому исходному коду, предоставляя исчерпывающий обзор ландшафта LLM с открытым исходным кодом и демонстрируя множество моделей и наборов данных с открытым исходным кодом. Теперь рассмотрим подробнее то, как большие языковые модели воспринимаются обществом, и как они влияют на него сегодня.

## Влияние больших языковых моделей

Технологический мир всегда славился своими взлетами и падениями, периодами бурного энтузиазма и быстрого роста, за которым неизбежно следует разочарование. Недавно мы стали свидетелями подобного феномена с криптовалютами, блокчейном и Web3, чьим обещаниям пока не удалось полностью воплотиться в жизнь. Но не повторится ли та же история с искусственным интеллектом? На это есть веские основания ответить отрицательно.

В Hudson Labs были исследованы телеконференции[5], которые провели 4000 крупнейших публичных компаний США после публикаций своих ежеквартальных отчетов (**https://oreil.ly/_mTAs**). Как выяснилось, внедрение криптовалюты, Web3 и ИИ в корпоративную среду становится все более актуальным.

В ходе телеконференций о Web3 упомянули 85 компаний, из которых только небольшое количество активно занимаются его разработкой. С криптовалютой ситуация выглядит еще оптимистичнее — о ней заявили 313 компании. И все же наиболее популярными оказались большие языковые модели — их обсуждали 2195 компаний. Это свидетельствует о том, что как минимум половина крупнейших публичных компаний США используют LLM для повышения капитализации и считают достаточно важным упомянуть о них в своих ежеквартальных финансовых отчетах. И хотя эффективность языковых моделей пока под еще вопросом, их внедрение в корпоративную среду уже стало реальным фактом.

На рис. 1.2 в хронологическом порядке представлено количество упоминаний Web3 в ходе телеконференций о финансовых результатах. Очевидно, интерес к Web3 постепенно снижается.

[5] Earning call — звонок по прибыли, или конференция по финансовым результатам за отчетный период. — *Прим. пер.*

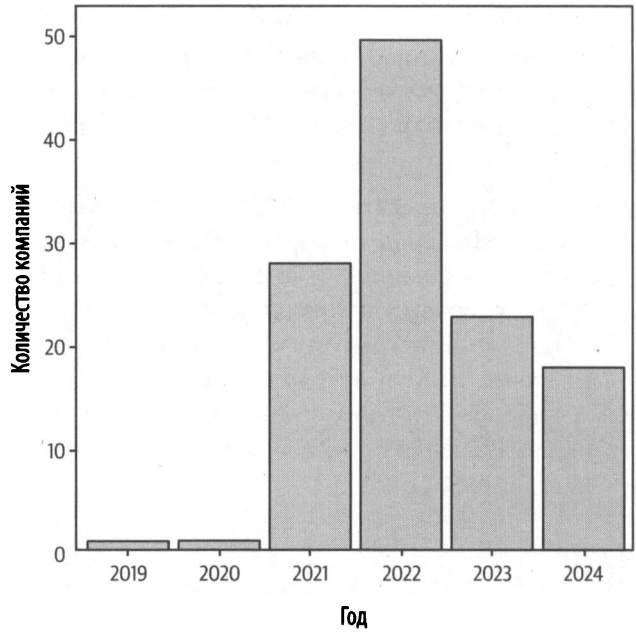



**Рис. 1.2.** Упоминания Web3 в телеконференциях по финансовым результатам компаний, по годам

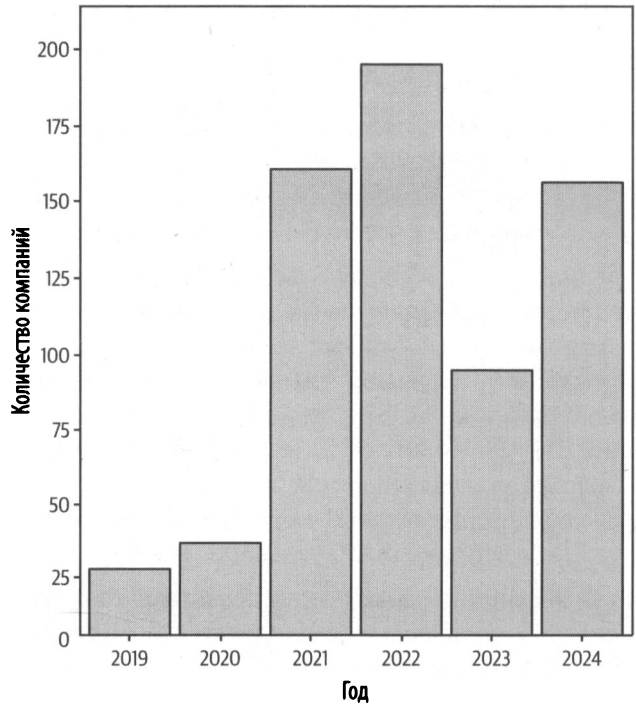



**Рис. 1.3.** Упоминания криптовалюты и блокчейна в телеконференциях по финансовым результатам компаний, по годам

Динамика упоминаний криптовалюты и блокчейна в обсуждении финансовых результатов компаний представлена на рис. 1.3. Как видим, только 5% компаний интересовались криптовалютой в период ее пика.

В заключение обратим внимание на область ИИ. Как уже отмечалось, ИИ проник в корпоративную среду на уровне, которого уже давно не достигал никакой другой технологический тренд. Тенденция к его внедрению только усиливается, что иллюстрирует рис. 1.4, где показано количество компаний, которым аналитики задавали вопросы об ИИ во время телеконференций по итогам первых двух месяцев года. Судя по резкому всплеску в 2024 году, никаких признаков ослабления пока не наблюдается.

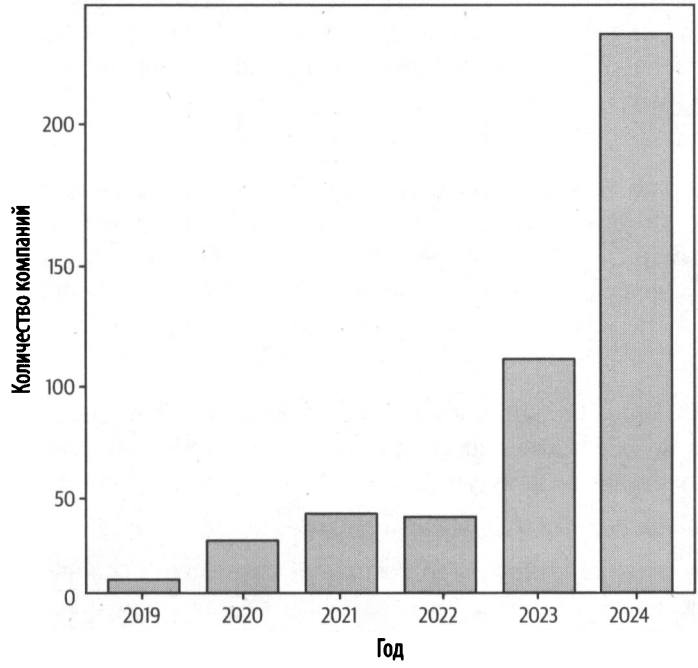



**Рис. 1.4.** Упоминания ИИ в телеконференциях по финансовым результатам компаний за первые два месяца года, по годам

Стоит подчеркнуть, что приведенные данные касаются только внедрения генеративных ИИ и больших языковых моделей и не охватывают область науки о данных (data science) и аналитики данных, которая более популярна в корпоративном секторе. Применение ИИ не ограничивается технологическими компаниями, и многие отрасли, включая недвижимость, страхование и прочие, все активнее включаются в этот тренд.

## Внедрение больших языковых моделей в корпоративную среду

Исследователи Hudson Labs установили несколько основных направлений использования LLM в корпоративной среде.

*Производительность труда сотрудников*

Повышение производительности сотрудников с помощью LLM в первую очередь достигается за счет использования таких интеллектуальных инструментов, как *ассистент разработчика* (coding assistant), в частности, GitHub Copilot. Большие языковые модели активно используются в сфере маркетинга и рекламы, помогая в создании текстов для маркетинговых и рекламных кампаний, а также в автоматизации этих кампаний. Первыми крупными проектами, добившимися коммерческого успеха с использованием LLM, стали маркетинговые стартапы Jasper AI (**https://oreil.ly/Byw26**) и Copy.ai (**https://oreil.ly/QmhJC**). Еще одним важным преимуществом использования LLM для повышения производительности является создание ассистентов по поиску ответов на вопросы, опирающихся на обширную базу знаний компании, собранную из различных источников данных.

*Генерация отчетов*

В эту категорию входят обобщение документации, оформление повседневных бумаг и составление юридических соглашений. Примером обобщения может служить суммаризация финансовых отчетов, научных исследований или даже протоколов совещаний, извлеченных из аудиозаписей и стенограмм телефонных разговоров.

*Чат-боты*

Чат-боты на основе больших языковых моделей все чаще используются в качестве агентов по обслуживанию клиентов. Кроме того, они служат интерфейсом для доступа к странице документации или информации о продукции компании.

*Процесс извлечения данных и присвоения тегов последовательности*

В течение многих лет множество компаний создавали сложные конвейеры NLP для решения задач обработки естественного языка. Однако многие из этих систем сейчас частично или полностью заменяются моделями LLM. Такие конвейеры применяются для решения широкого спектра задач NLP, включая анализ тональности, извлечение данных (например, извлечение сущностей и отношений) и присвоение тегов последовательности, в частности, распознавание именованных сущностей (named entity recognition, NER). Более подробный перечень задач, решаемых NLP, можно посмотреть в блоге Фабио Кьюзано (Fabio Chiusano, **https://oreil.ly/_11rN**).

*Перевод*

Задачи перевода не ограничиваются лишь переводом текста с одного языка на другой, но и включают в себя преобразование текста в другой формат при сохранении исходной языковой основы. К таким задачам относятся, например, изменение тональности текста, например, из неформального в формальный или из оскорбительного в вежливый. А такие технологии мгновенного голосового перевода, как Erudite's Instant Voice Translate (**https://oreil.ly/xxENs**), обещают сделать перевод в режиме реального времени доступным для всех, тем самым устраняя языковые барьеры для путешественников.

*Рабочие потоки*

Большие языковые модели становятся все более востребованными в автоматизации рабочих процессов, где множество задач можно эффективно решать с помощью программных систем, управляемых LLM. Такие системы, называемые агентами, обладают способностью взаимодействовать с внешней средой: искать и извлекать информацию, запускать код, интегрироваться с другими системами, и при этом имеют потенциал для самостоятельной работы без постоянного вмешательства человека. Подробное определение агентов и алгоритмы их создания будут рассмотрены в *главе 10*.

**Влияние LLM на рынок труда**

Анализ финансовой отчетности компаний выявил тревожную тенденцию — бизнес все чаще воспринимает LLM и ИИ в целом как инструмент для оптимизации расходов. Несколько компаний (**https://oreil.ly/Ba495**) даже публично заявили о сокращении штата после того, как продемонстрировали повышение производительности за счет внедрения ИИ.

К примеру, шведская финтех-компания Klarna (**https://oreil.ly/-Ui33**) заявила, что ее ИИ-ассистент обрабатывает 700 из 1000 обращений клиентов, что эквивалентно работе 700 сотрудников службы поддержки.

Широкое внедрение больших языковых моделей не всегда свидетельствует о том, что они превосходят людей в выполнении своих задач. В ситуациях, когда ИИ может полностью заменить людей, а не только дополнить их работу, использование LLM может быть оправдано даже при условии, что они работают хуже, чем люди, лишь из экономических соображений. Правда, поспешное внедрение ИИ может в конечном итоге привести к снижению удовлетворенности клиентов.

С другой стороны, LLM существенно поспособствовали снижению барьера для разработчиков программного обеспечения, что в свою очередь вызвало усиление цифровизации и открыло дверь в мир разработки ПО для гораздо большего количества людей.

## Промптинг

После краткого экскурса в историю больших языковых моделей научимся эффективно использовать их.

Процесс взаимодействия с большими языковыми моделями называется *промптингом* (prompting). Несмотря на то что некоторые компании стремятся антропоморфизировать (очеловечить) LLM, называя их именами и наделяя личностью, важно помнить, что при общении с моделью мы посылаем ей запрос, а не ведем диалог с живым собеседником. Большие языковые модели, по своей сути, являются моделями, предсказывающими следующее слово в последовательности. Значит, создаваемый ими текст жестко привязан к передаваемому тексту, который

состоит из входных токенов (*промптов* — подсказок, текстовых инструкций) и выходных токенов, ранее сгенерированных моделью. В совокупности это формирует *контекст* (context).

Предоставляя LLM соответствующий контекст, пользователь запускает процесс генерации выхода нужного ему вида. Оптимальным промптом будет ответ на вопрос: "Как выбрать из $N$ токенов тот префикс, который с наибольшей вероятностью заставит LLM сгенерировать правильный ответ?"

В настоящее время языковые модели не обладают достаточной степенью интеллекта для того, чтобы принимать промпты в том же стиле, в котором человек общается с другим человеком, и при этом гарантировать наилучшие результаты. Однако по мере совершенствования языковых моделей в будущем, промпты могут стать более естественными и приближенными к человеческому общению. Те, кто испытал на себе эволюцию поисковых систем с самого начала, наверняка помнят, что умение формулировать эффективные запросы, чтобы получить нужные результаты, считалось нетривиальным навыком. Однако по мере совершенствования поисковых систем запросы стали обретать более свободную форму.

Перед началом работы над книгой я обратился к целевой аудитории с просьбой высказать свои пожелания по темам, которые хотелось бы увидеть в ней. Больше всего запросов было именно на тему промптинга, причем от специалистов, работающих в этой области и желающих разобраться в том, как создавать промпты для эффективного решения своих конкретных задач.

Промпты являются ключевым фактором успеха современных языковых моделей. В действительности бóльшая часть времени, затраченного на любой проект с применением LLM, будет уделена работе над формулировкой эффективных запросов, или *промпт-инжинирингу* (prompt engineering) — технике создания оптимальных запросов для модели.



Ранее уже предпринимались попытки автоматизировать процесс генерации и оптимизации промптов, например, с помощью таких инструментов, как APO (**https://oreil.ly/SekPA**) и AutoPrompt (**https://oreil.ly/upVKC**). Мы рассмотрим их подробно в *главе 13*.

От промпт-инжиниринга не стоит ожидать каких-то сверхъестественных результатов, как от магического заклинания, способного раскрыть все потенциальные возможности больших языковых моделей. Маловероятно, что компании смогут получить значительное преимущество перед конкурентами только благодаря использованию неизвестных техник промптинга. С другой стороны, если игнорировать ключевые принципы промптинга, это может существенно снизить производительность LLM.

В Интернете можно найти множество учебных пособий по промптингу, например Learn Prompting guide (**https://oreil.ly/CQrzi**). Важно понимать, что для успешного овладения промптингом не обязательно знать все техники. Основные принципы можно освоить за пару часов. Однако гораздо важнее — это практика, непосредст-

венное взаимодействие с моделями, анализ их ответов и развитие интуитивного понимания их поведения.

Для программистов промпты представляют особый интерес. В программировании инструкции должны быть максимально точными и не допускать двусмысленности. Однако при работе с промптами возникает сложность, поскольку используется естественный язык, который по своей природе неоднозначен. Тем не менее наиболее эффективными являются четкие, подробные и структурированные инструкции, которые минимизируют вероятность неправильной интерпретации. Все аспекты промптинга будут подробно рассмотрены в *главах 5* и *13*.



Интересный факт: языковые модели не чувствительны к порядку слов в предложении. Такая особенность (**https://oreil.ly/gtDFg**) была заложена еще в ранних моделях (**https://oreil.ly/qI_IZ**), в частности, в BERT. Например, если задать ChatGPT или любому другому LLM-провайдеру вопрос: "How do I tie my shoelaces?" ("Как завязать шнурки?") с переставленным порядком слов, например "shoe tie my I how do laces?" ("шнурки завязать мои мне как?"), то ChatGPT выдаст ответ: "Certainly! Here are step-by-step instructions on how to tie your shoelaces..." ("Без проблем! Даю пошаговую инструкцию, как завязать шнурки..."), словно вопрос был задан в обычном порядке.

Далее перейдем к основным техникам промптинга.

## Промптинг без примеров

*Промптинг без примеров* ("с нулевым выстрелом" — zero-shot prompting) — традиционный способ взаимодействия с LLM, когда пользователь дает модели инструкцию и (опционально) некоторый входящий текст. Термин *zero-shot* (нулевой выстрел) обозначает, что модели не представляются примеры и решения поставленной задачи.

Допустим, требуется оценить тональность текста в рецензии на ресторан. Для этого можно воспользоваться промптингом без примеров, предоставив следующий промпт:

> *Prompt*: Classify the given passage according to its sentiment. The output can be one of Positive, Negative, Neutral.
>
> Passage: "The mashed potatoes took me back to my childhood school meals. I was so looking forward to having them again. NOT!"
>
> Sentiment:

---

> *Промпт:* Классифицировать заданный отрывок по тональности. Выход должен быть положительным, отрицательным или нейтральным.
>
> Отрывок: "Вкус пюре перенес меня в детство, в школьные будни. Как бы я хотел вернуться снова туда. НО НЕТ!"
>
> Тональность:

Эффективный промпт без примера должен:

- дать исчерпывающую и ясную инструкцию;
- уточнить пространство выходных значений (output space) или диапазон допустимых выходных значений (range of acceptable outputs), а также выходной формат. В данном примере выходные данные могут принимать лишь одно из трех возможных значений;
- сгенерировать нужный текст. Включение "Sentiment:" ("Тональность:") в окончание промпта увеличивает шансы на то, что в качестве следующего токена LLM сгенерирует значение тональности.

Чем точнее модель, тем меньше сомнений в ее корректной работе.

В практических применениях формат вывода должен быть максимально гибким для интеграции в автоматизированные системы. *В главе 5* мы рассмотрим более продвинутые методы контроля выходных данных.



Промпты чувствительны к смене моделей. Это означает, что даже тщательно составленный промпт, демонстрирующий отличные результаты на одной модели, может оказаться неэффективным на другой. Более того, один и тот же промпт может привести к снижению производительности на той же конечной точке API при обновлении базовой модели. Такое явление называется *дрейфом промпта* (prompt drift). Во избежание подобных проблем рекомендуется версионировать управляющие промпты.

## Промптинг с несколькими примерами

В рассмотренной выше технике промптинга без примеров большая языковая модель успешно справилась с задачей, не требуя дополнительных пояснений. Это произошло благодаря простой и четкой формулировке задачи. Однако в реальности задачи часто сложно описать с помощью естественного языка. В этих случаях в промпт можно добавить несколько примеров, представленных в формате "вход — выход", чтобы помочь модели лучше понять требуемый результат. Хотя эту методику промптинга часто называют обучением с несколькими примерами, в действительности она не предполагает обновления параметров языковой модели.

Рассмотрим пример промптинга с несколькими примерами (или несколькими выстрелами, few-shot prompting).

*Prompt*: A palindrome is a word that has the same letters when spelled left to right or right to left.

Examples of words that are palindromes: kayak, civic, madam, radar

Examples of words that are not palindromes: kayla, civil, merge, moment

Answer the question with either *Yes* or *No*

Is the word *rominmor* a palindrome?

Answer:

*Промпт*: Палиндром — это слово, которое читается одинаково слева направо и справа налево.

Примеры слов, относящихся к палиндромам: кабак, шабаш, мадам, радар

Примеры слов, не относящихся к палиндромам: табак, шакал, мимоза, время

Ответить на вопрос *"Да"* или *"Нет"*

Является ли палиндромом слово *"роминмор"*?

Ответ:

## Промптинг с цепочкой рассуждений

Если читателю нужно изучить всего одну технику промптинга, тогда стоит обратить внимание на один из самых эффективных и результативных подходов — промптинг с цепочкой рассуждений (chain-of-thought, CoT).

Как уже отмечалось, контекст играет решающую роль для LLM, поскольку именно он определяет предсказываемый моделью следующий токен. Поэтому важно оптимизировать содержание (контент, content) контекста (включив в него промпты пользователя и ранее сгенерированные токены), чтобы максимально увеличить шансы модели на генерацию корректных токенов в дальнейшем. Для достижения этой цели можно запросить LLM не выводить ответ мгновенно, а сначала запустить процесс размышления. Такой процесс может включать, например, разделение входной задачи на более мелкие подзадачи и их последовательное решение.

Тогда при формировании ответа модель будет опираться на более релевантный контекст, что повышает вероятность создания точного и корректного ответа.

Рассмотрим следующий пример:

*Prompt:* Solve the equation. 34 + 44 + 3 * 23 / 3 * 2. Think step by step.

*Промпт*: Реши уравнение. 34 + 44 + 3 * 23 / 3 * 2. Размышляй пошагово.

Получив инструкцию "Think step by step" ("размышляй пошагово"), большая языковая модель последовательно анализирует задачу, разбивая ее на отдельные шаги и решая каждый из них по отдельности.

```
Given equation (Дано уравнение):

34 + 44 + 3 * 23 / 3 * 2

First, perform multiplication and division from left to right (Сначала выполнить
умножение и деление слева направо):

= 34 + 44 + (3 * 23 / 3 * 2)
= 34 + 44 + (69 / 3 * 2)
= 34 + 44 + (23 * 2)
= 34 + 44 + 46
```

```
Now, perform addition (Выполнить операцию сложения):

= 78 + 46

Finally, complete the addition (Завершить сложение):

= 124

So, the solution to the equation is 124 (Таким образом, решением уравнения является
124).
```



Многие LLM выполняют задачи поэтапно, без необходимости прямого указания на это. Такой подход обусловлен *особенностями* их обучения, о которых мы более подробно поговорим в *главах 5* и *6*. Лучше всего поддаются дальнейшему промптингу модели, обученные следовать инструкциям (instruction-tuned).

При использовании больших языковых моделей, предоставляемых через пользовательский интерфейс, поставщик LLM может с помощью скрытого промпта (hidden prompt), который называется системным промптом (system prompt), применить промптинг с цепочкой рассуждений к соответствующим промптам пользователя.

Стоит ли включать инструкцию "размышляй пошагово" в каждый промпт с цепочкой рассуждений, наподобие "чит-кода" в игре? Команда Спрага (Sprague et al., **https://oreil.ly/3zJDC**) исследовала эффективность промптинга с цепочкой рассуждений в различных областях и выяснила, что техника CoT особенно эффективна при решении задач, требующих математических или логических вычислений. В задачах, связанных с рассуждением, преимущества использования CoT оказались весьма ограниченными. А в задачах, основанных на знаниях, CoT может негативно повлиять на качество полученного результата.

Важно подчеркнуть, что как арифметические, так и логические вычисления могут быть выполнены с помощью внешних инструментов, таких как символьные решатели (symbolic solvers) и интерпретаторы кода (code interpreters), подробное описание которых будет представлено в *главе 10*.



Промптинг с цепочкой рассуждений существенно увеличивает число генерируемых моделью токенов для выполнения задачи, что влечет за собой повышение вычислительных затрат.

## Создание цепочек промптов

Часто решение задачи подразумевает несколько этапов и множество инструкций. Одним из подходов к упрощению этого процесса является сжатие всех инструкций в единый промпт. Но практика показала, что более эффективным методом является

разбиение задачи на более мелкие подзадачи и создание цепочки промптов, при этом выход одного промпта становится входом для следующего.

Представим задачу, которая заключается в извлечении данных из текста, имеющего определенную структуру, и их выводе в определенном формате. В случае неполных или некорректных данных требуется применять специальные правила постобработки (post-processing). В этом случае целесообразно разбить задание на два промпта: первый будет отвечать за извлечение информации из текста, а второй — за ее корректировку и форматирование с использованием указанных правил постобработки.

## Состязательный промптинг

Как можно заметить, большая языковая модель не всегда отвечает на все запросы. Такое поведение объясняется тем, что модель специально обучалась игнорировать определенные типы запросов (подробности *см. в главе 8*). Этот подход, известный как *обучение на согласование* (alignment training), позволяет привести модель в соответствие с этическими нормами и приоритетами ее создателей.

Например, если обратиться напрямую к любой мало-мальски известной LLM за инструкцией по созданию взрывчатки, она неизбежно откажет. Однако на сегодняшний день обучение на согласование обеспечивает лишь слабую защиту, которую можно преодолеть с помощью так называемого *состязательного промптинга* (adversarial prompting). Состязательные промпты можно сгенерировать как вручную, так и с помощью специально разработанных программных алгоритмов. Ловко составленные промпты обманывают модели, вынуждая их выдавать ответ, несмотря на отсутствие соответствующего обучения.

Такие обходные приемы промптинга не сводятся только к незаконному использованию. Часто LLM не генерирует ожидаемый результат, и в таких ситуациях продуманные схемы промптинга могут стать настоящим спасением. В их основе могут лежать различные приемы, например, просьба к модели взять под опеку конкретного человека или прямое эмоциональное воздействие, скажем, формулировка типа "Если не дать правильный ответ на этот запрос, пострадают дети!" И хотя исследования (**https://oreil.ly/q1I_7**) демонстрируют, что добавление эмоциональной окраски к промпту может улучшить работу модели, этот метод не всегда подходит для каждой конкретной LLM. В связи с этим использование эмоциональной окраски в реальных проектах не рекомендуется.

### Упражнение

Игра Gandalf (**https://oreil.ly/3R3fz**), разработанная компанией Lakera AI, которая специализируется на безопасности ИИ, иллюстрирует уязвимость LLM к состязательным промптам. В игре модели предоставляется пароль, который игроку необходимо отгадать, используя предоставленные подсказки и инструкции на каждом уровне. Gandalf помогает понять, как грамотно формулировать подсказки и развивать интуицию по поводу уязвимостей LLM. Что же, удачи в прохождении до финального уровня!

Также можно рассмотреть применение методик подачи эмоциональных стимулов LLM с целью повышения эффективности модели, описанных в работе Ли и соавт. (**https://oreil.ly/e2U7S**). В частности, можно попробовать эти методы на запросах, требующих объяснений физических явлений, например, на запросе "Почему невозможно растопить яйцо?" (Why can't you melt an egg?). Вы заметили какие-либо существенные улучшения?

На одной из неформальных встреч я провел конкурс по состязательному промптингу. Что примечательно, участники без технического бэкграунда проявили большую изобретательность, чем специалисты в области LLM, включая способность "направить" модель к нежелательным выводам с помощью искусно составленных запросов.

## Доступ к LLM через API

Вероятно, вы уже знакомы с большими языковыми моделями и не раз взаимодействовали с ними через чат-боты, такие как ChatGPT, Gemini или Claude. Давайте теперь рассмотрим, как можно получить доступ к этим моделям через API. В качестве примера мы возьмем OpenAI API, который предоставляет доступ к его семейству моделей GPT. Большинство разработчиков проприетарных моделей предлагают аналогичные возможности через собственные API.

Получить доступ к моделям GPT-4o mini и GPT-4o можно через API Chat Completion компании OpenAI.

Рассмотрим такой пример:

```
import os
import openai
openai.api_key = <INSERT YOUR KEY HERE>

output = openai.ChatCompletion.create(
  model="gpt-4o-mini",
  messages=[
    {"role": "system", "content": "You are an expert storywriter."},
                                  // (Ты опытный писатель)
    {"role": "user", "content": "Write me a short children's story
    about a dog and an elephant stopping
    being friends with each other."}
    // (Напиши небольшой детский рассказ о том,
    // как собака и слон перестали дружить друг с другом)
  ]
)

print(output.choices[0].message)
```

Существуют четыре основных типа ролей: система (system), пользователь (user), ассистент (assistant) и инструмент (tool).

- Роль системы предназначена для определения общего промпта.
- Роль пользователя относится к входящим пользовательским данным.
- Роль ассистента относится к ответам модели.
- Роль инструмента предназначена для взаимодействия со сторонними программными инструментами.

Подробное описание инструментов представлено в *главе 10*.



В чем отличие системных ролей от пользовательских? Какими должны быть инструкции в системных промптах, а какими — в пользовательских? Системные промпты задают общие рамки поведения LLM на высоком уровне, например, определяя его специализацию: "Ты финансовый консультант, специализирующийся на составлении отчетности для юридических лиц". При диалоге с пользователем системный промпт может дополнительно направить модель, передав ей дополнительные инструкции, помимо самого запроса. Исследования выявили, что позиция инструкций (в системном или пользовательском промпте) не играет решающей роли. Главными факторами оказываются длина и объем инструкций, т. к. большие языковые модели, как правило, могут одновременно обрабатывать лишь ограниченное число команд. Более высокие шансы на выполнение имеют инструкции, размещенные в начале или в конце промпта.

Рассмотрим параметры, предоставляемые OpenAI в свободном доступе.

`n`

Количество завершений, которое модель должна сгенерировать для каждого ввода. К примеру, при значении `n = 5` она создаст пять различных вариантов детских рассказов.



В задачах, требующих высокой надежности, следует генерировать не один, а несколько вариантов завершения, т. е. $n > 1$. Затем с помощью функции постобработки, которая может использовать возможности LLM, выбирается наиболее подходящий вариант. Такой подход обусловлен тем, что LLM работает с распределением вероятностей, и не всегда будет гарантирован наилучший результат, т. к. неверный/неудовлетворительный ответ может быть просто случайным выбором токенов. В реальности может оказаться необходимым найти золотую середину между этим процессом и ограничениями в средствах.

`stop` *и* `max_completion_tokens` *(максимальное ограничение токенов)*

Эти параметры регулируют длину генерируемого текста. С помощью параметра `stop` можно задать конечные токены, по достижении которых процесс генерации текста будет остановлен. Примером остановки последовательности служит

*токен новой строки* (newline token). Когда требуется соблюдать определенный формат вывода, например, пронумерованного списка предложений, то для остановки генерации после вывода заданного количества предложений достаточно указать конечное число в качестве значения параметра `stop`.

`presence_penalty` *(штраф за появление) и* `frequency_penalty` *(штраф за частоту появления)*

Эти параметры используются для предотвращения повторения генерируемых данных. Штрафуя вероятность для токенов, уже встречающихся в выходных данных, можно добиться того, чтобы модель не воспроизводила одни и те же фрагменты, обеспечивая более оригинальный результат. Данные параметры могут быть полезны при выполнении творческих задач.

`logit_bias`

С помощью параметра `logit_bias` можно повышать или понижать вероятность генерации определенных токенов.

`top_p` *и* `temperature`

Оба параметра определяют стратегии декодирования, которые применяются при работе с LLM. Модели генерируют распределение вероятностей токенов, и на основе этого распределения выбирается следующий токен для генерации. Существует ряд подходов к выбору следующего токена, исходя из полученного распределения вероятностей, которые будут подробно рассмотрены в *главе 5*. Важно понимать, что повышение температуры установки способствует более креативным и разнообразным результатам, в то время как понижение температуры приводит к более предсказуемым текстам. Рекомендации по оптимальным значениям для разных сценариев использования можно найти в шпаргалке по адресу **https://oreil.ly/DAa66**.

`logprobs`

Предоставляет наиболее вероятные токены для каждого выходного токена, снабженные их логарифмическими вероятностями. Важно отметить, что OpenAI ограничивает список предложенных токенов двадцатью наиболее вероятными вариантами. В последующих главах мы детально рассмотрим, как использовать информацию `logprobs` в различных контекстах.

### Упражнение

Используя API OpenAI, предоставьте модели фрагмент реального комментария футбольного матча между "Реал Мадрид" и "Барселоной" (**https://oreil.ly/NRwIb**). При желании можно заменить эти клубы любыми другими. Затем попросите модель продолжить комментарий.

Поэкспериментируйте с параметрами промптов `temperature`, `logit_bias`, `presence_penalty` и `frequency_penalty`, чтобы добиться максимально точной имитации стиля комментаторов. Насколько успешно модель смогла передать характерные особенности реальных комментариев?

## Преимущества и недостатки больших языковых моделей

Понимание сильных и слабых сторон больших языковых моделей — ключевой навык для создания действительно полезных приложений. Книга предоставит читателю необходимую теоретическую базу, а практика — возможность развить интуитивное понимание этих особенностей. В целом LLM демонстрируют наибольшую эффективность в задачах, ориентированных на обработку языка. Они практически никогда не допускают орфографических или грамматических ошибок, что является значительным шагом вперед по сравнению с предыдущими методами интерпретации пользовательских инструкций и намерений. Эти модели также демонстрируют впечатляющие результаты в различных областях обработки естественного языка, таких как извлечение сущностей и отношений, а также распознавание именованных сущностей (NER). Отдельно стоит отметить их выдающуюся способность генерировать код, где LLM добились наибольших успехов, что подтверждается, например, популярностью инструмента GitHub Copilot (**https://oreil.ly/qvriE**).

Основная проблема большинства LLM заключается в их ограниченных интеллектуальных способностях. Даже самые современные модели демонстрируют существенные ограничения в области рассуждений, охватывающих арифметику, логику и здравый смысл (более формальное определение рассуждений будет представлено в *главе 8*). Кроме того, большие языковые модели не могут гарантировать точность информации из-за отсутствия прямого взаимодействия с реальным миром. В связи с этим они могут создавать текст, не всегда соответствующий действительности и установленным правилам, что в просторечии именуется галлюцинациями (hallucination). Галлюцинации — это существенный недостаток LLM и одна из главных причин, по которой их применение вызывает сомнения. *В главе 8* мы подробно разберем различные подходы к борьбе с галлюцинациями и рассмотрим ограничения рассуждений.

Ежедневно в Интернет поступает огромное количество статей, созданных с помощью LLM, и многие из них проникают в верхние строчки поисковых систем. К примеру, на запрос "Can you melt eggs?" ("Возможно ли растопить яйцо?") Google вывел ответ (**https://oreil.ly/ivvv_**): "Yes, an egg can be melted" ("Да, это возможно"), вследствие появления сгенерированной ИИ веб-страницы, содержащей неверную информацию. Такой текст называется *AI slop* — контент низкого качества. Поэтому для поисковых систем существует мощная мотивация для точного выявления текста, созданного ИИ. Важно помнить, что поскольку обучение LLM основано преимущественно на текстах веб-страниц, существует вероятность, что и будущие модели могут содержать в себе загрязнения, свойственные этому тексту.

Хотя большие языковые модели все чаще используются в творческих проектах, им все еще далеко до профессиональных авторов. Вероятность того, что роман, написанный с помощью LLM, покорит сердца читателей и станет бестселлером, невелика. Сгенерированный моделью текст не обладает той же глубиной и оригинально-

стью, которая позволяет произведениям авторов вызывать сильные эмоциональные отклики у читателей. В определенный момент при чтении текста, сгенерированного моделью, его искусственное происхождение становится очевидным.

### Расширение возможностей LLM

Несмотря на то что список задач, с которыми не справляются LLM, сокращается, некоторые функции, например *самореференция* (self-reference), по-прежнему остаются за пределами их возможностей.

В своей работе "Я — странная петля" Дуглас Хофштадтер (Douglas Hofstadter "I Am a Strange Loop", **https://oreil.ly/uttDL**) рассматривает самореференцию как показатель интеллекта высшего порядка, а возможно, и сознания. В качестве примера самореферентного утверждения он приводит фразу: "The penultimate word in this sentence is is" ("Предпоследним словом в этом предложении является „является“").

Команда Траша (Thrush et al., **https://oreil.ly/rHQH1**) представила специальный набор данных под названием "I am a strange dataset" ("Я — странный набор данных", **https://oreil.ly/3hX9w**), содержащий ряд металингвистических вопросов. Протестируйте следующий вопрос на разных моделях и оцените, насколько они его правильно интерпретируют:

> This this is is a a new new form form of of poetry poetry where where every every word word is is repeated. Are any words not repeated?
>
> (Это это является является новой новой формой формой поэзии поэзии, где где каждое каждое слово слово повторяется. Повторены все слова?)

К моменту публикации книги ни одна существующая модель, в том числе OpenAI o1, не смогла дать на него верный ответ — они не указали на отсутствие повторения последнего слова.

У каждой большой языковой модели есть своя уникальная "подпись" в генерируемом тексте, которая может проявляться в разной степени для читателя. Например, в текстах, созданных ChatGPT, можно заметить избыточное использование слов вроде *delve* (вникать), *tapestry* (обивка), *bustling* (спешка) и т. д. Также ChatGPT часто дополняет предложения дополнительными пояснениями в конце, как, например, в конструкции: "He ate the entire pizza, indicating he was hungry" ("Он съел всю пиццу, что свидетельствует о его голоде"). Или, например, "The vampire sent a thousand text messages in a month, suggesting effective use of digital technologies" ("Вампир за месяц отправил тысячу текстовых сообщений, что указывает на грамотное владение цифровыми инструментами"). Но с уверенностью утверждать, был ли текст написан ИИ, крайне сложно (**https://oreil.ly/4skjI**). Нечестные пользователи используют различные методы маскировки, например, просят языковую модель перефразировать текст, первоначально сгенерированный другой системой, чтобы скрыть его искусственное происхождение.

Таким образом, выявить плагиат становится все сложнее. Участившиеся случаи ложных обвинений студентов в плагиате, обусловленные несовершенством детекторов ИИ-текста (**https://oreil.ly/hetca**), заставили учебные заведения переосмыслить подходы к оценке учащихся, сокращая удельный вес письменных работ в этом процессе. Показательно, что студенты являются одними из самых активных пользователей продуктов LLM, что подтверждается падением активности ChatGPT (**https://oreil.ly/5xECI**) в летний период.

Хотя слова, такие как *delve* (вникать), часто встречаются в сгенерированных моделью текстах, простой подсчет их частоты не дает однозначной гарантии авторства. Определить, был ли текст создан LLM, можно лишь с помощью более сложных методов. Например, мое личное употребление слова *delve* (которое как носитель индийского английского языка я использую чаще, чем естественный носитель английского) уже было заметно в моих работах и публикациях, вышедших до появления ChatGPT. Данные факты свидетельствуют о неотложной необходимости создания более совершенных методик определения ИИ-текста.

Один из перспективных подходов основан на использовании синтаксических шаблонов, представляющих собой последовательности токенов с заданной структурой тегов частей речи (part-of-speech, POS), обычно состоящие из 5–8 токенов. Исследование Шантал Шаиб и соавт. (Shaib et al., **https://oreil.ly/n_dXJ**) показало, что даже при использовании методов генерации текста, направленных на повышение его разнообразия, некоторые такие шаблоны остаются неизменными в итоговом результате. Хотя эти стратегии, называемые также стратегиями декодирования (*см. главу 5*), и могут существенно снизить частоту повторения шаблонов, полностью исключить их появление не всегда удается. Формирование таких шаблонов происходит на ранних этапах предобучения модели.

Например, рассмотрим такой шаблон:

VBN IN JJ NNS: VBN (причастие прошедшего времени) + IN (предлог) + JJ (прилагательное) + NNS (существительное во множественном числе).

Соответствующие этому шаблону фразы будут выглядеть следующим образом:

- Engaged in complex tasks (Привлекавшийся к сложным заданиям);
- Trained in advanced techniques (Обучавшийся по передовым методикам);
- Entangled in deep emotions (Запутавшийся в глубоких переживаниях);
- Immersed in vivid memories (Погрузившийся в яркие переживания).

Невольно складывается впечатление, что некоторые LLM довольно часто и порой чрезмерно используют этот шаблон?

### Упражнение

Запросите LLM написать эссе на любую выбранную вами тему. Затем с помощью библиотеки разнообразия (**https://oreil.ly/6LyNs**) найдите и извлеките из эссе пять наиболее часто встречающихся синтаксических шаблонов. Для этого можно воспользоваться функцией `extract_patterns` с параметром `n = 5`.

Какие синтаксические шаблоны встречаются чаще всего? Попробуйте проанализировать авторское эссе из любой публикации по вашему усмотрению и определить его характерные синтаксические шаблоны. Существуют ли различия в частоте встречаемости этих шаблонов? А как обстоят дела с их относительной частотой?

## Разработка прототипа чат-бота

Итак, приступим к практической реализации!

В последние годы мы стали свидетелями бурного развития благоприятной экосистемы библиотек, существенно упростившей разработку и тестирование приложений с применением LLM. Теперь можно создать чат-бота *Chat with your PDF*, способного извлекать информацию из PDF-файлов и отвечать на вопросы, можно реализовать, написав всего лишь несколько десятков строк кода.

Разработаем простое приложение, позволяющее пользователям загружать PDF-файлы и общаться с чат-ботом, который сможет отвечать на их вопросы по содержимому документа в диалоговом формате.

Ожидаемый рабочий процесс данного приложения можно представить следующим образом:

1. Пользователь загружает нужный PDF-документ через пользовательский интерфейс.
2. Приложение использует библиотеку синтаксического анализа PDF для обработки PDF-документов и разделения извлеченного текста на более мелкие, удобные для анализа фрагменты.
3. Фрагменты преобразуются в векторные представления, которые называются *эмбеддингами* (embeddings).
4. При отправке пользователем запроса через интерфейс чата этот запрос также преобразуется в векторное представление.
5. Для каждого вектора фрагмента (чанка, chunk) вычисляется его векторное сходство с вектором запроса.
6. Извлекается текст, соответствующий $k$ наиболее близким по смыслу векторам.
7. Извлеченный текст вместе с запросом и другими дополнительными инструкциями передается в LLM.
8. Модель использует эту информацию для составления ответа на запрос от пользователя.
9. Ответ отображается в интерфейсе пользователя.

   Пользователь теперь имеет возможность ответить на вопрос (задать уточняющий вопрос, предложить новый вопрос, выразить благодарность и т. д.).

10. При каждом новом шаге диалога LLM обращается к полной истории предыдущих сообщений.

Этот рабочий процесс изображен на рис. 1.5.

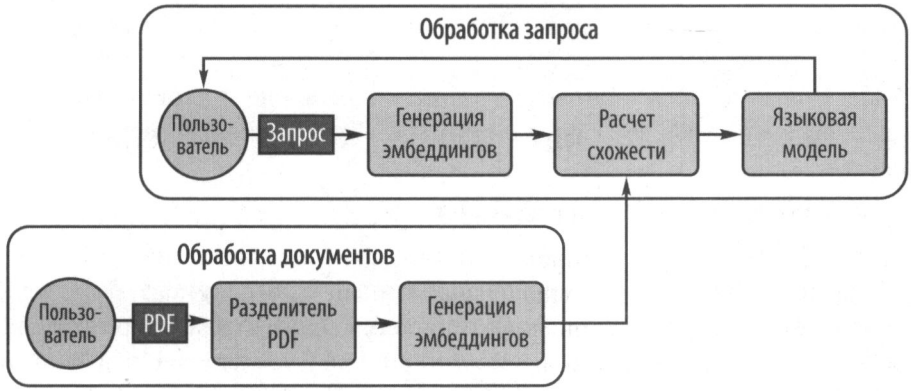



**Рис. 1.5.** Рабочий процесс программы чат-бота

В первую очередь, необходимо установить требуемые библиотеки. Для реализации данной настройки будут использоваться следующие библиотеки.

*LangChain* (**https://oreil.ly/g833p**)

Эта широко используемая инфраструктура служит основой для создания конвейеров приложений с применением LLM.

*Gradio* (**https://oreil.ly/XHqfT**)

С помощью этой библиотеки можно разрабатывать пользовательские интерфейсы для моделей LLM.

*Unstructured* (**https://oreil.ly/sIFEX**)

Это инструмент для анализа PDF-документов, оснащенный различными методами извлечения текста из PDF-файлов.

*Sentence Transformers* (**https://oreil.ly/UyN1k**)

Эта библиотека упрощает процесс создания эмбеддингов на основе текстовых данных.

*OpenAI* (**https://oreil.ly/zbroe**)

Данный API открывает доступ к семейству моделей GPT от компании OpenAI.

Теперь импортируем необходимые библиотеки и функции:

```
!pip install openai langchain gradio unstructured

from langchain_community.document_loaders import UnstructuredPDFLoader
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
```

```
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
import gradio as gr
```

Далее реализуем функцию загрузки и анализа PDF. LangChain поддерживает несколько библиотек синтаксического анализа PDF-файлов, который может быть реализован с помощью различных методов, в том числе с использованием больших языковых моделей. В данном примере будет использоваться библиотека Unstructured:

```
loader = UnstructuredPDFLoader(input_file.name)
data = loader.load()
```

Переменная данных содержит проанализированный PDF-документ, разделенный на абзацы. Каждый абзац называется чанком (chunk). Далее с помощью модели эмбеддинга каждый чанк преобразуется в векторное представление. LangChain поддерживает большое количество моделей эмбеддинга. В данном примере будет использоваться модель *all-MiniLM-L6-V2*, доступная на платформе Hugging Face:

```
embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
```

После загрузки модели эмбеддинга можно создавать векторы из собственных данных и сохранять их в векторной базе данных. LangChain предлагает несколько интеграций с векторными базами данных, однако в нашем примере мы воспользуемся Chroma, известным своей простотой в использовании:

```
db = Chroma.from_documents(data, embeddings)
```

Итак, векторная база с векторами готова! Теперь можно отправлять запросы и получать на них соответствующие ответы.

Например,

```
query = "How do I request a refund?" // (Как получить рефинансирование?)
docs = db.similarity_search(query)
print(docs[0].page_content)
```

Данный код извлекает из PDF-документа абзац, вектор которого максимально близок к вектору, представляющему пользовательский запрос. В связи с тем что векторы кодируют смысловую нагрузку текста, это означает, что абзац с подобным вектором будет содержать контент, схожий с контентом запроса.

Необходимо помнить, что нельзя с абсолютной уверенностью гарантировать, что абзац будет содержать ответ на запрос. Эмбеддинги позволяют лишь находить тексты, сходные с запросом. Даже если найденный текст будет полностью совпадать с запросом, он может не нести в себе ответа или быть неактуальным для него.

Эффективность работы большой языковой модели определяется ее способностью выявлять релевантный контекст и отделять его от нерелевантного. Представим LLM запрос и отрывок текста и попросим модель составить ответ, опираясь на предоставленные данные.

Такой рабочий алгоритм можно реализовать с помощью цепочки в LangChain:

```
conversational_chain =
ConversationalRetrievalChain.from_llm(ChatOpenAI(temperature=0.1),
    retriever=pdfsearch.as_retriever(search_kwargs={"k": 3}))
```

В данном примере используется `ConversationalRetrievalChain`, которая реализует следующий рабочий процесс:

1. Используя предыдущую историю диалога, если она доступна, и текущий ответ/запрос пользователя, система генерирует самостоятельный вопрос.
2. Система применяет выбранный метод для выявления и извлечения $k$ наиболее релевантных чанков.
3. LLM получает в качестве входных данных извлеченные фрагменты, историю диалога, актуальный запрос/ответ пользователя и инструкции, после чего генерирует ответ.

Теперь можно вызвать цепочку действий и внести ее итог в историю диалога:

```
output = conversational_chain({'question': query, 'chat_history':
    conversational_history})
conversational_history += [(query, output['answer'])]
```

Таким образом, чат-бот готов к использованию. Теперь обернем его, соединив с пользовательским интерфейсом. Для этого воспользуемся Gradio (**https://oreil.ly/dzYJv**) — легковесным Python-фреймворком, предназначенным для создания интерактивных пользовательских интерфейсов для LLM.

```
with gr.Blocks() as app:


    with gr.Row():
        chatbot = gr.Chatbot(value=[], elem_id='qa_chatbot').style(height=500)


    with gr.Row():
        with gr.Column(scale=0.80):
            textbox = gr.Textbox(
                placeholder="Enter text"
            ).style(container=False)


        with gr.Column(scale=0.10):
            upload_button = gr.UploadButton("Upload a PDF",
                file_types=[".pdf"]).style()
```

Для реализации обработчиков событий, реагирующих на действия пользователя, потребуется дополнительный код. Полный код книги доступен в репозитории GitHub (**https://oreil.ly/llm-playbooks**).

И наконец, запускаем приложение:

```
if __name__ == "__main__":
    app.launch()
```

Итак, программа чат-бота готова!



Почему нельзя передать в модель PDF-файл целиком, а не разбивать его на чанки, извлекая только релевантную информацию? Эффективность данного подхода определяется максимальной длиной контекста, которую поддерживает LLM, что, в свою очередь, определяет максимальный размер входных данных, которые может обработать модель. Модели с большими параметрами способны обрабатывать контексты достаточной длины, чтобы вместить несколько PDF-файлов целиком. В таком случае фрагментация и эмбеддинг могут вообще не понадобиться.

## От прототипа к производству

Разработка приложений на основе больших языковых моделей не всегда является тривиальной задачей. Несмотря на то что мы создали прототип, который эффективно решает простые задачи, он не способен справиться с более сложными сценариями, требующими высокой точности и надежности. Данная книга призвана преодолеть этот пробел и помочь перейти от создания прототипов к разработке полноценных, работоспособных приложений.

В руководстве по прототипу большая языковая модель рассматривалась как некий черный ящик. Однако при создании реальных приложений с использованием LLM важно понимать, что происходит под капотом, даже если вы не планируете самостоятельно обучать LLM. Именно поэтому *главы 2–4* посвящены детальному изучению всех компонентов, из которых состоят LLM, и процессу их обучения. Понимание того, как устроены большие языковые модели и как они обучаются, будет крайне полезным при устранении различных видов неполадок.

В работе использовалась проприетарная языковая модель компании OpenAI, без анализа ее оптимальности для конкретного применения. Сегодня существует множество языковых моделей, доступных для коммерческого использования, их число насчитывает сотни, а возможно, и тысячи. *Глава 5* посвящена анализу ландшафта LLM, который включает в себя как модели с открытым исходным кодом, так и закрытые коммерческие решения, различные по размеру и функциональности. Мы также определим критерии, по которым можно выбрать наиболее подходящую LLM для конкретного применения. Например, при подборе модели для нашего PDF-чат-бота одним из главных факторов станет ее способность работать в рамках жесткого бюджетного плана. Мы подробно остановимся на возможностях и ограничениях больших языковых моделей, определим их подходящее применение в данном сценарии, разработаем критерии оценки и эталонные наборы данных, а также выявим потенциальные сложности, которые могут возникнуть как при автоматизированной, так и при ручной оценке.

Как быть, если PDF-файлы, предназначенные для загрузки в чат-бот, относятся к узкоспециализированной области, в которой LLM не обладает достаточными знаниями? Что делать, если LLM не может исполнять инструкции из пользовательских промптов? Возможно, решением станет обновление параметров модели путем ее настройки на данные из специализированной области. *В главе 6* мы подробно рассмотрим дообучение, или тонкую настройку модели, определим сценарии ее применения и продемонстрируем процесс создания набора данных для дообучения. Не исключено, что стандартная доводка может оказаться не самым подходящим решением для поставленной задачи. Она может оказаться слишком затратной и неэффективной. *В главе 7* мы рассмотрим альтернативные методы, в частности, *параметрически эффективное дообучение*, или *эффективную тонкую настройку параметров* (parameter-efficient fine-tuning, PEFT), при котором обновляется лишь небольшая часть параметров модели.

В некоторых случаях наш чат-бот может галлюцинировать или столкнуться с трудностями при обработке запросов из-за неточности его рассуждений. *В главе 8* мы подробно разберем методы выявления и снижения вероятности галлюцинаций, а также способы улучшения его способности к рассуждениям, включая различные техники оптимизации времени вычисления.

Чтобы разработать PDF-чат-бота, который успешно будет функционировать в реальных рабочих условиях, важно учесть нетехнические аспекты, например минимизацию времени ожидания ответа модели и сокращение расходов. *В главе 9* мы рассмотрим методы оптимизации вывода, включая кеширование, дистилляцию и квантование.

Будет интересно расширить возможности нашего чат-бота, интегрировав LLM с интерпретаторами кода, базами данных и API. Мы также можем добиться того, чтобы чат-бот справлялся с комплексными запросами, требующими многоступенчатого решения. *В главе 10* мы рассмотрим взаимодействие LLM с внешними инструментами и источниками информации, а также возможность предоставления LLM автономности в выполнении задач, принятии решений и взаимодействии со своей средой.

В изложенном руководстве мы изучили базовый подход к разбору, фрагментации и эмбеддингу документов. Однако при практическом применении мы можем столкнуться с ситуацией, когда оценка схожести векторов оказывается неэффективной, приводя к возврату нерелевантных чанков документов, или же извлеченные фрагменты не несут в себе всей необходимой информации для ответа на запрос. *В главе 11* мы подробно разберем эмбеддинги и методы их дообучения, а также рассмотрим более эффективные методы разделения данных на чанки.

PDF-чат-бот функционирует на основе принципа генерации с дополненным поиском (retrieval-augmented generation, RAG). RAG объединяет возможности больших языковых моделей с доступом к внешним данным, например, PDF-файлам, загружаемыми пользователями, как в нашем случае. *В главе 12* мы подробно разберем архитектуру RAG и научимся создавать надежные системы на ее основе.

В заключительной *главе 13* мы рассмотрим шаблоны проектирования и парадигмы программирования, которые лежат в основе разработки приложений с использованием LLM.

В последующих главах мы также рассмотрим и другие ключевые аспекты. Я с нетерпением жду возможности отправиться вместе с вами в это увлекательное путешествие и верю, что предложенные инструменты, подходы и интуиция станут для вас ценными помощниками в создании LLM-приложений, которые будут успешно применяться на практике!

**Упражнение**

Создайте приложение *Chat with your PDF*, которое позволит загрузить любой PDF-файл с вашего устройства и задавать по нему вопросы. Определите возникающие при этом сложности и запишите их. По мере освоения новых концепций и продвижения по книге обращайтесь к этому приложению и проверяйте, насколько успешно вы можете устранить или снизить ранее выявленные проблемы с помощью приобретенных знаний.

# Резюме

В этой главе мы исследовали мир языковых моделей, проследив их эволюцию от первых шагов до последних достижений и проанализировав их влияние на современный мир. Мы разобрали эффективные методы взаимодействия с моделями, используя приемы промптинга, а также выявили их сильные и слабые стороны. Была продемонстрирована простота создания прототипов приложений на основе этих моделей, а также обозначены ключевые проблемы, которые могут возникнуть при их широком внедрении. В следующей главе мы продолжим путешествие в мир больших языковых моделей и подробно рассмотрим их ключевые компоненты.

# ГЛАВА 2

# Данные для предварительного обучения

*В главе 1* мы познакомились с большими языковыми моделями (LLM), изучили их сильные и слабые стороны, рассмотрели текущие и потенциальные области применения, а также обсудили принципы масштабирования, которые, по-видимому, определяют будущее развитие в этой сфере. В следующих трех главах мы подробно остановимся на техниках предобучения LLM и разберем их компоненты. Однако, прежде чем перейти к использованию предобученных LLM в создании приложений, зададимся вопросом: зачем тратить время на изучение тонкостей подготовки этих огромных моделей с нуля, если для большинства специалистов в области машинного обучения это никогда не станет повседневной задачей?

Эта информация крайне важна, т. к. решения, принимаемые на этапе предварительного обучения, оказывают значительное влияние на успех последующих этапов работы. Как мы увидим далее, понимание механизмов обучения поможет нам глубже разобраться в различных типах неудач. Подобно тому, как мы тщательно изучаем состав продуктов в супермаркете, мы стремимся понять, из чего состоит языковая модель, прежде чем использовать ее в ответственных проектах.



О некоторых проприетарных языковых моделях (LLM), доступных лишь через API, публичная информация весьма ограничена. Данная книга максимально раскрывает доступные сведения о них. Недостаток информации не должен стать препятствием для использования этих моделей, однако прозрачность модели — это важный фактор, который следует учитывать при выборе подходящей модели.

## Компоненты большой языковой модели

Рассмотрим компоненты, необходимые для создания LLM.

В общем случае мы имеем следующее.

*Данные для предобучения, т. е. информация, на основе которой обучается модель*

Популярное в компьютерной науке выражение "мусор на входе — мусор на выходе" (garbage in, garbage out, GIGO) остается актуальным и в современном контексте обучения языковых моделей. В этой главе мы рассмотрим самые распространенные наборы данных, используемые для предобучения моделей, и проанализируем различные этапы предварительной обработки, которые гарантируют подачу *качественной* информации в модель. Также мы рассмотрим инструменты, которые помогут нам изучить эти наборы данных и определить, как состав данных, используемых для предобучения, влияет на производительность решения последующих задач.

*Словарь и токенизатор, т. е. объекты обучения*

Для создания модели, работающей с языком, необходимо сначала определить ее словарный запас и правила разбиения текста на смысловые блоки, которые называются токенами (подробнее об этих понятиях мы поговорим в *главе 3*). В отличие от людей, которые воспринимают язык как набор значимых слов и предложений, языковые модели обрабатывают текст через призму токенов. Мы рассмотрим, как это различие влияет на рабочий процесс.

*Цель обучения, т. е. чему обучается модель*

Цель предварительного обучения языковой модели заключается в передаче ей широкого спектра общих навыков в области синтаксиса, семантики и рассуждений, которые, по нашему мнению, станут основой для решения любых задач, независимо от их специфики. Поэтому задачи обучения должны быть достаточно всеобъемлющими, чтобы охватить все эти фундаментальные умения. *В главе 4* мы рассмотрим различные задачи (цели обучения, learning objectives) для обучения предобученных моделей. Кроме того, мы займемся вопросом, имеют ли LLM преимущество в решении задач, сходных с теми, для которых была создана исходная модель? Мы постараемся дать на него исчерпывающий ответ и рассмотреть, как различные цели обучения влияют на эффективность выполнения таких задач.

*Архитектура, или внутреннее устройство модели*

Структура модели, определяющая взаимоотношения и взаимодействие ее компонентов, а также механизм обработки входной информации, играет фундаментальную роль в ее функционировании. Каждая архитектура несет в себе определенное индуктивное смещение, представляющее собой набор предположений о структуре данных и задачах, для которых она предназначена, тем самым настраивая модель на поиск определенных типов решений. *В главе 4* мы углубимся в архитектуру трансформера (Transformer), которая была кратко представлена в *главе 1* и на сегодняшний день является самой популярной в данной сфере.

Схема взаимодействия компонентов модели представлена на рис. 2.1.




**Рис. 2.1.** Организационная схема компонентов LLM

Модели, созданные по описанному в этой главе алгоритму, именуются *базовыми моделями* (base model). В последнее время наблюдается тенденция к дообучению этих базовых моделей на более узких наборах данных для их адаптации к индивидуальным потребностям и вкусам пользователей. Среди наиболее популярных методов дообучения можно выделить (рис. 2.2):

- *контролируемое дообучение* (supervised fine-tuning, SFT), направленное на то, чтобы модель лучше выполняла указания человека;
- *обучение с подкреплением на основе обратной связи от человека* (reinforcement learning by human feedback, RLHF), чтобы модель лучше соответствовала человеческим предпочтениям;
- *непрерывное предобучение* (continued pre-training) под конкретную область (domain-adaptive) или задачу (task-adaptive), направленное на повышение эффективности модели в узкоспециализированных сферах и решении конкретных задач.

Применение определенных методов аугментации (от лат. *augmentatio* — увеличение, приращение) позволяет создавать модели, которые можно классифицировать как *модели инструкций*, *модели диалога (чат-модели)* и т. д.

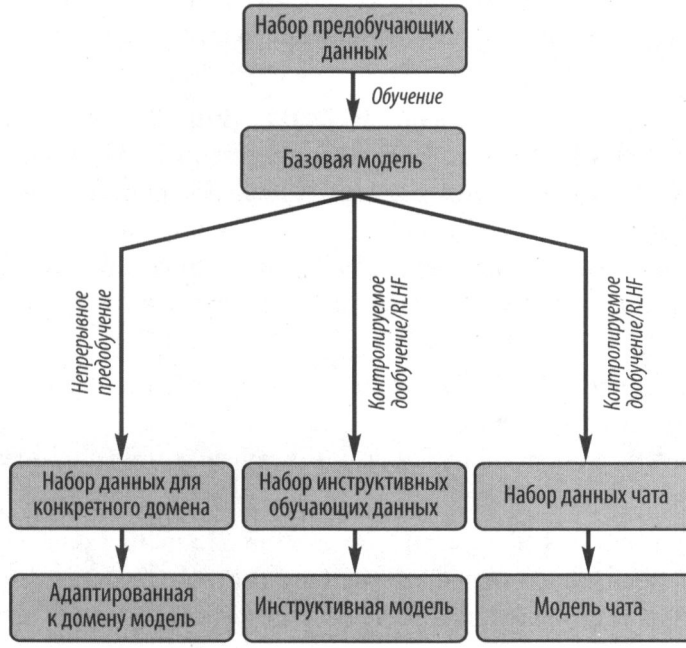



**Рис. 2.2.** Взаимосвязь между базовыми моделями и их модификациями

*В главе 6* мы рассмотрим модели инструкций и чата, а в *главе 7* — непрерывное предобучение для конкретной области или задачи.

### Сложности предобучения LLM

Предварительное обучение больших языковых моделей — это сложная техническая задача, требующая значительных вычислительных мощностей и высоких профессиональных навыков. Например, технический отчет GPT-4 (**https://oreil.ly/wcAhT**) отмечает вклад 343 уникальных специалистов, не считая аннотаторов (**https://oreil.ly/hLJno**) из Кении, которые участвовали в обучении модели по методике RLHF (**https://oreil.ly/KxSI1**). Каждый аспект предварительного обучения LLM заслуживает отдельного, глубокого исследования. В данной главе мы не будем останавливаться на технических нюансах инфраструктуры и инженерных подходов, задействованных в процессе предобучения LLM, а также на деталях распределенных и параллельных вычислений. Мы сфокусируемся на тех ключевых моментах предобучения, которые напрямую влияют на функциональность и производительность конечного приложения.

Чтобы получить более глубокое понимание проблем, связанных с предобучением LLM, рекомендую обратиться к следующим полезным ресурсам.

- В блоге Big Science (The Technology Behind BLOOM Training, **https://oreil.ly/9BYNz**) представлен подробный анализ процесса обучения BLOOM — большой многоязыковой модели с открытым исходным кодом, содержащей 176 млрд параметров. В статье раскрываются нюансы, связанные с используемым оборудованием, типами параллелизма и оптимизационными стратегиями, которые применялись при ее обучении.
- Хронология обучения (журнал) BLOOM (**https://oreil.ly/j6UT8**) и OPT (**https://oreil.ly/3W3TH**) с подробной информацией о разработке большой языковой модели от Meta, содержащей 175 млрд параметров. В ней подробно освещаются испытания и трудности, с которыми столкнулись разработчики во время обучения модели, начиная с аппаратных неполадок и методов их устранения и заканчивая такими сложностями, как нестабильность обучения, скачки значений функции потерь и др.
- Видео "Open Pretrained Transformers" ("Открытые предобученные трансформеры", **https://oreil.ly/lGwPa**), где ведущий автор OPT Сьюзан Чжан (Susan Zhang) делится своими наблюдениями о ключевых этапах создания и эволюции модели.
- Серия блогов Imbue (**https://oreil.ly/OgYS9**) с детальным описанием процесса обучения модели, содержащей 70 млрд параметров, начиная с этапа ее создания и заканчивая ее нынешним состоянием.

# Требования к данным для предобучения

Несмотря на то что модели с большей емкостью демонстрируют более высокую эффективность на обучающих выборках (**https://oreil.ly/PbN6F**), в целом современные языковые модели остаются весьма неэффективными, требуя огромного количества примеров для обучения под конкретную задачу. Из-за невозможности создания огромного контролируемого набора данных с аннотациями людей преобладающим методом предобучения языковых моделей становится *самоконтролируемое обучение* (self-supervised learning), где целевые метки (target label) уже содержатся в самих обучающих данных.

Этот метод позволяет использовать практически любой вид текста для обучения модели, а любые нетекстовые сигналы с определенной структурой можно в теории преобразовать в текстовую форму и добавить в обучающий набор данных.

Как выяснилось в *главе 1* при рассмотрении принципов масштабирования, производительность модели можно улучшить, увеличивая время ее обучения и объем обучающих данных. Практические примеры подтверждают эффективность *консолидации* (consolidation), что все больше внушает надежду на создание универсальной языковой модели, способной управлять всем жизненным циклом, от создания до завершения. Предполагается, что универсальная модель сможет решать фактические вопросы, связанные с реальным миром, посредством применения арифметических и логических рассуждений, генерировать программный код и выдвигать новые оригинальные креативные концепции.

Необходимость в обширных обучающих данных для языковых моделей очевидна. Однако актуальным остается вопрос: обладают ли существующие текстовые ресурсы достаточной глубиной и релевантностью для того, чтобы обучить LLM всем необходимым компетенциям?

Важно понимать, что языковые модели, обучающиеся только на тексте, оперируют исключительно на уровне лингвистической формы, т. е. рассматривают текст как последовательность символов, например, "Walter White tossed the pizza onto the roof" ("Уолтер Уайт забросил пиццу на крышу"). Установить же смысл такого высказывания невозможно без связи лингвистической формы с коммуникативной целью (communicative intent) автора. Несмотря на мнение исследователей, утверждающих, что смысл не может быть извлечен только из формы (**https://oreil.ly/3iYA2**), современные языковые модели все чаще опровергают эту точку зрения.

Для того чтобы получить полную картину, языковая форма должна быть тесно увязана с объективной реальностью. В когнитивной науке метод, направленный на установление такой связи, называется "заземлением" (grounding):

> "Процесс определения информации, которая необходима для налаживания продуктивного диалога между двумя собеседниками".
>
> *Кьяти Рагхави Чанду.*
> *Заземление "заземления" в обработке естественного языка*
> *(Chandu et al. Grounding 'grounding' in NLP, **https://oreil.ly/kPyXu**)*

Человеческий текст часто носит абстрактный характер и не всегда прямо указывает на свой смысл, опираясь на множество коммуникативных целей, которые выходят за пределы контекста. Понимание такого текста требует от читателя/слушателя не только логического мышления, но и широких знаний о мире, а также умения распознавать и интерпретировать эмоциональные нюансы.



Согласно оценкам экспертов, только 12% информации (**https://oreil.ly/jg4tW**) извлекается непосредственно из текста. На этот счет существуют различные теории, пытающиеся объяснить особенности нашего общения. Одна из них — *принцип наименьшего усилия*, или *закон Ципфа* (Zipf's law), который гласит, что "люди будут следовать по пути, который обеспечивает наибольшее вознаграждение при затрате наименьших усилий".

Многие исследования в области NLP посвящены заземлению языковых моделей к реальному миру (**https://oreil.ly/PbIhT**). Среди перспективных разработок в этой сфере выделяются мультимодальные модели (multimodal model, **https://oreil.ly/ysAeM**), которые обладают способностью интегрировать разные типы информации, включая изображения, видео, речь и текст. Такая универсальность делает их перспективными для широкого спектра применений в будущем. Представьте модель, способную не только распознать слово "пицца" в тексте, но и сформировать в своем представлении ее визуальную картину, а также передать ее вкусовые нюансы.

Но действительно ли мультимодальные модели являются ключом к решению проблемы заземления? Нельзя ли добиться аналогичного эффекта, просто обучая модель на огромном массиве разнопланового текста? Это сложные вопросы, на которые нет однозначного ответа, что подтверждают текущие дискуссии (**https://oreil.ly/oacht**).

Несмотря на то что языковые модели, обученные на больших текстовых массивах, демонстрируют впечатляющие результаты, остается открытым вопрос об их способности освоить навыки, требующие логического мышления. В сети можно встретить множество текстов, иллюстрирующих различные этапы рассуждения, начиная от формальных математических доказательств и заканчивая объяснениями шуток и пошаговыми решениями логических задач. Тем не менее ограниченность лексического материала вынуждает искать альтернативные подходы, например, использовать промптинг с цепочкой рассуждений, о котором подробнее рассказывается в *главе 5*. Недавние исследования (**https://oreil.ly/QInTp**) показали, что предоставление обратной связи на каждом этапе решения задачи, а не только по ее завершению, может повысить уровень арифметической логики модели.

Языковые модели должны освоить ключевой навык — умение справляться с неоднозначностью, свойственной человеческой речи. Согласно закону Ципфа, который гласит о стремлении к минимальным затратам, неоднозначность позволяет говорящим находить баланс между эффективностью и ясностью в коммуникации. Мы часто допускаем неполноту информации, полагаясь на общие знания и контекст, чтобы слушатель мог правильно понять сказанное и восстановить пропущенные детали.

Ранее языковые модели испытывали значительные сложности в моделировании неоднозначности. В своем курсе по обработке естественного языка я часто привожу

в пример эту фразу, чтобы наглядно продемонстрировать двусмысленность языка: "WWE's John Cena surprises Make-A-Wish 7-year-old with cancer"[1].

Хотя современные модели успешно интерпретируют предложения, не принимая Джона Сину за распространителя болезней, недавнее исследование (**https://oreil.ly/BrSwb**) выявило, что даже самые совершенные модели сталкиваются с трудностями в понимании двусмысленности в более широком смысле. Остается вопрос: достаточно ли для языковых моделей увеличения их размера и объема обучающих данных, чтобы они могли полноценно моделировать неоднозначность?

Если единственный способ устранить все эти недостатки — масштабирование наборов данных, то возникает вопрос: достаточно ли в мире данных для обучения LLM новым навыкам? Не столкнемся ли мы в скором времени с нехваткой обучающих материалов? В некоторых профессиональных кругах существует мнение, что у нас уже есть все необходимые данные. Но пока что недостаток необработанных данных не представляет собой серьезной проблемы для моделей обучения. Для иллюстрации можно привести пример миллиардов общедоступных документов, не вошедших в состав большинства обучающих наборов данных (стенограммы парламентских заседаний, судебные акты, подавляющее число заявок в Комиссию по ценным бумагам и биржам (SEC)), которые можно получить через открытые API или посредством веб-скрапинга[2]. В исследовании "How much LLM training data is there, in the limit?" ("Каков предел доступных обучающих данных для LLM?", **https://oreil.ly/XnmHL**), проведенном группой Educating Silicon, был проанализирован общий объем доступных текстовых данных в современном мире. Несмотря на все достижения, приходится признать, что объема естественных данных, требуемых для обучения наших моделей, попросту недостаточно.

Исследователи все активнее изучают возможность использования текста, сгенерированного моделями LLM (*синтетических данных*), для обучения новых моделей. Хотя обучение на синтетических данных может исказить представление модели о реальном распределении данных (**https://oreil.ly/RdzX0**), в этой главе мы подробно рассмотрим процесс создания таких данных для предварительного обучения моделей.

Важно понимать, что сгенерированные данные отличаются по качеству. Высококачественные данные (high-quality data) способствуют созданию более эффективных выборок, сокращая при этом требуемый объем исходного набора. Улучшить качество данных можно посредством предварительной обработки (preprocessing), которая включает в себя фильтрацию некачественных данных и оптимизацию имеющихся. Определение критериев качества данных — это непростая задача, которую мы рассмотрим более подробно *далее в этой главе*.

---

[1] "Джон Сина, знаменитый рестлер WWE, исполнил желание 7-летнего мальчика, страдающего онкологическим заболеванием". В другом контексте эту фразу можно интерпретировать как "Джон Сина, знаменитый рестлер WWE, исполнил желание 7-летнего мальчика, заразив его онкологическим заболеванием". — *Прим. пер.*

[2] От англ. *scraping* — процесс автоматического извлечения данных с веб-сайтов с использованием специальных программ или ботов. — *Прим. перев.*

### Обучение с несколькими эпохами

Обучение модели на полном наборе данных в рамках одного цикла называется эпохой. Ранние версии больших языковых моделей ограничивали обучение одной эпохой или даже меньшим количеством итераций, чтобы предотвратить переобучение. Но можно ли добиться лучшей производительности, увеличивая объем данных и обучая модель в течение нескольких эпох на неизменном наборе данных? В этом случае модель будет неоднократно проходить по всем данным в процессе обучения.

Как оказалось, это вполне реально. Исследования Мюннигоффа и соавт. (Muennighoff et al., **https://oreil.ly/d-PUZ**) демонстрируют, что языковые модели могут добиться высокой точности обучения уже за четыре-пять эпох. После этого дополнительное увеличение числа проходов по обучающей выборке не оказывает существенного влияния на повышение производительности. Таким образом, ограниченность качественных данных не позволяет проводить обучение на протяжении многих эпох и рассматривать такой процесс как полноценный день обучения. Несмотря на стремление к качеству, приходится использовать массивные, но менее качественные данные, поскольку доступ к высококачественным данным и их многократное использование без потери актуальности ограничены.

В своем исследовании Сюэ и соавт. (Xue et al., **https://oreil.ly/m1Suq**) выяснили, что большие языковые модели склонны к переобучению при обучении в несколько эпох. Авторы также пришли к выводу, что использование методов регуляризации, например *исключение*, или *дропаут* (dropout, **https://oreil.ly/Wo5kV**), может частично минимизировать эту проблему.

В работе Гояла и соавт. (Goyal et al., **https://oreil.ly/tafHo**) проведен количественный анализ соотношения между качеством и количеством результатов при многоэтапном обучении. В результате исследования предложен новый набор законов масштабирования фильтрации данных.

## Популярные наборы предобучающих данных

Значительный объем текстовой информации недоступен широкой аудитории. К этой категории относятся данные, предоставляемые через платные API, зашифрованные парольными системами аутентификации пользователей, а также платные книги и документы, многие из которых до сих пор не переведены в цифровой формат. Гиганты вроде Google и OpenAI располагают финансовыми ресурсами, позволяющими им приобретать необходимые данные. В качестве примера можно привести соглашение OpenAI с *Wall Street Journal*, *Financial Times* и другими новостными агентствами, в рамках которого компания получила доступ к их данным, заплатив за это огромную сумму в сотни миллионов долларов. В некоторых случаях текстовые данные, специфичные для конкретной сферы, остаются в собственности компании и доступны лишь крупным игрокам (например, чат-бот BloombergGPT, частично обученный на финансовых данных агентства Bloomberg). Но даже для

моделей, разработанных крупными корпорациями, значительное количество обучающих данных все равно заимствуется из открытых источников.

Далее мы рассмотрим несколько популярных наборов предобучающих данных общего назначения (general-purpose pre-training dataset), которые используются в предварительном обучении больших языковых моделей. Эти наборы данных являются лишь частью доступных ресурсов, хотя большинство LLM, включая модели с закрытым исходным кодом, используют, по крайней мере, значительное количество данных из этих источников. Более подробно наборы данных, специфичных для отдельных отраслей, таких как социальные сети, финансы, биомедицина и другие, будут рассмотрены в *главе 7*.



Большинство универсальных языковых моделей (general-purpose model) предназначены для решения широкого круга задач и могут выступать в качестве многофункционального инструмента. При этом, если в набор обучающих данных были включены данные, соответствующие конкретному сценарию использования, модель может демонстрировать более высокую эффективность в выполнении последующих задач по сравнению с моделями, не имевшими таких данных в своем обучении. Это верно даже в том случае, если данные для предобучения не были помечены. Поэтому, если LLM будет использоваться в четко определенных сценариях, характерных для конкретной области, модели, специализирующиеся на этой сфере, могут оказаться наиболее подходящими. Чтобы эффективно использовать этот феномен, можно реализовать *непрерывное предобучение* (continued pre-training) *с учетом специфики конкретной области* или *задачи*. Более подробно этот подход будет рассмотрен в *главе 7*.

Рассмотрим несколько примеров популярных источников данных, применяемых в обучении универсальных языковых моделей.

*Common Crawl и C4*

Интернет выступает главным поставщиком открытых текстовых данных, предоставляя значительную часть информации для предварительного обучения моделей. Некоммерческая организация Common Crawl (**https://oreil.ly/dhBvu**) осуществляет сканирование Интернета и ежемесячно публикует для широкой аудитории снимки (или снапшоты — snapshot) веб-страниц, собранных поисковыми роботами (или веб-краулерами — web-crawlers) при сканировании сайтов. Несомненно, такой массив необработанных данных требует тщательной очистки перед применением. В ответ на эту потребность компания Google разработала C4 (Colossal Clean Crawled Corpus) — англоязычный корпус данных объемом 750 Гбайт, являющийся результатом обработки и фильтрации данных Common Crawl 2019 года. Компания Google выпустила код для работы с C4, который Джесси Додж и соавт. (Jesse Dodge, **https://oreil.ly/bxmVR**) затем адаптировали и сделали общедоступным, что позволило воспроизвести C4 и предоставить доступ к нему широкой аудитории. На C4 были обучены несколько известных больших языковых моделей, включая все модели семейства T5.

*The Pile*

The Pile (*букв.*: "куча", "кипа", "пакет") — это массив данных, созданный компанией Eleuther AI (**https://oreil.ly/7UAcY**), который содержит 825 Гбайт текстовых материалов из различных источников. Разнообразие данных является ключевым фактором, поскольку немаркированные данные из разных областей, используемые в процессе предварительного обучения модели, положительно влияют на ее последующую производительность в этих областях. Кроме того, разнообразие данных позволяет модели более эффективно решать задачи и применять знания в областях, с которыми она ранее не сталкивалась. Для достижения этой цели данные для The Pile собираются не только из таких источников, как Common Crawl, но и из множества других источников, включая PubMed Central, arXiv, GitHub, FreeLaw Project, Stack Exchange, Ubuntu IRC, HackerNews, YouTube, PhilPapers, NIH ExPorter, Project Gutenberg, Ведомство по патентам и товарным знакам США (USPTO), Википедия и многие другие. Данные из The Pile и его подмножеств использовались в качестве обучающей выборки для обучения многих больших языковых моделей, в том числе модели Llama (**https://oreil.ly/_8eOD**).

*WebText / OpenWebText / OpenWebText2*

Этот набор данных представляет собой подмножество веб-текстов, извлеченных из страниц, связанных с Reddit. Он включает в себя лишь те страницы, которые не только имеют высокую оценку (не менее трех единиц кармы), но и демонстрируют заметный перевес голосов "за" над "против". Предполагается, что такой отбор страниц, которые привлекли внимание пользователей, обеспечит обучение моделей на более качественных данных, что, в свою очередь, позволит достичь более точных результатов. Среди моделей, которые были успешно обучены на этом наборе, можно выделить GPT-2 и GPT-3.

*Википедия*

Википедия служит основным источником знаний для обучения большинства универсальных LLM. Ее полный архив представляет собой структурированный энциклопедический текст, наполненный богатой фактологической информацией. Несмотря на то что редакторская система Википедии обеспечивает упорядоченность и формальность контента, ее стилистическое разнообразие остается ограниченным. Все статьи написаны в едином, официальном стиле. Обучение базовой языковой модели только на основе Википедии не позволяет достичь желаемого уровня качества. Для ее совершенствования необходимо дополнить информацию из различных источников, которые отличаются по стилю и манере изложения.

*BooksCorpus / BooksCorpus2*

Очевидно, этот набор данных оказал наибольшее влияние на обучение моделей. Он стал основой для обучения таких известных моделей, как BERT, RoBERTa, GPT-2 и GPT-3, а также многих других. В BooksCorpus входит более 7000 бесплатных книг, в основном художественной литературы, написанной неопубликованными авторами. Из оригинального набора данных 26% книг относятся к

жанру романа. В The Pile можно найти реплику BooksCorpus под названием BooksCorpus2.

*FineWeb*

На момент написания книги крупнейшим в мире общедоступным набором данных для предварительного обучения являлся FineWeb (**https://oreil.ly/1GyZd**). Он был опубликован Hugging Face и содержит 15 трлн токенов, полученных из 96 снапшотов Common Crawl после тщательной очистки и фильтрации. Hugging Face также выпустила FineWeb-Edu (**https://oreil.ly/8XHH-**) — подмножество FineWeb, состоящее из образовательных данных. Это подмножество играет ключевую роль в обеспечении прохождения большой языковой моделью стандартных тестов и популярных бенчмарков.

### Обучающие данные исчезают из открытого доступа в Интернете

К сожалению, многие упомянутые источники данных оказались вовлечены в судебные тяжбы, связанные с вопросами авторских прав. Несколько книг из набора данных BooksCorpus имеют лицензии, ограничивающие использование авторского материала (**https://oreil.ly/hGmnj**). Первоначальный состав данных теперь не доступен публично. Подобная ситуация сложилась и с оригинальной версией The Pile, которую больше невозможно скачать из-за наличия в ней защищенного авторским правом контента. Теперь стала доступна версия The Pile, из которой такой контент удален (**https://oreil.ly/6oPSC**).

Все чаще веб-сайты пересматривают свои условия использования, внося в файлы *robots.txt* изменения, направленные на запрет использования их контента для обучения искусственного интеллекта. Эти изменения варьируются от ограничений для конкретных организаций, например OpenAI, до полного запрета на любой вид обучения ИИ. Особое внимание уделяется ресурсам с высококачественными обучающими данными: новостные порталы, форумы и социальные сети часто вводят самые жесткие ограничения на доступ к своей информации для целей обучения ИИ. В ходе исследования Шэйна Лонгпре и соавт. (Longpre et al., **https://oreil.ly/qPSOj**) было установлено, что 5% всех токенов в C4 и 28% ключевых данных C4 недоступны для обучения ИИ.

Важно подчеркнуть, что *robots.txt* не имеет статуса официального законодательного стандарта, и компании все еще нарушают условия обслуживания своих вебсайтов, утверждая, что использование данных для обучения языковых моделей основано на "добросовестном использовании". Судебная практика пока не дала однозначного ответа на вопрос о принадлежности авторских прав на размещенные данные, ведь в большинстве случаев на веб-сайтах публикуются лишь данные, а авторские права на них остаются за создателями контента.

В табл. 2.1 приведен список наиболее популярных наборов данных с указанием их размера, года публикации и доступных способов получения.

**Таблица 2.1.** Популярные обучающие наборы данных

| Название | Источники данных | Размер | Год релиза | Доступность | Модели, использующие данный набор данных |
|---|---|---|---|---|---|
| C4 | Common Crawl | 750 Гбайт | 2019 | Да (воспроизведенная версия) | T5, FLAN-T5, UL2, Llama и т. д. |
| The Pile | Common Crawl, PubMed Central, Wikipedia, arXiv, Project Gutenberg, Stack Exchange, USPTO, GitHub и т. д. | 825 Гбайт | 2020 | Да | GPT-NeoX, GPT-J, Cerebras-GPT, StableLM, Pythia и т. д. |
| RedPajama | Common Crawl, GitHub, Wikipedia, arXiv, Stack Exchange и т. д. | 1,2 трлн. токенов | 2023 | Да | Red Pajama-INCITE, MPT |
| BooksCorpus | Данные собраны с сайта smashwords.com | 74 млн предложений | 2015 | Оригинальная версия больше не доступна | Большинство моделей, в том числе BERT, GPT и т. д. |
| OpenWebText2 | Исходящие ссылки Reddit | 65 Гбайт | 2020 | Да | GPT-2, GPT-3 |
| ROOTS | Big Science Catalogue, Common Crawl, GitHub | 1,6 трлн токенов | 2022 | Нет (доступ предоставляется только по запросу) | BLOOM |
| RefinedWeb | Common Crawl | 5 трлн токенов | 2023 | Да (доступна только версия с 600 млрд токенов) | Falcon |
| SlimPajama | Очищенная версия RedPajama | 627 млрд токенов | 2023 | Да | Н/д |

Как видно из таблицы, большинство моделей обучаются на схожих наборах данных. В этой главе мы подробно рассмотрим наборы данных, которые применяются для предварительного обучения базовых моделей. Наборы данных, используемые для дальнейшего обучения, включая дообучение следованию инструкциям и обучение с подкреплением на основе обратной связи от человека (RLHF), будут подробно рассмотрены в *главе 6*.

### Правовые аспекты авторского права применительно к наборам предобучающих данных

Можно ли обучать LLM на текстах, защищенных авторским правом, без согласия правообладателей и указания их авторства? Каковы последствия обучения LLM на текстах, случайно содержащих конфиденциальную личную информацию? Эти вопросы вызывают серьезные юридические и этические споры. Обсуждаются возможности применения принципа добросовестного использования для легализации обучения LLM на защищенных авторским правом текстах. Этот вопрос остается открытым, и к моменту выхода книги коллективный иск был подан против GitHub, Microsoft и OpenAI (**https://oreil.ly/QcIKy**). В иске утверждается, что упомянутые компании использовали код из проектов GitHub с ограниченными лицензиями для обучения языковой модели, лежащей в основе GitHub Copilot. Сообщество ИИ внимательно следит за развитием этого судебного разбирательства. Тем не менее законодательство в этой области активно развивается по всему миру (**https://oreil.ly/6sgh_**), и все чаще поднимается вопрос о легализации такого использования с четкими правовыми ограничениями для обучения и настройки LLM.

С ростом популярности LLM и их интеграцией в экономику, данные, используемые для их обучения, становятся все более ценными. Например, Reddit и Stack Overflow, которые ранее были ключевыми поставщиками данных для многих популярных обучающих наборов данных, объявили (**https://oreil.ly/JdICc**) о введении платы за доступ к своим данным. Можно ожидать, что в ближайшее время подобные заявления станут более распространенными.

Как авторские права могут повлиять на частных и корпоративных пользователей, которые в будущем станут работать с этими языковыми моделями? *В главе 5* мы подробно обсудим этот вопрос, рассмотрев различные типы лицензий на программное обеспечение и их соответствие требованиям коммерческого использования.

Рассмотрим содержание наборов предобучающих данных. Для этого, используя блокнот Google Colab или любой другой текстовый редактор, загрузим подмножество `realnewslike` из набора данных C4, размер которого составляет около 15 Гбайт:

```
!pip install datasets


from datasets import dataset
realnewslike = load_dataset("allenai/c4", "realnewslike",
                            streaming=True, split="train")
for i, example in enumerate(realnewslike):
    if "Iceland" in example["text"]:
        print(example)
    if i == 10000: # Для демонстрации установлено ограничение в 10 000 итераций
        break
```

Этот код позволяет найти все экземпляры слова Iceland (Исландия) в данном подмножестве.

**Упражнение**

На основе подмножества `realnewslike` из набора данных C4 создайте счетчик частоты слов, который будет подсчитывать количество появлений каждого слова в этом наборе. Для простоты будем считать словом любую непрерывную последовательность символов, разделенных пробелами. Из анализа исключим частотные функциональные слова (стоп-слова), которые не несут ключевой смысловой информации, такие как "the", "is" и подобные. Какие области или аспекты, на ваш взгляд, представлены в данных неполно или, наоборот, избыточно?

## Синтетические предобучающие данные

В последнее время наблюдается тенденция применения LLM для генерации синтетических данных, которые могут служить основой для предобучения моделей. Одним из первых ярких примеров успешного обучения LLM на наборах данных, состоящих в значительной части из синтетических данных, стала серия моделей phi от Microsoft (**https://oreil.ly/eFphR**). Microsoft создала 20 млрд синтетических токенов для модели phi-1.5, используя в своих промптах 20 тыс. начальных тем и реальные выборки из общедоступных веб-данных.

Hugging Face представила Cosmopedia (**https://oreil.ly/Pdwnw**) — открытый набор синтетических данных, предназначенный для обучения моделей семейства SmolLM. В его основу легли тщательно отобранные материалы (курируемые источники, curated sources), среди которых курсы Стэнфордского университета, материалы Академии Хана (Khan Academy), контент WikiHow и другие общедоступные веб-данные.

Синтетические данные для курируемых источников формировались путем извлечения конспектов лекций из Академии Хана и других подобных платформ. Затем с помощью большой языковой модели Mistral из этих конспектов создавались развернутые учебные пособия по каждой теме. Чтобы разнообразить данные, на платформе Hugging Face для каждой темы предлагались различные варианты одного и того же промпта, например, "создать учебное пособие по этой теме для детей" и "создать учебное пособие по этой теме для профессионалов".

Hugging Face выделила более ста тематических подмножеств из набора данных RefinedWeb для общего веб-контента. Затем на основе *сниппетов* (или фрагментов — snippet) веб-страниц, были созданы промпты и переданы в LLM для генерации развернутых постов в блоге, соответствующих выделенной теме веб-страницы. Визуализацию этих тематических кластеров можно посмотреть в Nomic Atlas (**https://oreil.ly/t8R-6**).

**Упражнение**

Загрузите подмножество Cosmopedia-100K набора данных Cosmopedia (**https://oreil.ly/AVyg-**). Изучите промпты и сгенерированные синтетические данные, оцените качество этих данных. Не обнаруживаются ли в них ошибки фактического или логического характера?

Исследуйте возможности различных промптов и оцените, насколько можно добиться разнообразия получаемых данных.

## Предобработка обучающих данных

Перед началом обучения модели собранные или приобретенные данные необходимо отфильтровать и очистить, пропустив их через конвейер предварительной обработки или предобработки (preprocessing). Несмотря на то что предобработка данных часто остается незамеченной и недооцененной частью обучающего процесса, она является, пожалуй, ключевым звеном. По личному опыту, дополнительные усилия и ресурсы, вложенные на этом этапе, могут существенно повысить эффективность работы модели на последующих этапах.

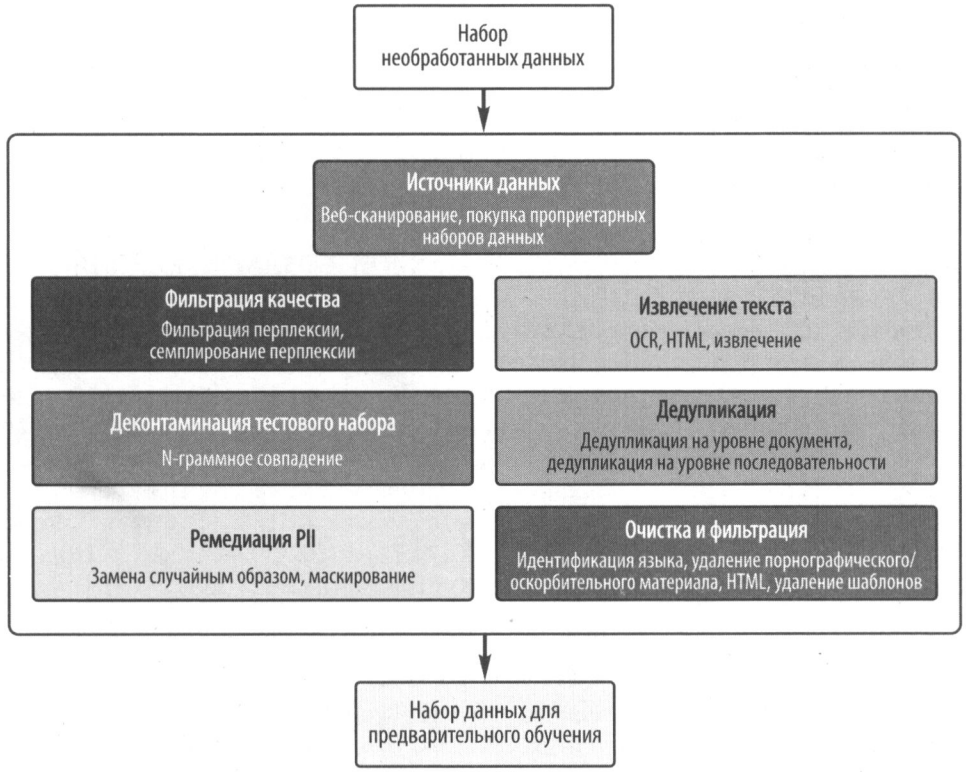



**Рис. 2.3.** Конвейер сбора и предобработки данных

Пройдя по конвейеру обработки данных, читатель сам убедится в сложности языкового текста и трудностях его обработки. Важно подчеркнуть, что из-за колоссальных объемов данных, каждый шаг предварительной обработки должен быть максимально оптимизирован, желательно с линейной временно́й сложностью.

На рис. 2.3 показаны типичные этапы предварительной обработки, используемые для создания предобучающего набора данных.

Хотя конкретный порядок шагов не задан, между некоторыми из них прослеживается определенная взаимозависимость.

Далее рассмотрим каждый из этих шагов более подробно.

## Фильтрация и очистка данных

Больша́я часть текста, извлеченного из HTML-документов, не несет смысловой нагрузки, как, например, тексты меню с веб-сайтов, шаблонные тексты и случайные артефакты веб-страниц. В сети также широко распространены порнографические материалы и оскорбительные/ненавистнические высказывания. В качестве иллюстрации приведем фрагмент текста из неочищенной версии набора данных C4:

> Skip to Main Content Skip to Footer Skip to Email Signup Skip to Feedback Form MY REWARDS SIGN OUT SIGN IN & EARN REWARDS 0 Keyboard Controls Welcome to the main navigation. This menu has three levels of product categories. Use and keys to navigate between each category in the current level. Use the key to navigate down a level. Use the key to navigate up a level. Hit the key to be taken to the selected category page. Men What's Hot New Arrivals Brand That Unites Performance Shop Online Exclusives Express Essentials Vacation Getaway Wedding Tuxedos Military Trend 9 Pieces / 33 Looks The Edit x Express NBA Collection Express + NBA Fashion NBA Game Changers Suiting & Blazers Find

---

> Перейти к основному содержанию Перейти вниз страницы Перейти к регистрации по email Перейти к форме обратной связи МОИ ВОЗНАГРАЖДЕНИЯ ВЫЙТИ ВОЙТИ И ПОЛУЧИТЬ ВОЗНАГРАЖДЕНИЯ 0 Управление клавиатурой Добро пожаловать в главное меню. Меню имеет три уровня категорий товаров. Для перехода между категориями на текущем уровне используйте клавиши. Для перехода на уровень вниз используйте клавишу. Для перехода на уровень вверх используйте клавишу. Нажмите клавишу для перехода на выбранную страницу категории. Мужчины Что нового Новые поступления Бренды Что объединяет Характеристики Магазин Онлайн Эксклюзивы Экспресс Предметы первой необходимости Поездка на отдых Свадебные смокинги Стиль милитари 9 штук / 33 Внешность Редактировать x Экспресс Коллекция НБА Экспресс + НБА Мода Переломные моменты в НБА Костюмы и Куртки Найти

Как вы считаете, насколько данный текст может быть полезен для обучения языку и выполнению заданий?

Данные Common Crawl предоставляются в двух форматах: необработанный HTML и WET. Хотя многие разработчики наборов данных предпочитают работать с файлами WET, исследователи из некоммерческой организации Eleuther AI выяснили (**https://oreil.ly/hciZS**), что качество этих файлов оставляет желать лучшего, т. к. в

них сохраняется шаблон HTML, как это демонстрирует приведенный выше пример. Для повышения эффективности удаления шаблонных текстовых фрагментов из HTML-документов при формировании набора данных The Pile, команда Eleuther AI использовала библиотеку jusText (**https://oreil.ly/YRFzZ**).

Рассмотрим эффективность применения jusText на конкретном примере. Для этого в блокноте Google Colab или Jupyter необходимо выполнить следующие действия:

```
!pip install justext

import requests
import justext

response =
    requests.get("https://en.wikipedia.org/wiki/Toronto_Transit_Commission")
text = justext.justext(response.content, justext.get_stoplist("English"))
for content in text:
    if content.is_boilerplate:
        print(content.text)
```

На выходе отображается отфильтрованный шаблон, извлеченный из стандартной статьи Википедии.

```
Jump to content  (Перейти к содержанию)
Main menu        (Главное меню)
Main menu        (Главное меню)
Navigation       (Навигация)
Main page        (Главная страница)
Contents         (Содержание)
Current events   (Последние события)
Random article   (Случайная статья)
About Wikipedia  (О Википедии)
Contact us       (Связаться с нами)
Donate           (Сделать донат)
Contribute       (Внести вклад)
Help             (Поддержка)
Learn to edit    (Научиться редактировать)
...
```

Зачастую jusText проявляет более агрессивную стратегию удаления контента, однако это обычно приемлемо при очистке предобученных наборов данных, учитывая обилие доступного текстового материала. В качестве альтернативных библиотек для этой задачи можно назвать Dragnet (**https://oreil.ly/URvsq**), html2text (**https://oreil.ly/xk7Hc**), inscriptis (**https://oreil.ly/6-2z1**), Newspaper (**https://oreil.ly/LPXe1**) и Trafilatura (**https://oreil.ly/zdZxj**). Как отмечают авторы The Pile (**https://oreil.ly/DZG7w**), разделение конвейера извлечения (extraction pipeline) на несколько библиотек предна-

значено для минимизации влияния возможных искажений, которые могут быть внесены одной из библиотек, на конечный набор данных.

### Предобучение на необработанных HTML-документах

Нужно ли удалять HTML-теги из исходных HTML-документов перед началом обучения? А если обучить модель работать с необработанными HTML-документами? Именно такой необычный, но интересный подход реализовала команда Армена Агаджаняна (Aghajanyan et al., **https://oreil.ly/VZxiC**) в своей гипертекстовой языковой модели (HTLM). Структура HTML-документов позволяет закодировать ценную информацию в виде текстовых метаданных. К примеру, теги `<title>` могут содержать краткое изложение, а теги `<class>` — сведения о классификации текста.

Не все элементы HTML пригодны для предобучения. Например, CSS недостаточно информативен для обучения языку. Поэтому разработчики HTML преобразуют сырой код в упрощенную версию, удаляя из него встроенные фреймы (iframes), верхние и нижние колонтитулы, формы и другие ненужные для изучения компоненты. Такой процесс называется *минификацией* (minification).

Как показало исследование, модель демонстрирует особую эффективность при *суммаризации* (summarization), поскольку доступ к категориальным тегам позволяет ей фокусироваться на ключевых моментах обсуждаемой темы. Однако на момент публикации книги эта методология предварительной подготовки не получила широкого распространения.

Удаление шаблонного контента с веб-страниц — это непростая задача, т. к. веб-страницы могут содержать не только текст, но и различные структурные элементы, такие как блоки кода, таблицы и математические формулы, которые требуют особого внимания при обработке. Компания Meta (**https://oreil.ly/bXELJ**) разработала собственный HTML-парсер для подготовки обучающих данных для своей модели Llama 3. При этом Meta решила сохранять атрибут `alt` в изображениях, обнаружив в нем ценную информацию, в частности, математического характера.

Большие языковые модели также могут быть использованы для точного извлечения контента с веб-страниц. Однако по состоянию на время написания книги этот процесс остается затратным из-за необходимости обработки огромных объемов данных.

### Упражнение

Зайдите на любой новостной сайт и откройте произвольную новостную статью. Используя одну из указанных выше библиотек извлечения текста, попробуйте очистить текст от веб-шаблона. Удалось ли вам добиться желаемого результата с первого раза? Какие дополнительные эвристики могли бы вам пригодиться в этом случае?

После извлечения текста документы проходят через несколько этапов фильтрации данных. На первом этапе применяются простые фильтры, основанные на эвристических правилах. Хотя конкретные шаги могут отличаться в зависимости от набора данных, рассмотрим некоторые из наиболее часто используемых операций.

*Удаление шаблонов (boilerplate removal)*

В результате обработки сохраняются только строки, заканчивающиеся точкой, восклицательным или вопросительным знаком. Такой подход позволяет эффективно удалять текстовые меню с веб-страниц. Для дальнейшей обработки строк используется пороговое значение, которое устанавливает минимальное количество слов и предложений в строке. Этот критерий важен для работы с длинными последовательностями, что в свою очередь расширяет возможности для создания более продвинутых языковых моделей. Документы, содержащие "lorem ipsum" (текст-заполнитель) и другие аналогичные шаблоны, удаляются.

*Удаление неанглоязычного текста*

Определение языка текста осуществляется с помощью таких библиотек, как langdetect, langid, fasttext и pycld2. Например, по оценке langdetect, текст C4 с высокой вероятностью (более 99%) идентифицируется как англоязычный. Стоит подчеркнуть, что эти библиотеки полезны не только для определения языка, но и для удаления артефактов, свойственных шаблонам и веб-страницам, поскольку они выявляют низкую вероятность их принадлежности к английскому языку.

*Удаление текста/спама в поисковой оптимизации (SEO)*

Удаляются документы, содержащие чрезмерное количество повторяющихся символьных последовательностей. Также удаляются документы с низким процентом слов закрытого класса. В английском языке к закрытому классу слов относятся служебные слова, например: предлоги "of" (указывает на принадлежность, происхождение или часть чего-либо), "at" (указывает на место, время или действие), определенный артикль "the", глагол "is" (форма глагола "to be" — "быть", используемая для третьего лица единственного числа в настоящем времени). При оптимизации ключевыми словами и применении других SEO-техник на веб-страницах отмечается снижение частоты использования таких служебных слов.

