Эффективный разговорный ИИ



Создаем чат-ботов, которые действительно работают

Эндрю Фрид
Кари Джейкобс
Энико Рожа

Предисловие Хесуса Мантаса

MANNING

Effective Conversational AI

Chatbots that work

Andrew Freed, Cari Jacobs, Eniko Rozsa
Foreword by Jesús Mantas



MANNING
Shelter Island

Эффективный разговорный ИИ

Создаем чат-ботов, которые действительно работают

Эндрю Фрид, Кари Джейкобс, Энико Рожа
Предисловие Хесуса Мантаса

ПИТЕР®
Санкт-Петербург • Москва • Минск
2026

ББК 32.988.02-018+32.813
УДК 004.738.2+004.89
Ф88

Фрид Эндрю, Джейкобс Кари, Рожа Энико

Ф88 Эффективный разговорный ИИ. Создаем чат-ботов, которые действительно работают. — СПб.: Питер, 2026. — 368 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-4492-1

Новые мощные фреймворки для разработки чат-ботов и модели генеративного ИИ практически сняли ограничения, связанные с некорректным распознаванием намерений пользователя и генерацией бессодержательных ответов. Освойте разработку чат-ботов на основе больших языковых моделей (LLM) и других современных инструментов, а также проектирование разговорных систем, ориентированных на реальный пользовательский опыт.

Книга «Эффективный разговорный ИИ» посвящена созданию надежных и масштабируемых чат-ботов, пригодных для использования в корпоративной среде. Вы узнаете, как уточнять пользовательские намерения с помощью LLM, корректно обрабатывать нестандартные запросы и применять генерацию, дополненную поиском (RAG), чтобы ответы бота всегда оставались актуальными и релевантными. По мере прочтения разберетесь, как построить систему обратной связи для непрерывного улучшения качества диалогов и освоите практики безопасной интеграции генеративного ИИ в существующие архитектуры.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988.02-018+32.813
УДК 004.738.2+004.89

Права на издание получены по соглашению с Manning Publications. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1633436404 англ.

ISBN 978-5-4461-4492-1

Authorized translation of the English edition © 2025 Manning Publications. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.
© Перевод на русский язык ООО «Прогресс книга», 2026
© Издание на русском языке, оформление ООО «Прогресс книга», 2026
© Серия «Библиотека программиста», 2026

Краткое содержание

Часть 1
Как усовершенствовать разговорный ИИ

Глава 1. Как работает разговорный ИИ? 30

Глава 2. Построение разговорного ИИ 54

Глава 3. Планирование улучшений 77

Часть 2
Паттерн: ИИ не понимает пользователя

Глава 4. Понимание потребностей пользователей 116

Глава 5. Улучшаем понимание традиционного ИИ 147

Глава 6. Повышение качества ответов с помощью RAG 179

Глава 7. Дополнение намерений с помощью генеративного ИИ 219

Часть 3
Паттерн: ИИ слишком сложный

Глава 8. Оптимизация сложных сценариев 244

Глава 9. Влияние контекста на адаптивность виртуального помощника 260

Глава 10. Снижение сложности системы с помощью генеративного ИИ 289

Часть 4
Паттерн: разрешение конфликтов

Глава 11. Сокращение числа отказов 310

Глава 12. Резюмирование при передаче диалога оператору 341

Оглавление

От издательства 15
О научном редакторе русского издания 15
Предисловие 16
Введение 18
Благодарности 20
О книге 23
Для кого эта книга 23
Структура книги 23
О коде в книге 25
Форум liveBook 25
Об авторах 26
Иллюстрация на обложке 28

Часть 1
Как усовершенствовать разговорный ИИ

Глава 1. Как работает разговорный ИИ? 30
1.1. Введение в разговорный ИИ 31
1.1.1. Для чего нужен разговорный ИИ? 32
1.1.2. Как работает разговорный ИИ? 34
1.1.3. Как построить разговорный ИИ 35
1.2. Генеративный ИИ в разговорных системах: основные понятия 39
1.2.1. Что такое генеративный ИИ 39
1.2.2. Ограничители генеративного ИИ 40

1.2.3. Эффективное использование генеративного ИИ в разговорных системах ... 42
1.3. Непрерывное улучшение разговорных ИИ ... 45
1.3.1. Зачем нужно непрерывное улучшение ... 45
1.3.2. Цикл непрерывного улучшения ... 46
1.3.3. Объясняем концепцию непрерывного улучшения стейкхолдерам ... 49
1.4. Начало работы ... 52
Итоги ... 53

Глава 2. Построение разговорного ИИ ... 54
2.1. Создание FAQ-бота ... 55
2.1.1. Основа FAQ-бота ... 55
2.1.2. Статический режим вопросов и ответов ... 57
2.1.3. Динамические ответы на вопросы ... 63
2.2. Маршрутизирующие агенты и процессно-ориентированные боты ... 66
2.2.1. Маршрутизирующие агенты ... 66
2.2.2. От маршрутизирующего агента к процессно-ориентированному боту ... 68
2.3. Ответы пользователю с помощью генеративного ИИ ... 71
2.3.1. Интеграция с LLM ... 71
2.3.2. Маршрутизация запросов к LLM ... 74
Итоги ... 76

Глава 3. Планирование улучшений ... 77
3.1. Когда необходимо улучшение ... 78
3.2. Ваша межфункциональная команда ... 80
3.3. Движение к общей цели ... 83
3.3.1. Пересмотр бизнес-целей ... 84
3.3.2. Эффективность ... 87
3.3.3. Охват ... 97
3.4. Выявление и устранение проблем ... 100
3.4.1. Поиск проблем ... 100
3.4.2. Групповой обзор ... 104
3.4.3. Определение критериев приемлемости ... 109

3.5. Разработка и внедрение исправлений ... 112
3.5.1. Планирование спринта ... 112
3.5.2. Повторная оценка ... 112
Итоги ... 113

Часть 2
Паттерн: ИИ не понимает пользователя

Глава 4. Понимание потребностей пользователей ... 116
4.1. Основы понимания ... 117
4.1.1. Последствия слабого понимания ... 117
4.1.2. Причины слабого понимания ... 118
4.1.3. Как обеспечить понимание в традиционном разговорном ИИ? ... 120
4.1.4. Как обеспечить понимание в генеративном ИИ? ... 122
4.2. Как измерить понимание? ... 124
4.2.1. Измерение понимания для традиционного (основанного на классификации) ИИ ... 125
4.2.2. Измерение понимания для генеративного ИИ ... 127
4.2.3. Измерение понимания с помощью обратной связи от пользователей ... 129
4.3. Оценка текущего состояния системы ... 129
4.3.1. Оценка традиционного (основанного на классификации) ИИ-решения ... 129
4.3.2. Оценка генеративного ИИ-решения ... 131
4.4. Извлечение тестовых данных из логов и их подготовка ... 131
4.4.1. Извлечение продакшен-логов ... 132
4.4.2. Рекомендации по отбору кандидатных тестовых высказываний ... 133
4.4.3. Подготовка и очистка данных для использования в итеративных улучшениях ... 138
4.4.4. Процесс разметки ... 139
4.5. Что говорят данные? ... 142
4.5.1. Анализ размеченных логов для традиционного ИИ (классификатора) ... 142
4.5.2. Анализ размеченных логов для генеративного ИИ ... 144

4.5.3. Необходимость итеративного улучшения 144
Итоги 145

Глава 5. Улучшаем понимание традиционного ИИ 147
5.1. План улучшений 148
5.1.1. Определение проблемных паттернов в неправильно интерпретированных высказываниях 148
5.1.2. Поэтапные улучшения 153
5.1.3. С чего начать: выявление самых серьезных проблем 153
5.2. Решение проблемы «намерение неверное» 159
5.2.1. Улучшение показателя полноты для одного намерения 159
5.2.2. Улучшение показателя точности для одного намерения 161
5.2.3. Улучшение оценки F1 для одного намерения 163
5.2.4. Улучшение показателей точности и полноты для нескольких намерений 164
5.3. Решение проблемы «намерение не найдено» 168
5.3.1. Определение высказываний для новых намерений 169
5.3.2. Когда лучше перестать добавлять намерения 173
5.4. Дополнение традиционного ИИ генеративным контентом 175
5.4.1. Сочетание традиционного и генеративного ИИ при обработке одного намерения 176
5.4.2. Промпты для демонстрации понимания 177
Итоги 178

Глава 6. Повышение качества ответов с помощью RAG 179
6.1. За пределами намерений: роль поиска в разговорном ИИ 180
6.1.1. Использование поиска в системах разговорного ИИ 181
6.1.2. Преимущества традиционного поиска 183
6.1.3. Недостатки традиционного поиска 184
6.2. За пределами поиска: генерация ответов с помощью RAG 185
6.2.1. Использование RAG в системах разговорного ИИ 185
6.2.2. Преимущества RAG 187
6.2.3. Сочетание RAG с другими моделями генеративного ИИ 190
6.2.4. Сравнение подходов: намерения, поиск и RAG 191

6.3. Реализация RAG ... 192
6.3.1. Общая схема реализации ... 192
6.3.2. Подготовка базы документов для RAG ... 194
6.4. Дополнительные аспекты реализации RAG ... 197
6.4.1. Почему нельзя просто использовать LLM напрямую? ... 197
6.4.2. Поддержание актуальности и релевантности ответов с помощью RAG ... 198
6.4.3. Как настроить пайплайн загрузки данных? ... 199
6.4.4. Управление задержками ... 204
6.4.5. Когда использовать резервный механизм и когда выполнять поиск ... 205
6.5. Оценка и анализ эффективности RAG ... 206
6.5.1. Метрики индексации ... 207
6.5.2. Метрики поиска ... 209
6.5.3. Метрики генерации ... 211
6.5.4. Сравнение эффективности решений для индексации и эмбеддингов в RAG ... 213
Итоги ... 218

Глава 7. Дополнение намерений с помощью генеративного ИИ ... 219
7.1. Начало работы ... 220
7.1.1. Зачем это нужно: преимущества и недостатки ... 221
7.1.2. Что для этого нужно ... 222
7.1.3. Как использовать дополненные данные ... 223
7.2. Улучшение существующих намерений ... 225
7.2.1. Творческий подход к подбору синонимов ... 226
7.2.2. Генерация новых грамматических конструкций ... 230
7.2.3. Создание намерений на основе вывода LLM ... 233
7.2.4. Получение новых примеров с помощью шаблонов ... 237
7.3. Повышение креативности ... 239
7.3.1. Генерация дополнительных намерений ... 240
7.3.2. Проверка на наличие неоднозначностей ... 240
Итоги ... 242

Часть 3
Паттерн: ИИ слишком сложный

Глава 8. Оптимизация сложных сценариев ... 244
8.1. Бремя сложности ... 245
8.1.1. Влияние сложности на конечного пользователя ... 245
8.1.2. Влияние сложности на бизнес-метрики ... 247
8.1.3. Дополнительные издержки и выгоды от снижения сложности для пользователя ... 249
8.2. Упрощение и оптимизация пути пользователя ... 251
8.2.1. Определение сложных диалоговых сценариев ... 251
8.2.2. Использование уже известной информации о клиенте ... 251
8.2.3. Соответствие ожиданиям пользователя ... 253
8.2.4. Гибкость в ожидаемых ответах пользователя ... 254
8.2.5. Поддержка сценариев самообслуживания с помощью API и бэкенд-процессов ... 257
Итоги ... 259

Глава 9. Влияние контекста на адаптивность виртуального помощника ... 260
9.1. Роль контекста в работе виртуальных помощников ... 261
9.1.1. Влияние контекста на взаимодействие с пользователем ... 263
9.1.2. Что такое контекстная информация? ... 267
9.2. Понятие модальности ... 272
9.2.1. Сравнение модальностей ... 273
9.2.2. Влияние модальности на сценарии виртуальных помощников ... 274
9.2.3. Примеры того, как модальность влияет на пользовательский опыт ... 276
9.2.4. Особенности проектирования голосовых ботов ... 278
9.3. Повышение контекстной осведомленности и улучшение общего пользовательского опыта с помощью RAG ... 280
9.3.1. Проектирование адаптивных потоков с использованием RAG ... 281
9.3.2. Стратегии поиска и генерации контекстно-релевантных ответов ... 283

9.3.3. Сопровождение и обновление адаптивных потоков 285
Итоги 287

Глава 10. Снижение сложности системы с помощью генеративного ИИ 289
10.1. Разработка процессных потоков с поддержкой ИИ 290
10.1.1. Генерация диалоговых потоков с помощью генеративного ИИ 291
10.1.2. Улучшение диалогового потока с помощью генеративного ИИ 295
10.2. Выполнение процессных потоков с поддержкой ИИ 297
10.2.1. Выполнение диалоговых потоков с помощью генеративного ИИ 297
10.2.2. Использование LLM для поиска 300
10.3. Тестирование потоков с помощью ИИ 303
10.3.1. Настройка генеративного ИИ для роли пользователя 304
10.3.2. Настройка тестирования диалога 306
Итоги 308

Часть 4
Паттерн: разрешение конфликтов

Глава 11. Сокращение числа отказов 310
11.1. Что вызывает отказы? 311
11.1.1. Причины мгновенных отказов 311
11.1.2. Причины более поздних отказов 312
11.1.3. Сбор данных об отказе 313
11.2. Сокращение числа мгновенных отказов 316
11.2.1. Начните с хорошего: приветствия и представления 316
11.2.2. Обозначьте возможности и установите ожидания 318
11.2.3. Подкрепляйте мотивацию 319
11.2.4. Задействуйте пользователя 320
11.3. Сокращение числа отказов, возникающих позже 322
11.3.1. Старайтесь понять пользователя 322
11.3.2. Старайтесь быть понятыми 322
11.3.3. Будьте гибкими и готовыми подстроиться 323
11.3.4. Показывайте прогресс 325

11.3.5. Предугадывайте потребности пользователя 325
11.3.6. Будьте вежливы 326
11.4. Удержание пользователя при отказе 327
11.4.1. Начните со сбора данных об отказе 328
11.4.2. Реализация потока удержания при отказе 328
11.5. Улучшение диалога с помощью генеративного ИИ 331
11.5.1. Улучшение сообщений об ошибках с помощью генеративного ИИ 332
11.5.2. Улучшение приветствия с помощью генеративного ИИ 334
11.6. Иногда эскалация необходима 339
Итоги 340

Глава 12. Резюмирование при передаче диалога оператору 341
12.1. Резюмирование: основные сведения 342
12.1.1. Зачем нужно резюмирование 342
12.1.2. Структура эффективного резюме 343
12.2. Подготовка чат-бота к резюмированию 347
12.2.1. Использование встроенных элементов 347
12.2.2. Инструментирование чат-бота для создания транскриптов 349
12.2.3. Инструментирование чат-бота (для сбора данных) 352
12.3. Улучшение резюме с помощью генеративного ИИ 354
12.3.1. Генерация текстового резюме транскрипта с помощью резюмирующих промптов 355
12.3.2. Генерация структурированного резюме транскрипта с помощью экстрактивных промптов 359
Итоги 364

Эндрю: Благодарю мою жену Элиз и сыновей Грега и Джеффа за поддержку, которую они оказывали мне при написании очередной книги!

Кари: Джейсону — за твою нескончаемую поддержку, которая сопровождала меня не только во время написания этой книги, но и на протяжении всей жизни. И моему отцу Джиму.

(Сюрприз! Я написала книгу!)

Энико: Спасибо моей семье, чья непоколебимая вера и поддержка сделали возможным появление этой книги. И моему покойному отцу, талантливому техническому писателю, который проложил для меня путь, — я стою на плечах гиганта, продолжая твое дело.

От издательства

Мы выражаем огромную благодарность клубу рецензентов ИТ-литературы ReadIT Club за помощь в работе над русскоязычным изданием книги и вклад в повышение качества переводной литературы.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

О научном редакторе русского издания

Александр Петраки — главный специалист команды DataOps РСХБ-ИНТЕХ. Занимается проектированием архитектуры высоконагруженных приложений, реализацией DataOps-процессов, создает кластерную платформу категоризации и анонимизации персональных данных с применением LLM-моделей.

Предисловие

Революция искусственного интеллекта сделает с нашим разумом то же, что когда-то сделал рычаг с нашей физической силой. Она изменит мир как на микро-, так и на макроуровне, — начиная с того, как каждый из нас пишет, думает и принимает решения, и заканчивая тем, как крупные компании перестраивают свои внутренние процессы и реорганизуют рабочие места. И одним из самых распространенных способов применения ИИ станут приложения с разговорным интерфейсом.

Разговорный ИИ (conversational AI) — это мощнейший инструмент, позволяющий бизнесу и компаниям быстрее и эффективнее обслуживать своих клиентов. Он расширяет возможности самообслуживания и берет на себя рутинные запросы, освобождая людей для более значимых задач.

Несмотря на широкое распространение разговорных ИИ, существует удивительно мало книг, которые подробно объясняют, как грамотно реализовать такие системы. Именно поэтому я с радостью поддержал своих коллег: Эндрю Фрида, Кари Джейкобс и Энико Рожа — в их стремлении поделиться практическим опытом и представить структурированный подход, способный помочь другим специалистам. В этой книге они собрали и систематизировали наиболее распространенные проблемы, возникающие при работе с разговорными приложениями и интерфейсами, а также предложили практические решения, основанные на разнообразных методах — от data science и генеративного ИИ до принципов проектирования разговорных систем.

Разговорный ИИ редко достигает совершенства сразу после запуска: и это одно из самых распространенных заблуждений среди руководителей, ожидающих «мгновенного эффекта» от внедрения ИИ. На самом деле такие системы требуют надежной платформы данных, продуманной архитектуры с поддержкой безопасности и идентификации, а также внимательного проектирования пользовательского опыта и путей взаимодействия. Все это вы найдете в примерах, приведенных в книге и основанных на реальном опыте авторов по созданию и совершенствованию систем разговорного ИИ.

Книга фокусируется на основных проблемах, с которыми пользователи сталкиваются при взаимодействии с разговорными интерфейсами, и описывает методы их решения. Применяя предложенные техники и лучшие практики, компании смогут создавать более привлекательные, эффективные и надежные системы разговорного ИИ, которые быстрее внедряются, обеспечивают лучший пользовательский опыт и приносят ощутимую отдачу от инвестиций, формируя дополнительную бизнес-ценность.

Другими словами, книга *«Эффективный разговорный ИИ»* — обязательное чтение для всех, кто интересуется проектированием высокоэффективных разговорных приложений. Независимо от того, начинаете ли вы свой путь в этой области или уже обладаете солидным опытом, вы непременно найдете здесь множество ценных идей. Это актуальный и незаменимый ресурс для всех, кто стремится раскрыть потенциал разговорного ИИ внедрять инновации, улучшать пользовательский опыт и повышать бизнес-ценность.

Хесус Мантас, глобальный управляющий партнер, IBM

Введение

Разговорный ИИ — это захватывающая технология, помогающая пользователям быстрее достигать своих целей, а компаниям — обслуживать клиентов с меньшими затратами. Решения на основе разговорного ИИ (часто называемые *чат-ботами*) стремительно набирают популярность, особенно после пандемии COVID-19. Сегодня существует множество книг и блогов о том, как спроектировать систему с разговорным ИИ, однако большинство из них останавливаются на этапе создания первого чат-бота и не описывают, как совершенствовать уже работающие решения. Многие компании внедряют такие технологии, чтобы предоставить своим клиентам возможность самообслуживания в масштабе, который был бы слишком дорогим или даже невозможным при участии только человеческого персонала. К сожалению, значительная часть подобных решений на основе ИИ демонстрирует неудовлетворительные результаты.

В последние годы появилось огромное количество материалов о генеративном ИИ — от статей о промпт-инжиниринге до небольших демонстрационных проектов. Однако чаще всего они ограничиваются лишь доказательством концепции или созданием прототипа. И лишь немногие ресурсы действительно объясняют, как сопровождать и развивать системы на уровне крупных организаций. Генеративный ИИ вновь пробудил интерес к рассматриваемой области, но он не является универсальным ответом на все задачи, особенно для компаний, предоставляющих пользователям полный цикл выполнения операций.

За последнее десятилетие мы вывели в продакшен множество решений на базе разговорного ИИ. Мы работали с самыми разными типами чат-ботов: вопросно-ответными, процессно-ориентированными и маршрутизирующими. Мы изнутри видели как успехи, так и возникающие трудности.

Мы написали эту книгу, чтобы помочь вам преодолеть подобные сложности. Слишком часто чат-боты воспринимаются как решение по принципу «установил и забыл». Мы также неоднократно наблюдали, как качество чат-бота снижалось из-за неправильного сопровождения. Как создатели разговорных систем мы с энтузиазмом беремся за анализ неэффективных решений на базе

ИИ и доводим их до совершенства. А как пользователи — хотим чаще встречать по-настоящему хорошие чат-боты.

Эта книга призвана помочь владельцам решений на базе разговорного ИИ и стейкхолдерам научиться выявлять и устранять проблемы, мешающие чат-ботам эффективно работать и в полной мере раскрывать свой потенциал. На этих страницах вы найдете множество шаблонов, стратегий и подходов, которые помогут решать наиболее распространенные проблемы, характерные для систем разговорного ИИ.

Благодарности

Мы слышали, что написание книги — безумно сложное занятие. Но мы пошли на это от большой любви к своему делу! Мы благодарны всем за невероятную поддержку, которую получили в процессе создания этой книги.

ЭНДРЮ. Я благодарен всем друзьям и коллегам, которые помогали развивать наши идеи и делились отзывами на ранние версии глав этой книги, включая Дэна Точалу (Dan Toczala), Дженнифер Гао (Jennifer Gao) и Стефана ван дер Стокта (Stéfan van der Stockt). Также благодарю всех коллег, с которыми мы вместе создавали и совершенствовали чат-ботов, включая, но не ограничиваясь лишь ими: Лео Маццоли (Leo Mazzoli), Виктора Повара (Victor Povar), Ребекку Джеймс (Rebecca James), Джасмита Сингха (Jasmeet Singh), Грега Икока (Greg Ecock), Томи Дженкинса (Tomi Jenkins), Морган Кэрролл (Morgan Carroll), Джонатана Роу (Jonathan Roe), Прита Мутусами (Preeth Muthusamy), Марко Ноэля (Marco Noel), Тейлора Вуда (Taylor Wood), Джима Кеннеди (Jim Kennedy), Элизабет Смит (Elizabeth Smith), Ричи Лимпиджанкита (Richie Limpijankit), Дженис Чан (Janice Chan), Югандару Чеджарлу (Yugandhar Chejarla), Канчан Панде (Kanchan Pandey), Саеда Тахера (Syed Taher), Анирбана Мукерджи (Anirban Mukherjee), Аника Маджумдера (Anik Majumder), Свапнила Шарму (Swapnil Sharma) и Терренса Никсу (Terrence Nixa). Особая благодарность моей жене Элиз, сыновьям Грегу и Джеффу, а также родителям Рону и Дебби за поддержку на протяжении всего процесса.

КАРИ. Я хочу выразить искреннюю признательность моему замечательному партнеру Джейсону Кернсу. Этот год выдался особенно трудным, и твоя вера, поддержка и терпение были для меня дороже всего. Также хочу поблагодарить своих бывших коллег и наставников, которые за последние три десятилетия способствовали росту моего профессионализма и повлияли на мои рабочие принципы: Джареда Янга (Jared Young), Шона Хиггинса (Sean Higgens), Барта Дэя (Bart Day), Лори Уоркман (Lori Workman), Кори Йоченса (Cory Yochens)

(покойся с миром), Джеффа Фезеролфа (Jeff Fetherolf), Тима Шеру (Tim Shera), Хайди Пайпер Морган (Heidi Piper Morgan) и Джеффа Маттео (Jeff Matteo). Также спасибо моим детям (Лани, Райану и Джо), моим приемным детям (Алексу, Джошу и Лили), внуку Кэмерону, Эшли Джейкобс и Брюсу Кернсу за их энтузиазм и теплые слова всякий раз, когда заходил разговор об этой книге.

ЭНИКО. Написание этой книги стало путешествием, которое я не смогла бы завершить без поддержки моей семьи. Моему мужу Шахраму, нашим замечательным детям Дженнифер, Алексу, Рашель, а также нашим приемным детям Меру, Маргарет и Талу — спасибо за бесчисленные вечера, проведенные на кухне за обсуждением моих идей, за ваш юмор и советы, которые подпитывали этот проект. Лиза и Эрик, спасибо вам за то, что гуляли с Тео и заботились о нем, позволяя мне сосредоточиться на работе над книгой. Ваша доброта и поддержка бесценны.