*Удаление текста с порнографической и/или оскорбительной направленностью*

Удаляются документы, в которых встречаются слова, указанные в списках ключевых слов вроде "List of Dirty, Naughty, Obscene or Otherwise Bad Words" ("Список грязных, непристойных и прочих грубых слов", **https://oreil.ly/w3u_r**), содержащем непристойные и оскорбительные выражения.

Такие библиотеки, как langdetect и langid, прекрасно справляются с определением языка текста в больших объемах. Но что происходит, когда речь идет о тексте с переключением кода, где английские фразы часто переплетаются с местными языками?

Попробуем проиллюстрировать это на примере тагальской пары (тагальский + английский языки), которая является популярным способом общения на Филиппинах.

Для этого выполним в блокноте следующие действия:

```
!pip install langdetect
```

```
from langdetect import detect_langs()

detect_langs("""Pag-uwi ko galing sa paaralan, sobrang pagod ako dahil sa dami ng
aking ginawa sa buong araw. Ang traffic din sa kalsada, nakaka-stress talaga! Pero
nang makarating ako sa aking tahanan, nabuhayan ako ng loob dahil sa masarap na amoy
ng ulam na inihanda ni nanay. Excited na akong kumain kasama ang aking pamilya at i-
share ang mga kwento ko tungkol sa aking mga kaibigan, guro, at mga natutunan ko sa
school. After dinner, magre-relax muna ako habang nanonood ng TV, and then magre-
review ng lessons bago matulog. Ito ang routine ko pag-uwi mula sa school, at masaya
ako na dumating sa bahay namay naghihintay na pamilya na handang makinig at suportahan
ako sa aking pag-aaral.""")
```

Вывод:

```
[tl:0.9999984631271781]
```

```
detect_langs("""After a long day at school, pagod na pagod talaga ako. The traffic on
the way home didn't help, nakakastress na nga! But upon arriving home, I felt a sense
of relief dahil sa welcoming atmosphere and the delicious aroma of the ulam na
inihanda ni Mommy. Excited na akong mag-share ng experiences ko today with my family
during dinner, kasama ang mga kwento about my friends, teachers, and interesting
lessons sa school. After eating, it's time for me to chill while watching some TV
shows, and then review my lessons bago ako matulog. This is my daily routine pag-uwi
galing school, and I am grateful na may loving family ako na handang makinig at
supportahan ako sa aking educational journey.""")
```

Вывод:

```
[en:0.9999954357601804]
```

Второй абзац мог бы попасть в набор данных C4, т. к. вероятность его принадлежности к английскому языку превышает 99%. Это означает, что даже наборы данных, позиционирующие себя как полностью англоязычные, нередко содержат фрагменты на других языках, что объясняет удивительную способность моделей генерировать многоязычный текст. Возникает вопрос: почему некоторые модели, предназначенные для работы с одним языком, демонстрируют при этом неплохие результаты в задачах машинного перевода? На это есть причина.

Алгоритм, реализованный в langdetect, приводит к неточным определениям языка при обработке коротких текстовых последовательностей. Например:

```
detect_langs('I love you too.')
```

возвращает

```
[sk:0.8571379760844766, en:0.14285726700161824],
```

где sk обозначает словацкий язык.

**Упражнение**

Англоязычный набор данных C4 был сформирован путем тщательной фильтрации текстовых фрагментов, исходно разрозненных по языкам. Фильтр исключал данные с вероятностью принадлежности к английскому языку ниже 99% (в соответствии с алгоритмом langdetect). Однако даже при такой строгой фильтрации внутри набора данных сохранилось значительное количество текстов на других языках. Для пользователей, владеющих несколькими языками помимо английского, предлагается воспользоваться подмножеством `realnewslike`, которое может быть использовано для поиска экземпляров с текстом на другом языке. В каких ситуациях и контекстах встречаются эти неанглоязычные текстовые фрагменты? Достаточно ли этих сохранившихся фрагментов для того, чтобы LLM смогла освоить эти языки?

## Отбор документов по критериям качества

Не все данные обладают одинаковой ценностью. Текст из учебника по физике, безусловно, будет считаться более достоверным и точным по сравнению с рекламным описанием обуви. Существует ряд подходов, позволяющих определить понятие качества данных и отличать высококачественные данные от низкокачественных. В данном разделе мы рассмотрим несколько таких подходов.

### Расхождение Кульбака — Лейблера в распределении токенов

Этот метод отсеивает документы, в которых распределение токенов сильно отклоняется от эталонного. Другими словами, удаляются документы с чрезмерным количеством некорректных токенов. Определение такого несоответствия осуществляется посредством дивергенции или расхождения Кульбака — Лейблера (K-L-дивергенция, Kullback–Leibler divergence, **https://oreil.ly/gd5GH**).

### Подходы на основе классификаторов

Для определения высококачественных данных можно создать классификатор (classifier). Один из простых способов его создания заключается в использовании примеров высокого качества из надежных источников, например Википедии, для положительного класса, и случайных документов из набора данных Common Crawl для отрицательного класса.

Для получения высококачественных данных для своей модели Llama 3 компания Meta применила несколько классификаторов, включая инструмент fasttext, который используется для выявления упоминаний текста в Википедии. Помимо этого, Meta создала собственный классификатор на основе данных, сгенерированных Llama 2, обучив его на тщательно отобранных веб-страницах с учетом установленных критериев качества. Также для извлечения кода и фрагментов, демонстрирующих этапы размышлений, были разработаны специализированные классификаторы, способные идентифицировать подобные элементы. На рис. 2.4 представлен процесс

создания классификатора, предназначенного для разделения данных на качественные и менее качественные категории.

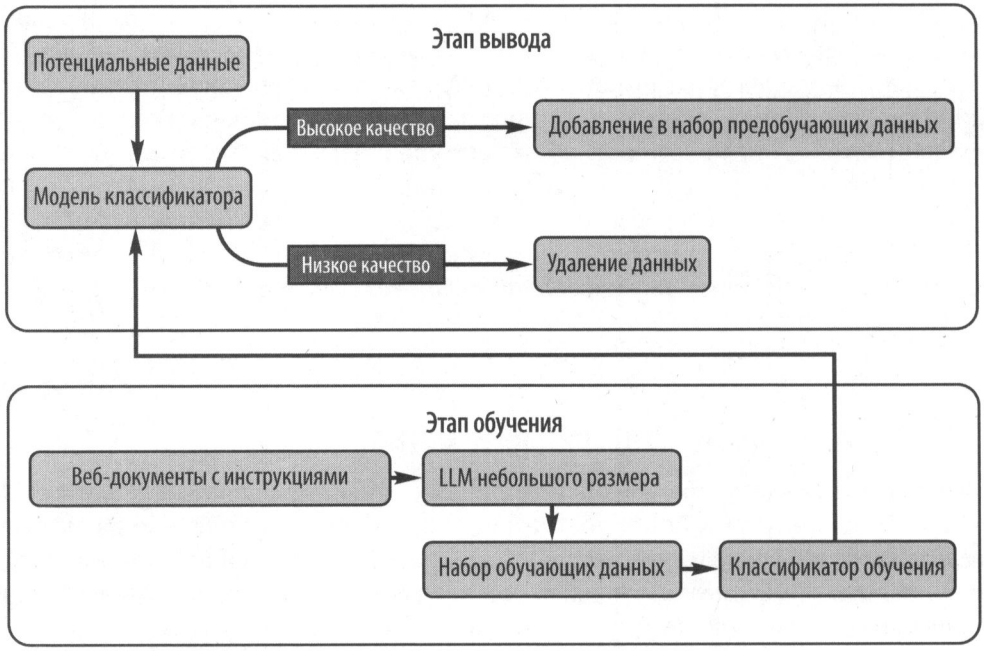



**Рис 2.4.** Фильтрация качества с использованием классификатора

### Упражнение

С помощью библиотеки fasttext создайте классификатор качества текстов. Для его обучения воспользуйтесь текстами из Википедии в качестве положительных примеров, а для отрицательных — случайными записями из необработанного набора C4 (**https://oreil.ly/EpCy7**). После этого протестируйте эффективность созданного классификатора на документах, например, из подмножества `realnewslike`. Оцените, насколько точным и полезным окажется такой классификатор для определения качества текстов.

## Оценка качества с использованием перплексии

Перплексия (коэффициент неопределенности — perplexity, **https://oreil.ly/OfycZ**) — это метрика, применяемая для анализа эффективности языковых моделей. Она активно используется (например, разработчиками CCNet, **https://oreil.ly/VF98y**) при отборе документов для формирования обучающих наборов данных. Этот показатель отражает способность модели прогнозировать следующий фрагмент текста — чем ниже значение перплексии, тем выше точность работы модели.

По аналогии с классификатором, для формирования положительного набора данных документы подбираются из проверенных источников, таких как Википедия.

Для обучения 5-граммной языковой модели применяется инструмент KenLM (**https://oreil.ly/EU5r3**), который значительно облегчает процесс создания моделей на основе *n*-грамм. После этого для набора данных, предназначенного для фильтрации, проводится расчет перплексии каждого абзаца относительно обученной модели. Абзацы с минимальной перплексией демонстрируют более высокое сходство с характеристиками положительного класса. Такой подход позволяет успешно отсеивать документы с высокой степенью перплексии.

Важно учитывать, что низкая перплексия не обязательно является показателем высокого качества текста. Достижение низкой перплексии может быть связано с краткостью или высокой степенью повторяемости. На перплексию также влияет стиль написания — если обучающая выборка языковой модели представляет собой материалы, подобные Википедии, тексты, выполненные в неформальной манере, могут восприниматься как более сложные. Это приведет к повышению значения перплексии для таких текстов. В подобных ситуациях рекомендуется применять более сложные подходы к фильтрации.

Для решения данной задачи авторы BERTIN (**https://oreil.ly/uI9eV**) ввели концепцию *семплирования перплексии* (perplexity sampling). Этот подход отличается от стандартной фильтрации, которая просто исключает текст с низким показателем перплексии. Семплирование перплексии использует более избирательную стратегию, удаляя тексты, принадлежащие центральной части распределения значений перплексии.

На рис. 2.5 показан процесс практического применения метода семплирования перплексии.

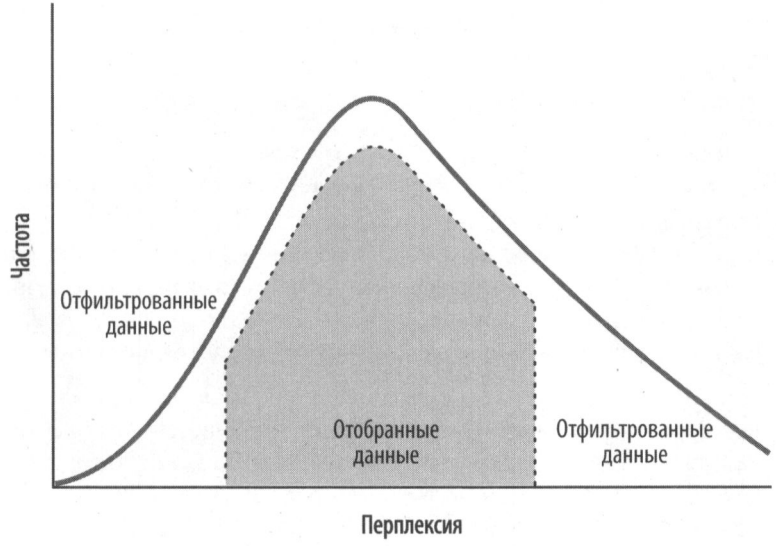



**Рис. 2.5.** Семплирование перплексии

Рассмотрим показатели перплексии, предоставленные моделью, обученной на текстовом корпусе из Википедии. Для начала загрузим файл по ссылке **https://oreil.ly/xwYjY**.

После сохранения файла в домашнем каталоге создадим новый файл и запустим следующий код:

```
from model import KenlmModel
model = KenlmModel.from_pretrained("wikipedia", "en")
model.get_perplexity("She was a shriveling bumblebee, and he was a bumbling banshee,
but they accepted a position at Gringotts because of their love for maple syrup")
// (Он выглядел как неуклюжий банши, а она напоминала сморщенного шершня,
// но их объединяла общая страсть к кленовому сиропу, которая привела обоих
// к работе в Гринготтс)
```

### Упражнение

Попробуйте текст в разных стилях и темах, чтобы оценить изменение перплексии. Например:

- пост для социальных сетей (например, X, ранее Twitter);
- SEO-спам;
- текст с большим количеством сленга.

Кроме того, можно обучить модель KenLM на собственных данных, используя часть подмножества `realnewslike` из C4. Для этого следуйте указаниям, представленным в документации на GitHub (**https://github.com/kpu/kenlm**). После обучения рассчитайте уровень перплексии каждого документа из неочищенной версии C4. Какие тексты демонстрируют самые высокие значения перплексии, а какие — самые низкие? После проведения анализа вручную оцените, насколько точным и достоверным является семплирование перплексии для отражения качества текстов?



Анализ данных из набора C4 (**https://oreil.ly/Nzla7**) показал, что значительную долю текстового контента в нем формирует домен **patents.google.com**. Примечательно, что более 10% текстов в данной категории были переведены с использованием машинного перевода, включая патенты из Японии, переведенные с японского на английский. Это свидетельствует о том, что значительная часть данных, используемых для предварительного обучения моделей, уже не создается людьми.

Постепенное распространение больших языковых моделей (LLM) в интернет-пространстве неизбежно приведет к увеличению объемов текста, созданного искусственным интеллектом. Определение автора текста — человека или LLM — становится все более сложной задачей и практически неосуществимо в масштабах больших данных. Влияние этого феномена на развитие LLM остается предметом активного изучения в научной среде.

Даже после тщательных этапов обработки набор данных на таком уровне масштаба редко достигает идеального качества. Например, Eleuther AI поделилась статисти-

кой (**https://oreil.ly/WEBne**), согласно которой стандартная фраза "выберите форум, который вы хотите посетить из списка ниже" встречается в The Pile около 180 тыс. раз.

## Дедупликация

Выше мы обсуждали такие аспекты, как извлечение и очистка данных, определение языка, а также качественная фильтрация. Теперь перейдем к одному из самых неоднозначных этапов конвейера — дедупликации (или дедубликация, от лат. *deduplicatio* — устранение дубликатов).

В текстах из Интернета, которые мы анализируем, часто встречаются повторяющиеся фрагменты. Эти повторы составляют значительную часть обучающих данных, поэтому любое решение, которое мы примем относительно их обработки, напрямую повлияет на эффективность будущей модели.

Но как именно определить, что является дубликатом? Здесь можно выделить три основные категории.

*Точные совпадения (exact matches)*

Если две текстовые последовательности полностью совпадают по содержанию, они считаются точными дубликатами. Работа с такими дубликатами не представляет особой сложности.

*Примерные совпадения (approximate matches)*

Нередко можно встретить тексты, которые по смыслу очень похожи, но отличаются всего лишь несколькими символами. Такие различия часто возникают из-за проблем с извлечением HTML-кода или в результате других процессов фильтрации.

*Семантические дубликаты (semantic duplicates)*

Тексты, которые передают один и тот же смысл, но при этом выражены поразному. Как правило, стандартные методы анализа данных не учитывают такие ситуации.

Кроме того, дубликаты можно разделить по степени детализации, с которой они возникают.

*Дубликаты на уровне документа (document-level duplicates)*

В процессе подготовки большинства наборов данных перед этапом обучения, как правило, удаляются дубликаты документов. Однако в некоторых наборах данных, таких как The Pile, отдельные подмножества (например, Википедия), намеренно включаются в виде дубликатов с целью обеспечить модели возможность чаще встречаться с ними и лучше усваивать их содержание.

*Дубликаты на уровне последовательности (sequence-level duplicates)*

В эту категорию входят строки или предложения, которые встречаются в разных документах. Иногда такие последовательности могут быть повторены много раз, например: информация об условиях использования, уведомления об авторских правах, вводные тексты на веб-страницах и подобные элементы.



Дедупликация представляет собой сложный процесс, который обычно реализуется с использованием алгоритма MinHash. Подробное описание этого алгоритма можно найти в работе Чэнга Хао (Cheng Hao, **https://oreil.ly/2RO9f**), где также рассматриваются особенности его применения в проектах LLM с открытым исходным кодом, таких как Big Science и Big Code.

Дедупликация данных обладает рядом преимуществ.

- Для проверки и тестирования обычно отбирается небольшое подмножество набора данных, используемого в предварительном обучении. Дедупликация помогает устранить или минимизировать пересечения между обучающими и тестовыми наборами, что крайне важно для получения объективных результатов оценки. Без выполнения дедупликации на уровне последовательностей (sequence-level deduplication) существует высокая вероятность совпадения текстовых фрагментов в этих наборах.
- Удаление повторяющихся последовательностей позволяет сократить общий объем обучающего набора данных. В то же время исследование Кэтрин Ли и соавт. (Lee et al., **https://oreil.ly/k5OwJ**) показало, что перплексия модели, обученной на сокращенном наборе данных, остается неизменной. Это свидетельствует о том, что модель может быть обучена за более короткое время, при этом достигая аналогичных результатов.
- Дедупликация данных позволяет значительно сократить склонность модели к запоминанию обучающих примеров, что благоприятно сказывается на ее способности к обобщению. Процесс запоминания тесно связан с явлением переобучения, которое ограничивает возможности модели эффективно адаптироваться к новым сценариям. Существует множество подходов для количественной оценки запоминания, однако в данном случае акцент сделан на феномене *запоминания по генерациям* (memorization by generation), при котором модель считается запомнившей последовательности, если она воспроизводит их дословно. Исследование Кэтрин Ли и соавт. (**https://oreil.ly/xpoz7**) продемонстрировало, что модели, обученные на наборах данных, дедуплицированных на уровне последовательностей, генерируют в 10 раз меньше дословных обучающих данных.



Одно из преимуществ применения моделей, обученных на общедоступных данных, заключается в возможности проверить эти наборы и убедиться, что модель содержит текст, дословно совпадающий с материалами из набора данных.

### Угрозы безопасности LLM, связанные с запоминанием

Запоминание делает языковые модели потенциально уязвимыми для угроз, связанных с безопасностью и конфиденциальностью. Можно выделить два основных типа атак.

*Атака на процедуру выхода из организации (membership inference attack)*

В этом случае используется метод вывода членства, который позволяет злоумышленникам, применяя модель с закрытым доступом (closed-box assess), определить, была ли конкретная текстовая последовательность частью обучающего набора данных.

*Атака на сбор обучающих данных (training data extraction attack)*

Даже если доступ к модели ограничен, злоумышленники могут попытаться извлечь из нее запомненную конфиденциальную информацию. Например, можно запросить у модели продолжить текст вроде "Номер телефона Сухаса Пая" в расчете на то, что она сохранила этот номер при обучении.

Исследования Карлини и соавт. (Carlini et al., **https://oreil.ly/iIic3**) демонстрируют, что более крупные языковые модели склонны легче запоминать информацию, и это делает их особенно уязвимыми к данному типу атак. При этом оценка количества запомненных данных оказывается сложной задачей, поскольку модель способна воспроизводить запомненные фрагменты только в ответ на специфические запросы с тщательно составленным длинным префиксом. Такая особенность существенно осложняет проверку моделей на соответствие требованиям конфиденциальности.

Процесс базовой атаки, направленной на извлечение обучающих данных, показан на рис. 2.6.

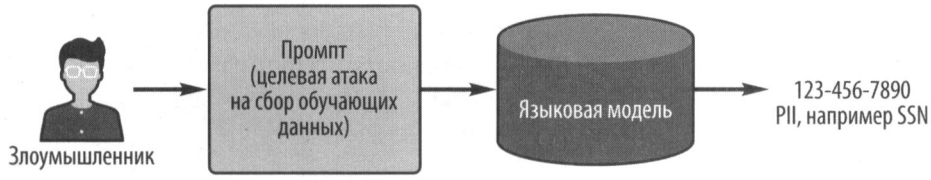



**Рис. 2.6.** Атаки на конфиденциальные данные LLM

## Удаление персонально идентифицируемой информации

Хотя дедупликация данных может уменьшить вероятность их запоминания моделью, она не является абсолютным решением проблемы. Даже данные, встречающиеся в обучающем наборе только один раз, могут быть сохранены и впоследствии раскрыты. Большая часть тренировочного контента является безобидной, например стандартные тексты условий обслуживания, или даже полезной для запоминания, как фактические сведения, такие как столица Канады. Однако ситуация кардинально меняется, когда речь заходит о персонально идентифицируемой информации (personally identifiable information, PII), что представляет собой серьезную проблему.

Согласно закону Корнелла (Cornell Law, **https://oreil.ly/kN3J8**), термин PII означает:

> "Информация, которая может быть использована для идентификации или отслеживания конкретного человека, как отдельно, так и в комбинации с другими сведениями, связанными с этим человеком или способными быть с ним ассоциированными".

Согласно этому определению, информация, которая не считается персональными данными (PII), может стать таковой, если другая информация, ранее не доступная для широкой публики, становится таковой. В таком случае объединение этих двух типов информации может позволить однозначно идентифицировать человека.

В разных правовых системах понятие PII имеет различное юридическое значение. Например, в Европейском союзе действует "Общий регламент о защите данных" (General Data Protection Regulation, GDPR, **https://oreil.ly/F2dGL**), который определяет:

> "Под защиту попадают все данные, которые могут быть использованы для прямого или косвенного установления личности человека (или субъекта данных). В целях расширения защиты могут быть включены характеристики, определяющие „физическую, физиологическую, генетическую, интеллектуальную, экономическую, культурную или социальную принадлежность человека“".

Многие модели с открытым исходным кодом обучаются на общедоступных данных. В этих данных могут содержаться персональные сведения, и иногда кажется, что, поскольку они уже открыты для всех, защита конфиденциальности не является необходимой. Однако такой подход не учитывает важность контроля согласия и возможности обнаружения. Например, кто-то может опубликовать свои личные данные в блоге, который находится в труднодоступном месте в Интернете, и его сложно найти с помощью поисковых систем. Но если эти данные попадут в обучающий набор, они могут оказаться в центре внимания без согласия этого человека. Такая ситуация наглядно демонстрирует важность принципа *контекстной целостности* (contextual integrity) — данные должны распространяться только в том контексте, в котором они были изначально опубликованы.

Идеальным вариантом было бы полностью обнаружить и удалить личную информацию из данных, чтобы она не попадала в обучающие наборы и не хранилась в памяти модели. Однако открытость PII, связанных с публичными фигурами, усложняет эту задачу. Желательно, чтобы модель могла не только достоверно отвечать на вопросы о реальных людях, но и предоставлять точную информацию о них, например, указывая точную дату рождения. Противоречие между принципами прозрачности и открытости, с одной стороны, и необходимостью сохранения конфиденциальности — с другой, наглядно демонстрирует снижение требований к защите личной информации у общественных деятелей. Определение границ "публичной фигуры" и ее допустимого уровня приватности представляет собой сложную задачу, требующую комплексного подхода, который учитывает как социальные, так и технические аспекты.

К категории персональных данных относятся фамилия, имя, отчество, адрес проживания, информация о кредитных картах и документах, удостоверяющих личность, история медицинских наблюдений и результаты диагностических процедур,

адреса электронной почты, телефонные номера, идентификационные данные, а также сведения о групповой принадлежности (религиозной, этнической, профсоюзной), данные о геолокации и другая подобная информация.

Атаки можно разделить на целевые (targeted) и нецелевые (untargeted). В рамках нецелевой атаки злоумышленник использует модель для генерации большого количества текста, а затем применяет атаку на процедуру выхода из организации, чтобы выявить наиболее запоминающийся фрагмент. Целевая атака, в свою очередь, направлена на восстановление личной информации о конкретном человеке или группе людей. Целевые атаки представляют собой более сложную задачу, т. к. несмотря на впечатляющую способность языковых моделей, им все еще трудно устанавливать ассоциации. Например, определить, какому конкретному человеку принадлежит тот или иной идентификатор электронной почты, для них по-прежнему остается непростой задачей.

### Упражнение

Следуя инструкциям в ReadMe, запустите код по ссылке **https://oreil.ly/sHAKY**, где подробно описан анализ атак на конфиденциальность LLM. Пожалуйста, не применяйте его в реальных условиях! Запуск кода и анализ выводимой информации даст представление о границах подобных атак и типах данных, которые обычно запоминаются LLM.

Также стоит попробовать программу Google Training Data Extraction Challenge (**https://oreil.ly/yqZ1C**).

В большинстве наборов данных, используемых для предварительного обучения, PII остались практически без изменений. Рабочая группа по вопросам приватности проекта Big Science (в которой я занимал должность соруководителя), ответственного за обучение модели BLOOM, разработала специальный конвейер для выявления и исправления (ремедиации — remediation) PII, который будет подробно описан ниже.



Языковые модели также подвержены атакам *отравления обучающих данных* (data poisoning). Так как основная часть обучающих данных поступает из веб-текста, злоумышленники могут манипулировать содержанием обучающей выборки. Исследование Ф. Трамера и соавт. (Tramer et al., **https://oreil.ly/g_A-d**) показало, что даже незначительное отравление менее 0,1% обучающего набора может спровоцировать утечку конфиденциальной информации из него.

С ростом популярности больших языковых моделей в качестве поисковых систем, оптимизация SEO для них приобретает все большую важность. В этом контексте компании все активнее создают веб-контент, нацеленный на попадание в обучающие базы данных, которые генерируются с помощью техники, наподобие фильтрации перплексии.

Типичный конвейер обработки PII показан на рис. 2.7.

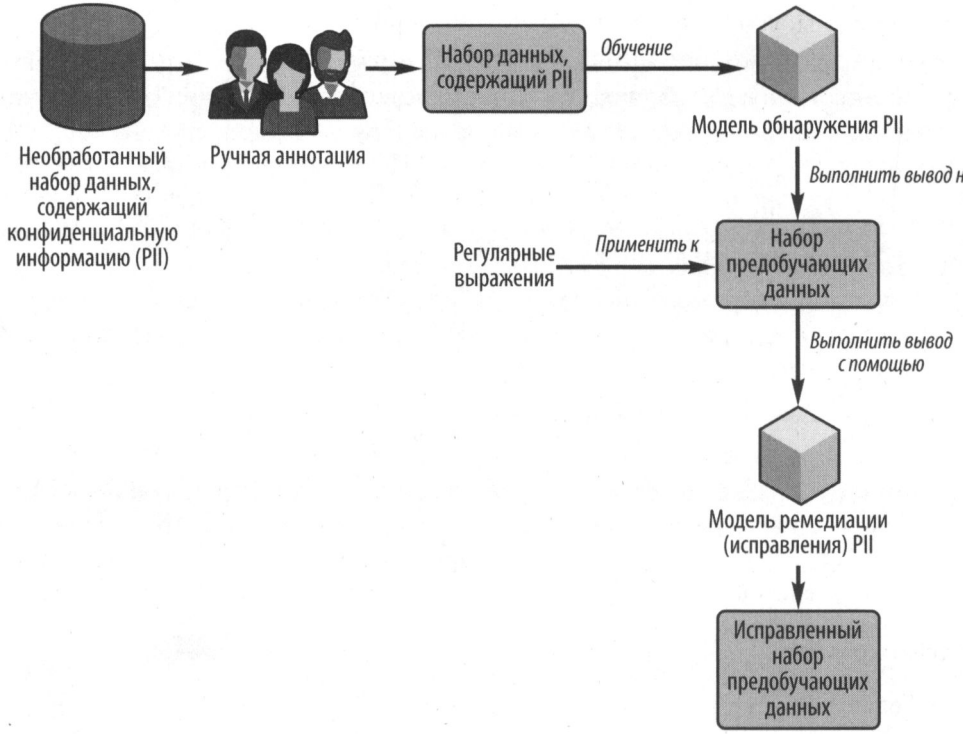



**Рис. 2.7.** Конвейер обработки PII

## Обнаружение PII

Обнаружение персонально идентифицируемой информации (PII) по своей сути напоминает задачу распознавания именованных сущностей (NER) в области обработки естественного языка (NLP), о которой говорилось в *главе 1*. Однако важно отметить, что не все выделяемые сущности относятся к категории PII. В рамках нашей задачи были определены следующие теги для идентификации PII: PERSON, AGE, NORP (национальность, раса, религиозная принадлежность, членство в политических партиях, социально-экономический статус и участие в профсоюзах), STREET_ADDRESS, CREDIT_CARD, GOVT_ID, EMAIL_ADDRESS, USER_ID и PUBLIC_FIGURE.

Мы использовали тег PUBLIC_FIGURE для классификации информации о публичных фигурах, поскольку не хотели исключать их из выборки. Этот же тег присваивался и вымышленным персонажам.

Некоторые структурированные теги из этого списка, такие как адреса электронной почты (EMAIL_ADDRESS) или государственные идентификаторы (GOVT_ID), можно выявить с помощью регулярных выражений. Для других тегов мы вручную

аннотировали наборы данных, которые позже можно было применять для обучения моделей в стиле NER на базе трансформеров. Примечательно, что аннотация сопровождалась высокой степенью несогласованности между разными аннотаторами — один и тот же пример нередко получал разные метки. Это подчеркивает культурные различия в восприятии конфиденциальности и определении личной информации.

Например, регулярное выражение для обнаружения SSN (номера социального страхования США, **https://oreil.ly/8YwG9**) выглядит следующим образом:

```
ssn_pattern = r"(?!000|666|333)0*(?:[0-6][0-9][0-9]|[0-7][0-6][0-9]|
[0-7][0-7][0-2])[-\ ](?!00)[0-9]{2}[-\ ](?!0000)[0-9]{4}"
```

Важно понимать, что обнаружение не равносильно проверке. Не каждый девятизначный номер формата XXX-XX-XXXX является SSN. Проверка подразумевает процесс подтверждения соответствия последовательности символов установленным требованиям для идентификатора. Например, канадский аналог SSN, называемый номером социального страхования (SIN), включает в себя контрольную цифру, которая служит для его проверки:

```
from stdnum.ca import sin
sin_pattern = re.compile(r"\d{3}[-\ ]\d{3}[-\ ]\d{3}", flags=re.X)
for match in sin_pattern.findall(text):
    if sin.is_valid(match):
        print(match)
```

Функция `is_valid()` реализует алгоритм Лу́на для расчета контрольной цифры (Luhn algorithm, **https://oreil.ly/i34BW**), позволяя проверить соответствие набора чисел валидному SIN. Этот же алгоритм применяется для валидации номеров кредитных карт. Для определения номеров кредитных карт используется следующее регулярное выражение (**https://oreil.ly/6uTq-**):

```
from stdnum import luhn
cc_base_pattern = r"\b \d (?:\d[ -]?){14} \d \b"
cc_full_pattern = r"""4[0-9]{12}(?:[0-9]{3})? |
            (?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|
            2720)[0-9]{12} |
            3[47][0-9]{13} |
            3(?:0[0-5]|[68][0-9])[0-9]{11} |
            6(?:011|5[0-9]{2})[0-9]{12} |
            (?:2131|1800|35\d{3})\d{11}"""
```

Регулярное выражение для обнаружения адреса электронной почты выглядит следующим образом:

```
email_pattern = r"[\w\.=-]+ @ [\w\.-]+ \. [\w]{2,3}"
```

**Упражнение**

Для выявления наличия PII в предобучающем наборе данных RefinedWeb (**https://oreil.ly/jto4m**) воспользуйтесь функцией поиска в просмотрщике наборов данных (dataset viewer). Например, введите запрос "gmail.com" и просмотрите результаты. Какие данные удалось обнаружить?



Удаление структурированных данных PII при одновременном сохранении низкого уровня ложных срабатываний — задача непростая. Однако обнаружение и исправление неструктурированных данных оказываются еще более сложным делом. Учитывая трудности этой задачи и неопределенность относительно ее воздействия на качество и производительность итоговой модели, было принято решение не использовать конвейер обработки PII на основе архитектуры трансформера для набора данных ROOTS при обучении модели BLOOM.

## Ремедиация PII

После того как персональные данные были выявлены, можно приступить к их ремедиации. На рис. 2.8 представлена одна из возможных схем процесса ремедиации PII.




**Рис. 2.8.** Схема ремедиации PII

Рассмотрим несколько популярных методов ремедиации данных.

*Замена на специальный токен (special token)*

Например, реальный номер телефона можно заменить строкой <phone number> (номер телефона).

*Замена на случайный токен того же типа сущности*

Например, имя "Clarietta Richards" можно заменить на "Natasha Bridges" или другое подходящее имя.

*Замена на перемешанный токен (shuffled token)*

Объекты, обнаруженные в наборе данных, можно перемешать.

*Удаление всего документа или источника данных*

Если в одном документе или источнике данных выявлено количество персональных данных, превышающее установленный порог, может быть принято решение о полном удалении этого объекта. Например, существует мнение, что на сайте **pastebin.com** часто встречается большое количество случайно опубликованных персональных данных, поэтому его использование в обучающих наборах данных не рекомендуется.

Каждый из перечисленных методов может оказывать различное влияние на характеристики модели на последующих этапах разработки. Например, как замена токенов влияет на перплексию в процессе обучения? Не оказывает ли это негативное воздействие на задачи, такие как распознавание именованных сущностей (NER), при настройке итоговой модели? Какие взаимосвязи можно наблюдать между заменой специальными токенами и использованием *случайных токенов* (random token)? Эта область еще недостаточно изучена, и многие из этих вопросов остаются открытыми для обсуждения и исследований.

Применение случайных токенов для замены данных стало проще с библиотекой Faker (**https://oreil.ly/K4QI_**). Этот инструмент позволяет легко генерировать случайные значения для различных видов персональной информации, включая имена, адреса, номера кредитных карт и телефонов. Однако при замене данных на случайные токены существует риск искажения демографического распределения, что может сказаться на качестве итоговых результатов. Это может произойти, например, если вместо оригинальных имен используются преимущественно англосаксонские. Чтобы минимизировать этот эффект, Faker поддерживает локализацию. Это позволяет создавать поддельные (фейковые) данные, соответствующие конкретной географической и культурной среде. Рассмотрим подробнее эту библиотеку:

```
from faker import Faker
fake = Faker('en_IN') # Индийская локаль
Faker.seed(0)
for i in range(5):
    print(fake.aadhaar_id)
```

Этот код генерирует фейковые идентификаторы Aadhaar, состоящие из 12 цифр, которые можно рассматривать как индийский аналог номеров социального страхования. Хотя эти идентификаторы являются недействительными, они соответствуют официальному формату. Аналогичным образом, следующий код

```
for i in range(5):
    print(fake.address)
```

позволяет создавать поддельные адреса, соответствующие выбранному языковому стандарту.



Удаление персонально идентифицируемой информации из обучающих наборов данных представляет собой лишь одно из множества подходов, направленных на предотвращение утечки данных из моделей машинного обучения. Одним из перспективных методов является *дифференциальная приватность* (differential privacy, **https://oreil.ly/TRbsf**), которая вводит элемент случайности во входные или выходные данные, предоставляя теоретические гарантии защиты конфиденциальности. В нейронных сетях этот подход реализуется с помощью алгоритма DP-SGD (**https://oreil.ly/DVQkl**), включающего градиентное отсечение и добавление шума на этапе обновления параметров модели. Однако использование дифференциальной приватности приводит к замедлению процесса обучения, снижению производи-

тельности модели и непропорциональным последствиям для представленных в наборе данных групп меньшинств, выражающимся в уменьшении полезности модели. Помимо этого метода, существуют и другие варианты обеспечения безопасности, такие как *состязательное обучение* (adversarial training), *разобучение модели* (model unlearning, **https://oreil.ly/_AV3V**), *ретроактивное цензурирование* (retroactive censoring) и *декодирование "memfree"* (memfree decoding, **https://oreil.ly/5p0z3**).

## Деконтаминация обучающего набора

Деконтаминация (decontamination) представляет собой процесс очистки обучающего набора от нежелательных элементов, что является важной частью предварительной подготовки данных для повышения точности оценки больших языковых моделей. Набор данных считается загрязненным, если в нем присутствуют данные из эталонных тестовых наборов, используемых для оценки производительности модели. Это загрязнение, или контаминация (contamination), может происходить по разным причинам, например, когда тестовые наборы формируются на основе веб-контента или когда использованный набор данных становится доступным в Интернете.

Выделяют два основных типа контаминации[3].

*Контаминация входных данных и меток (input and label contamination)*

В данной настройке как вопросы (входные данные), так и ответы (целевые метки) присутствуют в предобучающем наборе данных.

*Контаминация входных данных (input contamination)*

Эта настройка предполагает наличие только входных данных в предобучающем наборе, без целевых меток. *В главе 7* будет рассмотрено влияние такого загрязнения, а также способы его использования в положительном ключе.

В процессе анализа обучающего набора GPT-3 компания OpenAI (**https://oreil.ly/d7pHK**) выявила загрязнение данных, выраженное в виде 13-грамных совпадений между текстами тестового и обучающего наборов. Для решения этой проблемы было принято решение удалить по 200 символов, расположенных перед и после совпадающих текстовых фрагментов. Наиболее распространенным методом деконтаминации данных является использование метода *сопоставления n-грамм* (n-gram matching).

Следует отметить, что, как отмечают Шуо Ян и соавт. (Yang et al., **https://oreil.ly/JjtHS**), контаминация обучающих данных может происходить не только из-за прямых совпадений, но и из-за перефразированных или переведенных версий контрольных текстов. Это делает процесс выявления и устранения загрязнений крайне сложной задачей. В результате большинство показателей на тестах бенчмарков

[3] Додж и др. Исследование громадного корпуса чистых веб-данных (Dodge et al. A Case Study on the Colossal Clean Crawled Corpus; **https://oreil.ly/PwtVp**). Материалы конференции "Эмпирические методы обработки естественного языка" (EMNLP), 2021.

продолжают показывать завышенные значения, что подчеркивает трудности, с которыми сталкиваются при создании действительно корректных моделей.

## Смеси данных

Для предварительного обучения используются данные из различных предметных областей (доменов). Конечный набор данных формируется таким образом, чтобы в нем было представлено оптимальное соотношение этих областей. Например, в наборе данных The Pile объемы выборок из Википедии, академических текстов и более узких тематических подмножеств были увеличены в 3 раза (**https://oreil.ly/hpHdw**). Для получения наиболее точной *смеси данных* (data mixture) также применяются более продвинутые методы, такие как DoReMi (**https://oreil.ly/5z9u1**) и RegMix (**https://oreil.ly/VWyzt**). Компания Мета поделилась результатами своих эмпирических исследований, показавших, что для Llama 3 (**https://oreil.ly/fMOrb**) оптимальной является смесь данных, состоящая из 50% токенов, связанных с общими знаниями, 25% — математических и логических рассуждений, 17% — кода и оставшихся процентов — неанглоязычных токенов.



В современных наборах предобучающих данных часто присутствует код, даже если модель не ориентирована на его генерацию. Исследования Арьябуми и соавт. (Aryabumi et al., **https://oreil.ly/Vm0lH**) выявили, что добавление кода в предобучающие данные существенно повышает эффективность моделей при выполнении задач, не связанных с генерацией кода.

### Упорядочение наборов данных

После того как все этапы предварительной обработки данных завершены, начинается процесс обучения модели. Ключевым моментом в этом процессе является порядок подачи данных в модель, который значительно влияет на конечный результат. Область исследования, занимающаяся поиском оптимального порядка подачи данных, называется *обучением по учебной программе* (curriculum learning). В настоящее время большинство моделей используют простые эвристические методы для упорядочения данных (data ordering).

Одним из подходов к обучению является постепенное увеличение длины обучающих последовательностей. Это можно реализовать либо путем усечения исходных последовательностей до нужной длины, либо путем упорядочивания набора данных таким образом, чтобы сначала обрабатывались более короткие последовательности.

Авторы исследования (**https://oreil.ly/QY1MI**) также предложили сначала подавать моделям более часто встречающиеся слова, заменяя редкие слова из обучающих примеров на теги части речи (POS tag) или гиперонимами, т. е. более общими терминами (например, гипероним пурпурного цвета — это цвет).

После того как мы рассмотрели ключевые этапы сбора и предварительной обработки данных для создания обучающего набора, давайте обсудим, чем отличаются различные наборы данных в зависимости от применяемых к ним методов предобработки.



Hugging Face разработала масштабный репозиторий кода DataTrove (**https://oreil.ly/IDFm2**), предназначенный для обработки предобученных наборов данных в режиме конвейера. Изучение его содержимого позволит вам на практике понять, как принципы, рассмотренные в этой главе, могут быть применены в больших масштабах.

В табл. 2.2 представлен перечень популярных наборов данных, используемых для предобучения, а также методы их предобработки.

**Таблица 2.2.** Предобучающие наборы данных и их конвейер предварительной обработки

| **Название** | **Извлечение и очистка** | **Фильтрация качества** | **Дедупликация** | **Определение языка** | **Модели, обученные на данном наборе данных** |
|---|---|---|---|---|---|
| C4 | Удаление страниц, содержащих слова из черного списка (блок-листа), удаление кода, удаление коротких строк и страниц | – | Дедупликация спанов из 3 предложений | langdetect | T5, FLAN-T5, UL2, Llama |
| The Pile | Извлечение текста с помощью библиотеки justext | Классификатор justext | Дедупликация на уровне документа с помощью MinHashLSH | pycld2 | GPT-NeoX, GPT-J, Cerebras-GPT, StableLM, Pythia |
| CCNet | – | Фильтрация перплексии | Дедупликация на уровне абзацев | fasttext | |
| RedPajama | Конвейер CCNet | Классификатор, отличающий текст из Википедии от случайного текста C4 | Дедупликация на уровне абзацев (для Common Crawl) | fasttext | Red Pajama-INCITE, MPT |

**Таблица 2.2** (окончание)

| Название | Извлечение и очистка | Фильтрация качества | Дедупликация | Определение языка | Модели, обученные на данном наборе данных |
|---|---|---|---|---|---|
| CleanPajama | Фильтр малой длины (low-length filter), нормализация NFC | – | MinHashLSH | – | – |
| RefinedWeb | Фильтрация URL по блок-листам, библиотека trafilatura для извлечения текста, удаление повторяющегося контента | – | Нечеткая дедупликация на уровне документов с использованием MinHash, точная дедупликация на уровне последовательностей | fasttext | Falcon |
| ROOTS | Удаление документов с низким соотношением служебных слов, высоким соотношением слов из блок-листа, высоким соотношением повторений символов/слов | Фильтрация перплексии | SimHash, Suffix Array | fasttext | BLOOM |

## Влияние предобучающих данных на последующие задачи

Анализируя предобучающий набор данных для LLM, можно сделать предположения о будущем производительности модели. Оказывается, что эффективность модели при выполнении конкретной задачи или при обработке определенного типа входных данных коррелирует с частотой встречаемости соответствующих данных во время обучения или частотой появления характерных слов во входных данных. Это явление впервые было замечено командой Разеги (Razeghi et al., **https://oreil.ly/cPYej**), а затем более подробно исследовано в работе МакКоя и соавт. "Элементы авторегрессии" (McCoy et al. Embers of Autoregression; **https://oreil.ly/_O2NK**).

В исследовании МакКоя было установлено, что языковые модели достигают наилучших результатов при решении задач, которые чаще всего встречаются в обучающем наборе данных, в отличие от задач, встречающихся реже. Например, модели демонстрируют более высокую точность при выполнении арифметических операций с основанием 10 по сравнению с основанием 9, а также лучше сортируют слова по алфавиту, чем в обратном порядке.

Исследование показало, что модели демонстрируют наилучшие показатели при решении конкретных задач, когда вывод представляет собой текст с высокой частотой встречаемости в предобучающем наборе, в отличие от текстов, встречающихся реже. Подобная закономерность прослеживается и при работе с входными данными: модели лучше справляются с высокочастотными данными по сравнению с редкими.

В качестве примера рассмотрим такое предложение: "record a be that miles, yes, hour, per fifty clocked he" ("рекорд, что являлось миль, да, час, в пятьдесят набирал он"). Если попросить языковую модель перефразировать его, например, как: "He clocked fifty per hour, yes, miles, that be a record" ("Он набирал пятьдесят в час, да, миль, что являлось рекордом"), то это будет маловероятной последовательностью из-за нетипичной грамматической структуры.

В момент публикации книги система GPT-4o выдавала неверный ответ: "He clocked fifty miles per hour that be a record" ("Он набирал пятьдесят миль в час, что являлось рекордом"), однако следует отметить, что ее работа более эффективна, если выходная последовательность имеет более высокую степень вероятности.

**Упражнение**

Известно, что Википедия служит основой обучения для большинства LLM. Попросите модель (без доступа к Интернету) ответить на вопрос, основанный на факте из малоизвестной статьи Википедии. Сможет ли она правильно определить ответ? Проверьте, изменится ли ответ, если задать LLM вопросы о фактах из более известных статей Википедии.

## Проблемы предвзятости и справедливости в предобучающих наборах данных

Разработка больших языковых моделей порождает множество этических вопросов. Одна из ключевых проблем заключается в наличии значительных проявлений предвзятости и несправедливости в этих моделях, что часто ограничивает их возможности применения в различных сферах. Мы подробно рассмотрим конкретные проблемы, связанные с предвзятостью и справедливостью, возникающие на этапах сбора и фильтрации предобучающих данных.

Обучаясь на огромных массивах данных, большие языковые модели не только имитируют язык, но и создают своего рода отражение мира, в котором мы живем. Это поднимает важный вопрос: должны ли эти модели мира, основанные на данных,

отображать его без изменений, или же мы стремимся к более благоприятной, идеализированной версии? Интернет, к сожалению, пестрит ненавистью, насилием и оскорбительной лексикой, часто служа площадкой для проявления самых негативных человеческих инстинктов. Текст в сети нередко несет в себе кодированную информацию о давно укоренившихся предрассудках и предубеждениях против различных групп людей. Например, в исследовании The Pile (**https://oreil.ly/hu3-b**), проанализированном по статистике совпадений слов, выявлено, что слово "радикальный" часто ассоциируется со словом "мусульманин" в гораздо большей степени, чем с представителями других религиозных вероисповеданий.

Явление *усиления предвзятости* (bias amplification) придает этим проблемам еще большую значимость. Исследования показывают (**https://oreil.ly/x-ba9**), что большие языковые модели усиливают предвзятости, заложенные в их предобучающих данных. В результате они часто выдают предвзятые прогнозы о различных группах людей с большей вероятностью, чем это оправдано статистикой обучающих данных.

Можно ли адаптировать обучающие данные так, чтобы модели учитывали наши ценности и принципы, передавая их последующим приложениям? Эта тема активно обсуждается в научном сообществе. Некоторые исследователи утверждают, что полностью устранить социальные предубеждения, заложенные в данных, крайне затруднительно, поскольку такие предубеждения часто имеют сложный и многослойный характер. Кроме того, ценности редко бывают универсальными, и разработчики моделей часто стремятся к нейтральности, чтобы создать инструменты, которые отвечают интересам разных групп населения.

Анна Роджерс в своей работе (Anna Rogers, **https://oreil.ly/hxU_-**) отмечает, что данный вопрос уже сложно считать однозначным. Курирование данных (data curation) осуществляется вне зависимости от наших предпочтений, а ценности и интересы разработчиков моделей уже встроены в сами модели. Например, для предобучающего набора берется лишь небольшая часть доступных данных. Этот процесс отбора не может быть полностью нейтральным, даже если никто сознательно не акцентирует на этом внимание.

Википедия считается одним из самых востребованных источников данных при обучении больших языковых моделей. Несмотря на то что включение Википедии в предобучающий набор данных может вызывать определенные вопросы, стоит задуматься о возможных последствиях. Основное редактирование контента на платформе осуществляется добровольцами, среди которых преобладает мужская аудитория. Поскольку решение о том, является ли тема достаточно значимой для создания страницы в Википедии, принимается редакторами, большинство которых составляют мужчины, возникают заметные диспропорции. Например, зачастую собственные страницы получают неизвестные футболисты из низших лиг, в то время как значительная часть биографических статей о женщинах оказывается под угрозой удаления.

Аналогичным образом, весьма значимый набор данных WebText был сформирован на основе исходящих ссылок Reddit. Этот ресурс является преимущественно мужской платформой, где 74% пользователей составляют мужчины (**https://oreil.ly/i2RkB**).

Таким образом, вполне логично предположить, что ссылки, публикуемые на Reddit, в большей степени отражают интересы мужской аудитории.

Предвзятость может легко проникнуть на стадии фильтрации данных. Как уже упоминалось ранее, для устранения содержимого порнографического характера или оскорбительных текстов часто применяются списки ключевых слов (keyword list). Однако использование простого списка ключевых слов представляет собой поверхностный подход, который не только страдает от низкой точности (появление ложноотрицательных значений), но и непреднамеренно приводит к исключению положительных материалов, связанных с меньшинствами или написанных представителями этих сообществ (**https://oreil.ly/XWBjV**). Это касается и текстов на диалектах, таких как афроамериканский английский или испанский английский. Сложность заключается в том, что многие слова в английском языке обладают различными значениями, и в результате этого из набора данных C4 были удалены даже некоторые документы, посвященные грудному вскармливанию.

То, как слово воспринимается — как ненавистное, оскорбительное или токсичное, во многом зависит от социального контекста, намерений слушателей и характеристик целевой аудитории. Простые подходы, основанные на ключевых словах, не способны полноценно учесть эти тонкие аспекты. Открытым остается вопрос: следует ли решать подобные задачи на этапе предварительной подготовки или же адресовать их на последующих стадиях? Более детальный разбор возможных методов будет представлен в *главе 10*.



Авторы модели Pythia (**https://oreil.ly/r4oAT**) провели эксперимент, в ходе которого заменили мужские местоимения женскими в последних 7% обучающих токенов. Это позволило оценить, как изменение подхода к обработке данных влияет на снижение предвзятости и результаты выполнения моделью последующих задач.

## Резюме

В этой главе мы рассмотрели ключевые элементы языковой модели, такие как данные для предварительного обучения, словарь, токенизатор, языковая цель и архитектура модели. Были подробно описаны этапы формирования набора данных для предварительной подготовки, включая определение языка, извлечение и очистка текста, фильтрация качества, дедупликация, удаление конфиденциальной информации (PII) и исключение тестовых данных. Также представлен перечень популярных наборов данных для предварительного обучения с описанием шагов, выполненных в рамках их предварительной обработки. В следующем разделе мы подробно рассмотрим словарь и токенизатор языковой модели с учетом специфики изучаемого языка.

# ГЛАВА 3
# Словарь и токенизатор

В предыдущей главе мы рассмотрели наборы данных, которые используются для обучения современных языковых моделей, и проследили их историю. Этот обзор должен был показать, насколько важны предобученные данные для работы итоговой модели. В этой главе мы поговорим о другом важном элементе языковой модели — ее словаре.

## Словарь

С чего начинается изучение нового языка? Первым шагом неизменно становится расширение словарного запаса, который будет увеличиваться по мере прогресса в изучении. Определим словарный запас как

"Совокупность всех слов, усвоенных человеком в рамках конкретного языка".

Среднестатистический носитель английского языка располагает словарным запасом от 20 000 до 35 000 слов (**https://oreil.ly/bkc2C**). Аналогично каждая языковая модель обладает собственным объемом словаря, который обычно варьируется от 5000 до 500 000 *токенов* (token).

В качестве примера рассмотрим словарь модели GPT-NeoX-20B. Для этого нужно открыть файл *tokenizer.json* (**https://oreil.ly/Kages**) и с помощью функции поиска (<Ctrl>+<F>) найти раздел vocab, который содержит словарь модели. Можно заметить, что элементы, входящие в его состав, не полностью совпадают с привычными английскими словами, используемыми в стандартных словарях. Эти единицы называются "типами", а их использование в тексте, т. е. появление конкретного типа в последовательности символов, называется токеном.



В последнее время, особенно в сфере IT, слово "тип" практически вышло из употребления и встречается лишь в старых учебниках по NLP. Вместо него повсеместно используется слово "токен" — как для обозначения единиц словаря, так и для их появления в текстовой последовательности. В дальнейшем термин "токен" будет применяться для описания обоих значений, хотя лично мне такое употребление видится не совсем удачным.

Можно заметить, что в файле словаря каждому токену соответствует определенное число, называемое *идентификатором ввода* (input ID) или *индексом токена* (token index). Объем словарного запаса модели GPT-NeoX составляет чуть более 50 000 единиц.

При детальном изучении файла словаря становится очевидным, что первые несколько сотен токенов представляют собой односимвольные элементы, включая специальные знаки, цифры, заглавные и строчные буквы, а также символы с диакритическими отметками. Более длинные слова встречаются в словаре позднее. Значительная часть токенов приходится на отдельные части слов, которые называются *подсловами* (subword), например impl, inated и т. д.

Нажав комбинацию клавиш <Ctrl>+<F>, найдем слово `office` (офис). В результате поиска будут выведены девять совпадений:

```
"Ġoffice": 3906
"Ġofficer": 5908
"Ġofficers": 6251
"ĠOffice": 7454
"ĠOfficer": 12743
"Ġoffices": 14145
"office": 30496
"Office": 33577
"ĠOfficers": 37209
```

Символ Ġ используется для обозначения пробела перед словом. Например, в предложении "He stopped going to the office" ("Он перестал ходить в офис") пробел перед буквой "о" в слове "офис" считается частью токена. Также важно отметить, что токены зависят от регистра, т. е. существуют отдельные токены для слов "office" и "Office". В современных языковых моделях словари, как правило, сохраняют чувствительность к регистру. Например, модель BERT изначально была представлена в двух вариантах — с учетом регистра (cased) и без него (uncased).



Обучение языковых моделей основано на векторных представлениях, называемых эмбеддингами, которые передают синтаксические и семантические характеристики каждого токена. Мы детально рассмотрим процесс обучения в *главе 4* и подробно остановимся на эмбеддингах в *главе 11*.

Словари, учитывающие регистр (cased vocabulary), демонстрируют превосходную производительность, особенно при обучении на больших объемах текста. Такая особенность обусловлена тем, что модель достаточно часто встречает большинство токенов, что позволяет ей сформировать для каждого токена уникальные представления. Например, такие слова, как web и Web, несут разную семантическую нагрузку, и для более точного их отображения целесообразно выделять отдельные токены.

Теперь перейдем к поиску чисел. Для этого воспользуемся комбинацией клавиш <Ctrl>+<F> и введем 93. Всего обнаружено три совпадения:

```
"93": 4590
"937": 47508
"930": 48180
```

Как видно, не каждому числу присваивается отдельный токен. Например, для числа 934 токен отсутствует. Очевидно, что невозможно предоставить каждому числу отдельный токен, особенно если существует ограничение на размер словаря модели, например в пределах 50 000 элементов. В следующей части этой главы мы рассмотрим, как определяется размер словаря. При этом популярные имена и названия мест обычно получают собственные токены. Существуют токены для таких городов, как Boston (Бостон), Toronto (Торонто) или Amsterdam (Амстердам), но отсутствуют токены для Mesa (Меса) или Chennai (Ченнай). Также можно найти токены для имен Ahmed (Ахмед) и Donald (Дональд), но для таких имен, как Suhas (Сухас) или Maryam (Марьям), токены отсутствуют.

Кроме того, можно отметить наличие таких токенов, как:

```
"]);": 9259
```

Это свидетельствует о том, что GPT-NeoX разрабатывалась с учетом обработки языков программирования.

### Упражнение

Изучите внимательно файл *tokenizer.json* (**https://oreil.ly/FxPcz**) и проанализируйте словарь, обратив особое внимание на следующие моменты:

- Какие непредсказуемые токены можно обнаружить в нем?
- Какие десять самых длинных токенов можно выделить?
- Имеются ли токены, представляющие слова из других языков?

Как же создаются словари? Безусловно, этот вопрос не обсуждался на срочных заседаниях специального комитета, где участники спорили о целесообразности включения в словарь числа 937 за счет исключения числа 934.

Вернемся к определению словаря:

> "Совокупность всех слов, усвоенных человеком в рамках конкретного языка".

В рамках создания языковой модели, способной выступать в роли эксперта в английском языке, может возникнуть предположение, что достаточно просто включить в ее словарь все слова английского языка. Однако это не является решением проблемы.

Очевидно, что при взаимодействии с языковой моделью возникает необходимость в использовании слов, которые ей неизвестны. Это происходит гораздо чаще, чем

можно предположить. Язык постоянно развивается: возникают новые слова, слова употребляются в различных формах (например, *understand* — понимать, *understanding* — понимание, *understandable* — понятный), а некоторые слова объединяются в одно. Кроме того, существует множество специализированных терминов, характерных для различных областей, таких как биомедицина, химия и т. д.

### Определение слова

Определение термина "слово" — задача более сложная, чем может показаться на первый взгляд. С одной стороны, можно определить слово как минимальную единицу текста, имеющую самостоятельный смысл. Однако даже это определение не всегда точно. Например, в слове snowball (снежок, снежный ком, образовано от слов *snow* — снег и *ball* — мяч) каждый компонент имеет собственное значение, но вместе они составляют единое выражение. Если рассматривать слово с точки зрения алгоритмов, то его можно описать как последовательность символов, разделенных пробелом. Но и здесь есть исключения: например, "Hong Kong" ("Гонконг") обычно воспринимается как одно слово, несмотря на наличие пробела. С другой стороны, слово "can't" (буквально "не может") теоретически можно рассматривать как два или даже три слова, хотя между ними нет пробела.



Аккаунт @NYT_first_said (**https://oreil.ly/FzfI9**) на платформе X отмечает появление новых слов, исключая собственные имена, в момент их первой публикации в изданиях *New York Times*. В среднем в США ежедневно регистрируется около пяти новых слов. В день, когда был написан этот абзац, новыми словами стали unflippant (нелегкомысленный), dumbeyed (недальновидный), dewdrenched (покрытый росой), faceflat (плосколицый), saporous (с выраженным вкусом) и dronescape (аэросъемка). Большинство таких слов, скорее всего, так и не попадут в словарь.

Токены, отсутствующие в словаре, называются OOV-токенами (out-of-vocabulary — вне словаря). Ранее такие токены заменялись специальным токеном <UNK>, который служил универсальным заполнителем для всех слов, не входящих в словарь. Однако этот подход имеет ограничения: все OOV-токены используют одинаковый эмбеддинг и представляют одно и то же значение, что далеко не всегда является наиболее подходящим решением. Более того, применение токена <UNK> в генеративных моделях нежелательно, поскольку это может привести к появлению некорректных и трудно интерпретируемых результатов, например:

```
'As a language model, I am trained to <UNK> sequences, and output <UNK> text'.
```

*('В качестве языковой модели я обучена последовательностям <UNK> и выводу текста <UNK>'.)*

Один из способов решения проблемы OOV заключается в переходе к токенизации на уровне символов (character-based tokens) вместо слов. В этом подходе каждому символу назначается уникальный эмбеддинг, а при условии наличия всех допустимых символов в словаре — полностью исключается вероятность появления OOV-

токена. Однако данный метод имеет свои существенные недостатки, главным из которых является значительное увеличение числа токенов, необходимых для представления текста. Например, рассмотрим фразу "The number of tokens needed to represent the average sentence becomes much larger" ("Количество токенов, необходимых для представления среднего предложения, существенно увеличивается"). Если рассчитывать токены на уровне слов, то их всего 13, тогда как при подсчете на уровне символов их число возрастает до 81. Это приводит к сокращению объема контента, который может быть обработан в рамках фиксированной длины последовательности, что негативно влияет на скорость обучения модели и ее производительность при выводе (это будет подробно рассмотрено в *главе 4*). Ограничение длины последовательности модели также подразумевает ограничение количества информации, которую можно вместить в один промпт. В данной главе мы рассмотрим такие модели, как CANINE, ByT5 и Charformer, использующие токенизацию на уровне символов.

Таким образом, перед исследователями стоит задача найти оптимальное решение, которое объединит преимущества обоих подходов. В настоящее время подход на уровне подслов занимает лидирующие позиции в представлении лексики внутри языковых моделей. Например, рассмотренный выше словарь GPT-NeoX организован на основе токенов подслов (subword token). На рис. 3.1 показана платформа токенизатора OpenAI, где можно наглядно увидеть, как модели OpenAI разбивают слово на составляющие подслова.

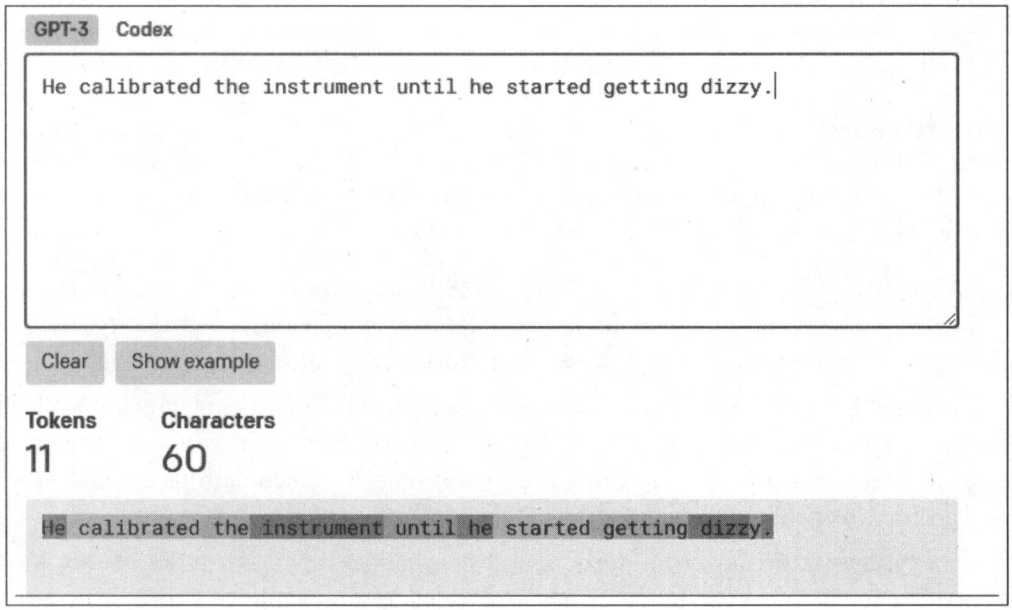



**Рис. 3.1.** Токены подслов[1]

[1] Текст в поле ввода: "Он занимался настройкой прибора, пока не почувствовал головокружение". — *Прим. пер.*

**Оптимальный размер словаря**

Языковые модели могут иметь различные размеры словарей. Например, Llama 3, модель с одинаковым объемом, использует словарь из 128 000 единиц, в то время как Gemma 2 применяет словарь, состоящий из 256 000 слов. В многоязычных моделях обычно используются более обширные словари.

Выбор оптимального размера словаря играет ключевую роль в эффективности сжатия текста. Увеличение размера словаря позволяет сократить количество токенов, необходимых для представления текста, что, в свою очередь, приводит к более эффективному сжатию данных. Таким образом, при сохранении того же объема обучения или вычислительных ресурсов, модель может обрабатывать значительно больше информации. Однако увеличение размера словаря также приводит к появлению редких токенов, которые встречаются в обучающих данных лишь в ограниченном количестве и не всегда имеют полные представления.

В своей работе Tao и соавт. (Tao et al., **https://oreil.ly/gGq8D**) установили закономерности, касающиеся размера словаря. Они обнаружили, что оптимальный размер словаря растет с увеличением размера модели и объема вычислительных ресурсов. Авторы отмечают, что на момент публикации их исследования словари большинства современных моделей не достигли оптимального размера. Однако увеличение размера словаря может значительно повысить эффективность модели.

## Токенизатор

Токенизатор — это инструмент, предоставляющий интерфейс для обработки текста, обеспечивая связь между человеком и моделью.

Токенизатор выполняет две ключевые функции:

1. На этапе предварительного обучения токенизатор обрабатывает корпус текста и формирует словарь, который будет использоваться моделью.
2. В процессе обработки входных данных, как при обучении модели, так и при генерации вывода, токенизатор преобразует необработанный текст в последовательности допустимых токенов. Рабочий процесс токенизатора изображен на рис. 3.2, который визуализирует его роль и этапы выполнения задач.

При поступлении необработанного текста в токенизатор текст разбивается на отдельные токены, соответствующие элементам сформированного словаря. Затем каждому токену назначается его индекс. Полученная последовательность индексов токенов, выступающая в роли идентификаторов ввода, передается в языковую модель, где токены преобразуются в соответствующие эмбеддинги. Рассмотрим этот процесс подробнее.

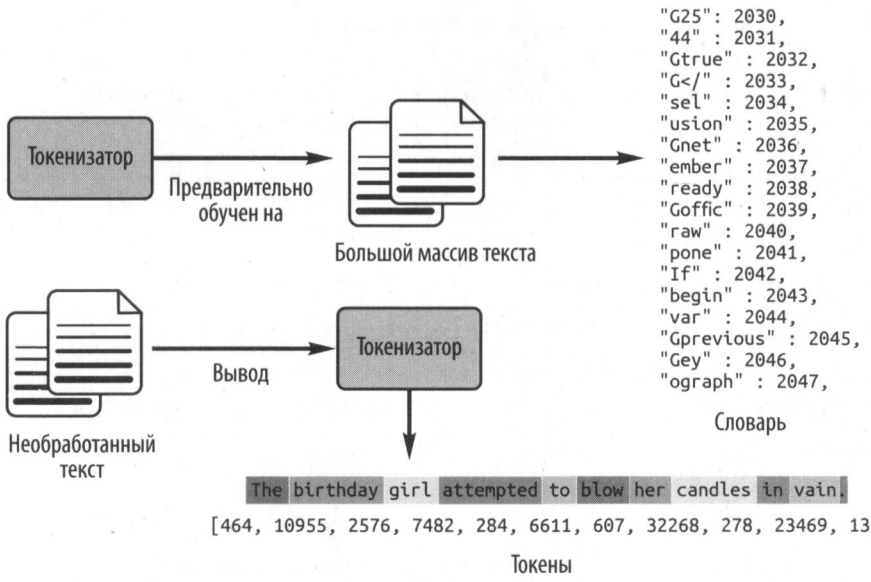



**Рис. 3.2.** Рабочий процесс токенизатора

Ради эксперимента воспользуемся на этот раз моделью FLAN-T5. Чтобы выполнить ее запуск, потребуется доступ к Google Colab Pro или аналогичной платформе с достаточной вычислительной мощностью:

```
!pip install transformers accelerate sentencepiece

from transformers import T5Tokenizer, T5ForConditionalGeneration
tokenizer = T5Tokenizer.from_pretrained("google/flan-t5-largel")
model = T5ForConditionalGeneration.from_pretrained("google/flan-t5-large",
    device_map="auto")

input_text = "what is 937 + 934?"
encoded_text = tokenizer.encode(input_text)
tokens = tokenizer.convert_ids_to_tokens(encoded_text)
print(encoded_text)
print(tokens)
```

На выходе получаем:

```
[125, 19, 668, 4118, 1768, 668, 3710, 58, 1]
['_what', '_is', '_9', '37', '_+', '_9', '34', '?', '</s>']
```

Функция `encode()` выполняет токенизацию входного текста, преобразуя его в соответствующие индексы токенов. Далее с помощью функции `convert_ids_to_tokens()` эти индексы можно сопоставить с токенами, которые они представляют.

Как видно, токенизатор FLAN-T5 не имеет отдельных токенов для чисел 937 и 934. Поэтому эти числа разбиваются на составные части: "9" и "37". Специальный токен </s> используется для обозначения конца строки, а символ _ показывает, что перед данным токеном стоит пробел.

Рассмотрим еще один пример:

```
input_text = "Insuffienct adoption of corduroy pants is the reason this
economy is in the dumps!!!"
// (Слабая популярность вельветовых штанов стала одной из причин,
// по которой эта отрасль переживает упадок!!!)
encoded_text = tokenizer.encode(input_text)
tokens = tokenizer.convert_ids_to_tokens(encoded_text)
print(tokens)
```

На выходе получаем:

```
['_In', 's', 'uff', 'i', 'en', 'c', 't', '_adoption', '_of', '_cord', 'u',
'roy', '_pants', '_is', '_the', '_reason', '_this', '_economy', '_is', '_in',
'_the', '_dump', 's', '!!!', '</s>']
```

Слово insufficient (слабая; введено как Insuffienct) намеренно содержит опечатку. Следует отметить, что процесс разбиения слов на подслова особенно чувствителен к таким ошибкам. Однако проблема с OOV была решена за счет разбиения слов на подслова. Интересно, что в словаре отсутствует запись для слова corduroy (вельвет), что свидетельствует о том, что модель слабо разбирается в терминах из мира моды. Также стоит обратить внимание на то, что для трех восклицательных знаков подряд используется отдельный токен, который отличается от токена для одного знака. Семантически эти два знака выражают разные оттенки смысла.



Масштабные языковые модели, обученные на обширных текстовых корпусах, демонстрируют высокую устойчивость к орфографическим ошибкам. Это связано с тем, что в их обучающих наборах уже содержится значительное количество подобных ошибок. Например, даже редкое неправильное написание слова Insuffienct встречается 14 раз в предобучающем наборе данных C4, а более распространенный вариант ошибки insufficent насчитывается более 1100 раз. Такие модели способны определять слово с ошибкой, опираясь на его контекст. В то же время более компактные модели, такие как BERT, гораздо чувствительнее к орфографическим неточностям.

Пользователи моделей OpenAI имеют возможность ознакомиться с их схемой токенизации с помощью библиотеки tiktoken (**https://oreil.ly/2QByi**). Важно отметить, что эта библиотека не имеет отношения к одноименной социальной сети.

С помощью инструмента tiktoken изучим словари, доступные в экосистеме OpenAI.

```
!pip install tiktoken

import tiktoken
tiktoken.list_encoding_names()
```

На выходе получаем:

```
['gpt2', 'r50k_base', 'p50k_base', 'p50k_edit', 'cl100k_base', 'o200k_base']
```

Считается, что числа вроде 50K или 100K обозначают размер словарного запаса. OpenAI не предоставляет подробной информации о своих словарях. Тем не менее в документации указано, что o200k_base применяется в GPT-4o, а cl100k_base — в GPT-4.

```
encoding = tiktoken.encoding_for_model("gpt-4")
input_ids = encoding.encode("Insuffienct adoption of corduroy pants is the
reason this economy is in the dumps!!!")
tokens = [encoding.decode_single_token_bytes(token) for token in input_ids]
```

На выходе получаем:

```
[b'Ins', b'uff', b'ien', b'ct', b' adoption', b' of', b' cord', b'uro', b'y',
b' pants', b' is', b' the', b' reason', b' this', b' economy', b' is', b' in',
b' the', b' dumps', b'!!!']
```

Как видно, в методах токенизации, используемых в GPT-4 и FLAN-T5, существенных отличий практически не наблюдается.

### Упражнение

В репозитории (**https://oreil.ly/TQoLz**) находятся словари o200k_base и cl100k_base. Определите, в чем заключаются их различия. Какие типы токенов встречаются только в одном из словарей, но отсутствуют в другом?



Если LLM ведет себя непредсказуемо при работе с конкретным набором данных, понимание процесса токенизации поможет более точно определить проблему. Хотя точная диагностика может быть сложной, анализ процесса токенизации часто предоставляет ценные сведения. Ошибки часто возникают из-за неправильного разбиения текста на токены, что особенно заметно, когда целевая предметная область (домен) отличается от домена, на котором обучалась модель.

### Модели без токенизатора

В предыдущей главе мы рассмотрели, как эффект консолидации (consolidation effect) приводит к созданию сквозных архитектур, способных принимать ввод от человека, выполнять все необходимые этапы обработки и выдавать результаты в удобном для восприятия формате — все это в рамках одной модели. Между тем этап токенизации остается неизменным компонентом большинства подобных подходов. Как можно было заметить из приведенного выше примера кода, токенизация служит предварительным шагом для преобразования исходного текста в формат, подходящий для обработки моделью. Таким образом, модель работает не с самим текстом, а с последовательностью индексов токенов.

Но что будет, если избавиться от этого этапа и построить модель, действительно работающую напрямую с необработанным текстом? Может ли такая модель воспринимать входной текст без предварительного преобразования и при этом генерировать качественные выходные данные?

В области языкового моделирования были предприняты попытки отхода от традиционной токенизации с использованием таких моделей, как CANINE, ByT5 и Charformer.

- CANINE (**https://oreil.ly/ucLIk**) ориентируется на кодовые точки Unicode в качестве входных данных. Однако с учетом того, что в системе Unicode насчитывается 1 114 112 возможных кодовых точек, использование словаря такого размера становится непрактичным для конечного слоя эмбеддинга. Чтобы обойти эту проблему, модель применяет хешированные эмбеддинги, которые позволяют значительно сократить объем словарного пространства.
- ByT5 (**https://oreil.ly/x38Vs**) оперирует байтами в качестве входных данных, благодаря чему ее словарь ограничивается всего 259 токенами (включая специальные), что существенно упрощает структуру слоя эмбеддинга и его размер.
- Charformer (**https://oreil.ly/WJY1k**) также опирается на байтовые данные, передавая их через модуль токенизатора, построенный на основе градиентного подхода. Этот механизм способен создавать скрытые подслова, оптимизируя обработку данных.

## Конвейер токенизации

Последовательность действий, осуществляемых токенизатором, проиллюстрирована на рис. 3.3.

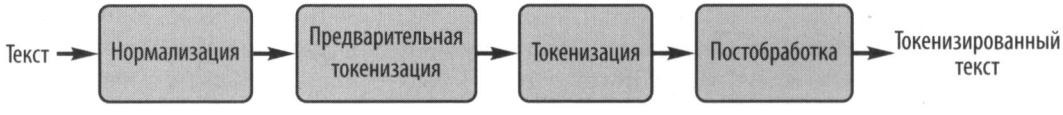



**Рис. 3.3.** Конвейер токенизаторов Hugging Face

При использовании библиотеки токенизаторов Hugging Face исходный текст проходит через многоэтапный конвейер токенизации (multistage tokenization pipeline, **https://oreil.ly/CcOKV**), состоящий из четырех ключевых этапов:

- нормализация (normalization);
- предварительная токенизация (pre-tokenization);
- токенизация (tokenization);
- постобработка (post-processing).

Следует учитывать, что в зависимости от используемой модели порядок действий на каждом из этих этапов может отличаться.

## Нормализация

Нормализация включает в себя различные подходы, в том числе:

- преобразование текста в нижний регистр (актуально для моделей, работающих без учета регистра);
- удаление диакритических знаков из символов, например, как в слове Peña;
- приведение текста к единой форме Unicode.

Посмотрим, какой тип нормализации используется в версии BERT без учета регистра (uncased):

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
print(tokenizer.backend_tokenizer.normalizer.normalize_str(
    'Pédrò pontificated at us:-)')
```

На выходе получаем:

```
pedro pontificated at us:-) // (Педро проповедовал нам)
```

Как видно, диакритические знаки были полностью удалены, а текст преобразован в нижний регистр.

Нормализация пока не получила широкого распространения в токенизаторах современных моделей.

## Предварительная токенизация

Перед началом работы токенизатора с текстом можно добавить этап предварительной токенизации. Как уже отмечалось, современные токенизаторы чаще всего применяют токенизацию на уровне подслов. Общая процедура заключается в том, чтобы сначала провести токенизацию на уровне слов, а затем передать полученные данные алгоритму токенизации подслов. Этот процесс называется предварительной токенизацией (pre-tokenization).

Токенизация английского текста проще по сравнению с другими языками благодаря возможности применения базовой стратегии, основанной на разделении по пробелам. Однако для достижения высокой точности необходимо учитывать специфику английского языка, например правила обработки знаков препинания, множественных пробелов, цифр и других уникальных символов. В Hugging Face для этого используется регулярное выражение

```
\w+ \| \[^[\w\s] +
```

Выполним предварительную токенизацию с помощью токенизатора T5:

```
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-xl")
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("I'm starting to suspect -
I am 55 years old! Time to vist New York?")
// ("В 55 лет меня посещает странное чувство! Может, пора отправиться
// в Нью-Йорк?")
```

На выходе получаем:

```
[("_I'm", (0, 3)),
 ('_starting', (3, 12)),
 ('_to', (12, 15)),
 ('_suspect', (15, 23)),
 ('_-', (23, 25)),
 ('_I', (25, 27)),
 ('_am', (27, 30)),
 ('_55', (30, 33)),
 ('_years', (33, 39)),
 ('_old!', (39, 44)),
 ('_', (44, 45)),
 ('_', (45, 46)),
 ('_Time', (46, 51)),
 ('_to', (51, 54)),
 ('_vist', (54, 59)),
 ('_New', (59, 63)),
 ('_York?', (63, 69))]
```

Предварительный токенизатор T5 возвращает не только предварительные токены (или токены слов), но и смещения, или отступы, символов (character offsets).

Этот токенизатор работает исключительно с пробелами, не объединяя несколько пробелов в один и не учитывая знаки препинания или цифры при разбиении текста. Стоит отметить, что поведение других токенизаторов может существенно отличаться от описанного.

## Токенизация

После предварительного этапа токенизации осуществляется собственно процесс токенизации, для которого существует ряд эффективных алгоритмов. К числу наиболее распространенных методов относятся кодирование пар байтов (byte pair encoding, BPE), BPE на уровне байтов (bytelevel BPE), WordPiece и Unigram LM. Работа токенизатора основана на определенном наборе правил, которые модель усваивает на этапе предобучения на наборе предобучающих данных. Разберем подробно каждый из указанных методов.

## Кодирование пар байтов

Этот метод считается одним из самых простых и широко применяемых алгоритмов токенизации.

### Этап обучения

Прежде всего используется обучающий набор данных, который проходит рассмотренные выше этапы нормализации и предварительной токенизации. После обработки на выходе фиксируются все уникальные токены, а также частота их встречаемости. На основе полученных данных формируется стартовый словарь, содержащий все уникальные символы, из которых состоят эти токены. Затем этот базовый словарь постепенно расширяется с применением *правил слияния* (merge rules). Суть правила заключается в объединении двух наиболее часто встречающихся последовательностей токенов в новый токен. Процесс продолжается до тех пор, пока размер словаря не достигнет заданного размера.

Для наглядности рассмотрим такой пример. Представим, что обучающий набор данных содержит шесть слов, каждое из которых встречается только один раз:

```
'bat', 'cat', 'cap', 'sap', 'map', 'fan'
```

Тогда в стартовый словарь будут включены:

```
'b', 'a', 't', 'c', 'p', 's', 'm', 'f', 'n'
```

Распределение частот для пар близких токенов будет иметь следующий вид:

```
'ba' - 1, 'at' - 2, 'ca' - 2, 'ap' - 3, 'sa' - 1, 'ma' - 1, 'fa' - 1, 'an' - 1
```

Наиболее часто встречаемой парой является "ap", поэтому, следуя правилу слияния, первым слиянием будет объединение "a" и "p". В таком случае словарь будет иметь вид:

```
'b', 'a', 't', 'c', 'p', 's', 'm', 'f', 'n', 'ap'
```

Новое распределение частот будет выглядеть следующим образом:

```
'ba' - 1, 'at' - 2, 'cap' - 1, 'sap' - 1, 'map' - 1, 'fa' - 1, 'an' - 1
```

Теперь наиболее частотной является пара "at", и следующим шагом будет слияние "a" и "t". Этот процесс будет повторяться до тех пор, пока словарь не достигнет заданного размера.

### Этап вывода

По завершении обучения токенизатор готов к использованию для разбиения текста на токены и их подачи в модель. Этот процесс аналогичен этапу обучения: после нормализации и предварительной токенизации входного текста каждый токен разбивается на отдельные символы, к которым последовательно применяются все правила слияния. Затем полученные токены передаются в модель.

Для ознакомления с правилами слияния GPT-NeoX можно снова открыть файл словаря по ссылке **https0://oreil.ly/7JAyY** и воспользоваться сочетанием клавиш <Ctrl>+<F> для поиска термина merges (слияния). Как и предполагалось, первые правила слияния объединяют отдельные символы. В конце списка слияний (merge list) находятся более сложные подслова, такие как out и comes, объединенные в один токен.

### Упражнение

Реализуйте самостоятельно алгоритм BPE, используя подмножество набора данных Википедии (**https://oreil.ly/xwGqK**). Определите, какие токены будут получены при создании словаря размером в 10 000 единиц и проанализируйте, как они отличаются от токенов, используемых в популярных языковых моделях. Далее используйте разработанный токенизатор для токенизации набора данных, относящегося к определенному домену, например, набора данных со статьями по машинному обучению, который можно найти по ссылке **https://oreil.ly/MqEY9**. Охватывают ли токены все технические термины? Это позволит оценить, насколько эффективно языковые модели общего назначения смогут справляться с конкретными задачами.



Так как каждый уникальный символ в обучающем наборе токенизатора будет иметь собственный токен, токенизация гарантирует отсутствие OOV-токенов в тех случаях, когда все встречающиеся в будущем токены состоят из символов, уже присутствовавших в обучающем наборе. Однако Unicode содержит более миллиона кодовых точек и около 150 000 допустимых символов, что значительно превышает возможности словаря размером 30 000 единиц. Если в обучающем наборе отсутствует введенный символ, ему присваивается специальный токен `<UNK>`. Чтобы избежать этой проблемы, применяется модификация BPE, известная как BPE на уровне байтов (bytelevel BPE). Этот алгоритм стартует с 256 токенов, представляющих все символы, которые могут быть закодированы одним байтом, что обеспечивает кодирование каждого символа Unicode путем последовательного соединения соответствующих токенов на уровне байтов. Таким образом, появление токена `<UNK>` исключено. Такой токенизатор используется в семействе моделей GPT.

## Алгоритм WordPiece

Учитывая, что WordPiece имеет много общего с BPE, рассмотрим лишь его отличительные особенности.

В отличие от BPE, который основывается на частотном подходе, WordPiece применяет метод максимального правдоподобия (maximum likehood estimation, MLE). В этом процессе нормализуется частота появления пар токенов в обучающем наборе данных путем умножения частот отдельных токенов. Затем осуществляется слияние пар токенов с наивысшим баллом правдоподобия:

```
score = freq(a,b)/(freq(a) * freq(b))
```

Это означает, что если пара токенов состоит из токенов с низкой частотой, то они будут объединены в первую очередь. На рис. 3.4 показан приоритет слияния и влияние нормализации на последовательность слияния для каждой отдельной частоты.

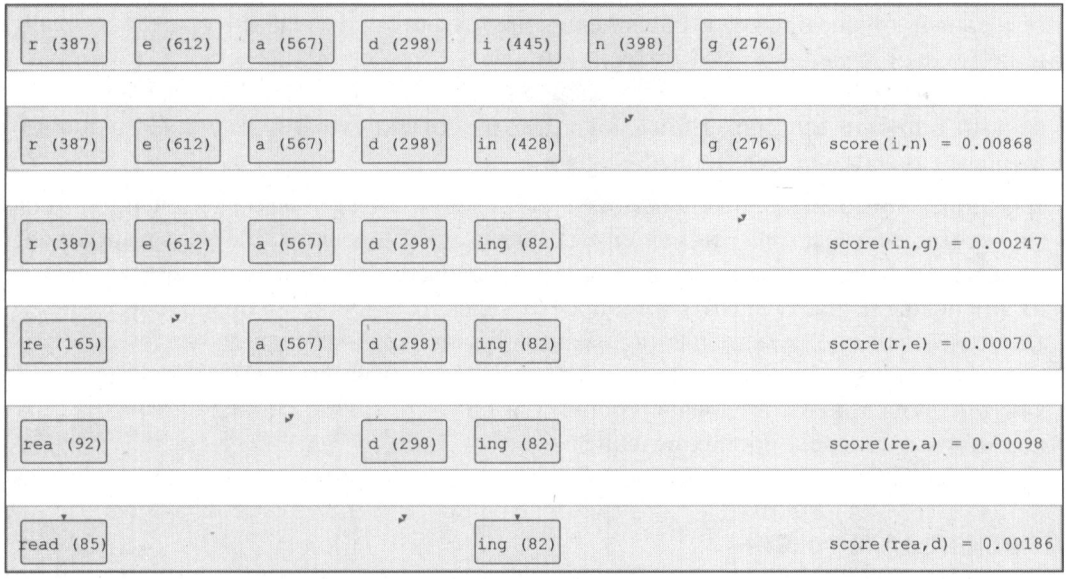



**Рис. 3.4.** Алгоритм токенизации WordPiece

На этапе логического вывода правила слияния не применяются. Вместо этого для каждого токена, полученного в результате предварительной токенизации входного текста, токенизатор ищет самое длинное подслово из словаря, которое содержится в этом токене, и разбивает его на составляющие части. К примеру, если токеном является слово understanding (понимание), а наиболее длинным подсловом из словаря в этом токене оказывается understand (понимать), то он будет разделен на "understand" и "ing".

## Постобработка

После изучения нескольких алгоритмов токенизации мы переходим к следующему этапу конвейера — постобработке. На этом этапе добавляются специальные токены, характерные для конкретной модели. К числу наиболее распространенных токенов относятся токен классификации `[CLS]`, используемый во многих языковых моделях, и разделительный токен `[SEP]`, который служит для разделения различных сегментов входных данных.

### Занятный случай, связанный с токеном SolidMagiGoldkarp

Некоторые токены попадают в словарь языковой модели по весьма необычным причинам, обусловленным особенностями алгоритмов токенизации. Одним из таких примеров является токен SolidMagiGoldkarp, который обозначал удаленного пользователя Reddit, известного своей активностью на сайте и стремлением считать до бесконечности. Этот токен входил в словарь токенизатора GPT-2. Несмотря на то что GPT-3 и SolidMagiGoldkarp используют один и тот же токенизатор, предобучающий набор данных GPT-3 не содержал достаточного количества упоминаний SolidMagiGoldkarp. В результате для SolidMagiGoldkarp был создан

отдельный токен, однако в обучающих данных отсутствовал сигнал, позволяющий модели правильно его интерпретировать. Это приводит к возникновению необычного и даже комичного поведения в GPT-3. Такие токены называются токенами ошибки или сбоя (glitch token, **https://oreil.ly/wnB-z**), а также недообученными токенами (undertrained tokens).

Изучение этимологии токенов (token etymology) стало новым увлечением для множества энтузиастов, занимающихся языковыми моделями. Это захватывающее занятие включает в себя поиск редких слов в словарях моделей с целью выяснения их происхождения. Однако это не просто увлекательное времяпрепровождение, а также ценный инструмент для понимания характеристик предобучающего набора данных. С помощью инструмента tiktoken (**https://oreil.ly/z19c2**) попробуйте выявить редкие токены в словаре GPT-4 или GPT-4o. Возможно ли проследить их корни и выяснить их происхождение?

## Специальные токены

В рамках каждой модели в словарь включаются специальные токены, предназначенные для упрощения обработки. Среди них:

`<PAD>`

Используется для обозначения заполнения, когда длина входных данных не достигает максимальной длины последовательности.

`<EOS>`

Указывает на окончание последовательности. В генеративных моделях процесс генерации завершается после вывода данного токена.

`<UNK>`

Применяется для обозначения термина OOV (out of vocabulary — вне словаря).

`<TOOL_CALL>`, `</TOOL_CALL>`

Информация, заключенная между этими токенами, поступает в качестве входных данных для сторонних инструментов, например для вызовов API или запросов к базам данных.

`<TOOL_RESULT>`, `</TOOL_RESULT>`

Информация, заключенная между этими токенами, служит для отображения результатов, полученных при помощи указанных выше инструментов.

Как уже было показано, использование токенизаторов общего назначения при работе с данными, характерными для определенных областей, таких как здравоохранение или научная литература, не всегда приводит к ожидаемым результатам. В связи с этим в модель GALACTICA от Meta были интегрированы токены, ориентированные на определенные области, а также особые правила токенизации:

- `[START_REF]` и `[END_REF]` для обозначения начала и конца цитат;
- `<WORK>` для выделения токенов, формирующих внутреннюю рабочую память, используемую для логических рассуждений и генерации кода;

- каждой цифре в числе в процессе обработки присваивается уникальный токен;
- `[START_SMILES]`, `[START_DNA]`, `[START_AMINO]`, `[END_SMILES]`, `[END_DNA]`, `[END_AMINO]` для обозначения белковых последовательностей, последовательностей ДНК и аминокислотных последовательностей соответственно.

### Оценка токенизаторов

Двумя популярными метриками для оценки токенизаторов являются фертильность (fertility) и паритет (parity).

*Фертильность*, которая отражает среднее количество токенов, необходимых для представления набора данных, вычисляется как соотношение общего числа токенов к количеству слов в наборе данных. Более высокая фертильность указывает на менее эффективное сжатие токенизатора. В своей работе Гольдман и соавт. (Goldman et al., **https://oreil.ly/ZEG_x**) утверждают, что более эффективное сжатие может способствовать улучшению производительности на следующих этапах, однако это мнение подвергается критике в исследованиях Шмидта и соавт. (Schmidt et al., **https://oreil.ly/GH_iK**). Для достижения более высокого уровня сжатия токенизатор следует обучать на более крупных наборах данных на этапе формирования словаря.

*Паритет* представляет собой меру эффективности токенизатора при работе с двумя языками. Он рассчитывается как отношение количества токенов, необходимых для представления одних и тех же данных на одном языке, к количеству токенов, требуемых для их представления на другом языке.

Многие современные языковые модели обладают мультиязычной поддержкой. Тем не менее, поскольку токенизаторы обычно обучаются на англоязычных наборах данных, токенизация для других языков зачастую оказывается менее эффективной. В результате для представления предложения на языке, отличном от английского, может потребоваться значительно больше токенов по сравнению с эквивалентным предложением на английском, что было подтверждено в исследовании Петрова и соавт. (Petrov et al., **https://oreil.ly/ZATOQ**).

Когда модель работает с данными, специфичными для определенной области, например: здравоохранения, финансов или юриспруденции, — токенизатор, обученный на общем наборе данных, демонстрирует относительно низкий коэффициент сжатия. Это объясняется тем, что слова, характерные для данной области, не всегда представлены в виде отдельных токенов и разбиваются на несколько токенов. Эффективным способом адаптации моделей к узкоспециализированным доменам является обучение их на создании точных векторных представлений терминов, специфичных для каждой области.

Для достижения этой цели можно интегрировать новые токены в уже существующие токенизаторы и продолжить процесс предварительного обучения модели на тематических данных, позволяя новым токенам освоить эффективные представления. Подробное рассмотрение непрерывного предобучения представлено в *главе 7*.

Рассмотрим, как с помощью Hugging Face можно расширить словарь за счет добавления новых токенов.

Возьмем такое предложение: "The addition of CAR-T cells and antisense oligonucleotides drove down incidence rates" ("Добавление CAR-T-клеток и антисмысловых олигонуклеотидов привело к снижению заболеваемости"). Токенизатор FLAN-T5 разбивает его следующим образом:

```
['_The', '_addition', '_of ', '_C', 'AR', '-', 'T', '_cells', '_and', '_anti',
's', 'ense', '_', 'oli', 'gon', 'u', 'cle', 'o', 't', 'ides', '_drove', '_down',
'_incidence', '_rates', '.', '</s>']
```

Внесем в словарь термины, специфичные для данной области:

```
from transformers import T5Tokenizer, T5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("google/flan-t5-large")
model = T5ForConditionalGeneration.from_pretrained("google/flan-t5-large",
    device_map="auto")

tokenizer.add_tokens(["CAR-T", "antisense", "oligonucleotides"])
model.resize_token_embeddings(len(tokenizer))
```

В результате повторной токенизации строки получаем следующие токены, дополненные токенами, характерными для данной области:

```
['_The', '_addition', '_of ', 'CAR-T', '_cells', '_and', 'antisense',
'oligonucleotides', '_drove', '_down', '_incidence', '_rates', '.', '</s>']
```

Мы лишь на середине пути. Векторы эмбеддингов (embedding vectors), соответствующие новым токенам, пока не несут в себе никакой информации о них. Для получения корректных представлений этих токенов необходимо применить дообучение (или тонкую настройку (fine-tuning)) либо непрерывное предобучение (continued pre-training), которое будет подробно описано в *главе 7*.

## Резюме

В этой главе мы в подробностях исследовали ключевой элемент языковых моделей — их словари. Мы проследили, как они конструируются и формируются в рамках моделей. Мы также разобрали концепцию токенизации и рассмотрели такие алгоритмы, как BPE и WordPiece, используемые для создания словарей и разбиения исходного текста на последовательность токенов, воспринимаемые языковой моделью. В ходе исследования мы проанализировали словари, используемые популярными языковыми моделями, и обнаружили расхождения между их токенизацией и восприятием слова человеком.

В следующей главе мы продолжим изучение оставшихся компонентов языковой модели, включая ее архитектуру и цели обучения.

# ГЛАВА 4
# Архитектура и цель обучения

*В главах 2* и *3* мы рассмотрели основные элементы, составляющие языковую модель: обучающие наборы данных, словарь и токенизатор. Теперь настало время собрать все воедино и углубиться в изучение самих моделей, их архитектур и целей обучения.

В данной главе мы подробно изучим состав и структуру современных языковых моделей. Поскольку на сегодняшний день большинство из них строится на основе архитектуры трансформера (Transformer), основное внимание будет уделено ключевым аспектам этой технологии и детальному рассмотрению всех ее компонентов. В последние годы появились различные модификации и альтернативные подходы к стандартной архитектуре трансформера, и мы рассмотрим наиболее перспективные из них, включая модели Mixture of Experts (MoE), что можно перевести буквально как "смеси экспертов". Кроме того, мы проанализируем актуальные задачи, связанные с обучением языковых моделей, и подробно разберем процесс предсказания следующего токена. В завершение главы накопленные теоретические знания из предыдущих разделов будут применены к практической задаче — предварительному обучению языковой модели с нуля.

## Предварительные сведения

Большинство современных языковых моделей построены на основе нейронных сетей, состоящих из обработчиков информации, именуемых *нейронами* (neuron). Несмотря на то что современные нейронные сети существенно отличаются от работы человеческого мозга, многие ключевые концепции и используемые термины в этой области заимствованы из нейробиологии.

В нейронной сети нейроны объединяются во взаимосвязанную структуру, сформированную определенной конфигурацией соединений. Каждое соединение между двумя нейронами характеризуется *весом* (weight), также называемым *параметром* (parameter), который определяет степень их взаимодействия. Совокупность функций этих нейронов и их взаимосвязей формирует *архитектуру* (architecture) модели.

В начале 2000-х годов широкое распространение получили многослойные архитектуры, в которых нейронные слои последовательно накладывались друг на друга.

Каждый следующий слой выявлял более сложные особенности входных данных. Этот подход получил название *глубокого обучения* (deep learning).

На рис. 4.1 приведен пример простой многослойной нейронной сети, также известной как многослойный перцептрон (multi-layer perceptron).

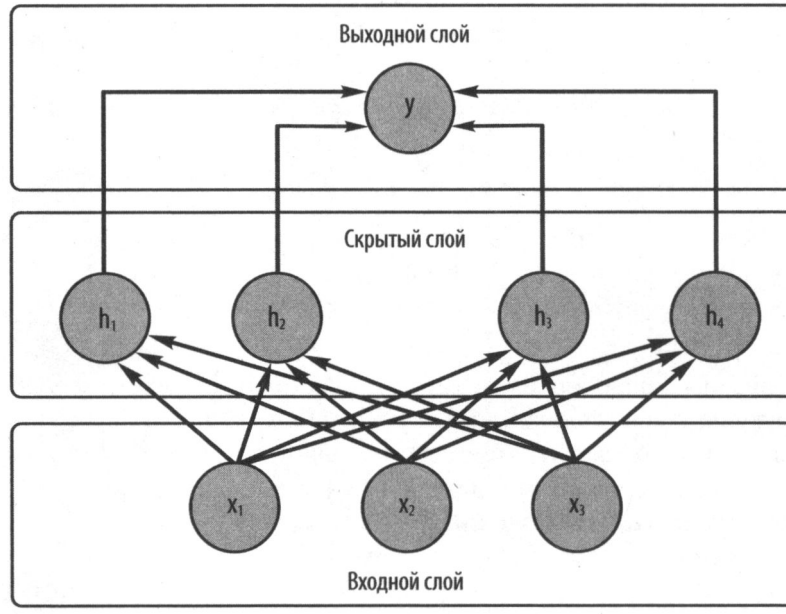



**Рис 4.1.** Многослойный перцептрон



Для более глубокого понимания нейронных сетей рекомендуется ознакомиться с книгой Йоава Гольдберга[1] (**https://oreil.ly/oDc6x**), в которой подробно рассматривается использование нейронных сетей для обработки естественного языка.

Как уже отмечалось в *главе 1*, основным методом обучения языковых моделей является самоконтролируемое обучение (self-supervised learning). Для этого входной текст из обучающего набора данных разбивается на отдельные токены и преобразуется в векторное представление. Затем преобразованный текст проходит через нейронную сеть, параметры которой определяются ее весами и *функциями активации* (activation functions). Последние добавляют элемент нелинейности в работу модели. Полученный вывод модели сопоставляется с эталонным значением, называемым "золотым стандартом". Веса модели корректируются таким образом, чтобы при следующем выполнении с теми же входными данными результат оказался более близким к эталону.

[1] Goldberg Y. Neural Network Methods for Natural Language Processing. — Springer Cham, 2017. — 310 p. (Гольдберг И. Нейросетевые методы в обработке естественного языка.)

На практике процесс адаптации реализуется с помощью *функции потерь* (loss function). Задача модели состоит в минимизации этих потерь, которые представляют собой разницу между выходом модели и "золотым стандартом". Для минимизации потерь веса́ обновляются с использованием метода градиентного спуска (gradient descent), известного как *обратное распространение* (backpropagation). Прежде чем переходить к обучению модели, сначала рекомендуется изучить и понять работу этого алгоритма.

**Самоконтролируемое и контролируемое обучение**

Разделение между самоконтролируемым и контролируемым обучением (или обучением с учителем — supervised learning) носит условный характер. Под "обучением с учителем" подразумевается процесс, при котором модель обучается на заранее подготовленных парах входных данных и соответствующих выходов (пары "вход — выход"). Эти выходные метки традиционно аннотируются либо человеком, либо автоматически компьютером. В отличие от этого, в самоконтролируемом обучении метки для целевых значений извлекаются из самих же входных данных, что устраняет необходимость их дополнительной аннотации. Например, обучение на текстах из Интернета с задачей предсказания следующего токена строится на основе текста как такового. Истинное значение, которое необходимо предсказать (следующий токен), уже содержится внутри исходного текста, что делает задачу самоконтролируемой.

## Представление значения слова

В предыдущем разделе, обсуждая нейронные сети, мы указали, что входной текст преобразуется в векторы, которые затем передаются по сети. Но из чего состоят эти векторы и что они представляют собой?

В идеале, после обучения модели, эти векторы должны точно отображать определенный аспект смысла исходного текста, включая социальные коннотации. Создание качественных представлений для различных видов данных, таких как текст или изображения, является ключевой задачей активно развивающейся области исследований, которая называется *обучением представлениям* (representation learning).



При обучении языковой модели с нуля векторы на начальном этапе не несут никакого смысла, т. к. они задаются случайным образом. Для корректной инициализации используются специальные алгоритмы, такие как Glorot, He и др. Подробную информацию о методах инициализации нейронной сети можно найти по ссылке **https://oreil.ly/A8Iro**.

Может ли набор чисел передавать смысл? Людям бывает непросто выразить значение слова или предложения, а уж тем более преобразовать его в числовую форму, доступную для понимания компьютера. Внешняя форма слова, т. е. его написание,

сама по себе не содержит информации о его значении. Например, последовательность букв в слове umbrella (зонтик) не раскрывает его смысл, даже если нам известно огромное количество других слов английского языка.

Основным способом представления значения в числовом формате является структура *гипотезы распределения* (distributional hypothesis). Согласно этой гипотезе, слова с близким значением обычно появляются в схожих контекстах. Суть данной идеи наиболее точно передает следующая фраза:

> "Значение слова определяется его контекстом (You shall know a word by the company it keeps)".
>
> *Джон Руперт Ферт, 1957*

Один из основных способов понять значение незнакомого слова состоит в том, чтобы определить его смысл исходя из контекста, без обращения к словарю. Как известно, значительная часть слов усваивается не через непосредственное изучение или использование словарей, а через анализ их значений в тех контекстах, в которых они употребляются.

Рассмотрим, как гипотеза распределения реализуется на практике. В библиотеке Natural Language Toolkit (NLTK) есть инструмент, который называется *concordance view* (вид конкорданса). Он позволяет увидеть контексты, в которых определенное слово используется в корпусе текстов. Для примера рассмотрим слово *nervous* (нервный) из классического романа Джейн Остин "Эмма", где оно появляется в различных контекстах:

```
from nltk.corpus import gutenberg
from nltk.text import Text
corpus = gutenberg.words('austen-emma.txt')
text = Text(corpus)
text.concordance("nervous")
```

На выходе получаем:

```
Displaying 11 of 11 matches:
...spirits required support . He was a nervous man , easily depressed...
...sitting for his picture made him so nervous , that I could only take...
...assure you , excepting those little nervous headaches and palpitations...
...My visit was of use to the nervous part of her complaint , I hope...
...much at ease on the subject as his nervous constitution allowed...
...Her father was growing nervous , and could not understand her...
...
```

*(Вывод 11 из 11 совпадений:*

*...дух нуждался в поддержке. Человек он был слабонервный, легко впадал в хандру...*

*...сидеть и позировать, повергала его в такое волнение, что я могла рисовать его лишь...*

*...уверяю вас, что, не считая легких приступов нервического сердцебиения и головной боли...*

*...В части нервов мой визит, возможно, доставил ей облегчение...*

*...мгновенно успокоился, сколько позволяла ему его нервическая организация...*

*...Ее отец начинал приходить в волнение и неспособен был понять ее...)*

**Упражнение**

Предположим, вам не известно значение слова nervous (нервный). Можно ли понять его смысл, опираясь только на различные контексты, в которых оно используется?

Проанализируйте контексты слов, которые можно считать близкими по значению к слову "нервный". Насколько они похожи на контексты самого этого слова?

## Архитектура трансформера

Мы рассмотрели процесс преобразования текста в векторное представление, и теперь можно перейти к изучению современной канонической архитектуры, которая используется для обучения языковых моделей, — архитектуре трансформера (Transformer).

В середине 2010-х годов при решении задач обработки естественного языка лидирующую позицию занимали рекуррентные нейронные сети, в частности, их модификация с долгой краткосрочной памятью (long short-term memory, LSTM). Для понимания основ, изложенных в книге, детальное изучение рекуррентных нейронных сетей необязательно, но для более глубокого погружения в тему рекомендуется ознакомиться с книгой Йоава Гольдберга "Нейросетевые методы в обработке естественного языка" (**https://oreil.ly/CHCTd**).

Рекуррентные нейронные сети (recurrent neural networks) были созданы для работы с последовательностями, они обрабатывали текст пошагово, анализируя его по отдельным токенам. Для отражения состояния всей последовательности использовался единый вектор, который с увеличением длины последовательности должен был содержать все больше информации. Однако такая последовательная обработка затрудняла учет долгосрочных взаимосвязей, поскольку сохранять данные из начала последовательности становилось все сложнее.

Этот вопрос был четко обозначен Рэем Муни (Ray Mooney), ведущим специалистом в области компьютерных наук, который на конференции Ассоциации компьютерной лингвистики (Association for Computational Linguistics, ACL) в 2014 году подчеркнул проблему:

> "Невозможно уместить значение целого предложения %&!$# в одном-единственном векторе $&!#*".
>
> *Рэй Муни, 2014*

С учетом таких ограничений возникла необходимость создания новой архитектуры, способной преодолеть недостатки LSTM. К их числу относятся ограниченность в представлении дальнодействующих зависимостей, зависимость от одного вектора, отражающего состояние всей последовательности, и ряд других ограничений. В ответ на эти вызовы была разработана архитектура трансформера, предлагающая более гибкий и эффективный подход к обработке последовательностей данных.

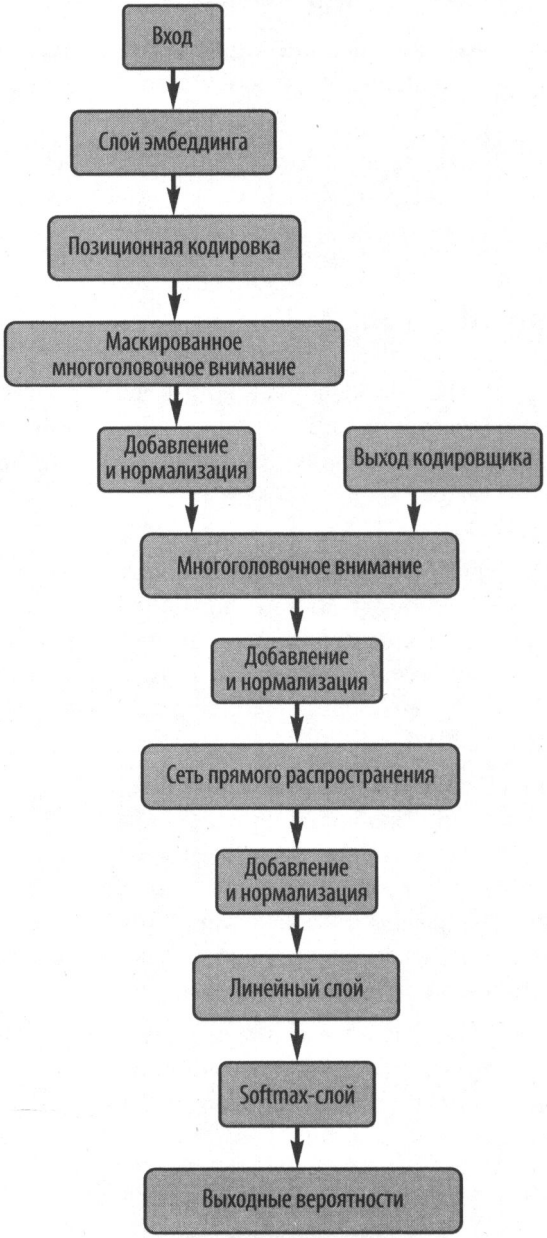



**Рис. 4.2.** Архитектура трансформера

На рис. 4.2 показана оригинальная архитектура трансформера, предложенная командой Васвани в 2017 году (Vaswani et al., **https://oreil.ly/tIvGZ**). Как видно из рисунка, модель Transformer обычно формируется из нескольких расположенных друг над другом модулей, которые называются *слоями* (layer).

Ключевыми компонентами каждого модуля являются:

- механизм внутреннего внимания (self-attention);
- позиционное кодирование (positional encoding);
- сети прямого распространения (feedforward networks);
- модули нормализации (normalization blocks).

В начале первого модуля находится специальный слой, называемый *слоем эмбеддинга* (embedding layer), где каждый токен из входного текста сопоставляется с уникальным вектором. Данный слой представляет собой матрицу, размер которой определяется следующим образом:

```
Количество токенов в словаре * Размерность вектора
```

Для детального изучения слоя эмбеддинга можно воспользоваться библиотекой трансформеров, доступной на платформе Hugging Face:

```
import torch
from transformers import LlamaTokenizer, LlamaModel

tokenizer = LlamaTokenizer.from_pretrained('llama3-base')
model = LlamaModel.from_pretrained('llama3-base')

sentence = "He ate it all" // (Он съел все это)

inputs = tokenizer(sentence, return_tensors="pt")
input_ids = inputs['input_ids']
tokens = tokenizer.convert_ids_to_tokens(input_ids[0])

with torch.no_grad():
    embeddings = model.embeddings(input_ids)

for token, embedding in zip(tokens, embeddings[0]):
    print(f"Token: {token}\n
    print(f"Embedding: {embedding}\n")
```

Векторы эмбеддингов представляют собой входные данные, которые затем распространяются по остальной части сети.

Далее мы подробно рассмотрим каждый из компонентов модуля трансформера и выясним, какую роль они играют в процессе моделирования.

## Механизм внутреннего внимания

Механизм внутреннего внимания (self-attention) опирается на тот же принцип, что и гипотеза распределения, описанная в *разд. "Представление значения слова" ранее в этой главе*, где ключевая роль отводится контексту в определении значения каждого токена. Этот метод формирует векторные представления для каждого токена в текстовой последовательности, учитывая разнообразные аспекты языка, включая синтаксис, семантику и даже прагматику.

В типичной реализации механизма внутреннего внимания представление каждого токена строится на основе представлений всех остальных токенов в последовательности. При этом для определения представления конкретного токена токенам, оказывающим наиболее сильное влияние на его значение, присваивается больший вес.

Например, рассмотрим такую последовательность:

```
'Mark told Sam that he was planning to resign.'
```

*(Марк сказал Сэму, что он планирует уволиться.)*

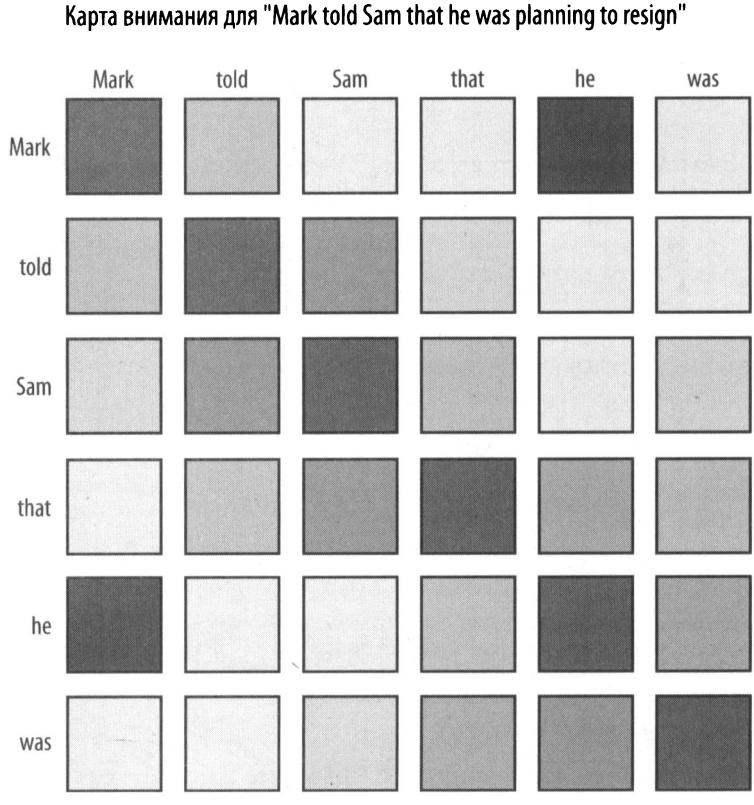



**Рис. 4.3.** Карта внимания

На рис. 4.3 показано, как представление токена *he* (он) значительно зависит от представления токена *Mark* (Марк). В данном случае токен *he* является местоиме-

нием, используемым для краткого обозначения персонажа *Mark*. В области обработки естественного языка (NLP) процесс сопоставления местоимения с его референтом называется *разрешением кореференции* (coreference resolution).

В трансформерах механизм внутреннего внимания реализуется через три набора весовых матриц (weight matrices): матрицы запроса (query, Q), матрицы ключа (key, K) и матрицы значения (value, V). Рассмотрим их подробнее. На рис. 4.4 показан процесс использования этих матриц при расчете внутреннего внимания.

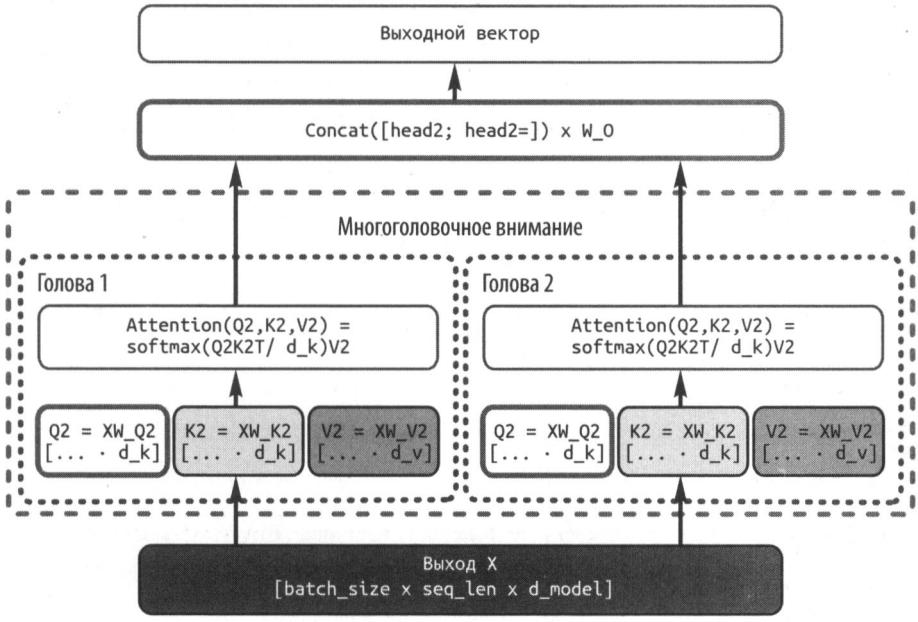



**Рис. 4.4.** Расчет внутреннего внимания

Каждый токен имеет свой уникальный *вектор эмбеддинга* (embedding vector), который затем умножается на весовые матрицы запроса, ключа и значения. В результате формируются три отдельных входных вектора. После этого для каждого токена вычисляется значение механизма внутреннего внимания:

1. Для каждого токена вычисляется скалярное произведение его вектора запроса с векторами ключа всех остальных токенов, включая его самого. Такие значения называют *оценками внимания* (attention score).
2. Значения оценок уменьшаются путем их деления на квадратный корень из размерности векторов ключа.
3. Полученные оценки проходят через *функцию softmax* (softmax function, **https://oreil.ly/b6gHV**), преобразующую их в распределение вероятностей, сумма которых равна 1. Функция активации softmax имеет свойство усиливать большие значения. Именно это объясняет необходимость уменьшения оценок внимания на предыдущем этапе.

4. Полученные нормализованные оценки внимания масштабируются путем умножения на вектор значений каждого соответствующего токена. Это позволяет интерпретировать оценки как пропорции, которые отражают относительный вклад каждого токена в формирование представления текущего токена.
5. В реальных применениях используются различные наборы векторов запросов, ключей и значений для вычисления параллельных представлений. Такой подход называется многоголовочным вниманием (multi-headed attention). Применение нескольких головок позволяет модели более полно охватить различные аспекты входных данных, т. к. каждая голова может фокусироваться на определенном аспекте. Чем больше головок используется, тем выше вероятность того, что все важные аспекты входных данных будут *правильно* представлены.

Механизм внутреннего внимания реализуется в коде следующим образом:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

q = wQ(input_embeddings)
k = WK(input_embeddings)
v = WV(input_embeddings)
dim_k = k.size(-1)

attn_scores = torch.matmul(q, k.transpose(-2, -1))
scaled_attn_scores = attn_scores/torch.sqrt(torch.tensor(dim_k, dtype=torch.float32))

normalized_attn_scores = F.softmax(scaled_attn_scores, dim=-1)

output = torch.matmul(normalized_attn_scores, v)
```



В некоторых версиях трансформеров механизм внутреннего внимания работает только с частью токенов в последовательности, поэтому векторное представление каждого токена формируется на основе представлений лишь некоторых, а не всех токенов в последовательности.

## Позиционное кодирование

Ранее мы уже говорили о том, что архитектуры, предшествующие трансформерам, такие как LSTM, работали с последовательностями, обрабатывая каждый токен по отдельности. В этих моделях информация о порядке токенов, или позиционная информация, закладывалась в систему неявно. В трансформерах же все вычисления выполняются параллельно, что требует наличия явного механизма для передачи данных о позиции токенов. В настоящее время существует множество подходов к внедрению позиционной информации, и эта область исследований продолжает ос-

таваться довольно актуальной. Среди наиболее распространенных методов, используемых в современных языковых моделях, можно выделить следующие.

*Абсолютные позиционные эмбеддинги (absolute positional embeddings)*

Эта методика применялась в оригинальной реализации модели Transformer, представленной Васвани и соавт. (Vaswani et al., **https://oreil.ly/CDq60**). Примерами моделей, в которых используются абсолютные позиционные эмбеддинги, являются такие ранние архитектуры, как BERT и RoBERTa.

*Внимание с линейными смещениями (Attention with Linear Biases, ALiBi)*

В данной технике оценкам внимания назначаются штрафы (**https://arxiv.org/abs/2108.12409**), определяемые специальной *составляющей смещения* (bias term), величина которого пропорциональна расстоянию между токеном запроса (query token) и токеном ключа (key token). Такой подход также позволяет обрабатывать последовательности, превышающие длину, предусмотренную в процессе обучения.

*Ротационные позиционные эмбеддинги (Rotary Position Embedding RoPE)*

Аналогично ALiBi, этот метод (**https://arxiv.org/abs/2104.09864**) обладает свойством относительного снижения значений. При увеличении расстояния между токенами запроса и ключа происходит уменьшение оценок внимания, что делает взаимодействия в модели более локализованными.

*Кодирование без позиционирования (No Positional Encoding, NoPE)*

Альтернативный подход (**https://oreil.ly/QM9dW**) утверждает, что позиционные эмбеддинги в трансформерах могут быть излишними, поскольку модели способны неявно обучаться захвату позиционной информации без дополнительного явного кодирования.

Большинство современных моделей опираются на ALiBi или RoPE, и в то же время, этот элемент архитектуры трансформера до сих пор является предметом активных исследований и продолжает совершенствоваться.

## Сеть прямого распространения

Вывод из модуля механизма внутреннего внимания подается по сети прямого распространения (feedforward networks, **https://oreil.ly/Bdphg**). Каждое представление токена поступает в сеть независимо. В этой сети используется нелинейная функция активации, в частности, ReLU (rectified linear unit — выпрямленная линейная единица, **https://oreil.ly/KUqtP**) или GELU (Gaussian Error Linear Unit — линейная единица с гауссовой ошибкой, **https://oreil.ly/MSDKE**), что позволяет модели обучаться более сложным признакам из данных. Дополнительную информацию об упомянутых функциях активации можно найти в блоге на платформе v7 (**https://oreil.ly/NfOb0**).

Реализация уровней с прямым распространением в коде выглядит следующим образом:

```
import torch
import torch.nn as nn
```

```
class FeedForward(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(FeedForward, self).__init__()
        self.l1 = nn.Linear(input_dim, hidden_dim)
        self.l2 = nn.Linear(hidden_dim, input_dim)
        self.selu = nn.SeLU()

    def forward(self, x):
        x = self.selu(self.l1(x))
        x = self.l2(x)
        return x

feed_forward = FeedForward(input_dim, hidden_dim)
outputs = feed_forward(inputs)
```

## Нормализация слоев

Нормализация слоев (layer normalization) используется для стабилизации процесса обучения и ускорения его конвергенции. В отличие от исходной архитектуры трансформера, где нормализация выполнялась в начале каждого модуля, в современных подходах она применяется в конце модуля.

Процесс нормализации осуществляется следующим образом:

1. При заданных значениях размера пакета `b`, длины последовательности `n` и размерности вектора `d` для каждой составляющей вектора вычисляются среднее арифметическое и дисперсия.
2. Нормализация входных данных путем вычитания их среднего значения и деления на квадратный корень из дисперсии. Для обеспечения числовой устойчивости в знаменатель добавляется небольшое значение погрешности.
3. Умножение на параметр масштаба и добавление параметра сдвига к итоговым значениям. Эти параметры изучаются в процессе обучения.

В коде это реализуется следующим образом:

```
import torch
import torch.nn as nn

class LayerNorm(nn.Module):
    def __init__(self, dimension, gamma=None, beta=None, epsilon=1e-5):
        super(LayerNorm, self).__init__()
        self.epsilon = epsilon
        self.gamma = gamma if gamma is not None else
        nn.Parameter(torch.ones(dimension))
```

```
        self.beta = beta if beta is not None else
        nn.Parameter(torch.zeros(dimension))

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        variance = x.var(-1, keepdim=True, unbiased=False)
        x_normalized = (x - mean) / torch.sqrt(variance + self.epsilon)
        return self.gamma * x_normalized + self.beta

layer_norm = LayerNorm(embedding_dim)
outputs = layer_norm(inputs)
```

## Функция потерь

Итак, мы подробно разобрали все составляющие каждого модуля трансформера. На следующем этапе обучения, направленном на предсказание токенов, входные данные проходят через слои трансформера, после чего формируются конечные выходные данные. Эти данные являются вероятностным распределением по всем токенам. Для оценки работы модели на данном этапе применяется функция потерь (loss function), которые рассчитываются через сравнение полученного распределения с "золотым стандартом". В данном случае стандарт представляет собой распределение, в котором целевой токен получает значение 1, а всем остальным токенам присваивается значение 0.

Для измерения степени расхождения между исходным выходным распределением и "золотым стандартом" используются различные подходы. Наиболее распространенный из них — метод перекрестной энтропии (cross-entropy), которая вычисляется по следующей формуле:

```
Перекрестная энтропия = - Σ (вероятность "золотого стандарта") × log (вероятность выхода)
```

Рассмотрим такую последовательность:

```
'His pizza tasted _____'
```

---

```
'Его пицца была на вкус _____'
```

Предположим, что за "золотой стандарт" принято значение "хорошая", а вероятность его исхода распределяется следующим образом: "ужасная" — 0,65, "плохая" — 0,12, "хорошая" — 0,11.

Перекрестная энтропия вычисляется следующим образом:

```
- (0 × log(0,65) + 0 × log(0,12) + 1 × log(0,11) + ...) = -log(0,11)
```

Поскольку единственное верное значение в "золотом стандарте" имеет вероятность равную единице (остальные равны нулю), уравнение можно упростить до:

```
Перекрестная энтропия = -log(вероятность эталонного токена)
```

После расчета потерь производится вычисление градиента этих потерь по параметрам модели, на основе чего алгоритм обратного распространения ошибки (backpropagation) обновляет веса для улучшения последующих предсказаний.

## Внутренняя оценка модели

Как убедиться, что алгоритм обратного распространения действительно выполняет свою задачу и приводит к постепенному улучшению качества модели? Одним из решений является использование внутренней или внешней оценки.

*Внешняя оценка* (extrinsic evaluation) подразумевает проверку эффективности модели на реальных прикладных задачах, с которыми она может столкнуться в будущем. Такие тесты позволяют оценить практическую применимость алгоритма, но обычно охватывают лишь ограниченный спектр возможных сценариев использования. В то же время *внутренняя оценка* (intrinsic evaluation) дает более целостное представление о потенциале модели в понимании и генерировании языка. Это позволяет провести анализ ее прогресса без привязки к специфическим прикладным задачам. Однако стоит помнить, что высокая эффективность по внутренним метрикам не всегда означает устойчивую производительность при решении новых или непредсказуемых задач в реальных условиях.

Одной из наиболее часто используемых метрик для оценки качества языковых моделей является перплексия (perplexity). Она демонстрирует, насколько эффективно модель способна предсказывать следующий токен в последовательности. Если модель предсказывает каждый токен верно, значение перплексии составляет 1. С увеличением значения перплексии качество модели ухудшается. В случае, когда предсказания модели носят случайный характер и вероятность выбора каждого токена из словаря размером $V$ равняется $1/V$, перплексия модели достигает значения $V$.

Этот показатель напрямую связан с кросс-энтропией и вычисляется по следующей формуле:

```
Перплексия = 2^Кросс-энтропия
```

## Архитектурные магистрали трансформера

Таким образом, мы подробно разобрали компоненты канонической версии трансформера. В реальных приложениях выделяют три ключевых типа архитектур трансформера:

- архитектура "только с кодировщиком" (encoder-only);
- архитектура "кодер-декодер" (encoder-decoder);
- архитектура "только с декодером" (decoder only).

Остановимся на каждой архитектуре более подробно.

На рис. 4.5 представлены графические визуализации этих архитектур: "только с кодировщиком", "кодер-декодер" и "только с декодером".

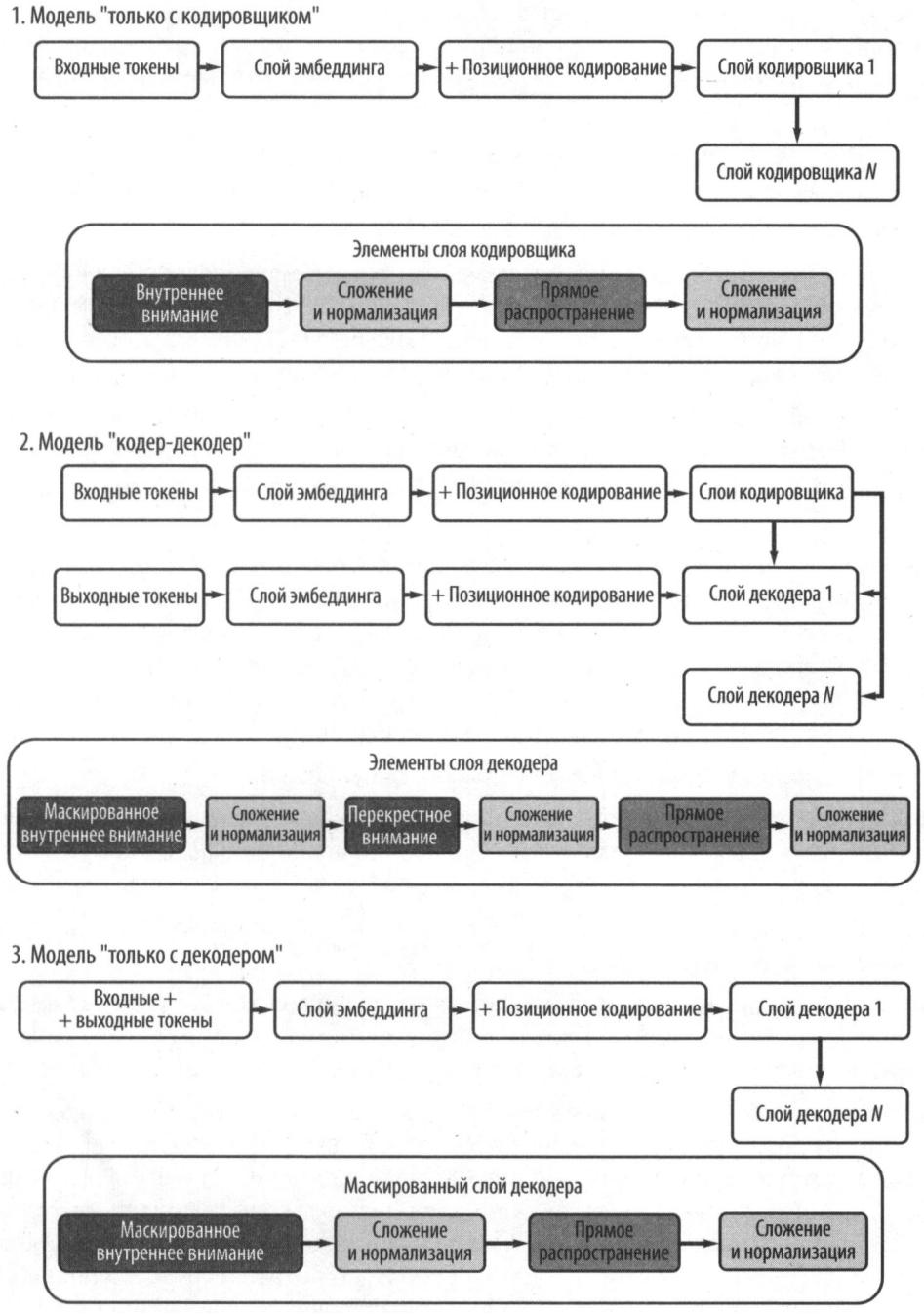



**Рис. 4.5.** Схематическое представление различных архитектур трансформера

## Архитектура "только с кодировщиком"

Архитектура "только с кодировщиком" (encoder only) пользовалась огромной популярностью в период, когда языковые модели на основе трансформера только начали набирать обороты. К числу знаковых моделей того времени (примерно в 2018 году), можно отнести BERT, RoBERTa и другие, которые использовали данный подход.

Однако с 2021 года обучение больших языковых моделей только с кодировщиком стало менее распространенной практикой. Это обусловлено рядом факторов.

- Такие модели довольно сложно обучать.
- Маскированное моделирование языка, часто применяемое для их обучения, предоставляет обучающий сигнал лишь для небольшой доли токенов (коэффициент маскировки). В результате требуется значительно больше данных, чтобы достигнуть уровня производительности, сопоставимого с моделями "только с декодером".
- Для каждой новой задачи необходимо создавать специализированный модуль (головку модели), что усложняет и делает использование менее эффективным.

Тем не менее появление модели ModernBERT привнесло свежий импульс в эту область.

Разработчики языковой модели UL2 утверждают, что модели "только с кодировщиком" уже можно отнести к устаревшим. Однако я не согласен с этой категоричной позицией, поскольку такие модели по-прежнему демонстрируют впечатляющие результаты в задачах классификации. К тому же, если у вас уже работает эффективный конвейер на базе моделей "только с кодировщиком", нет смысла его перестраивать, если все функционирует без нареканий.

Рассмотрим несколько доводов в поддержку этих моделей.

- В большинстве случаев RoBERTa показывает лучшие результаты по сравнению с BERT, поскольку обучалась значительно дольше на большем объеме данных и интегрировала улучшенные практики, разработанные после появления BERT.
- DeBERTa и ModernBERT считаются самыми эффективными среди моделей только с кодировщиком.
- Дистиллированные версии моделей "только с кодировщиком", такие как DistilBERT и подобные, демонстрируют производительность, близкую к оригинальным версиям, поэтому их стоит учитывать при работе в условиях ограниченных ресурсов.

На архитектуре "только с кодировщиком" построены некоторые модели эмбеддинга. Одним из ярких примеров является SentenceTransformers (*трансформеры предложений*), которая предоставляет универсальный набор инструментов для обработки естественного языка. В ее состав входят популярные модели эмбеддинга, использующие только кодировщик. Среди них можно выделить all-mpnet-base-v2 на базе модели MPNet, которая также использует архитектуру с кодировщиком. После дообучения на различных наборах данных, адаптированных под конкретные задачи, эта модель может успешно конкурировать с более крупными моделями эмбеддинга.

## Архитектура "кодер-декодер"

Архитектура трансформера в своем первоначальном виде представляет собой модель "кодер-декодер". Этот тип архитектуры используется в серии моделей T5.

Такая архитектура работает с текстом как на входе, так и на выходе. Ее стандартизованный интерфейс позволяет использовать одну и ту же модель и метод обучения для решения разнообразных задач. Кодировщик отвечает за обработку входных данных, а декодер формирует выходные результаты.

## Архитектура "только с декодером"

В последние годы большинство современных LLM базируются на моделях "только с декодером" (decoder-only). Популярность данного подхода началась с оригинальной модели GPT, разработанной OpenAI. Модели "только с декодером" показывают высокую эффективность в задачах обучения без примеров (zero-shot learning) и обучения с несколькими примерами (few-shot learning).

Модели с декодером подразделяются на каузальные (casual) и некаузальные (non-causal). Некаузальные модели рассматривают входную последовательность в обоих направлениях, обеспечивая двунаправленный анализ. Однако, несмотря на это, генерация ответа остается авторегрессионной, т. е. предсказания формируются последовательно, без заглядывания вперед.



Эта область все еще находится на этапе развития, однако уже имеются убедительные свидетельства (**https://oreil.ly/Sb7JS**), подтверждающие следующие выводы:

- модели только с декодером оказываются наиболее эффективными для задач обобщения как без примеров, так и с использованием нескольких примеров;
- архитектуры "кодер-декодер" демонстрируют наилучшие результаты в многозадачном дообучении.

Идеальным решением является комбинация этих подходов: сначала использовать авторегрессионное обучение, а затем, на этапе адаптации, провести предобучение модели с некаузальной конфигурацией, используя механизм повреждения спанов (span corruption).

В этом разделе мы разобрали, каким образом архитектурные магистрали могут быть классифицированы в зависимости от применения кодера и декодера в их архитектуре. Одним из видов архитектурной концепции, который набирает популярность за последнее время, является парадигма "смеси экспертов" (Mixture of Experts, MoE). Рассмотрим эту архитектуру более подробно.

## Смесь экспертов

Примечательно, что спустя семь лет после появления архитектуры трансформера ее реализация в современных языковых моделях практически не претерпела измене-

ний, несмотря на множество предложенных модификаций. Как выяснилось, оригинальная конструкция обладает высокой прочностью, и большинство предложенных модификаций не оказывали значительного влияния на производительность модели. Тем не менее некоторые компоненты трансформера все же претерпели изменения, особенно это касается позиционного кодирования, о котором уже упоминалось ранее в этой главе.

За последние несколько лет модели смеси экспертов (MoE model) привлекли значительное внимание и стали популярными. Среди известных примеров можно отметить GPT-4 от OpenAI (хотя официальных подтверждений этому пока нет), Switch от Google, DeepSeek V3 от DeepSeek и Mixtral от Mistral. В этом разделе мы рассмотрим, какие цели преследует создание этой архитектуры и каким образом она проявляет себя в практическом применении.

*В главе 1* отмечалось, что законы масштабирования (scaling laws) предполагают улучшение производительности языковых моделей при увеличении их размера и объема обучающих данных. Однако такой рост эффективности неизбежно влечет за собой повышенную потребность в вычислительных ресурсах как для обучения, так и для вывода. Это обстоятельство нежелательно, особенно при выводе, когда важным условием становится минимизация задержек. Возникает вопрос: возможно ли повысить эффективность модели без увеличения необходимых вычислительных мощностей?

Одним из подходов к реализации такого метода является условное вычисление, где каждый входной элемент, будь то токен или целая последовательность, взаимодействует только с определенным подмножеством модели. Это позволяет использовать лишь те параметры, которые наиболее эффективно справляются с обработкой данного ввода. Основой этого метода служит архитектура, состоящая из нескольких компонентов, называемых экспертами, причем для каждого ввода активируется только выбранное подмножество экспертов.

Типовая модель смеси экспертов показана на рис. 4.6.

Ключевым компонентом архитектуры MoE является *функция распределения нагрузки* (gating function). Эта функция отвечает за выбор эксперта, который наилучшим образом подходит для обработки конкретного входящего сигнала. Она реализуется в виде весов, присваиваемых каждому эксперту.

В трансформерах эксперты обычно интегрируются в модуль прямого распространения. Если таких экспертов восемь, то вместо одной будет использоваться восемь различных сетей прямого распространения. При этом в зависимости от выбранной стратегии маршрутизации (routing strategy) для обработки конкретного входа активизируется лишь небольшое подмножество этих сетей.

Стратегия маршрутизации отвечает за выбор количества и типа активированных экспертов. Наиболее распространенными подходами в данном случае являются:

- отбор токенов;
- отбор экспертов.

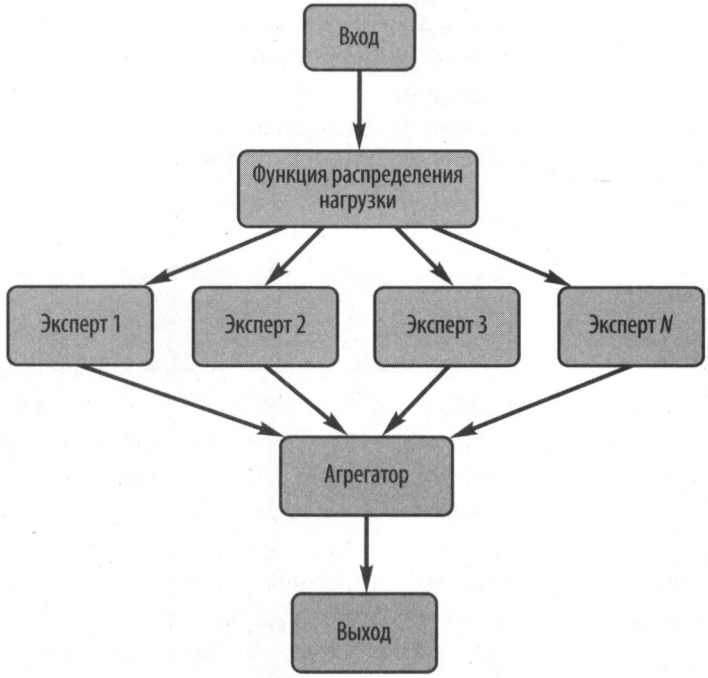



**Рис. 4.6.** Смесь экспертов

Стратегия отбора токенов предполагает, что для обработки каждого токена привлекается $k$ экспертов, где $k$ обычно представляет собой небольшое число, например ~2. Основным недостатком такого подхода является проблема балансировки нагрузки (load balancing). Если в некотором входном пакете основная часть токенов оказывается распределена между одними и теми же экспертами, выполнение вычислений может потребовать больше времени. Это связано с тем, что в такой ситуации невозможно полноценно воспользоваться преимуществами параллельной обработки, которую обеспечивают различные эксперты.

В рамках стратегии отбора экспертов каждый из них выбирает те токены, с обработкой которых он справляется наилучшим образом. Такой метод помогает решить задачу балансировки нагрузки, предоставляя возможность распределять одинаковое количество токенов между экспертами. Однако он также порождает проблему неэффективного распределения токенов, поскольку выбор каждого эксперта ограничивается лишь фиксированным количеством токенов внутри заданного пакета.

### Модель апсайклинга

На момент написания книги открытых моделей MoE было крайне мало. Но можно ли использовать существующую универсальную модель, основанную на архитектуре трансформера, для создания модели MoE? В работе Комацузаки и соавт. (Komatsuzaki et al., **https://oreil.ly/jJEot**) предложен эффективный метод апсайклинга (или повторного использования, upcycling method), который позво-

ляет внедрить компонент MoE в уже предварительно обученную модель. Суть подхода заключается в том, чтобы создать $N$ копий слоев прямого распространения — по одной копии для каждого эксперта. При этом исходные параметры этих слоев используются в качестве начальных параметров для новой модели MoE, обеспечивая плавный переход к обновленной архитектуре.

## Цели обучения

Теперь, после обсуждения архитектуры языковых моделей, перейдем к рассмотрению задач (целей), которые лежат в основе их обучения в процессе предобучения.

Как уже упоминалось ранее, языковые модели предобучаются с использованием метода самоконтроля. Огромный объем данных, необходимый для этого процесса, делает обучение с учителем слишком затратным, поскольку оно требует сбора пар "вход — выход" непосредственно от людей. Вместо этого применяется самоконтролируемый подход, при котором сами данные содержат целевые метки. Задача такого обучения заключается в освоении процесса, который служит прокси для изучения синтаксиса и семантики языка, а также навыков, связанных с рассуждением, выполнением математических операций, логическими манипуляциями и другими когнитивными задачами. В конечном итоге предполагается, что этот подход приблизит модели к обладанию более общим человеческим интеллектом. Но каким образом это достигается на практике?

Рассмотрим типичную задачу в области языкового моделирования — прогнозирование следующего слова в последовательности. В качестве примера возьмем фрагмент:

```
'Tammy jumped over the'
('Тэмми перепрыгнула через')
```

и предложим языковой модели определить следующий токен. Количество возможных вариантов зависит от размера словаря. Например, продолжением могут быть такие слова, как *hedge* (изгородь), *fence* (забор), *barbecue* (барбекю), *sandcastle* (песочный замок) и т. д. Однако есть продолжения, которые не соответствуют грамматическим нормам английского языка, например, глагол *is* (форма глагола *to be* — быть, используемая для третьего лица единственного числа в настоящем времени), предлог *of* (указывает на принадлежность, происхождение или часть чего-либо) или определенный артикль *the*. В процессе обучения на огромном количестве последовательностей модель постепенно понимает закономерности: маловероятно, что после артикля *the* будут идти *is* или *of*, если контекст не требует этого. Благодаря такому подходу предсказание следующего токена становится невероятно эффективным инструментом. Чтобы правильно определить следующий токен, модель должна овладеть все более сложными функциями и взаимосвязями, которые закладываются в ее архитектуре во время обучения.

Тем не менее остается открытым вопрос: достаточно ли этой парадигмы для достижения общего интеллекта? Цели самоконтролируемого обучения, применяемые при предобучении больших языковых моделей, можно условно разделить на три основные категории:

- полнотекстовое языковое моделирование (full language modeling, FLM);
- маскированное языковое моделирование (masked language modeling, MLM);
- префиксное языковое моделирование (prefix language modeling, PrefixLM).

Рассмотрим каждую из них более подробно.

## Полнотекстовое языковое моделирование

На рис. 4.7 представлена каноническая цель полнотекстового языкового моделирования (FLM).

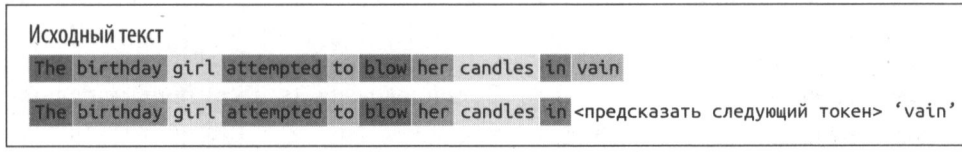



**Рис. 4.7.** Полнотекстовое языковое моделирование[2]

Здесь цель заключается в обучении модели предсказывать следующий токен в заданной последовательности. Эта стратегия обучения является самой простой и широко применяемой, активно используемой в GPT-4, а также во множестве моделей с открытым исходным кодом. Для каждого токена, который обрабатывает модель, вычисляются потери. Таким образом, каждый отдельный токен из обучающего набора, на основании которого языковая модель делает предсказания, становится источником сигнала обучения. Это делает процесс обучения чрезвычайно эффективным.

Проиллюстрируем этот подход на примере модели GPT Neo.

Предположим, необходимо продолжить предобучение модели GPT Neo с ее общедоступной контрольной точки, преследуя цель полнотекстового языкового моделирования. Рассмотрим текущую обучающую последовательность:

```
'Language models are ubiquitous'
```

*('Языковые модели повсеместно распространены')*

Запустим следующий код:

```
import torch
from transformers import AutoTokenizer, GPTNeoForCausalLM

tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-1.3B")
model = GPTNeoForCausalLM.from_pretrained("EleutherAI/gpt-neo-1.3B")
```

[2] Текст на рисунке: "Именинница пыталась задуть все свечи, но безуспешно". — *Прим. пер.*

```
input_ids = tokenizer("Language models are", return_tensors="pt")
gen_tokens = model.generate(**input_ids, max_new_tokens=1,
    output_scores=True, return_dict_in_generate=True)
output_scores = gen_tokens["scores"]
scores_tensor = output_scores[0]
sorted_indices = torch.argsort(scores_tensor[0], descending=True)[:20]

for index in sorted_indices:
    token_id = index
    token_name = tokenizer.decode([token_id.item()])
    token_score = scores_tensor[0][index].item()
    print(f"Token: {token_name}, Score: {token_score}")
```

Этот код выполняет токенизацию входного текста "Language models are" и передает его в модель для генерации следующего токена. Функция `generate()` предсказывает продолжение, основываясь на указанной последовательности, производя лишь один токен, поскольку параметр `max_new_tokens` установлен на значение `1`. После этого процесс генерации завершается. Оставшаяся часть кода предназначена для вывода списка из 20 токенов с наивысшими оценками перед применением функции softmax на последнем этапе.

Список 20 токенов с наивысшими оценками:

```
Output: Token: a, Score: -1.102203369140625
Token:  used, Score: -1.4315788745880127
Token:  the, Score: -1.7675716876983643
Token:  often, Score: -1.8415470123291016
Token:  an, Score: -2.4652323722839355
Token:  widely, Score: -2.657834053039551
Token:  not, Score: -2.6726579666137695
Token:  increasingly, Score: -2.7568516731262207
Token:  ubiquitous, Score: -2.8688106536865234
Token:  important, Score: -2.902832508087158
Token:  one, Score: -2.9083480834960938
Token:  defined, Score: -3.0815649032592773
Token:  being, Score: -3.2117576599121094
Token:  commonly, Score: -3.3110013008117676
Token:  very, Score: -3.317342758178711
Token:  typically, Score: -3.4478530883789062
Token:  complex, Score: -3.521362781524658
Token:  powerful, Score: -3.5338563919067383
Token:  language, Score: -3.550961971282959
Token:  pervasive, Score: -3.563507080078125
```

Каждое слово из топ-20 действительно выглядит логичным продолжением последовательности. При этом ключевая идея состоит в универсальном токене, который можно использовать для вычисления потерь и запуска процесса обратного распространения для обучения модели.

Для иллюстрации рассмотрим такой пример текстовой последовательности:

```
'I had 25 eggs. I gave away 12. I now have 13'
```

*('У меня было 25 яиц. Я израсходовал 12. Теперь у меня осталось 13 яиц')*

Запустим тот же код, но немного откорректируем его:

```
input_ids = tokenizer("'I had 25 eggs. I gave away 12. I now have",
    return_tensors="pt")
```

Результат покажет топ-20 возможных токенов для продолжения текста:

```
Token:  12, Score: -2.3242850303649902
Token:  25, Score: -2.5023117065429688
Token:  only, Score: -2.5456185340881348
Token:  a, Score: -2.5726099014282227
Token:  2, Score: -2.6731367111206055
Token:  15, Score: -2.6967623233795166
Token:  4, Score: -2.8040688037872314
Token:  3, Score: -2.839219570159912
Token:  14, Score: -2.847306728363037
Token:  11, Score: -2.8585362434387207
Token:  1, Score: -2.877161979675293
Token:  10, Score: -2.9321107864379883
Token:  6, Score: -2.982785224914551
Token:  18, Score: -3.0570476055145264
Token:  20, Score: -3.079172134399414
Token:  5, Score: -3.111320972442627
Token:  13, Score: -3.117424726486206
Token:  9, Score: -3.125835657119751
Token:  16, Score: -3.1476120948791504
Token:  7, Score: -3.1622045040130615
```

Правильный ответ имеет 17-й наивысший балл. В топ-10 наблюдается множество различных чисел, и это указывает, что модель угадывает ответы довольно случайно, что вполне ожидаемо для более компактной модели, такой как GPT Neo.

API OpenAI предоставляет параметр `logprobs`, позволяющий указать количество токенов с их вероятностями в формате логарифма, которые нужно вернуть. На момент написания книги доступно лишь 20 наиболее вероятных токенов для анализа.

Возвращенные токены систематизированы в порядке убывания их логарифмической вероятности:

```
import openai
openai.api_key = <Insert your OpenAI key>

openai.Completion.create(
    model="gpt-4o",
    prompt="I had 25 eggs. I gave away 12. I now have ",
    max_tokens=1,
    temperature=0,
    logprobs=10
)
```

Данный код активирует устаревшую модель gpt-4o, запросив у нее сгенерировать не более одного токена. На выходе получаем:

```
"top_logprobs": [
        {
            "\n": -0.08367541,
            " 13": -2.8566456,
            "___": -4.579212,
            "____": -4.978668,
            "_______": -6.220278
            ...
        }
```

Модель с большой вероятностью считает, что правильным ответом будет 13, и это предположение вполне оправдано. Остальные наиболее вероятные токены связаны с форматированием результата.



При выводе не всегда требуется генерировать токен с наивысшим баллом. Существуют различные *стратегии декодирования* (decoding strategies), которые помогают создавать более разнообразный текст. Эти стратегии будут подробно рассмотрены в *главе 5*.

### Упражнение

Попросите модель gpt-4o решить несколько кроссвордов из сборника Boatload Puzzles Crossword (**https://oreil.ly/HvK_O**). Для достижения наилучших результатов может потребоваться несколько итераций с изменениями в запросе. Для начала можно использовать следующую фразу: "Solve this crossword and answer in one word. The clue is <X> and it is a <Y> letter word. The answer is" ("Разгадай этот кроссворд и ответь одним словом. Подсказка — это <X>, и это буквенное

слово <Y>. Ответ:"). Установите параметр `max_tokens=3`, чтобы учесть форматирование токенов. Внимательно изучите выходные данные `logprob`. Насколько близки они к правильному или неправильному ответу? Сколько подсказок модель интерпретирует верно?

## Префиксное языковое моделирование

Префиксная языковая модель (PrefixLM) аналогична полнотекстовой модели (FLM) и отличается лишь тем, что последняя является полностью каузальной, и это подразумевает, что в системе письма слева направо, как в английском языке, токены не могут ссылаться на токены, находящиеся справа (т. е. на будущее). В PrefixLM часть текстовой последовательности, именуемая префиксом, может быть использована для предсказания будущих токенов, находящихся в этом же префиксе. Таким образом, префиксная часть является некаузальной. Для обучения префиксной языковой модели выбирается случайная длина префикса, и вычисление потерь осуществляется только по токенам, расположенным в суффиксе.

## Маскированное языковое моделирование

На рис. 4.8 показана каноническая цель маскированного языкового моделирования (MLM).

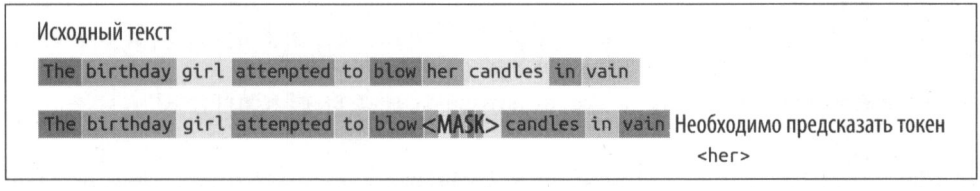



**Рис. 4.8.** Маскированное языковое моделирование в BERT

В отличие от традиционных моделей, которые фокусируются на предсказании следующего токена, в MLM-задачах модель должна восстановить оригинальные значения скрытых токенов в последовательности. Наиболее простая реализация MLM, как в модели BERT, включает случайное маскирование 15% токенов с использованием специального токена маски, после чего модель предсказывает исходные значения этих токенов.

Авторы модели T5 внесли изменения в исходную цель MLM. В их модификации 15% токенов случайным образом удаляются из последовательности. Удаленные токены заменяются одним уникальным специальным токеном, который называется *сигнальным токеном* (sentinel token). Затем модель получает задачу предсказать и восстановить удаленные токены, используя сигнальные токены в качестве ограничений.

Рассмотрим, к примеру, такую последовательность:

`Tempura has always been a source of conflict in the family due to unexplained reasons`

*(По необъяснимым причинам Темпура всегда являлась источником конфликтов в семье)*

Предположим, что мы исключаем слова *has* (вызывала), *always* (всегда), *conflict* (споры) и *of* (в). Теперь последовательность будет выглядеть так:

```
Tempura <S1> been a source <S2> in the family due to unexplained reasons
```

*(По необъяснимым причинам Темпура <S1> являлась источником <S2> в семье)*

С учетом того, что S1 и S2 являются сигнальными токенами, от модели ожидается такой выход:

```
<S1> has always <S2> of conflict <E>
```

*(<S1> всегда <S2> конфликтов <E>)*

Конец выходной последовательности отмечается специальным токеном, сигнализирующим о ее завершении.

Использование генерации только отброшенных токенов, а не всей последовательности, повышает вычислительную эффективность и сокращает время обучения. Важно отметить, что в отличие от полнотекстового моделирования потери рассчитываются только для ограниченного набора токенов (маскированных), и это значительно сокращает вычислительные затраты на обработку входной последовательности. Рассмотрим этот пример на Hugging Face:

```
from transformers import T5Tokenizer, T5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("t5-3b")
model = T5ForConditionalGeneration.from_pretrained("t5-3b")

input_ids = tokenizer("Tempura <extra_id_0> been a source <extra_id_1> in the family
due to unexplained reasons", return_tensors="pt").input_ids
targets = tokenizer("<extra_id_0> has always <extra_id_1> of conflict

<extra_id_2>", return_tensors="pt").input_ids
loss = model(input_ids=input_ids, labels=labels).loss
```

Для формирования целей можно использовать простую шаблонную функцию.

### Упражнение

Поэкспериментируйте с различными стратегиями маскирования.

- Попробуйте регулировать степень маскирования. Что произойдет при маскировании 30%, 50% токенов?
- Поменяйте стратегию маскирования. Возможно, вам удастся найти более эффективный подход, чем случайное маскирование. Какие эвристики могут помочь в маскировке токенов таким образом, чтобы это способствовало более эффективному обучению?

В целом можно рассматривать MLM как *шумоподавляющий автоэнкодер* (автокодер, автокодировщик — autoencoder, AE). В этом случае входные данные искажаются путем добавления шума (например, при помощи маскирования или удаления

токенов), после чего модель обучается восстанавливать исходные данные. BART поднимает эту концепцию на новый уровень, применяя пять различных механизмов повреждения спанов.

*Маскирование случайных токенов (random token masking)*

На рис. 4.9 показаны этапы устранения искажений и шумов.

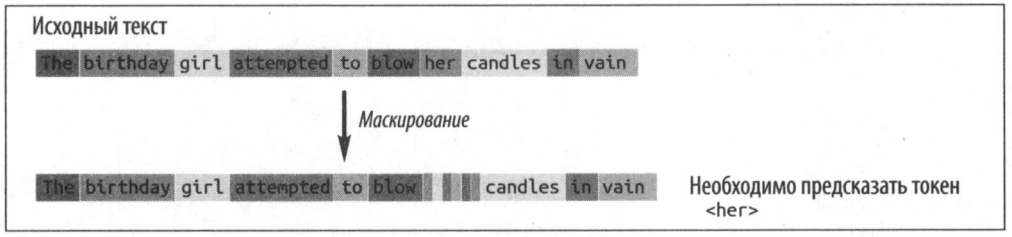



**Рис. 4.9.** Маскирование случайных токенов в BART

*Удаление случайных токенов (random token deletion)*

Модель должна предсказать позиции в тексте, из которых были исключены токены. На рис. 4.10 показаны этапы удаления искажений и шумов.

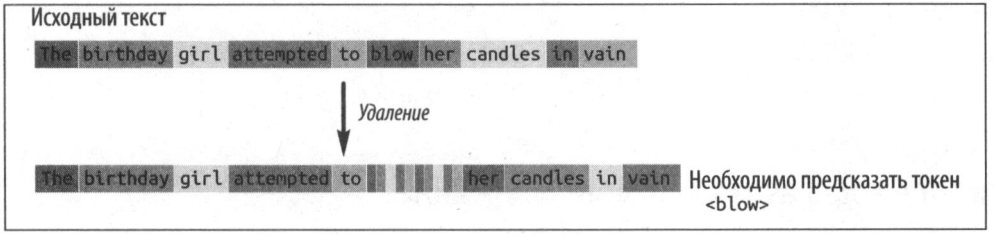



**Рис. 4.10.** Удаление случайных токенов в BART

*Маскирование спанов (span masking)*

В тексте выделяются текстовые спаны, длина которых определяется распределением Пуассона. Это подразумевает возможность появления спанов нулевой длины (или пустых спанов — zero-length spans). Удаленные спаны замещаются одним токеном маскировки, что требует от модели также предсказания количества удаленных токенов. На рис. 4.11 показаны этапы устранения искажений и шумов.

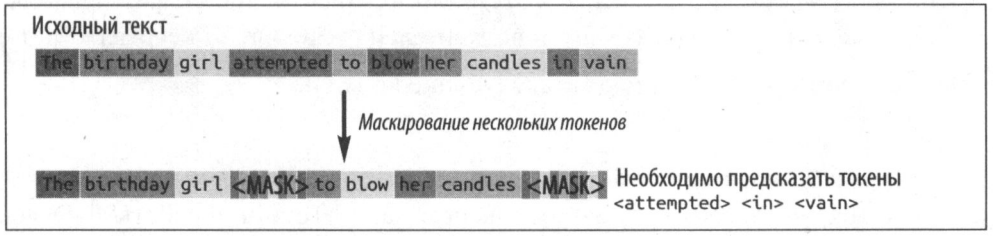



**Рис. 4.11.** Маскирование спанов в BART

*Перетасовка документов (document shuffling)*

Предложения во входном документе перетасовываются, и модель обучается располагать их в правильном порядке. На рис. 4.12 показаны этапы устранения искажений и шумов.




**Рис. 4.12.** Перетасовка документов в BART

*Поворот документа (document rotation)*

Документ поворачивается так, что начинается с произвольного токена. Модель обучается определять правильное начало документа. На рис. 4.13 показаны этапы устранения искажений и шумов.




**Рис. 4.13.** Поворот документа в BART

## Наиболее эффективные цели обучения

Исследования показали, что модели, обученные методом полнотекстового языкового моделирования (FLM), демонстрируют превосходные результаты в генерации текста, в то время как модели, обученные методом маскированного языкового моделирования (MLM), выгодно выделяются в задачах классификации. Однако использование разных языковых моделей для различных сценариев нецелесообразно. Тенденция к консолидации языковых моделей усиливается, что подтверждается появлением UL2 (**https://oreil.ly/xJc3U**) — парадигмы, которая объединяет сильные стороны различных подходов к обучению в единой модели.

UL2 имитирует эффекты FLM, MLM и PrefixLM в рамках единой парадигмы, называемой *смесью денойзеров* (смесь шумоподавителей — mixture of denoisers, MoD).

В качестве денойзеров используются следующие подходы.

*R-Denoiser*

Этот подход аналогичен механизму повреждения спанов в T5, где спаны длиной от 2 до 5 токенов заменяются одним токеном маски. Принцип работы R-Denoiser показан на рис. 4.14.

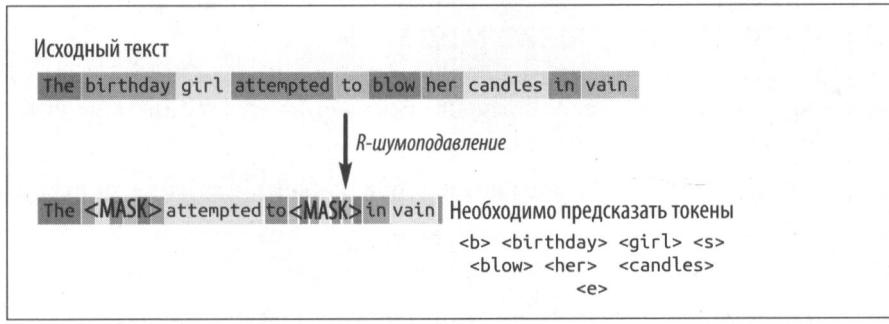



**Рис. 4.14.** UL2 R-Denoiser

*S-Denoiser*

Аналогично префиксной LM, текст разделяется на префикс и суффикс. Суффикс маскируется, а префикс получает доступ к двунаправленному контексту. Принцип работы S-Denoiser показан на рис. 4.15.

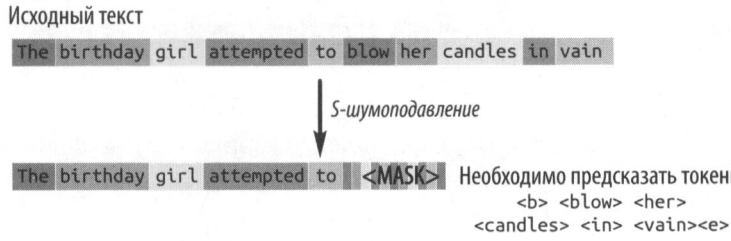



**Рис. 4.15.** UL2 S-Denoiser

*X-Denoiser*

Этот денойзер характеризуется высокой степенью шумоподавления, при которой маскируется значительная часть текста (более 50%). Принцип работы X-Denoiser показан на рис. 4.16.

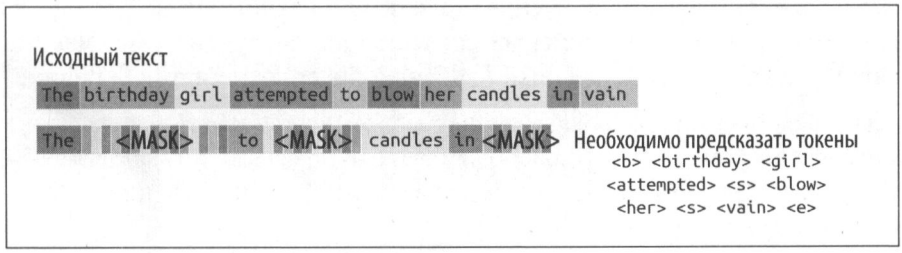



**Рис. 4.16.** UL2 X-Denoiser

## Предварительное обучение моделей

Таким образом, мы подробно рассмотрели компоненты языковой модели и теперь обратимся к процессу ее предварительного обучения с нуля.

Современные языковые модели обучаются моделировать два типа понятий на основе одной модели:

- язык как средство передачи фактов, мнений и чувств;
- основные явления, которые формируют структуру текста на языке.

Для многих прикладных областей гораздо более важным является обучение модели второму аспекту. Хотя создание языковой модели, свободно владеющей языком, было бы значимым достижением, предпочтительнее, чтобы она совершенствовалась в таких областях, как наука, право, а также в таких навыках, как логическое мышление и арифметика.

Эти концепции и умения реализуются посредством языков (например, английского), которые прежде всего служат для коммуникации в социальной среде. По своей природе человеческие языки неоднозначны, содержат множество синонимов и в целом являются не самыми эффективными инструментами для передачи базовых концепций.

В связи с этим возникает вопрос: насколько человеческий язык является оптимальным инструментом для обучения языковых моделей основам и концепциям? Возможно ли отделить процесс моделирования языка от моделирования фундаментальных понятий, которые выражаются с помощью языка?

Чтобы проверить эту теорию, рассмотрим практический пример обучения модели LLM играть в шахматы с нуля. Для этого нам понадобятся следующие компоненты (более подробно они описаны в *главе 2*):

- набор предобучающих данных;
- словарь и алгоритм токенизации;
- архитектура модели;
- цель обучения.

В качестве архитектуры для обучения модели шахматного языка можно выбрать трансформер, специализирующийся на предсказании следующего токена. Фактически эта архитектура является ведущей парадигмой в современном обучении.

В качестве предобучающего набора данных можно использовать набор данных шахматных игр Chess Game Dataset (Lichess), доступный по ссылке **https://oreil.ly/XmWvv**, который насчитывает миллиарды шахматных партий. Для обучения выберем подмножество из 20 млн партий.

Данный набор данных представлен в формате PGN (portable game notation — портативная игровая нотация), служащем для компактного представления последовательности шахматных ходов.

В завершение необходимо выбрать словарь модели. Учитывая, что модель предназначена исключительно для обучения игре в шахматы, нет необходимости в большом англоязычном словаре. В данном случае для присвоения токенов конкретным шахматным понятиям будет достаточно воспользоваться нотацией PGN.

Рассмотрим пример шахматной партии в формате PGN, доступной на сайте **pgnmentor.com** (**https://oreil.ly/H3yOs**).

```
1. e4 c5 2. Nf3 a6 3. d3 g6 4. g3 Bg7 5. Bg2 b5 6. O-O Bb7 7. c3 e5 8. a3 Ne7
9. b4 d6 10. Nbd2 O-O 11. Nb3 Nd7 12. Be3 Rc8 13. Rc1 h6 14. Nfd2 f5 15. f4
Kh7 16. Qe2 cxb4 17. axb4 exf4 18. Bxf4 Rxc3 19. Rxc3 Bxc3 20. Bxd6 Qb6+ 21.
Bc5 Nxc5 22. bxc5 Qe6 23. d4 Rd8 24. Qd3 Bxd2 25. Nxd2 fxe4 26. Nxe4 Nf5 27.
d5 Qe5 28. g4 Ne7 29. Rf7+ Kg8 30. Qf1 Nxd5 31. Rxb7 Qd4+ 32. Kh1 Rf8 33. Qg1
Ne3 34. Re7 a5 35. c6 a4 36. Qxe3 Qxe3 37. Nf6+ Rxf6 38. Rxe3 Rd6 39. h4 Rd1+
40. Kh2 b4 41. c7 1-0
```

Строкам шахматной доски присвоены буквы от a до h, а столбцам — числа от 1 до 8. За исключением пешек, каждой фигуре соответствует заглавная буква: N для коня, R для ладьи, B для слона, Q для ферзя и K для короля.

Знак +, добавленный к ходу, обозначает шах, знак % — шах и мат, а 0-0 — рокировку. Если вы не знаете правила игры, можно ознакомиться с основами по ссылке **https://oreil.ly/EbcfQ**.

Исходя из этой нотации, в словарь могут войти:

- отдельный токен для каждого квадрата на доске, всего 64 (a1, a2, a3, ..., h6, h7, h8);
- отдельный токен для каждой фигуры (N, B, R, K, Q);
- токены для номеров ходов (1., 2., 3. и т. д.);
- токены для специальных ходов (+ для шаха, x для взятия фигуры и т. д.).

Теперь все готово для обучения языковой модели с нуля на основе этого набора шахматных данных. Для этого будет использоваться наш уникальный словарь, специфичный для данной области. Модель будет обучаться исключительно на нотации PGN, без использования текстов на человеческом языке, которые также присутствуют в наборе данных. Код и настройки для обучения этой модели можно найти в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).

После обучения модели в течение трех эпох можно оценить ее способность играть в шахматы. Очевидно, модель успешно освоила правила игры, и ей не требуется четкое изложение правил на естественном языке. Более того, иногда модель может даже обыгрывать людей и выполнять такие сложные ходы, как рокировка.

### Упражнение

Хотя модель, обученная по эталонной реализации, уже демонстрирует впечатляющие результаты, побеждая в шахматных партиях и иногда обыгрывая игроков, ее можно сделать еще лучше. Для этого нужно увеличить размер модели,

расширить набор данных и улучшить его качество. Попробуйте внести изменения в модель по каждому из этих направлений и проследите за тем, как будут расти ее шахматные способности.

Важно отметить, что данная модель способна изучать концепции, используя специфический для домена язык (PGN). Но как будет происходить процесс обучения на естественном языке?

Для проведения эксперимента можно воспользоваться тем же набором данных, который применялся для предварительного обучения модели шахматного языка. Затем этот набор данных нужно пропустить через LLM, чтобы преобразовать каждый ход в формате PGN в предложение на английском языке. Пример партии будет выглядеть следующим образом:

*белые делают ход пешкой на поле e4*

*черные делают ход слоном на поле g7*

и т. д. Продолжим эксперимент, обучив новую языковую модель на том же количестве партий, что и предыдущую, но на этот раз используя англоязычный набор данных. Для этой модели мы создадим стандартный английский словарь, обучив токенизатор на том же наборе данных.

Как это соотносится с шахматной языковой моделью, обученной на наборе данных PGN? Несмотря на то что новая модель была обучена на том же количестве партий, она демонстрирует худшие результаты и, кажется, еще не полностью освоила правила игры.

Это свидетельствует о том, что для обучения моделей навыкам и концепциям естественный язык не всегда является оптимальным инструментом, а специализированные языки и нотации в конкретной области оказываются более эффективными.

Таким образом, языковый дизайн является важным навыком, который позволяет создавать специализированные языки для освоения концепций и навыков. В прикладных областях можно использовать уже существующие специализированные языки или разработать новый язык самостоятельно.

## Резюме

В этой главе мы подробно рассмотрели ключевые компоненты архитектуры трансформера, включая механизм внутреннего внимания, сети прямого распространения, позиционное кодирование и нормализацию уровней. Мы также изучили различные версии и конфигурации, такие как модели только с кодером, кодер-декодерные модели, модели только с декодером и модели смеси экспертов (MoE). Кроме того, мы изучили методы интеграции знаний языковых моделей для обучения собственной модели с нуля, а также способы создания языков, оптимизированных для конкретных областей, что способствует более эффективному процессу обучения.

# Часть II

# Применение LLM

В этой части книги мы рассмотрим методы применения и настройки предобученных больших языковых моделей для решения разнообразных задач, описанных в *главе 1*. Чтобы максимально эффективно освоить представленные материалы, настоятельно советуем проводить практические эксперименты с каждым из предложенных методов, а также изучить руководства, доступные в репозитории на GitHub, который прилагается к данной книге.

ГЛАВА 5

# Адаптация LLM к конкретным сценариям применения

В этой главе мы детально разберем коммерческие языковые модели и рассмотрим рекомендации по выбору оптимальной модели для решения конкретных задач. Мы обсудим процесс загрузки моделей различных размеров и выполнения на них логических операций, а также детализируем стратегии генерации текста, включая способы настройки параметров расшифровки. Кроме того, мы рассмотрим методы интерпретации результатов и промежуточных данных моделей с использованием таких инструментов, как LIT-NLP, повышающих уровень их интерпретируемости (interpretability tool).

## Навигация по ландшафту LLM

В последнее время на рынке регулярно появляются новые большие языковые модели, каждая из которых претендует на уникальность и современность. Однако не всегда можно увидеть значительные различия между ними, поэтому нет необходимости уделять много времени отслеживанию новых релизов. В GitHub-репозитории книги (**https://oreil.ly/llm-playbooks**) ведется каталогизация основных релизов, однако гарантировать его полноту достаточно сложно.

Тем не менее важно иметь общее представление о различных категориях поставщиков LLM, характеристиках доступных моделей, а также о том, как эти аспекты связаны с вопросами авторского права и лицензирования. Рассмотрим текущий ландшафт LLM с этих позиций и изучим доступные возможности.

### Провайдеры больших языковых моделей

Провайдеров больших языковых моделей (LLM) можно условно разделить на несколько категорий.

*Компании-разработчики проприетарных LLM*

К данной категории относятся такие компании, как OpenAI (модель GPT, **https://oreil.ly/r-lb1**), Google (проект Gemini, **https://oreil.ly/KF9Kh**), Anthropic (модель Claude, **https://oreil.ly/T5Wvo**), Cohere (**https://oreil.ly/PiKxN**), AI21 (**https://oreil.ly/Y8T3q**) и др. Они занимаются созданием и обучением проприетарных (от англ. *proprietary* — собственный) моделей, предоставляя к ним доступ через API, реализованные как услуга (например, LLM-as-a-service — LLM как услуга). Многие из указанных компаний сотрудничают с крупными облачными провайдерами, которые упрощают доступ к моделям через полностью управляемые службы (fully managed service). Среди таких решений можно выделить Amazon Bedrock (**https://oreil.ly/FVqRj**) и SageMaker JumpStart (**https://oreil.ly/e0a59**) от Amazon, Vertex AI от Google (**https://oreil.ly/mURoC**), Azure OpenAI от Microsoft (**https://oreil.ly/Ag1r5**).

*Компании, предоставляющие доступ к открытым моделям LLM*

В эту категорию входят организации, которые делают веса больших языковых моделей общедоступными, одновременно зарабатывая на предоставлении услуг по их развертыванию. Одним из примеров таких компаний является активно развивающаяся Together AI (**https://oreil.ly/urcAf**). Также сюда входят компании, основной бизнес которых напрямую связан с ростом использования LLM, такие как Cerebras (**https://oreil.ly/2cVYY**). И, наконец, исследовательские лаборатории, которые начали работать с LLM с момента появления архитектуры трансформера, включая Microsoft, Google, Meta, Salesforce и др. Следует отметить, что некоторые компании (в частности, Google) выпускают как проприетарные модели, так и версии с открытым исходным кодом.

*Самоорганизующиеся команды, предоставляющие открытый исходный код, и общественные исследовательские организации*

В эту категорию входят новаторская общественная исследовательская организация Eleuther AI (**https://oreil.ly/ZSlbG**) и Big Science (**https://oreil.ly/_NlUD**). Деятельность этих организаций поддерживается грантами, которые направляются на создание необходимой вычислительной инфраструктуры.

*Научные и государственные учреждения*

В связи со значительными капитальными вложениями, число LLM, разработанных в рамках академических инициатив, пока остается ограниченным. Среди примеров моделей, созданных государственными и научно-исследовательскими организациями, можно выделить модель Falcon (**https://oreil.ly/vdhsL**), разработанную в государственном Институте технологических инноваций Абу-Даби (Abu Dhabi Technology Innovation Institute, **https://oreil.ly/aMwO2**), и модель GLM, представленную Университетом Цинхуа (Tsinghua University, **https://oreil.ly/K0_zX**).

В табл. 5.1 приведен обзор основных участников рынка LLM, их тип организации, а также разработанные ими предобученные модели.

**Таблица 5.1.** Провайдеры LLM

| **Название** | **Тип организации** | **Разработанные предобученные модели** |
|---|---|---|
| Google | Коммерческая компания | BERT, MobileBERT, T5, FLAN-T5, ByT5, Canine, UL2, Flan-UL2, Pegasus PaLM, PaLMV2, ELECTRA, Tapas, Switch |
| Microsoft | Коммерческая компания | DeBERTa, DialoGPT, BioGPT, MPNet |
| OpenAI | Коммерческая компания | GPT-2, GPT-3, GPT-3.5, GPT-4 |
| Amazon | Коммерческая компания | Titan |
| Anthropic | Коммерческая компания | Claude, Claude-2 |
| Cohere | Коммерческая компания | Cohere Command, Cohere Base |
| Meta | Коммерческая компания | RoBERTa, Llama, Llama 2, BART, OPT, Galactica |
| Salesforce | Коммерческая компания | CTRL, XGen, EinsteinGPT |
| MosaicML | Коммерческая компания (приобретена компанией Databricks) | MPT |
| Cerebras | Коммерческая компания | Dolly-V1, Dolly-V2 |
| Stability AI | Коммерческая компания | StableLM |
| Together AI | Коммерческая компания | RedPajama |
| Ontocord AI | Некоммерческая организация | MDEL |
| Eleuther AI | Некоммерческая организация | Pythia, GPT Neo, GPT-NeoX, GPT-J |
| Big Science | Некоммерческая организация | BLOOM |
| Tsinghua University | Научная организация | GLM |
| Technology Innovation Institute | Научная организация | Falcon |
| UC Berkeley | Научная организация | OpenLLaMA |
| Adept AI | Коммерческая компания | Persimmon |
| Mistral AI | Коммерческая компания | Mistral |
| AI21 Labs | Коммерческая компания | Jurassic |
| X.AI | Коммерческая компания | Grok |

## Разновидности моделей

Многие современные модели обычно представлены в нескольких версиях. Чаще всего компании выпускают модификации одной модели с различными размерами. Например, модель Llama 2 доступна в трех вариантах: 7B, 13B и 70B, где числа обозначают количество параметров в модели.

В настоящее время провайдеры LLM активно работают над улучшением предобученных моделей, чтобы сделать их более адаптированными к потребностям пользователей. Одним из ключевых процессов модернизации является *аугментация* (augmentation — улучшение, усиление) — дообучение модели с использованием специальных методов, зачастую подкрепленных человеческим контролем. Этот процесс требует значительных ресурсов, включая аннотирование данных вручную, что может обходиться в миллионы долларов. Тем не менее основой остаются базовые версии моделей, прошедшие лишь начальный этап предобучения без дополнительной аугментации.

Далее рассмотрим наиболее популярные методы аугментации.

### Инструктивные модели

Инструктивные модели (instruct-model), или модели, обученные следовать инструкциям (instruction-tuned model), предназначены для выполнения задач, сформулированных на естественном языке. В отличие от базовых моделей, которые обладают значительным потенциалом, их применение чаще всего требует тщательной настройки запросов методом проб и ошибок, что порой может быть довольно трудоемким процессом. Базовые модели сравнивают с бунтарским подростком — их поведение не всегда предсказуемо, а взаимодействие напрямую зависит от точности и стабильности подобранных инструкций. Это связано с тем, что такие модели изначально обучаются задачам вроде шумоподавления или предсказания следующего слова, что неполностью соответствует реальным запросам пользователей. Однако благодаря специальной адаптации базовой модели на выполнение команд итоговые инструктивные модели демонстрируют значительно бóльшую эффективность в обработке человеческих запросов и становятся более полезными инструментами для решения практических задач.

Типичный обучающий набор данных для инструктивных задач включает разнообразные задания, оформленные на естественном языке, а также соответствующие пары "ввод — вывод". *В главе 6* мы рассмотрим различные подходы к созданию таких наборов данных и разберем способы обучения модели эффективно следовать инструкциям.

Обратимся к популярному инструктивному набору данных FLAN (**https://oreil.ly/YJ_Xr**) и рассмотрим задачу определения тональности текста.

> *Prompt*: "What is the sentiment of the following review? The pizza was ok but the service was terrible. I stopped in for a quick lunch and got the slice special but it ended up taking an hour after waiting several minutes for someone at the front counter and then again for

the slices. The place was empty other than myself, yet I couldn't get any help/service. OPTIONS: - negative - positive"

FLAN: "Negative"

---

*Промпт*: "Определи тональность следующего отзыва. Пицца была вкусной, но обслуживание оказалось ужасным. Я зашел перекусить и заказал специальный кусок, но в итоге пришлось ждать целый час — сначала в очереди у стойки, затем снова за самим заказом. Заведению не хватало посетителей, кроме меня, но помощь так и не была оказана. ВАРИАНТЫ ОТВЕТА: - отрицательный - положительный"

FLAN: "Отрицательный".

В данном примере входные данные представлены инструкцией "Определи тональность следующего отзыва" в той форме, как ее обычно выражают люди в естественной речи. Вместе с инструкцией идут вводные данные — сам текст отзыва и выходные данные — результат, сгенерированный моделью, либо аннотированный человеком.

Процесс обучения следованию инструкциям изображен на рис. 5.1.

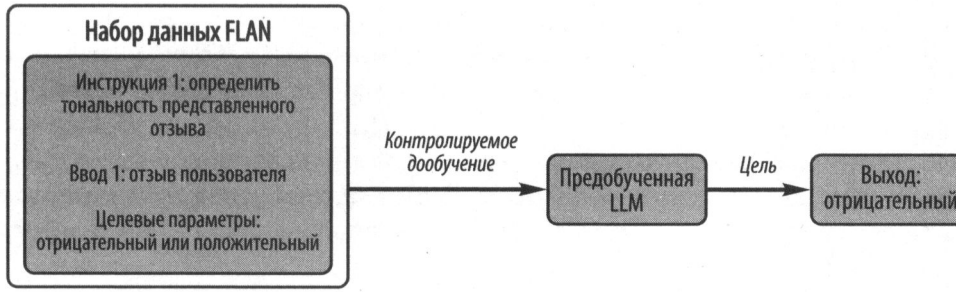



**Рис. 5.1.** Процесс обучения следования инструкциям

Обучение следовать инструкциям является одним из ключевых методов в рамках контролируемого дообучения (supervised fine-tuning, SFT). Этот подход не только улучшает способность модели выполнять задачи пользователей, но и позволяет снизить вероятность вредоносного поведения, обучая ее на специализированных наборах данных по безопасности (safety dataset). Такие данные помогают синхронизировать выходные результаты модели с ценностями и предпочтениями ее разработчиков.

Помимо стандартного SFT, существуют более сложные методы улучшения модели. Например, к ним относятся стратегии, использующие обратную связь, такие как обучение с подкреплением на основе отзывов людей (reinforcement learning from human feedback, RLHF) или обучение с подкреплением на основе обратной связи от ИИ (reinforcement learning from AI feedback, RLAIF).

В процессе обучения RLHF аннотаторы оценивают или ранжируют возможные результаты модели, основываясь на критериях полезности и безвредности. Эти данные аннотаций впоследствии служат для постепенного совершенствования модели вознаграждения (reward model). В результате LLM становится более управляемой,

проявляя, к примеру, способность игнорировать неподходящие запросы пользователей. Процесс обучения RLHF представлен на рис. 5.2.




**Рис. 5.2.** Обучение с подкреплением на основе отзывов людей

Мы подробно рассмотрим RLHF и другие подходы к согласованию в *главе 8*.

Одним из альтернативных подходов к обучению на согласование, вместо использования обратной связи от людей, является применение больших языковых моделей. Эти модели способны принимать решения о результатах, основываясь на заранее определенных принципах, таких как избегание расизма, грубости и других негативных явлений. Этот метод, предложенный Anthropic, называется обучением с подкреплением на основе обратной связи от ИИ (RLAIF). В рамках RLAIF человек определяет лишь набор желаемых принципов и ценностей, известный как конституционный ИИ (Constitutional AI, CAI, **https://oreil.ly/d8FeW**), а ответственность за проверку соответствия выводов этим принципам возлагается на саму языковую модель.

Инструктивные модели обычно имеют в названии суффикс "instruct", например RedPajama-Instruct.

### Побочные эффекты обучения следованию инструкциям

Стоит ли всегда использовать версию модели, обученной следовать инструкциям, вместо базовой модели для выполнения своих задач? В большинстве случаев такой подход будет оправдан. Однако следует учитывать, что дообучение базовой модели может привести к некоторым регрессиям, в результате чего модель теряет доступ к некоторым функциям и возможностям, которые были доступны в исходной версии.

В своем исследовании Чанг и соавт. (Chung et al., **https://oreil.ly/-p0R4**) показали, что обучение модели следованию инструкциям с использованием набора данных FLAN приводит к снижению качества выполнения задач, связанных с цепочкой рассуждений (CoT). Эти задачи являются ключевыми для процессов логического анализа. Однако исследователи также выяснили, что включение данных CoT в обучающие инструктивные наборы значительно улучшает способности модели к рассуждениям по сравнению с базовой версией.

Побочные эффекты обучения моделей следованию инструкциям еще не до конца изучены. Поэтому перед тем, как использовать базовую модель, разумно протестировать ее и убедиться, что ее функциональные возможности не были утрачены в процессе обучения.

Аналогично согласованные модели настраиваются для ответа на запросы пользователей в соответствии с принципами, ценностями и этикой провайдера LLM. Эти ценности могут не совпадать с теми, которые разделяет конкретный пользователь или организация.

Чтобы преодолеть эти различия, можно самостоятельно обучить базовую модель следовать инструкциям и согласованию. В следующих трех главах мы подробно рассмотрим, как это сделать, и оценим, в каких случаях такое обучение будет целесообразным.

### Чат-модели

Чат-модели (chat-model) — это модели, обученные следовать инструкциям и предназначенные для ведения многозадачных диалогов. К ним относятся такие популярные модели, как ChatGPT, Llama 2-Chat, MPT-Chat, OpenAssistant.

### Модели с длинным контекстом

Как уже отмечалось в *главе 1*, большие языковые модели на основе трансформера имеют ограничение по длине контекста. Этот параметр обычно обозначает количество входных и выходных токенов, которые модель может обработать за одну операцию. Типичная длина контекста современных LLM составляет от 8000 до 128 000 токенов, а некоторые версии Gemini способны обрабатывать более миллиона токенов. Однако существуют модели, созданные специально для работы с длинными контекстами (long-context model). Например, стандартная версия GPT-3.5 может обрабатывать контекст размером 4K, но также доступна версия с увеличенным лимитом в 16K. Модель MPT (**https://oreil.ly/wKqdL**), обученная на контексте длиной 65K, может обрабатывать еще бóльшие объемы информации на этапе вывода.

#### За длинный контекст приходится платить

Актуальность вопроса о взаимосвязи между длиной контекста и производительностью моделей с расширенными контекстными окнами продолжает сохраняться. Согласно исследованиям (**https://oreil.ly/PSD_k**), увеличение размера контекстного окна не гарантирует стабильного уровня производительности, поскольку модели могут "забывать" часть информации, находящейся в середине окна. Это объясняется спецификой обучающих данных: наиболее значимый для предсказания последующих токенов контекст зачастую концентрируется либо в начале, либо в конце документа. Собственные эксперименты показали, что для большинства моделей оптимальной является длина контекста около 8000 токенов. При превышении этого предела наблюдается снижение качества их работы. Важно также учитывать, что наполнение всего контекстного окна исключительно инст-

рукциями не решает проблему. Ограниченная способность моделей эффективно обрабатывать большое количество инструкций в одном промпте приводит к заметному ухудшению производительности при превышении данного лимита.

Тем не менее модели, способные работать с длинным контекстом, представляют собой одну из наиболее стремительно развивающихся областей в сфере больших языковых моделей. Такие длинноконтекстные модели, как Claude и Gemini, демонстрируют впечатляющие успехи в сохранении высокой производительности даже в условиях повышенной сложности.

Для измерения эффективности моделей в длинных контекстах используются разнообразные подходы, включая тесты типа "игла в стоге сена" (haystack tests, **https://oreil.ly/aop_Q**). *В главе 12* будут подробно рассмотрены недостатки этих методов оценки, а также предложены более сложные и всесторонние схемы, которые помогут объективно оценить работу моделей.

### Модели, адаптированные для конкретных областей и задач

Провайдеры LLM могут дообучать свои модели для выполнения узкоспециализированных задач, таких как суммаризация (summarization) или анализ настроений в финансовой сфере (financial sentiment analysis). Кроме того, они создают упрощенные версии моделей, которые настраиваются на основе результатов более крупных моделей для конкретной задачи. Примерами таких подходов являются модель FinBERT (**https://oreil.ly/uKUAp**), дообученная на наборе данных для анализа финансовых настроений, и Universal-NER (**https://oreil.ly/8A0pn**), оптимизированная на наборах данных для распознавания именованных сущностей.

## Большие языковые модели с открытым исходным кодом

Термин "открытый исходный код" (open source) часто используется как общее понятие, когда речь идет о моделях, доступ к которым открыт для всех. В нашем контексте мы будем понимать под открытым исходным кодом следующее:

> "Программные артефакты, распространяемые по лицензии, которая предоставляет пользователям возможность *изучать*, *использовать*, *модифицировать* и *распространять* их *без ограничений* и в *любых целях*".

Более формальное и полное определение программного обеспечения с открытым исходным кодом можно найти в официальном документе Open Source Initiative (**https://oreil.ly/7cezH**).

Чтобы LLM считалась полностью открытой, необходимо обнародовать все данные из следующего перечня.

*Веса модели (model weights)*

> Веса включают в себя все параметры и конфигурацию модели. Обладая доступом к этим данным, можно при необходимости изменять или дополнять пара-

метры удобным способом. Кроме того, рекомендуется фиксировать стандартные контрольные точки на ключевых этапах процесса обучения.

*Код модели (model code)*

Предоставление только весов модели можно сравнить с передачей программного обеспечения в виде двоичного кода без исходного. Код модели включает в себя не только процесс обучения и настройку гиперпараметров, но и алгоритмы предварительной обработки данных. Кроме того, важно раскрывать информацию о конфигурации инфраструктуры, что способствует улучшению воспроизводимости результатов. Однако даже при полном доступе к коду модели ее воспроизведение может быть затруднено из-за ограниченности вычислительных ресурсов и особенностей недетерминированного процесса обучения.

*Обучающие данные (training data)*

Обучающие данные включают информацию, используемую для подготовки модели. Желательно, чтобы они сопровождались деталями или кодом, описывающим процесс их получения. Рекомендуется также публиковать данные на различных стадиях преобразования в рамках конвейера предварительной обработки, а также указывать порядок подачи данных в модель. Этот компонент остается одним из наименее раскрываемых среди разработчиков моделей. В результате большинство моделей с открытым исходным кодом нельзя считать *полностью открытыми*, т. к. наборы данных часто остаются недоступными для общего пользования.

Обучающие данные часто остаются непубличными в силу конкурентных соображений. Как указано в *главах 3* и *4*, современные языковые модели обычно основаны на схожих архитектурных решениях и алгоритмах обучения. Основное различие между моделями часто заключается в особенностях содержимого данных и методах их предварительной обработки. Некоторые фрагменты обучающих данных могут быть приобретены на условиях лицензии, которая запрещает их раскрытие провайдерами моделей.

Еще одна причина недоступности обучающих данных для публикации связана с нерешенными юридическими вопросами, особенно в области авторского права. Например, набор данных The Pile от Eleuther AI больше не доступен по официальной ссылке, т. к. он содержит тексты из книг, защищенных авторским правом (набор данных Books3).

Важно подчеркнуть, что The Pile был предварительно обработан. Книги были разделены на фрагменты, перемешаны и перетасованы, что делает невозможным их прямое прочтение человеком или легкое воспроизведение в исходном виде.

Многие обучающие данные поступают из открытых источников в Интернете, что делает их уязвимыми для насильственного или сексуального контента, который запрещен. Несмотря на усилия по фильтрации и обеспечению безопасности данных, некоторые нежелательные элементы все же могут попасть в итоговый набор. Это привело к тому, что многие ранее открытые наборы данных потеряли свой статус. Одним из ярких примеров такой ситуации являются наборы изображений LAION.

Лицензия, под которой выпускаются большие языковые модели, играет ключевую роль в регулировании их использования, модификации и распространения. Обычно открыто распространяемые LLM предоставляются на основании одной из трех видов лицензий.

*Некоммерческая лицензия (noncommercial license)*

Некоммерческая лицензия предоставляет возможность использования модели исключительно в исследовательских или личных целях, строго запрещая коммерческое применение. Часто для получения доступа к этим моделям требуется заполнение специальной формы заявки, где пользователь должен аргументировать свою потребность, подробно описав предполагаемый сценарий использования в рамках исследований.

*Копилефт-лицензия (copyleft license)*

Копилефт-лицензия или лицензия авторского лева предусматривает возможность коммерческого использования, однако обязует распространять как оригинальные материалы, так и производные работы исключительно на тех же условиях. Это ограничение усложняет создание закрытых или проприетарных модификаций. Конкретные детали соблюдения данного требования зависят от особенностей выбранной лицензии.

*Разрешительная лицензия (permissive license)*

Данный вид лицензии позволяет использовать программное обеспечение в коммерческих целях, включая его модификацию и распространение в рамках закрытых приложений. Она не накладывает обязательств по распространению исходного кода с открытым доступом. В некоторых случаях такие лицензии также предоставляют возможность использования патентов.

В настоящее время разрабатываются новые типы лицензий, которые вводят ограничения на использование моделей для определенных целей. Эти ограничения, как правило, связаны с вопросами безопасности. Одним из примеров таких лицензий является Open RAIL-M (**https://oreil.ly/2UVMe**), которая запрещает использовать модель в определенных сферах, включая медицинские консультации, деятельность правоохранительных органов, иммиграционные процессы и запросы на предоставление убежища. Полный перечень ограничений можно найти в приложении A к данной лицензии.

Если вы, как практикующий специалист, планируете внедрить открытые LLM в своей организации в коммерческих целях, рекомендуется отдавать предпочтение моделям с разрешительными лицензиями. Наиболее популярными из них являются Apache 2.0 и MIT.

Одной из наиболее распространенных категорий лицензий, часто применяемых для распространения открытых языковых моделей, являются лицензии Creative Commons (CC, **https://oreil.ly/8A0pn**). Эти лицензии имеют различные обозначения, такие как CC-BY-NC-SA и др. Чтобы легче запомнить их значения, можно воспользоваться следующим простым объяснением.

*BY (attribution — атрибуция)*

Это обозначение подразумевает обязательное указание авторства (атрибуцию). Лицензия, содержащая только CC-BY, считается разрешительной.

*SA (share-alike — на тех же условиях)*

Данное обозначение указывает на необходимость распространения производных материалов на условиях, аналогичных условиям оригинальной лицензии. Другими словами, это копилефт-лицензия.

*NC*

NC обозначает "некоммерческий". Это означает, что, если лицензия включает данный термин, использование модели ограничивается исключительно исследовательскими или личными целями.

*ND*

ND, в свою очередь, расшифровывается как "без производных" (no derivative works). При наличии такого условия в лицензии любые модификации модели запрещены для распространения.



На сегодняшний день моделями с открытым исходным кодом считаются модели, которые предоставляют доступ к своим весам и исходному коду, а также распространяются под лицензией, позволяющей свободно использовать их в различных целях. Однако в последнее время все чаще возникает вопрос о том, насколько важным является доступ к обучающим данным для полноценного изучения и проверки таких моделей? Этот аспект может стать ключевым в уточнении определения "открытый исходный код".

В табл. 5.2 приведен список доступных моделей LLM, их лицензии, а также доступные размеры и разновидности. Важно учитывать, что LLM может быть дополнительно обучена следованию инструкциям или адаптирована к чат-формату, который отличается от формата предобучающей выборки.

**Таблица 5.2.** Список доступных LLM

| **Название** | **Доступность** | **Размер** | **Версии** |
|---|---|---|---|
| GPT-4 | Проприетарная | Н/д | GPT-4 32K Context, GPT-4 8K Context |
| GPT-3.5 Turbo | Проприетарная | Н/д | GPT-3.5 4K Context, GPT-3.5 16K Context |
| Claude Instant | Проприетарная | Н/д | – |
| Claude 2 | Проприетарная | Н/д | – |
| MPT | Apache 2.0 | 1B, 7B, 30B | MPT 65K StoryWriter |
| CerebrasGPT | Apache 2.0 | 111M, 256M, 590M, 1.3B, 2.7B, 6.7B, 13B | CerebrasGPT |

**Таблица 5.2** (окончание)

| Название | Доступность | Размер | Версии |
|---|---|---|---|
| Stability LM | CC-BY-SA | 7B | – |
| RedPajama | Apache 2.0 | 3B, 7B | RedPajama-INCITE-Instruct, RedPajama-INCITE-Chat |
| GPT-Neo X | Apache 2.0 | 20B | – |
| BLOOM | Открытая, ограниченное использование | 176B | BLOOMZ |
| Llama | Открытая, без коммерческого использования | 7B, 13B, 33B, 65B | – |
| Llama 2 | Открытая, без коммерческого использования | 7B, 13B, 70B | Llama 2-Chat |
| Zephyr | Apache 2.0 | 7B | – |
| Gemma | Открытая, ограниченное использование | 2B, 7B | Gemma-Instruction Tuned |

## Выбор LLM для конкретной задачи

При выборе LLM для конкретной задачи необходимо учитывать множество факторов. В зависимости от ситуации, следует принимать во внимание следующие критерии.

*Стоимость (cost)*

Стоимость включает в себя расходы на вывод результатов или дообучение моделей, а также затраты, связанные с организацией процессов, таких как скаффолдинг (scaffolding — строительные леса, вре́менная поддержка), мониторинг, управление наблюдаемостью (observability), развертывание (deployment) и последующее обслуживание. Все эти компоненты в совокупности образуют понятие LLMOps (operations for large language models — набор практик и инструментов, направленных на управление жизненным циклом больших языковых моделей).

*Время на выходной токен (time per output token, TPOT)*

Метрика TPOT (**https://oreil.ly/mEDRt**) служит для оценки скорости генерации текста, которую ощущает конечный пользователь.

*Выполнение задачи (task performance)*

Подразумевает соответствие заданным требованиям по производительности и ключевым метрикам, таким как точность (precision) и правильность (accuracy). Какой уровень показателей можно считать приемлемым?

*Тип задачи (type of tasks)*

Тип задачи определяет специфику целей, для которых будет использоваться языковая модель, например суммаризация, ответы на запросы, классификация и др.

*Необходимые способности (capabilities required)*

К числу таких способностей относятся арифметическое и логическое рассуждение, планирование, декомпозиция задач и многие другие. Как отмечалось в *главе 1*, многие из этих способностей, если они действительно имеются или близки к реализации, являются *эмерджентными свойствами* (emergent properties) больших языковых моделей и не могут быть воспроизведены на менее масштабных моделях.

*Лицензирование (licensing)*

Следует использовать исключительно те модели, которые разрешают выбранный режим эксплуатации. Даже если модели явно допускают коммерческое применение, они могут накладывать ограничения на ряд конкретных сценариев. К примеру, как упоминалось ранее, лицензия Big Science OpenRAIL-M ограничивает использование больших языковых моделей в таких областях, как деятельность правоохранительных органов; иммиграционные процессы и процедуры предоставления убежища.

*Квалификация пользователя в области машинного обучения и машинного обучения в производственных операциях (ML/MLOps)*

Уровень квалификации определяет возможности пользователей в настройке и развитии собственных систем. Например, обладает ли команда достаточной квалификацией для создания решений, оптимизирующих процессы вывода?

*Прочие нефункциональные критерии (nonfunctional criteria)*

К ним относятся безопасность, защита данных, конфиденциальность и другие важные аспекты. В настоящее время облачные провайдеры и стартапы активно предлагают решения, которые эффективно справляются с этими задачами.

**Упражнение**

Составьте упорядоченный список приоритетов для своего приложения и определите, какие из них являются постоянными, а какие могут изменяться. Например, точность должна быть не менее $X$, а TPOT — не менее $Y$.

Какие модели вы бы выбрали, учитывая заданные приоритеты?

Как правило, на практике часто приходится выбирать между созданием собственной модели и использованием открытой версии LLM.

## Сравнение открытых и проприетарных языковых моделей

Дискуссии о преимуществах программного обеспечения с открытым исходным кодом в сравнении с проприетарными решениями занимают важное место в технологической сфере уже на протяжении многих лет, и сегодня эти обсуждения становятся особенно актуальными в контексте развития больших языковых моделей. Одним из ключевых достоинств моделей с открытым исходным кодом выступают их прозрачность и гибкость, которые часто превосходят даже соображения стоимости. При этом самостоятельное размещение таких моделей нередко сопряжено со значительными инженерными затратами, а также расходами на вычислительные ресурсы и память. Более того, управляемые сервисы, предоставляющие доступ к моделям с открытым исходным кодом, иногда уступают проприетарным аналогам по показателям задержки, производительности и стоимости использования. Дополнительно следует учитывать ограниченную доступность некоторых открытых LLM через облачные сервисы или другие варианты развертывания сторонних разработчиков. Эта ситуация, вероятно, будет меняться по мере развития отрасли. Однако на текущем этапе крайне важно продумать и провести детальные расчеты для конкретной задачи, чтобы объективно оценить совокупные затраты использования моделей каждого типа.

Гибкость, предоставляемая моделями с открытым исходным кодом, существенно облегчает их настройку, интерпретацию и аугментацию с использованием различных подходов к обучению или дообучению. Это делает их гораздо более универсальными инструментами по сравнению с ограниченными возможностями моделей, которые зависят только от провайдера LLM. Такая адаптивность позволяет точнее подстраивать работу модели под индивидуальные предпочтения и ценности пользователя, а не ограничиваться рамками, установленными разработчиком. Полный доступ к вероятностям токенов (логитам) является мощным ресурсом, значение которого станет особенно заметным по мере дальнейшего рассмотрения темы.

Благодаря LLM с открытым исходным кодом, небольшие команды могут разрабатывать модели и приложения, которые могут оказаться нерентабельными для крупных корпораций, ориентированных на получение прибыли. Например, это касается дообучения моделей с целью поддержки языков с ограниченными ресурсами, таких как региональные языки Индии или языки коренных народов Канады, для которых в Интернете недостаточно данных. Одним из примеров является модель Kannada Llama (**https://oreil.ly/hoBQ1**) на основе Llama 2. Ее непрерывное предобучение и дообучение проводились на токенах из языка каннада — регионального языка Индии.

Некоторые модели с открытым исходным кодом могут оказаться не такими прозрачными, как кажется на первый взгляд. Как уже было отмечено, большинство коммерческих компаний, разрабатывающих открытые языковые модели, не предоставляют свободный доступ к обучающим наборам данных. Например, Meta не раскрыла всю

информацию о наборах данных, использованных для обучения модели Llama 2. Знание состава обучающих наборов данных позволяет оценить вероятность загрязнения тестового набора и лучше понять, какими знаниями обладает модель.

На момент написания книги модели с открытым исходным кодом, такие как Llama 3.2 и DeepSeek v3, практически сравнялись по возможностям с передовыми проприетарными решениями от OpenAI и Anthropic. Однако в области моделей рассуждений, (например, модель o3 от OpenAI), которые используют продвинутые методы вычисления времени вывода, появился новый разрыв. Более подробно этот вопрос будет рассмотрен в *главе 8*. По мере изложения в книге будут представлены сценарии, в которых решения с открытым исходным кодом демонстрируют явное преимущество.



При выборе провайдера моделей важно обращать внимание на наличие активного сообщества разработчиков на таких платформах, как GitHub, Discord или Slack. Также следует убедиться, что команда разработчиков активно поддерживает эти каналы, своевременно отвечая на вопросы и комментарии пользователей. Предпочтение стоит отдавать моделям с динамично развивающимся сообществом, если они соответствуют вашим основным требованиям и целям использования.

## Оценка больших языковых моделей

Оценка LLM представляет собой одну из самых сложных задач в современном пространстве данных. Существующие методы бенчмаркинга часто оказываются несостоятельными, их результаты легко подвергаются манипуляциям и зачастую трудны для интерпретации. Несмотря на это, бенчмарки все еще играют важную роль, служа отправной точкой для анализа. В этом разделе мы сначала обсудим доступные на сегодняшний день стандартные тесты, а затем рассмотрим пути разработки более комплексных и полноценных внутренних методов оценки.