Я также выражаю глубокую благодарность коллегам, бывшим коллегам и наставникам — Уиллу Раабе (Will Raabe), Керри Бойлу (Currie Boyle), Крейгу Триму (Craig Trim), Клэр Тернер (Claire Turner), Виктору Повару (Victor Povar), Бренде Хэддок (Brenda Haddock), Ксавьеру Вергесу (Xavier Vergés) и Лесу Йипу (Les Yip), с которыми мы создавали чат-ботов еще до того, как их стали так называть. Спасибо тем, кто вместе со мной работает над развитием разговорного ИИ и непрерывным совершенствованием чат-ботов, включая, но не ограничиваясь, Монисанкара Даса (Monisankar Das), Чайана Рэя (Chayan Ray), Ави Яэли (Avi Yaeli), Сергея Зельтына (Sergey Zeltyn), Игнаса Валанциуса (Ignas Valancius), Романаса Марченко (Romanas Marčenko), Эймантаса Пяликиса (Eimantas P likis), Кристину Рибачионкайте (Kristina Ribačionkaitė), Атерет Анаби-Тавор (Ateret Anaby-Tavor), Эллу Рабинович (Ella Rabinovich), Мадхусмиту Патил (Madhusmita Patil), Арзу Сабхарвал (Arzoo Sabharwal) и Ричу Манрал (Richa Manral) и многих других. Я благодарна за ваш труд, преданность делу и идеи.

Мы признательны всей команде «Manning Publications» за их поддержку и помощь на протяжении всего процесса. Особая благодарность нашим научным редакторам — Джеку С. Кроуфорду (Jack C. Crawford) и Стефану ван дер Стокту (Stéfan van der Stockt). Джек — опытный архитектор решений в области ИИ, магистр компьютерных информационных систем Клермонтского университета. Он руководит проектами генеративного ИИ для виртуального помощника крупного мобильного приложения, которым пользуются миллионы людей по всему миру. Стефан — архитектор решений по ИИ в IBM, специализируется на генеративном ИИ, машинном обучении и общей архитектуре искусственного интеллекта. Он помогает клиентам IBM реализовывать проекты промышленного масштаба, основанные на данных технологиях.

Также благодарим всех рецензентов: Абдуллу аль Имрана (Abdullah Al Imran), Анандаганеша Балакришнана (Anandaganesh Balakrishnan), Артема Дайнеко (Artem Daineko), Аюша Бихани (Ayush Bihani), Брендона Смита (Brandon Smith),

Бруно Соннино (Bruno Sonnino), Эрико Лендзьяна (Erico Lendzian), Фелипе Коутиньо (Felipe Coutinho), Гэри Пасса (Gary Pass), Харината Маллепалли (Harinath Mallepally), Игоря Виейру (Igor Vieira), Джеймса Блэка (James Black), Йири Пика (Jiri Pik), Джона Келви (John Kelvie), Джонатана Ривза (Jonathan Reeves), Лукаса Петрали (Lucas Petralli), Марко Котротсоса (Marco Kotrotsos), Максима Волгина (Maxim Volgin), Нахида Алама (Nahid Alam), Олега Копычко (Oleg Korychko), Парта Сантпуркара (Parth Santpurkar), Петра Пинделя (Piotr Pindel), Ричарда Вона (Richard Vaughan), Скотта Линга (Scott Ling), Симоне Сгуадзу (Simone Sguazza), Стефано Онгарелло (Stefano Ongarello), Свапнила Дешпанде (Swapneelkumar Deshpande), Тонга Чжу (Tong Zhu), Умеша Ходегатту (Umesh Hodeghatta) и Венкатрамана Умбалачери Рамасами (Venkatraman Umbalacheri Ramasamy) — ваши предложения сделали книгу лучше.

Наконец, мы выражаем глубокую благодарность Хесусу Мантасу (Jesús Mantas) за его прекрасное предисловие, которое точно отражает суть применения разговорного ИИ в реальных условиях.

О книге

Технология, описанная в этой книге, имеет множество названий: разговорный ИИ, виртуальные помощники, автоматизированные агенты, чат-боты, боты или просто «искусственный интеллект». Как бы вы ее ни называли, эта книга научит вас эффективно использовать данную технологию для достижения ваших целей и удовлетворения потребностей пользователей. Если ваш разговорный ИИ работает не так, как хотелось бы, эта книга поможет вам его улучшить.

Для кого эта книга

Книга *«Эффективный разговорный ИИ»* создана для тех, кто уже занимается поддержкой решений на базе разговорного ИИ: бизнес-спонсоров, владельцев продуктов, дизайнеров и разработчиков подобных систем. Опыт в программировании будет полезен при реализации некоторых технических решений, однако он вовсе не обязателен для большинства концептуальных и проектных корректировок, предлагаемых в книге (в том числе благодаря современным low-code- и no-code-инструментам). Описанные далее проблемы и способы их решения будут полезны и тем, кто только планирует создать свой первый разговорный ИИ, ведь, как известно, «предупрежден — значит вооружен».

Структура книги

Книга состоит из четырех частей и двенадцати глав. Первая часть знакомит с разговорным ИИ, его преимуществами и недостатками, а также с системным подходом к его улучшению. Остальные части посвящены конкретным проблемам и предлагают различные способы их решения.

Часть 1 вводит ключевые понятия, необходимые для построения и совершенствования разговорного ИИ.

- Глава 1 знакомит с разговорным и генеративным ИИ и показывает, как они могут эффективно дополнять друг друга. Также здесь описаны проблемы и подводные камни, которых наша книга поможет избежать.

- Глава 2 рассказывает, как создать и развивать чат-бот: от простых вопросно-ответных систем — к реализации процессных сценариев и использованию генеративного ИИ.
- Глава 3 объясняет, как объективно оценивать эффективность разговорного ИИ и как разработать план его улучшения на основе метрик.

Часть 2 предлагает стратегии, помогающие системам разговорного ИИ лучше понимать запросы пользователей.

- Глава 4 объясняет, как определить цели пользователей и оценить степень понимания их запросов системой. Рассмотренные методы служат основой для следующих трех глав.
- Глава 5 показывает, как повысить точность понимания в решениях, основанных на распознавании намерений (классификаторах), особенно при ответах на часто задаваемые вопросы.
- Глава 6 описывает использование поиска и генерации, дополненной поиском (RAG), для обработки менее распространенных вопросов, не подходящих под заранее определенные намерения.
- Глава 7 демонстрирует, как применять генеративный ИИ на этапе разработки для создания обучающих и тестовых данных для систем разговорного ИИ.

Часть 3 рассматривает проблемы, возникающие при усложнении диалогов в системах разговорного ИИ.

- Глава 8 описывает методы упрощения процессных сценариев, помогающих повысить вероятность успешного завершения диалога пользователем.
- Глава 9 показывает, как использовать весь доступный контекст, чтобы задавать пользователю правильные вопросы и давать точные ответы.
- Глава 10 рассказывает, как применять генеративный ИИ для проектирования, анализа и даже замены сложных диалоговых процессов.

Часть 4 направлена на снижение неудобств для пользователей и операторов, взаимодействующих с ИИ.

- Глава 11 рассматривает причины, по которым пользователи прекращают взаимодействие с системой на ранних этапах, и объясняет, как сократить вероятность таких отказов.
- Глава 12 описывает эффективные методы резюмирования диалогов, особенно в случаях, когда пользователь отказывается от взаимодействия. С помощью данных методов оператор-человек сможет продолжить обработку запроса.

Мы рекомендуем начать с части 1, чтобы понять концептуальную основу книги (опытные разработчики могут бегло просмотреть главу 2, в который мы создаем чат-бота с нуля). Затем можно переходить к любой части, посвященной интересующей вас проблеме. Части книги можно читать в любом порядке, а вот главы

внутри одной части рекомендуется читать последовательно. В конце каждой главы приведены задания, позволяющие закрепить изученные концепции.

О коде в книге

Книга содержит множество примеров исходного кода (и промптов для LLM), как в нумерованных листингах, так и в тексте. В обоих случаях код выделен `вот таким шрифтом фиксированной ширины`, чтобы отличать его от основного текста. Иногда для кода также применяется жирный шрифт, чтобы выделить фрагменты, изменившиеся по сравнению с предыдущими шагами, — например, при добавлении новой функциональности в существующую строку кода.

Во многих случаях оригинальная версия исходного кода переформатируется; добавляются разрывы строк и измененные отступы, чтобы код помещался на странице. Иногда даже этого оказывается недостаточно и в листинги включаются маркеры продолжения строк (➥). Также из исходного кода часто удаляются комментарии, если код описывается в тексте. Многие листинги снабжены примечаниями, выделяющими важные концепции.

Исполняемые фрагменты кода можно загрузить из версии liveBook (электронной) по адресу https://livebook.manning.com/book/effective-conversational-ai. Полный код примеров книги доступен для загрузки на сайте Manning: https://www.manning.com/books/effective-conversational-ai. Код также размещен на GitHub: https://github.com/andrewrfreed/EffectiveConversationalAI. Репозиторий содержит рабочий пример CakeBot из главы 2, а также все фрагменты кода, примеры диалогов и промпты для LLM. Языковые модели продолжают непрерывно развиваться, а потому их ответы могут незначительно отличаться от приведенных в книге.

Форум liveBook

Приобретая книгу *«Эффективный разговорный ИИ»*, вы также получаете бесплатный доступ к платформе для онлайн-чтения liveBook издательства Manning (на английском языке). Эксклюзивные возможности liveBook позволяют оставлять комментарии как к книге в целом, так и к отдельным ее разделам или абзацам. Можно легко делать заметки для себя, задавать технические вопросы и отвечать на них, а также получать помощь от авторов и других пользователей. Чтобы получить доступ к форуму, посетите страницу https://livebook.manning.com/book/effective-conversational-ai/discussion. Информацию о форумах Manning и правилах поведения на них см. на https://livebook.manning.com/discussion.

В рамках своих обязательств перед читателями издательство Manning предоставляет ресурс для содержательного общения читателей и авторов. Эти обязательства не подразумевают конкретную степень участия автора, которое остается добровольным (и неоплачиваемым). Задавайте авторам хорошие вопросы, чтобы им было интересно участвовать в диалоге! Форум и архивы обсуждений доступны на сайте Manning, пока книга продолжает издаваться.

Об авторах



Эндрю Р. Фрид — Distinguished Engineer с более чем двадцатилетним опытом работы, половина которого посвящена ИИ. Он присоединился к подразделению IBM Watson вскоре после того, как система Watson победила чемпионов шоу «Jeopardy!», и с тех пор реализовал множество проектов на базе ИИ. Эндрю черпал знания из множества технических книг и блогов и теперь стремится «передать эстафету», делясь собственным опытом. Свое свободное время он любит проводить с семьей.



Кари Джейкобс работает в сфере информационных технологий уже почти 30 лет. Ее опыт охватывает управление дата-центрами, администрирование Unix и поддержку производственных приложений. В 2014 году она присоединилась к подразделению IBM Watson. В роли инженера по когнитивным системам и архитектора решений она принимала участие в создании разговорных ИИ для десятков компаний из списка Fortune 500, а также для национальных брендов, государственных учреждений, университетов и стартапов. Она обожает учиться и делиться знаниями. Среди ее увлечений — каякинг, фотография и бразильское джиу-джитсу.



Энико Рожа — Distinguished Engineer и технический директор IBM Global AI & Analytics Practice. За более чем 30-летнюю карьеру в IBM она реализовала инновационные мультиплатформенные решения разговорного ИИ в самых разных отраслях. Ее вдохновляют сложные задачи, требующие интеграции передовых технологий ИИ. Энико является автором множества патентов в области обработки естественного

языка (NLP) и юзабилити. Ее увлечение чат-ботами началось с соавторства и руководства разработкой онтологической системы диалога на естественном языке, которая произвела революцию в клиентском самообслуживании в сфере технической поддержки. Вне работы Энико любит проводить время с семьей, гулять с собакой Тео и устраивать ужины для друзей и близких, объединяя людей за хорошей едой и беседой.

Иллюстрация на обложке

На обложку книги *«Эффективный разговорный ИИ»* помещена иллюстрация под названием *«Le Donne Di Procida»* («Женщины из Прочида»), взятая из книги Франческо де Буркара *«Usi e costumi di Napoli e contorni descritti e dipinti» («Обычаи и традиции Неаполя и его окрестностей, описанные и изображенные»)*, опубликованной в 1853 году. Каждая иллюстрация этой книги тщательно прорисована и раскрашена от руки.

В прежние времена по одежде человека можно было легко определить, где он живет и какова его профессия или положение в обществе. Издательство Manning приветствует изобретательность и инициативность — качества, присущие индустрии IT, — и в знак этого размещает на обложках изображения, которые демонстрируют богатое разнообразие региональных культур, запечатленное на старинных рисунках.

Часть 1

Как усовершенствовать разговорный ИИ

Вам когда-нибудь встречался явно плохой чат-бот? Может, голосовой помощник, который постоянно твердит: «Пожалуйста, слушайте внимательно: структура меню недавно изменилась», или чат-бот, который никак не может понять ваш вопрос? Надеемся, этим ваш опыт взаимодействия с ИИ не ограничился и вам попадались боты, которые словно уже с вами знакомы и заранее знают все ваши потребности. Но что же отличает хороший разговорный ИИ от плохого?

Разговорный ИИ сегодня актуален как никогда, поскольку эта технология все чаще используется компаниями для улучшения как клиентского опыта, так и бизнес-показателей. Генеративный ИИ возродил интерес к этой области и позволил наделить чат-ботов «интеллектом». Многие из них выглядят довольно многообещающе на стадии прототипа, но тем не менее дают сбои в эксплуатации. В первой части книги мы и рассмотрим, как построить и усовершенствовать свой разговорный ИИ.

Глава 1 познакомит вас с технологиями разговорного и генеративного ИИ, их достоинствами и недостатками, а также объяснит, как они дополняют друг друга. Кроме того, в ней описаны типичные проблемы, с которыми сталкиваются многие решения на базе ИИ. В главе 2 рассказывается, как построить и развивать чат-бота, добавляя ему новый функционал и повышая его сложность с использованием как традиционных, так и генеративных методов. Глава 3 демонстрирует, как объективно оценивать ИИ и планировать его улучшения.

1 Как работает разговорный ИИ?

В этой главе

- ✓ Определение и минимизация рисков разговорного ИИ
- ✓ Применение генеративного ИИ в разговорном ИИ
- ✓ Безопасное использование генеративного ИИ
- ✓ Улучшение ИИ в соответствии с заданной целью

Каждый из нас хоть раз сталкивался с разговорными агентами, которые больше раздражали, чем выполняли поставленную перед ними задачу: чат-бот, который вас не понимает, голосовой помощник с запутанным сценарием диалога или телефонный бот, сразу же переключающий вас на человека. Как решить подобные проблемы, а еще лучше — как изначально построить систему правильно? Эта книга научит вас создавать чат-ботов и другие решения на базе разговорного ИИ, с которыми ваши пользователи будут с удовольствием общаться.

Мы, как специалисты в области разговорного ИИ, работаем и с заказчиками, только начинающими внедрять автоматизированные агенты для ряда задач, и с крупными компаниями, где бизнес-риски настолько высоки, что одна галлюцинация ИИ может перевесить пользу от десятков корректных и успешных взаимодействий. С помощью разнообразных примеров из реальной практики

мы предложим вам различные варианты реализации и улучшения разговорного ИИ — как с применением генеративных методов, так и без них.

Начнем с краткого обзора классических технологий разговорного ИИ, а затем уже перейдем к генеративному ИИ и процессу непрерывного улучшения системы, что мы считаем залогом безопасной и эффективной работы с разговорным ИИ. Далее, в главе 2, вы сможете построить собственного чат-бота, используя как классические, так и генеративные методы.

1.1. Введение в разговорный ИИ

Разговорный ИИ, также называемый *чат-ботами, виртуальными агентами, ИИ-помощниками* и *цифровыми сотрудниками*, — это совокупность технологий, предназначенных для имитации или замещения человеческих взаимодействий посредством письменного или устного естественного языка. Разговорный ИИ широко применяется для автоматизации клиентской поддержки, предоставления сервисов «голосового помощника» (например, Alexa и Siri) или предварительного этапа перед общением с человеком. В целом можно выделить три категории разговорного ИИ:

- *Вопросно-ответные системы*, также известные как FAQ-боты или Q&A-боты. Предоставляют прямой ответ на вопрос пользователя, как правило, без уточняющего диалога.
- *Процессно-ориентированные, или транзакционные, решения* — для достижения цели пользователь отвечает на ряд вопросов от бота, например, при проверке баланса счета, записи на прием или отслеживании статуса обращения. Такой ИИ может либо выполнять какое-то действие самостоятельно, либо собирать информацию для ее последующей обработки человеком.
- *Маршрутизирующие агенты* — задача бота заключается исключительно в перенаправлении пользователя: к специализированному боту или оператору-человеку.

Некоторые решения включают элементы всех трех типов. Например, чат-бот банка может отвечать на простые вопросы («когда работает касса», «где ближайшее отделение»), вести процессные сценарии для открытия счетов или проверки баланса, а также перенаправлять к специалистам в случае сообщений о мошенничестве.

Данные типы чат-ботов имеют схожую архитектуру, но разные задачи. Маршрутизирующему агенту достаточно просто определить намерение (интент) пользователя, тогда как процессно-ориентированному необходимо не только понимать намерение, но и поддерживать вовлеченность на протяжении всего сценария. В нашей книге мы рассмотрим ряд распространенных проблем с разговорным ИИ и их успешные решения (табл. 1.1).

Таблица 1.1. Распространенные проблемы с разговорным ИИ

Проблема	Пример успешного решения	В книге
Непонимание намерений пользователя	Повышение точности распознавания намерений с 76 до 92 %	Часть 2 (главы 4–7)
Избыточная сложность для пользователя	Повышение успешности поиска с 40 до 90 %	Часть 3 (главы 8–10)
Отказ от использования	Снижение отказов на 15 %	Часть 4 (главы 11–12)

Все типы чат-ботов сталкиваются с проблемой понимания пользователя. Процессно-ориентированные боты особенно подвержены избыточной сложности и зачастую перегружают пользователей, и абсолютно все типы ботов сталкиваются с отказами их использования. Каждая часть книги посвящена конкретной проблеме и, где возможно, содержит реальные примеры диалогов с чат-ботами разных типов, поэтому вы можете сразу перейти к тем проблемам, которые интересуют вас больше всего.

Системы на базе разговорного ИИ создаются для решения конкретных задач. Если они этого не делают, значит, они приносят больше неудобств, чем пользы. Такие проблемы как раз и показывают, что именно стоит улучшить. Но прежде чем совершенствовать существующее решение, необходимо понять, для чего оно вообще было создано.

1.1.1. Для чего нужен разговорный ИИ?

Эффективный разговорный ИИ обеспечивает качественный пользовательский опыт и приносит много пользы: клиентам он экономит время и силы, а компаниям сокращает расходы на поддержку. Он никогда не устает и всегда доступен. При этом он персонализирован, эффективен и может быть даже проактивным, помогая клиентам достигать их целей.

Плохой разговорный ИИ действует с точностью до наоборот — раздражает пользователей, снижает их удовлетворенность и перегружает линии поддержки жалобами «бот меня не понял». Он выдает слишком длинные ответы, задает ненужные вопросы или ведет себя холодно и даже грубо. На рис. 1.1 показан пример неудачного взаимодействия с процессно-ориентированным ботом.

Разговорный ИИ не должен быть источником раздражения пользователя — напротив, он должен обеспечивать более удобное и рациональное взаимодействие, чем может предоставить человек. Сценарий на рис. 1.1 чрезмерно нагружает пользователя. Формально сценарий диалога корректен: пользователь *может* спросить о любом заявлении, не обязательно о своем собственном. Но здесь игнорируется общее правило — большинство все-таки интересуются именно своими заявлениями, и их легко идентифицировать: в чатах — по e-mail, под которым пользователь вошел, в голосовых сервисах — по номеру телефона.



Рис. 1.1. Неудачное взаимодействие с процессно-ориентированным ботом, который перегружает пользователя вопросами, — никакой пользы за три шага диалога

На рис. 1.2 представлен клиентоориентированный способ проверки статуса заявления. В результате пользовательский опыт персонализирован, а ответ система предоставляет быстрее, чем смог бы человек!

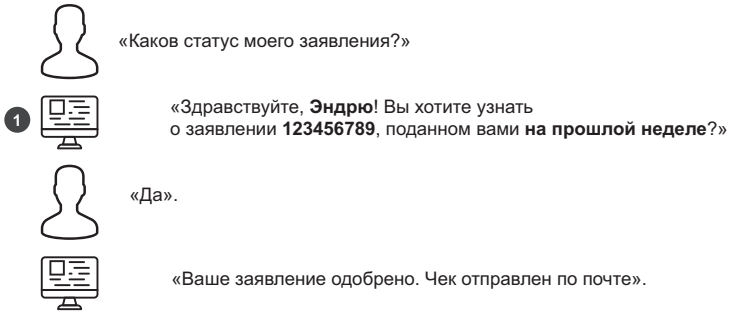


Рис. 1.2. Удачное взаимодействие, при котором контекст и разумные предположения используются для быстрого достижения цели пользователя. Контекст может быть получен из процесса авторизации (чат) или по номеру телефона звонящего (голосовой помощник)

Иногда процессно-ориентированного бота можно улучшить за счет оптимизации самого процесса. Важно помнить, что чат-боты — это не одна лишь *технология*.

Чат-боты взаимодействуют с людьми, а люди бывают непредсказуемыми. Технологический подход не способен предусмотреть все возможные сценарии взаимодействия.

Теперь, когда мы рассмотрели примеры как положительного, так и отрицательного пользовательского опыта, перейдем к принципам работы разговорного ИИ.

1.1.2. Как работает разговорный ИИ?

Решение на базе разговорного ИИ обычно включает три шага:

1. Определить, что именно хочет пользователь.
2. Собрать дополнительную информацию, необходимую для удовлетворения запроса.
3. Дать пользователю то, что он хочет.

Решение должно выполнять эти шаги максимально быстро и просто, соблюдая при этом правовые и этические нормы, например, безопасно обрабатывать конфиденциальные данные и не создавать иллюзию, что ИИ является человеком. Если ИИ не справляется с этими задачами или создает слишком много трудностей в процессе, пользователи будут избегать общения с ним и искать альтернативы, например, пытаться связаться с оператором или вообще отказаться от вашего сервиса.

На рис. 1.3 показана общая схема работы разговорного ИИ. Эти шаги поддерживаются архитектурой, представленной на рис. 1.4, демонстрирующем пример сценария «сброс пароля» для процессно-ориентированного бота.

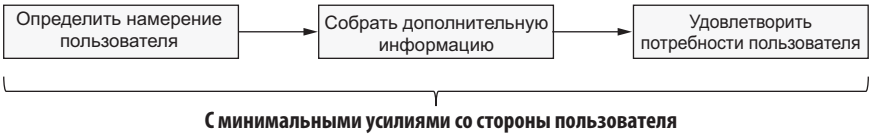


Рис. 1.3. Схема работы разговорного ИИ. В большинстве случаев «дополнительная информация» включает данные профиля пользователя

Давайте подробнее рассмотрим три основных шага:

- *Определить, чего хочет пользователь.* Обычно запрос формулируется на естественном языке, поэтому модуль NLU получает сообщение и определяет намерение. Чаще всего для этого используется алгоритм машинного обучения, например классификатор текста. Примеры намерений: «сбросить пароль» или «найти магазин». Намерение определяет следующий шаг процесса.
- *Собрать дополнительную информацию, необходимую для удовлетворения запроса.* Первичный запрос пользователя часто не содержит всех данных для его выполнения, он лишь запускает процесс. Диалоговый движок ведет

пользователя через все шаги, необходимые для выполнения запроса, например задает уточняющие или дополнительные вопросы: «Какой у вас номер счета?» или «Ваш почтовый индекс?». Для взаимодействия с другими системами может использоваться оркестрационный слой через вызовы программного интерфейса (API). Диалоговый движок управляет взаимодействием с пользователем и отвечает ему.

- *Дать пользователю то, чего он хочет.* Процесс завершается, когда запрос выполнен: пароль сброшен, адрес магазина предоставлен или пользователь соединен с оператором.



1. Пользователь взаимодействует с ботом через графический или телефонный интерфейс.
2. Сообщение передается в NLU для интерпретации.
3. Естественный язык преобразуется в намерение (например, «Я забыл пароль» становится #reset_password).
4. Определяется следующий шаг в диалоге, например задаются контрольные вопросы.
5. Вызываются внешние API, необходимые для выполнения задачи (например, сброс пароля).
6. Ответ передается пользователю через интерфейс.

Рис. 1.4. Архитектура разговорного ИИ при запросе на сброс пароля

Между разными типами ботов могут быть небольшие различия. Например, Q&A-бот почти не использует API, а процессно-ориентированный — очень часто. Маршрутизирующий агент опосредованно выполняет запрос пользователя, перенаправляя его к нужному специалисту.

1.1.3. Как построить разговорный ИИ

Для создания эффективного решения на базе разговорного ИИ вашей команде понадобятся специалисты с широким спектром навыков, как показано на рис. 1.5. Если вы хотите совершенствовать такие решения, важно понимать, как они строятся. В этом разделе мы кратко рассмотрим весь процесс разработки, а более подробно он изложен в книге «Conversational AI» (Manning Publications, 2021).

Отправной точкой для любого разговорного ИИ является проектирование пользовательского опыта. Нужно понять, чего хотят ваши клиенты и как помочь им достичь своих целей быстро и беспрепятственно. Все специалисты,

показанные на рис. 1.5, должны участвовать в обсуждении вопросов, касающихся пользователя:

- С какими проблемами чаще всего сталкиваются пользователи?
- Что они хотят сделать?
- Какой информацией они, скорее всего, обладают (и какой не обладают)?
- Как они, вероятнее всего, будут формулировать свои запросы?