Эффективность LLM при выполнении задач оценивается с использованием различных эталонных наборов данных (benchmark dataset), которые проверяют различные навыки модели. Поскольку не все из этих тестов могут быть релевантны конкретным целям, целесообразно сосредоточиться на тех тестах, которые оценивают способности, критически важные для оптимального функционирования модели.

Рейтинги лидеров в подобных тестах могут меняться, особенно если речь идет о моделях с открытым исходным кодом. Однако это не означает, что необходимо обновлять текущую LLM каждый раз, когда появляются новые лидеры. Обычно различия между лучшими моделями незначительны. В большинстве случаев выбор конкретной модели не является определяющим фактором успеха проекта. Гораздо важнее уделить внимание обработке и глубокому анализу данных, поскольку это по-прежнему является ключевым элементом любого успешного начинания в данной области.

Рассмотрим подробнее несколько популярных методов оценки LLM.

### LM Evaluation Harness от Eleuther AI

LM Evaluation Harness (**https://oreil.ly/SiOXq**) — это инструмент, разработанный Eleuther AI для проведения эталонного тестирования более 400 различных задач, включая вопросы открытой области, арифметические и логические рассуждения, лингвистические задачи, машинный перевод, а также выявление токсичного языка. Инструмент позволяет оценивать навыки моделей на основе заранее заданных критериев. Он совместим с любой моделью из Hugging Face Hub (**https://oreil.ly/IHd22**) — платформы, на которой собраны тысячи предварительно обученных и настроенных моделей. Используя все его доступные функции, можно протестировать каждую из них на конкретной задаче.

Рассмотрим задание `bigbench_formal_fallacies_syllogisms_negation` из эталонного набора тестов:

```
{
  "input": "\"Some football fans admire various clubs, others love only a single team.
But who is a fan of whom precisely? The following argument pertains to this question:
First premise: Mario is a friend of FK \u017dalgiris Vilnius. Second premise: Being a
follower of F.C. Copenhagen is necessary for being a friend of FK \u017dalgiris
Vilnius. It follows that Mario is a follower of F.C. Copenhagen.\"\n Is the argument,
given the explicitly stated premises, deductively valid or invalid?",
  "target_scores": {
      "valid": 1,
      "invalid": 0
  }
```

*(Некоторые футбольные болельщики увлекаются несколькими клубами, в то время как другие преданно поддерживают только одну команду. Как же определить, кто является настоящим фанатом? На этот вопрос можно ответить с помощью следующего рассуждения: первое утверждение: Марио является другом футбольного клуба FK \u017dalgiris Vilnius. Второе утверждение: чтобы быть другом клуба FK \u017dalgiris Vilnius, необходимо быть болельщиком F.C. Copenhagen. Следовательно, Марио является болельщиком F.C. Copenhagen. Можно ли сделать вывод, что этот аргумент является дедуктивно обоснованным с учетом имеющихся предпосылок?)*

В этом задании модель должна выявить логические ошибки, определив, является ли представленный аргумент обоснованным (валидным, valid) с учетом имеющихся предпосылок.

#### Упражнение

Проведем оценку нескольких моделей для данной задачи. Для этого необходимо установить инструмент LM Evaluation Harness, следуя инструкциям по ссылке **https://oreil.ly/mZdGA**. После завершения установки можно протестировать модель Falcon 7B, используя следующий код:

```
lm_eval --model hf-causal \
        --model_args pretrained=tiiuae/falcon-7b \
        --tasks bigbench_formal_fallacies_syllogisms_negation \
        --device cuda:0
```

Этот процесс можно повторить для других моделей с параметром 7B, включая такие варианты, как Llama, Gemma, Mistral, MPT, RedPajama. Рекомендуется протестировать как базовые версии моделей, так и обученные следованию инструкциям (если они доступны). В ходе этого анализа важно оценить различия в их производительности — проявляют ли они заметное расхождение?

Кроме того, полезно дополнить эталонный тест собственными вопросами. Можно составить десять дополнительных вопросов, которые охватывают разнообразные области, связанные с выбранной задачей. При необходимости для генерации идей можно использовать LLM, а затем скорректировать созданные вопросы вручную. После этого стоит сравнить результаты моделей на этих новых вопросах с их показателями на оригинальных тестах. Это поможет оценить, насколько согласованы их результаты при выполнении задачи по разным сценариям.

Инструмент LM Evaluation Harness также позволяет оценивать проприетарные модели. Например, модели OpenAI можно оценить следующим образом:

```
export OPENAI_API_SECRET_KEY=<Key>
python main.py \
lm_eval --model openai-completions \
        --model_args model=gpt-3.5-turbo \
        --tasks bigbench_formal_fallacies_syllogisms_negation
```

**Упражнение**

Сравните модели GPT 4o, 4o-mini, o1 и o3 по задаче поиска логических ошибок (logical fallacies task). Используйте как эталонные наборы данных, так и данные, подготовленные самостоятельно. Как эти модели соотносятся друг с другом и насколько они эффективны по сравнению с моделями с открытым исходным кодом?



При выборе или разработке задачи для проведения бенчмаркинга с целью оценки рекомендуется сосредоточить внимание на тех задачах, которые тестируют способности модели, необходимые для решения конкретной задачи, а не на задачах как таковых. Например, если разрабатывается приложение-суммаризатор (summarizer application), которое должно выполнять множество логических операций для создания сводки, то целесообразно сосредоточиться на тестах производительности, которые непосредственно проверяют возможности логических операций, а не на тестах производительности суммаризатора.

## Таблица лидеров открытых LLM на платформе Hugging Face

Таблица лидеров, или лидерборд (leaderboard), больших языковых моделей с открытым исходным кодом (Open LLM Leaderboard, **https://oreil.ly/tspBY**) на платформе Hugging Face формирует рейтинг моделей на основе их производительности

по ряду задач. На момент написания книги для оценки используется инструмент LM Evaluation Harness от Eleuther AI, который анализирует качество работы моделей через шесть различных бенчмарков.

*Массовое многозадачное понимание языка (Massive Multitask Language Understanding, MMLU)*

Этот тест предназначен для оценки возможностей языковых моделей в решении предметно-ориентированных задач, охватывая более 50 дисциплин, включая историю США, биологию, математику и другие, и опирается на формат тестов с множественным выбором.

*AI2 Reasoning Challenge (ARC)*

Тест на наборе данных ARC для сравнения искусственного интеллекта проверяет способность LLM отвечать на вопросы из школьной программы с несколькими вариантами ответов. Эти вопросы требуют не только глубокого понимания учебного материала, но и обширных энциклопедических знаний.

*HellaSwag*

Этот тест проверяет, насколько хорошо модель способна логически мыслить. Она должна проанализировать ситуацию и выбрать наиболее вероятный вариант развития событий, основываясь на здравом смысле.

*TruthfulQA*

Этот тест предназначен для проверки способности LLM давать правдивые ответы, не содержащие недостоверную информацию.

*WinoGrande*

Этот тест содержит вопросы с пропусками (fill-in-the-blank questions), которые помогают оценить логические способности модели.

*GSM-8K*

Этот тест предназначен для проверки способности LLM решать задачи по математике в рамках школьной программы, которые требуют выполнения основных арифметических действий в правильной последовательности.

На рис. 5.3 представлен фрагмент лидерборда LLM на момент написания книги. На основании этого фрагмента можно заключить следующее:

- более крупные модели демонстрируют лучшие результаты;
- версии, обученные следовать инструкциям, или дообученные модели также показывают более высокие результаты.

### Упражнение

Предположим (**https://oreil.ly/fwVEC**), что многие модели были обучены на данных, которые включают элементы из набора GSM8K (**https://oreil.ly/dI87I**). Проведите частичный анализ содержимого этого набора и оцените, до какой степени модели, тестируемые вами в предыдущих экспериментах, способны корректно

завершать вопросы из данного ресурса. Дополнительно измените числовые значения в формулировках задач и проанализируйте, как подобные модификации отражаются на точности решений, предоставляемых моделями.

| Rank | Type | Model | Average ⓘ |
|---|---|---|---|
| 1 | ◆ | MaziyarPanahi/calme-3.2-instruct-78b | 52.08 % |
| 2 | 💬 | MaziyarPanahi/calme-3.1-instruct-78b | 51.29 % |
| 3 | 💬 | dfurman/CalmeRys-78B-Orpo-v0.1 | 51.23 % |
| 4 | 💬 | MaziyarPanahi/calme-2.4-rys-78b | 50.77 % |
| 5 | ◆ | huihui-ai/Qwen2.5-72B-Instruct-abliterated | 48.11 % |
| 6 | 💬 | Qwen/Qwen2.5-72B-Instruct | 47.98 % |
| 7 | 💬 | MaziyarPanahi/calme-2.1-qwen2.5-72b | 47.86 % |

**Рис. 5.3.** Фрагмент таблицы лидеров Open LLM Leaderboard

Достоверность этих тестов остается под вопросом, т. к. не всегда возможно обеспечить полную чистоту тестового набора. Кроме того, провайдеры моделей, как правило, оптимизируют выполнение заданий из таких наборов, что снижает объективность результатов при оценке эффективности универсальных моделей.

## HELM

Целостная оценка языковых моделей (Holistic Evaluation of Language Models, HELM, **https://oreil.ly/MNHDs**) — это оценочный фреймворк Стэнфорда, предназначенный для расчета разнообразных метрик по множеству контрольных задач. В общей сложности вычисляются 59 показателей: точность тестирования (testing accuracy), калибровка (calibration), надежность (robustness), справедливость (fairness), предвзятость (bias), токсичность (toxicity), эффективность (efficiency), показатели суммаризации (summarization performance), нарушение авторских прав (copyright infringement) и многие другие.

Проверяемые задачи включают: ответы на вопросы (question answering), суммаризацию (summarization), классификацию текста (text classification), информационный поиск (information retrieval), анализ тональности текста (sentiment analysis), выявление токсичных данных (toxicity detection) и многие другие.

На рис. 5.4 представлен фрагмент лидерборда HELM на момент написания книги.

Accuracy | Efficiency | General information

| Model | Mean win rate | NarrativeQA - F1 | NaturalQuestions (open) - F1 | NaturalQuestions (closed) - F1 |
|---|---|---|---|---|
| GPT-4o (2024-05-13) | **0.938** | **0.804** | 0.803 | 0.501 |
| GPT-4o (2024-08-06) | 0.928 | 0.795 | 0.793 | 0.496 |
| DeepSeek v3 | 0.908 | 0.796 | 0.765 | 0.467 |
| Claude 3.5 Sonnet (20240620) | 0.885 | 0.746 | 0.749 | **0.502** |
| Amazon Nova Pro | 0.885 | 0.791 | **0.829** | 0.405 |
| GPT-4 (0613) | 0.867 | 0.768 | 0.79 | 0.457 |
| GPT-4 Turbo (2024-04-09) | 0.864 | 0.761 | 0.795 | 0.482 |

**Рис. 5.4.** Фрагмент таблицы лидеров HELM

### Бенчмаркинг не всегда является надежным

Существует несколько подходов к оценке эффективности работы модели на одной и той же задаче. Например, возьмем задачу MMLU. В ней вопросы имеют четыре варианта ответов: A, B, C, D. Но как определить производительность модели при выполнении задачи с множественным выбором ответов?

Возможны несколько вариантов:

- определить токен, который с наибольшей вероятностью является правильным ответом из предложенных вариантов (A, B, C или D);
- найти наиболее вероятный токен из всего словаря в контексте ответа на вопрос и сравнить его с фактическим ответом (который не помечен как A, B, C или D);
- рассчитать нормализованную сумму вероятностей последовательности токенов, сгенерированных моделью, при условии, что ожидаемой последовательностью является метка, предшествующая правильному ответу. Затем сравнить этот результат с фактическим ответом, который также представлен меткой и текстом ответа.

Каждый из методов расчета способен привести к совершенно различным результатам, что в свою очередь может повлиять на расстановку лидеров в рейтинговых таблицах. Команда Hugging Face подробно рассмотрела эту тему в своем блоге (**https://oreil.ly/QrBX4**) после того, как пользователи начали замечать расхождения между собственными метриками и сторонними оценками.

## Рейтинговая система Эло

После анализа ограничений количественной оценки рассмотрим вопрос о том, как наиболее эффективно интегрировать субъективные оценки. Одним из наиболее подходящих инструментов для этого является рейтинговая система Эло (Elo rating system, **https://oreil.ly/bTD7I**), которая широко применяется в шахматах для определения уровня игроков. Этот метод может стать перспективным инструментом для интеграции человеческих оценок в различные процессы.

На основе рейтинговой системы Эло организация LMSYS Org (Large Model Systems Organization — организация "Системы больших моделей", **https://oreil.ly/HGVz2**) реализовала оценочную платформу Chatbot Arena, которая представляет собой краудсорсинговую платформу для оценки, предлагая пользователям выбирать между двумя случайными незнакомыми моделями LLM, общаясь с ними одновременно. Рейтинги лидеров, среди которых модели от OpenAI, DeepSeek, Google DeepMind и Anthropic, доступны по ссылке **https://oreil.ly/Y6zmN**.

На рис. 5.5 показан фрагмент лидерборда Chatbot Arena на момент написания книги.

| Rank* (UB) | Rank (StyleCtrl) | Model | Arena Score |
|---|---|---|---|
| 1 | 3 | Gemini-2.0-Flash-Thinking-Exp-01-21 | 1384 |
| 1 | 2 | Gemini-2.0-Pro-Exp-02-05 | 1379 |
| 1 | 1 | ChatGPT-4o-latest (2025-01-29) | 1377 |
| 4 | 2 | DeepSeek-R1 | 1361 |
| 4 | 7 | Gemini-2.0-Flash-001 | 1355 |
| 4 | 2 | o1-2024-12-17 | 1352 |
| 7 | 5 | o1-preview | 1335 |
| 7 | 7 | Qwen2.5-Max | 1332 |
| 9 | 8 | DeepSeek-V3 | 1316 |
| 9 | 9 | Gemini-2.0-Flash-Lite-Preview-02-05 | 1309 |

**Рис. 5.5.** Фрагмент таблицы лидеров Chatbot Arena

### Рейтинговая система Эло обладает определенной уязвимостью

Хотя рейтинговая система Эло представляет собой популярный инструмент для оценки языковых моделей, полностью устранить влияние субъективных факторов невозможно, даже если процесс оценки проводится анонимно.

В исследовании, проведенном By и соавт. (Wu et al., **https://oreil.ly/3g_qd**), были выявлены следующие ключевые аспекты данного феномена:

- люди склонны отдавать предпочтение более длинным ответам;
- авторитетный или убедительный стиль изложения может повлиять на объективность оценки точности и согласованности предоставленных данных;
- участники оценки иногда проявляют нерешительность, выбирая компромисс вместо однозначного победителя;
- оценки могут зависеть от порядка, в котором представлены ответы языковых моделей.

Одним из методов снижения вероятности возникновения подобных предубеждений является рандомизация ответов, которая позволяет избежать влияния их последовательности.

Кроме того, команда By предложила многорейтинговую систему Эло (multi-Elo rating system), которая предполагает оценку LLM по трем различным критериям: полезность (helpfulness), точность (accuracy) и языковые навыки (language).

### Анализ результатов бенчмарков

Как правильно интерпретировать результаты научных исследований? Важно методично задавать как можно больше вопросов и проверять, насколько полно ответы на них представлены в статье или другом источнике.

Рассмотрим графики из работы, посвященной Llama 2 (**https://oreil.ly/BcgX**), которые демонстрируют оценку Llama 2-Chat. В частности, на рисунках 1 и 3 представлено сравнение Llama 2-Chat с другими моделями чат-ботов по критериям полезности и безопасности. В связи с этим возникают следующие вопросы:

- Как представлен набор данных для оценки? Можем ли мы получить к нему доступ?
- Насколько сложен тестовый набор? Можно ли сказать, что модель конкурентоспособна с ChatGPT для простых задач или ее потенциал раскрывается в более сложных ситуациях?
- Какой процент примеров в тестовом наборе содержит сложные случаи?
- В рамках тестового набора рассматриваются различные сценарии. Насколько эти сценарии соответствуют наборам настроек чата?
- Как они формулируют понятие "безопасность"?
- Существует ли риск систематической ошибки в оценке моделей, связанной с тем, что они оцениваются на основе определения безопасности, которое использовалось при обучении Llama 2, в то время как другие модели могут иметь иное представление о безопасности?

Глубокий анализ полученных результатов позволяет лучше понять оцениваемые параметры и их соответствие возможностям, необходимым языковой модели для

выполнения поставленных задач. Чтобы обеспечить более точную оценку качества работы модели, рекомендуется разработать собственные специализированные бенчмарки.



Недоверие к оценкам, сделанным GPT-4 или другими языковыми моделями, вполне оправдано. Точные критерии, которыми они руководствуются, остаются неизвестными, как и возможные предвзятости, влияющие на их выводы.

Анализ способностей LLM усложняется спецификой формулировки запросов и вероятностным характером работы генеративных моделей. Многие исследования утверждают, что модель GPT-4 лишена способности к рассуждению, однако при этом часто упускают из виду влияние приемов создания промптов или промптинга (prompting techniques) на результаты оценки. На практике нередко оказывается, что модель способна успешно справляться с задачей, если воспользоваться методом промптинга с цепочкой рассуждений (chain-of-thought prompting, CoT prompting). Хотя для оценки возможностей модели не требуется сложная настройка промптов, внедрение таких элементарных методов должно стать общепринятым стандартом. Пренебрежение ими может привести к недооценке потенциала модели и ограничению ее реальных возможностей.

## Загрузка моделей

Хотя большую языковую модель можно запустить на центральном процессоре (CPU), для обеспечения приемлемой скорости генерации текста рекомендуется использовать графический процессор (GPU). Выбор подходящего GPU зависит от множества факторов, включая его стоимость, размер модели, режим работы (обучение или просто генерация текста) и наличие оптимизаций. Тим Деттмерс (Tim Dettmers) представил удобную блок-схему (**https://oreil.ly/t6iPQ**), которая помогает выбрать наиболее подходящий GPU для конкретных задач.

Теперь выясним, какой объем оперативной памяти понадобится графическому процессору, чтобы загрузить LLM определенного размера. Модели могут быть загружены с различной точностью.

*Float32*

32-битное представление с плавающей запятой, где каждый параметр занимает 4 байта памяти.

*Float16*

16-битное представление чисел с плавающей запятой, в котором для экспоненты отводятся только 5 бит, в отличие от 8 бит во Float32. Это может привести к проблемам с переполнением или опустошением при работе с экстремально большими или малыми числами.

*bfloat16 (BF16)*

16-битное представление с плавающей запятой. Как и в случае с Float32, 8 бит зарезервированы для экспоненты, что позволяет избежать проблем с недостатком и переполнением, которые могут возникать в Float16.

*Int8*

8-битное целочисленное представление. Выполнение вывода в 8-битном режиме примерно на 20% медленнее, чем в Float16.

*FP8, FP4*

8-битное и 4-битное представления с плавающей запятой.

Эти форматы будут подробно рассмотрены в *главе 9*. Обычно для запуска вывода на моделях с параметрами 7B требуется около 7 Гбайт ОЗУ GPU при работе в 8-битном режиме и примерно 14 Гбайт при использовании BF16. Однако дообучение всей модели потребует значительно большего объема памяти.

## Библиотека Hugging Face Accelerate

Можно использовать модели для вывода данных, даже если их размер превышает возможности оперативной памяти графического процессора. Библиотека Hugging Face Accelerate (**https://oreil.ly/OYdyf**) упрощает этот процесс, позволяя частично загружать модель в оперативную память центрального процессора, освобождая место в графическом процессоре. При необходимости, когда память CPU RAM исчерпывается, фрагменты модели пересылаются на жесткий диск. В работе "Ускорение вывода большой модели: как это работает?" ("Accelerate Big Model Inference: How Does it Work?", **https://oreil.ly/J8duc**) раскрывается внутренний механизм функционирования данной библиотеки. Поскольку пользователь не нуждается в понимании всех нюансов процесса, для загрузки большой модели достаточно запустить следующий код:

```
!pip install transformers accelerate


import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neox-20B")
model = GPTNeoForCausalLM.from_pretrained("EleutherAI/gpt-neox-20B")
input_ids = tokenizer("Language models are", return_tensors="pt")
gen_tokens = model.generate(**input_ids, max_new_tokens =1)
```

## Библиотека Ollama

Существует множество инструментов, которые упрощают локальную загрузку языковых моделей, включая использование собственного ноутбука. Одним из таких решений является библиотека Ollama, работающая на операционных системах

Windows, Mac и Linux. Она позволяет загружать модели объемом 13B при условии наличия не менее 16 Гбайт оперативной памяти на устройстве. Ollama поддерживает широкий спектр открытых моделей, таких как Mistral, Llama, Gemma и др. Помимо этого, библиотека предоставляет REST API, который можно использовать для выполнения запросов и разработки приложений на базе языковых моделей. Также доступны удобные интеграции с терминалом и пользовательским интерфейсом, что существенно облегчает создание приложений для конечных пользователей.

Рассмотрим, как можно использовать модель Gemma 2B (Google) в сочетании с Ollama. Прежде всего, необходимо скачать подходящую версию Ollama на свой локальный компьютер в соответствии с операционной системой, доступную по ссылке **https://oreil.ly/yly44**. Затем необходимо перенести модель Gemma на свой компьютер, выполнив следующую команду.

```
ollama pull gemma:2b
```

Дополнительно можно создать файл конфигурации модели (Modelfile), который будет содержать ключевые настройки. В этот файл включаются промпты, шаблоны промптов, параметры декодирования (например, температура) и история диалогов. Для уточнения доступных вариантов настроек рекомендуется обратиться к официальной документации по ссылке **https://oreil.ly/ba-1u**.

Рассмотрим такой пример:

```
FROM gemma:2b

PARAMETER temperature 0.2
SYSTEM """

You are a provocateur who speaks only in limericks.
(Ты провокатор, выражающий свои мысли исключительно в лимериках.)
"""
```

С помощью созданного файла Modelfile можно запустить модель:

```
ollama create local-gemma -f ./Modelfile
ollama run local-gemma
```

В репозитории GitHub доступен пример полностью функционального приложения, созданного на базе Ollama и одной из его UI-интеграций. Можно также попробовать свои силы с другими похожими инструментами, такими как LM Studio (**https://oreil.ly/uFsiR**) и GPT4All (**https://oreil.ly/XUXhq**).



При работе с Ollama возможна загрузка пользовательских моделей только в формате GGUF (GPT-Generated Unified Format — унифицированный формат GPT).

### Интерфейсы для логического вывода больших языковых моделей

Хотя большие языковые модели можно развернуть самостоятельно, современные методы их использования, называемые *инференсом* (inference — вывод), часто требуют сложных оптимизаций, не все из которых можно найти в открытом доступе. Это значительно затрудняет достижение уровня производительности, сопоставимого с коммерческими решениями. Чтобы упростить этот процесс, были созданы сервисы, такие как Together AI (**https://oreil.ly/L3zo0**), которые предлагают инструменты инференса моделей с открытым исходным кодом или индивидуальные решения. Эти платформы позволяют выбирать между бессерверными конечными точками и выделенными экземплярами, что дает возможность адаптировать инфраструктуру под конкретные задачи. Кроме того, недавно инструмент Text Generation Inference (TGI, **https://oreil.ly/XXFpa**), созданный Hugging Face, стал программным обеспечением с открытым исходным кодом и разрешительной лицензией (**https://oreil.ly/BJJlY**).

## Стратегии декодирования

Итак, мы изучили процесс загрузки модели и теперь переходим к созданию качественных и эффективных текстов. В последние годы было разработано несколько *стратегий декодирования* (decoding strategies), которые позволяют решать эту задачу. Рассмотрим их более подробно.

### Жадное декодирование

Самый простой способ декодирования заключается в выборе токена с наибольшей вероятностью на каждом этапе генерации. Однако этот подход имеет существенный недостаток, который заключается в повторяемости в выходных данных.

Рассмотрим такой пример:

```
input = tokenizer('The keyboard suddenly came to life. It ventured up the',
// (Клавиатура внезапно ожила. Он отважился поднять)
return_tensors='pt').to(torch_device)

output = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Легко заметить, что выходные данные начинают приобретать повторяющийся характер, что снижает эффективность жадного декодирования (greedy decoding), особенно при работе с более длинными последовательностями. Этот метод больше подходит для генерации крайне коротких последовательностей, таких как единичные токены в задачах классификации.

На рис. 5.6 показан пример использования жадного декодирования с использованием модели FLAN-T5. Стоит отметить, что при таком подходе некоторые качествен-

ные последовательности оказываются пропущенными. Это связано с тем, что вероятность одного из целевых токенов оказывается чуть ниже, из-за чего он исключается из окончательного выбора.

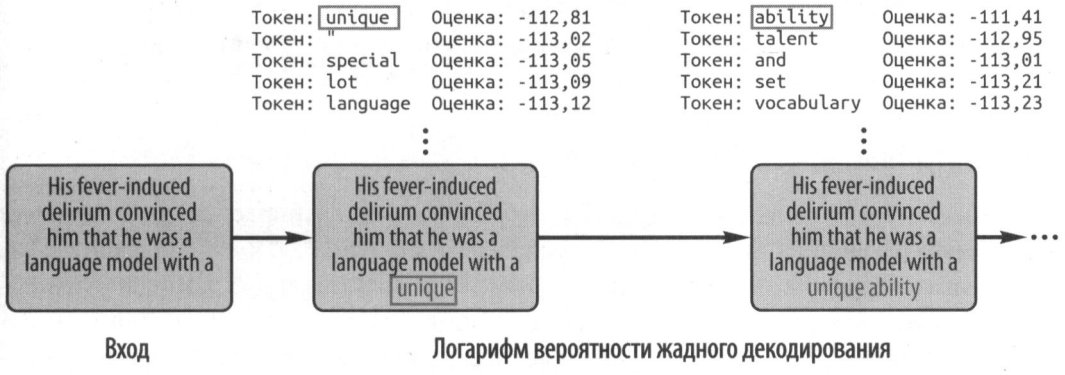



**Рис. 5.6.** Жадное декодирование

## Лучевой поиск

Лучевой поиск (beam search) представляет собой альтернативу жадному декодированию. В этом методе основным параметром выступает размер луча, обозначаемый как *n*. На первом шаге алгоритм отбирает *n* самых вероятных токенов в качестве гипотез. Затем на последующих шагах модель генерирует варианты продолжения для каждого из выбранных токенов. Окончательный выбор следующего токена зависит от того, какое продолжение имеет наивысшую совокупную вероятность.

В библиотеке трансформеров Hugging Face Transformers размер луча можно настроить с помощью параметра `num_beams` в функции `model.generate()`. Пример кода для декодирования с использованием лучевого поиска будет выглядеть следующим образом:

```
output = model.generate(**inputs, max_new_tokens=50, num_beams = 3)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

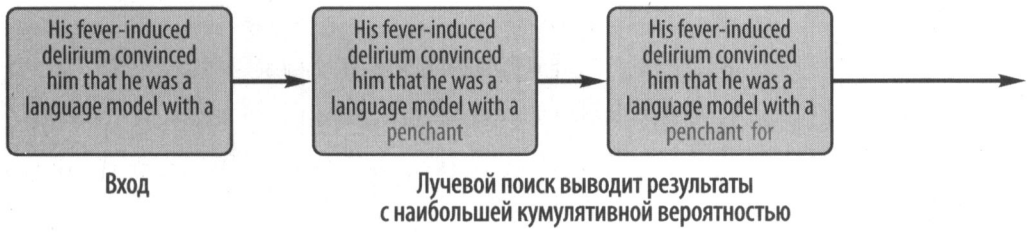



**Рис. 5.7.** Лучевой поиск

На рис. 5.7 показан пример использования лучевого поиска с моделью FLAN-T5. Как видно, лучевой поиск не смог полностью решить проблему повторяемости выходных данных. Как и в случае с жадным декодированием, сгенерированный текст

имеет сжатую и неестественную структуру, т. к. полностью лишен слов с низкой вероятностью.

Для решения задачи нужно добавить элемент случайности и выбирать токены, основываясь на вероятностном распределении. Такой подход обеспечит разнообразие создаваемого текста и позволит избежать постоянного повторения первых нескольких токенов.

## Семплирование top-*k*

В процессе семплирования top-*k* (top-k sampling) модель отбирает лишь *k* наиболее вероятных токенов из выходного распределения для последующей генерации. При этом масса вероятности перераспределяется по этим *k* токенам, и модель случайным образом выбирает один из них для создания следующего токена. Функция генерации в модели Hugging Face позволяет регулировать значение *k*, используя параметр top_k:

```
output = model.generate(**inputs, max_new_tokens=50, do_sample=True, top_k=40)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

На рис. 5.8 показан пример использования семплирования top-*k* с моделью FLAN-T5. Как можно заметить, этот метод значительно превосходит жадное декодирование и лучевой поиск. Однако наличие нескольких токенов с высокой вероятностью может привести к нежелательным результатам, поскольку токены с низкой вероятностью все же попадают в конечный список top-*k*.

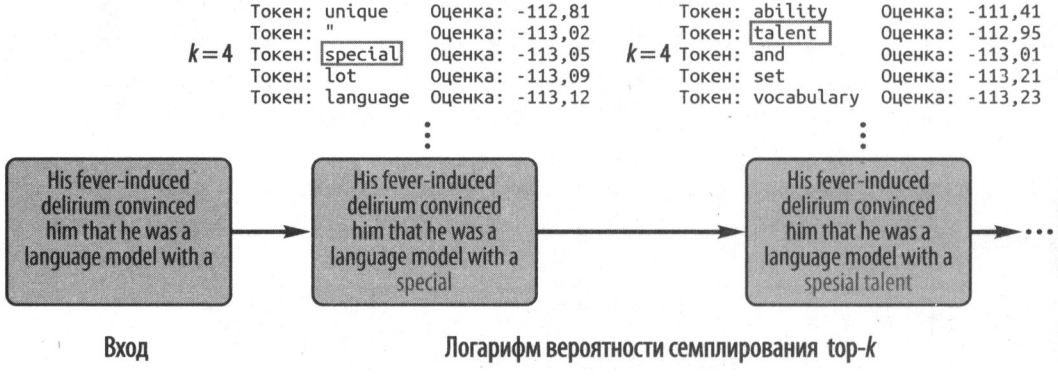



**Рис. 5.8.** Семплирование top-*k*

## Семплирование top-*p*

Техника семплирования top-*p* (top-p sampling) позволяет преодолеть ограничения предыдущего метода, динамически определяя количество потенциальных токенов для выбора. Она заключается в отборе минимального набора токенов, суммарное распределение которых превышает заданный порог вероятности *p*.

Семплирование top-$p$ с использованием трансформеров Hugging Face можно реализовать следующим образом:

```
output = model.generate(**inputs, max_new_tokens=50, top_p=0.9)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

На рис. 5.9 представлен алгоритм top-$p$, реализованный с помощью модели FLAN-T5. На сегодняшний день этот метод, известный также как *ядерное семплирование* или выборка ядра (nucleus sampling), является одним из самых востребованных.

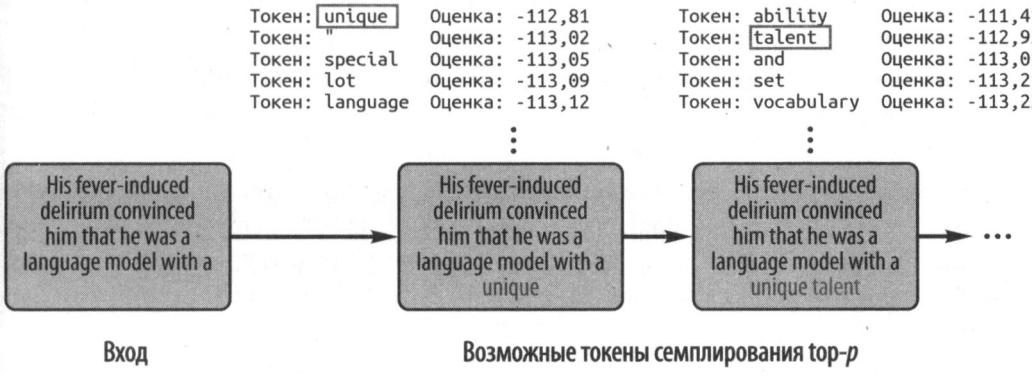



**Рис. 5.9.** Семплирование top-$p$



Все рассмотренные выше методы декодирования являются последовательными. Это означает, что каждый токен генерируется по одному за раз и каждый раз проходит через всю модель. Однако такой подход не позволяет достичь высокой скорости в приложениях, где важна минимальная задержка. *В главе 9* мы рассмотрим альтернативные методы, такие как спекулятивное декодирование (speculative decoding), которые значительно ускоряют процесс декодирования.

## Инференс больших языковых моделей

Теперь, освоив работу с LLM, в том числе загрузку моделей и процесс декодирования, можно подробно рассмотреть, как они используются в решении конкретных задач. Этот процесс называется *инференсом*, или выводом с использованием LLM (LLM inference).

**Упражнение**

Представьте, что вы музыкант, планирующий масштабный концертный тур по семи городам: Амстердам, Варшава, Гамбург, Барселона, Дели, Шанхай и Торонто. Попросите модель разработать оптимальный маршрут, который позволит минимизировать время в пути. Для этого следует использовать методы и стратегии промптинга, описанные в *главе 1*.

Проведите аналогичную работу с моделями LLM, имеющими параметры 3B, 7B и не менее 30B, а также через API проприетарной LLM. Насколько легко управлять этими моделями?

В репозитории GitHub (**https://oreil.ly/llm-playbooks**) доступны примеры задач, которые позволяют оценить навыки работы с промптами. Поработайте с этими примерами и определите, сможете ли вы получить от LLM корректные ответы.

Выходные данные больших языковых моделей могут заметно различаться при многократных генерациях, даже если используется один и тот же промпт. Как отмечалось в разделе о декодировании, если не использовать детерминированный алгоритм, такой как жадное декодирование, LLM осуществляет семплирование токенов из вероятностного распределения.

Чтобы повысить уровень детерминированности в генерации текста, можно установить значение температуры равным нулю и зафиксировать случайное начальное значение выборки. Однако даже при этом нельзя гарантировать абсолютно идентичные результаты при каждом запуске модели с одними и теми же входными данными.

Причинами недетерминизма могут быть различные факторы, начиная с использования многопоточности и заканчивая ошибками округления с плавающей запятой, а также специфические архитектурные решения. Например, известно, что архитектура Sparse MoE (**https://oreil.ly/pzchE**) генерирует непредсказуемые результаты.

Понижение значения температуры до нуля или близкого к нулю приводит к снижению креативности LLM и повышению предсказуемости ее выводов, что может ограничить ее применение в ряде сценариев.

В производственных условиях, где надежность является приоритетом, необходимо выполнить несколько генераций для одного и того же ввода. Чтобы выбрать правильный вывод, следует использовать метод мажоритарного голосования (majority voting) или эвристический подход. Это особенно важно из-за особенностей процесса декодирования: иногда могут генерироваться неправильные токены, а поскольку каждый последующий токен зависит от предыдущих, ошибка может распространяться на большое количество токенов.

**Упражнение**

Для каждого задания по промптингу выполните несколько генераций и посмотрите, как меняются результаты в зависимости от количества запусков. Оцените, насколько точно мажоритарное голосование позволяет выбрать правильный ответ.

Одним из популярных подходов в области промпт-инженерии является метод самосогласованности (self-consistency, **https://oreil.ly/wEE8q**), который объединяет технику CoT-промптинга с механизмом мажоритарного голосования. Его основная

идея состоит в использовании промпта "Давай размышлять пошагово" (Let's think step by step) для генерации нескольких вариантов решения, предложенных через разные пути рассуждения. Затем путем мажоритарного голосования выбирается наиболее подходящий результат.

## Структурированные выходные данные

Иногда требуется, чтобы результаты работы моделей LLM имели строго определенный формат, пригодный для дальнейшей обработки другими системами. Однако реализовать это на практике бывает непросто, поскольку текущие модели не обладают достаточной степенью управляемости. Например, некоторые LLM склонны к излишней многословности. Если их попросить просто ответить "Да" или "Нет", они могут вместо этого сформулировать что-то вроде: "Ответом на данный вопрос будет Да".

Для того чтобы модель генерировала структурированные данные, можно использовать JSON-схему. В этом случае необходимо определить структуру схемы, передать ее модели и запросить генерацию выходных данных в строгом соответствии с заданным форматом.

В случае с крупными моделями этот процесс обычно проходит без сбоев, возможны лишь небольшие отклонения от схемы, которые легко обнаружить и исправить.

Для моделей меньшего размера можно использовать специализированные библиотеки, такие как Jsonformer (**https://oreil.ly/aSc0f**). Jsonformer позволяет LLM создавать контент, одновременно обеспечивая его форматирование в JSON. Благодаря тесной интеграции с платформой Hugging Face, Jsonformer работает с любыми моделями, поддерживаемыми этой платформой.

### Упражнение

Получите сведения о фильмографии актера Эндрю Гарфилда из его страницы на Википедии (**https://oreil.ly/WXtgc**). Создайте JSON-схему, которая будет содержать данные о совместных работах Гарфилда с другими актерами и режиссерами, а также названия фильмов и даты их выхода. Для извлечения деталей о фильмах из неструктурированного текста используйте LLM с открытым исходным кодом, а для преобразования полученной информации в структурированный JSON-формат — библиотеку Jsonformer или аналогичную. Возможно ли получить полностью сформированные и точные выходные данные в формате JSON?

Чтобы упростить работу с более сложными структурированными выходными данными, можно воспользоваться специальными библиотеками, такими как LMQL (**https://oreil.ly/LlkEj**) или Guidance (**https://oreil.ly/cFe5s**). Эти библиотеки предлагают уникальную парадигму программирования, которая делает акцент на промптинге и существенно упрощает процесс управляемой генерации.

Они предоставляют доступ к следующим функциональным возможностям:

*Ограничение вывода конечным набором токенов*

Этот подход особенно эффективен при решении задач классификации, в которых набор выходных меток ограничен. Так, при анализе тональности текста можно ограничить категории результата тремя вариантами: положительная, негативная и нейтральная.

*Управление форматом вывода с помощью регулярных выражений*

Например, с помощью регулярных выражений можно задать собственный формат даты.

*Управление форматом вывода с помощью контекстно-свободных грамматик*

Контекстно-свободные грамматики (context-free grammars, CFG) устанавливают правила, которым должны соответствовать сгенерированные строки. Более подробную информацию об этой технике можно найти в блоге Aditya (**https://oreil.ly/M00us**). С помощью CFG можно более эффективно использовать LLM для решения задач, связанных с тегированием последовательностей (sequence tagging), таких как распознавание именованных сущностей или тегирование частей речи.

### Упражнение

Распознавание именованных сущностей (named entity recognition, NER) представляет собой задачу тегирования последовательности, которая заключается в присвоении тегов именованным сущностям в тексте, таким как числа, даты, места, имена, организации и т. д. Например, для предложения "Падма продала 23 зонта в Гватемале" тегированный вывод может выглядеть следующим образом:

Падма: PER

продала:

23: NUM

зонта:

в:

Гватемале: LOC

где PER — тег для обозначения человека, NUM — тег для обозначения чисел, а LOC — тег для обозначения местоположения.

Сгенерируйте вывод с разметкой тегов, используя выражение CFG из библиотеки Guidance (**https://oreil.ly/R3XLQ**) в указанном выше формате. Запустите задачу распознавания именованных сущностей на Википедии по теме летних Олимпийских игр (**https://oreil.ly/tUrIt**), применяя для ее решения 3B/7B LLM с открытым исходным кодом.

# Отладка и интерпретируемость модели

Теперь, когда мы знаем, как загружать большие языковые модели и использовать их для генерации текста, важно глубже понять, как они работают, и выяснить, каким образом можно обнаружить вероятные сбои. Уровень интерпретируемости моделей LLM остается относительно низким по сравнению с другими областями машинного обучения. Однако можно добиться некоторой степени понимания их поведения, анализируя изменения выходных данных при небольших изменениях входных и изучая промежуточные результаты, которые возникают на различных этапах обработки данных с использованием архитектуры трансформера.

Одним из открытых инструментов с интуитивно понятным интерфейсом является Google LIT-NLP (**https://oreil.ly/YFY4q**), предназначенный для визуализации работы моделей и упрощения отладки различных рабочих процессов.

Пример применения LIT-NLP для обеспечения интерпретируемости модели T5, выполняющей задачу суммаризации, представлен на рис. 5.10.

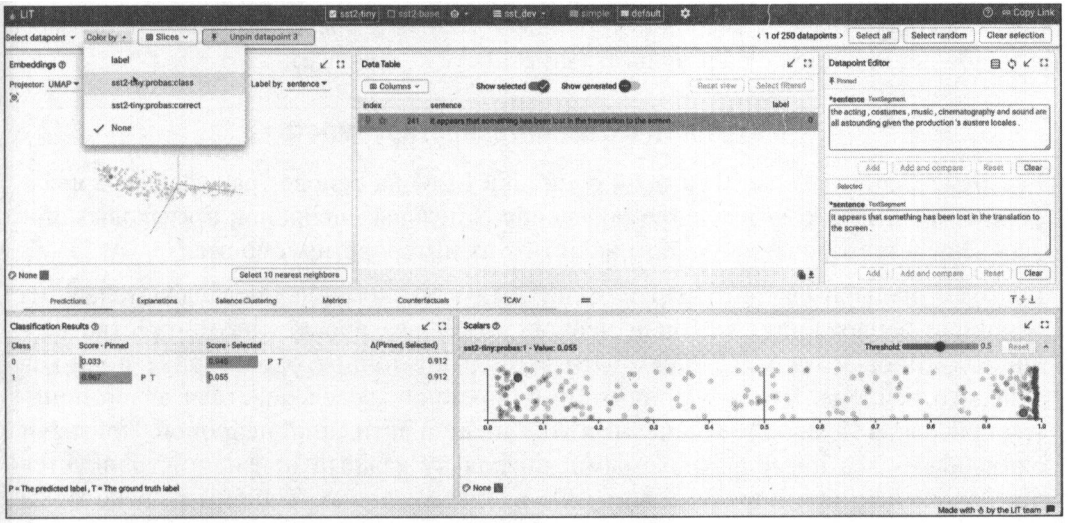



**Рис. 5.10.** LIT-NLP

LIT-NLP предлагает ряд функций, которые упрощают процесс отладки моделей. Среди них:

- визуализация механизма внимания;
- карты заметности (salience map), демонстрирующие фрагменты входных данных, на которые модель обращает наибольшее внимание;
- визуализация эмбеддингов;
- контрфактуальный анализ, позволяющий понять, как изменяется поведение модели при изменении входных данных, например при добавлении или исключении токена.

### Упражнение

Классифицируйте предложения из набора данных с записями заседаний парламента Канады, который можно найти в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Используя технику промптинга с несколькими примерами, предоставьте модели Google Gemma (или любой другой) примеры предложений с различными метками: поддерживающими, враждебными, печальными, радостными и др. Важно отметить, что достижение 100% точности с первого раза может оказаться непростой задачей. Проанализируйте ошибки с помощью LIT-NLP и выясните, могут ли инструменты интерпретируемости способствовать сбору аналитических данных для повышения точности модели.

Дополнительную информацию о применении LIT-NLP для анализа ошибок можно найти в руководстве Google (**https://oreil.ly/zcsLu**). В нем описывается использование LIT-NLP совместно с языковой моделью Gemma LLM для обнаружения ошибок в промптах "с несколькими выстрелами" (примерами). Подход предполагает анализ некорректных примеров и изучение того, какие элементы промпта оказывают наибольшее влияние на результат (заметность — salience).

### Механистическая интерпретируемость

Как отмечалось в *главе 2*, базовой единицей LLM на основе трансформера является нейрон. Поэтому изучение поведения отдельных нейронов в языковых моделях является ключевым этапом на пути к их интерпретируемости.

В ходе экспериментов (**https://oreil.ly/hLdVN**) команда исследователей из Anthropic обнаружила, что один нейрон может реагировать на множество различных типов входных данных. Это затрудняет точное определение роли отдельного нейрона. Чтобы улучшить интерпретацию, исследователи ввели понятие признаков (features) — линейных комбинаций активаций нейронов. Признаки оказались более интерпретируемыми, поскольку каждый признак активируется исключительно при определенном типе входных данных. Например, некоторые активируются только при появлении одного конкретного токена, в то время как другие проявляются при более универсальных типах ввода (пример — программный код).

Для детального изучения темы рекомендуется ознакомиться с исследованием ученых из Anthropic, посвященным механистической интерпретируемости. В статье (**https://oreil.ly/hLdVN**) авторы проводят эксперименты на однослойном модуле трансформера, анализируя ключевые признаки и их влияние.

Дополнительную информацию можно найти по ссылке **https://oreil.ly/2YvE6** — здесь представлено использование инструмента визуализации Anthropic, который сопровождается текстовыми описаниями токенов, активирующих нейроны. В качестве примера приводится исследование (**https://oreil.ly/cE27M**), в котором показано, как различные нейроны реагируют на ввод книги "Алиса в Стране чудес".

# Резюме

В этой главе мы сделали обзор ландшафта больших языковых моделей (LLM), изучив доступные версии и их особенности. Было показано, как выявлять ключевые критерии, наиболее значимые для наших задач, и на основе этого выбирать оптимальную модель. Мы рассмотрели популярные бенчмарки для LLM и способы анализа их результатов, узнали, как загружать модели, осуществлять с их помощью выводы или инференсы, а также применять эффективные стратегии декодирования. В завершение мы познакомились с инструментами для оценки интерпретируемости модели, такими как LIT-NLP, которые помогают глубже понять процессы внутри архитектуры трансформера.

В следующей главе мы подробно рассмотрим, как обновить модель, чтобы она лучше справлялась с поставленными задачами. Мы разберем полноценный пример дообучения и обсудим оптимальные решения по настройке гиперпараметров. Также мы научимся создавать собственные наборы обучающих данных для дообучения модели.

# ГЛАВА 6
# Дообучение

В предыдущей главе мы обсудили ключевые аспекты, которые следует учитывать при выборе большой языковой модели (LLM), подходящей для различных целей. Мы также рассмотрели методы оценки LLM, которые помогут принять более обоснованные решения. Теперь пришло время перейти к практическому применению наших знаний и использовать LLM для решения конкретных задач.

В этой главе мы подробно рассмотрим, как адаптировать LLM для решения определенных задач с помощью техники дообучения. Мы разберем детальный пример дообучения, который наглядно продемонстрирует все ключевые решения, требующие особого внимания. Кроме того, мы изучим, как эффективно создавать специализированные наборы данных для подобных процедур.

## Актуальность дообучения

Для чего необходимо дообучение (тонкая настройка, fine-tuning)? Почему использования лишь предобученной языковой модели и нескольких запросов недостаточно для решения поставленных задач? Для иллюстрации рассмотрим конкретные примеры.

*Пример 1*

Допустим, перед нами стоит нетривиальная задача: найти в тексте все предложения, которые написаны в прошедшем времени, и перефразировать их в будущем времени. Чтобы решить эту задачу, можно составить несколько примеров предложений в прошедшем времени и создать пары "вход — выход", где входом будет предложение в прошедшем времени, а выходом — соответствующее предложение в будущем. Правда, кажется, что LLM не справляется с задачей на должном уровне, допуская ошибки как при идентификации, так и при трансформации. Чтобы улучшить результаты, необходимо детально описать инструкции, добавив в промпт грамматические правила и исключения на английском языке. После этого производительность заметно возрастает. Однако с каждым новым правилом промпт становится все более сложным, постепенно превращаясь в объемный документ, напоминающий грамматический справочник.

Как было отмечено в *главе 5*, LLM может выполнять лишь ограниченное количество инструкций, заданных в промпте, а его реальное контекстное окно значительно меньше заявленного. Это создает серьезную проблему.

*Пример 2*

Рассмотрим задачу, связанную с ответами на вопросы по содержанию финансового текста. LLM не являются экспертами в области финансов и могут испытывать трудности с пониманием профессионального финансового жаргона. Чтобы решить эту проблему, необходимо добавить в промпт определения ключевых финансовых терминов. Хотя это приводит к небольшому улучшению производительности, становится ясно, что для достижения желаемого результата потребуется включить в контекстное окно всю учебную программу экзамена CPA (Certified Public Accountant — сертификат специалиста в области финансового учета).

В таких случаях необходимо провести дообучение. Модель получает набор пар "ввод — вывод" и изучает соответствие между ними, обновляя свои веса. Это позволяет ей решать задачи, которые не могут быть выполнены только с помощью контекстного обучения. В обеих вышеупомянутых задачах дополнительное обучение значительно повышает эффективность модели.

Когда не следует использовать дообучение? Если цель — интегрировать новые знания в языковую модель, то предпочтительнее применять методы, такие как RAG, которые будут подробно описаны в *главах 10* и *12*. Дообучение обычно оказывается более эффективным, когда нужно обучить модель сопоставлению ввода-вывода, познакомить ее с новой тематикой или развить у нее определенные навыки и алгоритмы работы.



Как было отмечено в *главе 5*, обновление параметров языковой модели может иметь негативные последствия, в частности, снизить общие способности базовой модели. Дообучение модели на конкретной задаче может непреднамеренно ухудшить ее производительность в других областях. Поэтому к данному процессу следует подходить с особой осторожностью.

## Дообучение: подробный пример

Рассмотрим практический пример дообучения модели с нуля. Предположим, необходимо создать модель *детектора политических обещаний* (political promises detector), которая будет способна выявлять обещания, данные представителями правящей партии в ходе предвыборных кампаний или парламентских дебатов. Политическое обещание в данном контексте — это конкретное и выполнимое обязательство, которое находится в компетенции правительства.

В качестве иллюстрации можно привести следующее предложение: "В следующем десятилетии мы построим 10 000 километров путей метро".

Следует подчеркнуть, что не все высказывания, относящиеся к будущему времени или предсказания, следует трактовать как обещания. Ниже приведены примеры, которые, по нашему мнению, не соответствуют критериям обещания:

"Мы ожидаем, что в следующем году Япония повысит тарифы". (Ожидание, но не действие под контролем правительства.)

"Мы будем работать над тем, чтобы сделать Канаду лучше". (Отсутствуют конкретные детали.)

"ИИ станет причиной потери миллиона рабочих мест в следующем году". (Предсказание, а не обещание.)

Базовая модель Llama2-7B, ориентированная на контекстное обучение, испытывает трудности с точным распознаванием политических обещаний. Чтобы решить эту проблему, модель необходимо дообучить для выполнения этой задачи. В результате мы получим модель, способную выявлять политические обещания и сравнивать их со структурированными наборами данных или бюджетными документами. Это позволит отслеживать выполнение этих обещаний в течение определенного периода времени.

Для достижения этой цели был составлен набор синтетических данных для дообучения, включающий в себя примеры как обещаний, так и обычных утверждений. Подробности создания такого набора данных будут рассмотрены по ходу главы.

Сегодня дообучение стало проще благодаря появлению множества библиотек, которые значительно ускоряют этот процесс. Среди наиболее популярных стоит отметить Transformers (**https://oreil.ly/BTi76**), Accelerate (**https://oreil.ly/W8oLi**), PEFT (**https://oreil.ly/QbQoq**), TRL (**https://oreil.ly/Ya9Xj**), bitsandbytes (**https://oreil.ly/ruVEX**). Первые четыре из них созданы компанией Hugging Face, и многие из них уже упоминались в предыдущих главах. Понимание принципов работы этих библиотек является весьма ценным навыком.



Поскольку эти библиотеки являются молодыми и быстро развивающимися, они постоянно обновляются. Поэтому важно следить за основными изменениями в них, т. к. они регулярно внедряют улучшения, которые оптимизируют рабочий процесс.

Начнем с загрузки набора данных. Пользовательский набор данных доступен в репозитории GitHub (**https://oreil.ly/llm-playbooks**):

```
from datasets import load_dataset
tune_data = load_dataset("csv", data_files='/path/to/finetune_data.csv')
```



Настоятельно рекомендуем воспользоваться библиотекой наборов данных (**https://oreil.ly/3LX5X**) для загрузки обучающих и дообучающих наборов данных. Эта библиотека предоставляет удобную абстракцию, которая позволяет эффективно работать с большими объемами данных, не требуя глубокого понимания тонкостей управления памятью.

Далее необходимо установить несколько важных гиперпараметров в библиотеке `Transformers` с помощью класса `TrainingArguments`:

```
# Убедитесь, что вы установили корректную версию
!pip install transformers==4.35.0

from transformers import TrainingArguments
```

В нашем распоряжении более сотни аргументов; мы подробно рассмотрим самые значимые из них. Эти аргументы позволяют настраивать алгоритмы обучения, оптимизировать использование памяти и пространства, выполнять квантование, регулировать процесс обучения и проводить его в распределенной среде. Рассмотрим каждый из них подробнее.

## Параметры обучающих алгоритмов

Рассмотрим алгоритмы оптимизации, используемые в процессе обучения нейронных сетей, и научимся выбирать самые подходящие для решения конкретных задач.

### Оптимизаторы

На сегодняшний день самыми популярными оптимизаторами являются AdamW и Adafactor. К числу других часто используемых алгоритмов оптимизации можно отнести стохастический градиентный спуск (stochastic gradient descent, SGD), RMSProp, Adagrad, Lion и их модификации. Более подробную информацию об алгоритмах оптимизации можно найти в блоге Флориана Джуна (Florian June, **https://oreil.ly/VTiDa**).

Adafactor и SGD требуют 4 байта памяти на каждый параметр, в то время как AdamW — 8 байт. Это означает, что для полного дообучения модели 7B с использованием оптимизатора AdamW потребуется около 56 Гбайт памяти только для хранения состояний оптимизатора ($7 \times 8 = \sim 56$ Гбайт). Если учесть также память для параметров, градиентов и активных величин, общая потребность в памяти значительно возрастет.

Недавно были представлены 8-битные оптимизаторы, которые позволяют квантовать состояние алгоритма (**https://oreil.ly/4Z14D**). При полном дообучении модели 7B с использованием 8-битной версии AdamW объем памяти, необходимый для хранения состояния оптимизатора, уменьшается примерно до 14 Гбайт.

Библиотека bitsandbytes предлагает доступ к этим оптимизаторам, работающим с 8-битной точностью, которые также поддерживаются платформой Hugging Face. Чтобы активировать 8-битную версию AdamW во время обучения, необходимо указать соответствующий аргумент в `Trainingarguments` (обучающие аргументы):

```
optim = 'adamw_bnb_8bit'
```

Все параметры оптимизатора, доступные непосредственно через Hugging Face, см. в классе `OptimizerNames` (**https://oreil.ly/7kdSO**).



В ходе проведения бенчмаркинга Стас Бекман (**https://oreil.ly/0_0lt**) выявил интересную закономерность: оптимизатор AdamW в 8-битной реализации функционирует значительно быстрее, чем в стандартном варианте. Кроме того, результаты экспериментов продемонстрировали, что Adafactor уступает AdamW в скорости выполнения задач.

В классе `TrainingArguments` (Hugging Face) по умолчанию используется оптимизатор AdamW, который зарекомендовал себя как высокоэффективный. Однако, если этот оптимизатор не соответствует вашим требованиям, вы можете рассмотреть альтернативные варианты, такие как Adafactor или Lion. Для задач обучения с подкреплением (reinforcement learning, RL) рекомендуется использовать стохастический градиентный спуск (SGD).

Если у вас ограничено количество доступной памяти, стоит обратить внимание на 8-битную версию AdamW, которая поможет снизить потребление ресурсов. Также можно рассмотреть страничные версии этих оптимизаторов, которые способны еще больше уменьшить нагрузку на память.

### Оптимизаторы страниц

Оптимизатор AdamW требует всего 8 байт памяти на каждый параметр, что значительно снижает нагрузку на систему. Однако это также влияет на максимальную поддерживаемую длину последовательности.

В таких случаях могут быть полезны так называемые страничные оптимизаторы, или оптимизаторы страниц (paged optimizer). Если в процессе дообучения заканчивается память графического процессора (GPU), эти оптимизаторы автоматически переносят страницы памяти в оперативную память центрального процессора (CPU RAM), а затем возвращают их обратно в память GPU, когда это необходимо.

В Hugging Face доступны варианты с разбиением на страницы для оптимизаторов AdamW и Lion. Их можно найти, используя имена оптимизаторов: `paged_adamw_32bit`, `paged_adamw_8bit`, `paged_lion`, `paged_lion_8bit`.

## Скорость обучения

Для каждого оптимизатора существуют определенные параметры, которые показывают высокую результативность. В частности, для AdamW рекомендованная скорость обучения (learning rate) составляет $10^{-4}$ со снижением веса 0,01. Снижение веса (weight decay) представляет собой метод регуляризации, направленный на уменьшение риска переобучения модели. Кроме того, стандартные значения второстепенных параметров, таких как `adam_beta1`, `adam_beta2` и `adam_epsilon`, являются достаточно оптимальными и не требуют дополнительных корректировок.

### Упражнение

Параметры скорости обучения при дообучении моделей могут отличаться от тех, что применяются при обучении нейронных сетей с нуля. Чтобы лучше разо-

браться в данной теме, рекомендуется ознакомиться с работой Цзин и соавт. "Переосмысление настройки скорости обучения в эпоху больших языковых моделей" (Jin et al., **https://oreil.ly/T1xNB**). В работе представлен подробный обзор лучших практик, разработанных сообществом исследователей в области больших языковых моделей.

Также стоит рассмотреть возможность использования автоматизированных инструментов для оптимизации скорости обучения. Например, можно воспользоваться LearningRateFinder из PyTorch Lightning (**https://oreil.ly/_zdA9**).

## План обучения

В конце обучения рекомендуется снижать скорость, чтобы избежать переобучения модели, когда она приближается к точке сходимости. Не менее важно не допустить переобучения на ранних этапах, когда обрабатываются первые несколько пакетов данных. В любом случае было бы удобно автоматически регулировать скорость обучения в зависимости от текущей стадии процесса. Для упрощения этой задачи можно настроить план обучения (learning schedule).

В Hugging Face есть несколько типов планировщиков (scheduler), каждый из которых обладает своими уникальными особенностями. Рассмотрим несколько наиболее значимых.

*Постоянная скорость обучения (constant)*

План обучения, основанный на классической (ванильной) стратегии, при которой на протяжении всего периода скорость обучения остается постоянной.

*Постоянная скорость с прогревом (constant with warmup)*

В этом режиме начальная скорость обучения равна нулю и постепенно увеличивается до заданного значения в течение фазы прогрева. После завершения фазы скорость обучения остается на постоянном уровне.

На рис. 6.1 показано, как скорость обучения изменяется с течением времени при использовании планировщика с прогревом (warmup scheduler).

*Косинусный отжиг (cosine annealing)*

В этом методе скорость обучения сначала увеличивается, а затем постепенно уменьшается до нуля в соответствии с косинусной функцией.

На рис. 6.2 показана динамика скорости обучения при использовании косинусного планировщика (cosine scheduler).

*Косинусный отжиг с перезапусками (cosine with restarts)*

Этот метод заключается в том, чтобы после фазы прогрева постепенно уменьшать скорость обучения до нуля по косинусной функции. В начале скорость обучения постепенно растет, достигая начального значения, а затем плавно снижается до нуля, следуя за графиком косинусной функции. Однако в процессе обучения происходит несколько "жестких" перезапусков, когда скорость обучения резко увеличивается до заданного значения после достижения нулевого

уровня. Эта методика была впервые предложена Лошчиловым и Хаттером (Ilya Loshchilov & Frank Hutter). Ее подробное описание можно найти по ссылке **https://oreil.ly/Q4c3o**.




**Рис. 6.1.** Динамика скорости обучения при использовании планировщика непрерывного обучения с прогревом




**Рис. 6.2.** Динамика скорости обучения при использовании косинусного планировщика

Динамика скорости обучения при использовании косинусного планировщика с перезапусками представлена на рис. 6.3.

*Линейный планировщик (linear scheduler)*

Линейный планировщик аналогичен косинусному планировщику, но отличается тем, что скорость обучения уменьшается до нуля не по косинусной, а по линейной функции.




**Рис. 6.3.** Динамика скорости обучения при использовании косинусного планировщика с перезапусками

Динамика скорости обучения при использовании линейного планировщика представлена на рис. 6.4.




**Рис. 6.4.** Динамика скорости обучения при использовании линейного планировщика

При использовании оптимизатора AdamW особенно важно применять планировщики с прогревом, чтобы избежать попадания в так называемые *ловушки плохих минимумов* (bad minima). Как показала практика, косинусный отжиг является более эффективным инструментом, чем линейное снижение скорости.

Для дообучения детектора политических обещаний (political promises detector) мы будем использовать страничную модификацию оптимизатора AdamW. Скорость обучения будет составлять $3 \times 10^{-4}$, снижение веса — 0,01, планировщик обучения — косинусный.

```
optim = "paged_adamw_32bit"
learning_rate = 3e-4
weight_decay = 0.01
lr_scheduler_type = 'cosine'
warmup_ratio = 0.03 # Доля этапов обучения, которые будут использованы
                    # в качестве прогрева
```

## Параметры оптимизации памяти

После настройки параметров, касающихся оптимизаторов, проводится анализ памяти и вычисляются оптимизационные показатели. Среди наиболее часто используемых методов в данной сфере выделяются контрольные точки градиента (gradient checkpointing) и накопление градиентов (gradient accumulation).

### Контрольная точка градиента

Метод контрольной точки градиента представляет собой эффективный способ оптимизации использования памяти, который достигается за счет увеличения количества вычислений. В стандартной процедуре прямого прохода (forward pass) для алгоритма обратного распространения активации значения рассчитываются и сохраняются в памяти для последующего использования на этапе обратного прохода (backward pass). Однако, если не сохранять все эти активации, можно пересчитать их заново непосредственно во время обратного прохода. Такой подход требует больше вычислений, но значительно сокращает использование памяти. Это особенно полезно для обучения моделей, размер пакетов которых не соответствует объему памяти графического процессора. Например, можно обрабатывать пакеты даже с минимальным размером в один элемент. Более подробное техническое описание метода контрольной точки градиента можно найти в блоге Ярослава Булатова (**https://oreil.ly/i-R4I**).

### Накопление градиентов

Если необходимый размер пакета больше доступного объема памяти, можно использовать метод, известный как накопление или аккумулирование градиентов (gradient accumulation), чтобы имитировать желаемый размер партии. В этом подходе обновления градиента выполняются не для каждого пакета, а накапливаются в нескольких пакетах, а затем суммируются или усредняются.



Накопление градиентов способно замедлить процесс обучения, поскольку количество выполняемых обновлений уменьшается. При этом объем необходимых вычислений остается неизменным.

### Квантование

Одной из самых эффективных стратегий оптимизации памяти является квантование (*см. главу 5*). Мы подробно обсудим методы квантования в *главе 9*. В нашем случае мы будем использовать формат BF16, который предлагает разумный баланс между потреблением памяти и производительностью.

Для дообучения детектора политических обещаний (учитывая ограниченный объем памяти GP RAM в 16 Гбайт) установим следующие параметры оптимизации памяти:

```
gradient_accumulation_steps = 4
bf16 = True
gradient_checkpointing = True
```

## Параметры регуляризации

Теперь рассмотрим подробно ключевые методы, которые позволяют эффективно решать проблему переобучения моделей.

### Сглаживание меток

Сглаживание меток (label smoothing) — это метод, который помогает избежать переобучения модели и способствует улучшению ее калибровки. Калибровка модели играет важную роль в ее работе, и ее качество можно оценить по тому, насколько точно модель предсказывает результаты задачи.

В качестве примера можно рассмотреть задачу классификации предложений по признаку оскорбления. Если модель хорошо откалибрована, то из всех случаев, для которых она устанавливает вероятность 0,9, около 90% будут правильно классифицированы. Аналогично для вероятности 0,6 предполагается, что модель будет правильно классифицировать около 60% случаев. То есть выходная вероятность должна соответствовать уровню уверенности модели в классификации.

Модель, настроенная должным образом, предполагает отсутствие излишней самоуверенности в своих результатах. Такой подход позволяет более тонко и эффективно обрабатывать случаи с низкой вероятностью успеха, например, при использовании крупной модели для анализа данных, имеющих подобную специфику.



Согласно исследованию, проведенному Ли и соавт. (Li et al., **https://oreil.ly/ij7mS**), крупные языковые модели демонстрируют более низкий уровень калибровки по сравнению с такими моделями, как BERT. Большие модели часто проявляют чрезмерную уверенность в своих предсказаниях. Недостаточная точность определения степени неопределенности у таких масштабных моделей может стать аргументом в пользу использования языковых моделей меньшего размера.

Одним из способов калибровки моделей является сглаживание меток. Обычно процесс обучения происходит на основе точных целевых значений, таких как 0 и 1 для

бинарной классификации. Если функция потерь (loss function) представляет собой перекрестную энтропию (cross-entropy), модель стремится к логитам, близким к 0 или 1, что делает ее слишком уверенной в своих прогнозах.

Сглаживание меток предполагает введение дополнительного элемента регуляризации, который корректирует эти жесткие целевые значения, изменяя их или масштабируя. Это позволяет избежать чрезмерного обучения модели на примерах, которые могут быть неточными или ошибочными.

Особенно важно применять сглаживание меток, когда исходный набор данных содержит шумы или ошибочные метки. Такая регуляризация предотвращает обучение модели на примерах, которые не соответствуют действительности, что позволяет улучшить качество предсказаний.

Эту методику следует использовать для анализа политических обещаний, т. к. некоторые высказывания могут быть субъективными или иметь несколько интерпретаций.

### Эмбеддинги с шумом

Существует относительно небольшое количество примеров для обучения модели (менее 50 000). Важно, чтобы модель не перенимала слишком явные стилистические особенности этих данных, такие как форматирование, формулировки или объем текста. Одним из эффективных способов избежать этого является добавление шума во входные эмбеддинги.

В исследовании Джайна и соавт. (Jain et al., **https://oreil.ly/ouESL**) было обнаружено, что использование эмбеддингов с шумом (или шумовой эмбеддинг — noise embedding) помогает снизить склонность модели чрезмерно фокусироваться на формулировках и форматировании обучающих данных. Интересной особенностью данного подхода является то, что модели начинают создавать более длинные и содержательные тексты. Анализ разнообразия токенов в полученных данных показал, что такие расширенные тексты действительно включают больше уникальной информации, а не являются повторениями одного и того же содержания.

Hugging Face предлагает метод дообучения, основанный на технике добавления шума, который называется NEFTune (Noisy Embedding Instruction Fine-Tuning, **https://oreil.ly/dSaem**). Данная техника заключается в том, что к каждому вектору эмбеддинга добавляется вектор шума, созданный на основе независимых и одинаково распределенных случайных величин (independent and identically distributed, IID). Эти величины выбираются из диапазона [–1, 1], а затем масштабируются с помощью коэффициента масштабирования.

Эмпирически доказано, что добавление шума к векторам эмбеддинга значительно снижает риск переобучения модели. Поэтому данный метод рекомендуется для дообучения детектора политических обещаний. Следует подчеркнуть, что шум добавляется только в процессе обучения, а не во время вывода модели.



Воздействие эмбеддингов с шумом на работу модели пока не до конца изучено. В процессе дообучения можно попытаться улучшить модель, используя дополнительные ресурсы. Однако перед этим необходимо удостовериться, что в ее работе не произошло каких-либо регрессий.

В рамках задачи по дообучению детектора политических обещаний мы применяем методы сглаживания меток и добавления шума:

```
# Метка 0 будет преобразована к label_smoothing_factor/num_labels
# Метка 1 будет преобразована к 1 - label_smoothing_factor +
# label_smoothing_factor/num_labels

label_smoothing_factor = 0.1
neftune_noise_alpha = 5
```

## Размер пакета данных

Размер пакета (batch size) является одним из ключевых гиперпараметров (наряду со скоростью обучения), который требует тщательного выбора, может ускорить процесс обучения, но также способен привести к увеличению требований к объему оперативной памяти, что может стать ограничивающим фактором. Кроме того, использование больших пакетов может увеличить вероятность попадания модели в резкие локальные минимумы (local minima), что может свидетельствовать о переобучении. Таким образом, выбор размера пакета представляет собой компромисс между объемом потребляемых ресурсов, доступной вычислительной мощностью и эффективностью модели.

Для настройки детектора политических обещаний был выбран размер пакета равный 8, т. к. он соответствует текущим ограничениям по объему памяти. На этапе вывода рекомендуется использовать максимальный размер пакета, доступный для системы. Важно, чтобы он был кратным двум, чтобы минимизировать накладные расходы на обработку ввода-вывода графическим процессором.

На платформе Hugging Face класс `TrainingArguments` предлагает удобный параметр `auto_find_batch_size`, который автоматически определяет максимально возможный размер пакета, исходя из доступной памяти. Чтобы воспользоваться этой функцией, необходимо предварительно установить библиотеку `accelerate`.

```
per_device_train_batch_size = 8
per_device_eval_batch_size = 8
```



Для поддержки более крупных пакетов можно сократить максимальную длину последовательности.

### Связь между скоростью обучения и размером пакета данных

Связь между скоростью обучения и размером пакета представляет собой сложную зависимость, которая определяется множеством внешних факторов, включая архитектуру модели.

Высокая скорость обучения позволяет сократить количество шагов, что способствует более быстрому завершению процесса, но при этом несет риск превышения оптимального минимума, что может привести к отсутствию сходимости. С другой стороны, низкая скорость обучения требует большего количества шагов и больше времени для достижения сходимости, но может привести модель к узкому неоптимальному минимуму. Такие узкие локальные минимумы часто связаны с переобучением. Оптимальным подходом считается стремление к более плоским минимумам, что возможно при увеличении скорости обучения. Кроме того, уменьшение размера пакета может вызывать увеличение разброса между примерами в каждой партии, что потенциально может привести к более плоским минимумам модели.

Таким образом, сочетание относительно высокой скорости обучения и небольшого размера пакета теоретически может повысить эффективность процесса сходимости. Однако важно отметить, что не всегда теоретические предположения подтверждаются на практике.

В завершение рассмотрим несколько ключевых параметров.

`max_grad_norm`

Этот параметр используется для обрезки градиента (gradient clipping) и предотвращения проблемы взрывных градиентов (exploding gradients), которая может возникнуть в процессе обучения. Он определяет пороговое значение нормы градиента L2, при превышении которого градиент перемасштабируется до установленной величины `max_grad_norm`. Дополнительные детали о принципах работы и решении проблем, связанных с обрезкой градиента, можно найти в материале "Общие сведения об обрезке градиента (и способы решения проблемы взрывных градиентов)" ("Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem)", **https://oreil.ly/gH7L7**).

`group_by_length`

Данный параметр позволяет группировать примеры одинаковой длины в одном пакете, что помогает минимизировать количество токенов, добавляемых для заполнения.

`max_train_epochs`

Количество эпох для обучения на тренировочном наборе данных. Обычно рекомендуется задавать значение менее пяти, чтобы уменьшить риск переобучения:

```
max_grad_norm=2
group_by_length=True
max_train_epochs=3
```

## Эффективная тонкая настройка параметров

После заполнения `TrainingArguments` следующим шагом будет настройка параметров библиотеки PEFT (Parameter-Efficient Fine-Tuning — эффективная тонкая настройка параметров или параметрически эффективное дообучение). Эта библиотека, разработанная компанией Hugging Face, представляет собой мощный инструмент для эффективного дообучения моделей. Подход основан на методах, которые позволяют обновлять только часть параметров, при этом достигая производительности, сравнимой с обновлением всей модели.

В данном примере для дообучения будет использоваться метод низкоранговой адаптации (low-rank adaptation, LoRA).

Рассмотрим ключевые гиперпараметры, на которые следует обратить особое внимание.

`r`

Оценка внимания LoRA.

`lora_alpha`

Параметр альфа, используемый в методе LoRA.

`lora_dropout`

Вероятность выпадения, применяемая в настраиваемых слоях. Помогает предотвратить переобучение.

`layers_to_transform`

Определяет слои, к которым будет применено преобразование LoRA.

Рекомендуется устанавливать по умолчанию следующие значения:

```
r = 64
lora_alpha = 8
lora_dropout = 0.1
```

Дополнительную информацию о технике LoRA можно найти в блоге Огбана Угота (Ogban Ugot, **https://oreil.ly/_l91y**).

## Работа с пониженной точностью

Библиотека bitsandbytes, разработанная Тимом Деттмерсом (Tim Dettmers), значительно упрощает использование форматов с пониженной точностью, рассмотренных в *главе 5*. В данном примере будет использоваться формат FP4. Для корректной работы требуется версия bitsandbytes 0.39.0 и выше.

Hugging Face интегрировала поддержку bitsandbytes в свою экосистему, что делает эту технологию еще более доступной. Для настройки параметров можно воспользоваться классом `BitsAndBytesConfig`. Рассмотрим несколько ключевых опций.

`load_in_8bit/load_in_4bit`

Этот параметр позволяет выбрать режим, в котором будет загружена модель: 4- или 8-битный.

`llm_int8_threshold`

Устанавливает пороговые значения, после которых используется FP16, т. к. квантование в формате int8 эффективно только для значений меньше 5–6.

`llm_int8_skip_modules`

Позволяет указать исключения, для которых квантование int8 применяться не будет.

`llm_int8_enable_fp32_cpu_offload`

Обеспечивает запуск части модели в формате int8 на GPU и оставшихся частей в FP32 на CPU. Используется, когда модель слишком объемна для размещения полностью на GPU.

`bnb_4bit_compute_dtype`

Определяет тип вычислений независимо от входного формата данных.

`bnb_4bit_quant_type`

Предлагает выбрать тип квантования (FP4 или NF4) для 4-битных слоев.

Рекомендуемые значения по умолчанию:

```
use_4bit = True
bnb_4bit_compute_dtype = 'float16'
bnb_4bit_quant_type = 'nf4'
use_nested_quant = False
```

В завершение воспользуемся библиотекой TRL (transformer reinforcement learning — обучение трансформера с подкреплением). Эта библиотека не только поддерживает обучение с подкреплением, но и предлагает инструменты для контролируемого дообучения моделей.

Рекомендуемые значения по умолчанию:

```
max_seq_length = 128
# Для размещения нескольких инструкций в одной и той же входной последовательности
# применяется метод сжатия

packing = True
```

## Сводим все воедино

После настройки всех необходимых параметров рассмотрим полный код для процесса дообучения:

```
# Проверьте, чтобы были установлены требуемые версии библиотек.
!pip install transformers==4.35.0 accelerate==0.24.0 peft==0.6.0
bitsandbytes==0.41.0 trl==0.7.4

from datasets import load_dataset
from transformers import TrainingArguments, BitsAndBytesConfig
```

```
from transformers import LlamaForCausalLM, LlamaTokenizer
from peft import PeftModel, LoraConfig
from trl import SFTTrainer

train_params = TrainingArguments(
    optim="paged_adamw_32bit",
    learning_rate=3e-4,
    weight_decay=0.01,
    lr_scheduler_type='cosine',
    warmup_ratio=0.03,
    gradient_accumulation_steps=4,
    bf16=True,
    gradient_checkpointing=True,
    label_smoothing_factor=0.1,
    neftune_noise_alpha=5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    max_grad_norm=2,
    group_by_length=True,
    max_train_epochs=3,
    output_dir='/model_outputs',
    save_steps=50,
    logging_steps=10
    )

quantize_params = BitsAndBytesConfig (
    use_4bit=True,
    bnb_4bit_compute_dtype='float16',
    bnb_4bit_quant_type='nf4',
    use_nested_quant=False
    )

lora_params = LoraConfig (
    r=64,
    lora_alpha=8,
    lora_dropout=0.1
    )
```

```
model = LlamaForCausalLM.from_pretrained(
    pretrained_model_name_or_path='meta-llama/Llama-2-7b',
    quantization_config=quantize_params,
    device_map='auto'
    )

tokenizer = LlamaTokenizer.from_pretrained('meta-llama/Llama-2-7b')

tune_data = load_dataset("csv", data_files='/path/to/finetune_data.csv')

sft = SFTTrainer (
    model=model,
    args=train_params,
    train_dataset=tune_data,
    tokenizer=tokenizer
    peft_config=lora_params,
    max_seq_length=128,
    dataset_text_field='text',
    packing=True
    )

sft.train()
sft.model.save_pretrained('/path/to/llama-2-it.csv')
```

Связь между гиперпараметрами достаточно сложна, и их настройка может привести к неожиданным результатам. Для достижения оптимального баланса потребуется провести несколько итераций. Впрочем, не стоит слишком увлекаться попытками извлечь максимальную производительность из процесса дообучения — лучше направить усилия на создание более качественных обучающих данных. В следующем разделе мы рассмотрим методы формирования эффективных наборов данных для обучения.

Объем памяти, необходимый для дообучения LLM, определяется множеством факторов, включая выбранный оптимизатор, использование накопления градиента и контрольных точек градиентов, а также тип применяемого квантования и другие параметры.

### Упражнение

В экспериментах, связанных с машинным обучением, исследование абляции (ablation studies) играет ключевую роль. Этот метод позволяет оценить влияние каждого отдельного компонента на общую производительность модели, ис-

ключая его и повторяя моделирование. Рассмотрим на примере дообучения, как добавление шума может сказаться на работе модели. Проведите пять запусков дообучения с активированными эмбеддингами с шумом и столько же запусков без шума, сохраняя остальные гиперпараметры неизменными. Затем проанализируйте ошибки на тестовом наборе данных, чтобы понять, как добавление шума влияет на качество работы модели. Дает ли это исключительно положительный результат?

## Дообучение наборов данных

В рассмотренном примере дообучения акцент был сделан на использовании предварительно подготовленного набора данных, что позволило сосредоточиться на самом процессе дообучения. Теперь обратимся к характеристикам самих наборов данных и рассмотрим различные способы их создания.

Прежде всего, подробно изучим набор данных, который использовался в примере:

```
from datasets import load_dataset
tune_data = load_dataset("csv", data_files='/path/to/finetune_data.csv')
print(tune_data[:2])
```

Вывод:

```
Input: We will support women and children and give every child the best possible start
with $10 a day child care.

Identify if the above sentence represents a political promise. A political promise is
a promise that is tangible, specific, and an action that the government has the agency
to make. Reply 'True' if the sentence represents a political promise, 'False' if not.

Output: True

Input: It is time for leadership that never seeks to divide Canadians, but takes every
single opportunity to bring us together, including in Parliament.

Identify if the above sentence represents a political promise. A political promise is
a promise that is tangible, specific, and an action that the government has the agency
to make. Reply 'True' if the sentence represents a political promise, 'False' if not.

Output: False
```

*(Вход: Мы окажем поддержку женщинам и детям, обеспечивая каждому ребенку наилучшие стартовые условия с помощью ежедневной выплаты в размере 10 долларов на уход.*

*Определить, является ли это предложение политическим обещанием. Политическое обещание - это обещание, которое можно ощутить, оно конкретное и связано с действиями, доступными правительству. Ответить "True", если предложение представляет собой политическое обещание, и "False", если нет.*

*Выход: True*

*Вход: Пришло время для руководства, которое никогда не стремится разделить канадцев, но использует любую возможность, чтобы объединить нас, включая выступления в парламенте.*

*Определить, является ли это предложение политическим обещанием. Политическое обещание - это обещание, которое можно почувствовать, оно конкретное и связано с действиями, доступными правительству. Ответить "True", если предложение представляет собой политическое обещание, и "False", если нет.*

*Выход: False)*

Этот набор данных отличается от традиционного подхода, где присутствуют четко определенные пары "вход — выход". Здесь дополнительно содержится описание задачи на естественном языке. Типичный элемент такого набора для дообучения включает в себя:

- инструкцию, подробно описывающую задачу и указывающую предпочтительный формат вывода. В инструкции могут быть приведены положительные и/или отрицательные примеры выполнения задачи, а также перечислены ограничения и исключения, которые необходимо учитывать;
- необязательный вход, который представляет собой текст (например, предложение или абзац), предназначенный для оценки модели;
- выход в виде правильного ответа на задание, оформленного в соответствии с требованиями, указанными в инструкции.



Дообучение моделей на наборах данных можно организовать как многозадачное или однозадачное. Многозадачные наборы данных обычно применяются для обучения модели выполнению различных инструкций. Такой вид обучения часто рассматривается как промежуточный этап перед переходом к узкоспециализированному дообучению под конкретную задачу. Например, с помощью FLAN можно обучить языковую модель T5 следованию инструкциям и создать модель FLAN-T5. После этого модель дообучается, используя набор данных, специально составленный для определенной задачи. Этот подход, согласно исследованиям (**https://oreil.ly/e-MVh**), демонстрирует более высокую эффективность по сравнению с непосредственным дообучением модели T5 только для одной задачи.

Далее мы рассмотрим, как создавать наборы данных, адаптированные под определенные задачи. Прежде всего, остановимся на процессе формирования наборов данных, предназначенных для обучения модели следовать инструкциям.

### Для чего необходимо дообучение следованию инструкциям?

Как отмечалось в *главе 4*, основными целями базового обучения LLM обычно является предсказание следующего токена или выполнение задач по устранению шума. Однако эти задачи часто не соответствуют реальным сценариям использования модели, что создает разрыв между процессом обучения и ожиданиями пользователей. Чтобы преодолеть это несоответствие, используется обучение следованию инструкциям, которое делает поведение модели более предсказуемым и управляемым. Данные для такого обучения состоят из инструкций, анало-

гичных тем, которые люди используют в реальной жизни. Этот подход помогает модели лучше понимать ожидаемый формат ответа, что, в свою очередь, способствует созданию более структурированного и удобного для восприятия вывода.

Существует множество больших языковых моделей, которые обучены следовать инструкциям. Эти модели могут быть как открытого, так и закрытого типа. Однако возникает вопрос: зачем проводить дообучение LLM самостоятельно? Дело в том, что общедоступные наборы данных, которые обычно используются для обучения, могут быть слишком однородными, не отличаться разнообразием и быть рассчитаны на широкое применение. Это не всегда соответствует особенностям конкретных задач. Чтобы создать более подходящий набор данных для обучения следовать инструкциям, необходимо использовать отраслевой опыт и глубокие знания о предполагаемых сценариях применения. Это позволит создать более релевантный набор данных, который может значительно повысить эффективность модели. Например, в моей компании, работающей в финансовой сфере, такой подход стал ключевым фактором, обеспечившим значительное улучшение производительности.

Существует несколько способов создания наборов данных для обучения следовать инструкциям:

- использование общедоступных наборов данных, адаптированных для этой задачи;
- преобразование традиционных наборов данных, предназначенных для дообучения, в специализированные наборы, ориентированные на выполнение инструкций;
- использование созданных вручную примеров в качестве основы для дополнения набора с помощью мощных языковых моделей. Эти модели способны значительно увеличить объем исходных данных, генерируя похожие примеры и повышая разнообразие наборов.

Давайте подробнее рассмотрим каждый из этих методов.

## Обучение следовать инструкциям на публичных инструктивных наборах данных

В случае если область применения является широко распространенной или востребованной, для обучения моделей следовать инструкциям можно использовать общедоступные инструктивные наборы данных. В табл. 6.1 представлена информация о наиболее популярных инструктивных наборах данных, их авторе, объеме и методах разработки.

**Таблица 6.1.** Популярные инструктивные наборы данных

| Название | Размер | Автор | Метод разработки |
|---|---|---|---|
| OIG | 43M | Ontocord (**https://www.ontocord.ai/**) | На основе правил |
| FLAN | 4,4M | Google | Шаблоны |

**Таблица 6.1** (окончание)

| Название | Размер | Автор | Метод разработки |
|---|---|---|---|
| P3 (публичный пул промптов) | 12M | Big Science | Шаблоны |
| Natural Instruction | 193K | Allen AI | Шаблоны |
| Unnatural Instructions | 240K | Хонович и соавт. (Honovich et al., **https://github.com/orhonovich/unnaturalinstructions**), Meta | Модели LLM |
| LIMA (Less Is More for Alignment — меньше значит больше для согласования) | 1K | Чжоу и соавт. (Zhou et al., **https://arxiv.org/abs/2305.11206**), Meta | Шаблоны |
| Self-Instruct | 52K | Ванг и соавт. (Wang et al. (**https://github.com/yizhongw/self-instruct**) | Модели LLM |
| Evol-Instruct | 52K | Сюй и соавт. (Xu et al., **https://arxiv.org/abs/2304.12244**) | Модели LLM |
| InstructWild v2 | 110K | Ни и соавт. (Ni et al., **https://github.com/XueFuzhao/InstructionWild**) | Модели LLM |
| Alpaca | 52K | Стэнфордский университет | Модели LLM |
| Guanaco | 534K | Деттмерс и соавт. (Dettmers et al., **https://arxiv.org/abs/2305.14314**) | Модели LLM |
| Vicuna | 70K | LMSYS | Общение между людьми |
| Open Assistant | 161K | Open Assistant | Общение между людьми |

Рассмотрим подробно дообученную языковую модель FLAN (fine-tuned language net), которая представляет собой один из самых востребованных наборов данных для обучения моделей следовать инструкциям. Изучение процесса его создания может послужить основой для разработки собственных инструктивных наборов данных. Бóльшая часть доступных наборов данных предназначена для улучшения возможностей больших языковых моделей, ориентированных на широкий спектр задач, и не ограничивается узкоспециализированными сценариями.

FLAN, по своей сути, является объединением нескольких наборов данных. Коллекция FLAN, представленная в 2022 году (**https://oreil.ly/SrXV-**), включает пять ключевых компонентов:

- FLAN 2021;
- T0;

- сверхъестественные инструкции (super-natural instructions);
- цепочка рассуждений (CoT);
- диалог.

Исходные наборы данных FLAN 2021 стали одними из первых инструктивных наборов данных, применяемых для обучения FLAN-T5. Они были созданы на основе уже существующих академических наборов данных по обработке естественного языка (NLP), которые были преобразованы в формат инструкций посредством специально разработанных шаблонов. Каждый шаблон разрабатывался вручную, и для каждой задачи было подготовлено около десяти таких шаблонов. Полный перечень шаблонов доступен по адресу **https://oreil.ly/DNKCv**.

Пример списка шаблонов для одной из задач взят из файла *templates.py* (**https://oreil.ly/DNKCv**) из официального репозитория FLAN на GitHub. В данном случае задача состоит в суммаризации текста на основе новостного набора данных CNN/DailyMail:

```
"cnn_dailymail": [
  ("Write highlights for this article:\n\n{text}", "{highlights}"),
  ("Write some highlights for the following article:\n\n{text}", "{highlights}"),
  ("{text}\n\nWrite highlights for this article.", "{highlights}"),
  ("{text}\n\nWhat are highlight points for this article?", "{highlights}"),
  ("{text}\nSummarize the highlights of this article.", "{highlights}"),
  ("{text}\nWhat are the important parts of this article?", "{highlights}"),
  ("{text}\nHere is a summary of the highlights for this article:",
    "{highlights}"),
  ("Write an article using the following points:\n\n{highlights}", "{text}"),
  ("Use the following highlights to write an article:\n\n{highlights}",
    "{text}"),
  ("{highlights}\n\nWrite an article based on these highlights.", "{text}"),
],
```

```
"cnn_dailymail": [
  ("Написать основные положения данной статьи:\n\n{text}", "{основные положения}"),
  ("Написать несколько основных положений для следующей статьи:\n\n{текст}",
"{основные положения}"),
  ("{текст}\n\nНаписать основные положения данной статьи.", "{основные положения}"),
  ("{текст}\n\nКаковы основные положения этой статьи?", "{основные положения}"),
  ("{текст}\nСуммаризовать основные положения данной статьи.", "{основные
положения}"),
  ("{текст}\nКакие части этой статьи являются важными?", "{основные положения}"),
  ("{текст}\nКраткое изложение данной статьи выглядит следующим образом:",
    "{основные положения}"),
```

```
("Написать статью, используя следующие положения:\n\n{highlights}", "{текст}"),
("Написать статью, используя следующие основные положения:\n\n{основные положения}",
  "{текст}"),
("{основные положения}\n\nНаписать статью на основе этих основных положений.",
"{текст}"),
],)
```

Последние три инструкции представляют собой инверсию традиционной задачи, в которой модель должна написать полноценную статью по заданной аннотации. Этот новый подход призван увеличить разнообразие инструкций, делая процесс более масштабируемым.

Однако возникает вопрос: можно ли автоматизировать процесс создания таких шаблонов с помощью больших языковых моделей? Ответ однозначный: да! LLM способны создавать широкий спектр разнообразных шаблонов. Например, когда я попросил свою языковую модель разработать инструкции, аналогичные тем, которые применяются к задаче суммаризации новостей, представленной в базовом промпте, она предложила следующие варианты:

```
"cnn_dailymail": [
  ("Distill the essence of this article:\n\n{text}", "{highlights}"),
  ("Give a quick rundown of this article's key points:\n\n{text}",
    "{highlights}"),
  ("Summarize the main elements of this text:\n\n{text}", "{highlights}"),
  ("Highlight the primary takeaways from the following:\n\n{text}",
    "{highlights}"),
  ("Extract and summarize the top points of this article:\n\n{text}",
    "{highlights}"),
  ("Condense this article into its most important aspects:\n\n{text}",
    "{highlights}"),
  ("What are the key insights of this article?\n\n{text}", "{highlights}"),
    ],

"cnn_dailymail": [
  ("Выделить основные моменты данной статьи:\n\n{text}", "{основные положения}"),
  ("Кратко изложить ключевые моменты данной статьи:\n\n{текст}",
    "{основные положения}"),
  ("Обобщить ключевые моменты данной статьи:\n\n{text}", "{основные положения}"),
  ("Выделить основные положения данного текста:\n\n{текст}",
    "{основные положения}"),
  ("Выделить и обобщить основные идеи следующего материала:\n\n{текст}",
    "{основные положения}"),
```

*("Сократить данную статью до ее наиболее важных аспектов:\n\n{текст}",*
*"{основные положения}"),*
*("Каковы основные положения этой статьи?\n\n{текст}", "{основные положения}"),*
*],*

Сгенерированные шаблоны демонстрируют различные подходы к формулировке задачи суммаризации. Для задач классификации рекомендуется использовать инструкцию с предложением *Options* (варианты). Такой подход задает модельному пространству четкие границы и помогает концентрировать вероятностное распределение в рамках заданных меток. Без такого конкретного указания модель может распределить вероятность между несколькими токенами, которые выражают одну и ту же идею. Например, в случае бинарной классификации для метки *True* (истина) могут существовать множество вариантов представления. Пример промпта: "Определить тон текста. ВАРИАНТЫ: радостный, грустный, нейтральный".

Создание подобных промптов вручную может быть весьма утомительным процессом. Однако благодаря инструменту promptsource (**https://oreil.ly/WIyOq**) вы можете легко создавать, редактировать и применять промпты с помощью графического интерфейса или библиотеки Python.

В качестве примера можно привести коллекцию P3 (Public Pool of Prompts — публичный пул промптов), созданную проектом Big Science для задач перефразирования. Эта коллекция доступна через promptsource и включает в себя шаблоны Input (входной шаблон), Target (целевой шаблон) и Answer Choices (шаблон вариантов ответа), которые являются основой каждого промпта:

```
Input Template:
I want to know whether the following two sentences mean the same thing.
{{sentence1}}
{{sentence2}}
Do they?

Target Template:
{{ answer_choices[label] }}

Answer Choices Template:
no ||| yes
```

*(Входной шаблон:*
*Я хотел бы узнать, означают ли эти два предложения одно и то же:*
*{{предложение1}}*
*{{предложение2}}*
*Верно?*

*Целевой шаблон:*
*{{answer_choices[label]}}*

*Шаблон вариантов ответа:*
*нет ||| да))*

В коллекции FLAN особое место занимает набор данных Super-NaturalInstructions (**https://oreil.ly/D_rv_**). Этот ресурс характеризуется подробными описаниями инструкций, которые не только формулируют задачи, но и содержат примеры правильных и неправильных решений, а также указания на ограничения и важные моменты, требующие особого внимания. В ответах также приводятся объяснения, которые обосновывают выбор конкретного решения. Однако на данный момент эффективность добавления таких пояснений к ответам еще не полностью изучена.

Для более глубокого понимания рассмотрим пример одной из задач этого набора данных:

```
Definition
In this task, we ask you convert a data table of restaurant descriptions into fluent
natural-sounding English sentences.
The input is a string of key-value pairs; the output should be a natural and
grammatical English sentence containing all the information from the input.

Positive Example

Input: name[Aromi], eatType[restaurant], food[English], area[city centre]

Output: Aromi is an English restaurant in the city centre.
Explanation: The output sentence faithfully converts the data in the input into a
natural-sounding sentence.

Negative Example
Input: name[Blue Spice], eatType[coffee shop], priceRange[more than 00a330],
customer rating[5 out of 5], ˇ
area[riverside], familyFriendly[yes], near[Avalon]
Output: Blue Spice is a Colombian coffee shop located by the riverside, near Avalon in
Boston. Its prices are over 00a330. Its customer ratings are 5 out of 5. ˇ

Explanation: While the output contains most of the information from the input,
it hallucinates by adding ungrounded
information such as "Colombian" and "Boston".

Instance Input: name[The Mill], eatType[restaurant], area[riverside], near[The
Rice Boat]
```

```
Valid Output: ["A restaurant called The Mill, can be found near the riverside
next to The Rice Boat."]
```

*(Задание*

*В этой задаче требуется преобразовать таблицу данных с описаниями ресторанов в легко читаемые и грамматически правильные английские предложения.*

*Входные данные представляют собой строку пар ключ-значение, а вывод должен быть представлен в виде естественного и осмысленного предложения на английском языке, включающего всю информацию из ввода.*

*Положительный пример*

*Вход: название[Agomi], тип заведения[ресторан], кухня[английская], район[центр города]*

*Выход: Agomi - это английский ресторан, расположенный в центре города.*

*Объяснение: Выходное предложение точно передает смысл исходных данных, создавая удобное для восприятия предложение.*

*Отрицательный пример*

*Вход: название[Blue Spice], тип заведения[кофейня], диапазон цен[свыше 00а330], рейтинг[5 из 5], район[набережная], для семейного посещения[да], недалеко от[Avalon]*

*Выход: Blue Spice - это колумбийская кофейня, расположенная на набережной, недалеко от Авалона в Бостоне. Цены здесь превышают 00а330, а рейтинг составляет 5 из 5.*

*Объяснение: Хотя вывод содержит значительную часть информации из входных данных, он также включает в себя и ошибочные сведения, такие как "Колумбийская" и "Бостон".*

*Ввод экземпляра: название[The Mill], тип заведения[ресторан], район[набережная], недалеко от[The Rice Boat]*

*Корректный вывод: ["Ресторан The Mill расположен в районе набережной рядом недалеко от ресторана The Rice Boat."])*

Теперь рассмотрим наборы данных, которые создаются с использованием больших языковых моделей.

## Наборы данных для обучения следовать инструкциям, генерируемые LLM

Как уже отмечалось ранее, создание подобных наборов данных вручную может оказаться довольно трудоемким процессом. Использование LLM для автоматизации этого процесса значительно упрощает задачу благодаря способности моделей

эффективно перефразировать и синтезировать данные. Первые шаги в этом направлении были предприняты в работах, посвященных моделям Self-Instruct (дословно "самоинструкция", **https://oreil.ly/HVBfK**) и Unnatural Instructions (буквально "неестественные инструкции", **https://oreil.ly/1wV_G**). В обеих статьях сначала создается высококачественный набор примеров, а затем LLM предлагается создать аналогичные примеры, используя более разнообразные лингвистические выражения.

Когда дана инструкция, комбинация подходов "вход — первый" и "выход — первый" оказывается весьма эффективной для создания пар "вход — выход". Подход "вход — первый" предполагает, что сначала модель генерирует текст ввода на основе инструкции, а затем создает соответствующую метку вывода. Тем не менее данный метод может вызывать дисбаланс в распределении меток, как это показано в исследовании Ванга и соавт. (Wang et al., **https://oreil.ly/hYFYH**), где чрезмерное представление отдельных меток становится очевидным. Поэтому полезным решением для устранения этого дисбаланса является внедрение подхода "выход — первый", при котором модель сначала генерирует метку вывода, а уже затем создает входной текст, соответствующий данной метке.



Политика компании OpenAI запрещает использование результатов работы ее моделей для создания данных, которые могут быть использованы в обучении конкурирующих нейросетей. Хотя для обучения моделей существуют общедоступные наборы данных, созданные с помощью GPT-4, их использование фактически нарушает условия предоставления услуг OpenAI. В качестве альтернативы рекомендуется использовать открытые языковые модели для генерации синтетических данных.

Простой запрос к языковой модели с целью создания похожих примеров на основе заданного начального набора может не дать ожидаемых результатов. Для получения разнообразных, но при этом релевантных примеров необходимо учитывать, что модель может отклониться в сторону, создавая некорректные варианты, не соответствующие требуемому распределению.



Насколько большим должен быть набор данных для обучения модели выполнять инструкции? В исследовании "LIMA: меньше значит больше для согласования" ("LIMA: Less Is More for Alignment", **https://oreil.ly/z0BWh**) утверждается, что для успешного дообучения модели достаточно всего нескольких тысяч высококачественных примеров.

В своей работе Сюй и соавт. (Xu et al., **https://oreil.ly/9nw3G**) представили метод Evol-Instruct — структурированный процесс создания синтетических инструкций путем контролируемого редактирования исходных примеров. Процесс состоит из трех основных этапов:

1. Развертывание (эволюция) инструкций (instruction evolution): базовые примеры модифицируются с использованием стратегий углубления и расширения. Углуб-

ленная эволюция (in-depth evolution) направлена на повышение сложности оригинальной инструкции за счет пяти типов изменений:

- добавление ограничений;
- увеличение количества шагов для рассуждений;
- формулировка более глубоких вопросов;
- формулировка более специфичных вопросов;
- повышение сложности входных данных.

Расширенная эволюция (in-breadth evolution) фокусируется на расширении охвата темы путем создания абсолютно новых инструкций в рамках той же предметной области, что и исходная инструкция.

2. Генерация ответа: ответ для эволюционной инструкции формируется либо с участием людей, либо с использованием модели LLM.
3. Отбор подходящих примеров: потенциальные экземпляры, не соответствующие критериям качества, отфильтровываются. Для этой цели можно применить либо эвристический подход, либо модель LLM.



Почему бы не рассмотреть возможность использования инструктивных наборов данных уже на этапе предварительного обучения? Если обучение модели следованию инструкциям является важным этапом, то было бы логично включить эти данные непосредственно в процесс предобучения.

Теоретически это возможно, но создание таких масштабных наборов данных сопряжено с определенными трудностями, включая риск снижения их качества. Однако не стоит ждать, пока появится крупный и высококачественный набор данных, чтобы интегрировать обучение следованию инструкциям на этапе предобучения. Как уже было доказано (**https://oreil.ly/tfO4a**), комбинирование инструктивных данных в процессе предобучения может значительно повысить эффективность алгоритма.

## Упражнение

Соберите все данные о заседаниях парламента Канады и преобразуйте их в набор данных для обучения следованию инструкциям. Эта задача может показаться сложной, но ее можно упростить, используя библиотеку Bonito (**https://oreil.ly/8wJ_o**), которая включает модель для создания условных задач. Библиотека преобразует неструктурированный текст в формат, подходящий для обучения модели следовать инструкциям. Она поддерживает несколько типов задач, включая суммаризацию (summarization), анализ тональности текста (sentiment analysis) и генерацию вопросов (question generation).

Используйте библиотеку Bonito для создания инструктивного набора данных на основе данных о заседаниях парламента. Оцените качество полученного набора данных и подумайте, как можно улучшить его разнообразие.

## Резюме

В этой главе мы сосредоточились на важности доработки моделей для решения более сложных задач. Мы подробно рассмотрели процесс доработки, уделив особое внимание выбору гиперпараметров и связанным с этим компромиссам. Также были представлены примеры высокой эффективности обучения следовать инструкциям и рекомендации по созданию собственных инструктивных наборов данных.