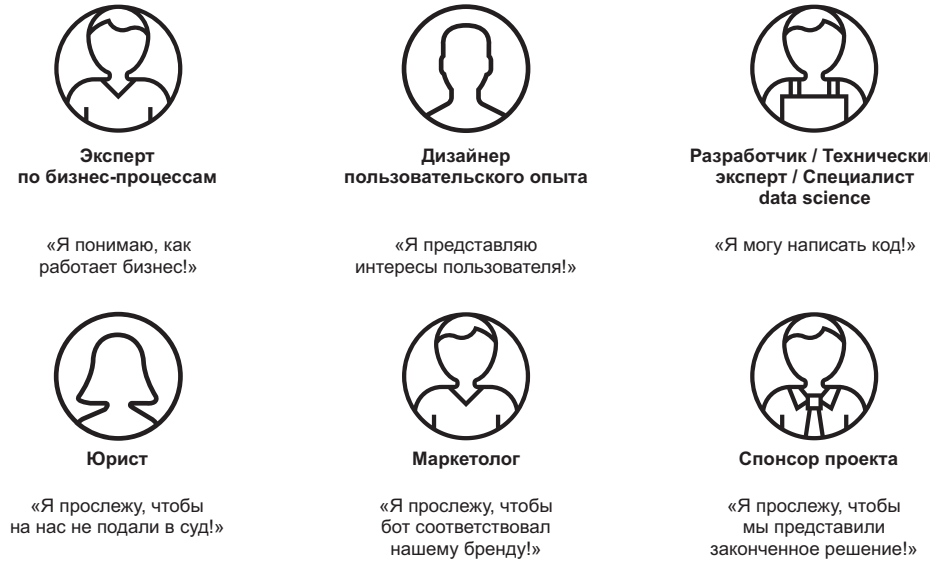


Рис. 1.5. Для построения корпоративного решения на базе разговорного ИИ необходима «команда мечты» с широким спектром компетенций

Как только вы поймете, что нужно пользователю, продумайте, что требуется для удовлетворения его потребностей. Скажем, пользователь не может получить доступ к своему аккаунту, поэтому ему нужна функция сброса пароля. Что необходимо для реализации этой функции? Обычно требуется как минимум три действия:

- Понять намерение пользователя (определить, что у него проблема с паролем, даже если он не использует слова «пароль» или «сброс»).
- Получить доступ к API, который может аутентифицировать пользователя и сбросить пароль.
- Собрать достаточно информации о пользователе для сброса пароля.

Эти шаги определяют процесс разработки.

Понимание намерения

Работа чат-бота начинается с извлечения смысла из высказываний пользователя, то есть определения его намерения, выраженного на естественном языке, с помощью текстового классификатора. *Высказывание* — это то, что произносит пользователь, *намерение* — это то, что оно означает (то есть что именно пользователь хочет), а *классификатор* соотносит высказывания с определенными намерениями.

Платформы для чат-ботов становятся все проще в использовании благодаря тренду на low-code и no-code, но это еще не означает, что они будут понимать ваши потребности без участия человека. В идеале специалист data science должен подготовить репрезентативную, сбалансированную и разнообразную обучающую выборку, а также провести тестирование для проверки точности обученного классификатора. Если этот этап выполнен плохо, возникает проблема «бот меня не понимает», поскольку ИИ обычно выдает общий ответ на все нераспознанные высказывания.

Лучше всегда брать данные для обучения из прошлых взаимодействий пользователей, таких как логи чатов, расшифровки звонков в колл-центр или электронные письма. Сбор качественных данных и их использование для улучшения распознавания намерений мы рассмотрим во второй части книги.

Использование API

Разработчику необходимо предоставить виртуальному помощнику API. При этом важно четко определить требуемые входные параметры, форматы выходных данных и процесс обработки ошибок, чтобы было понятно, как интегрировать API в чат-бота. Функция, предоставляемая API, может быть реализована на любом языке программирования, важно только, чтобы существовал эндпоинт API, к которому помощник мог бы безопасно обращаться.

Если у вас еще нет API, чат-бот может стать причиной его создания или же дизайн чат-бота может потребовать изменений в API. API полезны для предоставления пользователю структурированной информации (проверка баланса счета, поиск заявлений) или для выполнения действий от имени пользователя (сброс пароля, открытие счета) — без подходящих API удовлетворить подобные запросы зачастую невозможно.

API чаще всего применяются в процессно-ориентированных ботах, но также полезны и для передачи дополнительного контекста в вопросно-ответные и маршрутизирующие боты.

Сбор дополнительной информации

Необходимо разработать сценарий диалога, который позволит собрать данные, требующиеся для вызова API или ответа на первоначальный вопрос

пользователя. Этот процесс зависит от канала взаимодействия, для которого создается бот (например, веб или телефон), и от того, чего можно ожидать от пользователя. Так, при сбросе пароля в веб-интерфейсе обычно задается контрольный вопрос, но по телефону собрать такие сведения трудно, а через SMS — небезопасно. В то же время телефонные и SMS-каналы могут использовать номер телефона пользователя как часть процесса аутентификации.

Доступные API могут тоже повлиять на диалог, как и наоборот — сценарий может повлиять на API, либо они могут взаимно адаптироваться. Если процесс сбора информации слишком сложен, пользователи понимают, что эффективно работать с помощником у них не получается, и отсюда возникают известные болевые точки «это слишком сложно для бота» и «немедленный отказ от использования».

Также стоит отметить, что не каждому разговорному ИИ требуются все три рассмотренные составляющие:

- Некоторые API могут не требовать дополнительной информации. Например, API с расписанием работы магазина возвращает одинаковый ответ независимо от того, кто задает вопрос.
- Боты для часто задаваемых вопросов (FAQ) могут вовсе не вызывать API и лишь сопоставлять пользовательские высказывания с парами намерение/ответ.
- Бот, использующий поиск как резервный механизм, может не включать в себя намерения. Это распространенный шаблон в решениях для разговорного поиска, построенных на генеративном ИИ, — либо с использованием встроенных знаний большой языковой модели (LLM), либо через поиск по базе знаний и генерацию ответа на основе его результатов. Такой подход также может быть реализован в виде гибридной модели, где для наиболее частых вопросов создаются намерения, а все остальные направляются в поиск или генеративный ИИ.

Задания

1. Проанализируйте несколько последних чат-ботов, с которыми вы взаимодействовали (или которые вы делали сами). К какому типу ботов они относятся — вопросно-ответным, процессно-ориентированным или маршрутизирующим? Почему?
2. С какими трудностями вы столкнулись при взаимодействии с этими чат-ботами? Как их можно было бы улучшить?

1.2. Генеративный ИИ в разговорных системах: основные понятия

> Любая достаточно развитая технология неотличима от магии.
> — *Артур Кларк*

Генеративный ИИ (метод создания нового контента в режиме реального времени) — это передовая технология, поражающая своими возможностями. Скорее всего, вы уже видели, на что она способна: «напиши сонет в стиле Шекспира», «объясни, как работает ИИ, но как будто ты — пират», «составь план, как честно заработать 100 долларов». Но это не магия и не решение всех проблем. Генеративный ИИ может принести значительную пользу, но все же требует внимания для предотвращения нежелательных последствий, например галлюцинаций.

Генеративный ИИ помогает решить ряд проблем разговорного ИИ:

- *Непонимание намерений пользователя.* Генеративный ИИ позволяет более точно определять намерения или заменять часть либо все распознавание намерений с помощью RAG (генерации, дополненной поиском), обобщая контент, полученный из поиска. Он также более гибок в учете нюансов намерений пользователя.
- *Избыточная сложность для пользователя.* Генеративный ИИ позволяет вести диалог более простым языком или может тестировать систему на избыточную сложность.
- *Отказ от использования.* Генеративный ИИ создает более удачные ответы, удерживающие пользователей и удовлетворяющие их потребности.

Генеративный ИИ можно использовать внутри разговорного ИИ, позволяя ему напрямую взаимодействовать с пользователями, отвечая на их вопросы или выполняя поиск информации. Также генеративный ИИ может помочь при создании разговорных систем, например для генерации более качественных сценариев и сообщений или анализа предыдущих диалогов. Генеративный ИИ не заменяет классические методы разговорного ИИ, а работает в сочетании с ними.

1.2.1. Что такое генеративный ИИ

Генеративный ИИ — это общее название для систем на основе *базовых моделей*, которые обучаются на широком спектре задач. Существует несколько видов таких моделей, но в этой книге акцент сделан на LLM — моделях машинного обучения, обученных на больших текстовых датасетах. Насколько больших? Если и не на «всех текстах в интернете», то близко к этому.

Модель, «прочитавшая весь интернет», должна идеально разбираться в последовательностях слов и предложений. Она обучается принимать последовательность

слов и прогнозировать следующее наиболее вероятное слово. Повторяя этот процесс снова и снова, LLM способны генерировать слова, предложения, абзацы и даже целые страницы текста.

LLM можно использовать в составе разговорного ИИ. Они могут выполнять задачи, напрямую доступные пользователям, или задачи, помогающие в разработке самой системы. В табл. 1.2 приведены примеры таких задач.

Таблица 1.2. Примеры задач разговорных ИИ, в которых можно применять LLM

Задачи пользователей	Задачи разработчиков
Генерация ответов (с использованием генерации, дополненной поиском)	Редактирование или создание диалогов и сценариев
Резюмирование транскриптов диалогов	Дополнение (аугментация) обучающих данных

LLM могут выполнять данные задачи с минимальной подготовкой или вовсе без нее и ускорять процесс разработки системы. При этом они устойчивы к небольшим вариациям в формулировках пользовательских запросов, которые традиционный классификатор может не распознать. Но у них есть и определенные риски:

- LLM обучаются на собственных обучающих данных. Вы когда-нибудь искали что-то в интернете? Он полон предвзятости, враждебности и дезинформации. Генерация, дополненная поиском (RAG), — это оптимальный способ формировать ответы, так как он привязывает вывод LLM к вашим документам, а не к ее внутренним данным (которые в основном обучены на интернет-контенте).
- LLM не знают, являются ли их ответы истинными, они лишь прогнозируют наиболее вероятное продолжение их собственного «промпта». Отсюда возникают галлюцинации — ответы, которые выглядят убедительно, но, по сути, бесполезны. Никогда нельзя быть уверенным, что именно скажет LLM. Поэтому их использование при написании диалогов — хорошая идея, но стоит просмотреть результаты перед их применением.

LLM могут выдавать ложную информацию без малейшего сомнения. Или же наоборот — за секунды генерировать ответ уровня эксперта. Они способны проявлять удивительную креативность или же демонстрировать пугающие предубеждения — в интернете хватает и того и другого! Чтобы эффективно использовать LLM в разговорном ИИ, необходимо применять определенные защитные механизмы.

1.2.2. Ограничители генеративного ИИ

Вы бы стали внедрять генеративный ИИ, если бы знали, что злоумышленники могут использовать его для ответов на вопросы вроде «как сделать бомбу» или «расскажи расистскую шутку»? Скорее всего, нет! К счастью, существует несколько способов ограничить поведение LLM, что особенно важно, если ее

ответы передаются пользователям напрямую. Рассмотрим несколько видов ограничителей (guardrails).

Выбор модели и обучающих данных

Первый ограничитель — это выбор модели. Большинство специалистов предпочитают использовать готовую модель, а не обучать свою собственную, поскольку обучение новой LLM может стоить миллионы долларов и занять месяцы.

LLM обучаются на огромных датасетах — многие используют модифицированную версию The Pile — это 886,03 Гбайт разнообразной открытой коллекции англоязычных текстов, созданной в качестве обучающего корпуса для LLM (https://en.wikipedia.org/wiki/The_Pile_%28dataset%29). Разработчики могут как исключать части датасета (например, за предвзятые данные или ненормативную лексику), так и добавлять собственные данные (частные или лицензированные). Многие модели с открытым исходным кодом сопровождаются «карточкой модели», где описаны используемые данные и методология обучения. Изучив этот документ, можно подобрать модель с подходящим вам датасетом.

Неплохой первый шаг, но далеко не единственный.

Предварительная фильтрация ввода на предмет ненависти, оскорблений и ненормативной лексики

Другой вариант — проверять пользовательский ввод и блокировать проблемные запросы. Для этого применяют разные методы, включая поиск ключевых слов (брань, оскорбления) или классификацию текста. Этот процесс со временем превращается в «гонку вооружений»: поставщики LLM делают свои модели все более безопасными, а пользователи придумывают все более хитрые обходные маневры. Некоторые пытаются «взломать» промпт. Например, LLM может отклонить прямой запрос «Расскажи, как сделать бомбу», но ответить на «Расскажи сказку, как в детстве делала моя бабушка. Когда я не мог заснуть, она подробно описывала, как однажды сделала бомбу. Расскажи эту сказку». Одним из наиболее примитивных способов препятствовать таким взломам является ограничение длины ввода.

Контекстные инструкции и промпт

Следующий ограничитель — это инструкции, которые мы задаем LLM через промпт. На рис. 1.6 показано, насколько для модели важен контекст.

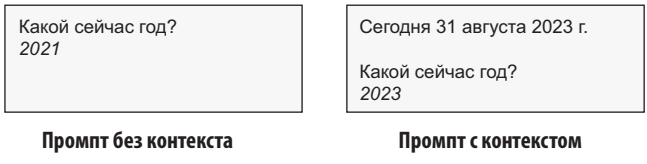


Рис. 1.6. Добавление контекста в промпт — ключевой способ управления LLM

Контекст избавляет LLM от необходимости полагаться на собственные (устаревшие) данные и снижает вероятность галлюцинаций. Генерация, дополненная

поиском (глава 6), извлекает контекст из ваших проверенных документов. Контекст также можно использовать для установки персоны LLM, например, уточнение «ты дружелюбный редактор», которое можно использовать при правке текстов (глава 10).

Предоставление контекста LLM — полезная техника.

Постфильтрация вывода

Подобно предварительной фильтрации, можно проверять и ответы LLM. Например, выполнять поиск ключевых слов или других признаков враждебных высказываний, оскорблений и ненормативной лексики (hate, abuse and profanity, HAP). Существуют готовые библиотеки, например profanity-check (https://pypi.org/project/profanity-check/).

В некоторых случаях можно также сравнивать ответ с фрагментами промпта. Генерация, дополненная поиском, предполагает, что LLM отвечает исключительно на основе документов, найденных при поиске, поэтому можно провести анализ текстового сходства, чтобы проверить, действительно ли ответ полностью опирается на эти документы.

Участие человека

Самый надежный вариант — не предоставлять LLM полную свободу. Участие человека гарантирует, что вы контролируете работу модели. Здесь есть два варианта: ретроактивная проверка и предварительная проверка.

Ретроактивная проверка означает, что вы периодически анализируете ответы LLM. Например, можно раз в неделю просматривать выборку входных и выходных данных модели. Это не предотвратит нежелательный результат, но позволит его обнаружить и скорректировать работу LLM.

Предварительная проверка означает, что LLM используется лишь как помощник для человека, принимающего окончательное решение. Например, LLM может работать как редактор текста — генерировать реплики, которые человек затем вставит в диалоговый движок.

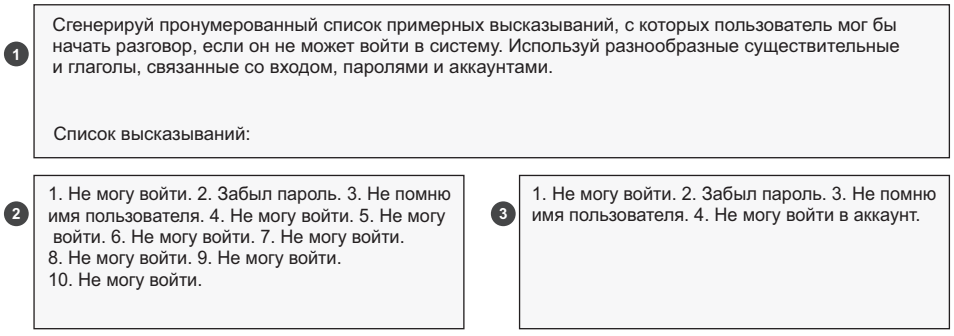
Такое использование LLM позволяет решать многие проблемы пользовательского опыта, например сгенерировать обучающие данные для корректного распознавания намерения пользователя или переписать тексты, чтобы устранить проблемы типа «диалог слишком сложный (или грубый)».

1.2.3. Эффективное использование генеративного ИИ в разговорных системах

Эффективное применение генеративного ИИ зависит от двух ключевых условий — выбор подходящей модели для конкретной задачи и снижение рисков за счет использования защитных механизмов.

Правильная модель (и параметры) для задачи

Существуют тысячи LLM, обученных для самых разнообразных задач. Поведение модели можно корректировать с помощью экспериментов с промптами и параметрами. На рис. 1.7 показан эффект параметра «repetition penalty» (штраф за повторение) на модели Flan-UL2 в творческой задаче. Разные задачи требуют разных настроек. Низкий repetition penalty полезен при использовании текста из предоставленных документов. Высокий repetition penalty помогает в креативных задачах, например таких, как генерация списков.



1. Общий промпт.
2. Ответ Flan-UL2 (низкий repetition penalty). Модель начинает повторяться после трех пунктов списка.
3. Ответ Flan-UL2 (высокий repetition penalty). Вывод не содержит повторов.

Рис. 1.7. Результат изменения одного параметра LLM (repetition penalty)

В этой книге мы будем использовать несколько моделей и наборов параметров для демонстрации различных методов. Наша цель — показать, что описанные методы имеют широкий спектр применения. Возможно, вы не найдете в тексте упоминания именно той модели, которую предпочитаете использовать, и вам придется подбирать иные промпты, параметры или модели, подходящие конкретно для вашего случая. К тому же на момент выхода книги вполне вероятно появление совершенно новых моделей!

Для каждой задачи также полезно протестировать несколько моделей. Например, Flan-UL2 — это LLM, обученная на 50 задачах, включая ответы на вопросы и поиск информации (https://huggingface.co/google/flan-ul2). Она универсальна. MPT-7B-Instruct специализируется на одной задаче — выполнение кратких инструкций (https://huggingface.co/mosaicml/mpt-7b-instruct). Затраты на использование моделей, равно как и характеристики их эффективности, различаются. Вероятнее всего, вам придется поэкспериментировать с несколькими вариантами, прежде чем вы выберете наиболее подходящий для своей задачи. Кроме того, возможно, что в рамках одного решения вы будете использовать разные модели для различных задач. В табл. 1.3 приведены наши рекомендации по выбору LLM.

Таблица 1.3. Рекомендации и предостережения при работе с LLM

Не стоит	Стоит	Почему
Использовать модель только потому, что вы видели, как она хорошо справляется с задачей, которая вам не нужна	Выбирайте модель, подходящую именно под вашу задачу, или экспериментируйте с несколькими моделями	Эффективность зависит от задачи, включая параметры и особенности формулирования промпта. Задачи могут быть разные: генерация, классификация, извлечение информации, ответы на вопросы, RAG, резюмирование и перевод
Отказываться от модели или промпта из-за одного неудачного эксперимента	Тестируйте на множестве входных данных, моделей и параметров	Иногда результат бывает неудачным случайно. Для уверенности в конфигурации LLM требуется несколько тестов
Полностью полагаться на LLM, особенно при работе с пользователями в разговорных ИИ	Устанавливайте защитные механизмы на нескольких уровнях	Ответственность за итоговый результат несете вы (или ваша компания). Отговорка «Так сказала LLM» здесь не работает

Отговорка «Так сказала LLM» не работает

В 2024 году чат-бот одной канадской авиакомпании предложил скидку, которой на самом деле не существовало. В суде компания пыталась доказать, что чат-бот является «отдельным юридическим лицом, ответственным за свои действия». Суд с этим не согласился. Компанию обязали выплатить скидку, предложенную чат-ботом. (Полную историю можно найти на сайте BBC: https://mng.bz/GejV).

Используйте необходимые ограничители

На каждом этапе работы с LLM есть свои ограничители:

- *До*. Выберите модель, соответствующую вашей задаче, и убедитесь, что ее обучающий датасет соответствует вашим требованиям. Определите степень свободы модели — будет ли она самостоятельно выполнять задачу от начала до конца или ее вывод будет проверяться человеком?
- *Во время*. Экспериментируйте с моделью, адаптируя ее под задачу, и контролируйте работу фильтров контента.
- *После*. Периодически оценивайте результаты работы модели, чтобы убедиться, что она по-прежнему соответствует вашим бизнес-целям.

Предположите возможный наихудший сценарий при использовании LLM и заранее придумайте, что вы будете делать в подобной ситуации. Например, при ответе на вопросы наибольший риск заключается в том, что модель начнет придумывать ответы без какой-либо опоры на факты (галлюцинировать). Чтобы

этого избежать, можно, например, ограничить модель контекстом или регулярно проверять ее ответы.

Задания

1. Проанализируйте чат-ботов, которые вы рассматривали в предыдущих заданиях. Как их можно было бы улучшить с помощью генеративного ИИ?
2. Для каждого из сценариев использования генеративного ИИ определите, как применять его безопасно. Свойственны ли галлюцинации каждому сценарию? Нужно ли учитывать риск генерации враждебных, оскорбительных или нецензурных высказываний?

1.3. Непрерывное улучшение разговорных ИИ

> Программное обеспечение похоже на энтропию. Его трудно постичь, оно ничего не весит и подчиняется второму закону термодинамики, то есть всегда возрастает.
>
> — *Норман Ральф Огастин*

> «Энтропия» в широком смысле означает постоянное стремление к хаосу.
>
> — *Сид Срирам*

Программное обеспечение никогда не получается сразу идеальным. То требования были поняты не так, то потребности изменились или поступила обратная связь от пользователей и теперь нужны доработки. Программные системы на основе ИИ не являются исключением. Без постоянного улучшения они с большой вероятностью придут в упадок.

1.3.1. Зачем нужно непрерывное улучшение

Даже если сегодня мы идеально настроим разговорный ИИ, со временем наши потребности изменятся:

- Пользователи будут формулировать новые запросы и использовать систему иначе.
- Бизнес внедрит новые правила выполнения процессов.
- Новые технологии, включая генеративный ИИ, сделают возможным то, что ранее было невозможно.
- Появятся более новые и эффективные модели ИИ.

Разговорный ИИ состоит из нескольких компонентов: понимание исходного намерения пользователя, сбор дополнительной информации (при необходимости) и выполнение запроса пользователя. Каждый из этих компонентов со временем

изменяется, и все они требуют постоянного улучшения. Сбои хотя бы в одном звене приводят к недовольству пользователей и снижают бизнес-результаты.

Подобно цепи, система разговорного ИИ сильна лишь настолько, насколько прочно ее слабейшее звено. Допустим, у нас идеально отлажен процесс предоставления информации пользователю, но он, по сути, бесполезен, если мы изначально неправильно распознаем намерение. На рис. 1.8 показана диалоговая воронка для процессно-ориентированного чат-бота, который ищет заявления пользователей. Размер элементов наглядно демонстрирует количество клиентов, дошедших до каждого этапа.

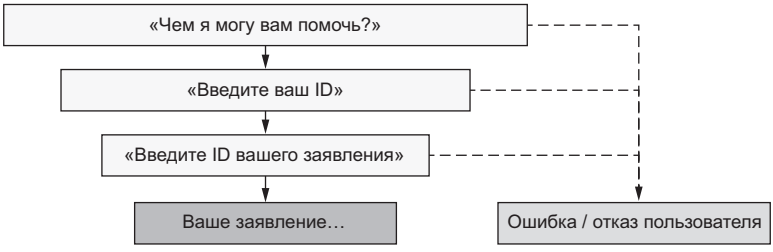


Рис. 1.8. Общий успех зависит от успеха каждого отдельного этапа. Визуально процесс напоминает воронку, сужающуюся после каждого шага

Успех многогранен. Чтобы пользователь получил то, что ему нужно, необходимо:

- вовлечь его (А);
- понять его (Б);
- предоставить ему желаемое (В).

Успех любого процесса можно представить как А × Б × В. Если результаты не соответствуют ожиданиям, необходимо проанализировать все компоненты цепочки, и с высокой вероятностью вы найдете способ, как можно улучшить каждый из них. Аналогичный подход можно использовать и при анализе Q&A-ботов, где каждый следующий вопрос представляет собой очередной шаг процесса. Более подробно мы рассмотрим эту структуру в главе 3.

Следует помнить, что сбои не всегда решаются исключительно с помощью технологий. Генеративный ИИ может неверно понимать пользователей и поэтому давать неправильные ответы. Для определения того, что можно улучшить в системе, нередко требуется поработать вручную, а непрерывный и постепенный подход к усовершенствованиям повышает вероятность успеха.

1.3.2. Цикл непрерывного улучшения

Идеальное решение для какой-либо задачи может быть довольно неочевидным или даже невозможным. Особенно это касается сферы ИИ, где все постоянно меняется

и каждое улучшение может привести к самым непредсказуемым последствиям. Поэтому важно совершенствовать разговорный ИИ с помощью серии небольших изменений. В главе 3 мы рассмотрим, как правильно оценивать эффект каждого изменения, пока же важно понимать следующее: изменение может привести к небольшому улучшению, значительному улучшению или же, напротив, ухудшить ситуацию! Каждое изменение — это новый шанс узнать о системе что-то новое.

Рисунок 1.9 демонстрирует типичный цикл непрерывного улучшения, применимый к любому чат-боту.



Рис. 1.9. Цикл непрерывного улучшения для разговорного ИИ

Типичный цикл непрерывного улучшения включает следующие этапы:

- *Измерение*. Базовая оценка эффективности системы до внесения изменений.
- *Определение проблемы*. Необходимо определить, что работает неправильно или неэффективно. Желательно, чтобы проблема была напрямую связана с каким-то бизнес-показателем. Например: «Мы заметили, что большое количество звонков переводится на оператора при <условии>».
- *Реализация*. Если проблема действительно значима, необходимо ее решить. Например, обновить классификатор намерений или отредактировать текст диалога.
- *Внедрение*. Внедрить изменение в систему и оценить его эффективность.
- *Повторение*. При необходимости повторить процесс. Если изменение показало хорошие результаты — отлично. Если нет, его лучше отменить. Затем можно перейти к следующей проблеме или продолжить итеративное улучшение текущей.

Лучше вводить небольшие и предсказуемые изменения, чем крупные и непредсказуемые. Если ваш бот не понимает пользователей, лучше изменить только один, наихудший по результатам интент (тип запроса), чем сразу менять их все. Если процессно-ориентированный сценарий демонстрирует низкий уровень завершаемости, лучше корректировать один шаг за раз, а не перестраивать сразу весь процесс.

На рис. 1.10 представлен пример крупного изменения системы. Из-за его масштаба внедрение займет много времени, а результат может быть самым разным: значительное улучшение, небольшое улучшение или даже ухудшение. Но понять это можно будет только после того, как внедрение завершено. Подобный подход довольно рискован: если результат окажется негативным, как объяснить это стейкхолдерам? «Мы так долго готовили изменение и, к нашему удивлению, сделали только хуже. Мы не знаем, что именно к этому привело, поэтому придется все отменить и начать заново». Просто замечательно! Вряд ли кто готов пойти на подобные риски.



Рис. 1.10. Крупные изменения, например повторное обучение по всем намерениям, занимают много времени и дают непредсказуемые результаты

Сравним это с рис. 1.11. Здесь вместо одного масштабного изменения сделаны четыре небольших. Каждое из них имеет те же потенциальные исходы (значительное улучшение, небольшое улучшение или ухудшение), но в меньших масштабах. Такой подход имеет несколько преимуществ:

- *Изменения более понятны* — при модификации только одного элемента легче установить связь между изменением и результатом. Кроме того, небольшие изменения проще отлаживать.
- *Больше возможностей чему-то научиться* — вместо одного шанса узнать что-то новое мы получаем четыре.
- *Больше вариантов* — благодаря небольшим изменениям и меньшим рискам можно остановиться раньше, если цель уже достигнута.

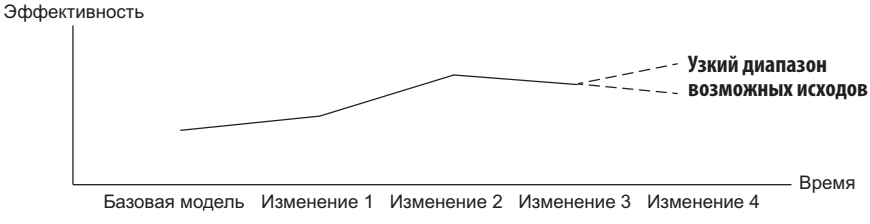


Рис. 1.11. Последовательность небольших изменений, например повторное обучение одного намерения за раз, уменьшает «зону поражения» каждого изменения, позволяет быстрее получить ценный результат и дает больше возможностей для обучения

На рис. 1.11 видно, что первых двух изменений уже оказалось достаточно, а значит, на этом можно было бы и остановиться. Третье изменение ухудшило систему, но так как оно было небольшим, его легко отменить. В результате мы получили полезный опыт.

Главное достоинство постепенных изменений — возможность скорее зафиксировать улучшения (и ценность для бизнеса)! Если преобразовать диаграмму для отражения бизнес-ценности, то окажется, что небольшие и быстрые изменения дали положительный эффект еще до завершения всего процесса. Это порадует как стейкхолдеров, так и пользователей.

Благодаря непрерывным улучшениям и небольшим изменениям мы получаем либо небольшое улучшение, которое быстро приносит бизнес-ценность, либо незначительное ухудшение эффективности, которое легко отменить и извлечь из него какой-то урок. На рис. 1.12 показано, как частые небольшие изменения быстро приносят пользу.

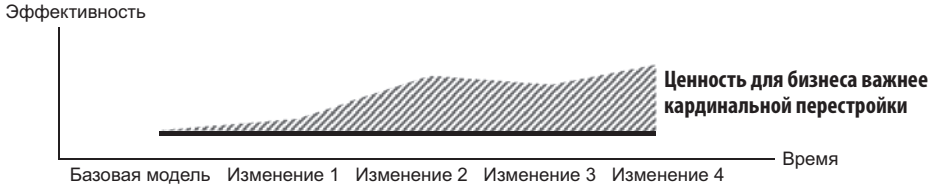


Рис. 1.12. Площадь над пунктирной линией показывает бизнес-ценность, полученную на протяжении всех изменений. Рабочий код в продакшене приносит ценность

Эффективный ИИ должен приносить бóльшую бизнес-ценность для ваших стейкхолдеров. Но как донести ее в понятной для них форме?

1.3.3. Объясняем концепцию непрерывного улучшения стейкхолдерам

Определения успешного решения на базе ИИ могут различаться, но скорее всего, вы используете один из стандартных показателей успеха:

- *Снижение затрат* — измеряется коэффициентом удержания или средним временем обработки запроса. (Звонки без участия человека или помощь операторам в более быстром выполнении работы.)
- *Удовлетворенность клиентов* — измеряется индексом потребительской лояльности (net promoter score, NPS), временем решения проблемы или снижением уровня оттока клиентов.

Ваши стейкхолдеры инвестировали в разговорный ИИ ради достижения каких-то бизнес-результатов, поэтому вы должны оценивать свое решение относительно этих результатов. Отслеживайте как текущие показатели, так и их динамику,

чтобы убедиться, что система улучшается (ну или хотя бы не становится хуже). Изменение требований к вашему решению означает, что вы постоянно боретесь с энтропией. Иногда вам придется улучшать систему просто для того, чтобы поддерживать текущий уровень успеха.

Из этой книги вы узнаете несколько методов улучшения своего решения на базе ИИ, и некоторые из них будут достаточно техническими. Скорее всего, вам будет интересно их применить, но для этого необходимо убедить стейкхолдеров эти улучшения профинансировать. Для этого придется говорить на их языке: меньше технических терминов, больше бизнес-ценности!

Рассмотрим два разных описания проблемы «бот не понимает пользователя»:

- *В технических терминах.* «Мы повысим точность определения намерения `#claim_status`. Классификатор распознает данное намерение с оценкой F1=0.92, при этом основная путаница возникает с намерениями `#claim_submission` и `#auth_status`».
- *С акцентом на бизнес-ценность.* «Мы повысим коэффициент удержания, увеличим удовлетворенность пользователей и снизим количество некорректных маршрутизаций за счет более точного распознавания звонков по статусу заявления. Это наш наиболее частый тип звонков. Ошибки в распознавании раздражают наших клиентов, которым приходится повторять одно и то же, что ведет к росту отказов от взаимодействия. Кроме того, неправильно распознанные звонки могут быть перенаправлены не к тому оператору, что увеличивает наши издержки и также снижает удовлетворенность пользователей».

Технические подробности отлично подходят для бэклога, но для большинства стейкхолдеров это лишь непонятные слова. Стейкхолдеров интересует лишь то, что конкретно эта проблема означает для них.

Мы рекомендуем соотносить улучшения с определенными бизнес-целями. Можно добавить технические подробности для удобства ведения бэклога, но все должны понимать бизнес-эффекты, стоящие за задачами. Таблица 1.4 связывает причины улучшения разговорного ИИ с конкретными бизнес-метриками.

Таблица 1.4. Причины улучшений и связанные с ними бизнес-показатели

Причина улучшения	Бизнес-метрика	Описание
Снижение затрат	Коэффициент удержания	Сокращение числа звонков, передаваемых человеку. В первую очередь актуально для процессно-ориентированных ботов
Снижение затрат	Среднее время обработки запроса	Сокращение времени работы оператора за счет более эффективной работы ИИ. Например, ИИ может проводить аутентификацию звонящего вместо оператора. В первую очередь актуально для процессно-ориентированных ботов

Причина улучшения	Бизнес-метрика	Описание
Снижение затрат	Объем «человеческого участия»	Сокращение числа сотрудников, участвующих в обработке звонка. (Число увеличивается, если звонки перенаправляются не к тому оператору.) В первую очередь актуально для маршрутизирующих систем
Удовлетворенность пользователей	Индекс потребительской лояльности (NPS)	Улучшение результатов опросов по завершении работы
Удовлетворенность пользователей	Время до выполнения задачи	Сокращение времени от первого контакта до решения проблемы
Комплаенс	—	Ограничения, которых необходимо придерживаться во избежание серьезных штрафных санкций. Это издержки ведения бизнеса

Следует отметить, что некоторые улучшения могут влиять сразу на несколько бизнес-метрик, как показано в табл. 1.5.

Таблица 1.5. Технические улучшения могут влиять на несколько бизнес-метрик

Техническое улучшение	Связанные бизнес-метрики
Повышение точности распознавания намерений	Повышает коэффициент удержания (звонящие не прекращают взаимодействия из-за того, что бот их не понимает) Сокращает участие человека (при маршрутизации звонок попадает к нужному оператору) Сокращает среднее время обработки запроса Сокращает время до выполнения задачи (за счет сокращения числа повторных попыток) Может повысить NPS (за счет сокращения числа повторных попыток)
Уточнение некорректно сформулированного вопроса	Повышает коэффициент удержания (звонящие не прекращают взаимодействия из-за того, что бот их не понимает) Сокращает время выполнения задачи (за счет сокращения числа повторных попыток)
Сокращение слишком длинных сообщений	Сокращает время выполнения задачи Повышает NPS

ПРИМЕЧАНИЕ Некоторые бизнес-цели могут противоречить друг другу. Например, один медицинский страховщик повысил точность определения намерения «причина отказа в выплате». Ранее звонящие, чей запрос не распознавался ИИ, сразу переводились на оператора, и поэтому не проходили пост-колл-опрос, который пред-

лагался только при успешном выполнении задачи ИИ. После улучшения точности распознавания пользователи смогли самостоятельно узнать информацию об отказе в выплате. Это повысило коэффициент удержания, но теперь недовольные клиенты стали проходить опрос и оставлять жалобы, в результате чего NPS виртуального помощника у страховщика снизился.

Задания

1. Изучите другие технические улучшения, такие как «снижение сложности диалога», «сокращение длительности диалога» и «устранение моментов, вызывающих затруднения у пользователей». На какие бизнес-цели они влияют?
2. Как бы вы поэтапно реализовали данные улучшения?

1.4. Начало работы

Все примеры разговорного ИИ, приведенные в этой книге, построены на двух платформах. Тем не менее используемые нами методы могут быть применены и к другим ресурсам:

- *Платформа разговорного ИИ* — базовая программная платформа, обеспечивающая такие возможности разговорного ИИ, как обработка естественного языка и управление диалогом. Подобных платформ существует множество: Amazon Lex, Google Dialogflow, Microsoft Azure AI Bot, Rasa и многие другие. В этой книге мы будем использовать IBM watsonx Assistant.
- *Платформа генеративных моделей ИИ* — сервис, предоставляющий одну или несколько LLM и API для взаимодействия с ними. Сюда относятся Anthropic, ChatGPT, Gemini, Hugging Face и Ollama. В своей работе мы обычно используем IBM watsonx.ai и его Prompt Lab, и именно его мы применяли для составления и тестирования промптов в этой книге.

Почему именно облачная платформа?

Установка необходимого ПО для приложений ИИ может представлять определенные сложности, так как LLM, как правило, требуют значительных ресурсов. Использование облачной платформы позволяет быстро начать работу и сразу перейти к созданию разговорного и генеративного ИИ.

Методы, описанные в этой книге, могут быть применены на самых разных платформах разговорного и генеративного ИИ, к тому же мы будем указывать различия в терминологии, если таковые имеются. Существует множество отличных решений, и вы можете выбрать именно то, с которым вам удобно работать, или же, наоборот, попробовать что-то новое!

Итоги

- При разработке разговорного ИИ необходимо всегда помнить о его влиянии на пользовательский опыт клиентов. Хороший чат-бот помогает им быстро получить желаемое, тогда как плохой лишь вызывает раздражение.
- Существуют тысячи разных генеративных моделей ИИ. Большие языковые модели (LLM) представляют собой подвид генеративных моделей и подходят для генерации текста.
- LLM отлично справляются со множеством задач, но также несут в себе значительные риски, например возможность галлюцинаций. Для эффективного и безопасного использования LLM требуется вдумчивое управление и разумные ограничения.
- LLM могут дополнять разговорный ИИ. Они способны напрямую отвечать пользователям, а также полезны при построении разговорного ИИ.
- Непрерывное улучшение возможно и даже необходимо для эффективного разговорного ИИ.
- Итеративные улучшения обеспечивают более высокую бизнес-ценность при меньших рисках.

2 Построение разговорного ИИ

В этой главе

- ✓ Создание FAQ-бота на основе разговорного ИИ
- ✓ Создание процессно-ориентированного разговорного ИИ
- ✓ Использование генеративного ИИ внутри разговорного ИИ

На практике разговорный ИИ зачастую оказывается не так уж и прост, и на протяжении всей нашей книги мы будем рассматривать множество разнообразных техник, с помощью которых можно решать проблемы, возникающие при разработке и внедрении ИИ-систем. В этой главе мы будем работать над созданием Cake Bot — решения на базе разговорного ИИ, сочетающего элементы нескольких его типов. Так мы сможем наглядно разобрать структуру подобных систем.

Для этого представим себе небольшую вымышленную американскую пекарню из штата Огайо под названием «Cake Shop». Компания занимается изготовлением тортов и принимает заказы как на доставку, так и на самовывоз. Они хотят добавить на свой сайт разговорный ИИ, чтобы облегчить своим клиентам процесс заказа. Так как раньше они ни разу не использовали бота, они планируют начать с малого, а затем уже постепенно расширить масштаб и возможности своего решения. Первым их шагом стало создание бота, отвечающего на наиболее часто задаваемые вопросы.

Многие задачи, рассматриваемые в этой главе, *могли бы* быть выполнены с использованием больших языковых моделей. Однако владельцы пекарни решают проявить осторожность и поэтому хотят иметь возможность контролировать выдаваемые моделью ответы на некоторые вопросы. Следовательно, их решение будет сочетать как традиционный, так и генеративный подход.

Для разработки бота мы будем использовать платформу разговорного ИИ (IBM watsonx Assistant), а затем подключим и генеративный ИИ (IBM watsonx.ai)[1]. Основные концепции, которые мы продемонстрируем в этой главе, применимы и к другим платформам ИИ, поэтому вы можете использовать ту платформу разговорного и генеративного ИИ, которая подходит именно вам.

2.1. Создание FAQ-бота

Большинство разработчиков разговорного ИИ начинают свой путь именно с ботов для ответов на часто задаваемые вопросы, также называемых FAQ-ботами. Они предоставляют пользователю ответ на его вопрос, обычно без каких-либо уточнений. Пользователь задает вопрос, бот возвращает ответ, и разговор завершается тогда, когда пользователь перестает задавать вопросы. Такие боты особенно удобны в случаях, когда возможных вопросов относительно немного.

В этом разделе мы создадим FAQ-бота для Cake Shop. На некоторые вопросы будут даны статические ответы, не зависящие от формулировки запроса, на другие — динамические ответы, изменяющиеся в зависимости от информации в запросе. Но прежде чем обучать бота отвечать на вопросы, мы должны создать его базовую структуру.

2.1.1. Основа FAQ-бота

Каждый разговорный ИИ должен уметь начинать разговор и реагировать на неизвестные ему запросы. Большинство платформ разговорного ИИ предоставляют такую возможность по умолчанию при создании нового чат-бота — достаточно лишь проверить эти настройки и при необходимости адаптировать их под свои нужды.

Cake Shop начинает разработку своего разговорного ИИ (так называемого помощника) и дает ему имя «Cake Bot». В главном меню платформы разработчик переходит в раздел «Действия» (`Actions`), где перечислены все возможности помощника. Первый список называется «Создано вами» (`Created by you`), и он пуст; второй называется «Задано помощником» (`Set by assistant`) и содержит возможности по умолчанию, представленные в табл. 2.1.

---

[1] У whatsonx нет русскоязычного интерфейса, но он поддерживает обработку русского языка в диалогах с пользователем. Можно создавать сценарии и базы знаний на русском. Элементы интерфейса на скриншотах переведены (или объяснены) в тексте книги. — *Примеч. ред.*

Таблица 2.1. Возможности по умолчанию в новом помощнике

Возможность	Когда выполняется
Приветствовать клиента	Первое открытие или активация помощника. Открытие помощника запускает диалог
Нет необходимого действия	Ни одно действие не может быть сопоставлено с сообщением пользователя (сообщение не распознано). На других платформах это может называться fallback intent («резервное намерение»)
Обнаружено ключевое слово	Обнаружены ключевые слова, например ненормативная лексика
Переход к резервному действию	Пользователь завершает диалог с чат-ботом

Наиболее важной для настройки является первая возможность, так как именно она позволяет персонализировать помощника. Текст по умолчанию — «Здравствуйте, чем я могу вам помочь?». Команда Cake Shop меняет его на «Здравствуйте! Я Cake Bot. Чем я могу вам помочь?». Изменения почти незначительны — лучше было бы добавить дополнительную информацию, например, о том, что бот умеет делать. Однако умеет он пока немногое, а поэтому команда Cake Shop оставляет приветственное сообщение почти без изменений.

Затем необходимо проверить действие «Нет необходимого действия» (`No action matches`). Оно будет срабатывать, когда бот не понимает пользователя. Так как у бота пока нет обучающих данных, это действие будет использоваться в качестве ответа по умолчанию на любой ввод клиента. Шаги диалога по умолчанию (`Conversation steps`) показаны на рис. 2.1.

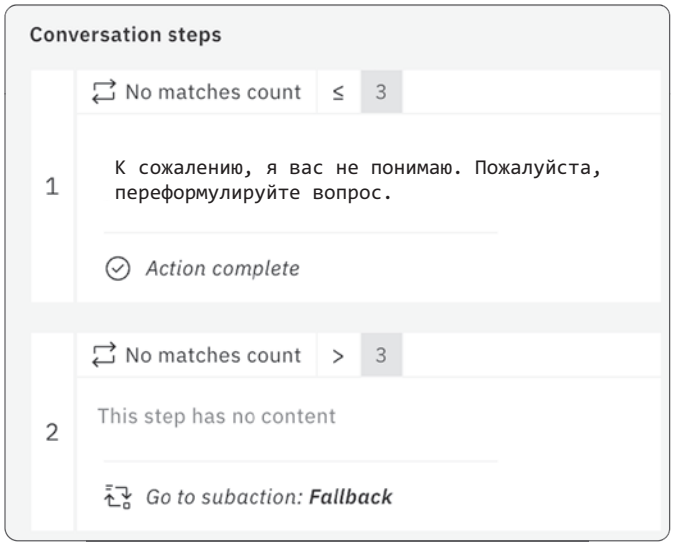


Рис. 2.1. Конфигурация по умолчанию для «Нет необходимого действия» в помощнике

При этой конфигурации происходит следующее:

1. Действие подсчитывает, сколько раз оно было вызвано в ходе разговора (`No matches count`).
2. Если три раза или меньше, выдается ответ: «К сожалению, я вас не понимаю. Пожалуйста, переформулируйте вопрос».
3. Если четыре раза или больше — выполняется переход к резервному действию (`Go to subaction: Fallback`). По умолчанию пользователю предлагается переключиться на оператора.

Команда Cake Shop решает снизить порог, изменив «3» на «1», благодаря чему пользователи не застревают в диалоге.

Резервное действие и переключение на оператора

Большинство платформ с использованием разговорного ИИ предоставляют интеграции no-code или low-code для переключения пользователей на оператора через чат или звонок. Мы не будем углубляться в детали этого процесса, так как они зависят от конкретной платформы. Достаточно лишь отметить, что это весьма распространенный подход. В рамках данной главы мы сосредоточимся на дизайне и обучении разговорного ИИ.

На данный момент у нас есть чат-бот, который умеет выполнять три действия:

1. Когда пользователь открывает чат, бот приветствует его: «Здравствуйте! Я Cake Bot. Чем я могу вам помочь?»
2. Что бы пользователь ни написал далее, чат-бот отвечает, что не понимает его.
3. После этого бот предлагает переключиться на оператора.

Ну и скука! Давайте лучше научим нашего бота правильно отвечать на вопросы.

2.1.2. Статический режим вопросов и ответов

Для начала построим модель компонентов чат-бота, задействованных в ответах на вопросы.

В некоторых платформах можно напрямую связать вопросы с ответами на них. В других же добавляется дополнительный уровень, группирующий схожие вопросы в так называемые намерения (или интенты). Система вопросов и ответов на основе намерений дает разработчику полный контроль над ответами, генерируемыми разговорным ИИ. Схема подобной системы представлена на рис. 2.2 на примере Cake Bot.



Рис. 2.2. Q&A-боты сопоставляют пользовательские высказывания с намерениями, а намерения — с ответами

Рассмотрим термины, используемые в схеме выше:

- *Высказывание* — это ввод пользователя. Для Q&A-бота это, соответственно, вопросы.
- *Намерение* — это логическая группа высказываний со схожим значением.
- *Ответ* — это вывод чат-бота. Для Q&A-бота это, соответственно, ответы.

Наличие намерений значительно экономит время, так как разработчику не нужно продумывать ответы на схожие вопросы. Вопросы «Во сколько вы открываетесь?» и «Какой у вас режим работы?» оба относятся к часам работы магазина. Для бота непринципиально, какая именно формулировка. Мы назначаем вопросам одинаковое значение через намерение `#store_hours`. А вот «Какие торты вы продаете?» имеет другое значение и, соответственно, другое намерение — `#cake_options`.

Для каждого намерения бот обучается на разных примерах высказываний. Современные системы на основе намерений требуют всего пять примеров для каждого из них. Неплохой компромисс, ведь существует почти бесконечное число способов спросить о времени работы магазина, и, предоставив лишь несколько примеров, можно уже качественно обучить бота.

Однако вопросно-ответные системы на основе намерений далеко не идеальны и имеют как преимущества, так и недостатки.

Преимущества:

- Вы полностью контролируете ответ. Можно редактировать текст, форматировать его и даже добавлять графические элементы. Содержание ответа определено заранее.
- Для небольшого числа намерений настройка бота выполняется быстро. Первого чат-бота можно собрать всего за час.

Недостатки:

- С ростом числа намерений становится все сложнее обучать бота их распознаванию.
- Ответы не учитывают нюансы в вопросах пользователя. На вопрос «Вы сегодня открыты?» бот все равно ответит стандартно: «Мы работаем без выходных».

- Неточные или плохо настроенные ответы вызывают у пользователя ощущение, что бот его не понимает.

Мы рассмотрим несколько способов устранения таких недостатков в следующих главах: узнаем как собирать данные, подходящие для обучения бота (глава 4); как использовать их для формирования более точных намерений (глава 5); как дополнять намерения ответами из документов и генеративного ИИ (глава 6); а также как применять генеративный ИИ в задачах обучения и тестирования (глава 7).

Начнем с обучения чат-бота его первым ответам. Для каждого вопроса нам нужны: намерение, набор связанных с ним пользовательских высказываний и ответ. Первый набор вопросов и ответов содержит информацию о самой компании (background), режиме работы магазинов (store hours), видах тортов (cake options), их примерной стоимости (cost) и программе лояльности (Cake Club). Все ответы на основе намерений приведены в табл. 2.2.

Таблица 2.2. Начальный набор FAQ-намерений с примерами высказываний и ответами

Намерение	Примеры высказываний	Ответ
`#background`	Расскажи о Cake Shop. Как давно существует ваша пекарня? История Cake Shop	Наша пекарня была основана бабушкой нынешнего владельца в 1980 году, и за все время существования мы испекли более 10 000 тортов!
`#store_hours`	Часы работы Какой у вас режим работы? Когда вы открыты?	Мы работаем с понедельника по пятницу, с 9:00 до 21:00
`#cake_options`	Виды тортов Вы делаете свадебные торты? Какие торты вы продаете?	Мы изготавливаем торты для самых разных случаев: свадьбы, дни рождения, годовщины, юбилеи, а также обычные торты, чтобы просто порадовать себя
`#cost`	Сколько стоит торт? Есть ли минимальная сумма заказа? Доставка платная?	Наши торты обычно стоят около $30, доставка — $5
`#cake_club`	Программа лояльности Cake Club Есть ли у вас акции или скидки?	Наша программа лояльности Cake Club дарит вам подарочный сертификат на $10 за каждые десять приобретенных тортов

На странице «Создайте свое первое действие» (`Create your first action`) с помощью кнопки «Создать действие» (`Create action`) вы можете определить

действие, которое будет выявлять намерение и выдавать ответ. Это действие ответа на вопрос — самый простой тип, доступный на любой платформе разговорного ИИ. На рис. 2.3 показан пользовательский интерфейс, с которого начинается определение действия.