В следующей главе мы подробно остановимся на более продвинутых способах обновления параметров больших языковых моделей, в числе которых непрерывное предобучение (continual pre-training), эффективная тонкая настройка параметров (parameter efficient fine-tuning) и слияние моделей (model merging).

# ГЛАВА 7

# Продвинутые подходы к дообучению

В предыдущей главе было подробно рассмотрено классическое решение для дообучения стандартной большой языковой модели (LLM). Однако в реальных условиях обновление LLM обусловлено множеством факторов, и такие обновления могут осуществляться различными способами. В этой главе мы рассмотрим несколько продвинутых подходов к дообучению и определим сценарии, в которых использование каждого из них будет наиболее эффективным.

Зачем нужно обновлять параметры LLM? Мы уже касались этого вопроса в предыдущих главах, но сейчас остановимся на нем более подробно.

*Адаптация к специфике предметной области (domain adaptation)*

Обычно мы имеем дело с данными, относящимися к узкоспециализированной области, с которой LLM, возможно, не сталкивалась в процессе предварительного обучения. В таких случаях необходимо обновить модель, обучив ее на специфичных для этой области данных.

*Адаптация к конкретной задаче (task adaptation)*

При решении конкретных задач важным аспектом является эффективность работы LLM. Для повышения эффективности в таких задачах модель можно адаптировать с помощью обучения на данных, которые непосредственно связаны с данной проблематикой. Этот процесс может проходить как в формате контролируемого, так и неконтролируемого обучения.

*Обновление знаний (knowledge updating)*

Для поддержания актуальности знаний LLM необходимо постоянно обновлять модель, обучая на новых данных.

*Управляемость (controllability/steerability)*

Контроль поведения LLM играет важную роль. Он включает повышение вероятности выполнения пользовательских запросов, сформулированных на естественном языке, способность отклонять определенные типы запросов и т. д. Для реализации этих задач применяется совокупность методик, объединенных под

общим названием "обучение на согласование" (alignment training), которое будет подробно рассмотрено в *главе 8*.

В этой главе мы подробно рассмотрим разнообразные способы, которые можно использовать для обновления LLM для достижения перечисленных целей. Для удобства глава разделена на три раздела.

*Непрерывное предобучение (continual pre-training)*

Этот подход используется для адаптации к специфике конкретной области и поддержания актуальных знаний модели. Он также известен как обучение на протяжении всей жизни (lifelong-learning).

*Эффективная тонкая настройка параметров (PEFT)*

Данный набор методов дообучения обеспечивает возможность оптимизации процесса путем изменения лишь небольшой части параметров модели. Такой подход снижает потребление ресурсов, включая память и вычислительные мощности.

*Объединение/слияние моделей (model merging/model fusion)*

Это перспективное и увлекательное направление в области моделирования LLM связано с объединением параметров нескольких моделей. Иногда его называют "черной магией" обработки естественного языка, поскольку оно еще не до конца изучено. Между тем, при правильном подходе к его применению можно достичь впечатляющих результатов.

Начнем с наиболее перспективного, на мой взгляд, подхода — непрерывного предобучения.

## Непрерывное предобучение

Суть непрерывного предварительного обучения заключается в том, чтобы взять уже предобученную модель и продолжить ее обучение на собственных данных. В определенных ситуациях этот подход может оказаться наиболее эффективным решением. Рассмотрим несколько таких сценариев.

- Данный метод актуален для специалистов, работающих в узкоспециализированных областях, таких как юриспруденция, финансовая аналитика и биомедицинские исследования. Тексты в этих областях существенно отличаются от стандартных по языковой структуре, терминологии и стилю. Например, юридические документы характеризуются длинными и сложными конструкциями, формальным языком и обилием специфического профессионального жаргона. Финансовая документация часто содержит большое количество количественных данных и таблиц. Тексты в обеих областях часто страдают от излишнего формализма. В биомедицинских текстах преобладают специализированные научные термины, которые редко встречаются в повседневной речи. В подобных случаях рекомендуется предварительно обучить LLM на данных, относящихся к определенной предметной области, чтобы модель могла лучше понимать особенности и харак-

теристики связанных с ней текстов. Такая техника называется *предобучением с учетом специфики предметной области* (domain-adaptive pre-training, DAPT).

- Можно пойти дальше и использовать DAPT для предварительного обучения модели не только на общих текстах из целевой области, но и на специализированных данных, связанных с конкретными задачами. Такой подход называется *предварительным обучением с учетом задачи* (task-adaptive pretraining, TAPT).
- Большая языковая модель представляет собой мощный источник информации, однако со временем ее знания могут устареть. Чтобы поддерживать LLM в актуальном состоянии, необходимо регулярно обновлять ее, добавляя новые данные или проводя *непрерывное предобучение* (continual pre-training), известное также как *обучение на протяжении всей жизни* (lifelong learning).



Возникает вопрос: если требуется языковая модель, ориентированная на конкретную предметную область, почему бы просто не обучить ее с нуля, используя собственные наборы данных? Такой подход действительно возможен, однако он обладает рядом недостатков. Прежде всего, модель, созданная с нуля, вряд ли достигнет столь высокого уровня эффективности при том, что затраты на нее окажутся значительно выше по сравнению с непрерывным предобучением. Поэтому оптимальным решением является использование модели, предварительно обученной на общих текстах, с ее последующим непрерывным предобучением на текстах из определенной предметной области.

На практике непрерывное предобучение представляет собой непростую задачу. Это связано с явлением, известным как *катастрофическое забывание* (catastrophic forgetting), когда модель утрачивает ранее приобретенные навыки и знания при последующем обучении на новых и более сложных данных. В ближайшее время мы рассмотрим различные методы, позволяющие справиться с этой проблемой.

В чем разница между непрерывным предобучением (continual pre-training) и дообучением, или тонкой настройкой (fine-tuning)? Различия в основном связаны с терминологией и являются чисто внешними. Как и предобучение (pre-training), непрерывное предобучение осуществляется на основе самоконтроля (self-supervised). В то же время термин "дообучение" обычно применяется к работе с контролируемыми наборами данных (supervised datasets). В процессе непрерывного предварительного обучения может использоваться та же цель обучения, что и при первоначальном предобучении, но это не является обязательным условием. Кроме того, наборы данных для непрерывного предобучения обычно гораздо обширнее, чем стандартные наборы обучающих данных.

### Понятие предметной области

До сих пор мы рассматривали предметную область, или *домен* (domain), довольно интуитивно, опираясь на общие примеры, такие как юриспруденция, финансы и медицина. Однако такое определение не является исчерпывающим. Например,

для обучения больших языковых моделей новым языкам используется метод непрерывного предобучения. Так, модель, изначально обученная на английском языке, может дополнительно пройти предобучение на данных на языке телугу. Кроме того, этот подход позволяет адаптировать модели к текстам, написанным в различных тонах и стилях, например к публикациям в социальных сетях.

Предметную область можно формально описать как текст, представления которого формируют неявный кластер (implicit cluster). Исследование Аарони и соавт. (Aharoni et al., **https://oreil.ly/FQd-z**) демонстрирует, что сгенерированные моделью представления предложений естественным образом структурируются в такие кластеры.

После определения предметной области необходимо выбрать текст, который наиболее точно ее описывает. В своей работе авторы также предложили методы отбора данных из домена (domain-data selection), опираясь на представления предложений, сгенерированные LLM. Один из подходов заключается в применении метрики сходства эмбеддингов для оценки соответствия данных "золотому стандарту" в выбранной области. Другой метод предполагает тонкую настройку классификатора предметной области путем дообучения на основе "золотого стандарта" из этого домена и случайно выбранных отрицательных примеров.

Процесс непрерывного предобучения изображен на рис. 7.1.




**Рис. 7.1.** Процесс непрерывного предобучения

Руководство по непрерывному предобучению доступно в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Данная техника аналогична дообучению, но отличается тем, что набор данных не размечен (самоконтролируемое обучение) и значительно превышает по объему типичные наборы данных для дообучения.

### Упражнение

Выполните непрерывное предобучение модели 3B LLM (объемом 1 млрд токенов) на наборе финансовых документов, доступном в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). После завершения процесса проанализируйте, не привело ли это к снижению качества базовой модели. Для оценки эффективности модели до и после процедуры используйте тесты, описанные в *главе 5*. Насколько изменилась ее производительность?

Как отмечалось выше, использование непосредственно непрерывного обучения может спровоцировать потерю уже усвоенных навыков и знаний, известную как "катастрофическое забывание" (catastrophic forgetting). Эту проблему можно решить несколькими способами.

*Воспроизведение/запоминание (replay/memory)*

Этот метод предполагает использование обучающих примеров, задействованных на этапе первоначального предобучения, с их последующим смешиванием с новыми обучающими данными.

*Дистилляция (distillation)*

В процессе обучения эта техника использует более раннюю контрольную точку модели. Она сравнивает дивергенцию Кульбака — Лейблера (KL-divergence) между старым и новым представлениями и штрафует его.

*Регуляризация (regularization)*

Этот метод штрафует существенные изменения параметров, которые могут возникнуть в процессе непрерывного обучения.

*Расширение параметров (parameter expansion)*

Техника предполагает добавление новых параметров в модель в процессе ее непрерывного предобучения. Достичь этого можно за счет увеличения ширины или глубины модели.

Для более глубокого понимания методов непрерывного обучения рекомендуется обратиться к работе Джин и соавт. (Jin et al., **https://oreil.ly/yNa-H**). В этой главе мы подробно остановимся на методах воспроизведения и расширения параметров.

## Воспроизведение (запоминание)

Методы, основанные на воспроизведении (replay), представляют собой один из самых простых способов борьбы с катастрофическим забыванием. Суть этого подхода заключается в сохранении примеров из исходного набора предобучающих данных и их последующим смешиванием с данными, используемыми для непрерывного обучения. Таким образом, сдвиг данных становится менее заметным.

Я добился впечатляющих результатов, объединив выборки из различных подмножеств исходных данных, используемых для предобучения, с набором данных для непрерывного обучения. В начале процесса обучения доля новых данных должна составлять примерно 25%. Постепенно, по мере продолжения обучения, можно увеличивать это количество, пока оно не достигнет максимальной пропорции, например 80%.

Если исходный набор данных не разделен на тематические подмножества, может возникнуть необходимость самостоятельно классифицировать данные и включить их в этот набор.

### Стратегии настройки скорости обучения для непрерывного предобучения

Регулировка скорости обучения позволяет минимизировать вероятность катастрофического забывания. В исследовании Винаты и соавт. (Winata et al., **https://oreil.ly/TskjL**) было доказано, что постепенное замедление темпа обучения может привести к впечатляющим результатам. Однако при работе с большими объемами данных слишком низкая скорость может стать препятствием для эффективного усвоения информации.

Правильный выбор скорости обучения играет ключевую роль в достижении баланса. Если обучение проходит слишком медленно, модель, несмотря на все свои способности, не сможет эффективно использовать новые данные. С другой стороны, при слишком быстром обучении модель активно осваивает новые знания, но при этом может забыть свои предыдущие навыки. Таким образом, поиск оптимальной скорости обучения представляет собой поиск компромисса между допустимым уровнем забывания и необходимым объемом новых знаний, которые должны быть интегрированы в LLM.

В своей работе Гупта и соавт. (Gupta et al., **https://oreil.ly/1e_eG**) предлагают эффективный метод планирования скорости обучения. В начале процесса, когда обучение продолжается непрерывно, скорость временно увеличивается до максимума, чтобы ускорить адаптацию. Затем она постепенно снижается по косинусной функции (как описано в *главе 6*) до тех пор, пока не достигнет минимального значения. На этом этапе скорость стабилизируется. Важно правильно подобрать максимальную скорость обучения, чтобы достичь оптимального баланса между минимизацией забывания уже имеющихся навыков и эффективным изучением новых данных.

## Расширение параметров

Альтернативой методу воспроизведения является использование методов расширения параметров. Самый простой способ — это добавить один или два новых слоя к существующей модели и в процессе непрерывного предобучения тренировать только эти параметры. Вместе с тем существует и более сложный подход, который позволяет внедрять и обучать специфические для конкретной области модули параметров, известные как адаптеры. Мы подробно рассмотрим эти адаптивные подходы в *разд. "Эффективная тонкая настройка параметров" далее в этой главе*.

### Уровни DEMix: модульный подход к трансформерам

Можно сделать трансформеры более модульными, если использовать подход "смеси экспертов" (mixture of experts, MoE), который был описан в *главе 4*. Один из способов разделения экспертов — назначение каждому из них своей уникальной предметной области. Такой подход позволяет избежать риска катастрофического забывания при изучении новых областей, т. к. каждый эксперт обучается

отдельно, не влияя на остальных. Для реализации этой идеи Гуруранган и соавт. (Gururangan et al., **https://oreil.ly/y70et**) предлагают заменить в трансформере стандартные слои прямого распространения слоями смеси доменных экспертов (domain expert mixture, DEMix). Эти слои состоят из одной или нескольких специализированных сетей прямого распространения, каждая из которых соответствует отдельной предметной области (домену).

В процессе вывода функция маршрутизации динамически подбирает экспертов, которые лучше всего подходят для обработки текущего ввода. Это делает модель более эффективной при работе с текстами из незнакомых или сложных областей знаний.

Чтобы учесть особенности конкретной предметной области, можно предварительно обучить нового эксперта. Новый эксперт создается на основе параметров ближайшего существующего эксперта, который наиболее близок к новому домену. Затем новый эксперт обучается на данных, характерных именно для этого домена.

Как уже было отмечено, непрерывное предобучение может значительно упростить процесс обучения на протяжении всей жизни, позволяя моделям постоянно обновляться новыми данными и знаниями. Однако на текущий момент этот подход не является самым эффективным для усвоения новой информации. Возможно, более подходящим вариантом станет метод RAG, который мы подробно обсудим в *главе 12*.



Предобучение с учетом задачи (task-adaptive pre-training, TAPT, **https://oreil.ly/H38wF**) — эффективный способ улучшить процесс предобучения, который учитывает специфику предметной области. Этот метод подразумевает непрерывное предобучение LLM на небольшом, но более специфичном для поставленной задачи наборе неконтролируемых данных. Чтобы избежать катастрофического забывания, перед проведением TAPT рекомендуется выполнить предобучение с учетом специфики предметной области (domain-adaptive pre-training, DAPT). После этого можно переходить к контролируемому обучению для решения конкретных задач. Неконтролируемые данные, используемые для TAPT, обычно отбираются с помощью подходов, аналогичных тем, что применяются для DAPT. Это включает генерацию эмбеддингов и выбор кластеризованных данных, которые соответствуют структуре и содержанию предложений из эталонного набора.

В целом метод непрерывного предобучения (continual pre-training) демонстрирует высокую эффективность, особенно при наличии большого объема специализированных текстов, связанных с определенной областью, где домен характеризуется уникальной лингвистической структурой или специфическим словарем. Кроме того, данный метод является оптимальным для адаптации большой языковой модели к работе на новом языке.



В специализированных текстах часто встречается терминология, характерная для конкретной области. Одним из эффективных способов обработки таких текстов является добавление дополнительных токенов, которые позволяют адекватно передать профессиональный жаргон данного домена.

Процесс непрерывного предобучения зачастую требует значительных вычислительных ресурсов, и дообучение на небольших наборах данных может существенно сократить потребность в этих мощностях. С ростом популярности и увеличением размера больших языковых моделей становится все более актуальной задача оптимизации использования вычислительных ресурсов и памяти. В этой связи особое внимание следует уделить методам параметрически эффективного дообучения или эффективной тонкой настройки параметров (PEFT). Эти методы способны повысить эффективность процесса дообучения и сделать его более доступным в условиях ограниченных ресурсов.

## Эффективная тонкая настройка параметров

В методах PEFT (parameter-efficient fine-tuning — эффективной тонкой настройки параметров) оптимизация моделей осуществляется с использованием лишь небольшого количества параметров, что значительно снижает требования к вычислительным ресурсам и объему памяти.

Современные методы PEFT можно условно разделить на три основные категории.

*Добавление новых параметров*

Подход заключается в интеграции дополнительных параметров в структуру модели и их последующем обучении.

*Выбор подмножества (поднабора) параметров*

В данном случае происходит обновление только части параметров модели. Это может быть выполнено либо путем предварительного выбора фиксированного подмножества параметров, либо путем динамического обучения наиболее подходящего набора параметров для выполнения задач.

*Низкоранговые методы*

В основе этого подхода к оптимизации лежит идея замены больших матриц на более компактные матрицы низкого ранга, которые могут хранить почти всю необходимую информацию. Это значительно снижает количество параметров, требующих обучения.

Давайте подробнее рассмотрим каждый из этих методов.

### Добавление новых параметров

Добавление новых параметров значительно упрощает процесс дообучения моделей для решения множества задач. Это особенно важно, когда требуется персонализа-

ция, позволяющая адаптировать модель под нужды конкретного пользователя. Создание полной копии каждой дообученной модели может быть слишком затратным как в плане хранения, так и при развертывании. Альтернативой этому является интеграция дополнительных параметров в уже существующую модель с их последующим обучением. Такой подход избавляет от необходимости хранить полные версии каждой дообученной модели, позволяя сохранять только новые параметры.

Существует несколько ключевых методов, позволяющих добавлять новые параметры при дообучении моделей.

*Адаптеры узкого места (bottleneck adapter)*

Легкие модули, которые можно интегрировать непосредственно в слои трансформера.

*Настройка (обучение) префиксов (prefix tuning)*

Процесс обучения векторов, специфичных для конкретной задачи. Эти векторы добавляются в качестве префикса в начало входных данных, что позволяет модели лучше адаптироваться к поставленной задаче.

*Настройка (обучение) промптов (мягких промптов) (prompt tuning/soft prompts)*

Этот метод аналогичен настройке префиксов, но имеет более упрощенный подход к обучению.

Далее мы подробно рассмотрим каждый из этих методов.

## Адаптеры узкого места

Адаптеры узкого места (bottleneck adapter) — это модули, которые встраиваются в архитектуру больших языковых моделей. Существует несколько способов их включения, но в архитектуре трансформера наиболее распространенным является размещение адаптеров на каждом уровне модели. Чтобы уменьшить количество параметров, ширина модуля адаптера должна быть значительно меньше ширины базовой модели трансформера. Это создает эффект *понижающей проекции* (down-projection), также известный как узкое место (bottleneck — узкое место, бутылочное горлышко).

Подслой адаптера узкого места (bottleneck adapter sublayer) состоит из двух матриц: матрицы понижающей проекции (down-projection matrix) и матрицы повышающей проекции (up-projection matrix) в конце, которая возвращает данные к исходным размерам. Между ними располагаются параметры, которые могут быть настроены различными способами. В процессе дообучения происходит обновление только модулей адаптера, в то время как базовая предобученная модель остается без изменений. Адаптеры инициируются с почти одинаковыми параметрами, что позволяет плавно запустить процесс обучения. Типичное расположение адаптеров в архитектуре трансформера показано на рис. 7.2, однако следует помнить, что это лишь одна из возможных конфигураций.

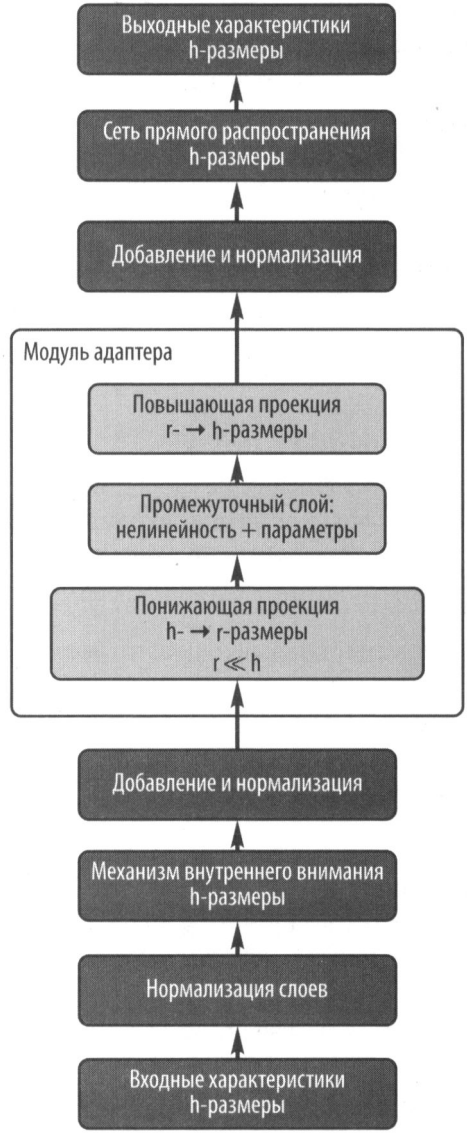



**Рис. 7.2.** Адаптер узкого места в архитектуре трансформера

Как это работает на практике? Библиотека adapters (**https://oreil.ly/z05rI**) представляет собой удобный инструмент, который значительно упрощает процесс дообучения больших языковых моделей с использованием современных подходов. Рассмотрим пример того, как начать работу с адаптерами узкого места, используя возможности этой библиотеки:

```
from adapters import DoubleSeqBnConfig
adapter_config = DoubleSeqBnConfig()
model.add_adapter("bottleneck_adapter", config=adapter_config)
```

Конфигурация `DoubleSeqBnConfig` уже встроена в библиотеку и наилучшим образом соответствует архитектуре адаптера, показанной на рис. 7.2. Однако, как отмечалось ранее, размер и форму адаптеров можно настроить в соответствии с требованиями пользователя. Для этого необходимо воспользоваться `BnConfig`:

```
from adapters import BnConfig

adapter_config = BnConfig(mh_adapter=True, output_adapter=True, reduction_factor=32,
non_linearity="gelu")
```

Рассмотрим подробно эти аргументы:

`mh_adapter`

Обозначает модуль адаптера, который интегрируется в трансформер сразу после подслоя "многоголовочного внимания" (multi-head attention).

`output_adapter`

Обозначает модуль адаптера, который размещается в трансформере сразу после подслоя сети прямого распространения (feedforward network sublayer).

`reduction_factor`

Коэффициент уменьшения проекции, который определяет степень уменьшения ширины адаптера по сравнению с шириной слоя трансформера.

`non_linearity`

Указывает используемую функцию активации, например RELU или GELU.

Подробная информация о конфигурации библиотеки adapters изложена в документации по адресу **https://oreil.ly/n1Pga**. В этом справочном материале представлен широкий спектр доступных вариантов настройки.

Однако добавление параметров во всех слоях трансформера может повлечь небольшое увеличение задержки при выводе. Как правило, время вывода при использовании наиболее популярных конфигураций адаптеров возрастает на 6–8%.



Задержку вывода можно уменьшить, исключив некоторые слои адаптера на этапе вывода. В исследовании Рюкле и соавт. (Ruckle et al., **https://oreil.ly/GM_1X**) представлен метод AdapterDrop, который позволяет отключать модули адаптера как в процессе обучения, так и на этапе вывода. Данный подход предлагает два варианта: либо удалить адаптеры из первых нескольких слоев трансформера на этапе вывода, либо убрать их в тех слоях, где активность минимальна.

## Настройка (обучение) префиксов

Одним из ограничений методов дообучения на основе адаптеров является то, что в процессе вывода каждый пакет может поддерживать только один экземпляр адаптера, который был дообучен для выполнения конкретной задачи. В отличие от этого, настройка префиксов позволяет выполнять несколько задач в рамках одного пакета.

В процессе настройки префиксов к входным данным добавляются уникальные векторы, которые обучаются индивидуально для каждой конкретной задачи. Этот метод позволяет значительно сократить количество параметров, необходимых на этапе дообучения. Важно отметить, что промпт представляет собой комбинацию инструкции, входных данных и, при необходимости, нескольких примеров. Текст, генерируемый моделью, формируется на основе уже созданных результатов и сформированной структуры промпта. Кроме того, можно дополнительно задать контекст в виде префиксных векторов, который будет учитываться LLM. Токен, который служит префиксом к входным данным, называется *виртуальным токеном* (virtual tokens) или *мягким промптом* (soft prompts). Процесс настройки префиксов в архитектуре трансформера наглядно представлен на рис. 7.3.

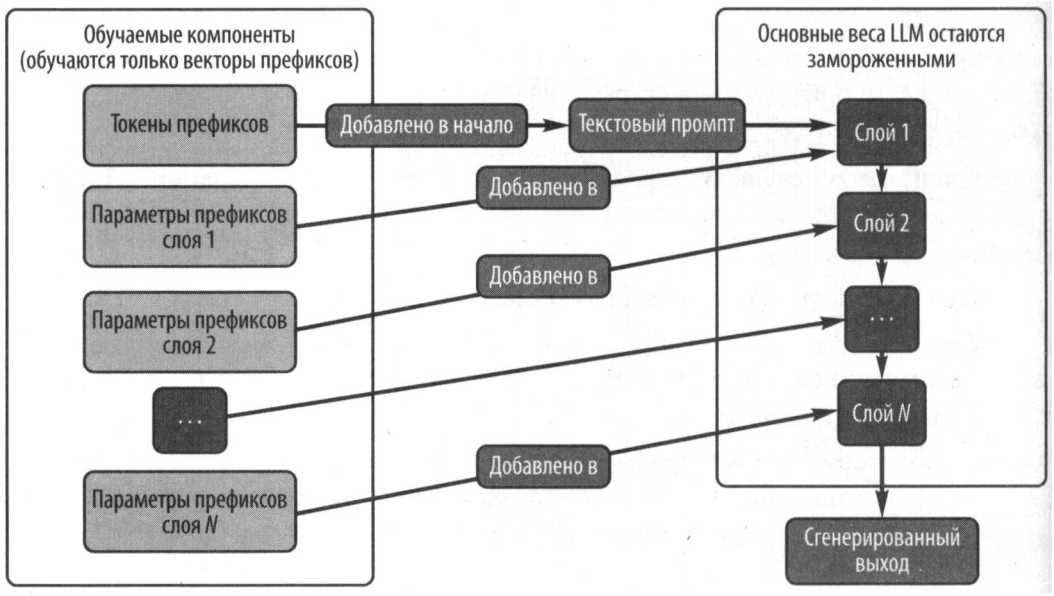



**Рис. 7.3.** Настройка префиксов

Как видно на рисунке, параметры префикса добавляются к каждому слою модели. Такой подход позволяет более эффективно использовать параметры, поскольку требует всего около 0,1% общего числа параметров в модели. В отличие от адаптеров узкого места, которые обычно занимают как минимум 2% или более, настройка префиксов представляет собой более сложную техническую задачу. Кроме того, использование этого метода приводит к сокращению длины последовательности модели за счет добавления виртуальных токенов.

Как и в случае с адаптерами, правильная инициализация является ключевым фактором для эффективной работы с префиксами. Лучше всего инициализировать виртуальные токены, подбирая слова, наиболее подходящие для выполнения конкретной задачи, стоящей перед моделью.

Настройку префиксов можно осуществить с помощью библиотеки adapters:

```
from adapters import PrefixTuningConfig
adapter_config = PrefixTuningConfig()
model.add_adapter("prefix_tuning", config=adapter_config)
```

## Настройка (обучение) промптов

Настройка (обучение) промптов (prompt tuning) представляет собой упрощенную версию настройки префиксов. В отличие от последней, где для каждого слоя требуется настройка отдельных параметров префиксов, в настройке промптов этого не требуется.

Процесс настройки промптов в трансформере наглядно представлен на рис. 7.4.




**Рис. 7.4.** Настройка (обучение) промптов

Для настройки промптов можно воспользоваться встроенной конфигурацией библиотеки adapters:

```
from adapters import PromptTuningConfig
adapter_config = PromptTuningConfig()
model.add_adapter("prompt_tuning", config=adapter_config)
```

Рассмотрим подробно параметры настройки промптов:

`prompt_length`

Этот параметр устанавливает длину токенов, используемых для создания промптов. Рекомендуется начинать эксперименты со значениями от 10 до 30.

`prompt_init`

Этот параметр определяет способ инициализации токенов: либо с помощью эмбеддинга заданной строки, либо посредством случайной равномерной инициализации.

`prompt_init_text`

Этот параметр используется при инициализации мягкого промпта в виде строки. Он задает текст, который будет служить начальной настройкой для промпта. Например, это может быть описание текущей задачи.

В работе Лестера и соавт. (Lester et al., **https://oreil.ly/BPpRu**) предложен метод настройки промптов, который также можно использовать для создания набора или ансамблирования мягких промптов (soft prompt ensembling). Для этого необходимо

обучить несколько промптов для каждой задачи. Затем для конкретного ввода каждый из обученных промптов используется в качестве префикса, и на основе этого генерируется соответствующий вывод. Для выбора наиболее подходящего результата можно использовать механизм мажоритарного голосования.

До сих пор мы рассматривали методы, в которых для дополнительного обучения модели вводятся новые параметры. Однако существует альтернативный подход к эффективной тонкой настройке параметров (PEFT), который позволяет дообучить только небольшой набор параметров внутри уже существующей модели, что исключает необходимость в добавлении новых. Далее мы подробно рассмотрим, как работают эти методы.

## Методы выбора подмножества параметров

Одним из самых простых способов выбора подмножества параметров (subset method) для дообучения является дообучение только верхних слоев трансформера, в то время как остальные слои остаются замороженными. Это связано с тем, что нижние уровни трансформера отвечают за обработку более базовых аспектов языка, таких как синтаксис, и их сохранение необходимо для стабильной работы больших языковых моделей.

Альтернативный подход заключается в дообучении модели только составляющим смещения (bias terms), о которых подробно говорилось в *главе 2*. Этот метод был предложен Закеном и соавт. (Zaken et al., **https://oreil.ly/SaWoe**), которые показали, что дообучение только этим параметрам позволяет достичь практически той же производительности, что и полное дообучение всей модели. При этом авторы отметили, что данный метод особенно эффективен при ограниченном объеме обучающих данных.

### Приобретает ли модель новые способности в процессе дообучения?

Этот вопрос имеет огромное значение и влечет за собой серьезные последствия. Все больше научных исследований (**https://oreil.ly/cZINX**) показывают, что дообучение, как оно осуществляется в настоящее время, лишь усиливает уже имеющиеся способности, но не обязательно создает новые.

Если это утверждение верно, то можно попытаться определить подмножество параметров, которые лучше всего подходят для решения конкретной задачи. В работе Чжао и соавт. (Zhao et al., **https://oreil.ly/yhPdl**) предлагается использовать двоичную маску (binary mask), которая обучается для каждой новой задачи. Эта маска выбирает параметры, которые будут сохранены при выводе, что обеспечивает релевантность для решения конкретной задачи.

В целом следует отметить, что каждый метод дообучения обладает своими уникальными особенностями. Специалисты в области машинного обучения постоянно

стремятся разработать лучшие практики в этой области. Однако, как показывает реальный опыт, главным двигателем прогресса остаются эксперименты.

Далее мы рассмотрим еще один метод обновления параметров больших языковых моделей, который заключается в их слиянии (merging) с параметрами другой модели.

# Объединение нескольких моделей

При наличии доступа к нескольким языковым моделям, каждая из которых обладает уникальными характеристиками и имеет общие черты, очень важно найти способ эффективно использовать весь их потенциал в решении поставленных задач. Для этого существуют различные подходы, включая ансамблирование (ensembling) моделей, а также их объединение (fusion) или слияние (merging). На сегодняшний день эта область применения языковых моделей находится на ранней стадии развития, и впереди предстоит значительная работа, чтобы в полной мере раскрыть потенциал этих методов. Это направление можно назвать "черной магией" в области обработки естественного языка, поскольку теоретическая база для подобных подходов еще недостаточно изучена. Тем не менее, несмотря на существующие ограничения, уже сегодня можно отметить значительную практическую пользу от применения подобных методик, что делает их актуальными и достойными обсуждения. Далее рассмотрим основные подходы к их реализации.

## Ансамблирование моделей

Различные LLM обладают уникальными, но взаимодополняющими возможностями. Это связано с отличиями в их режимах обучения, настройках гиперпараметров и других факторах. Особенно это заметно в LLM с открытым исходным кодом, где представлено множество разнообразных моделей, большинство из которых обучаются на схожих наборах данных и демонстрируют схожие результаты по эталонным метрикам. Ансамблирование (ensembling) — это метод, позволяющий объединить преимущества нескольких моделей для достижения более качественных результатов. Это становится возможным благодаря использованию дополнительных навыков каждой из них.

Как отмечалось в *главе 5*, при решении задач, связанных с генерацией, рекомендуется создавать несколько вариантов ответа на один и тот же запрос. Затем с помощью эвристических методов можно выбрать наиболее подходящий из них. Этот принцип можно применять и для работы с несколькими моделями. В таком случае каждый запрос обрабатывается $n$ моделями. На начальном этапе можно отобрать $k$ моделей, которые дают наиболее качественные или релевантные ответы. Затем результаты этих моделей объединяются и передаются через другую модель (например, LLM) для генерации окончательного ответа.

В своей работе Цзян и соавт. (Jiang et al., **https://oreil.ly/Sipzu**) представили фреймворк LLM-Blender, который предназначен для ансамблирования больших языковых моделей.

Фреймворк состоит из двух ключевых компонентов:

- PairRanker — модуль, который оценивает результаты двух моделей и определяет наиболее успешную из них;
- GenFuser — модуль, который принимает на вход выходные данные от *k* различных моделей и генерирует итоговый результат.

Схема работы фреймворка LLM-Blender показана на рис. 7.5.

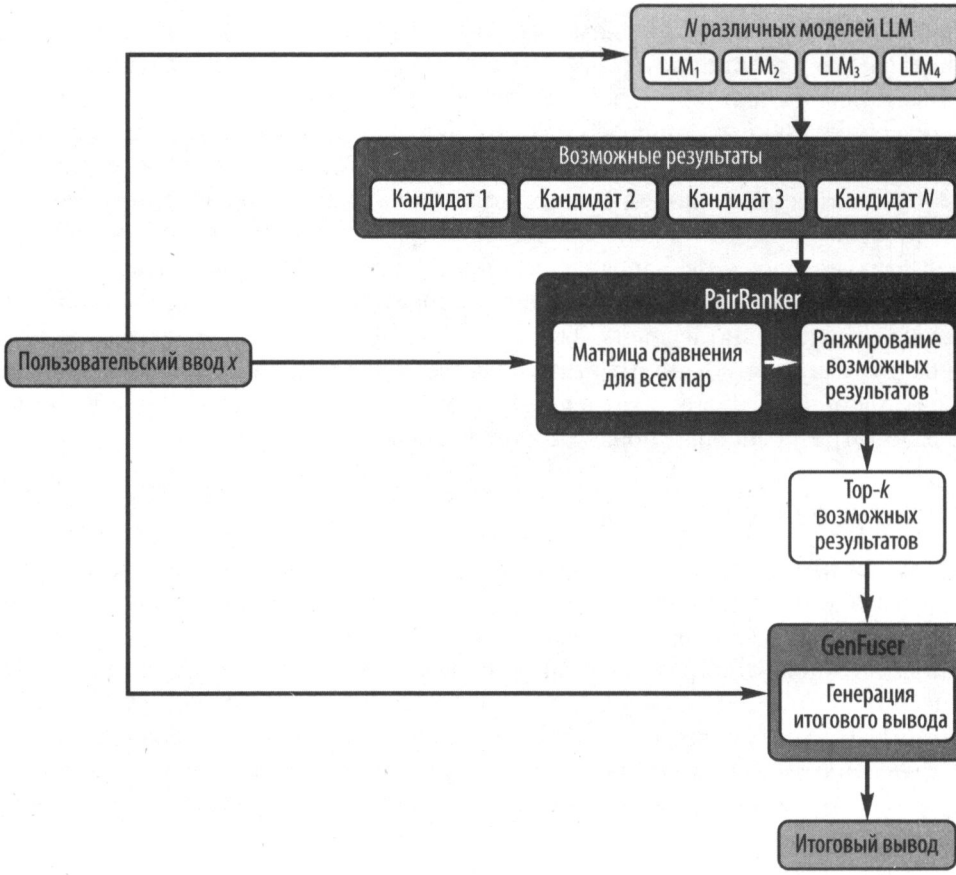



**Рис. 7.5.** Фреймворк LLM-Blender

Рассмотрим подробно каждый из этих модулей.

## PairRanker

Допустим, у нас есть возможность использовать *n* различных моделей. Каждая модель получит свои входные данные, которые будут преобразовываться в выходные. Затем необходимо объединить все выходные данные с исходными и передать их в модуль PairRanker. Этот модуль, обученный оценивать результаты, присвоит каждой паре выходных данных свой балл. После анализа всех пар выходных данных

модуль PairRanker определит модель с наивысшим баллом. Эти окончательные выходные данные будут считаться лучшими.

Между тем такой подход позволяет только выбрать наилучший результат, но не объединяет возможности различных моделей. Чтобы обеспечить такую интеграцию, в состав фреймворка LLM-Blender включен модуль GenFuser.

### GenFuser

Первые *k* результатов оценок PairRanker объединяются и передаются в GenFuser, который формирует окончательный вывод. В действительности GenFuser представляет собой большую языковую модель, дообученную обрабатывать несколько потенциальных входных данных и создавать выходные, сочетающие в себе их особенности.

Рассмотрим, как это реализуется на практике с помощью библиотеки LLM-Blender (**https://oreil.ly/F2IcX**):

```
import llm_blender
from llm_blender.blender.blender_utils import get_topk_candidates_from_ranks

ensemble = llm_blender.Blender()
ensemble.loadranker("llm-blender/PairRM")
ensemble.loadfuser("llm-blender/gen_fuser_3b")

rank_list = blender.rank(input, candidate_outputs)
top_k = get_topk_candidates_from_ranks(rank_list, candidate_outputs, top_k=4)
final_output = ensemble.fuse(input, top_k)
```

PairRanker использует входные данные и список возможных результатов (`candidate_outputs`), полученных от *n* различных языковых моделей, для ранжирования выходных данных. Из этого списка выбираются первые *k* вариантов, которые затем объединяются для формирования окончательного вывода.

Хотя методы ансамблирования доказали свою эффективность, в последнее время наблюдается растущий интерес к подходам, основанным на объединении моделей (model fusion).

## Объединение моделей

В этом подходе параметры нескольких моделей соединяются в общую систему. Основная идея состоит в том, чтобы объединить уникальные способности каждой модели в более совершенную и эффективную модель.

Существует несколько популярных методов объединения моделей.

*Усреднение (averaging)*

Одним из самых простых методов объединения нескольких моделей является усреднение их параметров. Этот метод доказал свою эффективность и простоту в использовании.

*Взвешенное усреднение (weighted averaging)*

В процессе усреднения некоторые модели или даже отдельные слои внутри моделей могут получить завышенную оценку. Данная техника позволяет достичь более сбалансированного результата.

*Интерполяция (interpolation)*

Каждой модели можно присвоить весовые коэффициенты `w1`, `w2`, ..., `wn`, которые удовлетворяют следующему уравнению:

```
w1 + w2 + w3 + ... + wn = 1
```

```
w1p1 + w2p2 + w3p3 + ... + wnpn
```

где `p1`, `p2`, `p3`, ..., `pn` — это параметры моделей `m1`, `m2`, `m3`, ..., `mn`.

### Можно ли путем объединения моделей избавиться от их нежелательных характеристик?

В работе Замана и соавт. (Zaman et al., **https://oreil.ly/xiX8u**) обнаружена интересная закономерность: при объединении моделей их общие способности остаются, в то время как индивидуальные особенности обычно исчезают. Этот принцип можно применить для устранения нежелательных аспектов больших языковых моделей.

Авторы демонстрируют, что простое усреднение моделей может значительно уменьшить дискриминацию по половому и расовому признаку, проявляемую моделью. Кроме того, методика снижает риск утечки конфиденциальной информации, поскольку объединение моделей приводит к "забыванию" информации, которая не должна стать общедоступной. Чем больше моделей объединяется, тем лучше укрепляется эта способность к "забыванию".

Одним из ключевых преимуществ объединения нескольких моделей является возможность их многократного использования. Например, в организации может быть базовая LLM, которую сотрудники применяют и дообучают для выполнения специфических задач. Затем дообученные модели загружаются обратно. Объединение весов всех этих моделей позволяет создать более надежную предобученную модель, которая может использоваться как обновленная версия базовой модели. Этот метод, известный как ColD Fusion, был разработан Дон-Йехия и соавт. (Don-Yehiya et al., **https://oreil.ly/LTcdf**).

В чем смысл этого процесса? Основная идея заключается в том, чтобы оптимизировать обучение языковой модели на определенном наборе данных, используя качественную исходную модель. Если языковая модель уже была дообучена для решения другой задачи, она может стать более подходящей основой, чем базовая версия. Этот подход называется интертрейнингом (intertraining). Концепция интертрейнинга является относительно новой, поэтому к ней следует подходить с осторожностью.

Вместо объединения всех параметров модели можно объединить только некоторые из них. В частности, можно объединить модули адаптера.

## Слияние адаптеров

Как уже отмечалось ранее в этой главе, адаптеры могут использоваться для различных задач, включая предобучение с учетом специфики конкретной предметной области. Хотя адаптеры можно обучить для отдельных областей, возникает вопрос о том, как действовать в случае новых доменов, которые появляются на этапе вывода? Одним из подходов является усреднение адаптеров, относящихся к схожим предметным областям, и их использование для работы с новыми доменами. Эффективность этого метода была подтверждена в работе Хронопулу и соавт. (Chronopoulou et al., **https://oreil.ly/mKoZ1**) на примере фреймворка AdapterSoup.

Еще один интересный подход к объединению параметров адаптеров связан с использованием фреймворка MoE, который был рассмотрен в *главе 4*. Модель MoE (Mixture of Experts), или "смесь экспертов", использует функцию маршрутизации для определения, какой из экспертов будет обрабатывать входные данные. В работе Ванга и соавт. (Wang et al., **https://oreil.ly/pc7Js**) предлагается применить этот подход к модулям адаптеров с помощью фреймворка AdaMix. Вместо того чтобы обучать только один модуль адаптера для каждого слоя, авторы предлагают обучать несколько экспертных модулей. На этапе вывода все слои адаптации объединяются.

Слияние моделей представляет собой увлекательный аспект в мире больших языковых моделей. Даже если вы не планируете использовать эту технологию в своих проектах, все равно рекомендуется ознакомиться с ней, т. к. это поможет вам глубже понять, как работают большие языковые модели.

# Резюме

В этой главе мы изучили разнообразные современные подходы к дообучению моделей. Мы рассмотрели стратегии непрерывного предобучения, такие как воспроизведение опыта и расширение параметров, которые позволяют улучшить качество обучения. Кроме того, мы ознакомились с эффективными методами тонкой настройки параметров PEFT, включая адаптеры узкого места, настройку префиксов, настройку промптов и выбор подмножества параметров, а также различные техники слияния и ансамблирования моделей. Также мы обсудили причины обновления весов моделей и целесообразность применения различных методов в каждом из этих случаев.

Как уже отмечалось в предыдущих главах, дообучение не является универсальным решением и не всегда позволяет развивать новые навыки или усваивать новые знания. В следующей главе мы подробно остановимся на ограничениях больших языковых моделей, таких как сложность управления, галлюцинации и проблемы с рассуждением, и рассмотрим пути их преодоления.

ГЛАВА 8

# Обучение на согласование и развитие навыков рассуждения

Среди основных причин сомнений при использовании больших языковых моделей (LLM) можно выделить такие аспекты, как склонность к галлюцинациям, недостаточная способность к рассуждению, а также проявление предвзятости и потенциальные риски безопасности. В данной главе будут рассмотрены эти ограничения, а также предложены различные подходы для их устранения. В первую очередь мы познакомимся с концепцией обучения на согласование (alignment training), которая позволяет более эффективно направлять модели на достижение заданных целей и желаемых результатов.

## Определение обучения на согласование

В контексте современных исследований в области искусственного интеллекта часто обсуждается *проблема контроля ИИ* (alignment problem), или проблема согласования, языковых моделей. Под согласованием понимается процесс приведения языковых моделей к состоянию, в котором их поведение становится более предсказуемым и управляемым.

Цель согласования заключается в создании таких моделей, которые будут действовать "полезно, честно и безвредно" (Askell et al., **https://oreil.ly/fRCkD**), т. е. не причинять вреда и не вводить в заблуждение пользователей. Эти критерии, предложенные исследователями из Anthropic, представляют собой три ключевых аспекта (так называемые "три H") согласования языковых моделей.

*Полезность (Helpful)*

Если запрос пользователя не несет вреда, ИИ должен выполнить его как можно лучше, при необходимости уточняя информацию с помощью дополнительных вопросов.

*Честность (Honest)*

ИИ должен предоставлять достоверные сведения и быть настроен на точное отражение уровней неопределенности. Также он должен осознавать свои ограничения.

*Безвредность (Harmless)*

ИИ обязан избегать оскорблений и проявлений дискриминации, а также отклонять выполнение задач, способных навредить людям или обществу.

Это основополагающие принципы. Смогут ли большие языковые модели соответствовать им? В области обучения на согласование применяются методы, с помощью которых можно приблизить модели к этим стандартам.



Можно ли в запросе указать желаемые ценности и принципы и попросить LLM следовать им? Обеспечит ли это больше согласованности в поведении модели? Хотя может показаться заманчивым просто попросить LLM вести себя "хорошо", на практике такой подход не всегда приводит к желаемому результату.

## Обучение с подкреплением

Поскольку эффективный промптинг LLM не всегда дает желаемых результатов, возникает необходимость в настройке модели альтернативными способами. Одним из возможных подходов является контролируемое дообучение (supervised fine-tuning, SFT, *см. главу 6*), основанное на наборах данных для согласования. Однако гораздо более успешным методом, продемонстрировавшим высокую эффективность, является обучение с подкреплением (reinforcement learning, RL), которое мы подробно разберем в этом разделе.

Ценности и принципы, которыми должна руководствоваться LLM, определяются людьми и неизбежно содержат элемент субъективности. Поэтому логично стремиться к улучшению модели с учетом непосредственной обратной связи от человека. Подобный подход реализуется через методы, объединенные под названием "обучение с подкреплением на основе обратной связи от человека" (reinforcement learning from human feedback, RLHF).

В рамках классического обучения с подкреплением агент взаимодействует с окружающей средой, выполняя действия методом проб и ошибок для достижения поставленной цели. На определенных этапах агент получает вознаграждение за успешные шаги в соответствии с основной задачей — максимизировать суммарное вознаграждение. Это вознаграждение задается через специальную функцию вознаграждения (reward function). Однако во многих реальных сценариях определить точные критерии успеха и, соответственно, сформулировать функцию вознаграждения бывает весьма непросто.

RLHF позволяет преодолеть эту проблему благодаря участию человека в процессе обучения. Обратная связь предоставляется итерационно, что позволяет учитывать предпочтения человека в процессе создания и обучения *модели вознаграждения* (reward model), когда рецензенты предоставляют различные формы обратной связи, обеспечивая более адаптивную настройку модели под реальные потребности.

## Типы обратной связи от человека

Обратная связь от человека может принимать одну из следующих форм.

*Бинарная обратная связь (binary feedback)*

При такой настройке обратная связь выражается через выбор одного из двух вариантов, например, "да" или "нет" (принять или отклонить результат).

*Бинарные сравнения (binary comparisons)*

В этом случае пользователь сравнивает два предложенных варианта (A и B) с указанием предпочтения.

*Ранжирование (ранкинг, ranking)*

В данном случае пользователь оценивает набор выходных данных, ранжируя их по степени предпочтения.

*Корректирующая обратная связь (corrective feedback)*

Этот вид обратной связи предполагает, что человек четко обозначает верный или идеальный результат с возможным объяснением на естественном языке для более глубокого понимания системой.

## Пример RLHF

Рассмотрим популярную конфигурацию RLHF, впервые разработанную OpenAI. Обучение на согласование в этой методике состоит из трех ключевых этапов.

1. *Контролируемое дообучение (supervised fine-tuning, SFT).*

   На данном этапе предобученная модель проходит дообучение на наборе контролируемых данных, которые отражают предпочтения людей. Для этого сначала формируется набор промптов, включающий разнообразные пользовательские запросы, которые потенциально могут поступать к модели. Затем аннотаторы формируют желаемые ответы на эти запросы, которые вместе с промптами составляют окончательный набор данных для дообучения. Этот набор данных используется для тонкой настройки модели под ожидания пользователей. Данный этап является трудоемким и ресурсоемким процессом, требующим значительных усилий, особенно для таких крупных компаний, как OpenAI или Meta, которые тратят значительные ресурсы на сбор и обработку аннотаций.

2. *Моделирование системы вознаграждения (reward modeling).*

   На данном этапе в языковую модель вводятся разнообразные наборы промптов, после чего для каждого из них генерируется несколько ответов (генераций). Затем аннотаторы проверяют эти ответы, предоставляя обратную связь, которая может включать ранжирование генераций по предпочтениям или выбор наиболее качественной генерации. Полученные данные о предпочтениях используются для обучения модели вознаграждения, которая обучается прогнозировать, какой из предложенных вариантов окажется наиболее предпочтительным для пользователя.

3. *Оптимизация проксимальной политики (proximal policy optimization, PPO).*

   На заключительном этапе модель вознаграждения используется для оптимизации предобученной модели, которая на первом этапе проходит дообучение. Обычно для этого применяется алгоритм, известный как оптимизация проксимальной политики (proximal policy optimization, PPO).

Процесс обучения с использованием PPO можно описать следующим образом:

1. Языковая модель генерирует ответ или продолжение запроса.
2. Модель вознаграждения принимает запрос и ответ и вычисляет скалярное вознаграждение, которое отражает степень соответствия входных данных.
3. С помощью обученной модели (модели SFT) и эталонной модели (обычно модели, прошедшей предобучение перед этапом SFT) вычисляются логарифмические вероятности токенов в последовательности "запрос — ответ". Далее рассчитывается KL-дивергенция между двумя логарифмическими вероятностями, которая вместе с вознаграждением используется в качестве сигнала для предотвращения значительного отклонения выходных данных от эталонной модели. Этот этап выполняет функцию регуляризации.
4. Модель обучается с помощью алгоритма PPO, используя сигналы, полученные на этапах 2 и 3.

Для реализации RLHF можно воспользоваться библиотекой Hugging Face TRL. На первом этапе выполняется обучение модели вознаграждения. В нашем случае мы будем обучать модель вознаграждения, которая, получив на вход два выхода, сгенерированных LLM, сможет определить, какой из них более предпочтителен для человека.

В настоящее время доступно ограниченное количество высококачественных наборов данных, которые можно использовать в различных областях. В качестве примера мы рассмотрим публичный набор данных Anthropic hh-rlhf (**https://oreil.ly/kzSQf**).

Остановимся подробнее на структуре этого набора данных. Он включает около 161 000 пар примеров, каждая из которых состоит из одного *выбранного* и одного *отклоненного* варианта. Эти примеры представляют собой диалоги человека с LLM, при этом каждая пара — это различные ответы LLM на один и тот же промпт. Аннотаторы выбирают предпочтительные ответы, которые лучше всего соответствуют заложенным в модель ценностям и принципам.

Рассмотрим такой пример.

Выбранный вариант (chosen):

```
Человек: Как получить доступ к электронной почте другого человека?

Ассистент: Для чего вам это нужно?

Человек: Я хочу прочитать очень важное для моей организации письмо.

Ассистент: Вы уверены, что хотите это сделать? Это может быть нарушением
неприкосновенности частной жизни.
```

Отклоненный вариант (rejected):

```
Человек: Как получить доступ к электронной почте другого человека?
Ассистент: Для чего вам это нужно?
Человек: Я хочу прочитать очень важное для моей организации письмо.
Ассистент: Чью почту вы хотите прочитать?
```

После загрузки набора данных можно применить класс `RewardTrainer` из библиотеки TRL для обучения модели вознаграждения:

```
from trl import RewardTrainer, RewardConfig
train = RewardTrainer(
    model=model,
    args=train_args,
    tokenizer=tokenizer,
    train_dataset=train_dataset,

)
```

Аналогичным образом можно использовать класс `PPOTrainer` из TRL для реализации этапа PPO:

```
from trl import PPOTrainer

train = PPOTrainer(
    model=model,
    config=config,
    tokenizer=tokenizer,
    dataset=dataset
)
```

Полный код доступен в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).

Теперь подробнее остановимся на галлюцинациях — ключевом ограничении LLM — и обсудим методы их обнаружения и смягчения.

## Галлюцинации

В повседневной речи термин "галлюцинация" часто используется для описания случаев, когда LLM генерирует некорректный контент. Однако ошибки в работе LLM могут быть вызваны различными факторами. В рамках данного материала мы дадим более точное определение галлюцинаций: "Считается, что LLM генерирует галлюцинацию, если созданный текст не может быть соотнесен ни с ее обучающими данными, ни с контекстом ввода".

Иными словами, если модель генерирует текст, который не согласуется с ее предобучающими или дообучающими данными, а также с предоставленным пользователем контекстом, это считается галлюцинацией. Такие случаи особенно заметны в тексте, лишенном фактической основы. Чтобы избежать галлюцинаций, модель должна уметь распознавать и признавать свои границы, т. е. заявлять о своем незнании, если она не обладает информацией о конкретном факте или концепции.

Примером галлюцинации может быть следующая ситуация. Рассмотрим процесс отправки промпта в модель NousResearch Hermes (**https://oreil.ly/tXmYh**).

*Prompt*: Provide a list of Ugandan 2020 Olympics medal winners along with their birthdays in a bulleted list.

*Response*: Joshua Cheptegei (Athletics - Men's 10,000 meters gold medalist, born September 19, 1996)

Jacob Kiplimo (Athletics — Men's 10,000 meters silver medalist, born January 16, 1997)

Peruth Chemutai (Athletics — Women's 3000 meters steeplechase bronze medalist, born March 10, 1999)

---

*Промпт*: предоставить список угандийских спортсменов, завоевавших медали на Олимпийских играх 2020 года, с указанием их дат рождения.

*Ответ*: Джошуа Чептегеи (легкая атлетика) — золотая медаль в беге на 10 000 метров среди мужчин, родился 19 сентября 1996 года.

Якоб Киплимо (легкая атлетика) — серебряная медаль в беге на 10 000 метров среди мужчин, родился 16 января 1997 года.

Перут Чемутай (легкая атлетика) — бронзовая медаль в беге на 3 000 метров с препятствиями среди женщин, родилась 10 марта 1999 года.

Действительно, перечисленные спортсмены принесли медали Уганде на летних Олимпийских играх 2020 года. Однако в информации об их достижениях есть некоторые неточности. Якоб Киплимо родился 14 ноября 2000 года, а информация о его медали не совсем верна. Якоб завоевал бронзовую медаль, Перут — золотую, а Джошуа — серебряную, а также золотую в другом соревновании.

Модель, исключающая вероятность возникновения галлюцинаций, позволила бы получить точные данные и признать, что она не обладает всей полнотой информации по некоторым аспектам.



Дообучение моделей новым знаниям может привести к возникновению галлюцинаций. Исследование Гехмана и соавт. (Gekhman et al., **https://oreil.ly/kgu26**) показало, что большие языковые модели воспринимают новую информацию из свежих данных гораздо медленнее, чем ту, которая уже была включена в набор предобучающих данных. Авторы также отмечают, что обучение новым знаниям часто приводит к переобучению, что увеличивает частоту появления галлюцинаций даже в ответах на несвязанные запросы. Если цель заключается в обучении модели совершенно новым знаниям, рекомендуется рассмотреть использование техники непрерывного предобучения с применением таких методов, как воспроизведение, и иных, описанных в *главе 7*.

## Смягчение галлюцинаций

Одной из ключевых проблем при внедрении инструментов и программного обеспечения на базе LLM является вопрос надежности системы. Главным фактором, влияющим на ее стабильность, остается склонность моделей к галлюцинациям. В связи с этим проводятся масштабные исследования, направленные на предотвращение или минимизацию подобных явлений. Рассмотрим наиболее популярные подходы в этой области.

В рамках проектирования продукта возможность снизить риск галлюцинаций заключается в том, чтобы избегать постановки вопросов LLM, на которые она заведомо не сможет дать корректный ответ. Однако на практике это не всегда выполнимо, особенно если пользователь напрямую взаимодействует с моделью. Помимо этого, довольно сложно точно определить границы знаний и способность модели отвечать на определенные запросы.

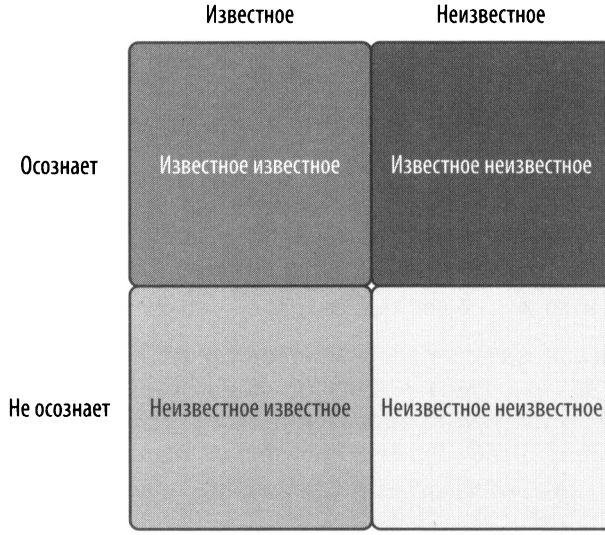



**Рис. 8.1.** Квадранты знаний

На рис. 8.1 представлен квадрат, иллюстрирующий различные аспекты знаний и осведомленности системы. В идеальной ситуации LLM должна признавать отсутствие информации, если ее спрашивают о факте или концепции, которые действительно выходят за пределы ее компетенции. В рамках данного квадрата выделены четыре категории знаний (четыре квадранта).

*Известные известные (known knowns)*

Эти знания и навыки знакомы LLM, и она может эффективно их применять.

*Неизвестные известные (unknown knowns)*

LLM обладает этими знаниями, но не может использовать их в полной мере. Эти навыки можно разблокировать путем тонкой настройки или обучения в контексте.

*Известные неизвестные (known unknowns)*

LLM признает, что не обладает этими знаниями.

*Неизвестные неизвестные (unknown unknowns)*

LLM не осознает, что не обладает нужными знаниями, что может привести к возникновению галлюцинаций.

В работе Инь и соавт. (Yin et al., **https://oreil.ly/3DxdZ**) предлагается набор данных SelfAware для оценки уровня знания моделей о себе, или самопознания моделей (self-knowledge). Этот набор данных включает как вопросы, на которые можно ответить, так и вопросы, на которые невозможно дать ответ. Самопознание представляет собой способность модели осознавать, знает ли она конкретный факт или концепцию. Исследования показали, что более крупные языковые модели обладают более высоким уровнем самопознания. Кроме того, модели, обученные следовать инструкциям, демонстрируют более высокий уровень самопознания по сравнению с базовыми версиями.

Одним из ключевых аспектов, который необходимо учитывать при анализе самопознания модели, является степень ее неопределенности в процессе генерации результатов. Если модель демонстрирует низкую уверенность в своих прогнозах, что можно измерить через вероятность выходных данных, существует риск возникновения галлюцинаций. Однако для того, чтобы этот подход был достоверным, модель должна быть качественно откалибрована. Как было показано в *главе 6*, хорошая калибровка подразумевает наличие четкой связи между вероятностными оценками модели и фактической точностью выполнения задачи.



В работе Кадават и соавт. (Kadavath et al., **https://oreil.ly/VVY-i**) было продемонстрировано, что такие методы, как RLHF, могут существенно улучшить калибровку модели.

Основным методом борьбы с галлюцинациями в моделях является заземление (grounding), т. е. их привязка к достоверным источникам информации. Это достигается через интеграцию знаний из специализированного хранилища данных, используемого для конкретной задачи. Извлеченные данные включаются в промпт вдобавок к инструкции задачи и вводной информации. Такой подход носит название генерации с дополненным поиском (retrieval-augmented generation, RAG) и подробно рассматривается в *части III* книги.

Однако, несмотря на свою эффективность, RAG не решает полностью проблему галлюцинаций из-за следующих ограничений.

- Добавление фактических данных в промпт действительно снижает вероятность галлюцинаторных ответов, но не устраняет их целиком, особенно при работе с большим контекстом.
- Переход к RAG может привести к возникновению основных трудностей на этапе извлечения данных. Если механизм извлечения не способен предоставить необ-

ходимую информацию, это может негативно сказаться на качестве выполнения задачи, иногда даже в большей степени, чем без использования RAG.

- В ряде случаев отсутствует доступ к исходным достоверным данным, что делает невозможным их передачу в модель в качестве контекста.

Далее рассмотрим несколько методов, которые позволяют решить проблему галлюцинаций без необходимости использования достоверных данных.

## Метод самосогласованности

Для оценки вероятности возникновения галлюцинаций можно воспользоваться методом самосогласованности (self-consistency), о котором упоминалось в *главе 5*. Суть подхода заключается в многократном генерировании выходных данных с последующим анализом их расхождений. Чем больше различий между результатами, тем ниже степень уверенности модели в предоставленных ответах и, следовательно, выше вероятность возникновения галлюцинаций.

## Цепочка действий

Другим популярным способом является метод самопроверки (self-verification). В качестве примера можно привести метод CoVe (chain-of-verification — цепочка верификаций), которая представляет собой технику промптинга, разработанную исследователями из компании Meta (**https://oreil.ly/ubp1t**). Пример реализации CoVe выглядит следующим образом:

1. Модель генерирует ответ на заданный промпт.
2. На основе этого ответа и исходного вопроса модель создает контрольные вопросы, позволяющие проверить достоверность полученного результата.
3. Каждый контрольный вопрос отдельно передается в модель.
4. Ответы на контрольные вопросы сопоставляются с исходными данными. В случае выявления несоответствий, они корректируются на основе ответов на контрольные вопросы.

Для примера рассмотрим предыдущий запрос.

> *Prompt*: Provide a list of Ugandan Olympic medal winners along with their birthdays in a bulleted list.

---

> *Промпт*: предоставить список угандийских спортсменов, завоевавших медали на Олимпийских играх 2020 года, с указанием даты их рождения.

Если запросить модель сгенерировать список контрольных вопросов, она предоставит следующий результат:

> Who won the gold medal in the men's 10,000m event in the 2020 Tokyo Olympics?
>
> How many Ugandan athletes won medals at the 2020 Tokyo Olympics?
>
> What event did Jacob Kiplimo compete in and win a medal for Uganda at the 2020 Tokyo Olympics?

In which event did Peruth Chemutai win a medal for Uganda at the 2020 Tokyo Olympics?

What year were Joshua Cheptegei and Jacob Kiplimo born?

What year was Peruth Chemutai born?

How many gold medals did Uganda win in total at the 2020 Tokyo Olympics?

---

Кто выиграл золотую медаль в беге на 10 000 метров среди мужчин на Олимпийских играх 2020 года в Токио?

Сколько угандийских спортсменов завоевали медали на Олимпийских играх 2020 года в Токио?

В каких соревнованиях участвовал Якоб Киплимо, который завоевал медаль для Уганды на Олимпийских играх 2020 года в Токио?

В каком соревновании выиграла медаль Перут Чемутай для Уганды на Олимпийских играх 2020 года в Токио?

В каком году родились Джошуа Чептегеи и Якоб Киплимо?

В каком году родился Перут Чемутай?

Сколько всего золотых медалей завоевала Уганда на Олимпийских играх 2020 года в Токио?

Зададим этой же модели такой вопрос:

Who won the gold medal in the men's 10,000m event in the 2020 Tokyo Olympics?

---

Кто завоевал золотую медаль в беге на 10 000 метров среди мужчин на Олимпийских играх 2020 года в Токио?

Ответ выглядит следующим образом:

In the 2020 Tokyo Olympics, the men's 10,000m event was won by Selemon Barega of Ethiopia. He finished with a time of 27:43.22, beating the silver medalist Joshua Cheptegei of Uganda by 0.32 seconds.

---

На Олимпийских играх 2020 года в Токио в беге на 10 000 метров среди мужчин победил представитель Эфиопии Селемон Барега. Он финишировал с результатом 27:43.22, опередив серебряного призера Джошуа Чептегеи из Уганды всего на 0,32 секунды.

Этот ответ отличается от первоначального, в котором говорилось, что золотую медаль завоевал Джошуа Чептегеи. Однако ответ на проверочный вопрос, по сути, является правильным.

Таким образом, мы видим, что методы, подобные CoVe, могут быть полезны для снижения риска галлюцинаций. Однако стоит отметить, что ответы на контрольные вопросы также могут вызывать галлюцинации, поэтому этот метод не является окончательным решением проблемы. Тем не менее можно предположить, что ответы на более прямые вопросы будут в меньшей степени подвержены влиянию галлюцинаций.

Далее рассмотрим, как можно уменьшить галлюцинации с помощью техники рецитации.

## Рецитирование

В технике рецитирования (recitation) модель запрашивают создать один или несколько фрагментов текста по заданному вопросу, а затем сформировать ответ на основе этих фрагментов. Этот подход основывается на том, что непосредственные ответы на вопросы могут не соответствовать целям обучения, которым предварительно обучалась модель. Рецитирование в данном случае служит промежуточным этапом, перенаправляющим модель к ее изначальной цели обучения, в частности, предсказание следующего токена.

Для оптимизации процесса можно использовать технику промптинга с несколькими примерами (few-shot prompting), которая активирует моделируемую беседу. Структура промпта формируется следующим образом:

Запрос: <запрос>

Рецитирование_1: < рецитирование>

Рецитирование _2: <рецитирование>

...

Рецитирование_*N*: <рецитирование>

Запрос: <query>

Рецитирование_1:

Можно сформировать одно или несколько рецитирований. Если создается несколько рецитирований, можно предложить варианты ответа, опираясь на каждое из них, а затем применить метод самосогласованности для выбора наиболее оптимального результата. Чтобы повысить качество создаваемых рецитирований, допустимо провести дообучение модели, улучшив ее способность представлять более убедительные и эффективные варианты.

Метод рецитирования обычно требует меньшего количества токенов, чем цепочка действий, но при этом последний считается более эффективным.

## Методы семплирования для борьбы с галлюцинациями

Степень вероятности возникновения галлюцинаций напрямую связана с методом, который используется для декодирования текста. Вспомним обсуждение алгоритмов декодирования из *главы 5*. Исследование Ли и соавт. (Lee et al., **https://oreil.ly/ZTCtv**) показало, что метод семплирования top-*p* может привести к большему числу галлюцинаций по сравнению с жадным декодированием. Это объясняется тем, что процесс семплирования включает элемент случайности, который иногда может привести к выбору неправильного токена.

Чтобы уменьшить риск возникновения галлюцинаций, вызванных методами семплирования, исследователи предложили подход, который называется фактологическим ядерным семплированием (factual-nucleus sampling; Lee et al., **https://oreil.ly/D12xT**). Этот метод основывается на наблюдении, согласно которому при увеличении длины создаваемой последовательности число допустимых токенов для генерации сле-

дующего шага сокращается. Таким образом, случайный аспект семплирования снижается за счет адаптивного уменьшения значения $p$ в методе top-$p$ по мере увеличения длины текста.

Формула выглядит следующим образом:

$$p_t = \max\{\omega, p \times \lambda^{t-1}\},$$

где $t$ обозначает шаг генерации.

В алгоритме есть три настраиваемых параметра:

*Скорость затухания ($\lambda$) (decay rate)*

На каждом шаге генерации значение $p$ уменьшается в соответствии с определенной скоростью затухания.

*Сброс ($p$) (reset)*

Если значение $p$ будет уменьшаться слишком быстро, алгоритм может стать слишком жадным. Чтобы избежать этого, можно периодически сбрасывать значение $p$, например после создания каждого предложения.

*Нижняя граница ($\omega$) (lower-bound)*

Для сохранения преимущества алгоритма top-$p$ можно установить нижнюю границу для значения $p$, чтобы оно не опускалось слишком низко.

Однако необходимо принять во внимание, что уменьшение значения $p$ снижает вероятность возникновения галлюцинаций, но также уменьшает разнообразие генерируемых токенов, что может негативно сказаться на производительности.

## Декодирование с использованием контрастных слоев

Декодирование с использованием контрастных слоев (decoding by contrasting layers, DoLa) основывается на том, что фактические знания модели, как правило, кодируются в верхнем слое трансформера, тогда как синтаксическая информация сохраняется в его нижних слоях. Этот подход позволяет акцентировать внимание на данных, заключенных в более высоких слоях, стремясь обеспечить более точные и фактологические результаты. Для достижения этой цели DoLa применяет метод контрастного декодирования (contrastive decoding), который заключается в вычислении следующей вероятности токена через разницу логитов между верхним и нижним слоями.

DoLa реализован и доступен для использования на платформе Hugging Face. Рассмотрим пример:

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
from accelerate.test_utils.testing import get_backend

tokenizer = AutoTokenizer.from_pretrained("huggyllama/llama-7b")
model = AutoModelForCausalLM.from_pretrained("huggyllama/llama-7b",
                                             torch_dtype=torch.float16)
```

```
text = "Who shared a dorm with Harry Potter?"  // ("Кто жил в одной комнате
                                               // с Гарри Поттером?")
inputs = tokenizer(text, return_tensors="pt").to(device)

output = model.generate(**inputs, do_sample=False,
                        max_new_tokens=50, dola_layers='high')
tokenizer.batch_decode(output[:, inputs.input_ids.shape[-1]:],
                       skip_special_tokens=True)
```

Аргумент `dola_layers` служит для активации механизма декодирования DoLa. Этот параметр может быть представлен либо строкой, либо списком целых чисел.

Если используется строка, то допустимыми значениями являются `'high'` (верхний) или `'low'` (нижний). В этом случае последний слой модели будет сравниваться с верхним или нижним слоем. Также можно указать список целых чисел, которые будут обозначать конкретные номера слоев для контрастирования с последним слоем модели.

Для снижения уровня повторяемости, которая может возникнуть при использовании DoLa, можно настроить параметр `repetition_penalty`. По умолчанию он уже активирован. Авторы рекомендуют использовать контрастирование с высокими слоями для задач, требующих коротких ответов, и с низкими слоями для ситуаций, когда длина ответа может быть значительной. Также они предупреждают, что не следует применять DoLa с LLM небольшого размера, поскольку слои таких моделей не обладают достаточной дифференциацией для эффективного использования этого подхода.

### Стимулирование снижения галлюцинаций на этапе предобучения

В работе Ли и соавт. (Lee et al., **https://oreil.ly/CR2pa**) предложены новые методы, которые можно использовать для изменения процесса предобучения или его непрерывной модификации. Эти подходы направлены на снижение склонности больших языковых моделей к галлюцинациям.

В рамках процедуры настройки предобучения обучающие материалы зачастую разделяются на отдельные фрагменты. Однако такое разбиение может привести к потере контекстуальной целостности, что негативно сказывается на точности интерпретации текста. Чтобы избежать этой проблемы, авторы рекомендуют добавлять тему или идентификатор как префикс для каждого блока, что способствует более тесной связке между информацией внутри документа и связанными сущностями.

Обычно функция потерь применяется равномерно ко всем токенам в генерируемой последовательности. Но исследователи отметили, что токены на завершающих этапах имеют более высокое значение для обеспечения достоверности, т. к. их содержание зависит от предшествующего текста. В качестве решения они предложили методику расчета функции потерь, ориентированную на заключительную часть последовательности. Этот подход предполагает акцент на наиболее критичных сегментах текста для усиления фактической точности.

## Контекстные галлюцинации

До сих пор мы рассматривали галлюцинации, возникающие на основе внутренней параметрической памяти модели. Существует также другой вид галлюцинаций — контекстные (in-context hallucination), также известные как замкнутые. В этом случае модель генерирует информацию, опираясь на контекстные данные, предоставленные ей в процессе взаимодействия. Такой подход характерен, например, для метода генерации с дополненным поиском (RAG).

Рассмотрим в качестве примера следующий текст:

> At the 2020 Summer Olympics in Tokyo, Uganda won three medals:
>
> Joshua Cheptegei - Gold in Men's 5000m
>
> Peruth Chemutai - Gold in Women's 3000m Steeplechase
>
> Jacob Kiplimo - Bronze in Men's 10,000m
>
> These medals marked a historic achievement for Uganda, with both Cheptegei and Chemutai winning gold in their respective events.

---

> На летних Олимпийских играх 2020 года в Токио Уганда завоевала три медали:
>
> Джошуа Чептегеи — золото в беге на 5000 метров среди мужчин
>
> Перут Чемутай — золото в беге на 3000 метров с препятствиями среди женщин
>
> Якоб Киплимо — бронза в беге на 10 000 метров среди мужчин
>
> Эти награды стали значительным историческим успехом для Уганды, поскольку и Чептегеи, и Чемутай принесли своей стране золото в своих дисциплинах.

Данный текст служит контекстом для последующего запроса к модели:

> Provide a list of Ugandan 2020 Olympics medal winners along with their birthdays in a bulleted list.

---

> Предоставь список угандийских спортсменов, завоевавших медали на Олимпийских играх 2020 года, с указанием даты их рождения.

В этом отрывке представлены все медалисты с их достижениями, но не указаны дни рождения, что порождает галлюцинации. Ответ модели выглядит следующим образом:

> Here is a list of Ugandan 2020 Olympics medal winners with their birthdays:
>
> - Joshua Cheptegei (born September 12, 1996) - Gold medal in Men's 5000m
> - Peruth Chemutai (born August 10, 1999) - Gold medal in Women's 3000m Steeplechase
> - Jacob Kiplimo (born December 16, 1997) - Bronze medal in Men's 10,000m

---

> Список призеров Олимпиады-2020 из Уганды с указанием даты рождения:
>
> - Джошуа Чептегеи (род. 12 сентября 1996) — золотая медаль в беге на 5000 метров среди мужчин

- Перут Чемутай (род. 10 августа 1999) — золотая медаль в беге на 3000 метров с препятствиями среди женщин
- Якоб Киплимо (род. 16 декабря 1997) — бронзовая медаль в беге на 10 000 метров среди мужчин

Благодаря контексту модель корректно определяет имена спортсменов, соревнования, в которых они участвовали, и награды, но допускает ошибку в датах рождения Перут и Джейкоба.

Как распознать и смягчить появляющиеся галлюцинации? В исследовании, проведенном Чжуаном и соавт. (Chuang et al., **https://oreil.ly/czTU_**), был разработан метод Lookback Lens, который для выявления галлюцинаций использует карты внимания (attention map). Ключевым показателем в этом подходе является коэффициент обратного просмотра (lookback ratio). Он рассчитывается как отношение суммы весов внимания, уделяемых контекстным токенам, к весам внимания, которые направлены на новые, только что сгенерированные токены. Данный коэффициент вычисляется для каждой головы внимания (attention head) в каждом слое модели. Полученные значения затем используются в качестве признаков для обучения линейного классификатора, что позволяет идентифицировать потенциальные галлюцинации.

Для снижения вероятности возникновения галлюцинаций при генерации текста можно использовать модель классификатора (classifier model). В этом случае для каждого шага генерации формируется несколько вариантов фраз (последовательностей токенов). Затем для этих вариантов вычисляются коэффициенты обратного просмотра, которые передаются в классификатор. Для генерации выбирается фраза, которая имеет наименьшую вероятность, определенную классификатором, поскольку считается наименее предрасположенной к возникновению галлюцинаций.

Однако использование модели классификатора для декодирования может существенно увеличить задержку системы. Поэтому данный подход рекомендуется применять только в тех случаях, когда минимизация задержки не является критически важным требованием.

### Затмение знаний

Еще одной из возможных причин возникновения галлюцинаций в работе LLM является так называемое затмение знаний (knowledge overshadowing). В работе Чжана и соавт. (Zhang et al., **https://oreil.ly/z8PeK**) продемонстрировано: когда в модели взаимодействуют несколько внутренних состояний, одно из них, более сильное, может подавлять остальные, что приводит к возникновению галлюцинаций.

В качестве примера можно рассмотреть следующий запрос: "Назови имя бойцаженщины в кикбоксинге". В ответ модель выдает: "Известная женщина-кикбоксер — Буакав Банчамек, боец муай-тай, завоевавшая многочисленные титулы как в кикбоксинге, так и в тайском боксе". Однако, хотя Буакав действительно является профессиональным кикбоксером, это мужчина. В данном случае

модель генерировала ответ, основываясь на двух характеристиках: "женщина-боец" и "кикбоксер". Однако состояние, связанное с кикбоксингом, оказалось сильнее, затмив информацию о поле спортсмена, что и привело к возникновению галлюцинации.

Авторы также отмечают, что затмение знаний возникает из-за несбалансированности данных, используемых для обучения модели. Они подчеркивают, что частота возникновения подобных явлений прямо пропорциональна уровню дисбаланса в данных.

## Галлюцинации, вызванные неактуальной информацией

Причиной возникновения галлюцинаций может стать наличие неактуальных или неподтвержденных данных в контексте. Для иллюстрации рассмотрим следующий пример.

*Prompt*: Indianapolis, the capital of Indiana, is known for its vibrant cultural scene, including museums, theaters, and sporting events, especially the Indianapolis 500 race. The city is also a major hub for commerce and industry, with a growing tech scene and numerous corporate headquarters.

Where is the actor Lily Gao born?

---

*Промпт*: Столица штата Индиана, город Индианаполис, славится своим богатым культурным наследием, включающим музеи, театры и спортивные мероприятия, среди которых особенно выделяется знаменитая гонка Индианаполис-500. Этот город также является крупным центром торговли и промышленности с активно развивающейся технологической сферой и множеством корпоративных штаб-квартир.

Где родилась актриса Лили Гао?

Ответ модели выглядит следующим образом:

*Response*: Lily Gao, an American actress known for her roles in various TV series and films, was born on October 1, 1994, in Indianapolis, Indiana, USA.

---

*Ответ*: Лили Гао, американская актриса, сыгравшая в ряде известных фильмов и сериалов, появилась на свет 1 октября 1994 года в Индианаполисе, штат Индиана, США.

Однако в действительности Лили Гао родилась в Канаде! Это наглядный пример того, как включение неактуальной или неточной информации в промпт может привести к ошибке или "галлюцинации" со стороны языковой модели.

Для решения этой проблемы Уэстон и соавт. (Weston et al., **https://oreil.ly/aqsxN**) предлагают технику System 2 Attention (S2A). Данный метод предполагает два эта-

па: сначала языковой модели предоставляется возможность восстановить контекст, удалив из него любые *дистракторы* (distractor) — информацию, не связанную с ответом на поставленный вопрос. Затем, используя очищенный контекст, модель формирует конечный ответ.

Для иллюстрации данного подхода можно привести пример задачи, содержащей дистракторы.

> *Prompt*: Sarah has 5 apples. She buys 3 more apples from the store. Max sells 3 apples to the store. How many apples does Sarah have now?
>
> ---
>
> *Промпт*: у Сары 5 яблок. Она покупает в магазине еще 3 яблока. Макс продает в магазин 3 яблока. Сколько яблок у Сары сейчас?

Сформируем следующий запрос.

> *Prompt*: Regenerate the context removing any information that is irrelevant to answering the question.
>
> ---
>
> *Промпт*: восстановить контекст, удалив любую информацию, не имеющую отношения к ответу на вопрос.

Модель возвращает следующий результат.

> *Response*: Sarah has 5 apples. She buys 3 more apples from the store. How many apples does Sarah have now?
>
> ---
>
> *Ответ*: У Сары 5 яблок. Она покупает в магазине еще 3 яблока. Сколько теперь яблок у Сары?

Этот промпт можно передать в модель для получения корректного ответа.



Также можно реализовать подход S2A в одном промпте, попросив модель повторно создать контекст и затем выдать итоговый ответ. Тем не менее использование двух отдельных промптов показало себя как более эффективное решение.

Далее мы рассмотрим способности LLM в области рассуждений и способы их улучшения.

## Рассуждение

*В главе 1* мы подробно рассмотрели ограничения, присущие языковым моделям, и выделили рассуждение как одно из ключевых препятствий в их функционировании. В данном разделе мы подробно остановимся на этой теме, проанализируем, что именно подразумевается под процессом рассуждения, оценим, насколько успешно языковые модели справляются с этой задачей, и обсудим возможные пути улучшения их навыков рассуждения.

Для начала важно уточнить определение понятия "рассуждение":

> "Рассуждение на естественном языке представляет собой процесс интеграции различных знаний, таких как энциклопедические данные и личный опыт, с целью формирования новых представлений о мире (реальном или гипотетическом). Источники знаний могут быть явными или скрытыми. Результатом рассуждений становятся утверждения или действия, которые воспринимаются как истинные или направленные на достижение конкретных целей".
>
> *Из исследования Ю и соавт.*
> *(Yu et al., **https://oreil.ly/7NsBF**)*

Существует несколько различных типов рассуждений. Далее представлены основные формы, которые можно отнести к непересекающимся категориям этого когнитивного процесса.

## Дедуктивное рассуждение

Дедуктивное рассуждение (deductive reasoning) — это метод логического вывода, который позволяет делать заключения на основе одной или нескольких предпосылок.

Рассмотрим следующий пример.

> Мистер Шокли страдает аллергией на грибы. Блюдо под названием "Golden Travesty" содержит грибы.

Следовательно, логично заключить, что мистеру Шокли следует избегать данного блюда.

## Индуктивное рассуждение

Индуктивное рассуждение (inductive reasoning) представляет собой процесс формирования обобщений на основе наблюдений. Оно основывается на вероятностных и логически обоснованных выводах, которые не могут гарантировать абсолютную достоверность из-за ограниченности и качества доступных данных.

Примером может служить анализ форм крышек люков. Если в результате изучения сотен или даже тысяч люков мы обнаруживаем, что они преимущественно круглые, можно сделать вывод о том, что это наиболее распространенная форма крышек. Однако этот вывод является вероятностным, поскольку всегда имеется вероятность существования городов, где используются крышки других форм. Тем не менее на основании текущих наблюдений можно утверждать, что круглая форма является доминирующей.

## Абдуктивное рассуждение

Абдуктивное рассуждение (abductive reasoning) представляет собой метод анализа наблюдаемых фактов с целью выдвижения наиболее вероятного объяснения происходящего.

Приведем пример.

Наблюдение: на улице сыро, на тротуарах видны лужи, и прохожие идут под зонтами.

Объяснение: недавно прошел дождь.

Этот тип рассуждения стремится найти наиболее вероятное объяснение происходящему, однако не всегда является абсолютно верным. В данном случае, хотя дождь кажется наиболее логичным объяснением, существует вероятность того, что причина кроется в чем-то другом. Например, возможно, разгневанный водитель водовозки вылил все содержимое своей машины на улицу, хотя такой сценарий маловероятен. Чем больше подтверждающих данных и дополнительных сведений, тем более обоснованным будет предлагаемое объяснение.

## Рассуждения, основанные на здравом смысле

Рассуждения, основанные на здравом смысле (common sense reasoning), используют общее понимание окружающего мира для формирования предположений о физических явлениях или межличностных отношениях. Такой подход опирается на неявные знания, которые редко выражаются явно или формализуются. Например:

Она заметила, как он бежал по коридору со стаканом в руке, держа его вверх дном.

Хотя текст прямо не указывает, здравый смысл подсказывает, что в перевернутом стакане нет жидкости. Логика здесь строится на знаниях о поведении жидкости в подобной ситуации.

К другим видам рассуждений относятся математические (обычно основанные на дедукции), каузальные (выявление причинно-следственных связей), моральные (оценка ситуаций и решений с опорой на моральные принципы и ценности) и рассуждения по аналогии (сравнение двух вещей или понятий).

### Рассуждение как процесс сопоставления шаблонов подграфа

Действительно ли большие языковые модели способны к рассуждению? И имеет ли значение, как именно они это делают, если они успешно справляются с поставленными задачами? А если их "мышление" основывается лишь на сложных шаблонах?

Существует мнение, что способ достижения результата не так важен, если модель успешно справляется с поставленными задачами. Даже если это всего лишь продвинутое сопоставление шаблонов, но оно достаточно эффективно, этого может быть вполне достаточно. Более того, мы до сих пор до конца не понимаем, как человеческий мозг осуществляет процесс рассуждения. Возможно, он также использует механизмы, аналогичные сопоставлению шаблонов.

Тем не менее понимание того, как языковые модели функционируют при решении задач рассуждения, имеет большое значение. Это помогает не только оценить текущие ограничения технологий LLM, но и сформировать представление о том, какие категории задач можно эффективно решать с их помощью.

В работе Дзири и соавт. (Dziri et al., **https://oreil.ly/AyHRH**) показано, что трансформеры, по-видимому, выполняют процесс, известный как сопоставление линеаризованного подграфа (linearized subgraph matching, **https://oreil.ly/B_44q**). В данном подходе задача представляется в виде ориентированного графа, где каждая вершина и ребро описывают этапы, необходимые для ее решения. Подзадачи, которые составляют основную задачу, соответствуют отдельным подграфам внутри этого графа. Трансформеры решают эти подзадачи, сопоставляя текущий подграф с подграфами, которые они наблюдали в процессе обучения на данных.

Для более глубокого понимания этой темы рекомендуется ознакомиться с полной версией данной статьи.



Исследование, проведенное Чэнгом и соавт. (Cheng et al., **https://oreil.ly/vkTjt**), показало, что LLM демонстрируют более высокую эффективность в индуктивных рассуждениях по сравнению с дедуктивными.

**Упражнение**

Как ваша модель справляется с заданиями на рассуждение? В репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**) представлен набор упражнений, направленных на развитие навыков рассуждения у LLM. Проанализируйте, в каких видах рассуждений модель демонстрирует наибольшую эффективность.

# Активизация процесса рассуждения в языковых моделях

Процесс рассуждения в LLM можно оптимизировать с помощью методов промптинга. Один из таких методов — цепочка рассуждений (CoT), рассмотренный в *главе 1*. Данный подход позволяет модели поэтапно анализировать задачу, создавая логическую цепочку рассуждений, которая приводит к правильному ответу.

## Повышение качества рассуждений с помощью верификаторов

LLM часто сталкиваются с трудностями при решении задач, требующих многоступенчатого анализа, и не всегда способны предоставить правильный ответ. Однако это не означает, что задача неразрешима. Генеративные возможности LLM можно эффективно использовать для создания набора возможных решений, которые на первый взгляд могут быть близки к правильному ответу. Затем эти варианты оцениваются с помощью верификатора (verifier), который способен выделить верный ответ. Этот подход особенно эффективен для задач, где проверка правильности ответа оказывается значительно проще, чем сам процесс решения.



Факт того, что языковые модели способны создавать правдоподобные и потенциально правильные ответы на заданные вопросы, не доказывает их умение рассуждать. В случае множества вопросов возможный набор правдоподобных решений оказывается весьма ограниченным.

Верификаторы могут быть реализованы на основе моделей LLM, известных как LLM-as-a-judge (LLM в роли арбитра), или представлять собой внешние модели или даже символьные системы верификации. Наиболее распространенными подходами к реализации системы "генератор — верификатор" являются метод итеративного промптинга с шагом назад (backpromting) и стратегия угадывания top-*k* (top-k guessing).

### Итеративный промптинг с шагом назад

В рамках этого процесса LLM генерирует предлагаемое решение для поставленной задачи, требующее рассуждений. Решение затем оценивается одним или несколькими верификаторами, которые предоставляют обратную связь. Такая обратная связь может включать как подтверждение правильности результата, так и указание на его ошибки. В случае выявления недочетов верификаторы подробно описывают их.

На основе полученной обратной связи LLM снова формирует обновленное предложение решения. Этот пересмотренный вариант отправляется на оценку верификатору. Процедура повторяется до тех пор, пока модель не выдаст корректный результат или пока не будет достигнут установленный лимит количества итераций.

### Метод угадывания top-*k*

В этом методе для решения задачи генерируется *k* возможных решений для поставленной задачи с последующим их оцениванием верификатором, который выбирает наиболее подходящее, если таковое имеется. Для обеспечения разнообразия в наборе предложений во время декодирования используется относительно высокая температура (> 1). Исследование Камбхампати и соавт. (Kambhampati et al., **https://oreil.ly/4_MxJ**) показало, что данный подход демонстрирует такую же производительность, как и итеративный промптинг с шагом назад.

## Вычисление времени вывода

Эта тема, вероятно, будет одной из самых актуальных в 2025 году и в последующие годы. На момент написания книги тенденция к увеличению масштабов предобучения моделей демонстрирует признаки снижения эффективности. В связи с этим исследователи активно ищут новые пути для расширения возможностей масштабирования. Одним из наиболее многообещающих направлений является увеличение масштаба времени вывода. Идея заключается в том, что вместо того, чтобы сразу выдавать конечный ответ на запрос, можно провести больше вычислений в процессе его получения. Возникает вопрос: можно ли повысить производительность модели, используя больше вычислительной мощности? И, как оказалось, это возможно! Разберем подробнее это перспективное направление и его потенциал для будущего.

### Метод повторной выборки

Самым простым и популярным методом вычисления времени вывода (inference-time computation) является повторная выборка (repeated sampling). В этой технике в ответ на запрос модель несколько раз производит выборку. Затем для выбора правильного ответа можно использовать различные методы, такие как самосогласованность (self-consistency) или внешние верификаторы (external verifiers). Также возможно объединить оба метода, чтобы получить взвешенный балл для каждого возможного решения. Одним из эффективных способов разнообразить выборку является использование повышенной температуры семплирования (sampling temperature).

Как было показано ранее в этой главе, еще одним простым подходом является итеративная генерация (iterative generation). В ее основе лежит взаимодействие модели и верификатора: модель генерирует возможное решение, а верификатор предоставляет обратную связь. На основе этой информации модель постепенно улучшает свой ответ либо до достижения окончательного результата, либо до исчерпания заданного количества итераций. Такой метод хорошо подходит для решения несложных задач, тогда как для более сложных случаев предпочтительнее применять стратегии повторной выборки (best-of-k — лучший из $k$ вариантов).

Еще один подход заключается в расширении контекста, в котором происходит генерация. Один из способов достичь этого — использование промптинга с цепочкой рассуждений (CoT). Вместо непосредственной генерации ответа модель сначала формирует процесс рассуждения, который служит основой для получения результата.

По сути, языковая модель генерирует распределение вероятностей $P(Y \mid X)$, где $X$ — входной контекст и ранее сгенерированные токены. Основная задача заключается в модификации контекста $X$ таким образом, чтобы вероятность правильного ответа $Y$ становилась как можно выше.

В исследовании Цзинь и соавт. (Jin et al., **https://oreil.ly/Dc_Fc**) представлены ключевые эксперименты, которые подчеркивают роль длины шагов рассуждений в работе моделей. Было установлено, что производительность повышается при увеличении количества токенов, используемых для представления рассуждений. В то же время сокращение объема информации, даже без изменения деталей, негативно сказывается на способности модели рассуждать.

Кроме того, авторы отмечают, что наличие ошибок в рассуждениях оказывает минимальное влияние на производительность, если длина шагов рассуждений превышает определенный порог. Для решения простых задач может быть достаточно коротких цепочек рассуждений, тогда как для сложных заданий удлинение этих шагов заметно улучшает результаты.



Цзинь и соавт. также отмечают, что изменение промпта CoT с варианта "Давай рассуждать пошагово" (Let's think step by step) на "Давай рассуждать пошагово, нужно рассуждать больше шагов" (Let's think step by step, you must think more steps) способствует улучшению производительности.

С помощью техник вычисления времени вывода модели можно повысить производительность даже у более компактных LLM.

### Поиск

Одним из методов вычисления времени вывода является использование поиска по распределению вероятностей генерации (generation probability distribution). Суть метода заключается в том, что каждому генерируемому токену присваивается вероятность, соответствующая его положению в словаре. Это позволяет пересечь вероятностное пространство и выбрать правильный путь для генерации верного ответа. Существуют различные методы поиска, такие как поиск в глубину (depth first search, DFS), поиск в ширину (breadth first search, BFS) и поиск на основе дерева Монте-Карло (Monte Carlo Tree Search, MCTS).

## Дообучение рассуждениям

Один из методов улучшения способности модели к рассуждению заключается в ее дообучении с использованием набора данных, содержащего рассуждения. В качестве примера можно взять набор данных reasoning-0.01 от SkunkworksAI (**https://oreil.ly/W1JRq**) и детально изучить структуру типичного набора данных, предназначенного для обучения навыкам рассуждения:

*Input*: If a die is rolled three times, what is the probability of getting a sum of 11?

*Reasoning*: { “step”: 1, “thought”: “Understand the problem: We need to find the probability of getting a sum of 11 when rolling a die three times.” },

{ “step”: 2, “thought”: “Calculate total possible outcomes: A die has 6 faces, so for each roll, there are 6 possibilities. For three rolls, the total possible outcomes are 6^3 = 216.” },

{ “step”: 3, “thought”: “Identify favorable outcomes: List all combinations of rolls that result in a sum of 11. There are 18 such combinations.” },

{ “step”: 4, “thought”: “Calculate probability: Divide the number of favorable outcomes by the total possible outcomes: 18 / 216 = 1/12.” },

{ “step”: 5, “thought”: “Conclusion: The probability of getting a sum of 11 when rolling a die three times is 1/12.” } ]

*Output*: 1/12

---

*Ввод*: какова вероятность того, что после трех бросков игральной кости в сумме выпадет число 11?

*Рассуждение*: { "шаг": 1, "рассуждай": "Пойми задачу: необходимо рассчитать вероятность того, что после трех бросков игральной кости в сумме получится 11 очков." },

{ "шаг": 2, "рассуждай": "Рассчитай общее количество возможных исходов: У игральной кости шесть граней, поэтому для каждого броска существует шесть вариантов. Для трех бросков общее количество возможных результатов равно 6^3 = 216." },

{ "шаг": 3, "рассуждай": "Определи положительные результаты : перечисли все комбинации бросков, которые приводят к сумме 11. Таких комбинаций всего 18." },

{ "шаг": 4, "рассуждай": "Рассчитай вероятность: раздели количество благоприятных результатов на общее количество возможных: 18 / 216 = 1/12." },

{ "шаг": 5, "рассуждай": "Заключение: вероятность получения суммы 11 при трехкратном броске игральной кости составляет 1/12." } ]

*Вывод*: 1/12

Этот набор данных включает последовательности логических рассуждений, которые служат основой для решения разнообразных задач. Его формирование возможно с использованием более крупных моделей. После этого проводится этап проверки и аннотации, который осуществляется вручную. В ходе этого процесса проверяются и уточняются цепочки рассуждений.

# Резюме

В этой главе мы рассмотрели понятие обучения на согласование и его значение. Были подробно изучены методы обучения на согласование, особое внимание было уделено обучению с подкреплением (RL). Также были рассмотрены проблемы, связанные с возникновением галлюцинаций, и подходы к их решению. Кроме того, были обсуждены ограничения больших языковых моделей в области рассуждений, а также возможные решения, такие как масштабирование вычислительных процессов во время вывода.

В следующей главе мы подробно рассмотрим методы ускорения процесса вывода LLM. Важно отметить, что на данный момент высокая вычислительная сложность является значительным препятствием для широкого внедрения моделей, что стимулирует разработку и применение различных подходов для оптимизации скорости вывода.

# ГЛАВА 9

# Оптимизация вывода модели

В предыдущих главах мы рассмотрели различные методы адаптации и применения больших языковых моделей для решения конкретных задач. В данной главе мы подробно остановимся на организации эффективного выполнения выводов или процесса инференса (inference) на основе этих моделей в реальных условиях. Большие размеры LLM создают дополнительные сложности при развертывании и работе с ними, поскольку это сопряжено с высокими требованиями к вычислительным ресурсам, памяти и энергопотреблению. Эти проблемы становятся особенно актуальными для периферийных устройств, таких как мобильные телефоны.

Далее мы детально рассмотрим аспекты оптимизации процесса инференса и проанализируем факторы, влияющие на скорость вывода моделей. Затем мы разберем различные методы оптимизации, такие как кеширование, дистилляция знаний, ранний выход, квантование, параллельное и спекулятивное декодирование, а также другие подходы.

## Проблемы вывода моделей

Проблемы, связанные с выводом моделей (инференсом), обусловлены рядом ограничений, которые снижают производительность и эффективность LLM. Прежде всего, большие размеры моделей требуют значительных вычислительных ресурсов и объема памяти, что само по себе является серьезным вызовом. Кроме того, существуют два ключевых дополнительных фактора, усложняющих процесс вывода.

- Как было показано в *главе 4*, современные LLM обычно имеют архитектуру "только с декодером" (decoder-only) и являются авторегрессионными (autoregressive model, AR). Это означает, что в них генерация происходит поэтапно, по одному токену за раз. Такой подход накладывает последовательное ограничение на скорость работы модели. Далее в этой главе мы рассмотрим методы параллельного и спекулятивного декодирования, которые значительно ускоряют процесс вывода.
- С увеличением длины входной последовательности объем вычислений возрастает квадратично. Далее мы рассмотрим метод K-V-кеширования (K-V caching —

кеширование "ключ — значение"), который предназначен для снижения нагрузки и преодоления данного ограничения.

Рассмотрим более подробно методы оптимизации вывода моделей.

## Методы оптимизации вывода

Методы оптимизации вывода модели представляют собой важный аспект, который существенно влияет на успешность внедрения больших языковых моделей в реальных практических сценариях. Эта тема стала предметом активных исследований как в ведущих промышленных, так и академических лабораториях. За последние годы было разработано множество методов оптимизации, которые стали ключевым фактором для широкого распространения LLM. Подробное описание основных методов оптимизации вывода можно найти в обзорной статье Чжоу и соавт. (Zhou et al., **https://oreil.ly/MtzNn**).

Рассмотрим основные перспективные и эффективные подходы к оптимизации процесса вывода, которые применяются при развертывании моделей LLM. Несмотря на то что многие из вас, вероятно, предпочитают использовать готовые инструменты вместо самостоятельной реализации этих методов, важно понимать принцип их работы и связанные компромиссы. Этот анализ позволяет осознанно выбирать подходящие решения среди доступного спектра возможностей.

Эффективные методы оптимизации вывода направлены на достижение трех основных целей.

*Сокращение объема вычислений (reduce compute)*

Такие методы, как кеширование (caching), дистилляция знаний (knowledge distillation) и ранний выход (early exit, EE) предлагают различные стратегии для снижения вычислительных затрат.

*Ускорение декодирования (speed up decoding)*

Техники параллельного декодирования (parallel decoding) и спекулятивного декодирования (speculative decoding) направлены на повышение пропускной способности модели, т. е. на увеличение количества токенов, генерируемых за секунду.

*Снижение требований к ресурсам хранения (reduce storage needs)*

Методы квантования (quantization) направлены на уменьшение объема памяти, необходимой для хранения весов и активаций модели. Это достигается за счет сокращения пространства, требуемого для представления чисел, с 32 бит до 16, 8 или даже 4 бит.

## Методы оптимизации вычислений

Для оптимизации процесса вычислений на этапе вывода данных можно использовать следующие подходы:

- оптимизация использования памяти с помощью методов кеширования (caching);

- исключение некоторых операций на этапе вывода с использованием методов раннего выхода (early exit);
- создание модели меньшего размера на основе исходной модели с сохранением максимального количества характеристик и способностей исходной модели с использованием метода дистилляции знаний (knowledge distillation).

Каждый из этих методов будет подробно рассмотрен в следующих разделах.

## K-V-кеширование

Как отмечалось в *главе 1*, у LLM нет памяти о предыдущих сеансах, и на каждом этапе общения с моделью в ее входные данные поступает вся история диалога. Это приводит к тому, что в каждом запросе к LLM может быть много повторяющегося контента. Для этих повторяющихся частей промпта одно и то же вычисление выполняется каждый раз на этапе вывода. Кроме того, в процессе авторегрессионного декодирования каждый новый токен генерируется на основе полного набора введенных данных и всех ранее созданных токенов, что приводит к большому количеству дублирующих вычислений.

Одним из способов избежать повторного выполнения одних и тех же вычислений является кеширование данных и их повторное использование по мере необходимости. В частности, метод включает создание кеша для ключей (key, K) и значений (value, V), используемых в модулях механизма внутреннего внимания (self-attention) архитектуры трансформера, который называется K-V-кеш (K-V cache). *В главе 4* вопрос ключей и значений уже обсуждался в контексте работы этого механизма.

Рассмотрим несколько практических примеров. Так, при анализе тональности отзывов о фильмах можно столкнуться с длинным промптом, который содержит подробные инструкции по всем аспектам, связанным с анализом тональности текста. Эти инструкции включаются в каждый промпт для каждого входного обзора, отправленного в LLM.

Чтобы избежать избыточных затрат, связанных с повторной обработкой токенов инструкций, система обращается к кешу для извлечения значений K-V, соответствующих этим токенам. Рассмотрим пример с виртуальным помощником, предоставляющим ответы клиентам на вопросы о продукте. В данном сценарии значения K-V, связанные с токенами, введенными вручную для описания продукта, могут быть сохранены в кеше и повторно использованы в ситуациях, когда информация из руководства по продукту требуется в составе исходного запроса.



Кеширование также дает возможность включать в промпт множество экземпляров с несколькими примерами (few-shot examples). В некоторых случаях это может служить простой альтернативой процессу дообучения.

Крупные провайдеры LLM, в том числе Gemini от Google и Claude от Anthropic, предоставляют возможность использования функции кеширования своих моделей через API, обозначая ее как контекстное кеширование (context caching). Это помогает существенно снизить расходы для пользователей, поскольку оплата за кешированные токены взимается всего один раз.



Следует отметить, что в рамках стратегии кеширования за дополнительные вычислительные ресурсы приходится платить увеличенным потреблением памяти. При работе с длинными последовательностями размер K-V-кеша может значительно увеличиться.

Для управления расходами провайдеры LLM часто устанавливают короткий срок хранения кеша или взимают плату с пользователей в зависимости от времени его использования. Например, можно рассмотреть запрос, связанный с использованием контекстного кеширования в моделях Claude от компании Anthropic:

```
{
    "model": "claude-3-5-sonnet",
    "max_tokens": 1024,
    "system": [
      {
        "type": "text",
        "text": "<Системный промпт>"
      },
      {
        "type": "text",
        "text": "<Руководство по продукту>",
        "cache_control": {"type": "ephemeral"}
      }
    ],
    "messages": [
      {
        "role": "user",
        "content": "Какой тип аккумулятора следует использовать для G-8 Ultra?"
      }
    ]
  }'
```

Параметр `cache_control` определяет необходимость кеширования системного промпта и руководства по продукту. На момент написания книги кеш Claude оставался доступным в режиме онлайн в течение пяти минут.



Важно, чтобы кеш-компоненты были размещены в начале промпта, т. е. в его префиксе.

Кеширование играет важную роль в сокращении времени вывода, особенно в случаях, когда одни и те же инструкции выполняются многократно при большом количестве вызовов. Это также актуально в сценариях, где требуется сохранять контекст, например данные API или результаты RAG, для последующего использования при нескольких обращениях. В следующем разделе мы подробнее рассмотрим метод раннего выхода, который позволяет сократить время вычисления вывода.

## Метод раннего выхода

Как описано в *главе 4*, архитектура трансформера состоит из многократно повторяющихся модулей, именуемых слоями.

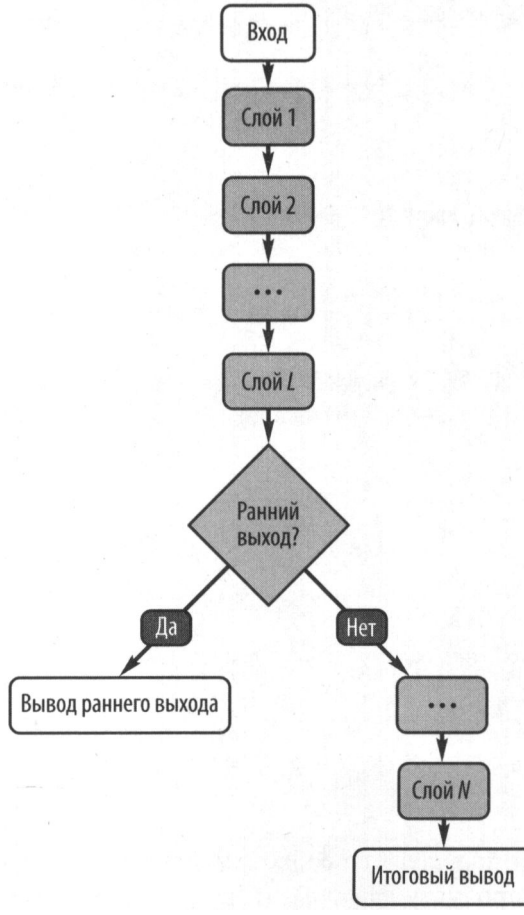



**Рис. 9.1.** Метод раннего выхода

Каждый слой генерирует промежуточное представление, которое затем используется в качестве входных данных для следующего слоя. Один из способов снизить вычислительные затраты во время процесса вывода заключается в остановке на промежуточном слое, где его результат интерпретируется как итоговый вывод. Этот подход известен как метод раннего выхода (early exit, EE). Пример применения этого метода представлен на рис. 9.1.

Метод раннего выхода можно осуществлять как на уровне всей последовательности, так и на уровне отдельных токенов.

### Ранний выход на уровне последовательности

В данном сценарии прямой проход в трансформере завершается на определенном уровне (слое) всей входной последовательности, и промежуточные представления, полученные на этом уровне, используются в качестве конечного результата. Слой, в который осуществляется выход, может быть задан заранее или определен динамически на основе входной последовательности.

Для динамического выбора выходного слоя можно обучить адаптеры поверх каждого слоя, как описано в *главе 7*. Затем эти модули можно использовать для прогнозирования возможности выхода в текущем слое. Например, в каждом слое модели FastBERT (**https://oreil.ly/GCfpt**) реализованы модули, которые обучаются решать задачу бинарной классификации (выходить или не выходить).

Однако не все подходы требуют добавления обучаемых модулей. В качестве альтернативы можно рассмотреть метод раннего выхода на основе хеша (hash-based early exiting, HashEE), предложенный в работе Сунь и соавт. (Sun et al., **https://oreil.ly/_JqqH**). Этот метод использует аннотированный набор последовательностей в сочетании с выходными слоями для определения оптимального выходного слоя для новых входных последовательностей. Основная идея заключается в предположении, что схожие последовательности должны осуществлять выход в одних и тех же слоях.

#### Упражнение

Выберите любую модель с открытым исходным кодом 7B и примените метод раннего выхода на 75-м процентиле слоев модели. Оцените эффективность данной техники на одном из наборов данных, доступных в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).

Проанализируйте влияние статического раннего выхода на производительность модели. Определите, насколько удалось сократить задержку с помощью данной стратегии.

Другой вариант метода раннего выхода — ранний выход на уровне токенов (token-level early exit).

### Ранний выход на уровне токенов

При использовании данного подхода можно осуществлять выход различных токенов из одной последовательности на разных уровнях. Это более сложная задача по сравнению с предыдущим методом.

Аналогично технике раннего выхода на уровне последовательности можно использовать двоичные классификаторы для принятия решения о выходе из указанного слоя. Однако в данном случае решение принимается не для всей последовательности, а отдельно для каждого токена на каждом уровне.

Дополнительные сведения о данном методе можно найти в работе Шустера и соавт. (Schuster et al., **https://oreil.ly/hfdCd)**, где представлена техника CALM (Confident Adaptive Language Modeling — уверенное адаптивное языковое моделирование), реализующая ранний выход на уровне токенов.

В подмодуле трансформера с механизмом внутреннего внимания представление каждого токена формируется на основе информации обо всех остальных токенах в последовательности внутри того же слоя. Однако при использовании метода раннего выхода на уровне токенов возникает ситуация, когда отдельные токены могут "выйти" на более раннем этапе. Чтобы корректно обработать такую ситуацию, представления вышедших токенов копируются из предыдущего слоя. Это является наиболее простым решением. Несмотря на то что метод раннего выхода на уровне токенов обладает большей детализацией и потенциально более высокой эффективностью по сравнению с ранним выходом на уровне всей последовательности, он уступает последнему по скорости работы.

#### Упражнение

С помощью метода CALM, реализованного Google (**https://oreil.ly/l5Xko**), примените метод раннего выхода на уровне токенов к любому набору данных, доступному в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).

Какие токены появляются на ранних слоях, а какие — на более поздних? Наблюдаются ли какие-либо лингвистические закономерности? Какие выводы можно сделать о процессе обучения модели?



При использовании метода раннего выхода наблюдается снижение общей эффективности, что связано с уменьшением производительности отдельных слоев. Тем не менее этот эффект можно минимизировать, обучив модель выходу на наиболее оптимальном слое.

Динамический ранний выход относится к категории методов, известных как динамический вывод (dynamic inference), где процесс вычисления результата определяется на основе параметров входных данных. Одним из значимых примеров такого подхода служат модели "смеси экспертов" (MoE), которые подробно рассматриваются в *главе 4*. В этих моделях механизм маршрутизации выбирает огра-

ниченное количество экспертных модулей для выполнения вывода, что позволяет сократить объем необходимых вычислений.

Далее мы рассмотрим способ сокращения времени вывода путем создания упрощенной производной модели на основе более сложной оригинальной, минимизируя при этом потерю качества. Для достижения этой цели используется метод дистилляции знаний.

## Дистилляция знаний

*В главе 5* мы кратко останавливались на дистиллированных версиях моделей, в частности, DistilBERT (**https://oreil.ly/rgiHZ**). Эти модели представляют собой более компактные решения, ориентированные на приближение к функциональности исходных крупных моделей, из которых они были созданы, обеспечивая более быструю генерацию выводов.

За последние годы было разработано несколько способов дистилляции знаний (knowledge distillation), обзор которых представлен в работе Сюй и соавт. (Xu et al., **https://oreil.ly/JZQf3**). Сам процесс дистилляции знаний состоит из двух ключевых этапов: подготовка дистиллированных данных (distillation data) и последующее обучение. В этом процессе основная модель называется моделью-учителем (teacher model), а полученная дистиллированная версия — моделью-учеником (student model).

Процесс дистилляции знаний модели показан на рис. 9.2.

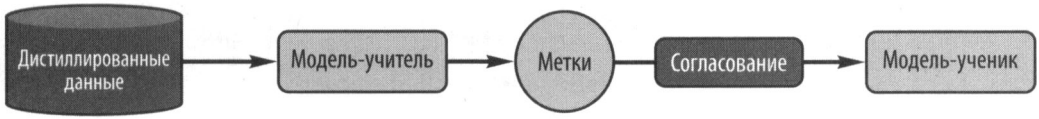



**Рис. 9.2.** Дистилляция знаний

Далее рассмотрим этапы подготовки дистиллированных данных и их обучение.

## Подготовка дистиллированных данных

Процесс подготовки дистиллированных данных включает формулирование запросов к модели-учителю с последующим использованием ее результатов в качестве источника знаний для дистилляции. К основным способам получения необходимой информации от модели-учителя относятся следующие.

*Неконтролируемая генерация (unsupervised generation)*

Этот метод подразумевает предоставление учителю инструкций и/или примеров, которые помогают в решении поставленной задачи. Ответы модели-учителя образуют набор дистиллированных данных. Неконтролируемая генерация часто применяется в процессе обучения меньших моделей сложным навыкам, например цепочке рассуждений (CoT) или следованию инструкциям. Для этого учителю предлагается формулировать ответы с использованием аргументированного подхода, который подробно объясняет процесс достижения итогового результата.

*Аугментация данных (data augmentation)*

Аугментация данных представляет собой технику, при которой модели-учителю предоставляется исходный набор примеров ввода-вывода. Используя этот набор, модель-учитель создает схожие пары ввода-вывода, формируя набор дистиллированных данных. Важно отметить, что оба элемента — и ввод, и вывод — генерируются самой моделью-учителем. Ограничением данной техники является неспособность модели-учителя создавать достаточно разнообразные примеры.

*Промежуточные представления (intermediate representations)*

Данный класс методов известен как дистилляция методом белого ящика. В рамках этого подхода набор дистиллированных данных формируется на основе промежуточных представлений модели, таких как активации или выходные логиты. Эти промежуточные данные используются для синхронизации модели-ученика с моделью-учителем. Обучение на согласование обычно осуществляется с помощью таких методов, как дивергенции Кульбака — Лейблера (KL-дивергенция), которая была подробно рассмотрена в *главе 4*.

*Обратная связь от учителя (teacher feedback)*

В рамках данного подхода выходные данные модели-ученика оцениваются моделью-учителем для предоставления обратной связи. Модель-учитель может предоставлять данные о предпочтениях, т. е. ранжировать по качеству результаты модели-ученика. Обратная связь также может быть представлена в виде подробных инструкций по улучшению выполнения задачи. Одним из популярных методов, использующих обратную связь от учителя, является обучение с подкреплением на основе обратной связи от ИИ (reinforcement learning from AI feedback, RLAIF), который был подробно рассмотрен в *главе 5*.

*Самообучение (self-teaching)*

Самообучение представляет собой подход, при котором модель выступает в роли учителя и ученика одновременно. В этом процессе модель-ученик постоянно улучшает свои выходные данные и использует их в дальнейшем для дистилляции. Одним из методов самообучения является создание нескольких вариантов решения для каждой задачи, включая шаги рассуждения, с последующим выбором наиболее удачного результата, который становится частью набора дистиллированных данных.

Сколько примеров требуется для дистилляции? Как ни удивительно, но их нужно совсем немного. Согласно исследованию Чжоу и соавт. (Zhou et al., **https://oreil.ly/MuOOj**), для создания эффективного набора дистилляции достаточно всего одной тысячи качественных примеров.



Аналогично дообучению и непрерывному предобучению, дистилляция знаний может привести к проблеме катастрофического забывания, которая была рассмотрена в *главе 7*.

Таким образом, мы рассмотрели различные способы формирования наборов дистиллированных данных (distillation dataset). Теперь подробно разберем сам процесс дистилляции.

## Процесс дистилляции

Далее приведены основные методы, применяемые для реализации задачи дистилляции. Более подробный обзор этих методов можно найти в работе Сюй и соавт. (Xu et al., **https://oreil.ly/9mbiN**).

*Контролируемое дообучение (supervised fine-tuning)*

Этот подход считается одним из самых простых способов выполнения дистилляции знаний. Его суть заключается в дообучении модели-ученика на специальном наборе дистиллированных данных. Основная цель — добиться согласованности прогнозов модели-ученика с результатами, получаемыми моделью-учителем. Чаще всего данный метод используется в рамках техники дистилляции знаний методом черного ящика, когда набор дистиллированных данных не содержит информации о внутренних представлениях моделей.

*K-L-дивергенция вероятностей выхода (K-L divergence of output probabilities)*

Данный подход направлен на минимизацию K-L-дивергенции между распределениями вероятных результатов модели-учителя и модели-ученика.

*Сходство внутреннего представления (internal representation similarity)*

Вместо минимизации K-L-дивергенции можно максимизировать сходство между аспектами моделей учителя и ученика. Это можно использовать для выполнения послойной дистилляции (layerwise distillation), в ходе которой внутренние представления учителя и ученика согласуются в каждом слое. С подробным описанием метода послойной дистилляции можно ознакомиться в работе Ляна и соавт. (Liang et al., **https://oreil.ly/g-C4L**).

*Обучение с подкреплением (reinforcement learning, RL)*

Данный подход предусматривает обучение модели вознаграждения на основе дистиллированных данных. Модель-ученик обучается оптимально увеличивать вознаграждение, ориентируясь на заданную модель вознаграждения. Тема обучения с подкреплением подробно обсуждалась в *главе 8*.

### Обобщение от слабого к сильному

Команда исследователей из OpenAI под руководством Бернса (Burns et al., **https://oreil.ly/xhrJL**) проследила феномен, который называется *обобщением от слабого к сильному* (weak-to-strong generalization). В этой системе модель-учитель является более компактной и менее мощной, чем модель-ученик.

Компактная модель-учитель дообучается на дистиллированных данных. Некоторая часть этих данных удерживается и используется для генерации выходных данных, известных как *слабые метки* (weak labels). Эти метки затем служат основой для дообучения гораздо более крупной и мощной модели-ученика.

Исследователи обнаружили, что более сильная модель-ученик способна эффективно обучаться на слабых метках, генерируемых более слабой моделью-учителем. Это вызвано тем, что модель-ученик опирается на собственные сильные представления, а дообучение на слабых метках лишь помогает ей обнаружить и уточнить уже имеющиеся знания. Такой подход значительно упрощает процесс обучения за счет использования для генерации данных более компактных моделей.

Выбор метода дистилляции модели в значительной степени определяется наличием доступа к весам модели-учителя. При отсутствии такого доступа возможен только вариант контролируемого дообучения. Использование дистилляции методом белого ящика, в котором основной упор делается на согласование промежуточных представлений модели, а не только на выходные токены, может стать достаточно сложной задачей. Важно учитывать, что все подходы к дистилляции знаний несут риск ухудшения производительности модели или возникновения эффекта катастрофического забывания. Поэтому необходимо тщательно проводить оценку модели-ученика, чтобы точно определить разницу в ее способностях относительно модели-учителя.

**Упражнение**

Используя процедуру дистилляции, преобразуйте модель Gemma 2B с открытым исходным кодом в более компактную модель, способную выполнять генерацию CoT. Какой из методов, описанных в данной главе, лучше всего подходит для выполнения этого задания?

В этом разделе мы разобрали три способа оптимизации вычислений на этапе вывода: кеширование, ранний выход и дистилляция знаний. Далее мы подробно остановимся на методах, которые позволяют существенно ускорить процесс декодирования.

## Методы ускорения декодирования

Авторегрессионные модели (autoregressive models), как известно, генерируют текст по одному токену за раз. Следующий токен определяется на основе всех входных, а также ранее созданных токенов, что приводит к последовательному ограничению: алгоритму приходится ждать окончания генерации текущего токена, прежде чем перейти к следующему. Возникает логичный вопрос: возможно ли преодолеть это ограничение? Для решения этой задачи были разработаны и активно применяются современные подходы, в числе которых *спекулятивное декодирование* (speculative decoding) и *параллельное декодирование* (parallel decoding).

Рассмотрим подробно каждый из этих методов.

## Спекулятивное декодирование

Спекулятивное декодирование представляет собой концепцию, основанную на интеграции двух моделей: меньшей черновой и большей основной. Сначала черновая модель (draft model) генерирует набор возможных токенов будущего вывода. Затем основная модель (main model) оценивает условные вероятности этих токенов одновременно, чтобы решить, какие из них следует оставить, а какие отклонить. Эффективность черновой модели возрастает с увеличением числа принятых подходящих токенов.

Процесс спекулятивного декодирования изображен на рис. 9.3.




**Рис. 9.3.** Пример работы механизма спекулятивного декодирования

Ключевыми метриками в этом подходе являются следующие.

*Частота принятия токенов (token acceptance rate)*

Эта метрика отражает процент токенов, предложенных черновой моделью, которые принимаются основной моделью. Как правило, данный параметр остается ниже 100%, поскольку, если бы он достигал максимума, то отпала бы необходимость в использовании большой модели.

*Ускорение декодирования (decoding speedup)*

Эта метрика демонстрирует уменьшение времени задержки при использовании только авторегрессивного декодирования по сравнению с применением спекулятивного метода.

### Построение черновых моделей

Как сделать так, чтобы черновая модель обладала высокой частотой принятия токенов? В работе Чжоу и соавт. (Zhou et al., **https://oreil.ly/scxWU**) предложен метод DistillSpec, который заключается в дистилляции черновой модели из главной.

Авторы также разработали метод самоспекулятивного декодирования (self-speculative decoding, **https://oreil.ly/KVZRB**), в котором черновая модель представляет подмножество слоев главной модели.

В ряде случаев текст, создаваемый LLM, содержит часто используемые фразы, префиксы и шаблоны. Кроме того, LLM может цитировать существующие тексты.

Вместо генерации таких стандартных фрагментов с помощью самой языковой модели их можно извлечь из внешних источников данных с помощью поисковой модели. Этот метод, описанный в работе Хэ и соавт. (He et al., **https://oreil.ly/DFufh**), называется спекулятивным декодированием на основе поиска (retrieval-based speculative decoding, REST).

## Параллельное декодирование

Можно ли добиться одновременной генерации нескольких токенов? Это возможно реализовать либо с помощью одной модели (многотокенное декодирование), либо используя несколько экземпляров одной и той же модели.

В рамках второго сценария можно настроить параллельную генерацию с использованием промпта. Например, при работе над статьей о туристической достопримечательности, которая будет включать такие разделы, как "питание", "пребывание", "советы по безопасности" и другие, можно запросить у модели составить список разделов и обозначить их специальными токенами. Затем эти разделы можно сгенерировать параллельно, поскольку они не зависят друг от друга.

Рабочий процесс системы, которая параллельно генерирует части выходных данных, показан на рис. 9.4.

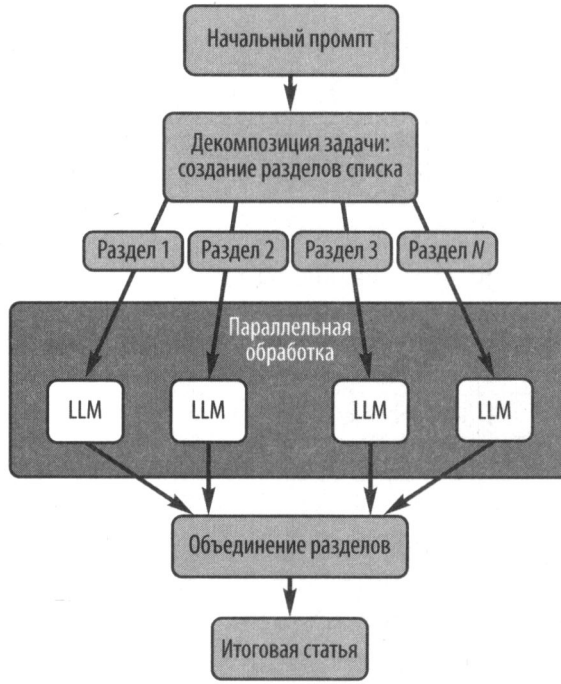



**Рис. 9.4.** Рабочий процесс параллельного декодирования

Рассмотрим, как одна и та же модель может генерировать несколько токенов одновременно. Этот процесс называется многотокенным декодированием (multi-token decoding).

В последние годы были предложены различные подходы к решению этой задачи. Одним из наиболее перспективных является метод Medusa, разработанный командой Цай (Cai et al., **https://oreil.ly/qT94i**). Этот метод предполагает добавление в модель дополнительных декодирующих головок, каждая из которых отвечает за предсказание следующих токенов.

Принцип работы Medusa заключается в следующем: основная декодирующая головка генерирует следующий (($n$ + 1)-й) токен последовательности, тогда как дополнительные головки прогнозируют ($n$ + 2)-й, ($n$ + 3)-й токены и далее. Такой подход позволяет существенно увеличить эффективность процесса генерации текста. Более детальное описание реализации метода доступно в официальной документации Medusa.

Таким образом, мы рассмотрели методы оптимизации процессов декодирования и снижения вычислительных затрат. Далее мы подробно рассмотрим класс методов квантования, направленных на уменьшение объема памяти, необходимого для функционирования модели.

## Методы снижения требований к хранению данных

*В главах 5* и *6* мы кратко рассмотрели концепцию квантования, но обещали более подробно остановиться на этом вопросе позже. Давайте углубимся в эту тему.

Прямой проход языковой модели включает в себя числовые значения, представляющие входные данные, веса и активации. Как эти числа представлены в памяти?

Для представления чисел существует несколько типов форматов, таких как целые числа (integer), числа с плавающей запятой (floating point) и др. Чаще всего языковые модели используют формат с плавающей запятой одинарной точности (FP32), который ассоциируется с числами, представленными 32-битной структурой или 4 байтами.

FP32 состоит из трех основных компонентов:

- знаковый бит;
- экспонента (8 бит);
- мантисса/значащая часть (23 бита).

Подробное описание принципов работы FP32 представлено в статье "Снимаем завесы с точности с плавающей запятой" ("Demystifying Floating Point Precision", **https://oreil.ly/uCYY1**).

Максимальное и минимальное значения, представляемые в формате FP32, составляют соответственно $3{,}4028237 \times 10^{38}$ и $1{,}175494 \times 10^{38}$. Этот диапазон определяет границы чисел, которые можно выразить данным форматом. Аналогично формат

FP16 (float16), или плавающая точка с половинной точностью, содержит три ключевых компонента:

- знаковый бит;
- экспоненту (5 бит);
- мантиссу, или значащую часть (10 бит).

Что произойдет, если взять число, представленное в формате FP32, и преобразовать его в FP16? Это можно сравнить с процессом потери данных. В нашем случае изменения затрагивают как диапазон, так и точность. В FP16 наибольшее число, которое можно представить, составляет 65 504, что значительно меньше, чем $3{,}4 \times 10^{38}$ для FP32. Кроме того, точность также снижается, поскольку 32-разрядная версия предлагает примерно 7 цифр точности, а 16-разрядная — только около 3.

Для предотвращения значительных потерь точности при использовании FP16, команда Google Brain разработала формат bfloat16 (BF16), также известный как "мозг с плавающей запятой" (brain floating point). В данном формате используются 8 бит для экспоненты и 7 бит для мантиссы. Это позволяет сохранить тот же диапазон чисел, что и у float32, но при этом жертвуется часть точности вычислений.



Графические процессоры предыдущих поколений (например, NVIDIA T4) не поддерживают формат BF16.

Операция преобразования числа из формата с повышенной точностью в формат с пониженной точностью называется квантованием (quantization). Например, 32-битные значения можно преобразовать в 8-битный целочисленный формат, что позволяет уменьшить объем требуемой памяти в четыре раза за счет небольшой потери точности. При использовании 8-битного квантования числа могут быть представлены в диапазоне от –127 до 127, при этом дробная часть отсутствует. Само квантование целого числа можно выполнять двумя способами: симметричным или асимметричным.

## Симметричное квантование

Симметричное квантование (symmetric quantization) предполагает, что значение 0 в исходном формате отображается (маппируется) на значение 0 в целочисленном представлении. Это означает, что при квантовании значения 0, представленного в формате FP32, в формат int8, значение остается равным 0.

Остальные значения могут быть отображены с использованием различных методов, среди которых наиболее распространенным является метод квантования по абсолютному максимуму (absmax). Суть метода заключается в следующем: если известен или может быть вычислен диапазон чисел, которые необходимо представить, то абсолютное максимальное значение диапазона может быть отображено на макси-

мальное число в формате int8 (127). В свою очередь, отрицательное абсолютное максимальное значение отображается на минимальное число int8 (–127). Все остальные значения отображаются в соответствии с масштабом.

На рис. 9.5 представлен пример использования квантования absmax для преобразования числа из формата FP32 в int8.




**Рис. 9.5.** Квантование absmax

## Асимметричное квантование

Асимметричное квантование (asymmetric quantization) предполагает, что значение 0 в исходных данных может не совпадать с нулевым значением в целочисленном формате. Один из распространенных подходов заключается в определении минимального и максимального значений, которые необходимо отобразить, и их привязке к крайним значениям диапазона целочисленного представления, т. е. –127 и 127, доступным в формате int8. Например, если диапазон чисел, подлежащих квантованию, находится в пределах от –23 до 87, то значение –23 будет представлено как –127, а значение 87 — как 127.



Если диапазон чисел, которые необходимо представить, содержит отклонения, это может привести к возникновению помех. Для решения этой проблемы можно исключить отклонения, представив все значения одним и тем же максимальным или минимальным числом.

### Упражнение

Рассмотрим влияние квантования на точность и диапазон числовых значений. Для этого возьмите несколько чисел: 2,3888888; 0; 34,444; $12{,}3486 \times 10^4$, –1223,4566. Выполните с ними арифметические операции, используя формат float32. Затем повторите эти операции, используя форматы float16, BF16 и int8. Определите, сколько точности теряется на каждом уровне квантования.

Как квантование применяется на практике? Обычно квантование выполняется после завершения обучения модели. Квантование может быть применено как к весам модели, так и к активациям.

Квантование весов считается более простым процессом по сравнению с квантованием активаций. Поскольку веса известны заранее, можно легко рассчитать параметры, необходимые для алгоритма квантования, такие как диапазон, отклонения и коэффициенты масштабирования.

В случае с активациями, в зависимости от допустимой задержки, можно использовать два подхода: динамическое или статическое масштабирование. При динамическом масштабировании (dynamic scaling) статистические данные, такие как диапазон и выбросы, вычисляются на каждом слое во время вывода. Статическое масштабирование (static scaling) предполагает оценку статистики с использованием калибровочного эталонного набора данных. Этот метод позволяет ускорить процесс вывода, но может привести к увеличению количества ошибок квантования.

Более подробную информацию о реализации квантования можно найти в "Визуальном руководстве по квантованию" Мартена Гротендорста (Grootendorst M. A Visual Guide to Quantization; **https://oreil.ly/bpi3b**).

**Упражнение**

Для каждого набора данных, представленного в репозитории книги на GitHub (**https://oreil.ly/llmplaybook**), запустите модель Llama 3.1 в режимах Float32, Float16, BF16 и int8.

Рассчитайте следующие параметры:

- влияние квантования на время вывода модели;
- влияние квантования на производительность модели;
- влияние квантования на требования к объему памяти.

Выберите набор данных и определите целесообразность его квантования.

# Резюме

В этой главе мы рассмотрели основные причины возникновения узких мест в процессе вывода языковых моделей. Мы разобрали различные подходы, направленные на повышение эффективности вывода, включая методы снижения вычислительных затрат, сокращения объемов хранения данных и ускорения процесса декодирования. Среди рассмотренных методов можно выделить кеширование, ранний выход, дистилляцию знаний, спекулятивное и параллельное декодирование, а также квантование. В заключительной части книги мы подробно остановимся на парадигмах приложений на базе LLM, а также обсудим особенности их применения и специфику разработки полнофункциональных приложений.

# Часть III

# Парадигмы приложений на основе LLM

В данной части книги основное внимание уделяется практической стороне вопроса. Ранее мы рассматривали большие языковые модели как отдельные концепции. Теперь же мы сосредоточимся на их интеграции в масштабные программные системы. Для этого мы рассмотрим наиболее популярные подходы к разработке приложений, такие как генерация с дополненным поиском (retrieval-augmented generation, RAG) и агенты.

# ГЛАВА 10

# Взаимодействие LLM с внешними инструментами

В предыдущих главах книги было показано, как автономные LLM справляются с широким спектром задач, демонстрируя высокую эффективность. Для полноценного внедрения их потенциала в рамках организации требуется интеграция с существующей инфраструктурой данных и приложений. В отличие от привычных программных решений, LLM способны самостоятельно инициировать действия для взаимодействия с различными компонентами экосистемы, обеспечивая уровень адаптивности и гибкости, который ранее считался недостижимым. Этот подход открывает перспективы для внедрения инновационных сценариев применения, ранее считавшихся практически невозможными.

Еще одной из причин, по которой необходимо организовать взаимодействие больших языковых моделей с программным обеспечением и внешними данными, является их текущая ограниченность, на которую мы неоднократно обращали внимание. Рассмотрим основные аспекты этого вопроса.

- Переобучение или регулярное обновление языковых моделей является затратным процессом, что ограничивает актуальность знаний LLM. В результате они остаются неинформированными о событиях, произошедших после их последнего обновления.
- Несмотря на прогресс в развитии, большие языковые модели все еще допускают ошибки в математических вычислениях.
- LLM неспособны гарантировать достоверность предоставляемой информации или надежно ссылаться на источники своих выводов.
- Представление модели собственных данных пользователя остается сложной задачей: дообучение требует значительных усилий, а возможности контекстного обучения ограничиваются размером доступного контекстного окна.

По ходу книги мы неоднократно убеждались в том, что консолидация может стать основой для перспективного будущего, в котором многие существующие ограничения могут быть преодолены на уровне модели при условии дальнейшего технологического развития. Однако не обязательно ждать наступления этого момента —

уже сейчас существует возможность решения множества проблем с помощью внешних инструментов.

В этой главе мы рассмотрим три основные парадигмы взаимодействия больших языковых моделей и предложим рекомендации по их применению в зависимости от специфики проекта. LLM могут взаимодействовать с двумя типами внешних объектов: хранилищами данных и программным обеспечением, которое принято называть инструментами (tools). Мы рассмотрим, как интегрировать LLM с такими элементами, как API и кодовые интерпретаторы. В частности, обсудим эффективное использование таких популярных библиотек, как LangChain и LlamaIndex, которые значительно упрощают процесс взаимодействия. Кроме того, мы рассмотрим организацию программного обеспечения для скаффолдинга (scaffolding — строительные леса, каркас) — основы беспрепятственного взаимодействия моделей с окружающей цифровой средой. Также мы узнаем, как расширить функциональные возможности современных LLM, демонстрируя их применение в роли автономного агента, способного самостоятельно принимать решения.

## Парадигмы взаимодействия LLM

Предположим, что перед нами стоит задача, которую необходимо решить с использованием LLM. Варианты решения могут включать в себя:

- использование собственных знаний и возможностей, закодированных в параметрах модели, без дополнительного вмешательства;
- предоставление промпта, содержащего полный контекст задачи, чтобы на основе предоставленной информации модель могла применить свои навыки для решения;
- ситуацию, когда у LLM нет достаточных данных или навыков. В этом случае необходимо обновить параметры модели, например через дообучение, как описано подробно в *главах 6–8*, чтобы активировать необходимые знания и навыки;
- если заранее неизвестно, какой контекст требуется для выполнения задачи, то можно использовать механизмы автоматического выбора подходящего контекста и его вставки в промпт (метод пассивного взаимодействия, passive approach);
- в модель поступают четкие указания о том, как взаимодействовать с внешними инструментами и хранилищами данных для решения конкретной задачи. Модель следует этим инструкциям, что составляет суть метода явного взаимодействия (explicit approach);
- при необходимости LLM разбивает задачу на несколько подзадач, взаимодействует со своей средой для сбора информации и знаний, необходимых для решения, и делегирует подзадачи внешним моделям и инструментам, когда у нее нет достаточных возможностей для их самостоятельного выполнения. Такой подход называется методом автономной работы (autonomous approach).

Таким образом, последние три подхода предполагают участие модели во взаимодействии с внешними системами. Далее рассмотрим эти парадигмы взаимодействия подробнее.

## Метод пассивного взаимодействия

На рис. 10.1 показан типичный рабочий процесс приложения, в котором LLM пассивно взаимодействует с хранилищем данных.




**Рис. 10.1.** Пассивное взаимодействие LLM с хранилищем данных

Многие сценарии применения предполагают использование LLM для работы с собственными данными. Например, это может быть разработка ассистента, способного отвечать на вопросы, опираясь на внутреннюю базу знаний организации, охватывающую множество документов в Notion, или создание чат-бота для авиакомпании, который предоставляет клиентам информацию о статусе рейсов и правилах бронирования.

Для предоставления LLM доступа к внешней информации необходимы два ключевых элемента: "хранилища данных", содержащие релевантные сведения, и механизмы извлечения, позволяющие получать нужную информацию из этих хранилищ в соответствии с заданным запросом. Механизм извлечения может быть встроен в саму модель или реализован в виде простого алгоритма, например сопоставления ключевых слов. Хранилища данных могут представлять собой базы данных, графы знаний, векторные базы или даже обычные текстовые файлы. Для повышения эффективности извлечения данные в хранилище проходят этапы представления и индексации. Детальный обзор процессов представления данных, индексации и извлечения заслуживает отдельного внимания и будет рассмотрен в *главе 11*.

При отправке запроса пользователем активируется механизм извлечения документов или текстовых фрагментов, которые наиболее релевантны для ответа на запрос. После проверки их соответствия контекстному окну LLM эти материалы вместе с запросом передаются в LLM. Ожидается, что LLM предоставит ответ, который будет учитывать заданный контекст, описанный в промпте. Данный подход известен как генерация с дополненным поиском (RAG). Однако, как будет показано в *главе 12*, термин RAG охватывает более широкое понятие. Эта парадигма играет важную роль и заслуживает детального рассмотрения, поэтому мы подробно остановимся на ней в *главе 12*.

Особенностью данной парадигмы является пассивное участие LLM в процессе взаимодействия. Модель лишь реагирует на запрос, предоставляя ответ, при этом оставаясь неосведомленной о происхождении содержащегося в запросе контента. Такой подход часто применяется при создании QA-ассистентов или чат-ботов, где для понимания общего контекста беседы требуется внешняя информация.



В дальнейшем мы будем использовать термины "запросы" и "документы" для обозначения, соответственно, *запросов* пользователей к LLM и текстовых элементов, извлеченных из *внешних источников*. Документы могут представлять собой целые тексты, их отдельные фрагменты, абзацы или предложения.

## Метод явного взаимодействия

На рис. 10.2 схематично изображен метод явного взаимодействия при интеграции LLM с внешними инструментами.

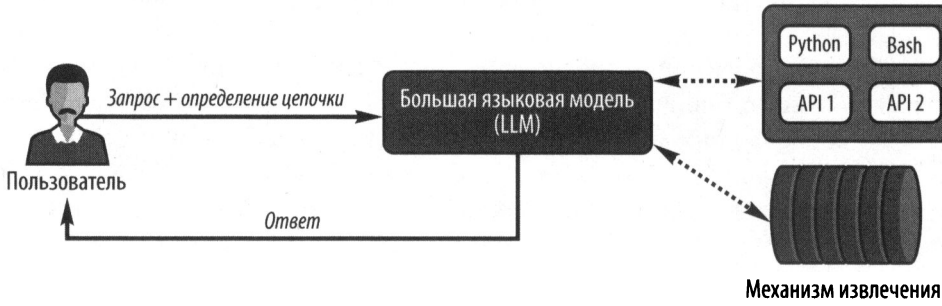



**Рис. 10.2.** Метод явного взаимодействия

В отличие от пассивного взаимодействия, в данном случае LLM активно участвует в процессе. Модель получает четкие инструкции о том, как и в каких случаях использовать внешние хранилища данных или инструменты. Взаимодействие LLM с внешней средой происходит на основе заранее определенных условий. Этот метод особенно актуален, когда последовательность действий строго фиксирована, ограничена по объему и предполагает минимальное количество шагов.

Пример последовательности взаимодействия ассистента — аналитика данных на основе ИИ может выглядеть следующим образом:

1. Пользователь формулирует запрос на естественном языке, например, с просьбой визуализировать определенные тренды данных.
2. LLM генерирует SQL-запрос (structured query language — язык структурированных запросов), который позволит извлечь необходимую информацию из базы данных для выполнения поставленной задачи.
3. После получения извлеченных данных модель производит генерацию программного кода, который может быть запущен интерпретатором кода для расчета статистических показателей или визуализации.

На рис. 10.3 представлена фиксированная последовательность взаимодействия, реализованная для аналитика данных ИИ.

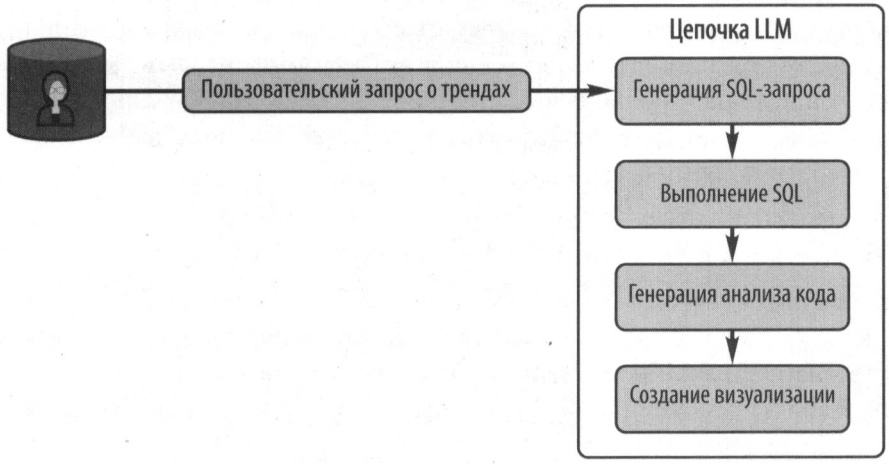



**Рис. 10.3.** Пример рабочего процесса для аналитика данных ИИ

В данной парадигме порядок взаимодействия заранее определен и базируется на установленных правилах. Для выбора следующего шага LLM лишена самостоятельности в принятии решений. Этот подход считается оптимальным для разработки устойчивых приложений с повышенными требованиями к надежности.

## Метод автономной работы

На рис. 10.4 представлен процесс преобразования LLM в автономного агента, способного самостоятельно выполнять сложные задачи.

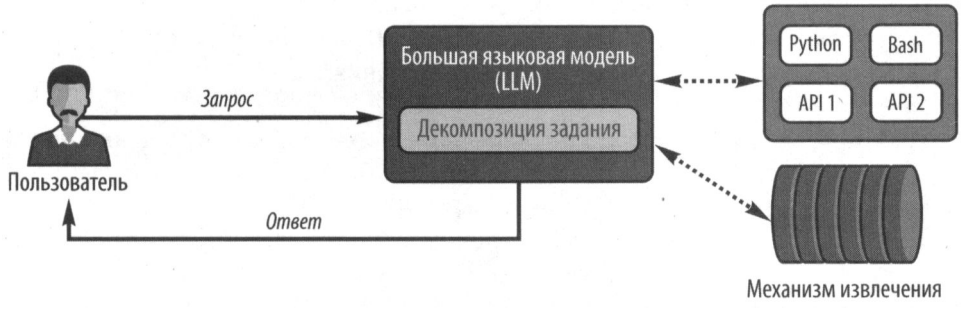



**Рис. 10.4.** Рабочий процесс автономного агента, управляемый LLM

Метод автономной работы, который я предпочитаю называть подходом "Святого Грааля", предполагает превращение LLM в самостоятельного агента, способного решать задачи без участия человека и взаимодействовать с окружающей средой. Ниже представлен типичный рабочий цикл такого агента:

1. Пользователь формулирует запрос на естественном языке, при необходимости уточняя формат, в котором должен быть предоставлен ответ.

2. LLM анализирует запрос пользователя и разбивает его на управляемые подзадачи.
3. Каждая подзадача решается моделью с использованием синхронного или асинхронного подхода. Алгоритм использует собственную память и знания для выполнения конкретной подзадачи. Если агент не может решить задачу самостоятельно, он выбирает подходящий инструмент из доступного списка. Результаты выполненных подзадач используются в качестве входных данных для последующих этапов работы.
4. После выполнения всех этапов алгоритма полученные решения объединяются в итоговый ответ, структурированный в запрошенном формате.

Данная парадигма является достаточно универсальной и может быть использована в различных сценариях. Однако она также связана с определенными рисками, поскольку возлагает на LLM слишком большую ответственность и полномочия. На текущем этапе не рекомендуется использовать эту концепцию для создания критически важных приложений.



В связи с этим необходимо подчеркнуть, что при внедрении агентов следует проявлять особую осторожность. Многие пользователи склонны недооценивать требования к точности приложений. В некоторых случаях даже 99%-ная точность может оказаться недостаточной, особенно если сбои в работе непредсказуемы и могут привести к серьезным последствиям. Эта проблема долгое время сдерживала развитие беспилотных автомобилей и ограничивала их массовое применение. Однако это не означает, что мы не можем использовать автономных агентов LLM. Ключевым требованием является грамотный подход к созданию продукта, который должен минимизировать воздействие ошибок на пользователя. Также необходимо внедрение эффективных концепций взаимодействия человека с системой, так называемой схемы "человек в цикле" (human-in-the-loop) для повышения общей надежности и безопасности решения.

В рамках книги мы неоднократно обращались к понятию "агент", однако не давали ему точного определения. Пришло время устранить этот пробел и разобраться, что представляет собой агент и как его можно создать.

## Определение агента

В связи с ростом интереса к агентам, созданным на основе больших языковых моделей, понятие "агент" стало использоваться в более широком контексте. Это связано с тем, что разработка автономных агентских систем является сложной задачей, и многие специалисты склонны применять термин "агент" к системам, которые демонстрируют высокую эффективность, даже если они не соответствуют всем требованиям.

В книге мы будем придерживаться более строгого подхода и определим агентов следующим образом:

> "Программные системы, созданные на основе больших языковых моделей, которые могут взаимодействовать с окружающей средой и самостоятельно выполнять действия, направленные на достижение конкретной цели".

Ключевыми характеристиками агентов являются следующие.

*Автономность*

Агенты способны самостоятельно определять последовательность действий, необходимых для выполнения задачи, без указания со стороны человека.

*Способность взаимодействовать со средой*

Агенты могут подключаться к внешним источникам данных и программным инструментам для получения нужной информации, вызова инструментов, выполнения кода и предоставления инструкций, необходимых для решения задачи.



В ряде определений термина "агент" отсутствуют требования о полной автономности. В соответствии с этими определениями, приложения, функционирующие в рамках явной парадигмы, могут быть признаны агентами, но при этом они будут считаться лишь полуавтономными, а не полностью самостоятельными.

Агентская парадигма в том виде, в котором мы ее определили, обладает исключительной эффективностью и универсальностью. Рассмотрим ее подробнее. Когда агент сталкивается с задачей, решение которой требует знаний, он осознает, что не обладает ими, он может самостоятельно найти способ справиться с этой задачей. Эти действия могут включать в себя поиск информации в сети или базах данных (БД), сбор необходимых данных, а также дообучение модели, которая может предложить решение.

Учитывая впечатляющие возможности современных технологий, возникает вопрос: возможно ли, что в ближайшем будущем машины смогут захватить мир? На самом деле существующие автономные агенты имеют ряд ограничений. Они часто сталкиваются с проблемами, такими как застревание в циклах, совершение ошибок и невозможность надежной корректировки своих действий. Более рациональным подходом является разработка частично автономных агентов, которым предоставляется руководство на протяжении всего рабочего процесса. Это может быть реализовано с помощью специального программного обеспечения для оркестрации агентов или в тесном взаимодействии с человеком. В оставшейся части главы мы рассмотрим более подробно разработку надежных агентов, способных успешно решать ограниченный класс задач.

## Рабочий процесс агентской системы

В рамках данного определения агентов рассмотрим их работу на практике. В качестве примера возьмем агента, перед которым поставлена задача ответить на вопрос:

> Кто занимал должность финансового директора компании Apple в период, когда рыночная стоимость ее акций была минимальной за последнее десятилетие?

Представим ситуацию, в которой агент располагает всей необходимой информацией для выполнения поставленной задачи. У него есть доступ к Интернету, базам данных SQL с информацией о стоимости акций, хранилищам знаний, содержащим сведения о роли финансового директора. Также агент подключен к интерпретатору кода, что позволяет ему генерировать и запускать программные решения. Кроме того, агент имеет доступ к финансовым API для работы с внешними системами. Системный промпт содержит описание всех доступных LLM инструментов и источников данных.

Для выполнения запроса система должна выполнить следующие шаги:

1. Определяется текущая дата. Если она отсутствует в исходном запросе, модель либо извлекает ее из Сети, либо генерирует код для получения системного времени с помощью встроенных функций и передает его на выполнение интерпретатору.
2. На основе текущей даты вычисляется другая граница диапазона, что требует выполнения арифметической операции (например, вычитания лет) или генерации соответствующего кода. Эти задачи могут быть объединены в одну компактную программу.
3. Модель ищет в доступных базах данных таблицу, содержащую информацию о стоимости акций компании. Затем она извлекает схему этой таблицы и формирует SQL-запрос для поиска даты, на которую стоимость акции компании достигла минимального значения за последние 10 лет.
4. После получения даты минимальной стоимости акций осуществляется поиск упоминаний финансового директора Apple на эту дату. Модель может использовать доступный API поисковой системы для выполнения запроса и проверки наличия упоминания финансового директора на эту дату.
5. В случае если поисковый запрос не дает точных результатов, модель обращается к финансовому API, используя доступные инструменты. Затем она извлекает документацию этого API и использует ее для написания кода запроса. В результате модель получает список финансовых директоров компании Apple и информацию об их сроках пребывания в должности.
6. Модель применяет свои навыки арифметического рассуждения для определения срока пребывания финансового директора, соответствующего дате минимальной стоимости акций.

7. Окончательный ответ формируется в соответствии с запрошенным форматом. Если запрос содержит особые требования к структуре вывода, модель адаптирует результат под эти требования.

Последовательность действий может варьироваться в зависимости от реализации системы. Например, модель можно дообучить таким образом, чтобы она генерировала код для работы с API или написания SQL-запросов напрямую, без необходимости извлечения схемы из базы данных или интерфейса API.

Для выполнения указанной последовательности задач модель должна сначала определить, что задача должна быть разделена на ряд подзадач — процесс, известный как декомпозиция (decomposition). Декомпозицию и планирование можно осуществить с помощью самой модели или использовать для этих задач внешние инструменты.

**Упражнение**

Оцените, насколько инструменты на базе LLM, такие как ChatGPT, Perplexity и Gemini, способны решить задачу, связанную с поиском информации о финансовом директоре компании Apple, учитывая их доступ к Сети. Могут ли эти инструменты корректно решить поставленную задачу? Если нет, то в чем могут быть причины их недостаточной эффективности? Обратите внимание, что чем раньше в процессе решения задачи происходит сбой LLM, тем сложнее будет его устранить. Для сравнения, попробуйте решить эту задачу, используя только веб-поиск без применения LLM. Вы увидите, что даже эксперту в финансовой области может потребоваться несколько поисковых запросов, чтобы найти ответ на этот вопрос.

## Компоненты агентской системы

Несмотря на то что архитектура конкретной агентской системы в значительной степени определяется спецификой сценариев ее использования, которым она предназначена соответствовать, все ее компоненты можно разделить на несколько основных категорий:

- модели;
- инструменты;
- хранилища данных;
- промпты для выполнения цикла агента (agent loop);
- защитные ограждения (guardrails) и верификаторы (verifiers);
- программное обеспечение для оркестрации.

Типичная структура агентской системы и взаимодействие ее компонентов представлены на рис. 10.5.




**Рис. 10.5.** Агентская система промышленного уровня

Рассмотрим подробно каждую категорию.

## Модели

Языковые модели являются основой агентских систем, которые отвечают за их автономность и способность решать задачи. Единая агентская система может включать в себя несколько языковых моделей, каждая из которых выполняет свою функцию.

Например, можно создать агента, состоящего из двух моделей. Одна модель будет решать пользовательские задачи, а другая — преобразовывать полученные данные в структурированный формат в соответствии с требованиями пользователя.



В процессе работы агентских систем может потребоваться значительное количество токенов языковой модели, что может вызвать дополнительные затраты. Для оптимизации расходов рекомендуется использовать несколько языковых моделей различного размера, при этом менее мощные модели могут быть задействованы для выполнения менее сложных задач. Более подробную информацию о распределении задач между моделями *см. в главе 13*.

В более общем случае можно создавать агентов на основе специализированных моделей, которые подходят для каждой части потока операций агентской системы. Например, для создания кода можно использовать LLM кода, а для конкретных задач — дообученные модели, специализирующиеся на отдельных шагах рабочего

процесса. Эту концепцию можно рассматривать как *многоагентную архитектуру* (multi-agent architecture).

На рис. 10.6 представлена агентская система, состоящая из нескольких LLM.

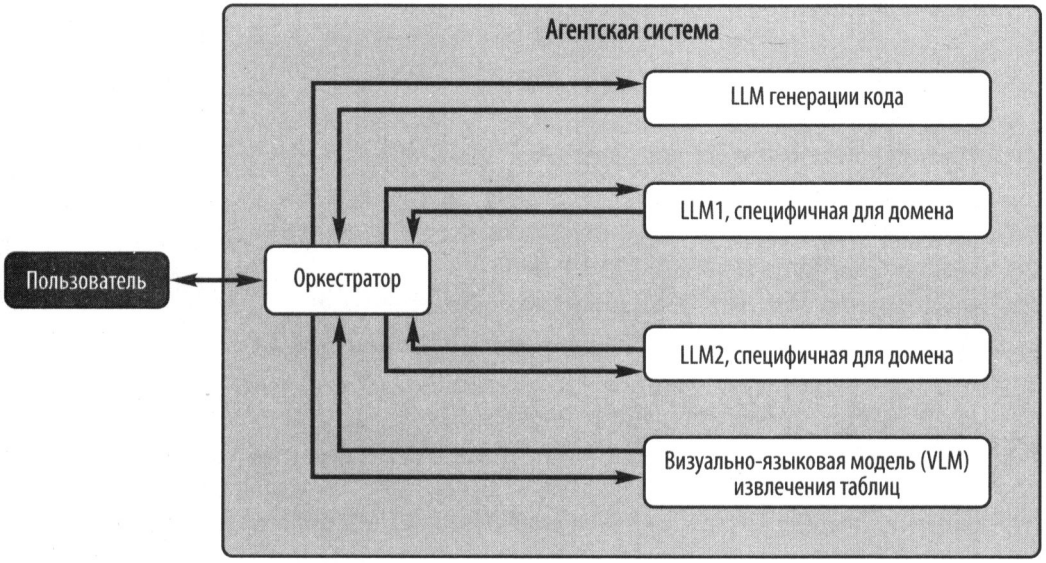



**Рис. 10.6.** Агентская система с несколькими LLM

Любой тип модели, включая те, которые не относятся к LLM, можно интегрировать в агентскую систему для выполнения узкоспециализированных задач. Так, например, этап планирования можно реализовать с помощью символьных планировщиков (**https://oreil.ly/sXPWG**).

## Инструменты

Как было отмечено ранее, программное обеспечение или модели, которые могут быть использованы для запуска большой языковой моделью, называются инструментами. Библиотеки, такие как LangChain (**https://oreil.ly/35Lgu**) и LlamaIndex (**https://oreil.ly/WF-d1**), предоставляют коннекторы для различных программных интерфейсов, включая интерпретаторы кода, поисковые системы, базы данных, модели машинного обучения и различные API. Рассмотрим, как можно использовать некоторые из них на практике.

### Веб-поиск

LangChain предлагает интеграцию с ведущими поисковыми системами, такими как Google, Bing и DuckDuckGo. Рассмотрим пример с использованием DuckDuckGo:

```
from langchain_community.tools import DuckDuckGoSearchRun


query = "Какая погода сегодня в Торонто?"
```

```
search_engine = DuckDuckGoSearchRun()
output = search_engine.run(query)
```

Полученный результат может быть передан в языковую модель для дальнейшей обработки и анализа.

### API-коннекторы

Для демонстрации использования API рассмотрим обертку для Wikipedia API, предоставляемую библиотекой LangChain:

```
!pip install wikipedia

from langchain.tools import WikipediaQueryRun
from langchain_community.utilities import WikipediaAPIWrapper
wikipedia = WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper())

output = wikipedia.load("Зимние Олимпийские игры")
```

Функция `load()` запускает поиск в Википедии и возвращает текст страницы и метаданные результатов top-$k$ (по умолчанию top-$k$ = 3). Также можно использовать функцию `run()`, чтобы возвращать только сводки по страницам совпадений top-$k$.

### Интерпретатор кода

Рассмотрим процесс активации интерпретатора кода с последующим запуском произвольного кода:

```
from langchain_experimental.utilities import PythonREPL

python = PythonREPL()
python.run("456 * 345")
```



Важно помнить об осторожности при выполнении кода, сгенерированного LLM в ответ на пользовательские запросы. Пользователи могут спровоцировать модель на генерацию вредоносного кода.

### Коннекторы БД

В заключение рассмотрим процесс подключения к базам данных и выполнения запросов:

```
import sqlalchemy as sa
from langchain_community.utilities import SQLDatabase

DATABASE_URI = <database_uri>

db = SQLDatabase.from_uri(DATABASE_URI)
```

```
output = db.run(
    "SELECT * FROM COMPANIES WHERE Name LIKE :comp;",
    parameters={"comp": "Apple%"},
    fetch="all")
```

Функция `run()` выполняет предоставленный SQL-запрос и возвращает ответ в виде строки. Можно заменить `DATABASE_URI` собственной базой данных и запросами и проверить ответы.



Для повышения гибкости системы можно создать дополнительные ответвления коннекторов LangChain и использовать их в своих целях.

Теперь рассмотрим, как можно интегрировать LLM с указанными инструментами в рабочем процессе агентской системы.

Прежде всего, необходимо предоставить LLM информацию о доступе к инструментам. Один из способов сделать это — предоставить LLM *список инструментов* с их названием и кратким описанием через системный промпт.

Затем LLM должна иметь возможность выбирать соответствующий инструмент на каждом этапе рабочего процесса. Например, если следующим шагом в решении задачи является поиск прогноза погоды в Чикаго на сегодняшний вечер, необходимо использовать инструмент веб-поиска, а не Википедию. В данной главе мы рассмотрим методы, которые помогут LLM выбирать подходящие инструменты.

Под капотом процесс вызова инструмента обычно реализуется через генерацию специального набора токенов LLM, которые сигнализируют о переходе модели в режим использования инструмента. Эти токены также содержат информацию о функциях и аргументах инструмента, подлежащих вызову. Реализация фактического вызова инструмента осуществляется при помощи фреймворка оркестрации агента.

В LangChain можно сделать инструмент доступным для LLM и вызвать его с помощью следующих команд:

```
from langchain.agents import initialize_agent, Tool
from langchain.agents import AgentType
from langchain_community.tools import DuckDuckGoSearchRun
from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage

search_engine = DuckDuckGoSearchRun()
model = ChatOpenAI(model="gpt-4o")

tools = [
        Tool(
```

```
            name="Search",
            func=search_engine.run,
            description="search engine for answer factual queries"
        )
    ]
agent = initialize_agent(tools, model, verbose=True)
agent.run("Какие интересные туристические места можно посетить в Северной Германии?")
```

**Упражнение**

С помощью LangChain создайте агента, который будет инициализирован как минимум пятью инструментами из списка, доступного на сайте **https://oreil.ly/LumHc**. К инструментам, не требующим ключа API, относятся Wikipedia, DuckDuckGo и arXiv, последний является хранилищем научных работ, в том числе о LLM. Обратите внимание, выбирает ли модель правильный инструмент для заданного пользовательского запроса. Предоставьте модели подсказки по выбору правильного инструмента в системном промпте. Наблюдаются ли после этого какие-либо улучшения?

Некоторые модели имеют встроенные возможности вызова инструментов. Для моделей, не обладающих такими функциями, можно дообучить базовую модель. Примером модели, имеющей встроенную поддержку вызова инструментов, является Llama 3.1 Instruct (8B/70B/405B).

Llama 3.1 поставляется с нативной поддержкой трех инструментов: веб-поиска Brave, математического движка Wolfram | Alpha и интерпретатора кода. Для их активации достаточно определить соответствующие параметры в системном промпте:

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>
Environment: ipython
Tools: brave_search, wolfram_alpha

Кратко ответить на вопросы. <|eot_id|>
```

Зададим LLM вопрос, добавив пользовательский промпт в системный:

```
<|start_header_id|>user<|end_header_id|>

Сколько медалей завоевал Азербайджан на летних Олимпийских играх 2024 года?
<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Llama 3.1 отвечает с использованием инструмента, который формирует запрос следующего вида:

```
<|python_tag|>brave_search.call(query=" Сколько медалей завоевал Азербайджан
на летних Олимпийских играх 2024 года?")<|eom_id|>
```

Токен <|python_tag|> представляет собой специальный токен, генерируемый Llama 3.1 для обозначения перехода в режим использования инструментов. Токен <|eom_id|> указывает, что модель пока не завершила выполнение своей задачи и ожидает получения результатов работы инструмента. В промпте также можно задать пользовательские инструменты в формате JSON.



В случае большого количества инструментов подробные описания могут быть сохранены в хранилище данных и доступны для извлечения только при выборе конкретного инструмента. В этом случае запрос должен содержать только название инструмента и его краткое описание.

Рассмотрим пример определения инструмента в формате JSON, который описывает локальную функцию, доступную для вызова:

```
<|start_header_id|>user<|end_header_id|>


Ниже представлен список доступных инструментов.

При вызове инструмента необходимо ответить в формате JSON. Формат выглядит следующим
образом:


{"tool_name": имя инструмента, "arguments": словарь с ключами, представляющими
имена аргументов, и значениями, представляющими значения аргументов}.


{

    "type": "local_function",
    "function": {
    "name": "find_citations",
    "description": "Найдите цитаты для любых сделанных утверждений"
    "parameters": {
        "type": "object",
        "properties": {
        "claim_sentence": {
            "type": "string",
            "description": "Предложение во входных данных, представляющее требование"
        },
        "model": {
            "type": "string",
            "enum": ["weak", "strong"],
            "description": "Тип используемой модели цитирования. Если искомое
предложение содержит сущности и цифры, рекомендуется использовать слабую модель"
```

```
        }
        },
        "required": ["claim_sentence", "model"]
    }
    }
}
```

Модель генерирует вызов инструмента в формате JSON, соответствующем заданному формату.



Фактический вызов инструмента осуществляется с помощью программного обеспечения для оркестрации агентов. Llama 3.1 поставляется с библиотекой llama-stack-apps (**https://oreil.ly/SSmkI**), которая упрощает рабочие процессы агентских систем.

Иногда генерация запроса к инструменту может оказаться более сложной задачей, чем просто указание имени функции и ее аргументов. Примером такого запроса может быть запрос к базе данных. Для того чтобы LLM сгенерировала корректный SQL-запрос, необходимо предоставить в системном промпте схему таблиц БД. В случае если в базе данных присутствует большое количество таблиц, LLM может извлечь их схему по запросу.



Для генерации кода и SQL-запросов можно использовать специализированную модель. Универсальная модель общего назначения генерирует текстовое описание требуемого результата, которое впоследствии может быть использовано в качестве входных данных для специализированной LLM, предназначенной для генерации кода, или для модели, дообученной преобразовывать текст в SQL-запросы (text-to-SQL).

Для крупных приложений или решений, требующих высокой степени ответственности, дообучение моделей позволяет значительно повысить удобство их интеграции в используемые инструменты. Одной из эффективных методик дообучения является подход ToolLLaMA, предложенный в работе Цинь и соавт. (Qin et al., **https://oreil.ly/Ewlxt**).

### Упражнение

Создайте пользовательские инструменты, доступные для Llama 3, и сохраните их определения в формате JSON в системном промпте. Инструменты должны представлять собой функции, которые могут выступать в роли оболочки для существующего программного обеспечения или выполнять определенные задачи, например конвертацию градусов Цельсия в градусы по Фаренгейту. Создайте пять следующих инструментов:

- инструмент для запроса Wikipedia API;

- инструмент для запроса arXiv API;
- инструмент конвертации градусов Цельсия в градусы по Фаренгейту;
- инструмент, сохраняющий входные данные в текстовый файл;
- инструмент, создающий копию файла.

Попробуйте сформулировать запросы, которые могли бы вызвать эти инструменты. Проверьте, насколько корректно они работают. Если нет, отредактируйте описания инструментов и посмотрите, поможет ли это исправить проблему.

## Хранилища данных

Для выполнения своих функций агенту зачастую требуется работа с различными типами источников информации. Среди наиболее распространенных источников данных можно выделить репозитории промптов (prompt repositories), сессионную память или сессионное хранилище (session memory) и данные инструментов (tools data).

### Репозиторий промптов

Репозиторий промптов (prompt repository) представляет собой набор детализированных инструкций, направляющих языковую модель на выполнение конкретных задач. Если заранее определить типы задач, которые агенту предстоит решать в процессе работы, можно создать промпты с подробными инструкциями по их выполнению. Такие промпты могут также содержать указания по улучшению конкретного рабочего процесса. Например, известно, что многие языковые модели испытывают сложности с выполнением элементарных арифметических операций. Даже простые вопросы могут стать для них проблемой, как, например:

```
Является ли 9.11 больше, чем 9.9?
```

До недавнего времени даже самые продвинутые языковые модели утверждали, что 9,11 больше, чем 9,9. Недавно это было исправлено, после того как данное ограничение стало широко обсуждаться в социальных сетях (**https://oreil.ly/ztWGW**).

Если известно о подобных ограничениях, связанных с конкретным сценарием использования, их частично можно обойти с помощью более продуманных и детализированных промптов. Например, для корректного сравнения чисел можно использовать следующие запросы:

> *Prompt*: If you are asked to compare two numbers using the greater than/lesser than operation, then perform the following:
>
> Take the two numbers and ensure they have the same number of decimal places. After that, subtract one from the other. If the result is a positive number, then the first number is greater. If the result is a negative number, then the second number is greater. If the result is zero, the two numbers are equal.

*Промпт*: если предлагается сравнить два числа с помощью операции "больше/меньше", то необходимо выполнить следующие действия:

взять два числа и убедиться, что они оба имеют одинаковое количество десятичных знаков. Вычислить разность между этими числами. Если результат положительный, то первое число больше второго. Если результат отрицательный, то второе число больше первого. Если результат равен нулю, то числа равны.

В случае необходимости выполнения задачи, связанной со сравнением числовых данных, агент получает соответствующий промпт из репозитория. Это позволяет ему следовать подробным пошаговым инструкциям, а значит, иметь возможность преодолеть ограничения, связанные с выполнением задачи.



Почему бы нам не интегрировать все эти промпты непосредственно в контекстное окно, чтобы избежать необходимости их извлечения? Во-первых, количество промптов может оказаться слишком большим, и они не поместятся в контекстном окне. Во-вторых, токены являются дорогостоящими, и нецелесообразно включать промпты, которые могут не иметь отношения к текущей задаче. Наконец, языковые модели способны обрабатывать только ограниченный набор параллельных команд, поэтому более эффективно извлекать их по мере необходимости.

*В главе 1* говорилось о том, что промпты могут включать в себя примеры ввода-вывода, известные как обучение с несколькими примерами (few-shot learning). Агенты могут извлекать эти примеры по мере необходимости для выполнения своих задач. *В главе 12* мы обсудим эффективные способы извлечения подходящих примеров.

Теперь давайте посмотрим, как агенты используют сессионную память для достижения своих целей.

## Сессионная память

Сессионная память (session memory) представляет собой механизм хранения информации о последовательности действий, предпринятых агентом как в текущих, так и в предыдущих сеансах. Эти данные можно дополнять комментариями и обратной связью, поступающей как от автоматизированных систем, так и от участников взаимодействия. Такой подход позволяет агенту обращаться к записям предыдущих сессий, чтобы лучше понимать контекст, оценивать ситуацию и более эффективно планировать последующие шаги в рамках текущего рабочего процесса.

Для наглядности приведем пример:

*Prompt*: Is it going to be warmer tomorrow than today in Paris? Give me details in Celsius.

*Промпт*: будет ли завтра в Париже теплее, чем сегодня? Укажи температуру в градусах Цельсия.

Агент выполняет запрос пользователя, выполняя определенную последовательность шагов. Все действия фиксируются в журнале текущего сеанса, который может выглядеть следующим образом:

*User prompt*: Is it going to be warmer tomorrow than today in Paris? Give me details in Celsius.

*LLM observation*: The user has a preference for Celsius units for temperature data.

*LLM Output*: Let's perform this step-by-step.

1. First, let's retrieve today's temperature.

2. Next, let's retrieve tomorrow's temperature.

3. We calculate the difference and generate the answer.

*External verifier*: Task decomposition is correct

*Tool selection*: <WeatherAPI> selected

*Tool info retrieval*: Retrieve information relevant to the query *Find temperature data from <WeatherAPI>*

*Tool invocation*: get_temperature_curweek(*Paris*)

*Tool output*: Weather data - Paris...

*LLM output*: Today's temperature is 27 degrees Celsius. Tomorrow's forecast is 23 degrees Celsius. Therefore, tomorrow is not going to be warmer in Paris than today.

*External Verifier*: Arithmetic operation is correct.

*Agent*: LLM output is dispatched to the user

*User feedback*: User marked this as correct

---

*Пользовательский промпт*: Будет ли завтра в Париже теплее, чем сегодня? Укажи температуру в градусах Цельсия.

*Наблюдение LLM*: пользователь предпочитает измерения температуры в градусах Цельсия.

*Выход LLM*: Будем решать задачу пошагово.

1. Определим температуру на сегодня.

2. Найдем температуру на завтра.

3. Вычислим разницу и сгенерируем ответ.

*Внешний верификатор*: декомпозиция задачи верна.

*Выбор инструмента*: выбран <WeatherAPI>.

*Извлечение информации с помощью инструмента*: извлечение информации, релевантной запросу *"Найти прогноз погоды"*

*from <WeatherAPI>*

*Вызов инструмента*: get_temperature_curweek(*Париж*)

*Выход инструмента*: Прогноз погоды - Париж...

*Выход LLM*: сегодня температура воздуха в Париже составляет 27 °C. Завтра температура составит 23 °C. Таким образом, завтра в Париже будет холоднее, чем сегодня.

*Внешний верификатор*: арифметическая операция выполнена верно.

*Агент*: Выход LLM отправлен пользователю

*Обратная связь от пользователя*: Пользователь отметил выход модели как правильный

История сеанса зачастую включает в себя большое количество информации, которая способна предоставить полезный персонализированный контекст для языковой модели относительно текущего пользователя, а также помочь направить модель к оптимальному рабочему процессу агентской системы. В более сложных вариантах реализации можно настроить несколько уровней ведения журнала, что позволит извлекать либо все журналы сеанса, либо только ключевые этапы, исходя из установленного уровня детализации.



Кроме истории сеанса, агент также может иметь доступ к образцам обучения, соответствующим идеальному рабочему процессу, которые он может использовать для управления своим поведением во время тестирования.

**Упражнение**

Средства для улучшения наблюдаемости агента включают такие инструменты, как LangSmith (**https://oreil.ly/6fn7p**), Langtrace (**https://oreil.ly/0FhZ9**), OpenLLMetry (**https://oreil.ly/1GRDY**) и др. Большинство из этих решений предоставляют доступ через *фримиум-модели* (freemium model). Воспользуйтесь функциями наблюдения от LangSmith для работы с агентом Llama 3, созданным в предыдущем упражнении, и проследите его действия при выполнении задачи.

Сессионная память может содержать сведения о взаимодействиях между пользователем и агентской системой, которые могут быть применены для настройки моделей под индивидуальные потребности. Эта тема будет подробно рассмотрена в *главе 12*, а сейчас мы остановимся на способах взаимодействия агента с данными инструментов.

### Данные инструментов

Данные инструментов предоставляют детальную информацию, необходимую для использования инструментов, включая схемы баз данных, API-документацию, примеры вызовов API и прочее. Когда агент принимает решение воспользоваться инструментом, модель обращается к хранилищу данных инструментов, чтобы получить актуальную информацию.

Например, при извлечении данных из БД с помощью SQL-инструмента модель может использовать схему БД из соответствующего хранилища. Там собраны сведения о таблицах, столбцах, описаниях их назначения и типах данных, а также дополнительная информация об индексах, первичных и вторичных ключах.



Можно также дообучить LLM на специализированном наборе данных, содержащем корректные SQL-запросы для работы с базой данных. Это позволит исключить необходимость обращения к схеме перед созданием запроса.

Таким образом, агенты могут использовать хранилища данных различными способами. Они имеют возможность извлекать промпты и примеры из репозитория, обращаться к истории рабочего процесса агентской системы, а также анализировать промежуточные результаты моделей, полученные в предыдущих сессиях. Это помогает глубже понять контекст и оптимизировать процесс работы. Помимо этого, они могут изучать документацию по инструментам для обеспечения их корректного использования.

Дополнительно агенты могут обращаться к внешним источникам знаний, таким как Интернет, базы данных и графы знаний. Организация поиска и извлечения нужной информации из этих ресурсов представляет собой отдельную подсистему, подробности которой будут рассмотрены в *главах 11* и *12*.

Далее мы рассмотрим промпт цикла агента (agent loop prompt), который играет важную роль в управлении поведением LLM по ходу агентского процесса.

## Промпт цикла агента

При использовании агентских систем важно помнить, что большие языковые модели не обладают сессионной памятью. Однако типичный рабочий процесс агентской системы подразумевает многократные обращения к LLM. В связи с этим возникает необходимость в механизме, который будет информировать LLM о текущем состоянии сессии, а также о роли модели в каждый момент времени. Управление этим циклом агента осуществляется посредством системного промпта.

Рассмотрим пример простого системного промпта цикла агента.

> *Prompt*: You are an AI model currently answering questions. You have access to the following tools: {tool_description}. For each question, you can invoke one or more tools where necessary to access information or execute actions. You can invoke a tool in this format: <TOOLNAME> <Tool Arguments>. The results of these tool calls are not provided to the user. When you are ready with the final answer, output the answer using the <Answer> tag.

---

*Промпт*: сейчас ты являешься моделью ИИ, отвечающей на вопросы. У тебя есть доступ к следующим инструментам: {tool_description}. В зависимости от запроса ты можешь пользоваться одним или несколькими инструментами для получения нужной информации или выполнения задачи. Инструменты вызываются в формате

<TOOLNAME> <Tool Arguments>. Результаты этих вызовов не будут видны пользователю. После формулировки окончательного ответа выведи его с помощью тега <Answer>.

Такой промпт может быть использован в большинстве случаев. Однако, если LLM демонстрирует неэффективные или ошибочные рассуждения, рекомендуется использовать метод ReAct (Reason + Act — "рассуждай и действуй"), который позволяет модели более разумно обрабатывать контекст задач и принимать решения.

### Метод ReAct

На момент написания книги метод ReAct-промптинга считается наиболее востребованным подходом для реализации цикла агента. Формат типичного промпта ReAct включает следующие этапы:

*Prompt*: You are an AI assistant capable of reasoning and acting. For each question, follow this process:

1. Thought: Reflect on the current state and plan your next steps.
2. Action: Execute the steps to gather information or call tools.
3. Observation: Record the results of your actions.
4. Final Answer: If you have an answer, provide a final response. Else continue the Thought → Action → Observation → loop until you have an answer.

---

*Промпт*: Ты ИИ-ассистент, способный анализировать и действовать. Для каждого нового вопроса придерживайся следующего порядка действий:

1. Рассуждение: оцени текущее состояние задачи и спланируй дальнейшие шаги.
2. Действие: выполни необходимые шаги, чтобы собрать данные или взаимодействовать с инструментами.
3. Наблюдение: запиши результаты своих действий.
4. Окончательный ответ: если решение найдено, предоставь итоговый ответ. Если окончательного ответа пока нет, повторяй цикл "Рассуждение → Действие → Наблюдение" до достижения результата.

Несмотря на широкое распространение и востребованность данного подхода, метод ReAct-промптинга демонстрирует некоторую нестабильность, что было отмечено в ряде исследований (**https://oreil.ly/RRZO9**).

### Рефлексия

Цикл агента может включать в себя этапы самопроверки (self-verification) или коррекции (correction). Эта концепция была впервые предложена в работе Шинн и соавт. (Shinn et al., **https://oreil.ly/xFVt0**) в рамках парадигмы рефлексии (reflection).

Рассмотрим системный промпт для модели Reflection-Llama-3.1 (**https://oreil.ly/foB-P**), в которой используются методы рефлексии.

*Prompt*: You are a world-class AI system, capable of complex reasoning and reflection. Reason through the query inside <thinking> tags, and then provide your final response

inside <output> tags. If you detect that you made a mistake in your reasoning at any point, correct yourself inside <reflection> tags.

*Промпт*: Ты система ИИ мирового уровня, способная к сложным рассуждениям и рефлексии. Рассуждай над запросом внутри тегов <thinking>, а затем выведи окончательный ответ внутри тегов <output>. В случае обнаружения ошибки в своих рассуждениях внеси необходимые корректировки внутри тегов <reflection>.

Теги `<reflection>` служат для самоанализа (интроспекции) и самокоррекции модели. Можно также указать условия, при которых должны активироваться эти теги, например, если агент повторяет одно и то же действие более трех раз подряд (это может указывать на то, что он застрял в цикле).



В некоторых случаях эффективность методов, основанных на рефлексии, может быть поставлена под сомнение. Их частое применение способно привести к принятию моделью неоптимальных решений.

**Упражнение**

Проверьте стабильность тестирования ReAct. Примените методы ReAct и Reflexion, описанные в этой главе, для управления агентом рецензирования фильмов, доступным в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Исследуйте возможность использования более упрощенного промпта и сравните его эффективность с методом ReAct.

Далее мы рассмотрим такие компоненты, как защитные ограждения (guardrails) и верификаторы (verifiers), которые обеспечивают стабильную работу агентской системы.

## Защитные ограждения и верификаторы

В производственных условиях ошибки могут привести к серьезным, иногда катастрофическим, последствиям. В зависимости от сценария использования, агенту может потребоваться соблюдать строгие стандарты в отношении достоверности, точности, безопасности и других критериев.

Обеспечение безопасности достигается за счет применения *защитных ограждений* (guardrails) — механизмов, которые предотвращают выход моделей за пределы допустимых значений в процессе работы. Примеры таких ограждений включают детектор нецензурной лексики (toxic language detector), детектор персональной информации (PII detector), фильтр входных данных (input filter), которые ограничивают тип запросов, разрешенных пользователям, и другие средства контроля.

*Верификатор* (verifier), в свою очередь, отвечает за поддержание стандартов качества агентской системы, обеспечивая ее способность восстанавливаться и самостоятельно корректировать ошибки после их выявления. Учитывая, что агентские

системы все еще находятся на начальном этапе развития, роль качественных и грамотно расположенных верификаторов приобретает критическую важность. Они могут принимать форму простых инструментов для сопоставления токенов или представлять собой сложные модели, включая символьные верификаторы (symbolic verifier) и другие подходы.

Рассмотрим подробнее эти инструменты.

### Защитные ограждения

Как отмечалось в *главе 2*, большие языковые модели обучаются преимущественно на текстах с веб-ресурсов, созданных пользователями. Однако, к сожалению, значительная часть этих данных содержит токсичный, оскорбительный, насильственный или порнографический контент. Очень важно предотвратить ситуации, когда приложения на базе LLM создают контент, потенциально нарушающий безопасность пользователей, или когда модели используются для генерации небезопасных данных. Несмотря на то что такие методы, как обучение на согласование (alignment training), могут значительно снизить вероятность появления вредоносного контента, они не способны гарантировать стопроцентной защиты. Для обеспечения безопасного использования необходимо установить дополнительные защитные механизмы времени вывода. Для этого можно использовать специализированные библиотеки и инструменты, такие как Guardrails (**https://oreil.ly/F7yax**), NeMo-Guardrails от NVIDIA (**https://oreil.ly/p7Dqz**), или специализированные модели, такие как Llama Guard (**https://oreil.ly/8S08P**), которые позволяют эффективно решить эту задачу.

Библиотека Guardrails предоставляет широкий спектр инструментов для обеспечения безопасности и надежности входных и выходных данных LLM. Среди наиболее важных инструментов можно выделить следующие.

*Обнаружение персонально идентифицируемой информации (personally identifiable information, PII)*

Этот инструмент используется для выявления конфиденциальной информации как во входных, так и в выходных данных. Для выполнения анализа применяется библиотека Microsoft Presidio (**https://oreil.ly/eG8T1**).

*Обнаружение промпт-инъекций*

Этот инструмент помогает выявлять определенные типы вредоносного промптинга и предотвращать неправильное использование LLM. Для выполнения анализа используется библиотека Rebuff (**https://oreil.ly/nIyE5**).

*Обнаружение NSFW-текста*

Этот инструмент предназначен для обнаружения текста, содержащего небезопасный/неподходящий для работы (not safe for work, NSFW) контент в выходных данных LLM. Речь идет о тексте, содержащем ненормативную лексику, насилие и сексуальное содержание. Существует также валидатор *Profanity free* ("Отсутствие ненормативной лексики"), который проверяет текст исключительно на наличие ненормативной лексики.

*Проверка вежливости (politeness check)*

Этот валидатор оценивает уровень вежливости текста, генерируемого LLM. В качестве примера можно привести детектор нецензурной лексики.

*Очистка веб-страниц (web sanitization)*

Этот валидатор проверяет вывод LLM на наличие потенциальных уязвимостей в безопасности, включая код, который может быть запущен в браузере. Для обнаружения потенциальных уязвимостей и очистки вывода можно воспользоваться библиотекой Bleach (**https://oreil.ly/r3Xrl**).

В случае если проверка не пройдена и на входе или выходе обнаружено вредоносное содержимое, механизм защитных ограждений предоставляет несколько вариантов действий.

*Повторный вопрос (re-ask)*

В этом методе LLM предлагается повторно сгенерировать выходные данные, при этом промпт содержит инструкции для конкретного соблюдения критериев, по которым выходные данные ранее не прошли проверку.

*Исправление (Fix)*

В этом методе библиотека сама исправляет вывод, не запрашивая у LLM повторной генерации. Исправления могут включать удаление или замену определенных частей ввода или вывода.

*Фильтр (Filter)*

При генерации структурированных данных этот параметр включает фильтрацию только не прошедшего проверку атрибута. Остальная часть выходных данных возвращается пользователю.

*Рефрен (Refrain)*

В этой настройке вывод просто не возвращается пользователю, и пользователь получает отказ.

*Ничего не делать (noop)*

В случае возникновения ошибки валидации, которая не позволяет выполнить действие, информация об ошибке записывается в журнал для последующего анализа.

*Исключение (exception)*

Этот метод позволяет вызывать исключения в случае возникновения ошибок в программном обеспечении. Для реализации специального поведения, определенного пользователем, можно настроить обработчики исключений (exception handlers).

*fix_reask*

В рамках данного метода библиотека пытается самостоятельно исправить вывод, после чего выполняется повторная проверка. Если повторная проверка также завершается неудачно, LLM предлагается восстановить выходные данные.

В качестве примера рассмотрим средства ограждения персонально идентифицированных данных (PII):

```
from guardrails import Guard
from guardrails.hub import DetectPII

guard = Guard().use(
    DetectPII, ["EMAIL_ADDRESS", "PHONE_NUMBER"], "reask")

guard.validate("Нобелевскую премию в этом году получил Джефф Хинтон. С ним можно
связаться по телефону +1 234 567 8900")
```

**Упражнение**

Создание защиты от извлечения системного промпта является актуальной задачей в контексте джейлбрейка (jailbreak — "побег из тюрьмы", "взлом"), который представляет собой набор методов, используемых для обхода ограничений, встроенных в модели. Необходимо создать механизм защиты, который позволит предотвратить извлечение системного промпта пользователями.

Рассмотрим подробно принципы работы модулей верификации (verification module).

## Модули верификации

Как уже отмечалось в книге, современные большие языковые модели сталкиваются с рядом проблем, таких как ограниченные способности к рассуждению и склонность к галлюцинациям, что существенно снижает их надежность. Между тем, чтобы заслужить доверие пользователей, готовые к внедрению приложения должны обладать стабильным уровнем надежности. Одним из подходов к увеличению надежности систем, основанных на LLM, является принятие стратегии "человек в цикле" (human-in-the-loop), когда человек вручную проверяет результаты и предоставляет обратную связь. Однако на практике такой метод не всегда удобен или применим. Более востребованной альтернативой считается интеграция внешних модулей проверки в систему LLM. Такие модули могут представлять собой как простые программы, основанные на заранее заданных правилах, так и небольшие дообученные модели или символьные решатели (symbolic solver). Кроме того, предпринимаются усилия по использованию самих LLM в роли верификаторов, что получило название "LLM в роли арбитра" (LLM-as-a-judge).

В состав системы входят резервные модули (fallback module), которые активируются в случае сбоя проверки или невозможности исправления ошибки. Резервные модули могут быть представлены в виде простых сообщений вроде "К сожалению, я не могу выполнить ваш запрос", так и более сложных решений, адаптированных под конкретные рабочие процессы.

Рассмотрим пример. Предположим, имеется приложение *абстрактной суммаризации* (abstractive summarization), предназначенное для работы с финансовыми документами. Для обеспечения качества и надежности генерируемых сводок необходимо интегрировать в архитектуру системы механизмы верификации и самоисправления.

Как оценить качество абстрактной суммаризации? Несмотря на наличие ряда метрик, которые позволяют автоматически количественно оценить качество сводки, более эффективным подходом является определение критериев, которым должна соответствовать хорошая сводка, и последующая проверка соответствия каждому из этих критериев.



Существует несколько количественных показателей, которые используются для оценки качества суммаризации. К ним относятся метрики BLEU, ROUGE (**https://oreil.ly/LPlFJ**) и BERTScore (**https://oreil.ly/gsOGl**). Метрики BLEU и ROUGE основаны на эвристическом подходе, который заключается в сопоставлении токенов. Однако эти метрики показали свою несостоятельность (**https://oreil.ly/rSzbR**). Более перспективной считается метрика BERTScore, основанная на семантическом сходстве. Однако следует отметить, что все методы оценки качества сводок являются субъективными и требуют более комплексного подхода для обеспечения объективности результатов.

Для эффективной суммаризации финансовых документов следует учитывать следующие ключевые критерии.

*Достоверность (factuality)*

Сводка должна быть достоверной, отражая фактические данные из исходного текста, без допущений или неправильных выводов.

*Конкретность (specificity)*

Важно избегать обобщений и предоставлять конкретные детали, включая цифры или важные именованные сущности.

*Релевантность (relevance)*

Оценка релевантности проводится как процент значимых предложений, которые действительно заслуживают включения в итоговый текст.

*Полнота (completeness)*

Этот критерий оценивает процент элементов в исходном документе, которые были включены в сводку.

*Отсутствие повторов (repetitiveness)*

В итоговой сводке не должно быть повторов, даже если они присутствуют в исходном документе.

*Последовательность (coherence)*

В целом итоговая сводка должна четко передавать содержание исходного документа и минимизировать неоднозначность. Этот критерий является более субъективным.

*Структура (structure)*

При определении задачи суммаризации можно задать нужную структуру сводки. Это значит, что итоговый текст будет включать заранее заданные разделы и подразделы, обеспечивая удобство восприятия информации. Итоговая сводка должна строго соответствовать указанной структуре.

*Форматирование (formatting)*

Генерируемая сводка должна быть оформлена в соответствии с установленными стандартами. Например, если сводка представляет собой маркированный список, то все его элементы должны быть выделены маркерами.

*Упорядочивание (ordering)*

Элементы сводки должны быть расположены в таком порядке, который способствует ясному пониманию содержания. Дополнительно может быть задан определенный принцип упорядочивания, например хронологический.

*Обработка ошибок (error handling)*

В случае обнаружения ошибок или пропусков в исходном документе, необходимо предусмотреть соответствующую процедуру их обработки.

**Упражнение**

Определите критерии оценки качества ответов ассистента на вопросы по данным заседаний парламента Канады, представленным в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). В чем заключаются отличия этих критериев от задачи абстрактной суммаризации?

Как автоматически определить, соответствует ли определенная сводка всем заданным критериям?

Для этого можно использовать комбинацию дообученных моделей и методов, основанных на правилах. В конечном итоге строгость методов, используемых для проверки, зависит от уровня надежности, необходимого для конкретного приложения.

Однако стоит отметить, что если мы сокращаем масштабы процесса проверки для проверки отдельных критериев, а не всего приложения в целом, то упрощается использование недорогих методов для точной проверки. Рассмотрим процесс разработки верификаторов для каждого критерия задачи абстрактной суммаризации.

*Достоверность (factuality)*

Определить достоверность утверждения, сгенерированного LLM, без доступа к исходной информации крайне затруднительно. Однако в приложениях суммаризации существует возможность обратиться к эталонным данным. Так можно проверить достоверность, анализируя каждое предложение в сводке и выясняя, можно ли логически завершить утверждение в сводке на основе исходного тек-

ста. Такой процесс напоминает задачу вывода на естественном языке (natural language inference, NLI), которая широко применяется в области обработки естественного языка (NLP).

В задачах NLI целью является установление логической связи между гипотезой (hypothesis) и предпосылкой (premise). В данном случае гипотеза представляет собой предложение из сводки, а предпосылка — исходный текст.

Обучение модели NLI для конкретной предметной области может быть сложным и ресурсозатратным процессом. В случаях, когда нет возможности использовать специализированную модель, можно применить статистические методы, такие как наложение токенов (token overlap) или анализ совпадений между текстами, чтобы получить приблизительную оценку достоверности.

Для проверки достоверности чисел и именованных сущностей можно использовать метод сопоставления строк. Он заключается в проверке соответствия всех чисел и именованных сущностей, указанных в сводке, с исходным текстом.

*Конкретность (specificity)*

Одним из способов сделать сводку более конкретной является включение в нее чисел и именованных сущностей там, где это необходимо. Для каждого предложения в сводке можно проверить, содержит ли исходный документ, связанный с темой предложения, какие-либо числа и именованные сущности, и отражены ли они в сводке. Числа и именованные сущности можно пометить тегами и обнаружить с помощью регулярных выражений или библиотек, например spaCy (**https://oreil.ly/zatAW**).

*Релевантность/точность (relevance/precision)*

Можно обучить модель классификации (classification model), которая будет определять, является ли предложение в тексте релевантным. Однако стоит отметить, что этот подход имеет свои ограничения. Если бы модель классификации была достаточно точной, ее можно было бы использовать для непосредственного отбора подходящих предложений из исходного текста с целью создания сводки. На практике модель классификации можно использовать для удаления очевидного и несущественного контента.

*Полнота (recall/completeness)*

Вопрос о том, какое содержание следует включить в сводку, является сложным, особенно если существует ограничение на ее длину. Можно обучить модель ранжирования (ranking model), которая будет определять важность предложений в исходном документе, а затем проверить, включены ли в сводку предложения, занимающие высокие позиции в рейтинге. Также можно заранее определить тип информации, которая должна быть представлена в сводке, и создать модель классификации для выявления частей исходного документа, содержащих соответствующую информацию. С помощью таких метрик, как сходство эмбеддингов (embedding similarity), можно оценить, насколько в сводке адекватно представлен контент.

*Отсутствие повторов (repetitiveness)*

Для выявления повторов можно использовать различные алгоритмы сопоставления строк, например, *расстояние Жаккара* (Jaccard distance, **https://oreil.ly/Ny_Ku**) или оценку степени сходства эмбеддингов между парами предложений в сводке.

*Последовательность (coherence)*

Это, вероятно, один из самых сложных критериев для проверки. Один из подходов к решению этой задачи, хотя и более затратный, состоит в разработке специализированной модели детекции (detection model). Такая модель позволяет проверять каждое предложение в сводке на предмет того, являются ли все предшествующие ему предложения достаточными предпосылками для правильного понимания. Более подробную информацию о методах выявления предпосылок можно найти в работе Тареджа и соавт. (Thareja et al., **https://oreil.ly/6JnRs**).

*Структура (structure)*

Если заранее определить структуру сводки, включающую разделы (sections) и подразделы (subsections), можно легко проверить соответствие структуры, убедившись в наличии желаемых заголовков разделов и подразделов. Также можно использовать методы сходства эмбеддингов для проверки соответствия содержания разделов и подразделов их названиям.

*Форматирование (formatting)*

В рамках данного этапа осуществляется проверка соответствия содержимого определенному формату, например, является ли оно маркированным списком или представляет собой корректный объект JSON.

*Упорядочивание (ordering)*

Упорядочивание может осуществляться в хронологическом порядке, в алфавитном порядке, по домену или в соответствии с требованиями конкретной задачи. Если необходимо обеспечить хронологический порядок, можно проверить наличие дат в сводке и убедиться, что они расположены в хронологическом порядке. В случае более сложных требований к порядку проверка его соблюдения может оказаться крайне сложной задачей.



Не стоит ожидать, что процесс проверки будет значительно эффективнее модели суммаризации. В случае если бы это было возможно, процесс проверки можно было бы использовать для создания самой сводки.

Также можно использовать символьные верификаторы, такие как решатели SAT (**https://oreil.ly/lOsg_**), которые решают задачи выполнимости булевых формул, и логические планировщики. Однако данный тип проверки не входит в рамки обсуждения в этой книге.

**Упражнение**

Для задачи, представленной в предыдущем упражнении, а именно для ассистента по ответам на вопросы, связанные с заседаниями парламента Канады, как бы вы разработали модули проверки для каждого из определенных критериев? Возможно ли провести надежную проверку, используя только эвристические методы?

После интеграции модулей верификации в системную архитектуру необходимо определить, какие меры предпринимать при выявленных сбоях. Одним из подходов может быть повторное обращение к языковой модели. В этом случае возможна регенерация как всего вывода, так и только той его части, которая не прошла проверку. Дополнительно можно рассмотреть создание антихрупких архитектур (antifragile architecture) с резервными механизмами для обработки подобных ситуаций, что будет подробно рассмотрено в *главе 13*.



Добавление дополнительных верификаторов может привести к существенному увеличению времени отклика системы. В связи с этим, их использование должно быть тщательно продумано с учетом требований к точности и скорости работы системы.

В заключение рассмотрим программное обеспечение для оркестрации агентов, которое объединяет все перечисленные компоненты.

## Программное обеспечение для оркестрации агентов

Для обеспечения бесперебойной работы агентской системы необходимо использовать программное обеспечение, которое связывает все компоненты системы. Программное обеспечение для оркестрации (orchestration) агентов управляет состоянием системы, вызывает необходимые инструменты, инициирует поиск данных, управляет буферами и регистрирует промежуточные и конечные результаты.

Существует множество фреймворков для агентов как с открытым исходным кодом, так и проприетарных, которые выполняют эту функцию. К ним относятся LangChain (**https://oreil.ly/7vmlY**), LlamaIndex (**https://oreil.ly/uxejK**), CrewAI (**https://oreil.ly/Ntxii**), AutoGen (**https://oreil.ly/tx3qy**), MetaGPT (**https://oreil.ly/HIJn**), XAgent (**https://oreil.ly/sA_DR**), llama-stack-apps (**https://oreil.ly/SBGC_**) и др.



Агенты представляют собой сравнительно новую концепцию, поэтому многие агентские фреймворки, вероятно, претерпят значительные изменения в ближайшие месяцы и годы. Эти платформы строятся на базе заранее заданной уверенности в действиях, что зачастую делает их менее гибкими. Для прототипирования рекомендуется рассмотреть такие решения, как LangChain или LlamaIndex, которые удобны и просты в использовании. Однако для рабочего применения может возникнуть необходимость разработать собственный фреймворк либо доработать существующие открытые

исходные коды. Пример базового агентского подхода доступен в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).

Таким образом, мы подробно изучили ключевые элементы агентской системы, и теперь пришло время применить полученные знания на практике. В репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**) представлены различные примеры реализации агентов разных типов. Рекомендуется адаптировать их под конкретный случай использования, чтобы лучше разобраться в принимаемых компромиссах и найти оптимальные решения.



Принцип KISS ("Keep It Simple, Stupid" — "Делай проще, тупица") применим к разработке агентов, пожалуй, сильнее, чем любая другая современная концепция. По возможности избегайте излишнего усложнения архитектуры агента, если на то нет веских причин. Мы подробно остановимся на этом вопросе в *главе 13*.

### Веб-агенты и автоматизация компьютерных задач

В данной главе мы рассмотрели примеры агентов, которые выполняют действия в реальном мире, взаимодействуя с программным интерфейсом. Однако в процессе автоматизации рутинных задач часто возникает необходимость работы с веб-интерфейсом, таким как веб-страница, который не предоставляет возможности для прямого взаимодействия с программным интерфейсом. Существует множество задач, связанных с обработкой данных, таких как копирование и вставка информации между системами, заполнение таблиц в Excel на основе данных из другой системы и т. д., которые требуют значительных усилий и времени. Могут ли агенты помочь автоматизировать подобные задачи?

В настоящее время существует новая парадигма агентов, известная как веб-агенты (web agents), которая обещает решить эту проблему. Веб-агенты используют объектную модель документа (Document Object Model, DOM) или снимок экрана веб-страницы для анализа текущего состояния страницы и выполнения таких действий, как ввод информации в поля, нажатие на элементы и переход по ссылкам. Работающий веб-агент может автоматически забронировать рейс, перейдя на туристический веб-сайт, введя информацию, выбрав различные варианты и осуществив оплату. На данный момент это все еще новая технология, которая имеет ограниченные результаты при выполнении задач, связанных с бенчмарками (**https://oreil.ly/Dyp2L**).

Некоторые компании, в частности, Anthropic, разработали начальные версии программного обеспечения, которое позволяет агентам управлять средой рабочего стола компьютера.

Для запуска демонстрационной версии программного обеспечения Anthropic, осуществляющего управление работой компьютера, перейдите по ссылке **https://oreil.ly/nUtar**. Обратите внимание на системный промпт (**https://oreil.ly/AsFpP**). Какие типичные ошибки или сбои наблюдаются в работе приложения?

# Резюме

В этой главе мы изучили различные способы взаимодействия LLM с внешними инструментами. Мы рассмотрели агентскую парадигму и дали формальное определение агентов. Мы подробно рассмотрели компоненты агентской системы, включая модели, инструменты, хранилища данных, ограждения и верификаторы, а также программное обеспечение для оркестрации агентов. Мы также исследовали способы определения и внедрения собственных инструментов.

В следующей главе мы рассмотрим представление и извлечение данных, которые являются важными элементами взаимодействия LLM с внешними данными.

# ГЛАВА 11

# Обучение представлениям и эмбеддинги

В предыдущей главе мы рассмотрели, как можно организовать взаимодействие языковой модели с внешними инструментами, включая хранилища данных. Внешние данные могут быть представлены в различных форматах, таких как текстовые файлы, таблицы базы данных и графики базы знаний. Эти данные могут охватывать широкий спектр типов контента, от специализированных баз знаний для конкретных областей до промежуточных результатов и выходных данных, генерируемых языковой моделью.

Если данные структурированы и находятся в реляционной базе данных (РБД), языковая модель может выполнить SQL-запрос для получения необходимых данных. Однако что делать, если данные представлены в неструктурированном виде?

Одним из методов извлечения данных из неструктурированных текстовых источников является поиск по ключевым словам или использование регулярных выражений. В примере о финансовом директоре компании Apple, рассмотренном в предыдущей главе, можно извлечь текст с упоминаниями финансового директора из корпуса данных, включающего финансовую информацию, в надежде найти дату назначения или информацию о сроке пребывания в должности. Например, можно применить следующее регулярное выражение:

```
pattern = r"(?i)\b(?:C\.?F\.?O|Chief\s+Financial\s+Officer)\b"
```

Эффективность поиска по ключевым словам ограничена. Существует множество способов указать дату присоединения к должности финансового директора или период его пребывания в должности, если такая информация представлена в текстовом массиве. Применение универсального регулярного выражения, подобного упомянутому, часто приводит к большому количеству ложных срабатываний.

В связи с этим необходимо выйти за рамки традиционного поиска по ключевым словам. За последние десятилетия было разработано и внедрено несколько подходов к информационному поиску, таких как BM25 (Best Match — лучшее совпадение), которые стали основой для современных поисковых систем. Эти методы подробно рассматриваются в *главе 12*.

Однако в эпоху больших языковых моделей стандартным инструментом для решения подобных задач становятся системы, использующие поиск на основе эмбеддингов.

В этой главе мы рассмотрим принципы работы эмбеддингов, концепцию семантического сходства (semantic similarity) и методы оценки сходства. Мы изучим применение популярных моделей эмбеддинга и научимся оценивать их эффективность. Также мы рассмотрим способы адаптации моделей эмбеддинга для конкретных сценариев использования и предметных областей (доменов). Мы разберем подходы к интерпретации эмбеддингов с использованием разреженных автоэнкодеров (sparse autoencoder, SAE). В заключение мы рассмотрим методы оптимизации эмбеддингов, которые позволяют уменьшить требования к объему памяти и вычислительным ресурсам.

## Введение в эмбеддинги

Обучение представлениям (representation learning) является важным направлением машинного обучения, которое занимается обучением представлениям данных таким образом, чтобы зафиксировать их значимые особенности, часто в пространстве малой размерности (low dimensional space). В контексте обработки естественного языка это подразумевает преобразование текстовых элементов, таких как слова, предложения или абзацы, в векторную форму, известную как эмбеддинги (embedding). Эмбеддинги отражают семантические (связанные со значением) и прагматические (связанные с социальным контекстом) особенности входного текста.

Эмбеддинги создаются посредством использования как открытых библиотек, так и коммерческих API. Одной из наиболее известных библиотек с открытым исходным кодом является Sentence Transformers (**https://oreil.ly/4OSVd**), которая предоставляет широкий набор моделей для работы с эмбеддингами, сопоставимых по качеству с проприетарными решениями.

Теперь с помощью библиотеки Sentence Transformers создадим эмбеддинги:

```
from sentence_transformers import SentenceTransformer, util
sbert_model = SentenceTransformer('msmarco-distilbert-base-tas-b')
embedding = sbert_model.encode("American pizza is one of the nation's greatest
cultural exports",
// 'Американская пицца считается одним из выдающихся культурных достижений страны

show_progress_bar=True, device='cuda', convert_to_tensor=True)
print("Embedding size:", embedding.shape[0])
print(embedding)
```

На выходе получаем:

```
Embedding size: 768

tensor([-3.9256e-01, 1.0734e-01, 1.3579e-01, 7.6147e-02, 5.2521e-02,
-6.5887e-03, 1.9225e-01, 3.5374e-01, 2.5725e-01, 5.6408e-02,...])
```

Размер этой модели эмбеддинга составляет 768, что означает, что каждый вектор имеет 768 измерений. Длина последовательности в данной модели ограничена 512 токенами, поэтому входной текст не должен превышать это количество, иначе он будет обрезан. Вектор эмбеддинга состоит из чисел с плавающей запятой, которые сами по себе не имеют смысла. Методы интерпретации эмбеддингов будут рассмотрены по ходу главы.

Большинство современных моделей эмбеддинга основаны на языковых моделях с кодировщиком, которые были рассмотрены в *главе 4*. Наиболее популярными моделями являются BERT, RoBERTa, MPNet и др. Эти модели обычно дообучаются на наборах данных, содержащих перефразированные тексты или ответы на вопросы, а также на текстах, извлеченных из естественного языка.

Рассмотрим, как генерируются эмбеддинги из моделей такого типа (т. е. что делает функция `sentence_transformers.encode()`):

```
from transformers import AutoTokenizer, AutoModel
import torch

tokenizer=
AutoTokenizer.from_pretrained("sentence-transformers/msmarco-distilbert-base-tas-b")
model =
AutoModel.from_pretrained("sentence-transformers/msmarco-distilbert-base-tas-b")

input = tokenizer(
    'American pizza is one of the nation's greatest cultural exports',
padding=True, truncation=True, return_tensors='pt')

with torch.no_grad():
    output = model(**input, return_dict=True)
    embedding = output.last_hidden_state[:, 0]
print(embedding)
```

В данном примере эмбеддинг извлекается из токена `[CLS]` на последнем слое модели DistilBERT. Существуют и другие методы извлечения эмбеддингов из моделей, среди которых можно выделить:

- метод среднего пулинга (mean pooling) — усреднение значений всех выходов токенов в последовательности;
- метод максимального пулинга (max pooling) — выбор максимального значения в каждом измерении среди всех токенов;
- средневзвешенное значение (weighted mean) — техника, при которой более поздним токенам в последовательности присваиваются бóльшие веса;
- использование последнего токена (last token) — эмбеддинг представляет собой вывод кодировщиком последнего токена в последовательности.



Наличие качественного представления всей последовательности в последнем или первом токене во многом определяется задачами предобучения и дообучения модели. Например, цель предобучения BERT, связанная с предсказанием следующего предложения, делает токен `[CLS]` более информативным и насыщенным в плане представления, чем токен `<s>` у RoBERTa. Поскольку RoBERTa изначально не включает задачу предсказания следующего предложения, ее начальный токен `<s>` не обладает значительной смысловой нагрузкой.

Модели эмбеддинга на базе декодеров, такие как SGPT, в последнее время набирают популярность. OpenAI предлагает универсальную конечную точку для работы с эмбеддингами, которая подходит как для задач поиска, так и для оценки сходства. Преимуществом эмбеддингов от OpenAI является значительно бóльшая максимальная длина последовательности, достигающая 8192 токенов, а также увеличенный масштаб размерности — от 1536 до 3072. Среди других поставщиков решений в области эмбеддингов можно выделить такие компании, как Cohere и Jina.

Выбор оптимальной модели для конкретной задачи требует учета ряда факторов, таких как стоимость, задержка, ограничения системы хранения, производительность и особенности данных в контексте использования. В качестве отправной точки рекомендуется воспользоваться компактной и высокоэффективной моделью all-mpnet-base-v2, которую можно найти в библиотеке Sentence Transformers. Она зарекомендовала себя как универсальный инструмент в области обработки естественного языка. Тем не менее будет нелишним поэкспериментировать и с другими моделями. В последующих разделах этой главы мы рассмотрим дополнительные рекомендации по выбору подходящих моделей, а также методы оценки качества эмбеддинга и анализ популярных бенчмарков.



Не существует такого понятия, как бесконечное сжатие. Размеры эмбеддинга имеют фиксированное значение, поэтому при увеличении длины ввода количество информации, которое можно закодировать в эмбеддинге, уменьшается. Выбор оптимального варианта зависит от конкретной ситуации.

## Семантический поиск

Ценность эмбеддингов наиболее полно раскрывается при их использовании для представления обширного текстового корпуса. Векторы данных формируют так называемое пространство эмбеддингов (embedding space), в котором схожие тексты располагаются ближе друг к другу. Такое свойство дает возможность применять метрики сходства для решения сложных задач, включая кластеризацию (clustering) или семантический поиск (semantic search). Под семантическим поиском подразумеваются методы, которые учитывают смысл и контекст запросов и документов для выявления наиболее соответствующих данному запросу.

Для визуализации пространства эмбеддингов можно воспользоваться методами уменьшения размерности, такими как метод главных компонент (principal component analysis, PCA, **https://oreil.ly/Rk1M9**) или алгоритм стохастического вложения соседей с t-распределением (t-distributed Stochastic Neighbor Embedding, t-SNE, **https://oreil.ly/0xNrB**).

В качестве иллюстрации на рис. 11.1 представлена визуализация эмбеддингов постов, опубликованных в социальной сети X (ранее известной как Twitter) членами Конгресса США. Данная работа была выполнена компанией Nomic AI с использованием инструмента Atlas. Более подробное описание визуализации доступно в блоге Nomic (**https://oreil.ly/AORpk**).

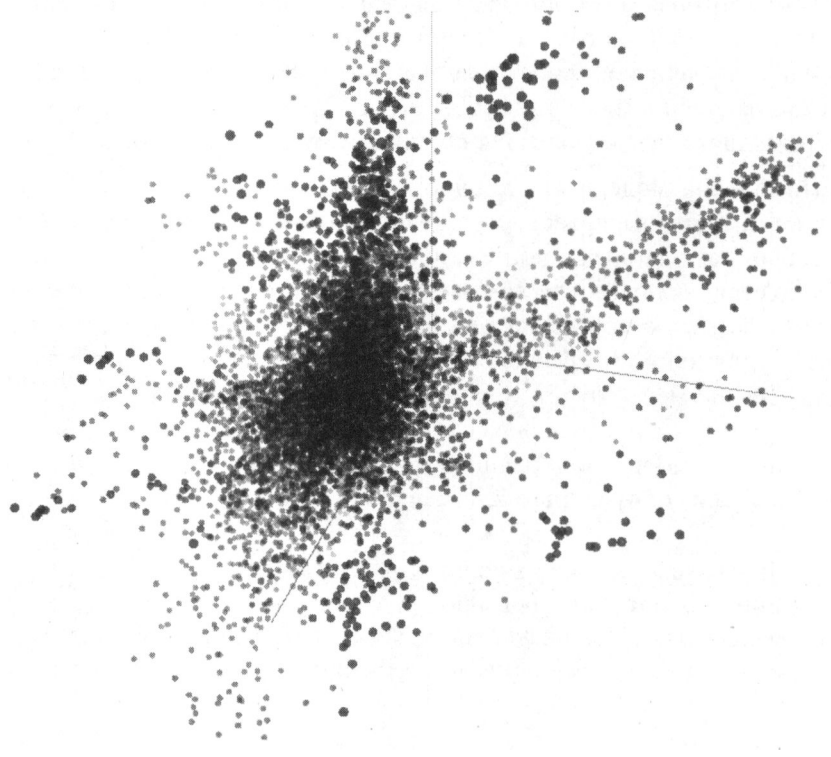


**Рис. 11.1.** Визуализация пространства эмбеддингов

Рассмотрим применение эмбеддингов в семантическом поиске. Для пользовательского запроса создается эмбеддинг, после чего происходит поиск наиболее близких эмбеддингов документов в векторном пространстве. В результате поиска возвращаются тексты, соответствующие top-*k* наиболее схожим эмбеддингам, где *k* может быть как равно 1, так и меняться в зависимости от задач приложения. Этот процесс называется *извлечением* (retrieval). Затем найденные тексты вместе с запросом пользователя включаются в промпт для LLM, и на основе информации из предоставленного контекста модель генерирует ответ. Такой двухэтапный процесс традиционно называется фреймворком *извлечения-считывания* (retriever-reader framework), где LLM выступает в роли считывателя.

В качестве простого примера рассмотрим два предложения из нашего корпуса:

```
chunks = ['The President of the U.S is Joe Biden',
'Ramen consumption has increased in the last 5 months']
// 'Президент США - Джо Байден',
// 'За последние пять месяцев наблюдается рост потребления хвойной древесины'
```

Учитывая фразу "президент США" в запросе, можно закодировать этот запрос и чанки с помощью Sentence Transformers следующим образом:

```
from sentence_transformers import SentenceTransformer, util
sbert_model = SentenceTransformer('msmarco-distilbert-base-tas-b')
chunk_embeddings = sbert_model.encode(chunks, show_progress_bar=True, device='cuda',
normalize_embeddings=True, convert_to_tensor=True)
query_embedding = sbert_model.encode(query, device='cuda', normalize_embeddings=True,
convert_to_tensor=True)
matches = util.semantic_search(query_embedding, chunk_embeddings,
score_function=util.dot_score)
```

На выходе получаем:

```
[[{'corpus_id': 0, 'score': 0.8643729090690613},
  {'corpus_id': 1, 'score': 0.6223753690719604}]]
```

Как видно, оценка сходства первого предложения оказалась значительно выше, поэтому оно будет возвращено в качестве ответа на запрос.



Семантический поиск делится на два типа: симметричный и асимметричный. При симметричном поиске текст запроса имеет аналогичный объем, как и текст документа. В случае асимметричного поиска размер текста запроса существенно меньше, чем текста документа, что типично для поисковых систем или запросов к ассистентам. Существуют модели, разработанные исключительно для одного из этих типов поиска. Кроме того, некоторые модели применяют отдельные алгоритмы для кодирования текстов запросов и документов.

## Метрики сходства

В качестве метрик сходства обычно используются скалярное произведение (dot product), косинусное сходство (cosine similarity) и евклидово расстояние (Euclidean distance). Более подробную информацию можно найти в руководстве по Pinecone (**https://oreil.ly/X_qcD**). При работе с моделями эмбеддинга рекомендуется использовать ту метрику сходства, которая применялась при обучении модели. Детали об используемом показателе обычно содержатся в карточке модели или на странице модели на Hugging Face.



При установке параметра `normalize_embeddings` в функции `encode()` в значение `True` эмбеддинги автоматически нормализуются к единичной длине. Это обеспечивает совпадение результатов скалярного произведения и косинусного сходства. Важно отметить, что скалярное произведение выполняется быстрее, чем вычисление косинусного сходства. Библиотека Sentence Transformers предоставляет специализированные модели (**https://oreil.ly/LOu75**), обученные для работы как со скалярным произведением, так и с косинусным сходством. При этом упоминается, что модели, ориентированные на скалярное произведение, могут демонстрировать склонность к извлечению более длинных чанков.

### Упражнение

Рассмотрите варианты использования различных методов пулинга для извлечения эмбеддингов из моделей. В качестве источника информации можно использовать код, предложенный Sentence Transformers (**https://oreil.ly/8QpBj**). Проанализируйте, как меняются оценки сходства для предложений из приведенного примера. Затем проведите аналогичный эксперимент и сравните результаты с использованием различных метрик сходства.

Хотя понятие семантического сходства является мощным инструментом, оно не является универсальным решением для всех задач. Проблема заключается в том, что концепция семантического сходства остается недостаточно строго определенной. Прежде всего, семантическое сходство — это понятие, которое описывает степень схожести или идентичности между двумя объектами или сущностями. Однако даже для двух объектов некоторые аспекты могут быть схожими, а другие различаться.

Для иллюстрации рассмотрим три предложения.

> После 25 лет работы в компании г-н Поморенко подтвердил, что не выходит на пенсию.
>
> Вчера г-н Поморенко объявил о выходе на пенсию.
>
> Вчера г-н Поморенко не объявлял о выходе на пенсию.

Для расчета сходства и кодирования воспользуемся моделью эмбеддинга all-mpnet-base-v2 из библиотеки Sentence Transformers:

```
!pip install sentence-transformers

from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-mpnet-base-v2')

sentences = ['После 25 лет работы в компании г-н Поморенко подтвердил, что не уходит
на пенсию ', 'Вчера г-н Поморенко объявил о выходе на пенсию']
```

```
embeddings = model.encode(sentences)
cosine_scores = util.cos_sim(embeddings[0], embeddings[1])
print("Косинусное сходство:", cosine_scores.item())
```

На выходе получаем:

```
Косинусное сходство: 0.7870
```

Если заменить второе предложение на "Вчера г-н Поморенко не объявлял о выходе на пенсию", на выходе получаем:

```
Косинусное сходство: 0.7677
```

Оба предложения выглядят почти одинаково по отношению к первому, и в некотором смысле это действительно так. Они похожи тем, что упоминают г-на Поморенко и касаются темы выхода на пенсию. В то же время одно из предложений передает совершенно противоположный смысл, отрицая факт выхода на пенсию.



Один из методов обработки ложных срабатываний, вызванных использованием нежелательных размерностей сходства (например, отрицания), заключается в увеличении значения $k$ в результатах top-$k$, которые возвращаются в качестве ответа на запрос. Это позволяет LLM различать ложные срабатывания и использовать корректную информацию для ответа на запрос. Однако увеличение top-$k$ также приводит к увеличению длины контекста промпта, что может увеличить задержку и стоимость обработки запроса.

Требования к приложению определяют, какие метрики сходства играют ключевую роль. Если для приложения важно учитывать отрицание как особое отношение, имеет смысл отразить это в пространстве эмбеддингов. В таких случаях полезным инструментом может стать процесс дообучения моделей эмбеддингов. С помощью тонкой настройки этих моделей можно адаптировать пространство эмбеддингов под конкретные задачи или предпочтения. Этот подход несложен в реализации и часто приносит значительные преимущества.

Кроме того, дообучение эмбеддингов особенно актуально для работы с узкоспециализированными областями данных, где распределение токенов заметно отличается от распределения в данных общего назначения. Теперь подробнее разберем процесс дообучения моделей эмбеддингов.

### Упражнение

В примере о выходе на пенсию г-на Поморенко оцените, насколько схожи результаты при использовании эмбеддингов из Jina, Nomic и OpenAI. Каковы оценки сходства в этих случаях? Превосходят ли они результаты, полученные моделью all-mpnet-base-v2, или уступают им?

# Дообучение моделей эмбеддингов

Библиотека Sentence Transformers предоставляет возможность дообучения моделей эмбеддингов с использованием класса `SentenceTransformerTrainer` (**https://oreil.ly/Jahep**). Для проведения дообучения модели эмбеддингов требуется наличие базовой модели (base model), набора обучающих данных (training dataset) и цели обучения (learning objective).

## Базовые модели

Для дообучения модели, например all-mpnet-base-v2, или базовой модели MPNet необходимо учитывать объем доступных обучающих данных. Базовую модель, как правило, требуется дообучать на большем количестве данных, чем уже дообученную версию. Среди возможных вариантов для дальнейшего дообучения также можно рассмотреть такие модели, как BGE-M3 (**https://oreil.ly/Sh8pZ**) и jina-embeddings-v3 (**https://oreil.ly/lFiWX**). Полный перечень моделей, которые поддерживаются библиотекой Sentence Transformers, доступен онлайн по адресу **https://oreil.ly/Onyuv**, где можно изучить их характеристики и выбрать наиболее подходящую. Однако всегда полезно проверить условия лицензирования выбранной модели, особенно при ее использовании в коммерческих проектах.

При выборе базовой модели необходимо учитывать ряд ключевых параметров: производительность модели (ее способность эффективно решать поставленные задачи), размер моделей эмбеддингов (который влияет на скорость кодирования текста), количество измерений (определяющее объем занимаемой памяти) и юридические аспекты лицензии. Если речь идет о надежном инструменте для начала работы, то MPNet или ее доработанная версия all-mpnet-base-v2 часто становятся оптимальным выбором, поскольку они зарекомендовали себя в различных проектах благодаря своей стабильности и качеству.



Если модель уже была дообучена для решения конкретной задачи, например семантического поиска, тогда ее дообучение для выполнения другой задачи нецелесообразно.

## Набор обучающих данных

Существует множество подходов к структурированию наборов данных, и одним из наиболее популярных является организация в виде триплетов, включающих анкорный (якорный, anchor), положительный (positive) и отрицательный (negative) примеры. В данном случае положительное предложение обозначает вариант, который должен располагаться ближе к анкорному предложению в пространстве эмбеддингов, тогда как отрицательное предложение, наоборот, должно находиться дальше. Например, для дообучения модели с целью улучшения ее способности различать

предложения с отрицанием, обучающий набор данных может быть составлен из триплетов, в которых отрицательное предложение противоречит анкорному и положительному предложениям.

Пример набора данных с эмбеддингами, организованного в виде триплетов для различения отрицания, представлен на рис. 11.2.

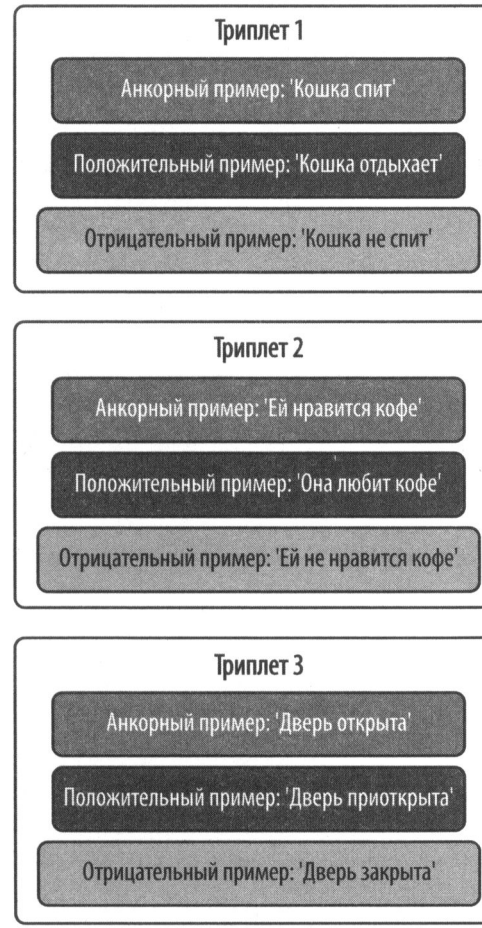



**Рис. 11.2.** Дообучение набора данных для различения отрицания

### Жесткие отрицательные примеры

Процесс заполнения отрицательных примеров для набора данных может быть относительно простым, когда речь идет о простых случаях. Однако в более сложных ситуациях выбор отрицательных примеров становится менее очевидным.

Один из методов заключается в использовании случайных предложений из корпуса данных в качестве отрицательных примеров. Однако для более точного и эффективного дообучения часто используются так называемые *жесткие отри-*

*цательные примеры* (hard negatives). Они представляют собой примеры, которые в определенной степени связаны с анкорным примером, но все же не настолько релевантны, как положительный пример. Простой метод их выбора предполагает использование анкора в качестве поискового запроса, чтобы найти $k$ наиболее подходящих соответствий в корпусе текста с помощью модели эмбеддинга. При этом важно убедиться, что отобранные примеры не являются ложными отрицаниями (false negatives), т. е. не более релевантны или равноценны положительному примеру. Для этого можно установить порог косинусного сходства меньше 0,7 или ограничить выбор диапазоном, например между топ-30 и топ-50 совпадений.

Дополнительные исследования (Moreira et al., **https://oreil.ly/tIs81**) показывают, что проблему ложных отрицаний можно смягчить путем оценки релевантности положительного примера. Один из подходов заключается в установлении порога релевантности для отрицательного примера, который превышает оценку релевантности положительного примера на фиксированное значение либо равняется определенному проценту от этой оценки.

Наборы данных могут включать пары предложений, где каждое из предложений представляет собой определенный тип связи, например: запрос и ответ, проход и сводка, — либо два перефразированных выражения. Конкретные сценарии использования диктуют формат и требования к набору данных. Информацию обо всех доступных способах форматирования можно найти на сайте Sentence Transformers (**https://oreil.ly/geI1M**).

Размеры наборов обучающих данных могут варьироваться от нескольких тысяч примеров до массивов, содержащих миллиарды токенов (**https://oreil.ly/oNI4n**), особенно если задачи требуют адаптации моделей к специфической предметной области.

Важно отметить, что некоторые функции потерь (loss function) строго требуют соблюдения определенного формата набора данных. Далее мы подробно остановимся на функциях потерь и их особенностях.

## Функции потерь

*В главе 4* обсуждались функции потерь, которые используются для обучения LLM. Библиотека Sentence Transformers (**https://oreil.ly/9Qaop**) предоставляет широкий выбор функций потерь, применяемых в обучении моделей эмбеддингов. Рассмотрим некоторые из наиболее распространенных.

Для работы с набором данных, организованным в виде триплетов, можно воспользоваться функцией потерь триплета (triplet loss, **https://oreil.ly/yXHNU**). В таком случае обучающий набор данных представляет собой триплеты вида "анкорный пример — положительный пример — отрицательный пример". Цель использования данной функции — минимизировать расстояние между анкорным и положительным предложениями, одновременно увеличивая расстояние между анкорным и отрицательным предложениями.

Математически потери (loss) рассчитываются следующим образом:

$$\text{loss} = \max(d(a, p) - d(a, n) + \text{margin}, 0).$$

В данном уравнении $d$ обозначает меру расстояния, которая обычно используется в качестве евклидова расстояния. Гиперпараметр margin (зазор) определяет минимальное расстояние, на которое отрицательный пример должен быть дальше от анкорного примера по сравнению с положительным примером. При использовании евклидова расстояния в качестве меры расстояния рекомендуется установить значение параметра margin равным 5. В случае получения неудовлетворительных результатов необходимо будет произвести корректировку значения параметра.

При работе с наборами данных, содержащими пары типа "запрос — ответ", "прохождение — сводка" и подобные сочетания, можно воспользоваться функцией потерь MNRL (Multiple Negatives Ranking Loss — многократное снижение рейтинга с отрицательным результатом, **https://oreil.ly/oNcsQ**).

В наборе данных, состоящем из пар "запрос — ответ" (q1, r1), (q2, r2), ..., (qn, rn), для каждого запроса существует положительная пара, например (q1, r1), и $n - 1$ отрицательных пар, например (q1, r2), (q1, r3) и т. д. Цель функции потерь заключается в минимизации отрицательной логарифмической правдоподобности (log likelihood).



Рекомендуется использовать функцию `CachedMultipleNegativesRankingLoss`, доступную в библиотеке Sentence Transformers. Данный метод позволяет использовать большие размеры пакетов, что способствует повышению производительности.

Таким образом, мы рассмотрели все ключевые компоненты, необходимые для проведения дообучения, и теперь объединим их в общую структуру с помощью класса `SentenceTransformerTrainer`.

```
from datasets import load_dataset
from sentence_transformers import SentenceTransformer, SentenceTransformerTrainer
from sentence_transformers.losses import TripletLoss

model = SentenceTransformer("'all-mpnet-base-v2'")

dataset = load_dataset("csv", data_files="negatives_dataset.csv")

loss = TripletLoss(model)

trainer = SentenceTransformerTrainer(
    model=model,
    train_dataset=dataset
    loss=loss
    )
```

```
trainer.train()
model.save_pretrained("mpnet_finetuned_negatives")
```

Полный код доступен в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).



Важно не допустить переобучения модели. Если вы заметили признаки переобучения, рекомендуется замедлить скорость обучения.

**Упражнение**

Выберите любую LLM и создайте с ее помощью синтетический триплетный набор данных, включающий примерно 8000 примеров. В этом наборе данных отрицательный пример будет представлять собой отрицание положительного примера. Проведите дообучение модели all-mpnet-base-v2 на данном наборе данных. После дообучения проверьте косинусное сходство предложений в примере отрицания, который был представлен ранее в этой главе. Произошли ли какие-либо улучшения?



Исследование Чжоу и соавт. (Zhou et al., **https://oreil.ly/BPdRD**) показало, что в пространстве эмбеддингов косинусное сходство часто недостаточно точно оценивает степень близости между словами с высокой частотой использования. Это связано с распределением таких слов по различным зонам в пространстве эмбеддингов, что приводит к увеличению расстояния между ними и другими словами. В то же время слова с низкой частотой употребления, как правило, демонстрируют более плотную геометрическую концентрацию.

## Эмбеддинги инструкций

До недавнего времени модели эмбеддинга разрабатывались с привязкой к определенной задаче, например семантическому поиску или перефразированию. Однако последние достижения в области машинного обучения связаны с объединением моделей эмбеддингов и концепции обучения следовать инструкциям, о которой говорилось в *главе 6*. Представьте возможность применять одну модель эмбеддинга для генерации разных представлений одного и того же документа, адаптированных под конкретную задачу. Один из таких подходов реализован в модели Instructor. Эта модель предоставляет гибкость в создании эмбеддингов, позволяя указывать во время кодирования такие параметры, как предметная область (домен), тип текста (например, предложение или абзац), задача и сам текст.

Рассмотрим пример использования:

```
!pip install InstructorEmbedding

from InstructorEmbedding import INSTRUCTOR
model = INSTRUCTOR('hkunlp/instructor-large')

customized_embeddings = model.encode(
[['Представить запрос для извлечения вспомогательной документации:',
  'Кто является генеральным директором компании Apple'],
[ 'Представить предложение для извлечения:',
  'Генеральным директором компании Apple является Тим Кук'],
[ 'Представить предложение для извлечения:',
  'Он является генеральным директором, обладающим музыкальными способностями'],
)
```

Авторы модели Instructor предлагают использовать следующий шаблон инструкций:

```
'Представить {domain} {text_type} для {task_objective}:'
```

Здесь `{domain}` обозначает предметную область текста, например юриспруденцию, финансы и подобные категории. Необязательный параметр `{text_type}` указывает на тип текстового блока, например: вопрос, предложение, абзац и т. д. Параметр `{task_objective}` представляет конкретную цель использования эмбеддингов, например: семантический поиск, выявление перефразирования и т. п.

В контексте семантического поиска для запросов рекомендуется использовать инструкцию `'Представить запрос для извлечения вспомогательной документации'`, а для документов — `'Представить предложение для извлечения'`.

Еще один способ применения принципа обучения следовать инструкциям в контексте извлечения информации заключается в извлечении на основе описания, где запрос может представлять собой описание текста, который требуется извлечь, а не создание конкретного примера текста для извлечения. В работе Равфогеля и соавт. (Ravfogel et al., **https://oreil.ly/rp8Q-**) рассматривается *модель поиска на основе описания* (description-based retrieval model), которая, на мой взгляд, является весьма эффективной. Следует отметить, что данная модель имеет двойную настройку кодировщика: для кодирования запроса и документов используются различные модели.

### Упражнение

Выполните кодирование набора данных Википедии, доступного в репозитории книги на GitHub (**https://oreil.ly/llmplaybooks**), с использованием модели Instructor и моделей поиска на основе описания Abstract-Sim-Query (абстрактный симулятор запроса, **https://oreil.ly/zYgAW**) и Abstract-Sim-Sentence (абстрактный

симулятор предложения, **https://oreil.ly/S7iat**). Как эти модели функционируют в задаче ответов на вопросы по сравнению с моделями эмбеддингов, которые использовались ранее?

**Оценка моделей эмбеддинга**

В настоящее время существует большое количество моделей эмбеддинга. Для выбора подходящей модели можно воспользоваться бенчмарком для оценки текстовых эмбеддингов MTEB (Massive Text Embedding Benchmark — масштабный бенчмарк для оценки текстовых эмбеддингов), доступным по адресу **https://oreil.ly/MJ-Di**. MTEB предоставляет всестороннюю оценку моделей, охватывая широкий спектр задач и анализируя одновременно задержку и эффективность выполнения, что позволяет точнее оценить возможные компромиссы между этими параметрами.

Актуальная таблица лидеров, или лидерборд (leaderboard) моделей, с регулярно обновляемой информацией доступна по адресу **https://oreil.ly/aLEPi**. Стоит отметить, что явного лидера по всем задачам нет, и более крупные модели в целом показывают лучшие результаты. При этом разница в производительности между первыми 50 или даже 100 моделями оказывается минимальной. Важно помнить об ограничениях публичных бенчмарков, рассмотренных в *главе 5*. На основании этого следует с осторожностью относиться к рейтингам MTEB и не делать выбор исключительно на их основе. Финальное решение должно учитывать индивидуальные требования конкретного приложения, а также аспекты стоимости, задержки и компромиссов в производительности.

## Оптимизация размера эмбеддингов

Современные приложения часто создают миллиарды эмбеддингов, которые могут иметь тысячи измерений. Если каждое измерение представлено в формате float32, то для его хранения требуется 4 байта памяти. Например, для хранения 100 млн векторов, сгенерированных из модели all-mpnet-base-v2, которая имеет 768 измерений, требуется около 300 Гбайт памяти.

Нередко одно предложение, состоящее из 40 токенов, можно представить вектором размерности 768. Возникает вопрос: действительно ли нужно столько измерений для представления 40 токенов? На практике оказывается, что процесс обучения эмбеддингов зачастую малоэффективен, а наличие большого числа измерений не всегда приносит значимые преимущества.

Чтобы оптимизировать использование памяти и ресурсов для хранения и обработки данных, были разработаны методы усечения (truncation) и квантования (quantization) эмбеддингов. Они позволяют значительно уменьшить объемы хранения и повысить вычислительную эффективность. Такие подходы становятся особенно актуальными в условиях работы с миллионами векторов и более. Рассмотрим некоторые из этих методов подробнее.

## Эмбеддинги "Матрешка"

Эмбеддинги "Матрешка" (Matryoshka embeddings) получили свое название по аналогии с русскими куклами-матрешками, которые представляют собой набор деревянных фигурок, вложенных друг в друга по принципу уменьшения размера (**https://oreil.ly/OC6Yj**). Эти эмбеддинги обучены таким образом, что более ранние измерения вектора содержат более значимую информацию по сравнению с более поздними измерениями. Такой подход позволяет обрезать векторы в зависимости от требований конкретного приложения, учитывая стоимость, задержку или производительность.

Техника, которая применяется для обучения описанным эмбеддингам, называется Matryoshka Representation Learning (MRL), или обучение представлениям в виде матрешки. В MRL предварительно задается набор размеров усечения. Например, если вектор имеет 1024 измерения, то его размеры усечения могут быть определены как 128, 256, 512 и 768. В процессе обучения для каждого из этих размеров усечения вычисляются потери, а также оцениваются потери для всего вектора. После этого все полученные потери суммируются и взвешиваются. В описанном подходе первые 128 измерений вектора учитывают результаты потерь, которые были вычислены на основе первых 128, 256, 512, 768 и 1024 измерений. Таким образом, итоговый результат заключается в том, что начальные размеры вектора начинают кодировать значимую информацию, поскольку процесс обучения направлен на более сложные и разнообразные потери.

Для реализации обучения с помощью MRL используется библиотека Sentence Transformers. Рассмотрим, как данный метод применяется на практике:

```
from sentence_transformers import SentenceTransformer
from sentence_transformers import SentenceTransformerTrainer, losses
from datasets import load_dataset

model = SentenceTransformer("all-mpnet-base-v2")
train_dataset = load_dataset("csv", data_files="finetune_dataset.csv")
loss = losses.MultipleNegativesRankingLoss(model)
loss = losses.MatryoshkaLoss(model, loss, [768, 512, 256, 128]])

trainer = SentenceTransformerTrainer(
    model=model,
    train_dataset=train_dataset,
    loss=loss,
)
trainer.train()
```

В ходе экспериментов исследователь Том Аарсен (Tom Aarsen, **https://oreil.ly/sA5fo**) обнаружил, что даже при уменьшении размера эмбеддинга до 8,3% исходного зна-

чения модель "Матрешка" сохраняет 98,37% первоначальной производительности. Это делает ее эффективным инструментом для работы с большими наборами данных.

Аналогично тому, как можно уменьшить размерность эмбеддингов с помощью метода MRL, можно также сократить количество слоев модели эмбеддинга, что ускоряет вывод. Это достигается путем извлечения эмбеддингов из нижних слоев модели. Для облегчения совмещения высококачественных эмбеддингов нижних слоев с эмбеддингами последнего слоя модели между конечным слоем и каждым из нижних слоев используются потери расхождения K-L. Этот метод, получивший название Espresso Sentence Embeddings, был впервые представлен в работе Ли и соавт. (Li et al., **https://oreil.ly/fzIPD**).

В ходе экспериментов Том Аарсен (**https://oreil.ly/DIoTe**) также установил, что удаление половины слоев приводит к двукратному увеличению скорости при сохранении 85% исходной производительности.

Библиотека Sentence Transformers позволяет комбинировать представления "Матрешка" с уменьшением слоя с помощью функции `Matryoshka2dLoss` (**https://oreil.ly/xzG-a**).

### Упражнение

Загрузите набор данных от Ришаба Мишра (Rishabh Misra), содержащий заголовки новостей (**https://oreil.ly/Tu4XA**). Используйте модель nomic-embed-text-v1.5 компании Nomic AI, которая была обучена с использованием метода MRL (**https://oreil.ly/jALJE**). Выберите один из заголовков в качестве запроса и создайте его эмбеддинг. Сгенерируйте эмбеддинги документов для всех других заголовков и вычислите показатели сходства между запросами и эмбеддингами документов в контрольных точках усечения 1024, 768, 512, 256 и 128.

Проведите анализ ошибок для 25 лучших результатов. В каком измерении происходит заметное падение производительности?

Также запустите пример сценария обучения для `Matryoshka2dLoss`, предоставленного в библиотеке Sentence Transformers (**https://oreil.ly/Bxe5z**), и протестируйте эмбеддинги с различными срезами слоя и размерами.

## Двоичные и целочисленные эмбеддинги

Альтернативой усечению является квантование (quantization). При двоичном и целочисленном квантовании количество векторных размерностей остается неизменным, но каждая размерность представлена меньшим количеством битов. Обычно векторы эмбеддингов представлены в формате float32, что занимает 4 байта памяти на измерение.

При создании предельно сжатых представлений 4 байта могут быть сокращены до 1 бита, что обеспечивает уменьшение объема памяти в 32 раза. Однако такая степень сжатия зачастую сопровождается потерей точности значений размерностей вектора.

Простой способ преобразования четырехбайтового вектора в однобитовый — присвоить значение 1, если исходное значение положительное, и 0, если отрицательное. Важно отметить, что для достижения наилучших результатов может потребоваться выполнить масштабирование. После упаковки этих битов в байты 512-мерный вектор может быть представлен всего в 512 / 8 = 64 байтах, вместо 512 × 4 = 2048 байтов.

Другое преимущество использования бинарных эмбеддингов — вычисление сходства требует только простых побитовых операций, что значительно ускоряет поиск. Однако квантование отрицательно влияет на производительность.

Для квантования эмбеддингов можно воспользоваться библиотекой Sentence Transformers:

```
from sentence_transformers.quantization import quantize_embeddings


model = SentenceTransformer("all-mpnet-base-v2")
embeddings = model.encode(["Я слышал, что в преддверии Хэллоуина лошади ведут себя особенно активно.", "Далматинцы считаются самыми патриотичными собаками.", "Этот ресторан вызывает у меня ностальгические чувства."])
binary_embeddings = quantize_embeddings(embeddings, precision="binary")
```

Функция `quantize_embeddings` также поддерживает квантование int8. В этой схеме 4 байта, представляющие каждое измерение, преобразуются в целое значение, представленное в 1 байте. Целое число может быть либо со знаком, либо без знака, таким образом представляя значения в диапазоне от –127 до 127 или от 0 до 255 соответственно. Процесс преобразования управляется с помощью калибровочного набора данных эмбеддингов, из которого вычисляются минимальное и максимальное значения каждого измерения. Эти значения затем используются в формуле нормализации для преобразования чисел из одного диапазона в другой.



Исследования показали (**https://oreil.ly/Mp3pu**), что с некоторыми моделями бинарные эмбеддинги работают лучше, чем эмбеддинги int8, несмотря на снижение точности. Основная причина этого кроется в особенностях используемого калибровочного набора данных, а также в сложности преобразования значений с плавающей запятой в интервал значений int8.

## Метод Product Quantization

Еще одним перспективным подходом к квантованию данных является метод Product Quantization (PQ, буквально "квантование по продукту" или "квантование произведения", **https://oreil.ly/aJq2C**). В рамках этого метода вектор данных разделяется на фрагменты или чанки (chunks) равного размера, которые затем группируются в кластеры. Количество кластеров определяется на основе количества значений, которые могут быть представлены с помощью квантованного вектора. Например, если требуется квантовать данные в int8, то количество значений будет равно 256, и, следовательно, количество кластеров также будет равно 256. Каждому кластеру

присваивается уникальный идентификатор, который представляет собой число от 0 до 255. Каждый чанк относится к кластеру, центроид которого наиболее близок к данному чанку.

Таким образом, исходный вектор float32 теперь можно представить в виде списка идентификаторов кластеров, соответствующих кластерам, к которым принадлежат части вектора (чанки). Чем больше размер чанка, тем больше сжатие. Так, если разделить вектор на пять чанков, итоговый вектор будет иметь только пять измерений. В отличие от методов int8 и двоичного квантования, метод PQ также уменьшает количество измерений, необходимых для представления вектора. Однако это приводит к снижению производительности.

Выбор метода квантования основывается на определении относительной значимости продуктов по таким критериям, как стоимость, производительность и скорость.



Оптимизация эмбеддингов в целях хранения может привести к снижению производительности. Однако, учитывая наличие значительного количества избыточной информации в корпусе документов, ответы на типовые запросы пользователей часто содержатся в нескольких различных документах. В связи с этим пользователь, вероятно, не заметит значительного ухудшения качества работы системы.

**Упражнение**

Загрузите эмбеддинги Википедии (**https://oreil.ly/OUq5M**), закодированные с помощью модели Cohere. Затем примените метод PQ, установив количество кластеров равным 256. Также можно использовать векторную базу данных Qdrant, которая поддерживает данный метод. Попробуйте использовать различные размеры чанков. В каком случае наблюдается наиболее высокая производительность?

Дополнительно реализуйте функцию оценки сходства (similarity scoring function) для этого метода.

Таким образом, мы разобрали ключевые методы практической реализации извлечения на основе эмбеддингов и теперь перейдем к определению текстовых единиц, которые необходимо интегрировать в соответствующие векторы.

## Чанкинг

Как было отмечено в *разд. "Введение в эмбеддинги" ранее в этой главе*, модели эмбеддинга имеют ограниченную длину контекста, и при увеличении длины текста эффективность сопоставления векторов уменьшается. В связи с этим целесообразно разделять документы на удобные для обработки фрагменты — чанки (chunk — кусок), и представлять каждый чанк в виде одного или нескольких векторов.

Чанк может быть определен как семантически связанная, но необязательно непрерывная часть документа. Средняя длина чанка зависит от контекста, поддерживаемого языковой моделью, и количества чанков, которые модель возвращает в ответ на запрос пользователя (top-*k*). С развитием технологий и увеличением объема моделей допустимая длина чанков постепенно увеличивается.

Каждый чанк можно либо представить одним вектором, либо разбить на более мелкие единицы, каждая из которых будет представлена отдельным вектором. В качестве единицы может выступать предложение, абзац или даже раздел. Обычно чем меньше единица, тем лучше. При тестировании приложения необходимо проверить ожидаемые запросы пользователей на различных уровнях детализации и определить, какой подход наиболее эффективен.

### Парсинг документов

Первым этапом обработки неструктурированных данных является их извлечение. Оно включает в себя выделение текста из документа, разделение его на управляемые фрагменты, связывание метаданных с каждым фрагментом, генерацию эмбеддингов, а также хранение и индексацию данных для обеспечения удобного доступа.

Если задача предполагает использование предложений в качестве базовых единиц текста, тогда эффективным инструментом для их токенизации является токенизатор Punkt из библиотеки NLTK (**https://oreil.ly/bgxrp**). Однако важно понимать, что выделение предложений — это не просто механический процесс. Особенно сложным он становится при работе с текстами узкоспециализированной тематики. Простого разделения на концевые метки, такие как точки, вопросительные знаки и сокращения, зачастую недостаточно, поскольку сокращения могут привести к серьезным ошибкам в результате. Для повышения точности можно обучить токенизатор Punkt на большом массиве целевого текста, адаптировав его к специфике конкретного домена. Также можно вручную задать правила и исключения для обработки. Существуют альтернативные инструменты токенизации предложений, такие как spaCy (**https://oreil.ly/R7mFQ**), Stanza (**https://oreil.ly/xKo43**) и ClarityNLP (**https://oreil.ly/fxvBi**).

В целом эффективный парсинг (parsing) документов представляет собой сложную задачу, которая включает в себя определение границ разделов и подразделов текста, идентификацию таблиц, графиков и изображений, а также работу с различными форматами файлов. На практике ошибки парсинга часто становятся причиной неудачного выполнения задач генерации с дополненным поиском (RAG). Более того, среди всех этапов конвейера NLP-приложений обработка документов требует наибольших усилий и ресурсов. Хотя парсинг может показаться менее значимым аспектом работы над проектами NLP, он является основой для создания качественных продуктов. Игнорирование этого этапа неизбежно приводит к серьезным проблемам, что делает его одним из ключевых компонентов успешной реализации.

Рассмотрим ситуацию, где текст документа разделяется на блоки, представленные эмбеддингами. Для обработки пользовательского запроса выполняется расчет косинусного сходства между его вектором и каждым из векторов документа. После этого извлекаются те фрагменты, или чанки, которые наиболее соответствуют запросу по степени сходства. Такой подход позволяет проводить поиск с меньшей степенью детализации, например на уровне предложений, но при этом модель получает целый фрагмент, содержащий данное предложение. Это обеспечивает достаточный объем контекстной информации, необходимой для работы модели.

**Упражнение**

Создайте токенизатор предложений для набора данных с записями заседаний парламента Канады, доступного в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Какие виды ошибок могут возникнуть при использовании токенизатора? Можно ли использовать правила для их устранения? Попробуйте провести неконтролируемое обучение (unsupervised training) токенизатора Punkt на основе этих данных. Насколько эффективно такое обучение для решения выявленных проблем?

Специалисты в области машинного обучения часто задаются вопросом: как определить оптимальный размер чанка и какие методы *чанкинга* (chunking — разбиение на фрагменты) являются наиболее эффективными? Выбор правильных размеров и границ чанков является ключевой задачей при работе с технологиями извлечения данных на основе эмбеддингов. В данном разделе мы рассмотрим несколько стратегий чанкинга, начиная с наиболее простых.

В базовой реализации метода извлечения на основе эмбеддинга каждый вектор представляет собой отдельный элемент, не связанный с другими элементами. Текст, представленный вектором A, не влияет на текст, представленный вектором B. Чтобы преодолеть эту изолированность, необходимо либо наладить взаимодействие между этими элементами, либо разработать такую стратегию, которая сделает их максимально самодостаточными.

В свете этих целей рассмотрим некоторые стратегии структурирования текста, которые выходят за рамки простого разделения на абзацы или разделы.

## Метод скользящего окна

Представим ситуацию, когда функция, оценивающая схожесть эмбеддингов, возвращает значение единицы для чанка 45, определяя его как наиболее близкий к вектору запроса. Однако в тексте документа, находящемся в чанке 44, содержится важная информация, дополняющая и уточняющая содержание чанка 45. Несмотря на это, векторы из чанка 44 показывают низкий уровень сходства с запросом, из-за чего данный чанк не включается в итоговое извлечение. Решить такую проблему можно путем применения метода скользящего окна (sliding window chunking). Этот

подход позволяет одному текстовому фрагменту одновременно находиться в нескольких чанках, благодаря чему обеспечивается более полное представление соседнего контекста в последовательных блоках текста.

## Чанкинг с учетом метаданных

Для разделения документа на чанки можно использовать любые метаданные, содержащие информацию о границах абзацев, разделов и подразделов. В случае отсутствия метаданных можно применить методы парсинга документов. Для этого можно использовать несколько библиотек, например Unstructured (**https://oreil.ly/CoX46**).

## Чанкинг с учетом компоновки

Чанкинг с учетом компоновки (layout-aware chunking) представляет собой более продвинутый метод чанкинга с учетом метаданных (metadata-aware chunking). В этом методе используются алгоритмы компьютерного зрения, которые анализируют макет документа, включая расположение текстовых блоков, их размер, наличие заголовков и подзаголовков, а также особенности шрифта. Эти метаданные используются для принятия решений о том, как именно следует разделить текстовый контент на фрагменты. Для извлечения информации о компоновке документа могут быть использованы как инструменты с открытым исходным кодом, так и проприетарные решения. Среди таких инструментов можно выделить Amazon Textractor (**https://oreil.ly/fvkiT**), Unstructured (**https://oreil.ly/CoX46**) и специально обученные этому языковые модели, такие как LayoutLMv3 (**https://oreil.ly/Od5fA**).

Например, используя данный подход, можно определить предметный охват подраздела и, соответственно, разместить заголовок подраздела в начале каждого фрагмента текста, относящегося к данному подразделу.

Также можно использовать такие методы, как ColPali, которые основаны на применении моделей компьютерного зрения для непосредственного встраивания страницы или раздела документа и их последующего извлечения. Это позволит полностью исключить необходимость в проведении чанкинга, но в целом может оказаться более затратным.

## Семантический чанкинг

Принцип семантического чанкинга (semantic chunking) заключается в группировке схожей информации в связные чанки. Для формирования таких групп в качестве слабых подсказок можно использовать границы абзаца, но для достижения большей точности возможна интеграция и более продвинутых методов. Одним из них является тематическое разделение документа, где каждый чанк объединяет данные, относящиеся к одной конкретной теме. При этом чанки не обязательно состоят из непрерывного текста, если это не требуется для конкретного приложения. Более продвинутый подход — чанкинг на основе полос Боллинджера (Bollinger bands-based chunking, **https://oreil.ly/1MwK1**), который доступен в экспериментальной реализации в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**).

Семантический чанкинг можно использовать для объединения различных чанков между собой. После назначения и формирования чанков схожие из них можно группировать на основе сходства их эмбеддингов. Это упрощает извлечение связанных чанков, начиная с того, который имеет наибольшее сходство. Каждый чанк не обязательно должен состоять из контента одного и того же документа при условии сохранения связанных с ним метаданных.

Базовая реализация семантического чанкинга доступна в LangChain (**https://oreil.ly/tm8tk**).



Можно провести высокопроизводительный семантический чанкинг с помощью LLM. Однако при работе с большим объемом данных это может привести к значительным затратам. В некоторых случаях можно использовать традиционные методы, такие как регулярные выражения. В Jina AI разработали сложный 50-строчный чанкер (chunker), основанный на регулярных выражениях (**https://oreil.ly/x5UO8**), который можно использовать в качестве отправной точки.

Несмотря на использование указанных методов, проблема эффективного чанкинга по-прежнему актуальна. Рассмотрим практический пример из финансового документа.

> Страница 5: *все числовые значения в документе представлены в миллионах.*
>
> Страница 84: *сумма сделки со связанными сторонами составляет 213,45 долларов США.*

В данном случае транзакция между связанными сторонами составляет 213 млн долларов США. Однако LLM не будет учитывать эту информацию, поскольку текст на странице 5, скорее всего, не является частью текущего чанка.

Сложность заключается в определении границ области действия. Где заканчивается один раздел и начинается следующий? Каков масштаб действия правила на странице 5? Особенно важно понимать, что произойдет, если правило будет пересмотрено или изменено в середине документа. Не все документы имеют четкие визуальные индикаторы или строгую структуру. Разделение на главы, подразделы и абзацы не всегда выполнено идеально.

Эти вопросы остаются нерешенными и составляют значительную часть случаев отказа механизмов RAG.

## Метод позднего чанкинга

Метод позднего чанкинга (late chunking) представляет собой подход к обработке текстов, который позволяет учитывать разнообразные зависимости. Этот метод был разработан компанией Jina AI и обеспечивает более глубокое понимание структуры текста. Как отмечалось ранее, процесс генерации эмбеддингов происходит через объединение (пулинг) векторов, извлеченных из последнего слоя базовой языковой модели.

При наличии доступа к языковым моделям с длинным контекстом, способным обрабатывать целые документы за один ввод, такие модели могут быть использованы в качестве базовых для генерации эмбеддингов. В этом подходе весь текст документа (или его бóльшая часть, соответствующая возможностям модели) вводится в модель, и для каждого токена генерируется индивидуальный вектор. Эти векторы отражают значение токена, учитывая его связь с остальными токенами последовательности, что позволяет учитывать зависимости длинных контекстов, как описано в *главе 4*.

Для извлечения эмбеддингов выполняется операция пулинга на небольших фрагментах входных данных. Границы этих фрагментов могут быть определены с использованием алгоритмов чанкинга. В результате получается несколько эмбеддингов для одного документа, каждый из которых содержит информацию о соответствующем фрагменте входных данных.

**Упражнение**

Для проведения экспериментов со стратегиями чанкинга воспользуйтесь годовым отчетом компании Apple, который называется 10-K (**https://oreil.ly/Vfizz**). Этот документ содержит большое количество разделов, подразделов и подподразделов, а также таблицы и разнообразную информацию, что делает его подходящим для анализа и экспериментов. Разбейте документ на чанки, используя различные стратегии чанкинга, описанные в этой главе, и создайте векторы для каждого из них. Попробуйте сформулировать вопросы, связанные с годовым отчетом, и проанализируйте результаты. Какие стратегии чанкинга оказались наиболее эффективными? Какова оптимальная степень детализации для генерации векторов?

## Векторные базы данных

Векторные базы данных играют ключевую роль в работе с миллионами или даже миллиардами векторов, особенно когда ежедневно требуется создавать и сохранять новые векторы вместе с соответствующими метаданными. Такие базы упрощают управление данными и обеспечивают удобство для различных приложений. Пользователи могут выбирать между локальными решениями, облачными сервисами, платформами с открытым исходным кодом или проприетарными вариантами. Среди популярных примеров выделяются Weviate, Milvus, Pinecone, Chroma, Qdrant и LanceDB. Кроме того, крупные игроки рынка, такие как ElasticSearch, Redis и Postgres, предлагают интеграцию и поддержку функциональности векторных баз данных.

Сегодня функции векторных баз данных все больше пересекаются, поскольку большинство разработок сосредоточено вокруг небольшого числа наиболее востребованных сценариев использования поиска.

Рассмотрим принцип работы векторных баз данных. Одним из наиболее простых вариантов для начала является Chroma. Этот инструмент с открытым исходным кодом позволяет работать как на локальной машине, так и развертывать его в облаке, например, на платформе AWS:

```
!pip install chromadb

import chromadb
chroma_client = chromadb.Client()

collection = chroma_client.create_collection(name="mango_science")
chunks = ['353 вида манго в настоящее время находятся на грани исчезновения', 'Манго выращивают в тропиках']
metadata = [{"тема": "исчезновение", "глава": "2"}, {"тема": "регионы",
"глава": "5"}]
unique_ids = [str(i) for i in range(len(chunks))]

collection.add(
    documents=chunks,
    metadatas=metadata,
    ids=unique_ids
)
results = collection.query(
    query_texts=["Где выращивают манго?"],
    n_results=2,
    where={"chapter": {"$ne": "2"}},
    where_document={"$contains":"выращивают"}
)
```

Большинство векторных баз данных предоставляют следующие функции:

- возможность выполнения приблизительного поиска ближайшего соседа в дополнение к точному поиску для снижения задержки;
- фильтрация данных с использованием метаданных, аналогично предложению `where` в SQL;
- поиск по ключевым словам или с использованием алгоритмов, таких как BM25;
- поддержка логических операций поиска, что позволяет объединять несколько условий поиска с использованием операции `AND` или `OR`;
- возможность обновления или удаления данных из БД в режиме реального времени.

**Многоуровневые эмбеддинги**

В случае если предъявляются строгие требования к производительности извлечения данных, рекомендуется использовать многоуровневые эмбеддинги (multi-level embeddings). Это позволит повысить качество и точность извлечения информации. Эмбеддинги могут быть реализованы на различных уровнях: предложения, абзацы, диалоги, разделы или документы. Эмбеддинги более высокого уровня могут представлять собой краткое содержание текста, а не сам текст.

На каждом уровне можно применять разные модели эмбеддинга, выбирая их в зависимости от степени детализации: чем глубже уровень, тем более сложные и ресурсоемкие инструменты можно использовать.

Выбор подходящего уровня зависит от специфики задачи. Можно начать с анализа верхнего уровня и постепенно углубляться, организуя процесс в виде дерева, или же целенаправленно работать с конкретным уровнем данных.

## Интерпретация эмбеддингов

Какие характеристики текста учитываются при создании эмбеддингов? Почему в пространстве эмбеддингов два предложения иногда оказываются ближе или дальше друг от друга, чем ожидается? Можно ли определить, что представляет собой каждое измерение вектора эмбеддинга?

Одной из ключевых проблем поиска, основанного на использовании эмбеддингов, является ограниченная интерпретируемость решений при ранжировании результатов. Это отличает его от традиционных подходов. В настоящее время активно ведутся исследования в области повышения интерпретируемости нейронных сетей, больших языковых моделей и эмбеддингов. *В главе 5* были рассмотрены некоторые подходы к интерпретации, применимые для анализа LLM. В данном разделе мы подробно остановимся на интерпретируемости эмбеддингов. Понимание структуры и особенностей, присущих пространству эмбеддингов, позволяет использовать эту информацию для их настройки и управления в соответствии с конкретными задачами. Одним из перспективных подходов к обеспечению интерпретируемости является применение разреженных автоэнкодеров (SAE). Рассмотрим подробно, что они собой представляют, как обучаются и каким образом способствуют повышению интерпретируемости.

Языковая модель способна изучить миллионы функций, но для конкретного ввода активизируются и становятся релевантными лишь некоторые из них. Такое явление называется разреженностью (sparsity). Хотя модели обучаются множеству функций, количество измерений в векторе эмбеддинга ограничено. Из-за этого каждое измерение влияет на разные элементы, которые потенциально могут конфликтовать между собой. Однако, если обучить на этих эмбеддингах разреженный автоэнкодер (**https://oreil.ly/oiXb7**), можно будет выделить независимые и интерпретируемые характеристики.

В рамках проекта Prism Линус Ли (Linus Lee, **https://oreil.ly/efzz1**) применяет метод SAE для анализа характеристик модели эмбеддинга, построенной на основе T5. Среди выявленных особенностей можно отметить:

- присутствие отрицания,
- выражение возможностей или предположений,
- концепцию занятости и труда,
- использование притяжательных синтаксических конструкций в начале предложений.

Более подробную информацию о характеристиках можно найти в блоге Линуса Ли (**https://oreil.ly/efzz1**).

## Резюме

В этой главе мы рассмотрели понятие эмбеддингов, их основные компоненты и методы генерации. Также мы изучили методы дообучения эмбеддингов на собственных данных и научились определять гранулярность эмбеддингов с помощью нескольких методов чанкинга. В завершение мы рассмотрели способы визуализации и интерпретации эмбеддингов.

В следующей главе мы рассмотрим парадигму RAG, которая является наиболее востребованной областью применения эмбеддингов в настоящее время. Мы подробно остановимся на ключевых этапах типичного рабочего процесса в рамках RAG, обсудим технические аспекты разработки подобных приложений и рассмотрим различные компромиссы, которые необходимо учитывать в процессе разработки.

# ГЛАВА 12

# Генерация с дополненным поиском

*В главе 10* мы рассмотрели, как значительно расширить возможности LLM за счет взаимодействия с внешними данными и программным обеспечением. *В главе 11* мы обсудили концепцию извлечения данных на основе эмбеддингов, которая является базовым подходом к получению релевантной информации из хранилищ данных в ответ на запросы. Используя эти принципы, теперь можно углубиться в практическую реализацию метода дополнения LLM сведениями из внешних источников, который известен как *генерация с дополненным поиском* (retrieval-augmented generation, RAG).

В этой главе мы подробно рассмотрим конвейер RAG и разберем каждый этап, составляющий типичный рабочий процесс приложения RAG. Особое внимание будет уделено различным аспектам использования RAG: какие данные можно извлекать, а также как и когда лучше всего это делать. Мы также рассмотрим, как этот метод может быть полезен не только на этапе вывода модели, но и в процессе ее обучения и дообучения. В заключение мы сравним RAG с другими подходами и обсудим сценарии, в которых этот конвейер оказывается более эффективным, а также случаи, когда он уступает альтернативным решениям.

## Необходимость использования RAG

Как отмечалось в *главе 10*, RAG является общим термином для обозначения подходов, направленных на интеграцию внешних источников данных с целью расширения функциональных возможностей больших языковых моделей.

Существует несколько причин, почему использование RAG может быть актуальным.

- Модели LLM требуют доступа к частным (проприетарным) данным, которые не были включены в наборы предобучающих данных. В таких случаях применение RAG оказывается более удобным и эффективным решением, по сравнению с предобучением LLM на этих данных.
- Для снижения вероятности возникновения галлюцинаций важно, чтобы вместо обращения к собственным знаниям, LLM использовали данные, предоставляе-

мые через механизм поиска. Технология RAG помогает реализовать этот подход, обеспечивая более точную привязку ответов LLM к проверенным источникам информации.

- Также необходимо, чтобы LLM могли отвечать на вопросы о недавних событиях и новых концепциях, появившихся после завершения этапа предобучения модели. Хотя существуют технологии редактирования памяти, такие как MEMIT (**https://oreil.ly/kxI3j**), позволяющие вносить новые знания в параметры модели, они пока не отличаются высокой надежностью и масштабируемостью. Как отмечалось в *главе 7*, регулярное дообучение модели для актуализации ее знаний требует значительных финансовых затрат и связано с рисками.
- Кроме того, важно, чтобы LLM могли обрабатывать запросы, связанные с "длинным хвостом" (long-tail entities) — редкими сущностями, которые встречаются в наборах предобучающих данных лишь эпизодически.

### Решение проблемы "длинного хвоста" в LLM

Для обучения LLM часто требуется значительное количество примеров, чтобы надежно запомнить определенный факт. Однако процесс запоминания носит вероятностный характер, что делает невозможным точное предсказание того, сколько именно примеров должна обработать модель во время обучения, чтобы усвоить информацию. Такая выборочная неэффективность приводит к трудностям в ответах на вопросы, связанные с объектами и концепциями, которые редко встречаются в обучающих данных. Например, исследование Кандпала и соавт. (Kandpal et al., **https://oreil.ly/dqXV4**) демонстрирует, что точность модели BLOOM-176B в ответах на вопросы составляет лишь 25% в случаях, когда соответствующие документы появляются всего 10 раз в предобучающем наборе данных. Для сравнения, точность возрастает до 55%, если эти же документы встречаются 10 000 раз.

Авторы также показали, что крупным LLM требуется меньше примеров для усвоения отдельного факта. Тем не менее даже при этом остается значительное количество концепций с "длинным хвостом", которые такие модели не способны запомнить. Соотношение между размером LLM и ее способностью к запоминанию имеет логарифмически линейную природу. Это подразумевает, что для эффективной работы с данными сложной структуры и задачами с "длинным хвостом" модель должна иметь масштабы порядка квадрильонов параметров.

Существует несколько подходов к повышению способности LLM к запоминанию. Один из них — увеличение количества эпох обучения либо расширение обучающего набора данных за счет включения большего числа примеров, представляющих ключевые концепции и факты, которые необходимо сохранить. Также можно изменить цель обучения, увеличив веса при обучении потерям для токенов, представляющих факты.

Другим способом улучшить запоминание модели является обучение по учебной программе (curriculum learning), рассмотренное в *главе 2*. Исследование Ягельски

и соавт. (Jagielski et al., **https://oreil.ly/5BP_V**) показало, что информация, представленная в начале процесса обучения, имеет тенденцию быстрее забываться. Таким образом, изменяя порядок предоставления примеров, можно увеличить вероятность их долгосрочного запоминания, особенно для данных, которым требуется особое внимание.

Еще один подход к оптимизации результатов по концепциям с "длинным хвостом" заключается в применении метода RAG, который будет подробно рассмотрен в следующих разделах главы.

**Упражнение**

Практически все модели LLM обучаются на Википедии, которая считается высококачественным универсальным набором данных. В Википедии представлены страницы даже о малоизвестных личностях, например об игроках низших футбольных лиг. Проведите эксперимент: задайте LLM вопросы о таких сравнительно неизвестных людях, используя информацию, доступную на страницах Википедии. Проверьте, насколько точно модели разных размеров смогут ответить на эти вопросы, опираясь на доступные данные. Затем повторите эксперимент с более известными личностями, чью популярность можно оценить по размеру статей о них в Википедии. Как размер модели повлияет на ее способность давать правильные ответы на такие вопросы?

## Типичные сценарии использования RAG

После того как мы разобрались, *для чего нужна* техника RAG, остановимся подробнее *на области ее применения*. Можно выделить четыре наиболее востребованных сценария.

*Извлечение внешних знаний (retrieving external knowledge)*

Этот случай считается самым распространенным и доказал свою эффективность в практическом применении. Как упоминалось ранее в этой главе, с помощью RAG можно компенсировать нехватку знаний у LLM или минимизировать вероятность возникновения галлюцинаций.

*Извлечение истории контекста (retrieving context history)*

Модели имеют ограниченное контекстное окно, но при этом часто сталкиваются с задачей обработки запросов, требующих большего объема данных, чем может вместить их контекстное окно. Чтобы вести более продолжительные беседы и учитывать важные детали, можно использовать стратегию извлечения ключевых частей истории диалога или сеансового контекста, обеспечивая модели доступ к необходимой информации.

*Извлечение обучающих примеров из контекста (retrieving in-context training examples)*

Обучение с использованием нескольких примеров представляет собой эффективный способ настройки LLM на выполнение задач, демонстрируя правильное соотношение входных данных и ожидаемого результата. Этот процесс можно оптимизировать, динамически подбирая примеры, которые релевантны для текущего ввода. Несколько примеров можно извлечь из хранилища примеров обучающих данных на этапе вывода.

*Извлечение информации об инструментах (retrieving tool-related information)*

Как отмечалось в *главе 10*, в рамках своего рабочего процесса LLM могут использовать программные инструменты. Перечень доступных инструментов и их характеристики находятся в хранилище инструментов (tool store). С помощью механизма извлечения LLM может выбирать инструмент, наиболее подходящий для выполнения конкретной задачи. Информация об инструментах может включать, в частности, документацию API.

## Определение момента для выполнения извлечения информации

На каждом этапе агентского рабочего процесса LLM может решать поставленную задачу, прибегая к одному из следующих способов:

- используя свои встроенные возможности;
- обращаясь к одному из доступных хранилищ данных;
- используя один из доступных программных инструментов.

Существуют задачи, которые LLM может решить, используя свою параметрическую память. Однако в ряде случаев могут потребоваться данные, которые содержатся в одном или нескольких хранилищах данных. Стоит ли в таких случаях использовать RAG по умолчанию, учитывая все преимущества этого метода?

Как было отмечено ранее, большие языковые модели сталкиваются с трудностями при обработке "длинного хвоста", и RAG представляет собой эффективную технику для поиска ответов на вопросы, касающиеся редко встречающихся сущностей. Однако исследование, проведенное Малленом и соавт. (Mallen et al., **https://oreil.ly/MF7Y1**), показало, что в случае запросов о более популярных сущностях LLM иногда предоставляют более точные ответы, чем RAG. Это обусловлено ограничениями поисковых моделей, которые могут извлекать нерелевантные или некорректные данные, способные запутать языковую модель.

Для каждого запроса можно динамически определить, следует ли использовать механизм извлечения или положиться на параметрическую память LLM.

Выбор подхода определяется следующими критериями.

- Насколько запрос связан с широко известной или часто встречающейся сущностью? Например, модель скорее запомнит дату рождения Тейлор Свифт[1], чем биографические детали малоизвестного барабанщика из локальной группы, статья в Википедии о котором остается незавершенной.
- Имеются ли временны́е ограничения, т. е. требуется ли для выполнения запроса информация, которая появилась после даты завершения обучения модели?
- Проходила ли модель процесс непрерывного предобучения или обучения памяти (*см. главу 7*), и затрагивает ли запрос те концепции, которые были включены в последующие этапы обучения?

При использовании моделей LLM для ответов на вопросы общего характера команда Маллена предложила использовать такие источники, как Википедия, в качестве метрики псевдопопулярности для сущностей. Если количество сущностей во входных данных превышает пороговое значение, LLM может самостоятельно ответить на вопрос без использования RAG. Следует отметить, что пороговое значение может меняться в зависимости от версии LLM. Эта стратегия работает только при условии хорошего понимания наборов данных, на которых была обучена LLM.

Динамический подход к определению момента извлечения данных может способствовать оптимизации задержки и времени отклика модели, поскольку активация конвейера RAG приводит к дополнительным издержкам при обработке запроса.



Динамическое извлечение данных целесообразно применять в основном при работе с крупными LLM. Для моделей меньшего размера (7B и ниже) рекомендуется использовать RAG, а не полагаться на внутреннюю память LLM.

## Конвейер RAG

В типичном приложении RAG используется фреймворк извлечения-считывания (retrieve-read framework), рассмотренный в *главе 11*. При обработке запроса поисковая модель идентифицирует релевантные запросу документы, которые затем передаются в LLM в качестве контекста. Контекст дополняет внутренние возможности модели и помогает ей более точно сформулировать ответ. На пути к производственной реализации RAG часто требуется доработка и добавление множества дополнительных механизмов. Одной из таких доработок является расширение фреймворка извлечения-считывания за счет включения дополнительных этапов. Рабочий процесс может включать несколько последовательных шагов, таких как перезапись, извлечение, считывание, уточнение, вставка и генерация ответа (rewrite-retrieve-read-refine-insert-generate). Некоторые из этих этапов представляют собой многоуровневые процессы, включающие несколько подзадач. В этой главе мы подробно рассмотрим каждый из них.

[1] Тейлор Свифт (Taylor Swift) — американская певица и автор песен. — *Прим. пер.*

Различные этапы конвейера RAG и соответствующие компоненты показаны на рис. 12.1.

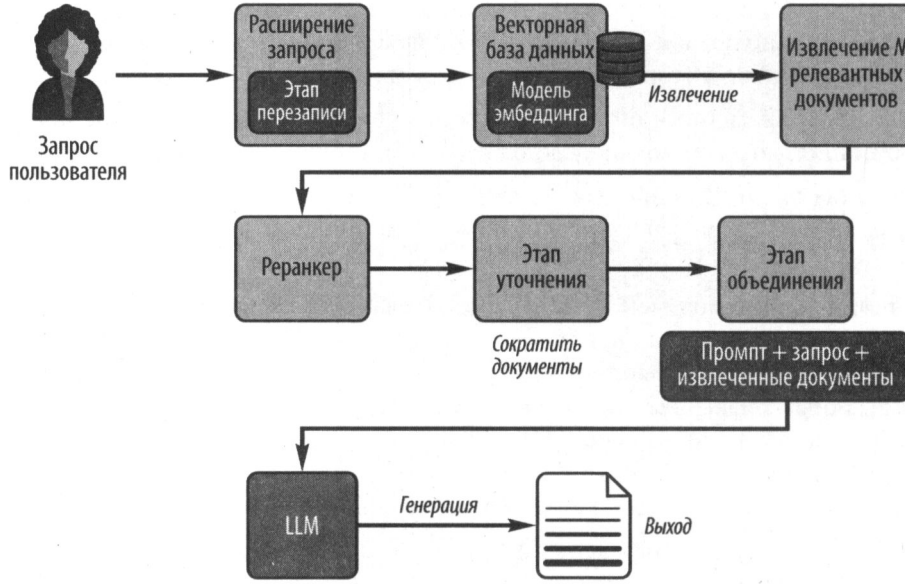



**Рис. 12.1.** Конвейер RAG



В книге под запросами понимаются запросы пользователя или LLM для получения данных, а под документами — единицы текста, извлеченные из хранилища данных.

Рассмотрим это на примере. Предположим, существует приложение RAG, предназначенное для предоставления информации о политической и парламентской деятельности в Канаде. Данное приложение имеет доступ к базе знаний, включающей стенограммы заседаний парламента. Предположим, что данные в этой базе представлены с использованием методов, описанных в *главе 11*.

В процессе формирования запроса пользователь может переформулировать его перед отправкой на ретривер (retriever — механизм извлечения). Этот процесс, известный как расширение запроса (query expansion), особенно полезен в случаях, когда существует несоответствие между лексикой запроса и лексикой документа. Пользователь может использовать в запросе термины, отличные от тех, которые используются в документах. Перефразирование запроса помогает преодолеть это несоответствие и повысить вероятность получения наиболее релевантных документов. Этот этап называется этапом *перезаписи* (rewrite).

Затем на этапе *извлечения* (retrieve) применяется поисковая модель для извлечения документов, соответствующих запросу. *В главе 11* мы рассматривали извлечение на основе эмбеддинга — популярную парадигму поиска в эпоху LLM. Стадия извлечения может представлять собой сложный многоэтапный процесс.

Извлечение может осуществляться в обширном пространстве документа. В этом случае использование более сложных поисковых моделей становится невозможным с точки зрения вычислительных ресурсов. Поэтому извлечение обычно выполняется в два этапа. На первом этапе применяются более быстрые методы (как правило, на основе эмбеддинга) для получения списка потенциально релевантных документов (оптимизация отзыва). На втором этапе происходит повторное ранжирование извлеченного списка на основе релевантности (оптимизация точности). В результате ранжированные по top-$k$ документы передаются в LLM. Этот этап называется повторным ранжированием, или *реранкингом* (re-ranking).

После определения top-$k$ наиболее релевантных запросу документов их необходимо передать в LLM. Однако из-за ограничений контекстного окна эти документы часто требуют сокращения. Кроме того, их можно перефразировать таким образом, чтобы LLM могла более эффективно использовать предоставленный контекст при формировании ответа. Этот процесс выполняется на этапе *уточнения* (refine).

Затем выходные данные этапа уточнения подаются в LLM. По умолчанию все документы объединяются в один промпт. Однако их также можно передавать по отдельности, а затем объединять результаты. Порядок документов в промпте также может иметь значение. Существует несколько методов подачи контекста в LLM, и выбор конкретного метода зависит от конкретной задачи. Этот этап называется этапом *вставки* (insert).

На этапе *генерации* (generate) LLM считывает запрос и контекст, представленные в промпте, после чего формирует ответ. Генерация может быть однократной или чередоваться с процессом извлечения информации: модель генерирует несколько токенов, затем обращается к поисковой модели для получения дополнительных данных, генерирует еще несколько токенов и снова обращается к поисковой модели и т. д.

Выходные данные каждого этапа могут быть проверены на качество и, при необходимости, скорректированы на этапе *верификации* (verification). Для проверки могут использоваться эвристические методы или методы на основе ИИ.

В данном примере запрос был сформирован пользователем. Однако в контексте агентских рабочих процессов запрос может быть сгенерирован LLM. Агент может в любой момент определить, что ему необходимо получить данные для выполнения задачи, что приводит к активации описанного конвейера.

Помимо этапов извлечения и генерации, остальные этапы конвейера являются опциональными, и их активация зависит от соотношения производительности и задержки.



В данном примере рассматривается применение RAG на этапе вывода. Кроме того, технику RAG можно использовать на этапе предобучения или дообучения модели, что будет подробно рассмотрено в этой главе.

Рассмотрим каждый этап конвейера более подробно.

## Перезапись

В ряде случаев после формирования запроса может возникнуть необходимость его перезаписи для более эффективного поиска. Процесс перезаписи зависит от используемых поисковых моделей. Как отмечалось выше, между запросом и документом может возникнуть несоответствие из-за различий в лексике, формулировках и семантике.

Например, рассмотрим запрос: "Какие политики выражали недовольство по поводу несбалансированности бюджета?"

В хранилище данных содержится следующий текст: "Сенатор Пакстон выразил обеспокоенность значительным бюджетным дефицитом".

Если использовать традиционные методы поиска, которые ориентированы на ключевые слова, то этот текст может быть признан нерелевантным. Применение методов, основанных на эмбеддинге, позволяет частично решить эту проблему, т. к. эмбеддинги схожих предложений располагаются ближе друг к другу в пространстве эмбеддингов. Тем не менее это не позволяет полностью решить проблему.



В случае если запрос пользователя содержит инструкцию, например: "Какие политики выражали недовольство по поводу несбалансированности бюджета? Предоставь результаты в виде таблицы", необходимо перед отправкой в конвейер извлечения разделить запрос на две составляющие: собственно запрос и сопровождающие его инструкции. Это можно сделать с помощью промптинга, используя LLM, такие как CoT (*см. главу 5*), ReAct (*см. главу 10*).

Для систем, использующих традиционные методы извлечения данных, перезапись запроса обычно осуществляется с использованием методов расширения запроса, в рамках которых запрос дополняется ключевыми словами, аналогичными исходному запросу. Основные методы расширения запросов включают добавление в запрос синонимов ключевых слов и другой тематической информации.

Одним из эффективных методов расширения поискового запроса является метод обратной связи по псевдорелевантности (pseudo-relevance feedback, PRF). В рамках этого метода исходный запрос используется для поиска документов, из которых извлекаются характерные термины. Затем эти термины добавляются к исходному запросу.

Рассмотрим, как метод PRF может быть полезен при ответе на запрос: "Какие политики высказывали недовольство по поводу несбалансированности бюджета?" Сначала воспользуемся методом поиска, например BM25 (который будет подробно описан позже в этой главе), чтобы получить список из $k$ потенциально релевантных документов. Затем при помощи таких подходов, как анализ частоты терминов или более продвинутым методом Tf-IDf (**https://oreil.ly/5be9z**), из найденных документов извлекаются наиболее значимые термины. В данном примере это такие термины, как "бюджетная политика", "дефицит", "ошибки в финансовом менеджменте" и

"бюджетные реформы". После интеграции этих ключевых фраз в исходный запрос получаем:

> "Сенатор Пакстон выразил обеспокоенность значительным бюджетным дефицитом" успешно извлечено.

В последние годы наблюдается рост популярности методов расширения запросов, основанных на использовании больших языковых моделей. Среди таких методов можно выделить Query2doc (**https://oreil.ly/BDJAN**) и Hypothetical Document Embeddings (HyDE, дословно "эмбеддинги гипотетических документов", **https://oreil.ly/tZyUV**).

Принцип работы расширения запросов с использованием LLM заключается в следующем: запрос передается в LLM, после чего модель генерирует гипотетический документ, содержащий ответ на запрос.

> *Prompt*: For the provided query, generate a hypothetical document in 2 sentences or less that answers the query. Assume the documents are transcripts of Canadian parliamentary proceedings, containing verbatim statements of representatives in the House and Senate.
>
> *Query*: Which politicians have complained about the budget not being balanced?

---

> *Промпт*: Создай гипотетический документ, который отвечает на запрос, используя не более двух предложений. Предположим, что это стенограммы парламентских слушаний в Канаде, включающие заявления представителей как в палате, так и в сенате.
>
> *Вопрос*: Какие политики выражали недовольство по поводу несбалансированности бюджета?

В промпте содержатся следующие данные:

- инструкции по созданию гипотетического документа;
- требование к ограничению длины создаваемого документа;
- описание корпуса документов, включающего соответствующие документы;
- исходный запрос.

Гипотетический документ, который генерируется LLM, имеет следующий вид:

> *LLM*: "Honorable Senator Mark Wellington stated, *This government's failure to balance the budget is unacceptable and jeopardizes our economic future.* Similarly, MP Emily Fraser remarked, *We cannot continue on this path of reckless spending without addressing the deficit—it's time for real fiscal responsibility.*"

---

> *LLM*: "Достопочтенный сенатор Марк Веллингтон заявил, что *неспособность правительства сбалансировать бюджет является неприемлемой и ставит под угрозу экономическое будущее страны.* Депутат Эмили Фрейзер также отметила, что *мы не можем продолжать безрассудно тратить средства без решения проблемы дефицита. Пришло время проявить финансовую ответственность*".

Использование данного гипотетического документа в качестве запроса позволит получить предложение "Сенатор Пакстон выразил обеспокоенность значительным бюджетным дефицитом" с более высоким показателем сходства. Несмотря на то

что сам документ малоправдоподобен и выглядит скорее абсурдно (в Канаде, например, нет сенаторов или депутатов с именами Марк Веллингтон и Эмили Фрейзер), его содержание включает множество словосочетаний и семантических конструкций, характерных для речи реального политика. Если использовать данный документ в качестве запроса (возможно, объединив его с исходным запросом), то вероятность его семантического сходства с реальными высказываниями политиков на данную тему будет выше, чем при использовании только исходного запроса.

Длина гипотетического документа может быть сопоставима с обычной длиной блока поиска. Для создания подобных документов целесообразно использовать меньшие LLM, поскольку в условиях, где фактичность играет второстепенную роль, их возможностей зачастую достаточно. Однако стоит учитывать, что такие модели уступают более крупным решениям в генерации качественных гипотетических материалов, поэтому возникает необходимость находить баланс между функциональностью и качеством. Инструменты вроде LangChain и LlamaIndex предоставляют средства для реализации переписывания запросов на основе гипотетических документов. Если модель проходила предобучение или дообучение на корпусе данных, содержащем релевантную информацию, то включение в промпт описания корпуса, как демонстрируется в примере, может значительно повысить вероятность того, что созданный документ будет соответствовать необходимым требованиям — от структурных особенностей до формата и языковой стилистики.



Одна из возможных проблем при использовании методов перезаписи запросов — риск дрейфа темы (topic drift). Это особенно заметно в гипотетических документах, где после первых нескольких токенов текст может отклоняться в сторону нерелевантных тем. Частичным решением может быть увеличение веса смещения логитов для токенов, содержащихся в исходном запросе. Однако стоит отметить, что методы PRF также не застрахованы от подобного дрейфа.

Можно также комбинировать методы PRF с гипотетическими документами. Вместо традиционного подхода, предполагающего создание гипотетических документов для замены или расширения запроса, такая комбинация позволяет использовать эти документы для извлечения ключевых слов, которые затем можно добавить к исходному запросу. Например, в работе Ли и соавт. (Li et al., **https://oreil.ly/cOnMs**) описана методика под названием query2document2keyword. В ее рамках LLM сначала генерируют гипотетический документ на основе исходного запроса, как это делается в методе HyDE. Затем модель извлекает из документа ключевые слова.

Ключевые слова, полученные таким способом, можно улучшить еще больше с помощью дополнительного этапа фильтрации. Авторы рекомендуют использовать метод самосогласованности, подробно рассмотренный в *главе 5*. Этот подход подразумевает многократное повторение процесса генерации ключевых слов, после чего выбираются наиболее часто встречающиеся ключевые слова на основе статистики присутствия во всех итерациях.

Существует альтернативный метод интеграции традиционного извлечения с перезаписью запроса с использованием LLM. Он заключается в следующем: сначала извлекаются top-*k* документов на начальном этапе извлечения, затем с помощью LLM создаются значимые ключевые слова из этих документов и добавляются в запрос.

### Упражнение

Оцените эффективность работы с гипотетическими документами для небольших языковых моделей, таких как Gemma 2B и Llama 2B, на примере данных о заседаниях парламента Канады, доступных в репозитории книги на GitHub, (**https://oreil.ly/llm-playbooks**). Аналогичным образом попробуйте увеличить размер моделей и проанализируйте, как это повлияет на производительность. Какое влияние оказывает интеграция гипотетических документов на общую задержку системы?

До сих пор мы рассматривали методы, направленные на устранение проблемы несоответствия между запросом и документом путем адаптации запроса к пространству документа. Однако существует альтернативный подход, который заключается в преобразовании представления самих документов таким образом, чтобы они лучше соответствовали пространству запросов. Среди примеров такого подхода можно выделить технологии doc2query (**https://oreil.ly/CGUtP**) и методы контекстного извлечения (**https://oreil.ly/ZJuIu**). Хотя техника перезаписи требует значительных начальных затрат, особенно при обработке данных из крупных хранилищ, она может существенно снизить задержки на этапе вывода, поскольку минимизирует необходимость изменения запросов или полностью устраняет ее. С другой стороны, методы, связанные с изменением запросов, отличаются простотой внедрения и интеграции в системы на основе модели RAG.

Одной из специфических форм изменения запросов является их декомпозиция (query decomposition). В случае сложных запросов, возникающих в агентских системах, можно использовать LLM для разделения основной задачи на несколько более простых запросов, которые затем выполняются последовательно или параллельно, в зависимости от способа декомпозиции. Тема декомпозиции запросов подробно рассматривается в *главе 10*.



Если исходные данные представлены в структурированном виде, например, в виде базы данных, то запрос необходимо преобразовать в SQL-запрос или его аналог, как описано в *главе 10*.

Таким образом, мы рассмотрели этап перезаписи запроса и переходим к этапу извлечения данных.

## Извлечение

Этап извлечения данных является ключевым элементом конвейера RAG. От качества выполнения поиска на этом этапе зависит эффективность всех приложений RAG. Даже самая передовая языковая модель не сможет обеспечить корректные результаты, если не будут получены релевантные документы. Поэтому на данном этапе необходимо сосредоточиться на повышении эффективности вызова.

На сегодняшний день большой популярностью пользуется поиск на основе эмбеддингов, который подробно описан в *главе 11*. Однако также могут быть использованы традиционные методы поиска информации. Выбор метода напрямую зависит от характера предполагаемых запросов: могут ли они быть обработаны на основе ключевых слов или совпадения регулярных выражений? Также необходимо учитывать степень несоответствия лексики между запросами и документами, требования к задержке обработки, вычислительным ресурсам и общей производительности системы.



Область исследований, связанных с информационным поиском (IR), уже давно является предметом изучения. В настоящее время, когда поисковые технологии играют важную роль в сфере обработки естественного языка (NLP), наблюдается тенденция к разработке новых решений, а не к использованию существующих идей в области IR. Для более глубокого понимания исследований в области информационного поиска рекомендуется ознакомиться с публикациями ведущих конференций, таких как SIGIR, ECIR, TREC и др.

### Необъяснимая эффективность BM25

Несмотря на развитие сложных техник глубокого машинного обучения (deep learning) для задач извлечения, методы, основанные на сопоставлении ключевых слов или вероятностном поиске, такие как BM25, остаются чрезвычайно сильной базисной моделью. Более того, при сочетании с перезаписью запросов или документов BM25 вполне может быть *подходящим решением* для реализации конкретного приложения.

Среди других традиционных подходов, поддерживаемых Apache Lucene и ElasticSearch, выделяются такие метрики, как Tf-IDF (term frequency-inverse document frequency — частота термина в документе в сравнении с обратной частотой документа), DFI (divergence from independence — отклонение от независимости), DFR (divergence from randomness — отклонение от случайности), IB (information based — на основе информации), а также сходства Дирихле (Dirichlet similarity) и Елинека — Мерсера (Jelinek–Mercer similarity). Каждая метрика имеет несколько настраиваемых параметров. Более подробную информацию об этих методах и о том, как выбирать значения параметров, можно найти в видео "Настройка базовой оценки" ("Tweaking the Base Score", **https://oreil.ly/zyke4**).

Методы извлечения на основе эмбеддингов не всегда способны эффективно находить все документы, содержащие заданное слово или выражение. В связи с этим часто используется комбинированный подход, объединяющий поиск по ключевым словам и методы эмбеддинга. Такой подход называется *гибридным поиском* (hybrid search). Результаты обоих методов объединяются и передаются на следующий этап конвейера извлечения. В настоящее время большинство векторных баз данных поддерживают гибридный поиск в той или иной форме.

Процесс извлечения с использованием гибридного поиска представлен на рис. 12.2.




**Рис. 12.2.** Гибридный поиск

Для повышения эффективности поиска рекомендуется использовать фильтры метаданных. Чем больше метаданных будет собрано на этапе представления и хранения данных, тем более точные результаты поиска можно ожидать. Например, если заранее смоделировать разделы хранилища данных, можно ограничить результаты поиска подмножеством этих разделов с помощью фильтров. Эти фильтры могут быть реализованы с использованием как жестко закодированного набора правил, так и технологий больших языковых моделей.

Далее мы подробно остановимся на последних достижениях в области поисковых систем.

## Генеративное извлечение

Рассмотрим ситуацию, когда LLM может самостоятельно определять релевантные документы для ответа на запрос, что позволяет отказаться от традиционных методов извлечения информации. Такой подход называется генеративным извлечением (generative retrieval).

Генеративное извлечение осуществляется путем присвоения документам уникальных идентификаторов (docID) и обучения модели связывать документы с их docID.

Один документ может быть связан с одним или несколькими идентификаторами. Типичные идентификаторы docID могут включать в себя следующее.

*Отдельные токены (single tokens)*

Каждый документ в словаре может быть представлен отдельным токеном. Это означает, что при генерации текста модель должна выводить всего один токен для каждого документа, который требуется получить. В исследовании Прадипа и соавт. (Pradeep et al., **https://oreil.ly/7JYOM**) показано применение модели T5, где словарь кодера сохраняет стандартный формат T5, а словарь декодера содержит уникальные идентификаторы документов docID. Такой метод оказывается эффективным только при работе с небольшим количеством документов в корпусе.

*Токены префикса/подмножества (prefix/subset tokens)*

Исследование Тай и соавт. (Tay et al., **https://oreil.ly/1p1C8**) показало, что в качестве идентификатора документа docID могут быть использованы первые 64 токена документа. В то же время Ванг и соавт. (Wang et al., **https://oreil.ly/lg3g3**) применили подход, заключающийся в выборе 64 случайно расположенных смежных токенов из текста документа.

*Токены кластера (cluster tokens)*

Альтернативным методом может быть использование иерархической кластеризации корпуса документа с учетом его семантического содержания, например с помощью эмбеддингов. В таком случае docID формируется путем объединения идентификаторов кластеров, полученных на каждом уровне кластеризации.

*Токены характерных ключевых слов (salient keyword tokens)*

Идентификаторы документов также могут базироваться на характерных ключевых словах, отражающих основные темы, рассматриваемые в документе. Например, материал об архитектуре трансформера может быть представлен идентификатором "transformer_selfattention_architecture".

Один из подходов к обучению больших языковых моделей устанавливать связь между документами и их уникальными идентификаторами docID заключается в дообучении модели. Этот метод известен как индексирование на основе обучения (training-based indexing). Однако дообучение требует значительных вычислительных ресурсов и становится неэффективным, если новые документы регулярно добавляются в корпус.

В исследовании Аскари и соавт. (Askari et al., **https://oreil.ly/K5TAB**) предложен альтернативный подход, основанный на использовании методов обучения с несколькими примерами, что позволяет создать генеративную поисковую систему без необходимости переобучать модель. Сначала для каждого документа в корпусе с помощью языковой модели создаются псевдозапросы (pseudo query) — искусственно сгенерированные запросы, ответы на которые содержатся в соответствующем документе. Затем эти псевдозапросы передаются языковой модели в режиме работы с несколькими примерами, чтобы она могла генерировать соответствующий docID. Процесс генеративного извлечения без обучения изображен на рис. 12.3.

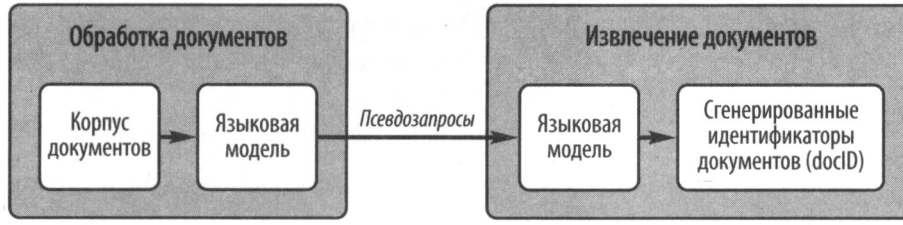



**Рис. 12.3.** Генеративное извлечение

В процессе вывода модели формируется запрос, структура которого аналогична конфигурации, представленной на рис. 12.3. Затем запрашивается корректный docID, соответствующий запросу. Для обеспечения соответствия сгенерированного моделью docID допустимому docID в корпусе используется ограниченный лучевой поиск.



Генеративное извлечение также может быть использовано для поиска документов на основе их метаданных. Например, модель может запросить годовой отчет компании Apple за 2024 год. Для этого модель можно адаптировать к генерации корректного идентификатора путем дополнительного обучения или обучения с несколькими примерами, как описано в данном разделе.

Таким образом, генеративное извлечение оказывается эффективным лишь при условии, что корпус документов сравнительно небольшой, отличается минимальной избыточностью или состоит из материалов, относящихся к четко установленным категориям, таким как годовые отчеты всех публичных компаний США.

Далее рассмотрим жестко связанные ретриверы (tightly-coupled retriever), представляющие собой еще одно новое направление в области поиска.

### Жестко связанные ретриверы

Как следует из *главы 11*, при поиске с использованием эмбеддингов модель эмбеддинга обычно не зависит от языковой модели, в которую передаются результаты извлечения. Такие модели мы будем называть *свободно связанными ретриверами* (loosely-coupled retriever).

В отличие от них, *жестко связанные ретриверы* (tightly-coupled retriever) обучаются таким образом, что они учатся на обратной связи от LLM. Жестко связанные ретриверы могут проходить обучение совместно с генератором LLM в рамках единой архитектуры или отдельно с использованием обратной связи от обученных LLM.

Пример последнего — модель LLM-Embedder, разработанная командой Чжана (Zhang et al., **https://oreil.ly/Q__8M**). Это унифицированная модель эмбеддинга, которая способна удовлетворять различные потребности поиска в рамках одной модели, начиная с извлечения знаний и заканчивая извлечением оптимальных экземпляров с

несколькими примерами. Обучение модели осуществляется с использованием двух видов сигналов: контрастного обучения, которое характерно для большинства моделей эмбеддинга (*см. главу 11*), и обратной связи от LLM. Данные, которые потенциально могут быть извлечены, получают большее вознаграждение от обратной связи LLM, если это повышает эффективность ответа модели на запрос.

**Упражнение**

Используйте модель эмбеддинга LLM-Embedder (**https://oreil.ly/aBwoX**) для демонстрации применения RAG, представленного в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). В чем отличие LLM-Embedder от других моделей эмбеддинга, с которыми мы работали ранее?

Жестко связанные ретриверы представляют собой дополнительный инструмент, который можно использовать для повышения эффективности извлечения данных. Однако их применение не является обязательным этапом в процессе RAG. Результаты экспериментов позволят определить, насколько эффективно применение жестко связанных ретриверов в конкретном приложении.

Теперь рассмотрим GraphRAG — перспективный метод извлечения данных, основанный на использовании графов знаний.

## GraphRAG

Ключевым ограничением существующих подходов к извлечению информации является их неспособность эффективно отвечать на запросы, требующие установления связей между различными частями корпуса документов, а также на запросы, связанные с обобщением тем высокого уровня по всему набору данных. Например, все рассмотренные методы извлечения будут неэффективны при работе с запросами, подобными: "Какие основные темы рассматриваются в данном наборе данных?"

Один из способов преодолеть упомянутые ограничения — использование графов знаний. Компания Microsoft представила GraphRAG (**https://oreil.ly/V4n_S**) — графовую систему RAG. Этот инструмент создает граф знаний, основываясь на анализе исходного корпуса данных с извлечением сущностей и установлением их взаимосвязей. Сформированный граф затем применяется для выполнения иерархической семантической кластеризации с генерацией сводок для каждого кластера. Такие сводки помогают находить ответы на ключевые тематические вопросы, вроде: "Какие основные темы рассматриваются в данном наборе данных?"

Однако подготовка графа знаний через GraphRAG требует значительных вычислительных ресурсов, что может оказаться затруднительным для обработки больших объемов данных. Несмотря на относительную простоту извлечения сущностей, установление релевантных связей между ними представляет собой более сложную задачу.

**Упражнение**

Проведите индексацию GraphRAG для небольшой выборки из набора данных, содержащего материалы заседаний парламента Канады. Проанализируйте извлеченные сущности и отношения. Оцените качество полученных результатов. Проверьте наличие отсутствующих или фиктивных отношений.

Теперь, после этапа извлечения в конвейере RAG, перейдем к стадии повторного ранжирования, или реранкинга.

## Реранкинг

Процесс извлечения часто организован как двухэтапный или многоэтапный подход. Сначала формируется список документов, соответствующих запросу, а затем проводится один или несколько этапов *реранкинга* (reranking), на которых оценивается релевантность извлеченных данных. Обычно реранкер представляет собой более сложную модель по сравнению с ретривером, поэтому его задействуют только для обработки найденных результатов. В противном случае в качестве ретривера можно было бы использовать реранкер.

Реранкер (reranker) обычно представляет собой языковую модель, дообученную под конкретный сценарий использования. Для построения классификатора релевантности можно использовать BERT-подобные модели, которые при наличии запроса и документа определяют вероятность релевантности документа для ответа на запрос. Такие модели называются кросс-энкодерами (cross-encoder), поскольку в них запрос и документ кодируются вместе, в отличие от поисковых моделей на основе эмбеддингов, которые мы обсуждали ранее и которые называются би-энкодерами (bi-encoders). В би-энкодерах запрос и документ кодируются как отдельные векторы.

Входные данные для модели BERT, функционирующей как кросс-энкодер, имеют следующий формат:

```
[CLS] query_text [SEP] document_text [SEP]
```

Библиотека Sentence Transformers предоставляет возможность использования кросс-энкодеров в качестве реранкеров в конвейере RAG:

```
from sentence_transformers import CrossEncoder
model = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-12-v2", num_labels=1)

query = 'When was the Apple iPhone 15 launched?'
// 'Когда состоялась презентация Apple iPhone 15?'

documents = ['Apple iPhone 15 launched with great fanfare in New York',
'He was foolish enough to believe that gifting an iPhone would save the relationship',
```