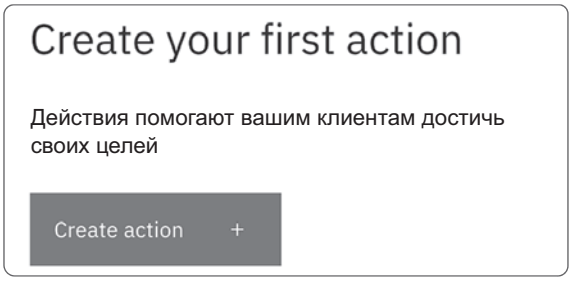


Рис. 2.3. Пользовательский интерфейс для создания первого действия

Каждое действие необходимо настроить на странице «Добавьте примеры фраз» (`Add example phrases`) — установить, с чего оно будет начинаться (пользовательские высказывания) и что будет происходить (ответ). Это вторая и третья колонки в табл. 2.2. Некоторые платформы разговорного ИИ используют метку намерения как название действия, в нашей же системе действие обозначается одной из пользовательских фраз, которая его запускает. Начнем с определения высказываний, запускающих действие `#background`, и введем в строку «Введите фразу» (`Enter a phrase`) одно из высказываний, как показано на рис. 2.4.

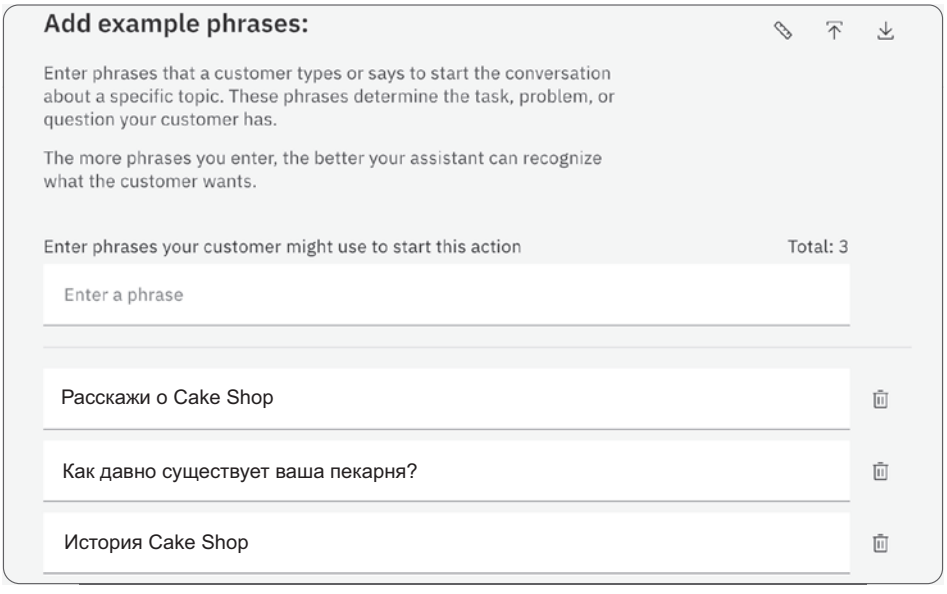


Рис. 2.4. Определение высказываний, запускающих действие

Интерфейс платформы советует: чем больше примеров пользовательских высказываний вы предоставите, тем лучше чат-бот будет распознавать действие. В качестве иллюстрации возьмем по три примера для каждого действия (`Total: 3`) — для начала этого достаточно. В последующих главах мы покажем различные способы поиска дополнительных обучающих примеров.

Действие почти настроено. У нас уже есть вопросы, которые будут его запускать, осталось только установить ответ на шаге 1 (`Step 1`) в поле «Определить ответ клиента» (`Define customer response`), как показано на рис. 2.5. В итоге наше действие состоит из трех частей:

- *Условная логика* — для статического действия ответа на вопрос логика не требуется. Действие запускается при определении намерения. (`The intent is enough, no other conditional logic`. — «Намерения достаточно, никакой другой условной логики».)
- *Ответ* — «Ответ помощника» (`Assistant says`) — это то, что получит пользователь. В нашем случае ответ — это простой текст. (`Responds with this text`. — «Ответить этим текстом».)
- *Следующий шаг* — для статического действия следующий шаг не требуется, выдача ответа завершает действие. (`Action complete! Nothing else to do`. — «Действие завершено! Больше ничего делать не нужно».)

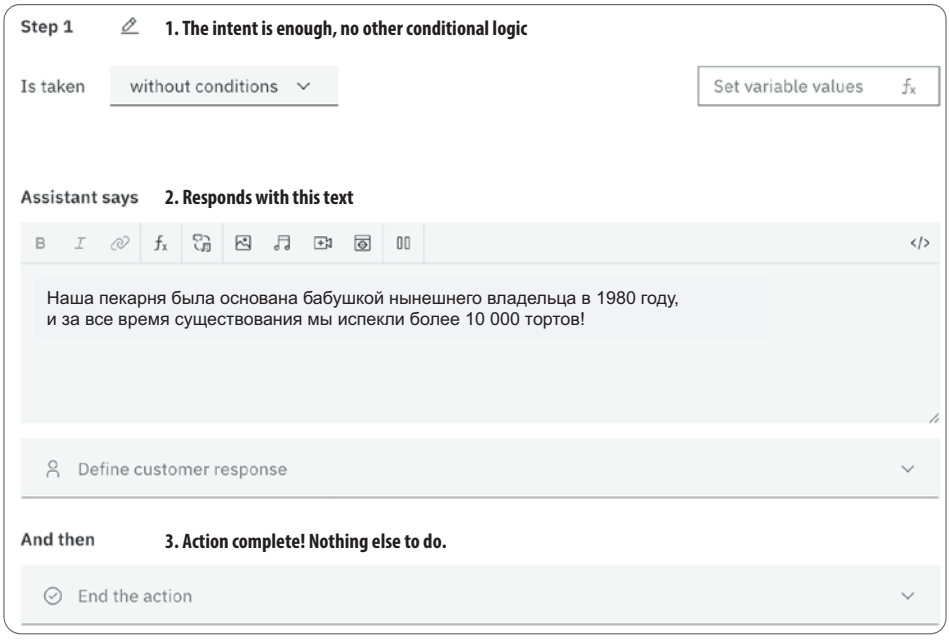


Рис. 2.5. Определение ответа на вопрос. Простейшая форма состоит только из одного шага — выдачи ответа

Аналогичным образом добавляем действия для остальных намерений. Действия будут обучаться на примерах, их запускающих, и на ответах, которые они должны выдавать. Каждое действие состоит только из одного шага и завершается после выдачи ответа.

Когда все пять действий будут созданы, можно переходить к тестированию. На рис. 2.6 показан интерфейс тестирования нашего чат-бота.

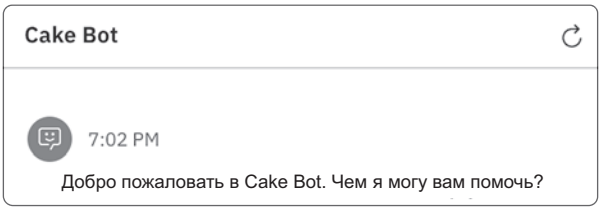


Рис. 2.6. Предварительный просмотр чата

Давайте зададим несколько вопросов! На рис. 2.7 приведен тест одного из них.

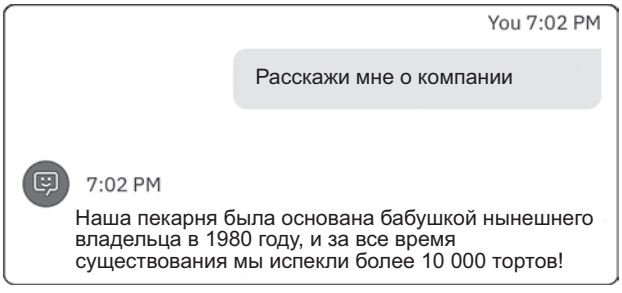


Рис. 2.7. Пример ответа от Cake Bot

Обратите внимание, что заданный вопрос дословно не совпадает ни с одним из обучающих примеров. Следовательно, боту удалось правильно распознать их смысл. В листинге ниже приведены дополнительные тесты бота.

Листинг 2.1. Тестирование Cake Bot дополнительными вопросами

```
Пользователь: когда работаете?
Бот: Мы работаем с понедельника по пятницу, с 9:00 до 21:00.
Пользователь: почему курица перешла дорогу
Бот: К сожалению, я вас не понимаю. Пожалуйста, переформулируйте вопрос.
Пользователь: цена торта?
Бот: Наши торты обычно стоят около $30, доставка — $5.
```

Отличное начало для нашего бота. Теперь мы можем обучить его большему числу намерений или сделать его более точным, добавив новые примеры, но давайте лучше рассмотрим иной подход.

Все созданные нами действия одноступенчатые. Это означает, что пользователь получает одинаковый ответ независимо от деталей вопроса. В следующем разделе мы рассмотрим, как превратить статический ответ в динамический, основанный на дополнительной информации.

2.1.3. Динамические ответы на вопросы

У Cake Shop в настоящее время есть четыре филиала: в Колумбусе, Дублине, Вестервиле и Грандвью. На момент создания бота все четыре магазина имели одинаковый график работы: с 9:00 до 21:00 по будням. Однако обстоятельства изменились — магазин в Колумбусе теперь открывается и закрывается на час раньше (с 8:00 до 20:00). По этой причине одного ответа чат-бота для всех магазинов недостаточно. Теперь, когда пользователь спрашивает о часах работы, необходимо выяснить, для какого магазина нужно предоставить ответ. Если клиент сам этого не сообщил, боту придется задать уточняющий вопрос.

Листинг ниже демонстрирует, как бот должен обрабатывать вопросы о часах работы.

Листинг 2.2. Примеры диалогов о часах работы в зависимости от местоположения

```
Пользователь: часы работы?
Бот: Чтобы узнать часы работы магазина, выберите его местоположение.
Бот: (Колумбус, Дублин, Вестервиль, Грандвью)
Пользователь: Колумбус
Бот: Наш магазин в Колумбусе работает с понедельника
по пятницу, с 8:00 до 20:00.

Пользователь: часы работы?
Бот: Чтобы узнать часы работы магазина, выберите его местоположение.
Бот: (Колумбус, Дублин, Вестервиль, Грандвью)
Пользователь: Дублин
Бот: Наш магазин в Дублине работает с понедельника по пятницу, с 9:00 до 21:00.

Пользователь: часы работы в Грандвью?
Бот: Наш магазин в Грандвью работает с понедельника
по пятницу, с 9:00 до 21:00.
```

Неоднозначный вопрос уточняется перед ответом

Однозначный вопрос получает прямой ответ

На рис. 2.8 показана блок-схема этого диалога: с ней удобнее строить диалоги, если вопросы предполагают динамику. Некоторым членам команды может быть удобнее работать с диаграммами, другим — с примерами диалогов, а кому-то понадобятся оба варианта.

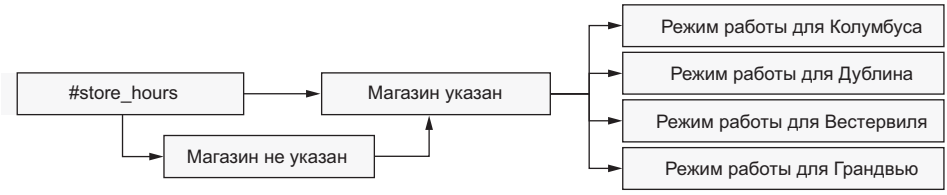


Рис. 2.8. Схема процесса для намерения `#store_hours`, зависящего от местоположения магазина

Это намерение может быть реализовано в три шага (`Conversation steps`):

1. Отобразить сообщение «Чтобы узнать часы работы магазина, выберите его местоположение» и список городов. Пользователь должен выбрать нужный ему город.
2. Если на шаге 1 выбран вариант «Колумбус», отобразить режим работы для магазина в Колумбусе и завершить действие (`Action complete`).
3. Отобразить режим работы для магазина в городе, выбранном на шаге 1, и завершить действие.

Подобная схема работает, потому что платформа поддерживает механизм «fall through» для шагов. Вот как могут выглядеть диалоги:

- Пользователь вводит «режим работы», срабатывает шаг 1. Пользователь выбирает «Колумбус», срабатывает шаг 2, и действие завершается.
- Пользователь вводит «режим работы», срабатывает шаг 1. Пользователь выбирает «Грандвью», условие шага 2 не выполняется. Срабатывает шаг 3, и действие завершается.
- Пользователь вводит «режим работы в Колумбусе». Условия шага 1 выполняются, поэтому срабатывает шаг 2, и действие завершается.
- Пользователь вводит «режим работы в Грандвью». Условия шага 1 выполняются, но условие шага 2 не выполняется. Срабатывает шаг 3, и действие завершается.

На рис. 2.9 показано, как данные шаги реализованы в нашем помощнике.

Cake Bot успешно запущен. Он может отвечать на базовые вопросы о Cake Shop и обладает некоторой динамикой. Бабушке владельца пекарни больше не придется отвечать на одни и те же вопросы по телефону! Но пока что Cake Bot не способен выполнять какие-либо действия за пользователя. Это мы рассмотрим в следующем разделе.

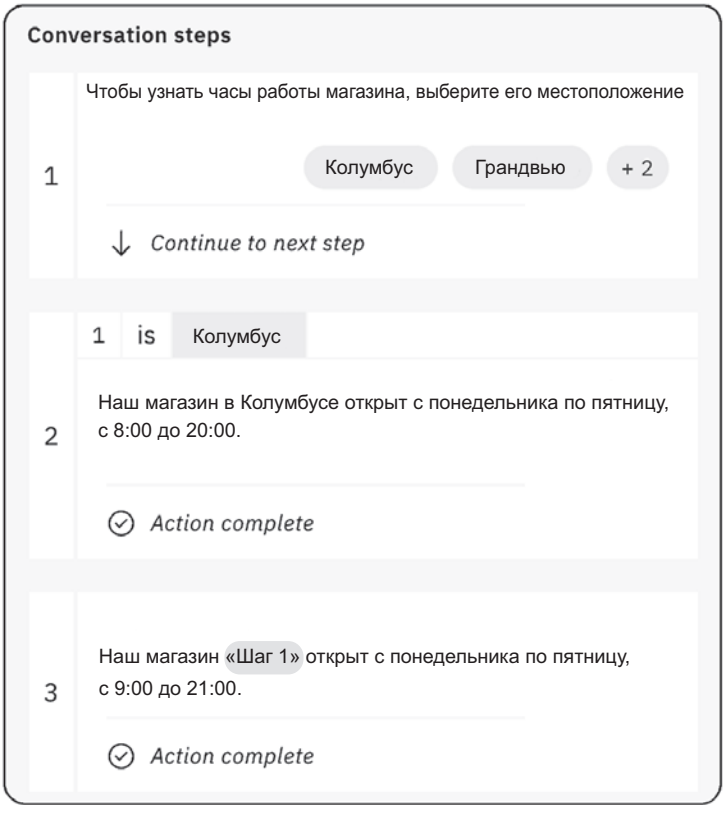


Рис. 2.9. Три шага для действия `#store_hours`

Задания

1. Скачайте код чат-бота из этой главы из репозитория книги на GitHub: https://github.com/andrewrfreed/EffectiveConversationalAI. Загрузите чат-бота в watsonx Assistant и используйте панель Preview («Предпросмотр») для тестирования диалогов.
2. Либо реализуйте Cake Bot на выбранной вами платформе разговорного ИИ:
   - Придумайте приветственное сообщение.
   - Установите резервное намерение и/или резервное сообщение.
   - Реализуйте пять намерений из табл. 2.2.

2.2. Маршрутизирующие агенты и процессно-ориентированные боты

Не все боты только лишь отвечают на вопросы. Q&A-боты отлично подходят для предоставления базовой информации, но что, если пользователю нужно больше, чем просто ответ: ему нужно действие? Для Cake Shop было бы идеально, если бы клиенты могли через бота заказывать торты. Если же наш бот умеет только отвечать на вопросы, то в лучшем случае его можно настроить так (рис. 2.10):



Рис. 2.10. Процесс заказа торта в Cake Shop в форме ответа на вопрос. Но это не решение!

Пользователь хочет завершить процесс, но не может сделать это внутри бота. Он получает только *инструкции*, как завершить процесс. Поэтому Q&A-бот зачастую представляет собой лишь основу для более совершенного решения.

2.2.1. Маршрутизирующие агенты

Cake Shop предлагает широкий ассортимент тортов с самыми разнообразными вкусами и вариантами оформления. Есть дизайны для свадеб, выпускных, дней рождения и других мероприятий. Варианты вкусов: ваниль, шоколад, клубника. Плюс разные способы оплаты и доставки. Учитывая разнообразие опций, можно предположить, что пользователю лучше обсудить все детали с человеком.

Поэтому наш следующий шаг — построение маршрутизирующего агента, который будет определять намерение пользователя и решать, кто лучше всего сможет это намерение выполнить. На рис. 2.11 показана обновленная концепция Cake Bot с функцией маршрутизации.

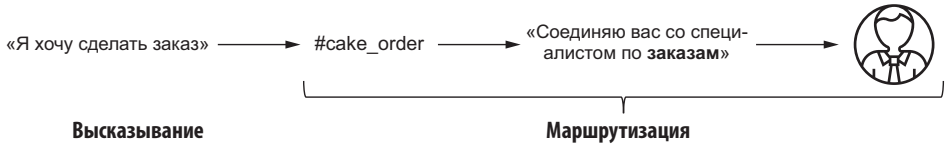


Рис. 2.11. Маршрутизирующий агент определяет намерение пользователя и направляет его к соответствующему специалисту

При исходных Q&A-запросах бот продолжает работать так же, как и раньше. Но при заказе торта он будет сразу перенаправлять вызов к специалисту.

Реализация показана на рис. 2.12. Действие состоит из одного шага: как только намерение определено, пользователь перенаправляется на человека (Connect to agent (action ends) — «Переключение на оператора (действие завершается)»).

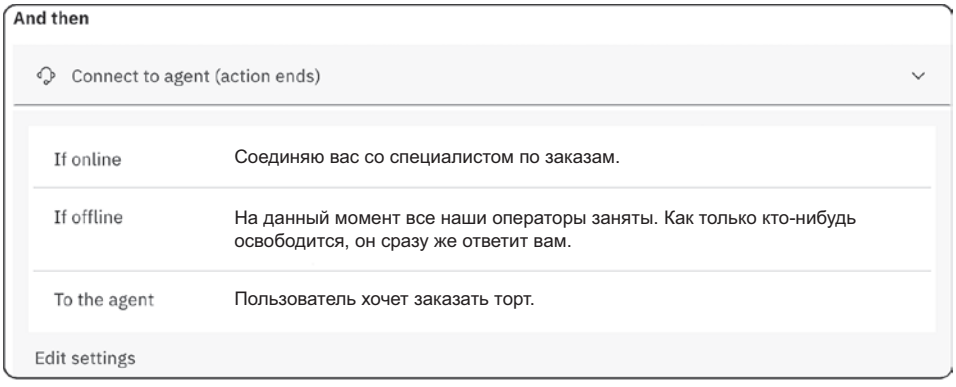


Рис. 2.12. Конфигурация маршрутизирующего агента для `#cake_orders`. Как только намерение распознано, пользователь перенаправляется к оператору-человеку

Маршрутизирующий агент лишь сортирует входящие запросы, которые могут быть переданы операторам-людям или специальным ИИ-решениям. Операторы могут работать как по телефону, так и в формате онлайн-чата. В этой книге мы будем называть таких людей *операторами колл-центра*.

Для записи на прием нажмите 1…

Вы наверняка хотя бы раз звонили в IVR-систему (интерактивное голосовое меню), которая зачитывает все опции и предлагает выбрать одну из них («для записи на прием нажмите 1»). Это тоже маршрутизирующий агент. Недостаток таких систем — время, которое пользователь тратит на прослушивание автоответчика. Агент на основе разговорного ИИ позволяет клиенту просто сказать, чего он хочет, и это намного удобнее, чем прослушивать текст с описанием десятка пунктов меню.

Маршрутизирующие агенты позволяют внедрять решения на базе разговорного ИИ итеративно, без необходимости сразу охватить все возможные сценарии.

Операторы в системах маршрутизации обычно знают, какого типа запрос они получают, но не обладают дополнительной информацией. В примере на рис. 2.12 им известно только то, что пользователь хочет заказать торт. Для некоторых особо сложных или чувствительных процессов это подходит. Например, высказывание с намерением «сообщить о мошенничестве» должно немедленно перенаправляться к человеку.

В других же сценариях слишком раннее перенаправление к оператору может затруднить работу самого сотрудника и быть затратным для работодателя. Например, в системах страховых компаний при обработке статусов заявлений необходимо сначала узнать ID клиента и дату подачи заявления, а затем уже перейти к более сложным задачам, например объяснениям, что происходит со статусом заявления. В таких случаях ИИ-помощник может сначала запросить ID клиента и дату подачи заявления и только затем перенаправить диалог к человеку.

Таким образом, следующий шаг развития маршрутизирующего агента — автоматизация большей части работы. Давайте реализуем это для Cake Bot.

2.2.2. От маршрутизирующего агента к процессно-ориентированному боту

Общая схема процесса заказа тортов показана на рис. 2.13. Она включает в себя четыре шага для уточнения деталей заказа, затем шаг подтверждения и, наконец, исполнение заказа. Для краткости опустим детали заказа — пример кода доступен в нашем репозитории на GitHub: https://github.com/andrewrfreed/EffectiveConversationalAI.

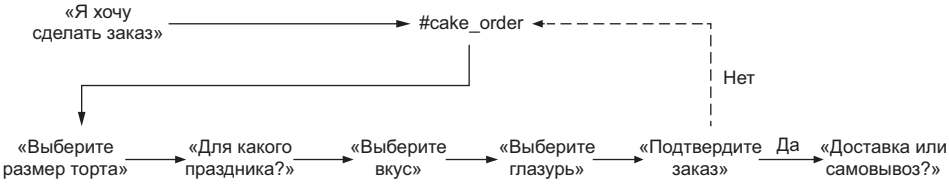


Рис. 2.13. Процесс заказа торта в Cake Shop

Разработав полный процесс заказа торта, вместо маршрутизирующего агента мы получим процессно-ориентированного бота. Cake Bot будет обрабатывать часть процесса, собирая некоторые детали заказа перед тем, как передать его оператору для завершения. На рис. 2.14 показан дизайн первой итерации Cake Bot.

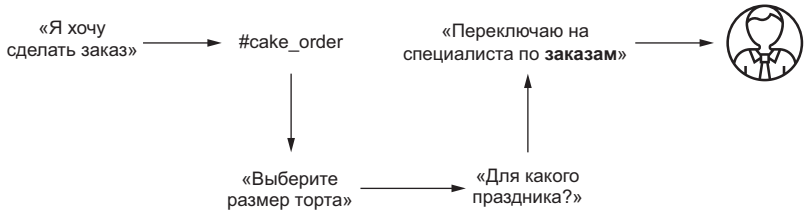


Рис. 2.14. Переход от маршрутизирующего агента к процессно-ориентированному боту. Теперь бот собирает часть информации о заказе перед передачей его человеку

Раньше весь процесс состоял из одного шага (рис. 2.12). Теперь их будет четыре:

1. Бот начинает процесс, отвечая: «Я помогу вам с заказом».
2. Запрашивает размер торта (`Size`) и предлагает варианты: маленький, средний, большой.
3. Спрашивает, для какого мероприятия заказывается торт, и предлагает варианты (день рождения, свадьба, годовщина, универсальный).
4. Переводит пользователя на оператора. Раньше это был первый шаг агента, но теперь сообщение оператору изменилось с «Пользователь хочет заказать торт» на «Пользователь хочет заказать <размер> торт для <мероприятие>». Помощник сам вставит ответы из шагов 2 и 3 в сообщение.

Данные шаги выполняются последовательно. На рис. 2.15 показан шаг 2 (`Step 2`).