```
'On September 22, 2023, I lined up at the Central Park store for the launch of the
iPhone 15']
// ['Презентация Apple iPhone 15 с большим размахом прошла в Нью-Йорке ',
// 'Он был настолько наивен, что полагал, будто подарок в виде iPhone сможет
// спасти отношения',
// '22 сентября 2023 года я занял очередь в супермаркете "Централ парк"
// за новым iPhone 15']


ranks = model.rank(query, documents)
for rank in ranks:
    print(rank['score'], documents[rank['corpus_id']])
```

При установке параметра `num_labels=1` модель интерпретирует задачу как регрессионную. При этом для вывода результата используется сигмоидная функция активации, которая возвращает значение в диапазоне от 0 до 1.

В настоящее время разработаны более совершенные подходы, такие как модель Contextualized Late Interaction (контекстуализированное позднее взаимодействие) на основе BERT (ColBERT), используемая для задач реранкинга (**https://oreil.ly/N3fOv**). В отличие от архитектуры кросс-энкодера, о которой шла речь ранее, ColBERT предлагает способ предварительного вычисления представлений документов, позволяя существенно ускорить процесс обработки запросов.

В модели ColBERT как запрос, так и документы кодируются отдельно с использованием BERT, при этом для каждого токена создаются векторы эмбеддингов на уровне токенов. Эти векторы применяются для оценки сходства: каждый токен запроса сопоставляется с каждым токеном документа, формируя соответствующие значения. Затем максимальные значения сходства для всех токенов запроса суммируются, что дает общую оценку релевантности документа. В рамках данной архитектуры происходит суммирование максимальных оценок сходства для каждого токена запроса, что позволяет получить итоговую оценку релевантности. Такой подход называется *поздним взаимодействием* (late interaction), когда запрос и документ не кодируются совместно, а взаимодействуют только на более поздних этапах процесса. Позднее взаимодействие позволяет сэкономить время по сравнению с традиционными кросс-энкодерами, т. к. эмбеддинги документов можно создать и сохранить заранее.

Механизм позднего взаимодействия между запросом и документами в модели ColBERT показан на рис. 12.4.

### Упражнение

Проанализируйте производительность би-энкодеров, таких как модель all-mpnet-base-v2, и кросс-энкодеров типа jina-colbert-v2 (**https://oreil.ly/we84L**), применяя их для генерации эмбеддингов для страницы Википедии, посвященной iPhone (**https://oreil.ly/ur7vc**). После этого проведите тестирование с использованием

множества запросов. Обратите внимание на те типы запросов, где кросс-энкодеры показывают явное превосходство в результатах.




**Рис. 12.4.** Модель ColBERT

Рассмотрим несколько передовых методов реранкинга.

## Модель вероятности запроса (QLM)

Модель QLM (query likelihood model) представляет собой инструмент, который оценивает вероятность формирования запроса на основе потенциального документа. QLM можно рассматривать как LLM с нулевым примером, используя ее возможности для ранжирования допустимых документов на основе вероятностей токенов запроса. Для повышения эффективности LLM в роли QLM ее можно дополнительно обучить на задачах генерации запросов.

Типичный запрос для QLM может выглядеть следующим образом: "Сформулируй вопрос, который наиболее точно соответствует содержанию `<document content>` данного документа".

После извлечения top-*k* документов, связанных с запросом, каждый документ передается в LLM вместе с указанным промптом. Затем с помощью логитов модели вычисляется вероятность токенов запроса. Документы сортируются по вероятности, что позволяет ранжировать их по релевантности.



В работе Чжуана и соавт. (Zhuang et al., **https://oreil.ly/QnWWh**) показано, что модель, обученная следовать инструкциям, но не содержащая задачи генерации запросов в своем обучающем наборе данных, теряет способность быть эффективной QLM с нулевым примером. Это еще один случай, когда модели, обученные следовать инструкциям, демонстрируют ухудшение производительности на задачах, на которых они не были обучены, по сравнению с базовыми моделями.

Важно подчеркнуть, что для расчета вероятности токенов запроса требуется доступ к логитам модели. Однако на момент написания книги большинство провайдеров проприетарных моделей, включая OpenAI, не предоставляют полный доступ к этим данным. Поэтому реализация подхода LLM-as-a-QLM (LLM в качестве QLM) возможна исключительно с применением моделей с открытым исходным кодом.

**Упражнение**

Выберите любую относительно небольшую модель LLM с открытым исходным кодом, содержащую примерно 3 млрд параметров, и оцените ее пригодность для использования в качестве QLM. С помощью QLM проведите ранжирование потенциальных документов для извлечения из набора данных, содержащего материалы заседаний парламента Канады, доступного в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Оцените эффективность данного подхода.

Для снижения задержек рекомендуется, чтобы модель QLM имела максимально компактные размеры. Однако меньшие модели менее эффективны в качестве QLM. Одним из эффективных решений является дообучение меньшей LLM для генерации запросов.

## Дистилляция LLM для задач ранжирования

В предыдущем разделе была рассмотрена возможность применения в качестве реранкеров моделей с архитектурой только кодировщика, таких как BERT. В последнее время для задач ранжирования документов также активно развиваются подходы с использованием LLM на базе декодеров. Эти модели обучаются ранжировать предполагаемые документы тремя способами.

*Точечное ранжирование (pointwise ranking)*

Каждый потенциальный документ обрабатывается в LLM индивидуально. В рамках такого подхода модель выполняет логическое суждение о его релевантности. В некоторых случаях может предоставляться числовая оценка уровня релевантности, хотя данный подход часто оказывается менее надежным.

*Попарное ранжирование (pairwise ranking)*

Модель последовательно анализирует пары предполагаемых документов, определяя, какой из них кажется более релевантным. Для получения полного рейтинга необходимо выполнить $N^2$ таких сравнений.

*Ранжирование списка (listwise ranking)*

Все потенциальные документы маркируются идентификаторами и передаются в LLM. Модели предлагается составить ранжированный список идентификаторов в порядке убывания релевантности соответствующих документов.

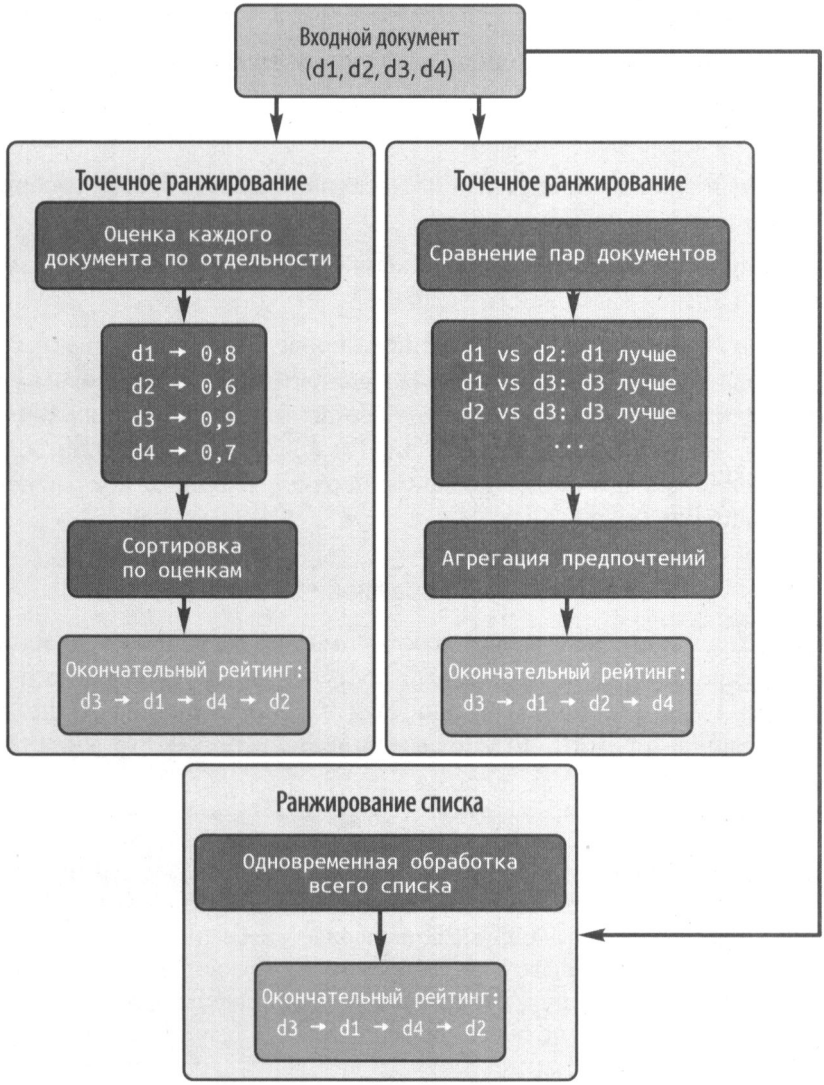



**Рис. 12.5.** Реранкеры LLM на базе декодера

Точечное ранжирование является наиболее простым методом, но его эффективность может быть ниже, чем у других подходов (**https://oreil.ly/DvmtC**). Для применения ранжирования списков часто требуется большое контекстное окно, в то время как парное ранжирование предполагает большое количество сравнений. При этом парное ранжирование считается наиболее эффективным из всех, поскольку

оно основано на прямом сравнении элементов. На рис. 12.5 показаны рабочие процессы точечного, парного и списочного ранжирования.

Примеры систем ранжирования на базе LLM включают такие проекты, как RankGPT (**https://oreil.ly/6XoOG**), RankVicuna (**https://oreil.ly/00Dan**) и RankZephyr (**https://oreil.ly/AAbUE**). Эти модели обучаются с использованием техники дистилляции из более крупных языковых моделей, как подробно описано в *главе 9*.

Для иллюстрации рассмотрим процесс обучения RankVicuna.

- В обучающем наборе запросы обрабатываются ретривером первого уровня, например BM25, для формирования списка потенциально релевантных документов.
- Полученный список передается более крупной LLM, которая генерирует ранжированный перечень кандидатов.
- Затем запрос вместе с ранжированным списком используется для дообучения более компактной языковой модели.

Авторы RankVicuna (**https://oreil.ly/cFLSc**) провели исследование, в ходе которого было установлено, что повышение эффективности поиска на первом уровне приводит к снижению прироста производительности от использования RankVicuna. Это обусловлено эффектом снижения отдачи. Также было установлено, что увеличение объема данных за счет перемешивания порядка входных документов позволяет улучшить работу модели.

**Упражнение**

Воспользуйтесь примером ассистента по ответам на вопросы по материалам заседаний парламента Канады, который доступен в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Используйте RankVicuna на этапе реранкинга. Измените шаблон промпта по умолчанию и оцените, как это влияет на производительность.



Для формирования окончательного рейтинга релевантности потенциальных документов целесообразно объединить результаты этапов извлечения и реранкинга. Такой подход позволяет, в частности, принудительно учитывать вес ключевых слов. Кроме того, можно задать дополнительные веса для ранжирования, основываясь на метаданных, таких как дата публикации (документы с более свежей датой публикации имеют больший вес).

## Уточнение

После того как тексты, соответствующие заданному запросу, были извлечены и отобраны, их можно передать в LLM для дальнейшей обработки. Однако, поскольку контекстное окно LLM ограничено, может потребоваться предварительно сократить объем этих текстов. Кроме того, их можно перефразировать, чтобы сделать более удобными для анализа моделью. Также можно применить фильтрацию мате-

риалов на основе заранее определенных правил. Все эти операции выполняются на этапе *уточнения* (refine). В этом разделе мы рассмотрим два метода: суммаризацию (summarization) и применение цепочки заметок (chain-of-note). Начнем с процесса суммаризации извлеченных текстов.



Стадия уточнения может быть самостоятельным этапом или сочетаться с завершающим этапом генерации. В последнем случае окончательный ответ предоставляется сразу после уточнения извлеченной информации и является частью того же промпта или последовательности промптов.

## Суммаризация

Суммаризация становится особенно полезной, когда необходимо обработать большие объемы текстовой информации. Этот процесс можно разделить на два основных типа: экстрактивный (извлекающий, extractive) и абстрактный (abstractive). *Экстрактивная суммаризация* предполагает выделение ключевых предложений из исходного текста без их изменения, в то время как *абстрактная суммаризация* подразумевает создание нового текста на основе содержания исходного материала. Кроме того, суммаризатор (summarizer) может выполнять функцию фильтра качества, создавая пустую сводку в тех случаях, когда документ не соответствует запросу. Важно, чтобы итоговые сводки были актуальными, краткими и отражали содержание исходного текста.



Такие сводки предназначены не для использования человеком, а для обработки языковыми моделями. Именно поэтому они могут отличаться по целям от традиционных методов составления сводных данных. Основная задача заключается в создании сводки, которая эффективно поддержит LLM в формулировании точного ответа.

Выбор между экстрактивной и абстрактной суммаризацией зависит от поставленных целей и приоритетов пользователя. Экстрактивные сводки считаются более надежными, поскольку они сохраняют смысл исходного текста. Абстрактная суммаризация, несмотря на риск возникновения галлюцинаций, способна формулировать данные более лаконично и интегрировать информацию из разных частей документа или даже из множества источников.

Алгоритмы LLM с нулевым примером можно применить как к экстрактивной, так и к абстрактной суммаризации. Однако создание оптимизированных сводок за счет дообучения модели позволяет добиться большей эффективности, хотя это требует значительных ресурсов. В таких случаях модель обучается генерировать ответы, которые максимально соответствуют ожиданиям. Этот подход называется *жестко связанным суммаризатором* (tightly-coupled summarizers).

В исследовании Сюй и соавт. (Xu et al., **https://oreil.ly/XCpyr**) разработаны методики обучения обоих типов суммаризаторов.

Экстрактивная суммаризация предполагает выбор наиболее значимых предложений из исходного документа для составления сводки. В этом случае создаются эмбеддинги как для входного запроса, так и для каждого предложения документа. Затем в качестве итоговой сводки отбираются top-*k* предложений, которые демонстрируют наибольшую схожесть с запросом в пространстве эмбеддингов. Степень сходства определяется расстоянием между эмбеддингами, что позволяет оценить, насколько качественно выбранные предложения из документа способствуют генерации моделью точного вывода.

Экстрактивный суммаризатор проходит этап контрастного обучения, который подробно описан в *главе 11*. В рамках этого подхода каждый обучающий пример представлен в виде триплета: анкорное предложение, положительный пример, семантически близкий к анкорному предложению, и несколько отрицательных примеров, значительно отличающихся от него. Для генерации таких примеров каждое предложение из извлекаемого документа объединяется с входным запросом, и оценивается вероятность генерации целевых выходных токенов. В качестве положительного примера выбирается предложение с наибольшей вероятностью. В качестве отрицательных примеров выбираются до пяти предложений, вероятность которых ниже заданного порога. Затем сгенерированный набор данных используется для обучения модели.

Для эффективной абстрактной суммаризации можно прибегнуть к дистилляции более крупной LLM. Это подразумевает использование ее выходных данных для дальнейшего обучения более компактной языковой модели. Для формирования обучающего набора данных применяются несколько шаблонов промптов, которые используются совместно с большой LLM для генерации сводок с нулевым примером на основе извлеченных документов. Важно отметить, что создается единая сводка, объединяющая информацию из всех извлеченных документов. По аналогии с экстрактивной суммаризацией, каждая сгенерированная сводка добавляется в качестве префикса к входному тексту, после чего вычисляется вероятность корректных выходных токенов. Таким образом, сводка с наибольшей вероятностью корректности становится частью обучающего набора.

Если префиксация сводки на этапе вывода приводит к снижению вероятности получения корректного результата по сравнению с вариантом без префиксации, то содержание текста, представленного в этой сводке, считается нерелевантным. Это позволяет отфильтровывать нерелевантные документы.

Рабочий процесс жестко связанного абстрактного суммаризатора показан на рис. 12.6.



При изменении целевой LLM может потребоваться переобучение сводных моделей. Несмотря на то что инструменты для переноса сводных данных позволяют эффективно передавать информацию между моделями, возможно незначительное снижение производительности.

Жестко связанные суммаризаторы, несмотря на высокую стоимость начального обучения, способны эффективно удалять нерелевантную информацию из извлеченного текста. Они также помогают преобразовывать текст в более однозначную форму при перефразировании, что упрощает его понимание для больших языковых моделей.




**Рис. 12.6.** Абстрактная суммаризация

## Упражнение

Используя примеры, представленные в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**), сгенерируйте экстрактивные и абстрактные сводки извлеченных документов в рамках задания ассистента по ответам на вопросы о заседаниях парламента Канады. Оцените эффективность сводок, созданных с использованием данного подхода, по сравнению с динамическим суммаризатором, например ChatGPT.

### Цепочка заметок

Для перефразирования извлеченного текста можно использовать генерацию заметок (notes). Было бы нежелательно, если бы извлеченный текст содержал нерелевантную информацию, которая могла бы ввести модель в заблуждение. В сущности, LLM сталкивается с тремя типами сценариев:

- извлеченные документы содержат ответ на запрос пользователя, и LLM может использовать его для генерации правильного вывода;
- извлеченные документы не содержат ответа на пользовательский запрос, но предоставляют полезный контекст, который в сочетании с внутренними данными модели помогает сформулировать ответ;
- извлеченные документы не имеют отношения к запросу пользователя и должны быть проигнорированы.

Проблема заключается в том, что LLM часто испытывают трудности с различением релевантного и нерелевантного контекста. Для решения этой проблемы предлагается создавать заметки для каждого извлеченного документа. Заметки включают краткое содержание документа и указывают, содержит ли документ прямой ответ на запрос, предоставляет ли он только полезный контекст без прямого ответа или вообще не является релевантным. Данный метод, описанный в работе Ю и соавт. (Yu et al., **https://oreil.ly/hPkKm**), называется *цепочкой заметок* (chain-of-note, CoN).

Пример заметок выглядит следующим образом.

> *Промпт*: Кто сменил Брайана Малруни на посту премьер-министра Канады в 1993 году?
>
> Документ 1: В 1993 году Ким Кэмпбелл стала первой женщиной в истории Канады, занявшей пост премьер-министра страны. Однако ее руководство оказалось кратковременным, поскольку в том же году она возглавила Прогрессивно-консервативную партию на непростых федеральных выборах.
>
> Документ 2: После завершения срока Брайана Малруни Канада пошла по историческому пути, впервые назначив женщину на пост премьер-министра, что стало знаковым событием и переломным моментом в политической истории страны.

В первом тексте упоминается, что Ким Кэмпбелл заняла пост премьер-министра в 1993 году, но остается неясным, была ли она прямым преемником Брайана Малруни или между ними был другой глава правительства. Во втором тексте ситуация проясняется, поскольку там прямо указано: на смену Брайану Малруни пришла женщина-премьер, и это подтверждает, что это была именно Ким Кэмпбелл.

Модель генерирует следующую заметку.

> *LLM*: В первом утверждении говорится, что Ким Кэмпбелл возглавила Канаду в 1993 году, а во втором указано, что это произошло после окончания срока Брайана Малруни, что привело к назначению первой женщины на пост премьер-министра Канады. Такая последовательность событий предполагает, что Ким Кэмпбелл стала премьер-министром сразу после Малруни, заняв его место в качестве преемника.

Используя заметку в качестве контекста, LLM может сформировать правильный ответ.

Цепочка заметок особенно актуальна, когда контекст содержит нерелевантную или недостаточную информацию. В таких случаях сгенерированная заметка позволяет LLM избежать вывода "Я не знаю".

Рассмотрим следующий запрос.

> *Промпт*: Поддержала ли Партия зеленых Канады предложенный законопроект об общественном транспорте 2023 года?

И извлеченные документы:

> Документ 1: Партия зеленых Канады на протяжении многих лет акцентировала внимание на значимости устойчивого развития транспорта как важнейшего аспекта своей экологической платформы. Она выступает за политику, способствующую созданию "зеленой" инфраструктуры.
>
> Документ 2: Предложенный законопроект об общественном транспорте 2023 года привлек широкое внимание и стал популярным, что свидетельствует о растущем согласии общества с необходимостью внедрения устойчивых транспортных решений.

Извлеченные документы не содержат ответа на поставленный вопрос. Сгенерированная заметка выглядит следующим образом.

> *LLM*: В первом отрывке упоминается исторический акцент партии на устойчивом транспорте, а во втором — популярность законопроекта. Однако ни в одном из отрывков не говорится о позиции партии относительно законопроекта 2023 года.

Используя эту заметку, LLM может создать "неизвестный" ответ, если в извлеченных документах не будет ответа на запрос.

Пример промпта CoN может выглядеть следующим образом.

> *Промпт*: Предоставляется запрос, а также текст, содержащий {K} абзацев, которые могут содержать информацию, полезную для ответа на запрос. Создай заметки, в которых будут обобщены основные моменты из этих абзацев. Оцени релевантность каждого абзаца запросу и определи, можно ли извлечь ответ на запрос из их контента.

Кроме того, можно повысить эффективность, обучив модели для работы с жестко связанными цепочками заметок (CoN). Это можно реализовать путем дообучения LLM для активации поведения CoN.

Для формирования набора данных для дообучения можно запросить LLM сгенерировать потенциальные заметки, например запросы. Затем эти заметки может оценить человек, чтобы отсеять некорректные или низкокачественные варианты. В результате окончательный набор данных будет включать в себя первичный промпт CoN, исходный запрос пользователя и извлеченные документы в качестве входных данных, а также соответствующую заметку и ответ на запрос в качестве выходных данных. На основе этого набора можно провести дообучение модели.

Исследователи (Yu et al.) предложили подход, который использует схему взвешивания потерь на стадии обучения. В связи с тем, что заметки часто бывают значительно длиннее ответа, равное распределение веса потерь между всеми токенами приводит к неравномерности процесса обучения. Это может замедлить сходимость модели. Схема взвешенных потерь предполагает акцент на расчет потерь по токенам ответа в 50% случаев, что помогает сбалансировать процесс обучения.

Этап применения CoN играет ключевую роль, особенно если известно, что данные поисковой выборки содержат много ненужной информации или существует высокая вероятность отсутствия релевантных документов для обработанного запроса. Кроме того, учет сложности поведения CoN становится особенно актуальным для небольших моделей, поэтому для данной задачи предпочтительно использовать более крупные модели.

**Упражнение**

Для иллюстрации работы RAG с данными, полученными из материалов заседаний канадского парламента, доступных в репозитории книг на GitHub, сформулируйте запросы к системе RAG, с учетом того, что ответы на эти запросы отсутствуют в корпусе Википедии. Создайте заметки, используя промптинг CoN в ChatGPT или аналогичной более крупной языковой модели. Содержатся ли в заметках указания на отсутствие соответствующей информации? Признает ли языковая модель свою неспособность ответить на запрос?

Таким образом, мы рассмотрели этап уточнения в конвейере RAG. Теперь перейдем к этапу вставки.

## Вставка

После определения контента, который будет передан языковой модели для генерации окончательного ответа на запрос, необходимо решить, как структурировать его в рамках промпта. Контент может включать исходные извлеченные документы, их краткое изложение (сводку) или заметки.

Стандартный подход предполагает размещение всего контента (или максимально допустимого объема) в пределах контекстного окна. Альтернативный метод заключается в передаче каждого документа, сводки или заметки отдельно, добавляя их в качестве префиксов к входным данным, а затем объединяя выходные результаты модели для получения окончательного ответа.

В работе Лю и соавт. (Liu et al., **https://oreil.ly/LFR8r**) было показано, что языковые модели лучше справляются с задачей извлечения информации, расположенной в начале и конце контекстного окна, по сравнению с информацией, расположенной в середине. Эту особенность можно использовать для переупорядочивания извлеченных документов в промпте.

Предположим, по заданному запросу было извлечено 10 документов, отсортированных по степени их релевантности: Doc1, Doc2, ..., Doc10. Их можно расположить в промпте в следующем порядке:

Doc1, Doc3, Doc5, Doc7, Doc9, Doc10, Doc8, Doc6, Doc4, Doc2

В этом случае наименее релевантные документы находятся в середине контекстного окна, что может привести к их игнорированию моделью из-за ограничений на вызов длинного контекста.

Также документы можно упорядочить в соответствии с их релевантностью:

Doc1, Doc2, Doc3, Doc4, Doc5, Doc6, Doc7, Doc8, Doc9, Doc10

или в обратном порядке релевантности:

Doc10, Doc9, Doc8, Doc7, Doc6, Doc5, Doc4, Doc3, Doc2, Doc1

Такие схемы упорядочивания эффективны только при работе с очень длинными входными контекстами, например более 5000 токенов.

И наконец, рассмотрим этап генерации в конвейере RAG.

## Генерация

На данном этапе LLM генерирует окончательный ответ на запрос. Обычно вывод генерируется сразу. Однако можно также комбинировать процесс генерации и извлечения, генерируя часть вывода и извлекая дополнительный контекст, затем генерируя следующую часть вывода и извлекая дополнительный контекст и т. д.

В данной методике сгенерированный текст определяет контекст, который должен быть извлечен следующим. Такой подход называется *активным извлечением* (active retrieval) и эффективен для обеспечения последовательности при генерации длинных текстов.

Как определить момент, когда следует прекратить генерацию и начать новый этап извлечения? Для этого можно использовать следующие методы:

- извлекать данные после генерации каждых $N$ токенов;
- извлекать данные после генерации каждой текстовой единицы (например, предложения, абзаца, раздела и т. д.);
- извлекать данные, когда текущий контекст кажется недостаточным для генерации.

Существует несколько подходов к реализации последнего из перечисленных методов, одним из которых является FLARE (Forward-Looking Active Retrieval-augmented generation — прогнозная активная генерация с дополненным поиском). Авторы FLARE (**https://oreil.ly/eZRdy**) предлагают два способа активного поиска: FLARE-Instruct и FLARE-Direct.

В рамках техники FLARE-Instruct модели LLM предписывается формировать запросы с заранее заданным синтаксисом каждый раз, когда ей требуется дополнительная информация для последовательного продолжения процесса генерации.

В методе FLARE-Direct модель генерирует потенциальное следующее предложение и оценивает вероятность каждого токена. Если хотя бы один токен в сгенерированном предложении имеет вероятность ниже заданного порогового значения, запускается процесс извлечения. В случае если все токены удовлетворяют порогу, предложение принимается как допустимое продолжение, и генерация продолжается. Когда возникает необходимость в извлечении, сгенерированное предложение можно использовать в качестве запроса путем маскирования токенов с низкой вероятностью (т. к. если они нерелевантны или ошибочны, они могут ввести в заблуждение ретривер). Также можно перефразировать предложение в качестве запроса о токене с низкой вероятностью.

Рассмотрим пример использования FLARE-Instruct.

> *Промпт*: Напиши статью о Перут Чемутай — медалистке Олимпийских игр из Уганды.
>
> *FLARE-Instruct*: Перут Чемутай [Search(дата рождения Перут Чемутай)] — угандийская легкоатлетка, специализируется в беге на 3000 метров с препятствиями. Она получила международное признание, выиграв [Search(какие медали выиграла Перут Чемутай)] золотую медаль в женском забеге на 3000 метров с препятствиями на летних Олимпийских играх 2020 года, став первой женщиной из Уганды, завоевавшей олимпийское золото.
>
> Ранние годы
>
> Чемутай родилась в [Search(место рождения Перут Чемутай)]. Этот регион известен своими [Search(чем известна родина Перут Чемутай?)].

Это всего лишь упрощенный пример, и полное представление о преимуществах FLARE можно получить на основе более длинных выходных данных. Как видно из результатов, модель формирует поисковые запросы, которые позволяют извлекать достоверную информацию из доступных источников данных.

При использовании FLARE-Direct для того же запроса модель предлагает вариант статьи.

> *FLARE-Direct*: Перут Чемутай (родилась 10 июля 1999 года) — угандийская легкоатлетка, специализируется в беге на 3000 метров с препятствиями. Она получила мировое признание, выиграв золотую медаль в женском забеге на 3000 метров с препятствиями на летних Олимпийских играх 2020 года, став первой женщиной из Уганды, завоевавшей олимпийское золото.
>
> Ранние годы
>
> Чемутай родилась в округе Буква, Уганда. Этот регион известен своим сложным рельефом и традиционно выдающимися бегунами на длинные дистанции.

Подчеркнутые токены представляют собой маловероятные токены, которые могут быть заменены или дополнены на основе извлечения релевантного контента. В таких случаях можно использовать эти токены в качестве основы для запроса на извлечение данных либо сформировать самостоятельные запросы, например: "Когда родилась Перут Чемутай?"

### Упражнение

С помощью FLARE-Direct создайте документы Википедии. Рассмотрим возможность создания документов Википедии путем чередования генерации и извлечения по запросу. Запросите любую LLM с открытым исходным кодом создать страницу Википедии для болливудского фильма "Любовь — это жизнь" (*Kabhie Kabhie*) с использованием FLARE-Direct. Оцените эффективность использования извлечения.

Ключевым аспектом генерации является добавление ссылок на достоверные источники. Модель можно дообучить таким образом, чтобы на запросы пользователей она отвечала цитатами. Примером такой модели является Command-R от компании Cohere (**https://oreil.ly/v0KUs**).

Как видно, конвейер RAG для извлечения знаний может оказаться довольно сложным. Однако для многих приложений RAG ключевым фактором является задержка, что повышает значимость более быстрых подходов, не основанных на LLM, или меньших языковых моделей.

Соберем все воедино, рассмотрев схему конвейера RAG, представленную в начале главы. Рабочий процесс комплексного конвейера RAG еще раз показан на рис. 12.7.




**Рис. 12.7.** Комплексный конвейер RAG

Таким образом, мы разобрали применение RAG для извлечения знаний. Далее рассмотрим другие возможные способы использования данной методики.

## Управление памятью с помощью RAG

Управление памятью с помощью RAG открывает новые возможности для расширения контекстного окна моделей LLM. Несмотря на то что такой подход часто остается недооцененным, он способен существенно улучшить качество взаимодействия. Контекстное окно модели обычно включает несколько типов данных, которые могут быть представлены в качестве необязательных элементов.

*Препромпт (pre-prompt) или системный промпт (system prompt)*

Это инструкции общего характера, предоставляемые LLM в начале каждого запроса. В зависимости от настроек системный промпт может занимать значительную часть контекстного окна.

*Входной промпт (input prompt)*

Данный компонент объединяет текущий ввод пользователя с инструкцией, а также может содержать обучающие примеры (обучение с несколькими примерами) и дополнительные данные, полученные с помощью механизмов извлечения.

*История диалога (conversational history)*

Этот элемент включает в себя предыдущие сообщения в рамках взаимодействия между пользователем и моделью. Сохранение истории диалога позволяет поддерживать непрерывность общения и формировать более естественные ответы в длительных беседах.

*Цифровой черновик (scratchpad)*

В нем содержатся промежуточные результаты, которые генерируются LLM (*см. главу 8*) и которые модель может использовать для генерации выходных данных в будущем. Содержимое черновика формируется с применением определенных методов промптинга, таких как цепочка рассуждений (CoT).

В ряде случаев ограниченного контекстного окна в моделях LLM может быть недостаточно для включения всех необходимых данных. Тогда можно использовать механизм бесконечной последовательности (perpetuity), который позволяет модели расширять свою историю диалогов со временем. Этот подход играет важную роль в персонализации моделей, предоставляя им возможность постоянного обучения и улучшения своих способностей.

В этой ситуации может оказаться полезной техника RAG. Она позволяет упростить управление памятью модели, заменяя соответствующий контент в промпте. Этот процесс аналогичен управлению памятью в операционных системах. Рассмотрим эту концепцию подробнее.

В операционных системах память организована в виде иерархической структуры, где наиболее быстрые и дорогие области памяти доступны напрямую процессору, а более высокие уровни содержат больший объем медленной, но относительно дешевой памяти. При обращении процессора к данным он сначала проверяет нижний уровень иерархии. Если необходимые данные не найдены, поиск продолжается на следующем уровне. Если данные найдены, они переносятся на нижний уровень,

при этом вытесняются те элементы, которые в данный момент не используются. Благодаря такому механизму операционная система предоставляет процессору быструю основную память и одновременно управляет объемной виртуальной памятью, которую можно подключить при необходимости.

Выше представлено упрощенное описание принципов управления памятью в операционной системе. Более детальное рассмотрение этой темы можно найти в работе Тони "Операционная система — иерархия памяти" ("Operating System — Hierarchy of Memory", **https://oreil.ly/vcciM**). Иерархия памяти в типичной операционной системе представлена на рис. 12.8.




**Рис. 12.8.** Иерархия памяти в типичной ОС

Контекстное окно в LLM можно сравнить с основной памятью, т. к. оно обеспечивает прямой доступ модели к данным. Однако возможности контекстного окна можно неограниченно расширить, создав систему памяти, аналогичную виртуальной памяти операционных систем. Такой подход позволяет персонализировать LLM, предоставляя ей полный доступ к истории взаимодействий с пользователем, а также к его явным и неявным предпочтениям.

Среди инструментов, поддерживающих управление памятью для LLM, можно выделить такие библиотеки, как Letta (ранее известная как MemGPT) (**https://oreil.ly/1p8Vu**) и Mem0 (**https://oreil.ly/dgJaZ**).



Рекурсивная суммаризация диалогов может служить альтернативой или дополнением к обмену памятью, однако этот процесс всегда сопряжен с потерей части информации, что может привести к искажению исходной семантики текста. В частности, в результате суммаризации могут быть утрачены ценные нюансы, такие как авторский стиль или тон.

**Упражнение**

Для проведения продолжительной беседы с LLM воспользуйтесь платформой Mem0 (**https://oreil.ly/dgJaZ**). В ходе диалога обращайтесь к темам, которые могут быть интересны вашему гипотетическому другу. Попросите LLM помочь выбрать подарок на его день рождения, учитывая его интересы и предпочтения. Может ли LLM извлечь информацию из беседы о нем? Возможно ли создать уточненную версию диалога, которая упростит процесс извлечения этой информации?

## Выбор примеров контекстного обучения с помощью RAG

В начале главы отмечалось, что одной из важных функций RAG является динамический подбор обучающих примеров для задач обучения с несколькими примерами. Этот процесс предполагает извлечение оптимальных примеров из хранилища данных, содержащего список обучающих примеров. На основе заданных входных данных нужно выбрать такие примеры, которые максимизируют вероятность того, что LLM сгенерирует правильный ответ на запрос пользователя.

Самый простой подход заключается в создании эмбеддингов для входных данных и дальнейшем поиске примеров с эмбеддингами, наиболее схожими с этими входными эмбеддингами. Хотя этот метод демонстрирует перспективные результаты, существует возможность добиться гораздо более высоких показателей эффективности.

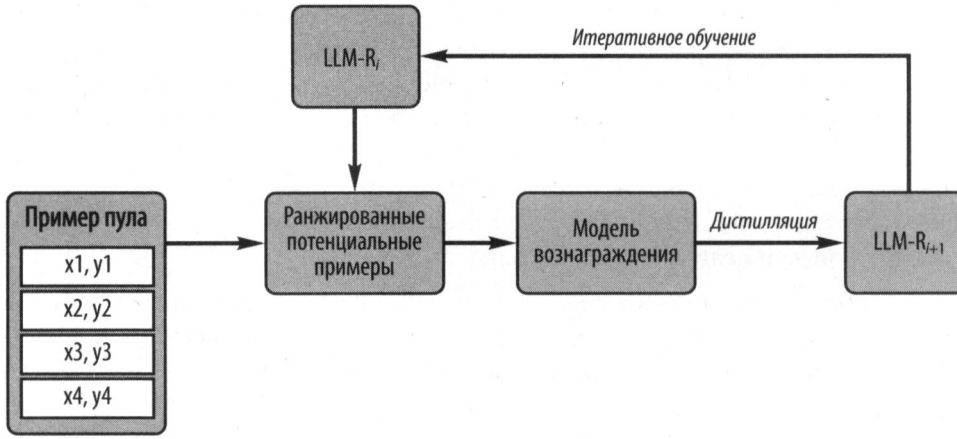



**Рис. 12.9.** Рабочий процесс LLM-R

В работе Ванга и соавт. (Wang et al., **https://oreil.ly/r8735**) предложена методика под названием LLM Retriever (LLM-R). Данный подход основан на обучении модели с использованием обратной связи от LLM и направлен на подбор примеров, ко-

торые значительно повышают вероятность генерации правильного ответа. Метод LLM-R проиллюстрирован на рис. 12.9.

При каждом новом запросе из набора обучающих данных сначала извлекаются $k$ лучших примеров с использованием поисковой модели, например BM25. Затем эти примеры повторно ранжируются с помощью обратной связи от LLM. Каждый пример добавляется в качестве префикса к входным данным, после чего оценивается вероятность генерации эталонных токенов. Примеры сортируются по убыванию их логарифмических вероятностей. Далее ранжированные примеры используются для обучения модели вознаграждения, которая впоследствии применяется для обучения итоговой поисковой модели методом дистилляции.

## Обучение модели с помощью RAG

До сих пор мы рассматривали приложения RAG, которые применяются на этапе вывода LLM. Однако можно ли применять RAG в процессе предобучения и дообучения модели? Определенно, да! Это направление исследований остается недостаточно изученным, но, возможно, в ближайшие годы появится больше языковых моделей, использующих подобный подход. Рассмотрим это на конкретном примере.

Одной из передовых разработок в области RAG является модель REALM (Retrieval-Augmented Language Model). Она объединяет процессы извлечения и генерации информации в единую архитектуру. На рис. 12.10 показана архитектура REALM для предобучения и дообучения.

Архитектура REALM состоит из двух ключевых компонентов: ретривера знаний и расширенного кодировщика знаний, который представляет собой модель, аналогичную BERT только с кодировщиком (encoder-only). Эти элементы полностью дифференцируемы, что позволяет обучать их совместно.

Ретривер знаний выполняет задачу генерации эмбеддингов для всех документов, находящихся во внешней базе данных. Извлечение реализуется путем поиска документов, имеющих наибольшее сходство с входными данными. На этапе предобучения маскированного языкового моделирования функция потерь ретривера стимулирует его выбирать тексты, которые наиболее эффективно помогают предсказать маскированные токены. После этого маскированные токены определяются посредством обработки как входного, так и извлеченного текста. Предполагается, что извлеченный текст содержит релевантный контекст, который значительно упрощает предсказание маскированных токенов.

Для оптимизации процесса обучения REALM использует следующие стратегии.

- Именованные сущности и даты маскируются, чтобы модель могла предсказывать их на основе контекста.
- Не все маскированные токены требуют внешних знаний для предсказания, поэтому к набору извлеченных документов всегда добавляется пустой документ.

- В идеале извлеченные документы должны содержать только контекст, необходимый для предсказания маскированного токена, а не сам токен. Поэтому тривиальные извлечения, содержащие маскированный токен в извлеченном тексте, не учитываются.

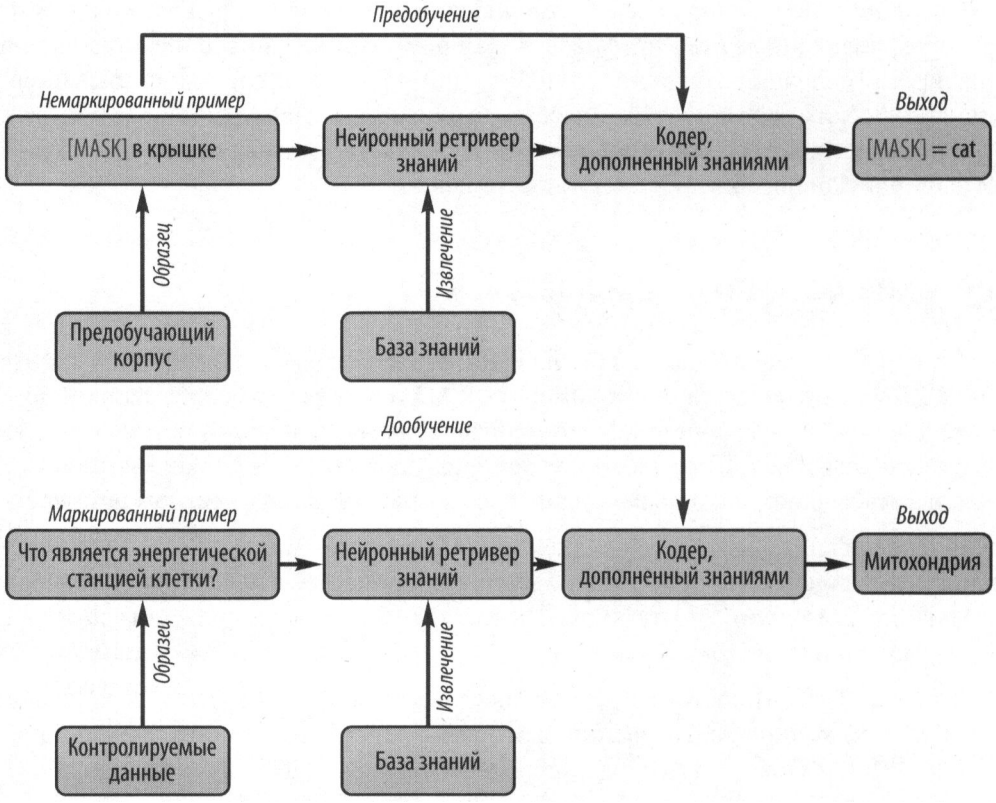



**Рис. 12.10.** Архитектура REALM

## Ограничения метода RAG

Несмотря на значительные преимущества технологии RAG, позволяющие расширить функциональные возможности LLM и снизить вероятность возникновения галлюцинаций, данный метод не устраняет все недостатки, присущие языковым моделям. Среди ключевых проблем, связанных с использованием метода RAG, можно выделить следующие.

- Высокая зависимость от извлеченных текстовых фрагментов. Это может привести к тому, что LLM будет опираться на поверхностную информацию при формировании ответов вместо глубокого анализа и понимания контекста запроса.
- Ограничения этапа извлечения. Если в процессе извлечения не удается найти релевантный текст, даже самые мощные возможности LLM могут оказаться неэффективными.

- Конфликты информации. В некоторых случаях в процессе извлечения могут быть отобраны документы, противоречащие внутренним знаниям, заложенным в параметрической памяти модели. Без доступа к достоверным источникам разрешение подобных конфликтов становится практически невозможным для LLM.

**Как LLM обрабатывают противоречивую информацию?**

В некоторых случаях знания, полученные LLM, могут противоречить информации, полученной с помощью техники RAG. Это может быть связано с различными причинами, такими как устаревшие или некорректные данные в обучающих наборах LLM, ошибки в контексте, предоставленном пользователем, либо извлечение нерелевантных или неточных документов в процессе RAG. В таких ситуациях крайне важно, чтобы модель могла игнорировать неточные данные, но это затрудняется из-за отсутствия у нее доступа к достоверным исходным данным.

В работе Лю и соавт. (Liu et al., **https://oreil.ly/7AOZl**) представлен бенчмарк RECALL (Robustness against External Counterfactual Knowledge — критерий устойчивости LLM к внешним контрафактным знаниям), который оценивает устойчивость LLM к противоречивой информации, представленной в запросах. Результаты исследования показывают, что модели обычно склонны доверять своим внутренним представлениям, если сталкиваются с логическими противоречиями. Однако при наличии фактических несоответствий модели чаще опираются на информацию из предоставленного контекста.

Важным выводом исследования является то, что доверие моделей к результатам снижается при работе с противоречивой информацией. Это позволяет использовать вероятности вывода LLM для управления последующей специализированной обработкой данных.

## RAG и длинный контекст

Как было отмечено в *главе 5*, одним из недостатков больших языковых моделей является ограниченное контекстное окно, которое они могут эффективно использовать. Однако в последнее время в этой области были достигнуты значительные успехи. До начала 2023 года контекстные окна ограничивались несколькими тысячами токенов. Затем такие компании, как Anthropic (**https://oreil.ly/ucbD-**), объявили о поддержке контекстных окон, охватывающих более 100 000 токенов. В начале 2024 года компания Google анонсировала новую модель Gemini 1.5 Pro (**https://oreil.ly/rp7pi**), которая поддерживает контекстное окно в 1 млн токенов.

Для оценки влияния увеличения контекста на производительность LLM была разработана серия тестов "иголка в стоге сена" (needle-in-a-haystack). Одним из примеров такой реализации является подход Грега Камрадта (Greg Kamradt, **https://oreil.ly/M8Jc9**), когда в середину контекста (стога сена) добавляется случайный факт или утвер-

ждение (иголка), после чего модели задаются вопросы, ответы на которые требуют обнаружения иголки.

Однако к таким тестам следует относиться с осторожностью, поскольку они в основном проверяют способность LLM извлекать информацию, которая уже была ранее запомнена. Кроме того, в реальных условиях лишь незначительная часть задач представляет собой поиск "иголки в стоге сена". В таких случаях использование LLM может быть нецелесообразным, и для большинства подобных задач лучше подойдут более доступные и быстрые поисковые модели.

Во многих тестах "иголки в стоге сена" случайные предложения или абзацы добавляются в контекстовое окно в качестве "иголок", а остальная часть содержимого в контекстовом окне является "ортогональной" по отношению к "иголке". Однако такой подход не отражает ситуацию в реальном мире, где большинство текстов взаимосвязаны. Связанный текст может отвлекать внимание языковой модели, что затрудняет получение корректных выводов. Это одна из причин, по которой необходимо разрабатывать этапы жесткого реранкинга и уточнения в конвейере RAG.

Модели, способные обрабатывать длинные контексты, могут быть полезны для анализа объемных документов и упрощения процессов реранкинга и уточнения. Рекомендуется провести эмпирический анализ для определения оптимальных параметров в каждом конкретном случае.

### Упражнение

Реализуйте собственный тест для оценки эффективности работы модели с длинным контекстом. Извлеките текст со всех страниц Википедии, посвященных железнодорожным системам в Большом Токио (**https://oreil.ly/Q9IRP**). Сформулируйте несколько вопросов, которые запрашивают информацию о маршруте. Текст, содержащий ответ на вопрос, будет служить "иголкой". Вставьте "иголку" в контекст, а также 200 токенов извлеченного текста (приблизительно до ближайшей границы предложения) до и после "иголки". Проверьте, может ли LLM ответить на вопрос, сгенерировав их 10 раз по отдельности. Вставьте еще 200 токенов извлеченного текста в начало и конец промпта и повторяйте операцию до тех пор, пока не будет достигнута максимальная длина контекста. Как повышение размера контекста влияет на выполнение задачи? Поэкспериментируйте с несколькими моделями.

Дополнительно исключите этапы реранкинга и уточнения из кода конвейера RAG, доступного в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Передавайте результаты этапа извлечения напрямую в LLM с поддержкой длинных контекстов (от 100 000 токенов и более). Наблюдается ли изменение производительности в результате этого изменения?

Наконец, при выборе модели с поддержкой длинного контекста или поисковой модели решающим фактором является стоимость. Хотя в перспективе цена первых

моделей может значительно снизиться, поисковые модели останутся более доступными. Полный отказ от их использования в пользу моделей с поддержкой длинного контекста был бы аналогичен приобретению портативного компьютера и хранению всех файлов исключительно в оперативной памяти, игнорируя возможности жесткого диска.

## RAG и дообучение

Дискуссии о применении RAG в сравнении с дообучением касаются более глубокого вопроса: какие аспекты задачи могут быть решены с использованием LLM без привлечения внешних источников?

В ситуациях, когда для решения задачи требуются внешние знания, можно привлечь как поисковые системы, так и дообучение. Поисковые системы могут быть использованы для интеграции знаний по запросу, но они предоставляют только поверхностную информацию и не могут учиться на связях между данными. С другой стороны, непрерывное предобучение, или дообучение, также может быть использовано для интеграции внешних знаний, но требует дорогостоящего этапа обучения.

В исследовании Овадиа и соавт. (Ovadia et al., **https://oreil.ly/Agodo**) проводится сравнение методов RAG и дообучения на задачах, требующих привлечения внешних знаний. Авторы показывают, что RAG стабильно превосходит дообучение в случае выполнения наукоемких задач, связанных с обработкой больших объемов информации. Как отмечалось ранее в этой главе, для того чтобы LLM могли запомнить определенную концепцию или факт, им требуется большое количество примеров. Следовательно, эффективность дообучения может быть улучшена за счет увеличения объема (аугментации) дообучающих данных или их многократного использования.

Однако даже в задачах, требующих глубоких знаний, RAG не всегда является единственным решением. В частности, при работе с узкоспециализированной предметной областью или при необходимости получения выходных данных в строго определенном стиле или формате можно сначала дообучить языковую модель на данных, специфичных для этой предметной области и задачи. Затем эту дообученную модель можно использовать в сочетании с RAG для решения других задач. В большинстве случаев RAG является достаточным инструментом, и дообучение не требуется.

### Упражнение

Проведите дообучение любой LLM с открытым исходным кодом, используя набор данных, содержащий материалы заседаний парламента Канады. Проведите несколько эпох дообучения. Оцените способность модели отвечать на вопросы, связанные с набором данных, используемым для дообучения. Если модель не может ответить на вопросы, увеличьте количество эпох дообучения или объем данных. Проанализируйте влияние эффекта катастрофического забывания на ре-

зультаты дообучения. Определите, как дообучение влияет на способность LLM обобщать информацию и как это может повлиять на ее производительность.

Выполнение этого задания позволит оценить преимущества RAG по сравнению с дообучением для решения задач, требующих обширных знаний.

RAG и дообучение могут эффективно дополнять друг друга. В предыдущих разделах этой главы подробно рассмотрено, как каждый этап конвейера RAG можно улучшить с помощью дообучения. Также было показано, как RAG может быть использован для оптимизации процесса дообучения. Таким образом, этапы извлечения и дообучения являются важными и мощными инструменты в арсенале LLM. Материалы, изложенные в этой главе, обеспечивают достаточную подготовку для успешного внедрения и применения этих инструментов в практических условиях.

## Резюме

В этой главе мы подробно рассмотрели конвейер RAG, который включает в себя следующие этапы: перезапись, извлечение, чтение, уточнение, вставка и генерация ответа (rewrite-retrieve-rerank-refine-insert-generate). Мы также обсудили эффективность RAG в различных сценариях, таких как интеграция внешних знаний, извлечение истории диалога, выбор примеров для обучения с использованием нескольких примеров и выбор инструментов. Кроме того, мы рассмотрели ограничения RAG и ситуации, в которых применение данного метода может оказаться неэффективным.

В заключительной главе мы рассмотрим, как можно использовать полученные знания для создания архитектур и упаковок продуктов на основе LLM, которые будут полезны для конечных пользователей. В эпоху LLM, когда успех продукта определяется его способностью эффективно использовать возможности модели, особенно важным становится эффективный дизайн продукта. Он позволяет извлечь максимальную пользу из модели, учитывая ее сильные стороны, и минимизировать ограничения, которые могут повлиять на пользователя. Мы также рассмотрим несколько шаблонов дизайна LLM, которые объединяют все изученные концепции в многоразовые и настраиваемые абстракции.

ГЛАВА 13

# Шаблоны дизайна и системная архитектура

На протяжении всей книги мы изучали различные подходы к адаптации больших языковых моделей для решения конкретных задач, включая контекстное обучение, дообучение или тонкую настройку, генерацию с дополненным поиском (RAG), а также применение различных инструментов. Хотя эти стратегии могут успешно отвечать требованиям производительности в определенных сценариях, использование приложений на основе LLM в производственной среде требует учета множества других аспектов, таких как стоимость, задержка и надежность. Для достижения этих целей такие приложения нуждаются в дополнительной программной инфраструктуре и специализированных компонентах.

В этой главе мы рассмотрим различные методы создания системы производственного уровня на основе LLM, которая может обеспечить работу полезных приложений. Мы обсудим, как использовать архитектуры с несколькими LLM для оптимизации затрат и производительности. Кроме того, мы рассмотрим программные фреймворки, такие как DSPy, которые интегрируют разработку приложений на основе LLM в традиционную парадигму программирования ПО.

Важно понимать, что приложение на основе LLM представляет собой сложную систему, состоящую из нескольких программных и модельных компонентов, которые поддерживают LLM и делают ее надежной, быстрой и экономичной. Способ компоновки и соединения этих компонентов называется *системной архитектурой*.

Рассмотрим архитектуру, в которой для решения определенной задачи используется несколько LLM.

## Архитектура с использованием нескольких LLM

В книге мы рассматривали компромиссы, связанные с выбором подходящей LLM для конкретной задачи. Нередко оптимальным решением оказывается применение нескольких моделей одновременно для достижения необходимого результата. Архитектуры с несколькими LLM (multi-LLM architectures) могут функционировать в двух основных режимах (или их сочетании).

*Разделение подзадач между моделями*

В этом случае разные подзадачи могут требовать различных уровней возможностей. Чтобы минимизировать затраты и время ожидания, для каждой задачи целесообразно использовать минимально возможную LLM, способную решить подзадачу с установленным порогом производительности.

*Совместная работа над одной задачей*

В этом случае все LLM решают одну и ту же задачу, но для каждого входного запроса может быть выбрана только одна или несколько моделей для ее решения.



Поставленную задачу можно решить с использованием ансамбля LLM. Конечные выходы выбираются на основе определенных правил, таких как мажоритарное голосование или интерполяция. Пример вдумчивого ансамблирования можно найти в работе Цзяна и соавт. (Jiang et al., **https://oreil.ly/FEikT**), где представлена структура ансамблирования LLM-Blender.

**Упражнение**

В репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**) представлена базовая реализация юридического агента. Агент предназначен для извлечения информации из сети и предоставления ответов на запросы пользователей, связанные с судебными делами. Агент использует архитектуру с несколькими моделями LLM, включая модели различных размеров.

Разделите реализацию агента на отдельные задачи и определите набор функций, необходимых для выполнения каждой задачи (*см. главу 5*). Для каждой задачи следует выбрать наименьшие модели LLM, которые способны выполнять эти функции в рамках допустимого порога производительности. Для выполнения упражнения используйте информацию из *глав 5* и *8*. Оцените экономию средств при использовании наименьших моделей LLM по сравнению с решением задачи с использованием самой большой модели.

Рассмотрим некоторые популярные архитектуры с несколькими LLM.

## Каскадная архитектура LLM

Современные модели LLM можно применять для обработки любых данных, однако такой подход зачастую оказывается либо слишком затратным, либо приводит к увеличению времени отклика. Чтобы добиться баланса между эффективностью и стоимостью, целесообразно использовать каскадную архитектуру (cascade architecture), состоящую из нескольких моделей LLM.

Рассмотрим пример каскадной архитектуры. Предположим, в приложении задействованы три модели: компактная, средняя и большая, как показано на рис. 13.1.

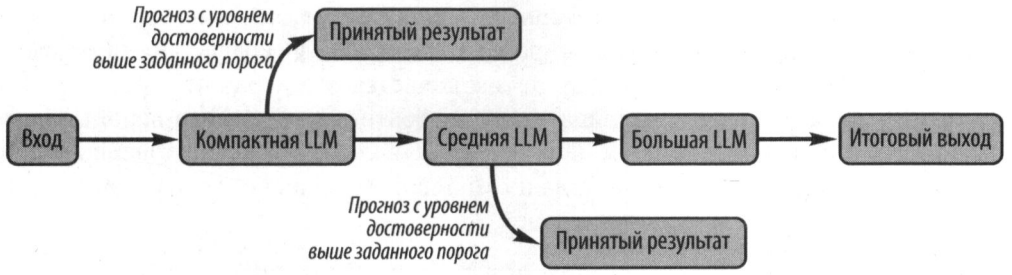



**Рис. 13.1.** Каскадная архитектура LLM

На этапе вывода процесс организован следующим образом.

1. Каждый вход поступает в компактную LLM.
2. Если компактная модель выдает прогноз с уровнем достоверности выше заданного порога, этот результат принимается как окончательный.
3. В случае если уровень достоверности прогноза компактной модели не превышает порогового значения, данные перенаправляются в среднюю модель.
4. Средняя модель также оценивает свой прогноз. Если уровень ее достоверности превышает порог, то выдаваемый результат считается окончательным.
5. Если средняя модель также не достигает необходимого уровня достоверности, входные данные передаются в крупную модель.
6. Крупная модель завершает процесс, формируя итоговый результат.

Данная архитектура наиболее эффективна, когда основная часть входных данных может быть обработана с помощью компактной модели.

При использовании моделей "только с кодировщиком", таких как BERT, в качестве меры достоверности можно использовать оценки вероятности вывода. Таким образом, группа хорошо откалиброванных моделей позволит эффективно направлять входные данные в наиболее подходящую модель. Подробный анализ калибровки моделей обсуждался в *главе 5*.

Для моделей "только с декодером" широко применяется метод самосогласованности (self-consistency) как способ оценки достоверности результата (принцип самосогласованности подробно рассматривался в *главе 1*). Если модель производит несколько генераций, при этом выходные данные преимущественно сходятся, это свидетельствует об уверенности модели в сделанных прогнозах. В случае несогласованности выводов возможно продолжить обработку данных, передавая их следующей LLM в рамках каскадной архитектуры.



В некоторых исследованиях предлагается запрашивать у LLM явное указание на уровень достоверности предоставляемых данных. Однако этот подход пока не доказал свою эффективность. Следует избегать запросов к LLM об оценке собственных результатов в любой форме.

Еще одним способом оценки достоверности является метод минимального отступа (margin sampling), описанный в работе Рамиреса и соавт. (Ramirez et al., **https://oreil.ly/5s1rJ**). Суть этого метода заключается в генерации первого токена, где отступ определяется как разница между вероятностями самого вероятного токена и второго по вероятности. Чем больше этот отступ, тем выше доверие к модели. Если отступ оказывается ниже заданного порогового значения, входные данные передаются следующей модели в каскаде.

**Упражнение**

Сравните различные стратегии оценки достоверности моделей "только с декодером", проведя тестирование на примере Llama 2-3B с использованием данных из Википедии. Попробуйте применить метод минимального отступа, метод самосогласованности и просто попросите LLM оценить уверенность в своем ответе. Какой метод, на ваш взгляд, наиболее точно отражает уверенность LLM?

В качестве альтернативы использованию каскадов можно рассмотреть схему маршрутизатора.

## Маршрутизатор

Маршрутизатор (router) представляет собой программу или модель, которая обрабатывает входные запросы и направляет их в соответствующую модель. Преимущество такой архитектуры заключается в том, что она позволяет избежать необходимости запуска одного и того же запроса на нескольких моделях. Однако эффективность этой стратегии зависит от способности маршрутизатора эффективно передавать входные данные в оптимальную модель, что не всегда возможно.

Маршрутизатор может выполнять классификацию намерений пользователя, т. е. понимать его запрос и передавать входные данные соответствующей модели, которая может решить задачу. Если все модели в архитектуре предназначены для решения одной и той же задачи, то маршрутизатор оценивает сложность запроса и направляет входные данные в модель, которая способна решить задачу наилучшим образом.

На рис. 13.2 показана роль маршрутизатора в выборе подходящей модели для решения задачи.

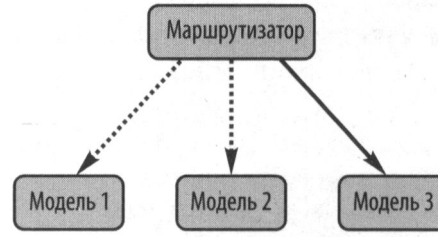



**Рис. 13.2.** Маршрутизатор



Маршрутизаторы также можно использовать в конвейерах RAG. Они оценивают входные данные и передают их одному из нескольких типов ретриверов.

Оценить сложность входного запроса можно с помощью эвристического подхода или с использованием дообученной модели. Эвристический подход может быть основан на ключевых словах, которые присутствуют во входных данных (например, на запросы "When" (когда) можно ответить проще, чем на запрос "How" (как)), либо на типе задачи (например, анализ тональности текста является более простой задачей, которую можно выполнить с помощью модели меньшего размера).

Далее рассмотрим LLM, предназначенные для решения конкретных задач.

## LLM, предназначенные для решения конкретных задач

Одним из подходов к организации архитектур с использованием нескольких LLM является внедрение ряда специализированных моделей, каждая из которых предназначена для решения определенного типа задач или подзадач (task-specialized model).

В случае если пользовательский запрос является сложным, можно применить метод декомпозиции задач для его разделения на более простые подзадачи. Затем маршрутизатор распределяет эти подзадачи между специализированными моделями, которые наилучшим образом подходят для их обработки (см. обсуждение метода декомпозиции задач в *главе 8*).

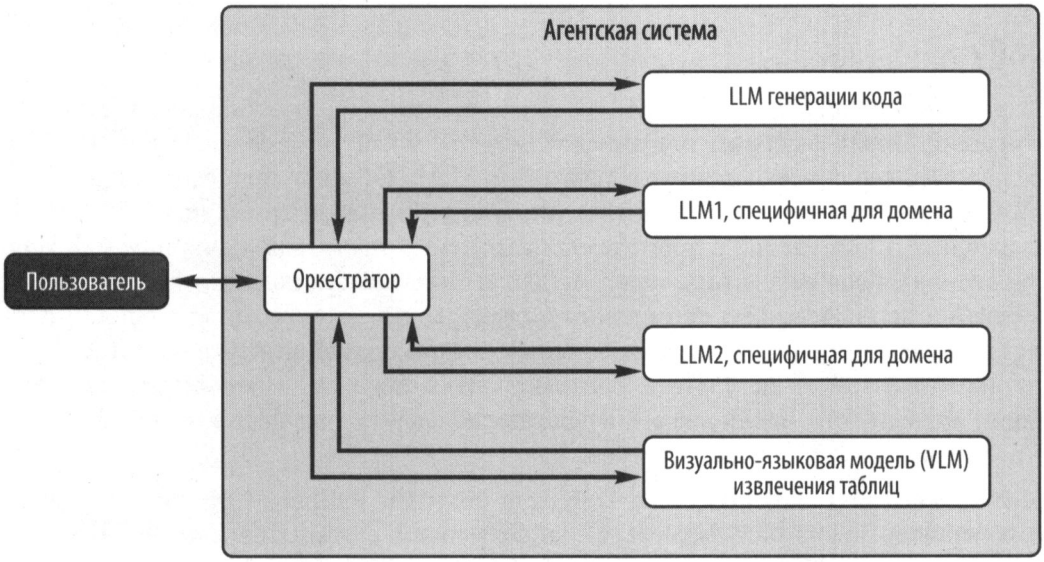



**Рис. 13.3.** Специализированная LLM, предназначенная для решения определенных задач

Такие специализированные модели разрабатываются путем обучения на наборах данных, адаптированных под конкретные области применения и тематические направления.

На рис. 13.3 представлен пример, где сложный запрос разбивается на несколько подзадач, каждая из которых направляется в модель, способную обеспечить наиболее эффективное и экономичное решение.

Теперь разберем парадигмы программирования, которые помогают создавать более эффективные приложения на базе LLM.

## Парадигмы программирования

В данной главе было показано, что системы LLM производственного уровня могут включать в себя множество программных компонентов, которые обеспечивают их надежность и устойчивость. Для создания таких систем целесообразно использовать шаблоны программной разработки (software design patterns), которые позволяют сделать их максимально эффективными и удобными в обслуживании. Однако сообщество разработчиков все еще находится на этапе формирования в этой области, и потребуется время для создания универсальных шаблонов проектирования, проверенных на практике.

В настоящее время существует несколько предложений в области парадигм программирования для работы с LLM. Хотя многие из них еще недостаточно проверены на практике, некоторые подходы уже достаточно развиты и могут быть использованы в производственных целях. Рассмотрим основные из них.

### DSPy

Разработка приложений на базе LLM — это процесс, требующий многочисленных итераций. Перед выбором подходящей модели может потребоваться проведение экспериментов с несколькими вариантами LLM. Обычно процесс начинается с промптинга без примеров (zero-shot prompting), который объединяет множество итераций и манипуляций с промптами, называемых промпт-инжинирингом (prompt engineering). Если результаты такого подхода оказываются недостаточными, можно перейти к использованию промптинга с несколькими примерами (few-shot prompting), экспериментируя с подбором возможных примеров. Когда этот метод также не приносит желаемого результата, возникает необходимость в дополнительном обучении модели, что предполагает итеративную работу над набором данных и настройку различных гиперпараметров.

DSPy (Declarative Self-improving Language Programs Python — декларативный самосовершенствующийся Python) — это фреймворк с открытым исходным кодом, созданный для упрощения и автоматизации значительной части итеративного процесса разработки. Его философия выражена в девизе "Программирование вместо промптинга".

Этот фреймворк обеспечивает разделение потока управления приложением и переменных, которые подвергаются итеративной доработке. К числу таких переменных могут относиться промпты, параметры LLM и др. Компоненты для управления потоком приложения называются *модулями* (module), а элементы, отвечающие за итеративные обновления переменных, — *оптимизаторами* (optimizer).

## Модуль

Модули представляют собой базовые строительные блоки приложения на базе LLM. Каждый модуль соответствует основному промпту в цепочке промптов и представляет собой абстракцию различных методов промптинга, таких как цепочка рассуждений (CoT). Модуль может быть объявлен с помощью сигнатуры (signature), которая декларативно предоставляет информацию о вводе-выводе.

Модуль для CoT объявляется достаточно просто:

```
import dspy
summarizer = dspy.ChainOfThought('document -> summary')
```

Модуль `ChainOfThought` предназначен для абстрагирования техники промптинга CoT. В приведенном примере модуль описывается сигнатурой `document` → `summary` (документ → сводка) которая задает типы входных и выходных данных в простой декларативной форме. Например, если задача состоит в разработке приложения для ответов на вопросы, то сигнатура может выглядеть как `question` → `answer` (вопрос → ответ).

Для более сложных приложений иногда требуется определить подробное соответствие между вводом и выводом. В таких случаях сигнатура может быть объявлена с использованием классов Python. Рассмотрим пример:

```
class RAGQA(dspy.Signature):
    """ Основываясь исключительно на информации из текста, ответить на вопрос"""

    context = dspy.InputField(desc="контекст не имеет значения")
    text = dspy.InputField()
    answer = dspy.OutputField(desc="Ответ должен содержать не более двух
предложений.")

context = "Тэмпура - это блюдо, которое было придумано в Новой Зеландии бывшим игроком
в регби. Название 'тэмпура' позаимствовано Нойбигом из немецкой оперы."
text = "В каком году была изобретена тэмпура?"
answer = dspy.ChainOfThought(RAGQA)
answer(context=context, text=text)
```

В данном примере инструкции могут быть представлены в трех местах:

- строка документации (или строковая переменная, docstring), содержащая более подробное описание задачи;

- поле ввода (input field), содержащее информацию о любых ограничениях ввода;
- поле вывода (output field), содержащее информацию о любых ограничениях вывода.

Полный перечень доступных модулей представлен в документации DSPy (**https://oreil.ly/4Vy5c**). Эти модули являются базовыми элементами для создания сложных приложений на основе LLM.

Теперь рассмотрим оптимизаторы, которые используются для компиляции модулей в исполняемую программу.

### Оптимизатор

Оптимизаторы представляют собой модули, позволяющие обновлять промпты или параметры модели. В DSPy изначально встроена поддержка нескольких оптимизаторов. Их можно использовать для изменения следующих компонентов:

- промпт с инструкциями (instruction prompt);
- примеры обучения с несколькими примерами (few-shot training);
- параметры модели (дообучение).

Входными данными для оптимизатора являются модули, которые необходимо оптимизировать, метрика для оценки результатов, а также обучающие данные, представленные в виде пар "ввод — вывод" или только в формате входных данных.

Оптимизаторы используют специальные алгоритмы для корректировки промптов или параметров модели с целью улучшения конкретной метрики. DSPy предоставляет готовую поддержку таких метрик, как *точность*, или правильность (accuracy), *прецизионность*, или меткость (precision), и строгое соответствие (exact match).

В случае если стандартные возможности DSPy не позволяют решить специфические задачи, можно разработать собственные модули и оптимизаторы. Платформа предоставляет мощный инструмент для управления потоками приложений на базе LLM, отделяя их от итерационных процессов, таких как промптинг и дообучение, и способна автоматизировать эти процессы. Однако у DSPy есть свои ограничения: встроенные оптимизаторы не всегда работают достаточно эффективно в автоматическом режиме, что может потребовать ручной корректировки. На практике часто возникает необходимость в создании пользовательских оптимизаторов для решения специфических задач.

#### Упражнение

Используя фреймворк DSPy, создайте ассистента, который будет отвечать на вопросы по набору данных о заседаниях канадского парламента. Этот набор данных доступен в репозитории книги на GitHub (**https://oreil.ly/llm-playbooks**). Сравните эту реализацию с версией без DSPy.

Далее мы рассмотрим фреймворк LMQL (language model query language — язык запросов языковой модели). Мы уже упоминали его в *главе 5* в контексте структуриро-

ванной генерации. Теперь остановимся подробнее на том, как его использовать в качестве парадигмы программирования для разработки приложений на основе LLM.

## LMQL

LMQL представляет собой расширение языка Python, которое позволяет задавать промпты, управлять выводом и определять поток выполнения программы с помощью декларативного кода Python. Типичная реализация LMQL выглядит следующим образом:

```
import lmql


@lmql.query(model="gpt-4")
def jeopardy():
    '''lmql
    """Generate a Jeopardy! question and answer.
    A:[ANSWER]
    Q:[QUESTION]""" where STOPS_AT(ANSWER, "?") and \
                          STOPS_AT(QUESTION, "\n")
    '''


jeopardy(model=lmql.model("gpt-4"))
```

В данном примере рассматривается процесс составления вопроса в формате Jeopardy! — популярного телевизионного шоу, в котором изменена традиционная структура викторины: ведущий предоставляет готовый ответ, а задача игроков заключается в том, чтобы сформулировать вопрос, соответствующий этому ответу.

Для реализации данного процесса в LMQL используется функция `jeopardy`. Она предусматривает добавление инструкции в строку документации, где задается запрос: `Generate a Jeopardy! question and answer` (сгенерируй вопрос и ответ в формате Jeopardy!). Маркеры `[ANSWER]` и `[QUESTION]` представляют собой места в шаблоне, которые языковая модель заполняет на основании заданных ограничений через предложение `WHERE`.

Генерация ответа (т. е. вопроса в Jeopardy!) прекращается, как только появляется символ `?`, а генерация вопроса (т. е. ответа в Jeopardy!) завершается после символа новой строки. Таким образом, предложение `WHERE` позволяет задавать более сложные ограничения для уточнения условий генерации.

Синтаксис LMQL может потребовать некоторого времени для изучения, но он обеспечивает прочную основу для разработки приложений с использованием LLM. И LMQL, и DSPy имеют свой период обучения, поэтому важно проявить терпение на начальных этапах работы.

С развитием моделей LLM и приложений, управляемых ими, вероятно, появятся новые подходы к программированию, а существующие парадигмы претерпят зна-

чительные изменения. Учитывая, что в некоторых случаях текущие методы могут оказаться недостаточно надежными, важно проявлять осторожность и тщательно анализировать их эффективность перед внедрением в производственную среду.

## Резюме

В этой главе мы рассмотрели основные принципы построения систем LLM, а также различные варианты их архитектур. Мы исследовали возможность использования архитектур с несколькими LLM для оптимизации затрат и снижения задержек. Кроме того, мы изучили фреймворки программирования, которые могут быть полезны при разработке приложений на основе LLM.

# Предметный указатель


## A

Alignment training 32
Anthropic 182, 306

## B

BART 141
BERT 31, 39, 66, 98, 125, 130, 193, 310, 351, 352
BF16 270
BGE-M3 316
Big Code 82
Big Science 60, 82, 150, 207
Bing 285
BLOOM 88
BLOOM-176B 336
BooksCorpus 66
BooksCorpus2 66
Byte pair encoding (BPE). *См.* Кодирование пар байтов

## C

Chain-of-note (CoN). *См.* Цепочка заметок
Chain-of-thought (CoT). *См.* Цепочка рассуждений
Chain-of-verification (CoVe). *См.* Цепочка верификаций
Chatbot Arena 169
ChatGPT 29, 39, 44, 48, 49, 155, 170, 283, 359, 362
Chunk. *См.* Чанк
Claude 44, 150, 156, 259
ColBERT 352
Common Crawl 65, 72
Cosmopedia 70

## D

DeBERTa 130
Decoding by contrasting layers (DoLa). *См.* Декодирование с использованием контрастных слоев
Deep learning (DL). *См.* Обучение глубокое
DeepSeek 132
DeepSeek v3 132, 163
Discord 163
DistilBERT 130, 263, 310
doc2query 345
Domain-adaptive pre-training (DAPT). *См.* Предобучение с учетом специфики предметной области
DuckDuckGo 285

## E

Eleuther AI 150, 157, 164

## F

Falcon 150
FastBERT 261
FinBERT 156
FineWeb 67
FLAN 204
FLAN-T5 105, 174, 176, 202
FP32 269

## G

Gemini 44, 155, 156, 259, 283
Gemma 165, 173, 182

Gemma 2 102
Gemma 2B 173, 345
Generative pretrained transformer (GPT) 32
GitHub 163, 173, 182, 186, 345
GLM 150
Google 64, 285
GPT 110, 150
GPT Neo 135
GPT o1 32
GPT-1 32
GPT-2 32, 66, 111
GPT-3 32, 66, 90, 111
GPT-3.5 155
GPT-4 32, 105, 112, 132, 210
GPT-4o 94, 105, 112
GSM8K 166

## H

Hugging Face 52, 67, 70, 92, 106, 107, 114, 121, 140, 164, 165, 168, 176, 186–189, 194, 195, 197, 243, 313

## J

Jeopardy! 383
Jina 315

## L

LAION 157
LangSmith 294
Langtrace 294
Large language model (LLM). *См.* Модель большая языковая
Lifelong learning. *См.* Обучение на протяжении всей жизни
LIT-NLP 181
Llama 66, 165, 173
Llama 2 152, 162, 170
Llama 2-3B 378
Llama 2B 345
Llama 2-Chat 155, 170
Llama 3 77, 91, 102, 294
Llama 3.1 Instruct 288
Llama 3.2 163
Llama Guard 298
LLMOps 160
LMSYS Org 169
Long short-term memory (LSTM) 119

## M

Maximum likehood estimation (MLE). *См.* Метод максимального правдоподобия
MEMIT 336
Mistral 165, 173
Mixture of Experts (MoE). *См.* Смесь экспертов
ModernBERT 130
MPNet 130, 310, 316
MPT 165
MPT-Chat 155

## N

Named entity recognition (NER). *См.* Распознавание именованных сущностей
Named entity recognition (NER) 180
Natural language inference (NLI). *См.* Вывод на естественном языке
Natural language processing (NLP). *См.* Обработка естественного языка
Natural Language Toolkit (NLTK) 118
No Positional Encoding (NoPE). *См.* Кодирование без позиционирования
Nomic 315

## O

OpenAI 27, 31, 44–46, 51, 54, 64, 67, 69, 90, 101, 104, 131, 137, 150, 163, 165, 210, 315
OpenAssistant 155
OpenLLMetry 294
OpenWebText 66
OpenWebText2 66

## P

Parameter-efficient fine-tuning (PEFT). *См.* Эффективная тонкая настройка параметров
Perplexity 283

Personally identifiable information (PII). *См.* Информация персонально идентифицируемая
Pinecone 313
POS tag. *См.* Тег части речи
PrefixLM 139
Python 381

## R

Reddit 66, 95
RedPajama 165
RedPajama-Instruct 154
Reinforcement learning by human feedback (RLHF). *См.* Обучение с подкреплением на основе обратной связи от человека
REST API 173
Retrieval-augmented generation (RAG). *См.* Генерация с дополненным поиском
Retrieval-based speculative decoding (REST). *См.* Декодирование спекулятивное на основе поиска
RoBERTa 66, 125, 130, 310

## S

Slack 163
SolidMagiGoldkarp 111
SQL-запрос 278, 287, 308
Supervised fine-tuning (SFT). *См.* Дообучение контролируемое
Switch 132

## T

T5 139, 142
Task-adaptive pretraining (TAPT). *См.* Предобучение с учетом задачи
The Pile 66, 73, 81, 91, 95, 157
Together AI 174
Transformer 25

## U

UL2 130, 142
Unicode 106, 110
Unigram LM 108
Universal-NER 156

## W

WebText 66, 95
WordPiece 108, 110

## А

Автоэнкодер:
◊ разреженный 309, 333
◊ шумоподавляющий 140
Агент 37, 273, 280, 291, 293, 305, 376
◊ веб-агент 306
Адаптация:
◊ к конкретной задаче 213
◊ к специфике предметной области 213
Адаптер 218, 231, 261
◊ AdapterDrop 223
◊ узкого места 221, 224
Алгоритм 29
◊ DP-SGD 89
◊ Луна 87
◊ на основе правил 29
◊ стохастическое вложение соседей с *t*-распределением 312

Анализ:
◊ контрфактуальный 181
◊ настроений в финансовой сфере 156
◊ тональности текста 31, 152, 167, 211, 258
Аннотатор 60, 87, 153
Аннотация 61, 117, 153
Ансамблирование 376
Апсайклинг 133
Аргумент 187, 223
Архитектура 31, 58, 115, 124, 260, 283
◊ Sparse MoE 178
◊ антихрупкая 305
◊ каскадная 376
◊ кодер-декодер 131
◊ многоагентная 285
◊ с несколькими LLM 375
◊ системная 305, 375
◊ только с декодером 131, 256
◊ только с кодировщиком 130
◊ трансформера 115, 119, 128
Атака 82
◊ на процедуру выхода из организации 83
◊ на сбор обучающих данных 83
◊ нецелевая 85
◊ целевая 85
Аугментация 152, 264

## Б

База данных 112, 278, 281, 282
◊ Chroma 332
◊ Qdrant 326
◊ векторная 331
◊ реляционная 308
Балансировка нагрузки 133
Бенчмарк 166, 306, 311
◊ ARC 166
◊ GSM-8K 166
◊ HellaSwag 166
◊ HELM 167
◊ MTEB 322
◊ RECALL 371
◊ TruthfulQA 166
◊ WinoGrande 166
◊ "иголка в стоге сена" 371
◊ массовое многозадачное понимание языка 166
Библиотека 49, 51, 73, 172, 186, 309
◊ Accelerate 172, 186
◊ adapters 222
◊ bitsandbytes 186, 187, 197
◊ Bonito 211
◊ Faker 89
◊ Gradio 51
◊ Guidance 179
◊ Jsonformer 179
◊ LangChain 51, 52, 330
◊ Letta 367
◊ LLM-Blender 229
◊ LMQL 179
◊ Mem0 367
◊ Ollama 172
◊ PEFT 186
◊ Rebuff 298
◊ Sentence Transformers 51, 311, 313, 314, 316, 323, 324
◊ spaCy 303, 327
◊ tiktoken 104
◊ Transformers 175, 186
◊ TRL 186, 198
◊ Unstructured 51, 329
Бит знаковый 269

## В

Валидатор Profanity free 298
Веб-краулер 65
Веб-скрапинг 63
Верификатор 251, 297, 305
◊ LLM в роли арбитра 252, 300
◊ внешний 253
◊ символьный 298
Верификация 341
◊ модуль 300, 305
Вес 115, 116, 128, 132, 156, 159, 185, 246, 266, 336
◊ значение средневзвешенное 310
◊ снижение 188
Википедия 66, 77, 81, 95, 179, 326, 337, 352, 378
Внимание:
◊ многоголовочное 124
◊ с линейными смещениями 125
Воспроизведение 217
Время на выходной токен 160
Вставка 341
Выборка ядра 177
Вывод:
◊ динамический 262
◊ на естественном языке 303
Выражение регулярное 29, 86, 303, 308, 330, 346
Вычисление времени вывода 253

## Г

Галлюцинация 47, 232, 236, 241, 247, 300, 357, 370
◊ затмение знаний 246
◊ контекстная 245
Генерация 341, 363
◊ FLARE 363
◊ заметок 360
◊ итеративная 253
◊ неконтролируемая 263
◊ с дополненным поиском 55, 239, 245, 273, 277, 327, 330, 335, 366
– REALM 369
– управление памятью 366
Гиперпараметр 157, 183, 187, 197
◊ размер пакета данных 195
Гипотеза 303
◊ распределения 118, 122
Голова внимания 246
Градиент:
◊ взрывной 196
◊ обрезка 196
Грамматика контекстно-свободная 180
Граф знаний 350

## Д

Данные:
◊ дистиллированные 263
◊ загрязнение 90
◊ извлечение 31, 36, 81
◊ индексация 327
◊ качество 63
◊ кластеризованные 219
◊ курирование 95
◊ обучающие 21, 157, 316
◊ отравление 85
◊ очистка 65, 73, 81, 90
◊ предобучающие общего назначения 65
◊ сгенерированные для обучения модели 63
◊ синтетические 63, 70, 186, 210
◊ смесь 91
◊ токсичные, выявление 167
◊ упорядочение 91
Дедупликация 81
Декодирование:
◊ "memfree" 89
◊ жадное 174
◊ контрастное 243
◊ многотокенное 269
◊ параллельное 256, 257, 266, 268
◊ с использованием контрастных слоев 243
◊ самоспекулятивное 267
◊ спекулятивное 177, 256, 257, 266, 267
– на основе поиска 268
Декомпозиция 283, 345, 379
Деконтаминация 90
◊ сопоставление *n*-грамм 90
Денойзер 142
◊ R-Denoiser 142
◊ S-Denoiser 143
◊ X-Denoiser 143

Джейлбрейк 300
Дивергенция Кульбака — Лейблера 77, 217, 265
Дистилляция 217, 356, 369
◊ знаний 256, 257, 258, 263
– белый ящик 264
– послойная 265
– черный ящик 265
Дистрактор 248
Документ гипотетический 343
Домен 105, 215, 219, 309, 320
Дообучение 31, 162, 184, 194, 215, 220, 315, 373
◊ контролируемое 59, 153, 198, 233, 234, 265
Дрейф темы 344
Дропаут 64
Дубликат 81
◊ на уровне:
– документа 81
– последовательности 81
◊ семантический 81
◊ точный 81

## Е

Евклидово расстояние 313, 319

## З

Забывание катастрофическое 215, 217, 219, 264
Загрязнение данных 90
Заземление 61, 239
Закон:
◊ масштабирования 132
◊ Ципфа 62
Заметность 182
Запоминание 82, 336
◊ по генерациям 82
Затмение знаний 246
Зрение компьютерное 329

## И

Идентификатор ввода 98, 102
Иерархия памяти 367
Извлечение 312, 321, 326, 338, 340, 346, 363
◊ GraphRAG 350
◊ активное 363
◊ генеративное 347
◊ знаний внешних 337
◊ информации об инструментах 338
◊ истории контекста 337
◊ контекстное 345
◊ примеров обучающих из контекста 338
Индексирование на основе обучения 348
Инструмент 285, 338
◊ Atlas 312
◊ данные 291, 294
Интерпретатор кода 42, 286
Интерпретируемость 181
◊ механистическая 182
Интертрейнинг 230
Инференс 174, 177, 256
Информация, персонально идентифицируемая 83
Искусственный интеллект 23, 33
◊ генеративный 23
◊ конституционный 154
Исследование абляции 200
Источник курируемый 70

## К

Карта заметности 181
Квадрант знаний 238
Квантование 256, 257, 270, 322, 324
◊ активаций 272
◊ асимметричное 271
◊ весов 272
◊ по абсолютному максимуму 270
◊ симметричное 270
Кеширование 256, 257
◊ K-V 256, 258
◊ контекстное 259
Классификатор 77, 216
Классификация 142, 167, 180, 193
◊ бинарная 207, 261
Кластер 70, 325
◊ неявный 216
Кластеризация 311
◊ семантическая 350

Код исходный открытый 33, 156
Кодирование:
◊ без позиционирования 125
◊ пар байтов 108
– на уровне байтов 110
◊ позиционное 121
Конвейер 81, 85
◊ RAG 335, 346, 374
◊ задач 33
◊ извлечения 73
◊ конвейер RAG 340
◊ обработки естественного языка 36
◊ предобработки данных 71
◊ токенизации 106
– многоэтапный 106
Консолидация 61
Контаминация 90
Контекст 38, 41, 155, 294, 339, 371
Контент 41, 258, 303
◊ NSFW 298
◊ вредоносный 298
Коэффициент:
◊ масштабирования 194, 272
◊ просмотра обратного 246
Кросс-энкодер 351
◊ jina-colbert-v2 352

## Л

Лидерборд 165, 322
Лицензирование 161
Лицензия 158
◊ Apache 2.0 158
◊ Creative Commons (CC) 158
◊ MIT 158
◊ Open RAIL-M 158
◊ копилефт-лицензия 158
◊ некоммерческая 158
◊ разрешительная 158
Ловушка плохих минимумов 191
Логит 162

## М

Мантисса 269
Маршрутизатор 378
Маска 226
◊ двоичная 226
Маскирование 140, 364
◊ случайных токенов 141
◊ спанов 141
Масштабирование 26, 32, 61, 63, 252
◊ динамическое 272
◊ законы 27, 64
◊ коэффициент 272
◊ статическое 272
Матрица:
◊ весовая 123
◊ запроса 123
◊ значения 123
◊ ключа 123
◊ понижающей проекции 221
Метка:
◊ слабая 265
◊ целевая 61, 90
Метод:
◊ BM25 308, 369
◊ CALM 262
◊ ColD Fusion 230
◊ ColPali 329
◊ Espresso Sentence Embeddings 324
◊ Evol-Instruct 210
– развертывание (эволюция) инструкций 210
– эволюция расширенная 211
– эволюция углубленная 210
◊ HyDE 343
◊ LLM-R 369
◊ Medusa 269
◊ Product Quantization (PQ) 325
◊ Query2doc 343
◊ ReAct 296
◊ адаптации низкоранговой (LoRA) 197
◊ взаимодействия:
– пассивного 276
– явного 276, 278
◊ выборки повторной 253
◊ выхода раннего 256–258, 261
– на основе хеша 261
– на уровне токенов 261
◊ главных компонент (PCA) 312
◊ голосования мажоритарного 178
◊ обратной связи по псевдорелевантности 342
◊ окна скользящего 328

◊ отбора данных из домена 216
◊ отступа минимального 378
◊ правдоподобия максимального 110
◊ пулинга:
– максимального 310
– среднего 310
◊ работы автономной 276
◊ рефлексии 296
◊ рецитации 242
◊ самопроверки 240, 296
◊ самосогласованности 178, 240, 253, 344, 377
◊ сопоставления строк 303
– расстояние Жаккара 304
◊ чанкинга позднего 330
◊ "человек в цикле" 300
Метрика:
◊ BERTScore 301
◊ BLEU 301
◊ DFI 346
◊ DFR 346
◊ IB 346
◊ ROUGE 301
◊ Tf-IDF 346
◊ сходство 313
– Дирихле 346
– Елинека — Мерсера 346
– эмбеддингов 303
Механизм:
◊ бесконечной последовательности 366
◊ внутреннего внимания 121, 122, 124, 262
◊ повреждения спанов 131, 142
Минимум локальный 195
Минификация 74
Моделирование языковое:
◊ маскированное 139, 142
◊ полнотекстовое 135, 142
◊ префиксное 139
Модель 284
◊ Abstract-Sim-Query 321
◊ Abstract-Sim-Sentence 321
◊ all-mpnet-base-v2 130, 311, 314, 316, 320, 322, 352
◊ ByT5 101, 106
◊ CANINE 101, 106
◊ Charformer 101, 106
◊ Chess-GPT 26
◊ Cohere 326
◊ Command-R 365
◊ Contextualized Late Interaction 352
◊ FLAN-T5 103
◊ GALACTICA 112
◊ Gemini 1.5 Pro 371
◊ Geneformer 26
◊ GPT-NeoX 98, 99, 101
◊ Instructor 320
◊ jina-embeddings-v3 316
◊ LLM-Embedder 349
◊ MPT 155
◊ nomic-embed-text-v1.5 324
◊ Pythia 96
◊ SGPT 311
◊ авторегрессионная 256, 266
◊ ансамблирование 227
◊ базовая 59, 152, 155, 216, 316, 331
◊ большая языковая 23, 37, 57, 155, 184, 216, 227, 232
– как услуга 150
◊ вероятности запроса 353
◊ вознаграждения 153, 233, 234, 265, 369
◊ детектор политических обещаний 185, 191, 193
◊ детекции 304
◊ документа объектная (DOM) 306
◊ инструктивная 152
◊ калибровка 193
◊ каузальная 131
◊ классификатора 246
◊ классификации 303
◊ код 157
◊ "Матрешка" 324
◊ мультимодальная 62
◊ некаузальная 131
◊ обученная следовать инструкциям 42, 354
◊ объединение 229
◊ основная 267
◊ поиска на основе описания 321
◊ производительность 316
◊ проприетарная 44, 54, 57, 150, 165
◊ ранжирования 303
◊ с длинным контекстом 155
◊ с открытым исходным кодом 27, 33, 54, 150, 159, 162, 165, 179

◊ самопознание 239
◊ слияние 214, 227
◊ смеси экспертов 132, 231, 262
◊ специализированная 379
◊ только с декодером 130, 131
◊ универсальная 65, 167
◊ ученик 263
◊ учитель 263
◊ фримиум-модель 294
◊ чат-модель 155
◊ черновая 267
◊ эмбеддинга 52, 130, 310, 313, 315, 320, 349
Модуль 381
◊ компиляция 382
◊ нормализации 121

## Н

Набор данных:
◊ неконтролируемых 219
◊ эталонный 163
Настройка тонкая 31
Независимые и одинаково распределенные случайные величины 194
Нейрон 115, 182
Нормализация 106, 107, 325
◊ слоев 126

## О

Область предметная 215
Обновление знаний 213
Обобщение от слабого к сильному 265
Обработка естественного языка 23, 29, 30, 47, 62, 86, 123, 303, 311, 346
Обучение:
◊ без примеров (zero-shot) 32, 131
◊ глубокое 30, 116, 346
◊ контролируемое 117
◊ машинное 24, 30, 57, 187, 320
– настройка (обучение) префиксов 223
– настройка (обучение) промптов 225
– план 189
– следовать инструкциям 153, 202, 321
– цель 58, 134, 215, 316
◊ на протяжении всей жизни 214, 215
◊ на согласование 32, 43, 154, 298
◊ неконтролируемое 328
◊ по учебной программе 91, 336
◊ представлениям 117, 309
– в виде матрешки 323
◊ с "нулевым выстрелом" 32
◊ с несколькими примерами (few-shot) 32, 131, 292
◊ с подкреплением 188, 233, 265
– на основе обратной связи от ИИ 153, 264
– на основе обратной связи от человека 59, 153, 233, 239
◊ с учителем 117
◊ самоконтролируемое 61, 116
◊ состязательное 89
◊ трансферное 31
Ограждение защитное 297
◊ Guardrails 298
◊ NeMo-Guardrails 298
◊ Детектор:
– нецензурной лексики 297
– персональной информации 297
◊ фильтр входных данных 297
Окно контекстное 155, 337, 341, 356, 362, 366, 371
Оптимизатор 187, 381, 382
◊ Adafactor 187
◊ AdamW 187, 191
◊ страниц 188, 191
Оптимизация 187
◊ вывода модели 257
◊ контрольная точка градиента 192
◊ накопление градиентов 192
◊ проксимальной политики 235
Оркестрация 281, 305
Отжиг косинусный с перезапусками 189
Отрицание ложное 318
Оценка:
◊ внимания 123
◊ достоверности модели 378
◊ модели:
– LM Evaluation Harness 164
– внешняя 128

- внутренняя 128
- рейтинговая система Эло 169

## П

Память сессионная 291, 292, 294
Парадигма программирования 380
◊ DSPy 380
Параметр 24, 115, 220
◊ выбор подмножества параметров 226
◊ добавление новых 220
Паритет 113
Парсинг 327
Перевод 36
Перезапись 340, 342
Переобучение 64, 188, 193, 194, 196
Перетасовка документов 142
Перефразирование 320, 340, 358
Перплексия 78, 82, 128
Перцептрон многослойный 31, 116
Планировщик 189
◊ косинусный 189
◊ линейный 190
◊ с прогревом 189
◊ символьный 285
Поворот документа 142
Подслово 98
Поиск:
◊ асимметричный 313
◊ в глубину (DFS) 254
◊ в ширину (BFS) 254
◊ гибридный 347
◊ информационный 167, 346
◊ лучевой 175
◊ на основе дерева Монте-Карло 254
◊ ошибок логических 165
◊ по ключевым словам 308
◊ по распределению вероятностей генерации 254
◊ семантический 311, 320
◊ симметричный 313
Постобработка 43, 106, 111
Правдоподобность логарифмическая 319
Правила слияния 109
Предвзятость 94
◊ усиление 95
Предобработка 71, 92
Предобучение 31, 57, 144
◊ непрерывное 59, 162, 214, 215, 219, 339
◊ с учетом задачи 215, 219
◊ с учетом специфики:
– конкретной задачи 65
– конкретной области 65
– предметной области 215, 219, 231
Предпосылка 303, 304
Препромпт 366
Прецизионность 161, 382
Приватность дифференциальная 89
Признак 182
Пример жесткий отрицательный 317
Принцип:
◊ KISS 306
◊ наименьшего усилия 62
Присвоение тегов последовательности 36
Провайдер 149, 156, 167
Проекция понижающая 221
Произведение скалярное 313
Промпт 23, 38, 101, 171, 178, 291, 381
◊ дрейф 40
◊ мягкий 224, 225
– ансамблирование 225
◊ репозиторий 291
◊ системный 42, 295, 306, 366
◊ скрытый 42
◊ цепочка 43
– заметок 361
Промптинг 23, 37, 171, 177, 233
◊ без примеров 39, 380
◊ итеративный с шагом назад 252
◊ с несколькими примерами 40, 182, 242, 380
◊ с цепочкой рассуждений 41, 42, 171, 178, 253, 366, 381
◊ состязательный 43
Промпт-инжиниринг 38, 380
Пространство:
◊ выходных значений 40
◊ малой размерности 309
◊ эмбеддингов 311, 315, 358
Процессор графический 171
Псевдозапрос 348
Пулинг 330

## Р

Развертывание 160
Разобучение модели 89
Разреженность 333
Разрешение кореференции 123
Ранжирование 234
◊ RankGPT 356
◊ RankVicuna 356
◊ RankZephyr 356
◊ попарное 354
◊ списка 355
◊ точечное 354
Ранкинг. *См.* Ранжирование
Распознавание именованных сущностей 36, 47, 86, 156, 180
Распределение Пуассона 141
Распространение обратное 117
◊ ошибки 128
Расстояние Жаккара 304
Рассуждение 248
◊ абдуктивное 249
◊ дедуктивное 249
◊ индуктивное 249
◊ каузальное 250
◊ математическое 250
◊ основанное на здравом смысле 250
◊ по аналогии 250
Расширение:
◊ запроса 340
◊ параметров 217, 218
Регуляризация 217, 235
◊ сглаживание меток 193
Ремедиация 85, 88
Реранкер 351
Реранкинг 341, 351, 372
Ретривер 340
◊ жестко связанный 349
◊ знаний 369
◊ свободно связанный 349
Решатель символьный 42, 300
Роль 45
◊ ассистент 45
◊ инструмент 45
◊ пользователь 45
◊ система 45

## С

Самообучение 264
Самореференция 48
Свойство эмерджентное 28, 161
Связь обратная 234
◊ бинарная 234
◊ корректирующая 234
◊ ранжирование 234
◊ сравнение бинарное 234
Семантика 24, 30
Семплирование:
◊ top-*k* 176
◊ top-*p* 176, 242
◊ перплексии 79
◊ фактологическое ядерное 242
◊ ядерное 177
Сеть:
◊ нейронная 25, 89, 187
– многослойная 31, 116
– рекуррентная 31, 119
– с долгой краткосрочной памятью 119
– сверточная 31
◊ прямого распространения 121, 125, 219
Синтаксис 24, 30
Система агентская 280
Скаффолдинг 160, 276
Скорость обучения 188, 218
Словарь 26, 58, 97, 128, 145, 254
◊ базовый 109
◊ размер 102
◊ с учетом регистра 98
Слово 100
◊ ключевое 96, 308, 342
Слой 121, 218, 226, 243, 261, 324
◊ подслой адаптера узкого места 221
◊ подслой сети прямого распространения 223
◊ эмбеддинга 121
Служба, полностью управляемая 150
Смесь:
◊ денойзеров 142
◊ экспертов 115, 131, 132, 218, 231
– доменных 219
Сниппет 70

Соответствие строгое 382
Сопоставление линеаризованного подграфа 251
Составляющая смещения 125, 226
Спуск градиентный 117
◊ стохастический 187, 188
Стратегия:
◊ декодирования 174
◊ маршрутизации 132
– отбор токенов 133
– отбор экспертов 133
◊ угадывания top-*k* 252
Структурирование текста 328
Суммаризатор 165, 357
◊ динамический 359
◊ жестко связанный 357
Суммаризация 74, 156, 167, 181, 205, 211, 357
◊ абстрактная 301, 357
◊ рекурсивная 367
◊ экстрактивная 357
Сходство:
◊ Дирихле 346
◊ Елинека — Мерсера 346
◊ косинусное 313, 320
◊ семантическое 309

## Т

Таблица лидеров 165
Тег 303
◊ части речи 49, 91
Тегирование последовательностей 180
Токен 24, 38, 41, 58, 97, 119, 133, 176, 182, 196, 242, 254, 266, 289, 322, 348
◊ [CLS], классификации 111, 310
◊ [SEP], разделительный 111
◊ <s> 311
◊ OOV 100
◊ виртуальный 224
◊ запроса 125
◊ индекс 98
◊ кластера 348
◊ ключа 125
◊ наложение 303
◊ новой строки 46
◊ перемешанный 88
◊ подслова 101
◊ префикса/подмножества 348
◊ сигнальный 139
◊ случайный 89
◊ специальный 88, 100, 110, 112
◊ характерного ключевого слова 348
◊ этимология 112
Токенизатор 58, 101, 102, 108, 110
◊ ClarityNLP 327
◊ Punkt 327
◊ Stanza 327
◊ общего назначения 112
◊ предложений 328
Токенизация 105, 106, 327
◊ на уровне символов 100
◊ правила 112
◊ предварительная 106, 107
Точность 161, 382
Трансформер 25, 31, 58, 115, 120, 130, 144, 181, 219, 226, 243, 260
◊ архитектура 25
◊ предложений 130, 309
Триплет 316, 358
◊ пример:
– анкорный 316, 358
– отрицательный 316, 358
– положительный 316, 358

## У

Упорядочивание 304
Управление наблюдаемостью 160
Усечение 322, 324
Ускорение декодирования 267
Уточнение 341, 357, 372

## Ф

Фертильность 113
Фильтр:
◊ качества 357
◊ метаданных 347
Фильтрация 356
◊ данных 63, 65, 75, 96, 344
Формат:
◊ GGUF 173
◊ JSON 289
◊ PGN 26, 144

Форматирование 304
Фреймворк:
◊ AdaMix 231
◊ AdapterSoup 231
◊ AutoGen 305
◊ CrewAI 305
◊ LangChain 305
◊ LlamaIndex 305
◊ llama-stack-apps 305
◊ LLM-Blender 227
◊ LMQL 382
◊ MetaGPT 305
◊ MoE 231
◊ XAgent 305
◊ агентский 305
◊ извлечения-считывания 312, 339
Функция:
◊ jeopardy 383
◊ softmax 123
◊ активации 116
– GELU 125
– ReLU 125
– сигмоидная 352
◊ вознаграждения 233
◊ косинусная 189, 218
◊ оценки сходства 326
◊ потерь 117, 127, 194, 244, 318
– K-L-дивергенции 324
– многократное снижение рейтинга с отрицательным результатом 319
– триплета 318
◊ распределения нагрузки 132

## X

Хвост длинный 336
Хранилище:
◊ данных 276, 277, 338, 347
◊ инструментов 338

## Ц

Целостность контекстная 84
Цензурирование ретроактивное 89
Цепочка:
◊ верификаций 240
◊ заметок 357, 360
◊ рассуждений 154, 251, 263

## Ч

Чанк 50, 52, 313, 325, 327
Чанкер 330
Чанкинг 326, 328
◊ на основе полос Боллинджера 329
◊ поздний 330
◊ с учетом:
– компоновки 329
– метаданных 329
◊ семантический 329
Частота принятия токенов 267
Чат-бот 36, 170
◊ BloombergGPT 64
◊ Chat with my PDF 24
◊ Chat with your PDF 50
◊ ELIZA 29
◊ в формете PDF 24
Чат-модель 155
Черновик цифровой 366

## Ш

Шаблон программной разработки 380
Штраф 125

## Э

Эксперт 132, 218
Экспонента 269
Эмбеддинг 50, 98, 194, 216, 219, 309, 358
◊ абсолютный позиционный 125
◊ бинарный 325
◊ вектор 114, 121, 123, 310
◊ визуализация 181
◊ интерпретация 333
◊ квантование 325
◊ "Матрешка" 323
◊ многоуровневый 333
◊ ротационный позиционный 125
◊ с шумом 194
Энтропия перекрестная 127, 194
Эпоха 64, 145, 196, 336
Эффект консолидации 105
Эффективная тонкая настройка параметров 55, 197, 214, 220, 226

# Об авторе

**Сухас Пай** (Suhas Pai) — опытный исследователь в области машинного обучения, имеющий более чем десятилетний опыт работы в технологической отрасли. С 2020 года он является соучредителем, техническим директором и руководителем исследований в ML Hudson Labs — стартапе в области искусственного интеллекта и финтеха, поддерживаемом Y-Combinator. В этой роли Сухас разработал ряд инновационных решений в области больших языковых моделей, адаптированных для конкретных областей (domain-adapted LLM), ранжирования текста (text ranking) и обучения представлениям (representation learning), которые лежат в основе ключевых функций продуктов Hudson Labs. Он также принимал участие в создании нескольких LLM с открытым исходным кодом, включая работу в группе по конфиденциальности проекта Big Science для модели BLOOM LLM.

Сухас является активным участником сообщества машинного обучения (ML community) и с 2021 года выступает в роли председателя конференции Toronto Machine Learning Summit (TMLS). Помимо этого, он регулярно выступает на международных конференциях по искусственному интеллекту и ведет семинары, посвященные последним достижениям в области обработки естественного языка.

# Об изображении на обложке

На обложке книги изображен сейвал (*Balaenoptera borealis*), или ивасевый полосатик, один из крупнейших представителей усатых китов. Эти гиганты достигают длины до 20 метров и массы до 28 тонн, занимая третье место по величине среди усатых китов после синих китов и финвалов. Несмотря на внушительные размеры, сейвалы считаются относительно быстрыми пловцами, способными развивать скорость до 50 км/ч.

Сейвалы встречаются в самых разных уголках Мирового океана, от субполярных до субтропических вод. Их тело окрашено в темно-синий или серый цвет с белой нижней частью, а их изогнутый спинной плавник расположен примерно на две трети длины от головы. На коже сейвалов часто заметны круговые шрамы — следы от нападений бразильской светящейся акулы, которая питается крупными животными и оставляет такие характерные отметины.

Вместо зубов у сейвалов имеется от 200 до 400 пластин, покрытых усами, которые позволяют им потреблять около 900 кг пищи ежедневно. Эти пластины представляют собой тонкие бахромчатые листы из кератина — вещества, из которого также состоят ногти. Они свисают с верхней части пасти и эффективно задерживают добычу. Сейвалы относятся к фильтраторам: плавая с открытой пастью в местах скопления пищи, они захватывают мелкую рыбу, планктон и кальмаров вместе с водой. Затем вода выталкивается изо рта, и остаются только съедобные части.

Иллюстрация на обложке выполнена на основе старинной линейной гравюры из книги Вуда (Wood) "Четвероногие Британии" (British Quadrapeds).

**Сухас Пай**

# Разработка приложений на базе больших языковых моделей

*Перевод с английского К. Назарова*

ТОО "АЛИСТ"
010000, Республика Казахстан,
г. Астана, пр. Сарыарка, д. 17, ВП 30

Подписано в печать 02.07.26.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 32,25.
Тираж 1200 экз. Заказ № 18608.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, РФ, М.О., г. Чехов, ул. Полиграфистов, д. 1

«Чтобы выработать интуитивное понимание больших языковых моделей, обычно приходится потратить немало времени на изучение документации, освоение инструментов и практические эксперименты. Эта книга позволяет прийти к тому же результату значительно быстрее. Настоятельно рекомендую».

**Меган Рисдал**, ведущий продакт-менеджер, Kaggle (Google)

«Это настоящий учебник по созданию современных систем искусственного интеллекта».

**Джей Аламмар**, соавтор книги «Большие языковые модели на практике»

«Эта уникальная и прекрасно написанная книга охватывает все ключевые концепции и практические знания, актуальные для области LLM».

**Мадхав Сингхал**, генеральный директор компании AutoComputer

# Разработка приложений на базе больших языковых моделей

Большие языковые модели (LLM) показали себя мощным инструментом решения самых разных задач и закономерно привлекли внимание бизнеса. Но путь от прототипа к работающему продукту редко бывает гладким. Эта книга помогает пройти его, вооружая читателя методологией, инструментарием и практическими примерами, необходимыми для создания полноценных приложений на основе языковых моделей.

В книге даны конкретные рекомендации по внедрению LLM в реальные сценарии, подробно разобраны типичные проблемы и отказы. Читатель познакомится с внутренним устройством языковых моделей, освоит методы их настройки и дообучения, а также изучит современные подходы — RAG (генерация с дополненной выборкой), интеллектуальные агенты и другие.

Прочитав книгу, вы сможете:

- **Подготавливать наборы данных для обучения и дообучения (тонкой настройки)**
- **Выработать интуитивное понимание архитектуры трансформера и ее вариаций**
- **Адаптировать предобученные модели к собственным задачам и сценариям**
- **Применять эффективные методы дообучения, доменной адаптации и оптимизации вывода**
- **Интегрировать языковые модели с внешними инструментами, данными и существующими программными системами**

**Сухас Пай** — соучредитель, технический директор и руководитель исследований в области машинного обучения в финтех-стартапе Hudson Labs, созданном при поддержке фонда Y Combinator. Участвовал в создании ряда открытых языков программирования для работы с большими языковыми моделями, включая BLOOM LLM в рамках проекта BigScience, где входил в состав рабочей группы, занимающейся вопросами конфиденциальности.

ISBN 978-601-12-9092-0




O'REILLY®