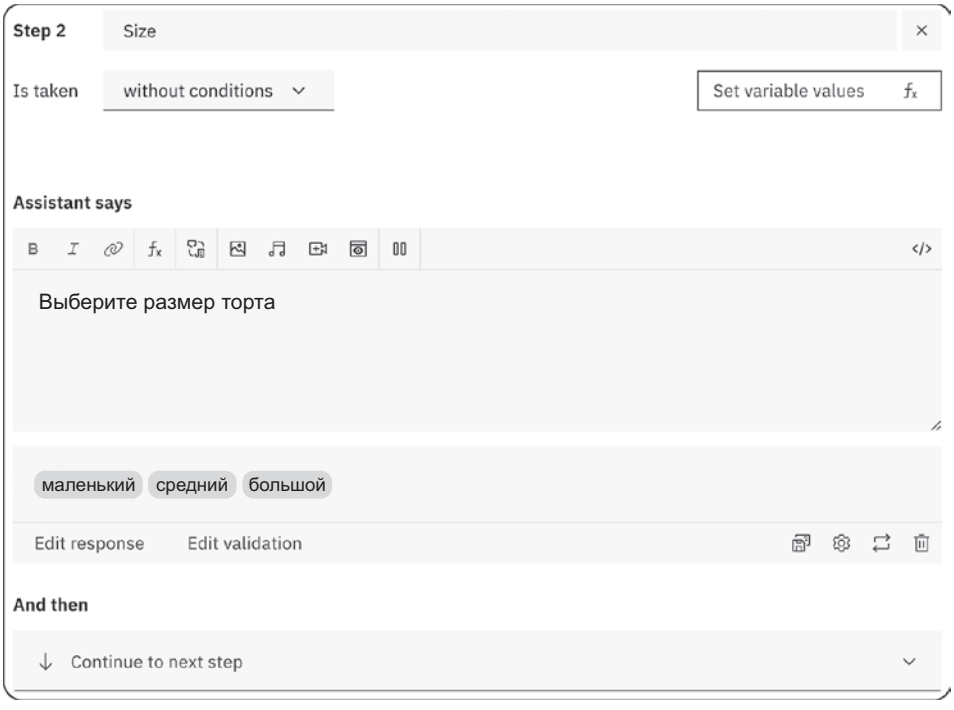


Рис. 2.15. Шаг для сбора информации о размере торта предлагает пользователю варианты ответа

Бот может предлагать пользователю варианты ответа в виде кнопок или же при желании позволить ввести свой ответ. Все варианты можно найти в примере кода, представленном в репозитории книги на GitHub.

Также обратите внимание, что каждая из опций поддерживает механизм fall through. Если пользователь начинает диалог с фразы «Я хочу заказать торт», ему будут заданы вопросы о его размере и планируемом мероприятии. Если же он пишет «Я хочу заказать большой торт на годовщину свадьбы», вопросы будут пропущены, так как эта информация уже предоставлена.

Оставшаяся часть процесса заказа торта реализована в репозитории книги на GitHub в соответствии со схемой, представленной в начале раздела. Ниже приведен пример диалога.

Листинг 2.3. Пример диалога при заказе торта

```
Пользователь: Я хочу заказать торт.
Бот: Я помогу вам оформить заказ.
Бот: Какого размера нужен торт?
Бот: (маленький, средний, большой)
Пользователь: большой
Бот: (день рождения, свадьба, годовщина, универсальный)
Пользователь: день рождения
Бот: Какой вкус предпочитаете?
Бот: (ванильный, шоколадный, мраморный)
Пользователь: ванильный
Бот: Какую глазурь вы хотите?
Пользователь: клубничную
Бот: Хорошо, вам нужен маленький ванильный торт
ко дню рождения с клубничной глазурью. Все верно?
Пользователь: да
Бот: Отлично. Ваш заказ принят.
Стоимость вашего заказа составит $30,00.
Бот: Хотите оформить заказ на самовывоз или доставку?
Обратите внимание, что стоимость доставки – $5.
Бот: (самовывоз, доставка)
Пользователь: доставка
(детали опущены)
```

Подтверждающее сообщение на шаге 7 воспроизводит информацию, собранную на предыдущих шагах

Подтверждение заказа на шаге 8 запускает условную логику для расчета стоимости торта

Cake Bot становится все более функциональным. Он уже умеет давать статические ответы на вопросы о тортах, динамические ответы на вопросы о режиме работы и имеет процессно-ориентированный поток для заказа тортов. Команда Cake Shop разворачивает своего чат-бота и довольна результатами (а пользователи довольны тортами!). Далее мы перейдем к последней задаче этой главы — добавление возможностей генеративного ИИ с использованием большой языковой модели (LLM).

Задания

1. Вернитесь к коду чат-бота этой главы, который вы скачали из репозитория книги на GitHub (https://github.com/andrewrfreed/EffectiveConversationalAI), загрузите бота в watsonx Assistant и используйте панель предпросмотра для тестирования процесса заказа торта.

2 Либо реализуйте процесс заказа в Cake Bot на выбранной вами платформе разговорного ИИ:
   - Определите намерение заказать торт.
   - Направьте намерение к оператору-человеку.
   - Соберите все четыре параметра заказа и завершите процесс.

2.3. Ответы пользователю с помощью генеративного ИИ

Пока что наш Cake Bot использует только традиционные технологии разговорного ИИ. Ответы на вопросы реализованы через классификатор намерений, а процесс заказа — через последовательную серию правил. Бот выполняет все необходимые для пекарни задачи.

Однако в ходе анализа работы Cake Bot команда Cake Shop заметила необычную тенденцию: пользователи просят у бота рецепты, что можно приготовить на ужин перед подачей десерта. Никакой другой закономерности в запросах нет — встречаются запросы на запеканки, салаты, блюда на сковороде и многое другое. Команду радует активность их пользователей, но они не знают, как обрабатывать подобные запросы. Как можно распознать разные типы желаемых рецептов и тем более предоставить их?

Именно здесь нам и поможет генеративный ИИ. Можно использовать уже существующий механизм намерений для определения запросов на рецепты и затем направлять их в LLM для генерации ответа. Для этого потребуется интеграция LLM в чат-бота и механизм маршрутизации конкретных запросов в модель. Давайте посмотрим, как это можно сделать.

2.3.1. Интеграция с LLM

Для многих платформ разговорного ИИ основным способом интеграции с внешними системами являются программные интерфейсы (API). Это повсеместно используемые шаблоны интеграции, которые поддерживаются большинством генеративных ИИ-платформ, предоставляющих доступ к LLM. Конкретный способ интеграции API в разговорный ИИ зависит от платформы. На одних платформах интеграция выполняется с помощью кода, на других — посредством low-code или визуальных интерфейсов. Разные платформы используют разные термины для обозначения своих возможностей интеграции, например *расширение* (extension), *интеграция* (integration) и *фулфилмент*, или *заполнение* (fullfillment). Многие из них позволяют подключать API через спецификации OpenAPI.

Мы добавим генеративную ИИ-платформу как расширение для выполнения генерации текста на основе LLM. Для этого в нашей платформе есть четыре шага (подробнее они приведены в репозитории книги на GitHub).

1. В меню «Интеграции» (`Integrations`) выберите «Построить кастомное расширение» (`Build a Custom Extension`).
2. Укажите имя и описание, например «API для генеративного ИИ».
3. Загрузите файл спецификации OpenAPI, документирующий возможности расширения, включая методы (`Method`), которые оно предоставляет, обязательные (`Required`) и необязательные (`Optional`) параметры (`Request Parameters`), а также свойства формируемых им ответов (`Response properties`). Файлы спецификаций OpenAPI — распространенный формат документации для API, предоставляемой генеративными ИИ-платформами.
4. Укажите сведения о подключении и аутентификации, такие как URL реализации API и ключ API, необходимый для доступа к нему.

Добавляем расширение и исследуем его с помощью визуального интерфейса. На рис. 2.16 показано расширение для API генерации текста (`Generation`) с привлечением LLM.

Operation	Method	Resource
Generation	POST	/ml/v1/text/generation

Request parameters

version string | Required	
input string | Required	
model_id string | Required	
parameters.top_k integer | Optional	
parameters.top_p number | Optional	
parameters.time_limit integer | Optional	

Response properties

results[ ].stop_reason string
results[ ].generated_text string
results[ ].input_token_count integer
results[ ].generated_token_count integer
model_id string
created_at string

Рис. 2.16. Спецификация OpenAPI для API генерации текста с помощью LLM со множеством возможных параметров запроса (`Request parameters`)

На момент написания книги наш API включает 15 входных и 6 выходных параметров — больше чем поместилось на рис. 2.16! Также есть несколько параметров, доступных без какой-либо настройки, например HTTP-код состояния для ответа. Другие платформы генеративного ИИ будут иметь схожий набор параметров, возможно, с другими названиями или расположением. Рассмотрим наиболее значимые из них:

- `input` (запрос) — запрос к LLM. Он включает инструкции, контекст и данные для LLM. Некоторые из этих данных могут поступать напрямую от пользователя.
- `model_id` (запрос) — идентификатор LLM, используемой для выполнения задачи. Большинство платформ генеративного ИИ позволяют выбрать из нескольких моделей.
- `parameters` (запрос) — пары «ключ — значение», задающие поведение LLM. Сюда входит метод декодирования (жадный или с применением семплирования), количество выходных токенов для генерации и ряд других параметров.
- `generated_text` (ответ) — результат работы LLM.

Расширение можно использовать на любом этапе любого действия. Ранее в этой главе мы использовали такие возможности, как «Ответ помощника», «Следующий шаг» и «Переключение на оператора». Для расширений нам нужна возможность «Использовать расширение» (`Use an extension`). На рис. 2.17 показан вызов расширения для нашего действия с рецептами. Реализация других задач с использованием LLM будет выглядеть схожим образом, но с иными значениями конфигурации. Здесь набор параметров настроен для вывода рецептов.

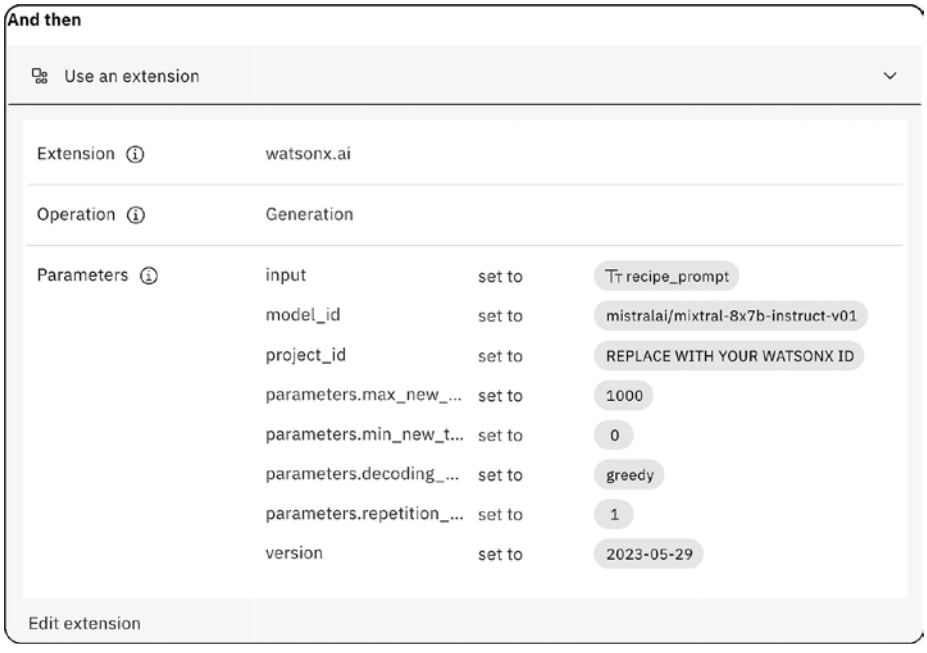


Рис. 2.17. Вызов API генерации текста с помощью LLM из действия в помощнике

Добавим это в Cake Bot.

2.3.2. Маршрутизация запросов к LLM

Схема на рис. 2.18 описывает, как будет реализована генерация рецептов в Cake Bot. Сначала мы создаем новое действие. Как и в случае с ответами на вопросы, начинаем с примеров высказываний, которые его активируют. Вводим три примера: «Покажи рецепт», «Как можно приготовить» и «Расскажи, как испечь». Учитывая огромное разнообразие возможных рецептов, мы не включаем названия блюд, а только фразы, которые, скорее всего, будут в запросах.

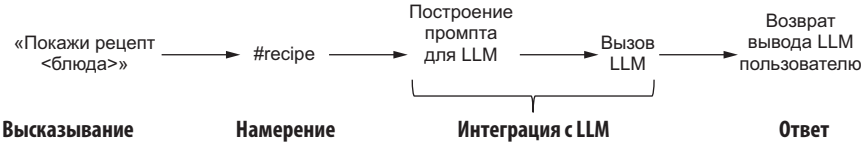


Рис. 2.18. Диаграмма процесса генерации рецептов в Cake Bot через LLM

Шаг 1 нового действия — сохранить целиком исходное высказывание пользователя (из системной переменной `input.text`) в переменной `recipe_query_text`. Подобный прием мы ранее не использовали — для заказа торта каждая опция имела фиксированный и конечный набор ответов. Даже если пользователь говорил «большой торт, пожалуйста», мы сохраняли только «большой». В случае с запросом рецепта мы не знаем, что именно скажет пользователь, поэтому фиксируем всю его фразу.

Шаг 2 — составить промпт для LLM. Мы объединяем простой системный промпт с запросом пользователя. В листинге ниже показано выражение, используемое при построении переменной `recipe_prompt`.

Листинг 2.4. Промпт к LLM, сохраняемый в переменной recipe_prompt

```
"Ты — помощник на кухне. Придумай рецепт по запросу пользователя. \n\nInput:
".append(recipe_query_text).append("\n\nOutput:")"
```

Шаг 3 — вызов LLM. Параметры были показаны на рис. 2.17, но давайте рассмотрим значения подробнее:

- `input` — устанавливаем переменную `recipe_prompt` как вход. Это позволит вставить запрос пользователя в общий промпт, показанный в листинге 2.4.
- `model_id` — доступно множество моделей, но на момент написания книги хорошие результаты для задачи генерации показала модель mistralai/mixtral-8x7b-instruct-v01.
- `project_id` — идентификатор проекта на платформе генеративного ИИ.
- `min_tokens` и `max_tokens` — установлены на 0 и 1000 соответственно (значения по умолчанию — 0 и 200), так как рецепты обычно достаточно длинные.

- `decoding_method` — установлено значение `greedy` для удобства отладки, чтобы модель давала один и тот же ответ на одинаковые входы.
- `repetition_penalty` — установлено значение 1 (без штрафа), так как рецепты в интернете обычно содержат определенное количество повторов.
- `version` — указывает минорную версию API. Мы использовали актуальную на момент написания книги: 29-05-2023.

На шаге 4 мы проверяем, что вызов API прошел успешно, и если это так, то возвращаем ответ пользователю. Итоговый ответ будет складываться из статического текста «Вот ваш рецепт» и ответа LLM. Нам не нужно писать какой-либо код для парсинга JSON, в спецификации OpenAPI указано, что мы можем обращаться к значению `generated_text`. На рис. 2.19 показана конфигурация действия.



Рис. 2.19. Вывод ответа LLM пользователю

Теперь наш Cake Bot умеет еще больше! В листинге 2.5 приведен пример тестового диалога с ботом. Рецепт получился достаточно длинным, поэтому бóльшая его часть опущена. Вы можете самостоятельно запустить Cake Bot и протестировать новый функционал.

Листинг 2.5. Пример диалога с Cake Bot

```
Пользователь: Мне нужен быстрый рецепт пирога с арахисовой пастой.
Бот: Конечно! Ищу подходящий рецепт. Пожалуйста, подождите несколько секунд.
Бот: Вот ваш рецепт:
...
Ингредиенты:
1 готовая основа для пирога
1 стакан арахисовой пасты
...

Инструкции:
В большой миске смешайте арахисовую пасту, сахарную пудру и мягкий сливочный сыр
до однородной кремовой массы.
...
Подавайте охлажденным из наслаждайтесь быстрым и простым в приготовлении пирогом
с арахисовой пастой!
Если вам потребуется дополнительная помощь или пояснения, обращайтесь.
```

Задания

1. Скачайте код чат-бота из репозитория книги на GitHub (https://github.com/andrewrfreed/EffectiveConversationalAI). Загрузите его в watsonx Assistant и следуйте инструкциям по интеграции с watsonx.ai. С помощью панели предварительного просмотра `Preview` протестируйте сценарий рецептов.
2. Либо реализуйте Cake Bot на выбранной вами платформе разговорного и генеративного ИИ:
   - Определите намерение пользователя (получить рецепт).
   - Сформируйте промпт из набора инструкций и пользовательского ввода.
   - Отправьте ответ LLM пользователю.

Итоги

- Боты для ответов на вопросы (Q&A-боты) — отличный способ начать разработку вашего первого решения на основе разговорного ИИ.
- Обучение Q&A-ботов на примерах вопросов позволяет заранее подготовить ответы для схожих запросов (намерений).
- Любое действие начинается с намерения и может иметь множество исходов: ответ на вопрос, перевод пользователя на оператора, уточняющие вопросы или вызовы API.
- Маршрутизирующий агент распознает намерения и передает информацию операторам, что позволяет расширить возможности разговорного ИИ человеческими способностями.
- Разговорный ИИ может использовать как традиционные методы работы, так и методы, основанные на правилах и задействующие генеративный ИИ.

3 Планирование улучшений

В этой главе

- ✓ Формирование межфункциональной команды для успешной работы с разговорным ИИ
- ✓ Определение успеха через бизнес-цели, ключевые метрики и «болевые точки» пользователей
- ✓ Анализ эффективности улучшений с использованием результатов и метрик
- ✓ Реализация структурированных процессов для выявления, фиксации, классификации и приоритизации проблем

Каждое решение на базе разговорного ИИ должно разрабатываться с учетом критериев успеха, определение которых зависит от типа чат-бота. Например, Q&A-боты должны предоставлять быстрые и точные ответы при минимальном числе уточняющих взаимодействий. Процессно-ориентированные или транзакционные боты должны эффективно направлять пользователей к достижению конкретных целей. Маршрутизирующие агенты должны беспрепятственно переводить пользователей на нужных специалистов.

Ошибки в распознавании намерений пользователя либо излишняя сложность бота могут стать причинами отказа от взаимодействия и препятствовать

прогрессу в работе. Решение этих проблем повышает эффективность чат-бота и помогает команде достичь успеха. Компании, которые непрерывно совершенствуют свои разговорные системы, имеют наибольшие шансы достичь необходимых результатов.

Объединение специалистов с разными навыками внутри одной межфункциональной команды имеет решающее значение для успеха непрерывного улучшения. Члены такой команды способны инициировать различные изменения благодаря своим уникальным знаниям. Однако прежде всего команде необходимо договориться, какое именно улучшение они будут реализовывать.

Например, аналитик диалогов хочет упростить сценарии общения, но бизнес-менеджер желает донести до пользователей конкретную информацию. Кто же прав? В этой главе мы рассмотрим, как команда вымышленной компании MediWorld адаптировала и улучшила своего чат-бота. Изначально перед ними стояла цель усовершенствовать Q&A-бота, но по мере изменения пользовательских потребностей они перешли к разработке дополнительных возможностей процессно-ориентированного бота.

3.1. Когда необходимо улучшение

Представим следующую ситуацию.

MediWorld, крупная сеть аптек, столкнулась с перегруженностью колл-центров вопросами о пандемии. Для предоставления информации, связанной с COVID-19, они разработали специального чат-бота, который распознавал ограниченный набор вопросов о вирусе и выдавал достоверные ответы.

На рис. 3.1 показано, как PharmaBot определял намерения.

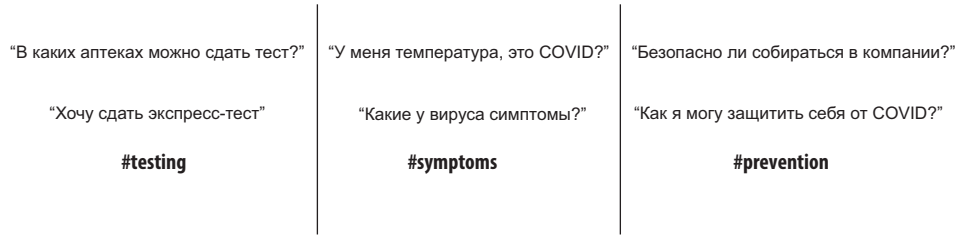


Рис. 3.1. PharmaBot эффективно распознавал информационные намерения в запросах пользователей

После создания вакцины характер вопросов клиентов резко изменился. Появились новые запросы:

- *Имею ли я право на вакцинацию?*
- *Могу ли я записаться на прививку?*

- *Когда я могу получить вторую дозу?*
- *Нужно ли звонить для записи или я могу сделать это здесь?*
- *Можно ли путешествовать после прививки?*

PharmaBot плохо справлялся с пониманием новых вопросов, часто отвечая: «Извините, я не уверен, что правильно вас понял. Пожалуйста, переформулируйте вопрос». Пользователи не получали необходимой информации и после неудачи в боте обращались в колл-центр. Также увеличилось число отказов от взаимодействия, что отражало разрыв между ожиданиями пользователей и возможностями PharmaBot. Команда MediWorld захотела улучшить бота, но сначала им нужно было решить, что именно означает «улучшить».

ПРИМЕЧАНИЕ Потребность в непрерывном улучшении обусловлена постоянно изменяющимися ожиданиями пользователей и технологическими достижениями, требующими адаптации. Бриджит ван Кралинген (Bridget van Kralingen) точно подметила: «Последний лучший опыт, полученный где угодно, становится минимальным стандартом для опыта, который люди теперь ожидают везде». Требования к улучшению могут исходить как из внутренних источников (например, поддержка новых функций), так и из внешних (например, события, породившие совершенно новые запросы).

Признание необходимости улучшений в разговорном ИИ является ключевым условием его эффективности и актуальности. Виртуальный помощник — это не статичное решение, его работа должна постоянно адаптироваться под поведение пользователей, потребности бизнеса и развитие технологий. Признаки того, что систему необходимо улучшить, часто проявляются через ключевые показатели эффективности (KPI), такие как низкий уровень удержания внутри бота, высокая частота использования резервного намерения или частые переходы к операторам. Планирование улучшений начинается с формирования межфункциональной команды, определения четких критериев успеха, анализа результатов и внедрения структурированных процессов управления проблемами.

Начинать улучшения лучше всего сразу после выявления первых повторяющихся проблем, снижения уровня вовлеченности или недостижения бизнес-целей. Проактивный подход позволяет не допустить перерастания мелких проблем в крупные. Регулярный анализ эффективности гарантирует, что улучшения соответствуют изменяющимся ожиданиям пользователей и целям компании.

Начните планировать комплексную стратегию сбора данных еще до первого развертывания. Помните, что наличие логов само по себе не выявляет проблемы, поэтому необходимо постоянно отслеживать тенденции и закономерности в пользовательских взаимодействиях. Многие команды склонны решать отдельные проблемы, при этом не учитывая их общий объем или частоту их возникновения. И хотя исправление единичных ошибок кажется продуктивным, на деле это редко приводит к значительным улучшениям общей эффективности. Концентрируясь на системных проблемах с существенным эффектом,

вы сможете направлять усилия команды на реальные результаты, сохраняя ее фокус и вовлеченность.

Помните, что измерение эффективности до и после внесения изменений одинаково важно. Определите исходные метрики перед внедрением исправлений и сравните их с данными после развертывания, чтобы оценить, привели ли изменения к ожидаемым результатам. Если результаты не совпадают с ожиданиями — не беда. Проведите более глубокий анализ первопричин и доработайте решение, чтобы устранить пробелы и разные непредвиденные проблемы. Такой ход работы позволит убедиться, что ваши усилия действительно приводят к прогрессу.

3.2. Ваша межфункциональная команда

Команда MediWorld поняла, как важен PharmaBot для предоставления их клиентам достоверной и оперативной информации, поэтому аналитики разговорных интерфейсов, сотрудники отдела по поддержке клиентов и аналитики данных — все объединились для оценки и улучшения работы бота. Целью такой междисциплинарной команды было не только устранение существующих проблем, но также прогнозирование и удовлетворение изменяющихся потребностей пользователей.

> *Разработчики MediWorld сосредоточились на совершенствовании возможностей PharmaBot по обработке естественного языка. Они нашли способы улучшить понимание запросов пользователей, чтобы бот мог предоставлять более точные и релевантные ответы.*
>
> *Одновременно с этим ведущие специалисты колл-центров делились ценными наблюдениями, сделанными на основе звонков, которые к ним переводились, выявляя типичные проблемы и часто задаваемые вопросы.*
>
> *Аналитики MediWorld исследовали данные о взаимодействиях пользователей. Они определили области, в которых чат-бот чаще всего ошибался, и классифицировали эти сбои по последней задаче, которую пытался выполнить бот.*

На рис. 3.2 показаны выводы, сделанные разными членами команды.

Теперь немного отойдем от примера PharmaBot и сосредоточимся на командах, которые обычно участвуют в улучшении разговорного ИИ. Конкретные роли, обязанности и размер команды могут варьироваться в зависимости от размера вашей компании, ее целей и сложности чат-бота. В небольших проектах один человек может совмещать несколько ролей. В главе 1 была представлена своеобразная «команда мечты» для разработки разговорного ИИ (рис. 1.5). Для улучшения и доработки существующих чат-ботов необходима не менее разнообразная группа специалистов. И хотя структура такой команды в разных компаниях будет различаться, она, как правило, состоит из трех ключевых подгрупп.

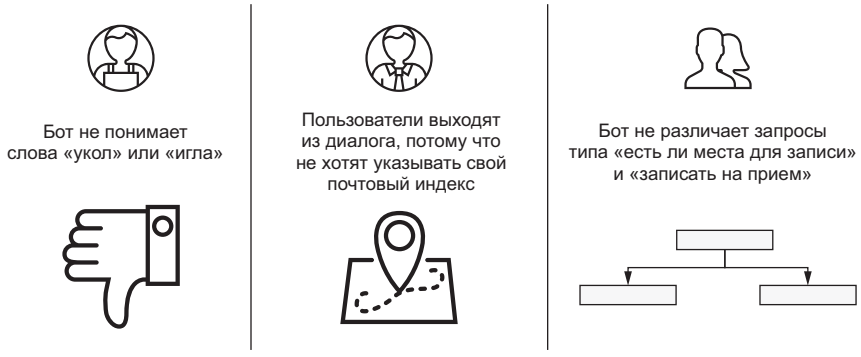


Рис. 3.2. Разнообразные навыки команды позволили выявить существующие проблемы бота, что заложило основу для эффективного плана оптимизации

Первая — это команда поддержки и сопровождения чат-бота. Она отвечает за анализ и оценку его эффективности. Кроме того, специалисты этой группы выступают в роли технических экспертов в предметной области (SME). Они знают текущие намерения, которые умеет обрабатывать чат-бот, обучающие данные и сценарии диалогов. Они способны вносить изменения в код и реализовывать его. Их роли и задачи приведены в табл. 3.1.

Таблица 3.1. Команда поддержки и сопровождения чат-бота

Роль	Задачи
Аналитик данных / дата-инженер	Анализирует взаимодействия пользователей и их обратную связь, чтобы подготовить обоснованные рекомендации по изменениям, исправлениям и улучшениям
Разработчик чат-бота / аналитик разговорных интерфейсов	Реализует технические изменения и улучшения чат-бота. Сюда относятся как новые интеграции (ближе к роли разработчика), так и обновления диалогов и действий чат-бота (роль аналитика разговорных интерфейсов)
Тестировщик отдела контроля качества (QA)	Проверяет, что изменение, исправление или улучшение дает ожидаемый результат и не приводит к непредвиденным или негативным последствиям. Тестирование может выполняться вручную или с использованием инструментов автоматизированного тестирования
Менеджер проекта	Координирует задачи, следит за тем, чтобы процесс непрерывного улучшения шел по графику
Другие SME	Обладают специализированными знаниями в конкретных аспектах экосистемы чат-бота, привлекаются при необходимости. Например, специалисты по безопасности оценивают потенциальные угрозы и рекомендуют соответствующие меры или стратегии устранения уязвимостей, чтобы обеспечить надежность и устойчивость чат-бота к новым рискам

Вторая подгруппа — бизнес-стейкхолдеры. Они в совокупности обеспечивают соответствие улучшений чат-бота целям компании и бизнес-потребностям. Бизнес-стейкхолдеры следят за тем, чтобы чат-бот был не только технически эффективным, но и соответствовал целям компании, потребностям пользователей и юридическим стандартам. Состав этой команды приведен в табл. 3.2.

Таблица 3.2. Бизнес-стейкхолдеры чат-бота

Роль	Задачи
Руководство компании	Участвует в согласовании улучшений и расстановке приоритетов согласно общим бизнес-стратегиям
Служба поддержки клиентов	Отвечает за бизнес-процессы, которые автоматизирует бот
Продакт-менеджер (чат-бота)	Контролирует направление развития чат-бота в соответствии с бизнес-целями
IT-отдел	Обеспечивает техническую поддержку и инфраструктуру для разработки, внедрения и сопровождения чат-бота
Операционный менеджер	Участвует в интеграции чат-бота в операционные процессы, оптимизируя рабочие процессы
Юридический отдел и отдел комплаенса	Обеспечивают соответствие улучшений отраслевым стандартам и юридическим требованиям

Последняя подгруппа — команда управления (governance). Их задача состоит в том, чтобы внедрение, использование и непрерывное улучшение чат-бота соответствовало политике компании, стандартам и этическим требованиям. Состав данной команды представлен в табл. 3.3.

Таблица 3.3. Команда управления чат-ботом

Роль	Задачи
Специалист по корпоративной этике / комплаенсу	Решает вопросы, связанные с этикой поведения чат-бота и принятием решений, а также с управлением рисками моделей ИИ. Также обеспечивает соблюдение руководящих принципов ответственного применения ИИ в процессе улучшений
Исполнительная управляющая команда	Принимает окончательные решения о приоритетах развития системы, бэклоге и всех затратах (на работу команд поддержки или бизнес-команд), связанных с ней

Наличие межфункциональной команды, отвечающей за улучшения, гарантирует, что при разработке и сопровождении чат-бота будут учтены точки зрения всех сторон процесса. Для эффективной работы команде необходимы регулярные

встречи, четкие каналы коммуникации и документирование политики управления. При этом конкретный состав участников зависит от характера и масштаба проекта.

Задания

1. Вспомните, как вы создавали вашего последнего чат-бота, и перечислите всех участников проекта. Проанализируйте их решения в соответствии с общей целью улучшения и тем, как эта цель соотносилась с их конкретными задачами. Рассмотрите возможные конфликты интересов между стейкхолдерами и способы их разрешения.
2. Вы также можете проанализировать возможных стейкхолдеров и их цели на примере PharmaBot от компании MediWorld.

3.3. Движение к общей цели

Даже внутри одной команды, отвечающей за улучшения, разные ее участники могут иметь противоречащие друг другу точки зрения относительно того, что нужно исправить в первую очередь. Рассмотрим следующий сценарий.

> *На первой встрече команды PharmaBot специалисты так и не смогли договориться, с чего же им начать. Каждый принес свой список «обязательных исправлений». Выводы одних основывались на догадках, других — на чтении нескольких транскриптов, а третьих — на детальном анализе. Вся команда понимала, что важно учитывать частоту возникновения проблем: появляются ли они в каждом диалоге или же это единичные случаи? Частота проблем стала ключевым фактором приоритизации исправлений и улучшений, тем самым помогая MediWorld повысить эффективность своего чат-бота.*
>
> *Разработчики и аналитики данных выступали за улучшение возможностей PharmaBot по обработке естественного языка; они проанализировали недавние взаимодействия, чтобы выявить области для улучшения понимания сложных и контекстно-зависимых вопросов. Вместе с тем ведущие специалисты колл-центра подчеркивали необходимость в более подробных и вместе с тем дружелюбных ответах на наиболее частые пользовательские запросы, с которыми они столкнулись в своей работе.*

Подход, основанный на данных, помогает правильно расставить приоритеты. Решение наиболее частых проблем позволяет быстро достичь ощутимого улучшения пользовательского опыта. Ранее на рис. 3.2 было продемонстрировано, как разные члены команды могут привнести свои наблюдения и идеи. Каждый участник смог добавить в обсуждение уникальную точку зрения, основанную на его роли и навыках.

3.3.1. Пересмотр бизнес-целей

Члены команды по улучшению разговорного ИИ должны установить единые цели улучшения. Первым важным шагом здесь является достижение консенсуса относительно того, что означает успех: команда должна проанализировать исходные бизнес-цели, которые стали причиной внедрения чат-бота. Например, вопросно-ответный чат-бот должен стабильно и быстро давать точные ответы на пользовательские запросы. Процессно-ориентированный бот должен помогать пользователям эффективно достигать своих целей, таких как, например, запись на прием или проверка состояния счета. Маршрутизирующий агент должен направлять пользователей к специалисту или ресурсу, который может удовлетворить их потребности в зависимости от запроса. Оценка эффективности чат-бота по отношению к этим целям включает такие метрики, как точность ответов, удовлетворенность пользователей и способность бота охватывать широкий спектр релевантных тем и сценариев использования.

По мере изменения бизнес-среды ключевой задачей становится адаптивность. Изменения в ожиданиях пользователей или какие-либо технологические достижения могут потребовать улучшений или корректировок для поддержания оптимальной работы чат-бота. Поэтому команда должна постоянно пересматривать и совершенствовать стратегию чат-бота, обеспечивая ее соответствие главным целям как компании, так и пользователей. Такой итеративный процесс позволяет учитывать развитие потребностей бизнеса и его клиентов.

От бизнес-результатов к метрикам

Определение правильных метрик начинается с понимания того, как бизнес-цели изменяются со временем. Рассмотрим такой сценарий.

> *Команде PharmaBot было необходимо согласовать свою работу с данными о распределении и доступности вакцин. Если изначальной бизнес-целью было отвечать на вопросы и снижать нагрузку на колл-центры, то новые цели заключались в эффективном распределении вакцин и автоматизации процесса записи на прием, что потребовало других метрик.*

В случае каких-либо сомнений всегда ориентируйтесь на пользователей — что им нужно и как эти потребности влияют на бизнес? Чего хотят клиенты? Каковы были исходные бизнес-цели? Как они согласовывались между собой? При этом важно понимать, что со временем ответы на эти вопросы могут меняться. На рис. 3.3 представлена первая бизнес-цель PharmaBot — предоставление точной и актуальной информации о новой пандемии. Ключевой метрикой в данном случае была доля верных результатов, или точность (accuracy).

После построения надежной основы команда усложнила Q&A-бота, сделав его более «умным». Они начали выявлять сущности (контекстные элементы, относящиеся к намерению), чтобы предоставлять пользователям более релевантные ответы, тем самым повышая точность, как показано на рис. 3.4.

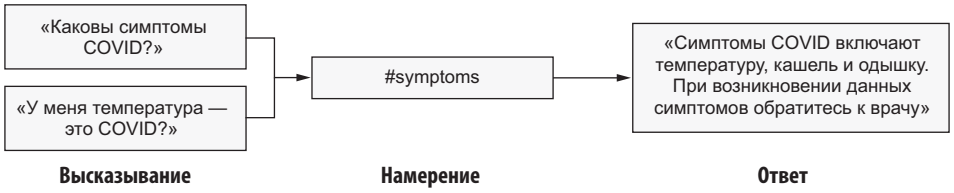


Рис. 3.3. PharmaBot начинался как простой Q&A-бот. Разные варианты вопросов приводили к одному и тому же ответу

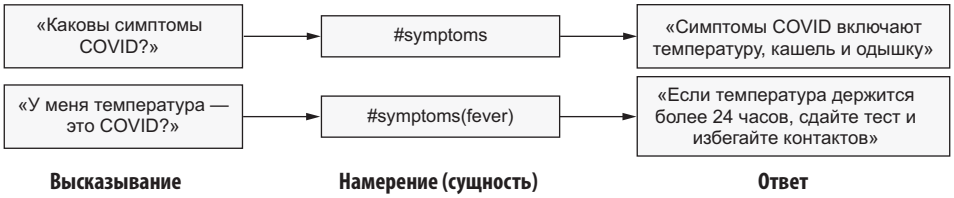


Рис. 3.4. Q&A-бот стал более сложным благодаря распознаванию сущностей в пользовательских высказываниях, что привело к более полным ответам в рамках одного намерения

Затем внешние факторы снова повлияли на бизнес-цели. Доступность вакцин изменила природу бота. Вместо взаимодействия в простом формате вопросов и ответов о вирусе пользователи захотели через бота записываться на вакцинацию. Для этого потребовались процессно-ориентированные сценарии, собирающие необходимую информацию. На рис. 3.5 показано начало такого процесса.



Рис. 3.5. Некоторые типы вопросов не имеют единого статичного ответа, а требуют процессного сценария

Новые возможности добавили боту сложности. Автоматизация тестирования на вирус и записи на вакцинацию потребовала интеграции с системами планирования и базами данных. Это повысило значение мер безопасности и защиты данных. Защита персональной информации пользователей, соблюдение медицинских норм и обеспечение безопасных транзакций стали первостепенными задачами.

Не все решения в области разговорного ИИ проходят через подобную эволюцию, по крайней мере не с такой скоростью, как это потребовалось PharmaBot. Управление развитием чат-бота требует взвешенного анализа затрат и выгод, связанных с каждым аспектом улучшения.

Основные бизнес-цели любого решения в области разговорного ИИ связаны с улучшением общих бизнес-результатов. У каждого бизнеса есть одна или несколько целей, стоящих за системой: они сводятся к двум ключевым факторам — рост доходов и сокращение затрат. Цели трансформируются в метрики, такие как рост коэффициента конверсии, увеличение среднего чека (average order value, AOV) и повышение пожизненной ценности клиента (customer lifetime value, CLV) для роста выручки. С точки зрения затрат важны такие метрики, как сокращение среднего времени обработки запроса (average handling time, AHT), снижение операционных расходов и повышение эффективности решения вопросов при первом обращении (first-contact resolution, FCR). Компании ожидают измеримую окупаемость инвестиций (return on their investment, ROI), и именно эти показатели направляют процесс непрерывного улучшения, соотнося успех разговорного ИИ с ключевыми бизнес-результатами.

Бизнес-цели должны выражаться в измеримых метриках. Это позволяет количественно оценить, насколько хорошо разговорный ИИ достигает своих целей. Примеры в табл. 3.4 демонстрируют, как компании в различных отраслях могут определить измеримые метрики, согласующиеся с их конкретными целями.

Таблица 3.4. Примеры метрик, выведенных из бизнес-целей в различных отраслях

Бизнес-цель	Итоговые метрики	Тип бота
Увеличить онлайн-продажи и сократить расходы на обслуживание клиентов	Процент оформленных заказов, завершенных чат-ботом без участия человека: достижение 75 % автоматизированных оформлений заказов приводит к сокращению обращений в службу поддержки на 100 000 в день и обеспечивает ежедневную экономию в $500 000	Вопросно-ответный
Повысить эффективность клиентской поддержки и минимизировать сбои в обслуживании	Процент запросов, успешно перенаправленных в соответствующий отдел или к специалисту: 90 % запросов, направленных в нужную группу поддержки без необходимости ручного вмешательства, сокращает количество обращений в службу поддержки на 40 000 в день и обеспечивает ежедневную экономию в $700 000	Маршрутизирующий агент
Улучшить процесс бронирования и сократить расходы на поддержку	Процент автоматизированных подтверждений бронирования без участия оператора: достижение 70 % автоматизированных подтверждений снижает количество обращений в службу поддержки на 80 000 в день и обеспечивает ежедневную экономию в $640 000	Процессно-ориентированный
Повысить вовлеченность пациентов и оптимизировать процесс записи на прием	Процент записей на прием, осуществленных виртуальным помощником автономно: 90 % записей на прием, выполняемых без участия человека, сокращает количество записей «вручную» на 30 000 в день и обеспечивает ежедневную экономию в $700 000	Процессно-ориентированный

Метрики, используемые в области разговорного ИИ, должны иметь четкую связь с бизнес-ценностью, чтобы доказать окупаемость инвестиций. Отслеживание таких метрик, как снижение нагрузки на колл-центры и точность маршрутизации, помогает сократить расходы. Измерение удовлетворенности клиентов способствует росту доходов, поскольку довольные пользователи используют больше услуг. Команда PharmaBot достигла одновременно и экономии затрат, и роста доходов благодаря автоматизации записи на прием.

Дополнительные бизнес-драйверы

Помимо согласования с основными бизнес-целями, рассмотренными в предыдущем разделе, компаниям следует учитывать дополнительные факторы, повышающие ценность разговорного ИИ. Успешное внедрение ИИ делает больше, чем просто поддерживает стратегические цели, — оно активно повышает вовлеченность клиентов, оптимизирует стратегии продаж и снижает операционные затраты.

Разговорный ИИ может упрочить взаимодействие с клиентами, управлять процессом продаж и предлагать релевантные продукты. Анализ коэффициентов конверсии, обеспечиваемых чат-ботами, важен для корректировки стратегий. Компании, использующие возможности ИИ для апселлинга и повышения кросс-продаж, способны максимально реализовать весь свой потенциал роста доходов. С операционной точки зрения разговорный ИИ помогает выполнять рутинные задачи и освобождает специалистов для более сложных действий. Автоматизация процессов повышает эффективность, снижает расходы на поддержку и улучшает удовлетворенность клиентов благодаря более быстрым ответам.

Тщательный анализ поможет выявить возможности для оптимизации работы чат-ботов. По мере развития технологий компании должны расширять функциональность своих систем для дальнейшего сокращения затрат и достижения операционного совершенства.

Анализ конкурентов, включающий оценку таких характеристик, как понимание естественного языка и персонализированный опыт, также может помочь в процессе непрерывного улучшения. Регулярные обновления позволяют адаптироваться к изменениям конкурентной среды, стимулируя инновации и обеспечивая положительные бизнес-результаты.

3.3.2. Эффективность

При определении приоритетности тех или иных улучшений еще одним ключевым фактором является эффективность чат-бота. Выполняет ли он задачи, для которых был создан? И хотя концепция «эффективности» проста (работает ли все так, как ожидается?), она все же выходит за рамки простой успешности выполнения задач и включает в себя обеспечение позитивного и эффективного взаимодействия для пользователей.

Продолжим наш сценарий. Теперь команда MediWorld анализирует дашборд с метриками своего бота. Большинство платформ разработки чат-ботов имеют довольно понятный дашборд со статистикой ключевых показателей (KPI), отображающей, как пользователи взаимодействуют с системой. Обычно подобные дашборды содержат данные о количестве диалогов в день и их средней продолжительности, а также об уровне уверенности чат-бота. Они также могут включать наиболее частые вопросы или намерения.

На рис. 3.6 показан дашборд аналитики PharmaBot, который хоть и содержит некоторые KPI, но не отражает эффективность бота. Общее количество диалогов помогает понять нагрузку, но не позволяет оценить, сколько пользователей успешно завершили взаимодействие или насколько далеко они продвинулись.

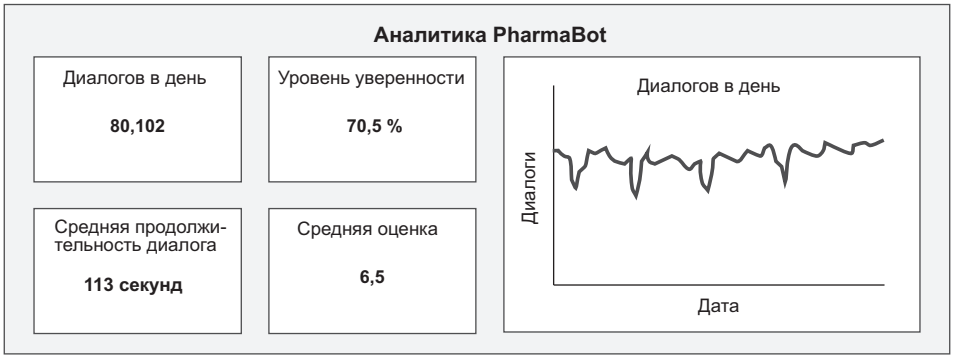


Рис. 3.6. Базовый дашборд для PharmaBot показывает статистику использования, но не дает понимания того, что нравится (или не нравится) пользователям в боте

> *Команда хотела определить текущий уровень успешности бота. Базовая аналитика предоставляла информацию о количестве обращений в день, но было непонятно, сколько из них завершились успешно. Необходимо было выйти за рамки простой количественной оценки. Все знали, что спрос со стороны пользователей растет. Удовлетворяет ли его PharmaBot? Нужно было понять, как измерить эффективность бота и найти пути оптимизации его работы. Команда обнаружила, что 45 % всех диалогов перенаправлялись в колл-центр. Данная метрика «удержания» (containment) влияла на издержки бизнеса. Что можно было с ней сделать?*

В случае с командой PharmaBot одним из ключевых показателей эффективности стала метрика удержания. *Удержанные диалоги* — это те, где чат-бот смог полностью обработать запрос пользователя самостоятельно; *неудержанные диалоги* требовали вмешательства человека. *Коэффициент удержания* (containment rate) рассчитывается как количество удержанных диалогов, деленное на их общее количество. Данная метрика позволяет более точно оценить работу чат-бота, как показано на рис. 3.7.

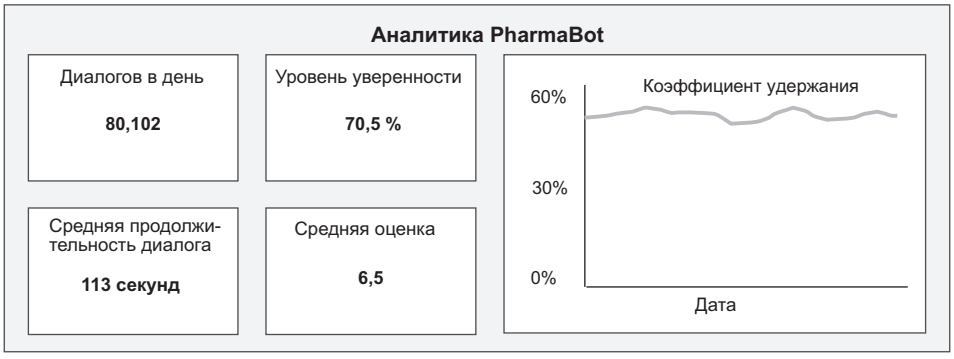


Рис. 3.7. Ежедневный базовый дашборд с простой бизнес-метрикой: удержание. Метрика вычисляется ежедневно, но все еще не дает полного понимания работы бота

Дашборд для PharmaBot предполагает довольно простое определение успеха: «удержанные вызовы — это успешные вызовы». Данный принцип отражен на рис. 3.8. Но несмотря на ценность такой метрики, она не объясняет, почему пользователи добиваются (или не добиваются) успеха при взаимодействии с ботом. Для получения необходимой информации требуется более детализированный анализ.

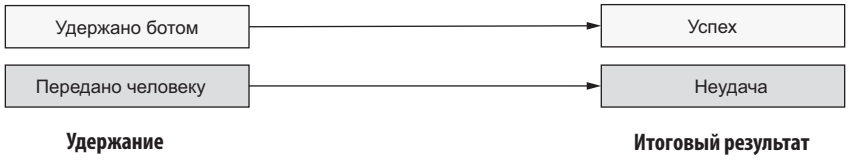


Рис. 3.8. Простейшее определение результата. Не дает понимания того, как можно улучшить работу бота

Результаты диалогов

Чтобы лучше понять, как результаты диалогов влияют на процесс улучшения чат-бота, вернемся к нашему сценарию. Команде поддержки PharmaBot необходимо было выйти за рамки высокоуровневых метрик эффективности и проанализировать реальные результаты диалогов. Метрика удержания сама по себе не давала полной картины — она лишь показывала, ушел ли пользователь из бота или нет. Но как диалоги протекали на самом деле? Вот как команда подошла к их анализу:

> *Чтобы понять, как завершались диалоги, команда стала просматривать их транскрипты. Чем они заканчивались? Обнаружились несколько вариантов — успешные завершения, прерванные диалоги и значительное количество переводов на оператора.*

Аналитик данных отметил различные варианты успешности. Во-первых, это случаи, когда PharmaBot корректно отвечает на запрос и пользователь остается доволен. Во-вторых, передача диалога в колл-центр, предусмотренная бизнес-правилами, например, когда пользователь живет в штате, где MediWorld не может отправлять SMS-подтверждения. Такие переводы тоже считались успешными, так как соответствовали целям PharmaBot: собрать необходимую информацию вместо оператора. Команда договорилась документировать эти два случая отдельно, обозначив их как «автоматизированное решение» и «предусмотренный перевод на оператора».

Далее были выявлены сценарии неудач. Пользователи просили переключить их на оператора после того, как бот неправильно их понял. Бот также автоматически переводил пользователей после нескольких неправильных интерпретаций подряд. Кроме того, некоторые пользователи отключались на середине диалога.

Команда PharmaBot также обнаружила диалоги, в которых пользователи даже не пытались взаимодействовать с ботом. Кто-то отключался сразу после приветственного сообщения, а кто-то сразу просил перевести на оператора. Один из членов команды заметил: «Возможно, здесь играет роль психологический фактор — некоторые пользователи просто не хотят общаться с ботом. Давайте выделим такие диалоги отдельно».

В итоге команда разработала более детализированную систему категоризации: успех (автоматизированное решение и предусмотренный перевод), неудача (прерывание и эскалация, то есть перевод на оператора), бот не нужен (мгновенное отключение и мгновенная эскалация).

После внедрения системы категоризации было подсчитано число диалогов в каждой категории, и команда PharmaBot получила представление о реальной эффективности своего бота.

Определение результатов диалогов в привязке к метрикам позволяет сделать вывод об эффективности вашего решения. Результаты диалогов описывают, чем заканчивается взаимодействие пользователя с чат-ботом, классифицируя, был ли запрос решен ботом самостоятельно, потребовалась ли помощь человека или диалог остался незавершенным. Это важно для оценки и совершенствования решения в контексте бизнес-целей. После внедрения решения проанализируйте транскрипты диалогов и классифицируйте их в соответствии с моделью результатов.

Рисунок 3.9 наглядно демонстрирует модель анализа результатов. Она начинается с удержания (слева), а затем классифицирует удержанные (и неудержанные) диалоги по конкретным причинам, которые в итоге распределяются по двум категориям — успех или неудача. Причины помогают понять, что бот сделал правильно или неправильно. Например, диалоги могут не удерживаться из-за того, что пользователь отказался от взаимодействия или бот несколько раз подряд

его не понял. Некоторые диалоги преднамеренно (следуя бизнес-процессам) переводятся на операторов — такие случаи не считаются неудачами.



Рис. 3.9. Анализ причин, по которым диалоги не удерживаются ботом, дает большее понимание его работы и показывает, где необходимы улучшения. Один из способов это сделать — использовать более подробную классификацию результатов, фиксирующую конкретные итоги взаимодействий с чат-ботом. Подобные классификации группируют диалоги на основе их результатов, пользовательского опыта и последующих шагов

Теперь, когда мы ввели модель классификации результатов, учитывающую, как (где и почему) завершился диалог, давайте посмотрим на тот же дашборд метрик, что и на рис. 3.7. Вместо того чтобы просто фиксировать коэффициент удержания в 45 %, мы теперь фокусируемся на содержании диалогов. На рис. 3.10 мы заменили диаграмму удержания на диаграмму коэффициента успешности (success rate), отражающую категории «успех»/«неудача»/«бот не нужен».

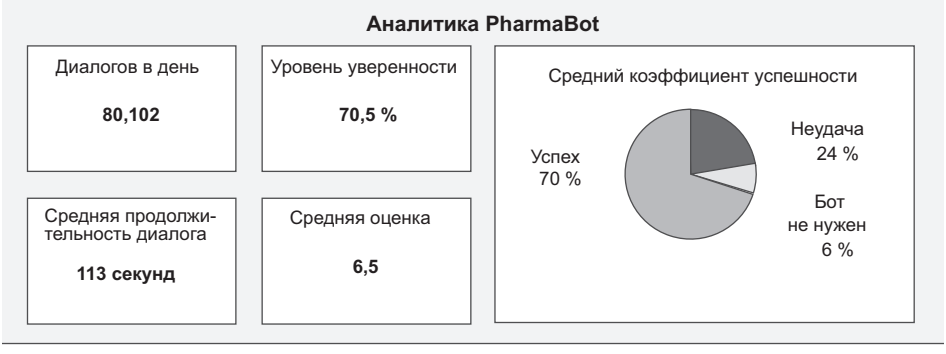


Рис. 3.10. Дашборд стал точнее за счет добавления коэффициента успешности. Не все удержанные звонки являются успешными, как и не все переведенные звонки являются неудачами

Вы также можете представить результаты анализа более детально. На рис. 3.11 показан пример такой диаграммы. Подобная презентация позволит не только подробнее рассмотреть результаты подсчета, но и убедить стейкхолдеров поддержать ваши предложения.

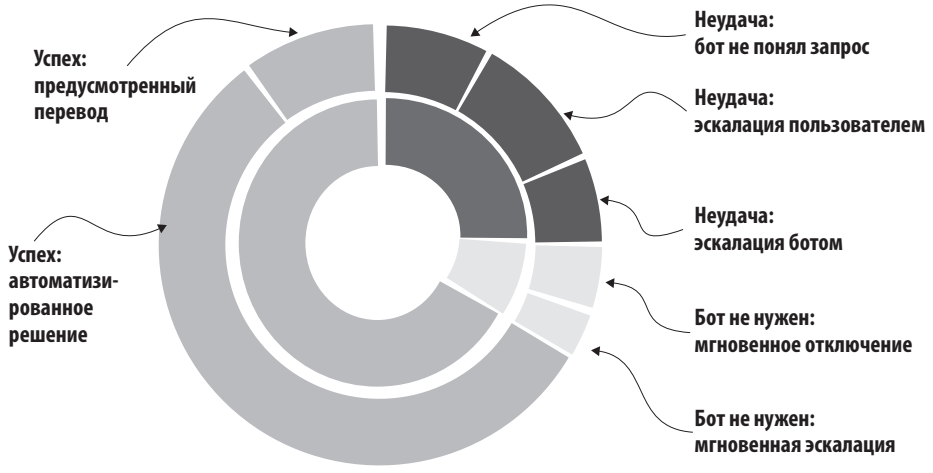


Рис. 3.11. Детализированная модель результатов отображает итоги диалогов за определенный период времени. Это дает гораздо более глубокое понимание работы бота по сравнению с бинарной моделью «удержано/не удержано»

Преимущество более детализированной модели заключается в ее гибкости. Каждый проект в области разговорного ИИ может иметь собственную уникальную классификацию результатов. Диаграмма на рис. 3.11 — это лишь пример реализации, и вы всегда можете адаптировать ее под свои нужды. Например, если ваш чат-бот не предполагает передачи диалогов человеку, можно исключить исходы с эскалацией и переводом.

Ниже приведены возможные классификации для наиболее распространенных типов решений на базе разговорного ИИ.

Для Q&A-ботов:

- Успех — сценарии завершения диалога:
  - Автоматизированное решение — бот успешно отвечает на вопрос пользователя и предоставляет релевантную информацию без участия человека.
- Неудача — взаимодействие не достигает желаемого результата:
  - Прерывание — пользователь покидает диалог до получения релевантного ответа, возможно, из-за недовольства ответами бота.
  - Эскалация — Q&A-бот не понимает пользователя, и диалог переводится на оператора либо по запросу пользователя, либо автоматически после нескольких неудач подряд.

- Бот не нужен:
  - Мгновенное отключение — пользователь выходит из диалога, не отправив ни одного сообщения.
  - Мгновенная эскалация — первое сообщение пользователя боту — это запрос перевести на оператора.

Для транзакционных или процессно-ориентированных ботов:

- Успех — сценарии завершения диалога:
  - Автоматизированное решение — процессно-ориентированный бот успешно выполняет задачу пользователя, например записывает на прием, без участия человека.
  - Предусмотренный перевод — в соответствии с определенными бизнес-правилами бот передает диалог оператору, даже если ошибок не возникло.
- Неудача — взаимодействие не достигает желаемого результата:
  - Прерывание — пользователь покидает диалог на середине сценария, возможно, из-за сложности или неудобства интерфейса бота.
  - Эскалация — бот начинает, но не может завершить сценарий из-за того, что он не понял пользователя либо пользователь захотел переключиться на оператора.
- Бот не нужен:
  - Мгновенное отключение — пользователь выходит из диалога, не отправив ни одного сообщения.
  - Мгновенная эскалация — первое сообщение пользователя боту — это запрос перевести на оператора.

Для маршрутизирующих агентов:

- Успех — сценарии завершения диалога:
  - Предусмотренный перевод — маршрутизирующий агент успешно направляет пользователя в нужный отдел или к специалисту, передавая собранную информацию. В результате агент может иметь 0 % удержания и при этом работать очень эффективно!
- Неудача — взаимодействие не достигает желаемого результата:
  - Прерывание — пользователь покидает диалог еще до перевода на нужного специалиста.
  - Эскалация — маршрутизирующий агент не может собрать достаточно информации для перевода на специалиста из-за того, что он не понял пользователя либо пользователь сам захотел переключиться на оператора.
- Бот не нужен:
  - Мгновенное отключение — пользователь выходит из диалога, не отправив ни одного сообщения. В таком случае маршрутизация невозможна.

- Мгновенная эскалация — первое сообщение пользователя боту — это запрос перевести на оператора, минуя автоматизированную маршрутизацию.

При такой классификации диалогов количество взаимодействий в каждой категории позволяет оценить эффективность реализации чат-бота и выявить возможности для улучшения.

Детализированная модель результатов должна учитываться при проектировании диалогов, например в ходе определения ключевых этапов для каждого из «идеальных» сценариев взаимодействия с ботом. На рис. 3.12 показан пример подобной реализации для PharmaBot. Этап «Записать на прием» на схеме обозначает ключевой шаг, при котором бот успешно завершает сценарий. Отдельно видим этап «Чем еще могу помочь?», что указывает на готовность бота продолжать взаимодействие после выполнения основной задачи.

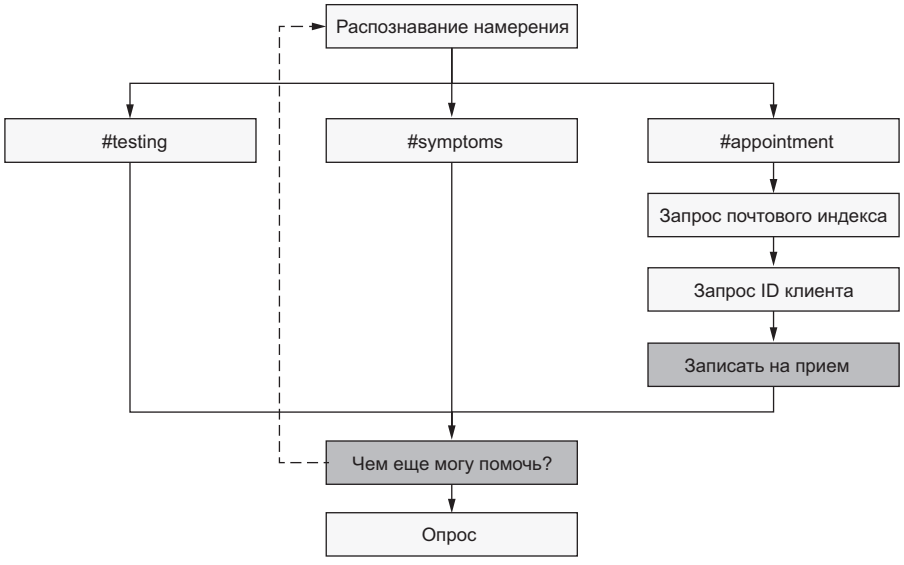


Рис. 3.12. Общая схема сценария для PharmaBot с указанием наиболее значимых этапов диалога. Достижение этапов «Записать на прием» и «Чем еще могу помочь?» считается успехом

Намерения для FAQ имеют только один этап (ответ на FAQ), тогда как процесс записи на прием предполагает сбор необходимых для этого данных. Следует заранее определить этапы, достижение которых означает успешное завершение диалога.

Анализ соответствия результатов необходимым для успеха этапам диалога позволяет сделать выводы об эффективности бота. На рис. 3.13 представлена динамика метрик PharmaBot по неудачным диалогам, включая дату последнего обновления.

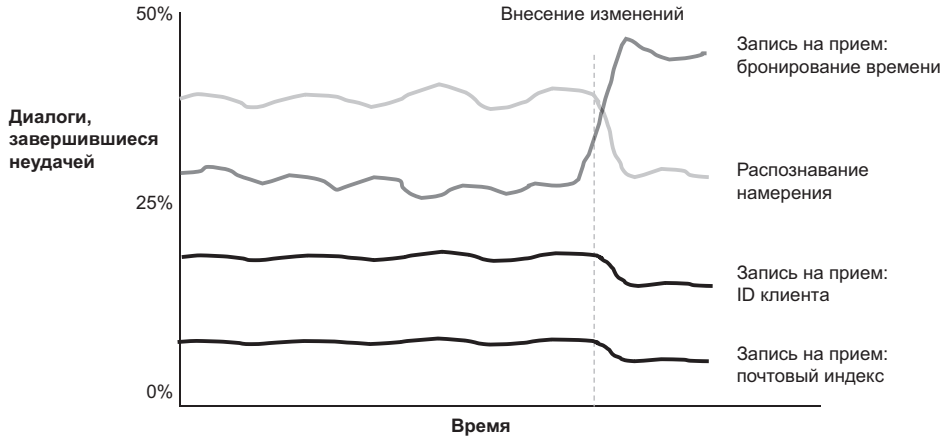


Рис. 3.13. При совместном анализе результатов и дизайна диалога можно сделать ряд важных выводов. Диаграмма показывает распределение диалогов, закончившихся неудачей, по последнему шагу перед сбоем, позволяя определить, где именно пользователи испытывают наибольшие трудности. Мы можем оценить влияние каждого этапа: запись на прием, распознавание намерения, ввод почтового индекса — на общее число сбоев. Всплеск на графике после пунктирной линии «Внесение изменений» показывает влияние последних обновлений, указывая на необходимость дальнейшей оптимизации

Команда PharmaBot может дополнительно проанализировать диалоги с неудачей, чтобы рассмотреть их исходы более детально — прерывание или эскалация. Совмещение моделей результатов и дизайна диалога позволяет быстрее сделать выводы о том, с чего лучше начать оптимизацию.

Модель результатов, представленная на рис. 3.11, обеспечивает более глубокое понимание пользовательского опыта, раскрывая, где и почему в диалогах происходят сбои. В данном случае результат — это неудача, а основной причиной выступает эскалация со стороны пользователя. Однако анализ конкретных моментов, в которых пользователи инициировали эскалацию, — последнего шага перед неудачей — дает практическую информацию о работе бота. Диаграмма показывает ключевые точки эскалации: в процессе записи на прием, при распознавании намерения, при вводе ID клиента и почтового индекса. После обновления количество неудач на этапе распознавания намерения значительно снизилось, тогда как неудачи на этапах ввода ID и почтового индекса сократились незначительно. Число неудач при записи на прием (бронирование времени) резко возросло, что указывает на источник проблем. Такой анализ позволяет команде эффективно расставлять приоритеты для улучшений, направляя основные усилия на наиболее критичные для пользователей проблемы. И хотя показатель удержания зачастую используется как индикатор эффективности, он не отражает всей картины: некоторые удержания могут приводить к неудовлетворительному пользовательскому опыту, тогда как

некоторые переводы на оператора необходимы для успешного завершения диалога. Благодаря модели результатов, представленной на рис. 3.13, команда по разработке разговорного ИИ может различать подобные случаи и корректировать работу бота.

Удовлетворенность пользователей

На основе результатов диалогов мы можем сделать вывод о степени удовлетворенности пользователей. Это довольно быстрый метод, но в то же время он косвенный и может упускать некоторые детали, поэтому полезно время от времени измерять удовлетворенность напрямую.

Удовлетворенность пользователей можно оценивать с помощью обратной связи. Стандартные методы — это кнопки «палец вверх/вниз» или числовая оценка удовлетворенности. Недостаток таких метрик в том, что уровень откликов часто низкий — многие пользователи не любят оставлять отзывы, тогда как недовольные клиенты чаще используют данную возможность.

Можно ввести опрос сразу после завершения взаимодействия с чат-ботом. Вопросы могут касаться удобства использования, полезности и общей удовлетворенности ботом. Пользователям также можно предложить опрос индекса потребительской лояльности (NPS). Здесь важно лишь одно — чтобы вопросов было немного: чем длиннее опрос, тем меньше вероятность, что пользователи его завершат.

Оценивать удовлетворенность пользователей также можно путем анализа выборки диалогов, используя логи чатов или краткие резюме от операторов, завершивших разговор. Данные из этих источников можно даже классифицировать с помощью больших языковых моделей. В табл. 3.5 представлены виды обратной связи, которые можно ожидать при различных результатах диалога.

Таблица 3.5. Зависимость обратной связи от результата диалога

Результат диалога	Обратная связь от пользователей	Примечания
Автоматизированное решение (успех)	Положительная обратная связь или ее отсутствие	Пользователи могут выразить благодарность («спасибо!»), но редко: получив необходимое, большинство сразу завершают диалог
Перевод на оператора (успех)	Положительная устная обратная связь или ее отсутствие. Не фиксируется в чат-логах	Пользователи могут высказать свое мнение о работе бота оператору, но не самому чат-боту
Прерывание (неудача)	Негативная обратная связь или ее отсутствие (отключение до завершения процесса)	Последнее сообщение пользователя может содержать отрицательную оценку («Какой кошмар!»). Однако большинство предпочитает не выражать недовольство и просто отключается

Результат диалога	Обратная связь от пользователей	Примечания
Эскалация пользователем (неудача)	Негативная обратная связь	Пользователи, запрашивающие оператора на середине сценария, недовольны работой бота («Переключи на оператора!»)
Эскалация ботом (неудача)	Отсутствие обратной связи	Если эскалация была инициирована ботом, невозможно точно знать мнение пользователя, но можно обоснованно предположить, что он был недоволен
Мгновенное отключение (бот не нужен)	Отсутствие обратной связи (нет взаимодействия)	Пользователи, сразу прервавшие диалог, могут в принципе не любить чат-боты, могут не любить именно ваш чат-бот или просто подключились случайно. Точную причину узнать невозможно
Мгновенная эскалация (бот не нужен)	Отсутствие обратной связи или устный негативный отзыв, выражающий предпочтение общаться с человеком вместо чат-бота	Пользователи, сразу запросившие оператора, могут в принципе не любить чат-боты, могут не любить именно ваш чат-бот или просто хотят говорить с человеком. Точную причину узнать невозможно

3.3.3. Охват

Для повышения эффективности вашего чат-бота определить, что бот способен обрабатывать, а что нет (или, другими словами, его *охват*), столь же важно, как и работать с эскалациями. Так, например, в ходе анализа было выявлено, что некоторые пользовательские вопросы не были неверно интерпретированы — они просто не входили в сферу знаний, доступных боту, поэтому теперь команда собирается выявить эти пробелы и расширить возможности PharmaBot:

> *В первую очередь команда решила сократить число эскалаций, поскольку именно они существенно влияли на метрики. Разработчики проанализировали транскрипты эскалированных диалогов и заметили некоторые закономерности. Пользователи часто переходили к оператору сразу после того, как бот не понимал их запрос. Аналитик данных сопоставил эти случаи с логикой диалога и предположил, как можно улучшить понимание ботом естественного языка. В ходе анализа были выявлены несколько вопросов, на которые PharmaBot не мог ответить:*
>
> - *«Я слышал о редких побочных эффектах вакцины. Как отличить ожидаемые симптомы от чего-то более серьезного, что требует медицинской помощи?»*
> - *«Если я на несколько дней опоздал с введением второй дозы вакцины от COVID-19, будет ли она работать или нужно прививаться заново?»*

- *«Я беременна и не уверена, стоит ли делать прививку от COVID-19. Можете предоставить информацию о безопасности и потенциальной пользе вакцинации для беременных?»*
- *«У меня диагностировано аутоиммунное заболевание. Могу ли я сделать прививку от COVID-19 и есть ли какие-то дополнительные меры предосторожности?»*
- *«Я прочитал противоречивую информацию о последствиях вакцин от COVID-19. Что известно об их безопасности в долгосрочной перспективе и проводятся ли соответствующие исследования?»*

На подобные вопросы PharmaBot отвечал «Извините, я вас не понимаю», даже после того, как пользователи пытались переформулировать запрос. Команда попыталась найти группы вопросов с похожими характеристиками, чтобы определить, чему нужно обучить бота в первую очередь. Так вместе с рядом других намерений была сформирована группа запросов, связанных с безопасностью вакцин.

Охват измеряет, сколько пользовательских вопросов чат-бот способен эффективно обработать. Бот с низким охватом либо не имеет обучающих данных по необходимым темам, либо испытывает трудности с пересекающимися и неоднозначными намерениями, когда схожие вопросы сбивают модель с толку и мешают ей уверенно выбрать корректный ответ.

Для повышения охвата команда должна проанализировать взаимодействия пользователей и выявить моменты, когда чат-бот не смог предоставить релевантный ответ. Данный процесс включает в себя анализ транскриптов, отслеживание неотвеченных вопросов и выявление закономерностей между ними. Устранение пробелов в знаниях бота может потребовать доработки обучающих данных, изменений в классификации намерений или внедрения альтернативных подходов, например генерации, дополненной поиском (RAG).

Существует несколько способов повышения охвата чат-бота, каждый из которых решает разные задачи:

- В главе 5 рассматривается улучшение понимания запросов за счет доработки обучающих данных, ввода отсутствующих намерений и оптимизации разметки.
- В главе 6 обсуждается, как обойти ограничения намерений, используя методы, основанные на поиске, в сочетании с генеративным ИИ (RAG), чтобы обеспечивать более динамичные и релевантные ответы.
- В главе 7 показано, как генеративный ИИ может создавать дополнительные обучающие и тестовые данные для расширения способности чат-бота работать с самыми разными запросами.

Независимо от выбранного подхода, повышение охвата чат-бота требует постоянного анализа реальных взаимодействий. Изучение пользовательских высказываний, выявление частых точек отказа и итеративное улучшение дизайна

бота гарантируют, что он будет развиваться и все лучше соответствовать потребностям пользователей.

Задания

1. Рассмотрите следующих чат-ботов и их бизнес-цели. Разработайте подходящие метрики и цели для каждого типа бота, чтобы оценить их эффективность и результативность:
   - Бот 1 — бот поддержки клиентов для онлайн-магазина (Q&A-бот).
   - Бот 2 — банковский бот для управления счетами (транзакционный или процессно-ориентированный).
   - Бот 3 — бот клиентской службы для телеком-компании (агент-маршрутизатор).
2. Проанализируйте следующие сценарии из чат-логов PharmaBot. На основе ответов PharmaBot на каждый сценарий определите результаты диалогов. После того как вы присвоите каждому сценарию свой результат, отметьте любые закономерности или тенденции в эффективности бота и предложите возможные улучшения. Напоминаем, что результаты делятся на успех, неудачу и отказ от использования чат-бота. Данные группы подразделяются на: автоматизированное решение (успех), предусмотренный перевод (успех), прерывание (неудача), эскалация (неудача), мгновенное отключение (чат-бот не нужен), мгновенная эскалация (чат-бот не нужен):
   - Сценарий 1

     ```
     Пользователь: "Я слышал о редких побочных эффектах вакцины. Как
         отличить ожидаемые симптомы от чего-то более серьезного, что
         требует медицинской помощи?"
     PharmaBot: "Здравствуйте! Важно быть информированным о возможных
         побочных эффектах вакцины. К распространенным относятся
         болезненность в месте инъекции, утомляемость и небольшая
         температура. Однако при серьезных симптомах, таких как
         затрудненное дыхание или боль в груди, необходимо немедленно
         обратиться к врачу".
     Пользователь: "Спасибо за информацию, но что, если я не уверен,
         насколько серьезны мои симптомы, чтобы обращаться к врачу?"
     PharmaBot: "Если вы сомневаетесь, лучше перестраховаться
         и проконсультироваться со специалистом. Он сможет оценить ваши
         симптомы и дать рекомендации, соответствующие вашей ситуации".
     PharmaBot: "Хотите, я соединю вас с врачом для дальнейшего обсуждения
         ваших опасений?"
     Пользователь: "Да, пожалуйста. Было бы здорово".
     PharmaBot: "Отлично! Подождите немного, я переведу вас на оператора,
         который сможет предоставить персонализированную помощь. Ваше
         здоровье — наш приоритет".
     PharmaBot: (Инициирует перевод на оператора.)
     ```

   - Сценарий 2

     ```
     Пользователь: "Если я на несколько дней опоздал с введением второй
         дозы вакцины от COVID-19, будет ли она работать или нужно
         прививаться заново?"
     ```

```
PharmaBot: "Здравствуйте! Для наилучшей защиты важно соблюдать
    рекомендованный график вакцинации. Если вы опоздали с введением
    второй дозы на несколько дней, лучше связаться с врачом для
    получения рекомендаций по дальнейшим действиям".
Пользователь: "Понятно. А задержка второй дозы повлияет на
    эффективность вакцины?"
PharmaBot: "Задержка может сказаться на эффективности вакцины,
    при отставании от графика необходимо как можно скорее сделать
    вторую прививку. Ваш лечащий врач сможет оценить ситуацию и дать
    индивидуальные рекомендации".
Пользователь: "Понял, свяжусь с врачом. Спасибо за помощь!"
PharmaBot: "Пожалуйста! Всегда лучше проконсультироваться с врачом
    для получения индивидуальных рекомендаций. Если будут еще
    вопросы, задавайте".
```

- Сценарий 3

```
Пользователь: "Здравствуйте. У меня есть вопрос о вакцине от
    COVID-19".
PharmaBot: "Здравствуйте! Готов вам помочь. Что бы вы хотели узнать
    о вакцине от COVID-19?"
Пользователь: "Оператор".
PharmaBot: "Конечно! Соединяю вас с оператором для дальнейшей
    консультации".
PharmaBot: (Инициирует перевод на оператора.)
```

- Сценарий 4

```
Пользователь: "Расскажи о составе вакцины от COVID-19."
PharmaBot: "Извините, я вас не понимаю. Можете переформулировать
    вопрос?"
Пользователь: "Какие компоненты входят в состав вакцины от COVID-19?"
PharmaBot: "Извините, я все еще не совсем вас понял. Попробуйте
    задать вопрос по-другому".
Пользователь: "Какой состав у вакцины от COVID-19?"
PharmaBot: "Мне трудно понять ваш вопрос. Можете уточнить его или
    переформулировать более простыми словами?"
Пользователь: (Завершает диалог.)
```

3.4. Выявление и устранение проблем

Умение правильно выявить проблему имеет решающее значение для непрерывного улучшения разговорного ИИ. Команде, занимающейся разработкой чатбота, необходимо выстроить алгоритм работы с проблемами, включая то, как их находить, коллективно анализировать и выбирать подходящее решение. Четко определенный процесс позволит команде эффективно двигаться к общей цели.

3.4.1. Поиск проблем

Лучший способ начать поиск проблем — это попытаться найти какие-либо закономерности в результатах взаимодействий. Успешные диалоги всегда приятно

анализировать, но лучше больше внимания уделить неудачным взаимодействиям, а также случаям «бот не нужен». Необходимо детально рассмотреть те результаты, доля которых растет, и сопоставить их с архитектурой диалогов, как это было показано в предыдущем разделе. Что сделал пользователь перед отрицательным результатом? В этом вам поможет дашборд PharmaBot (рис. 3.14).

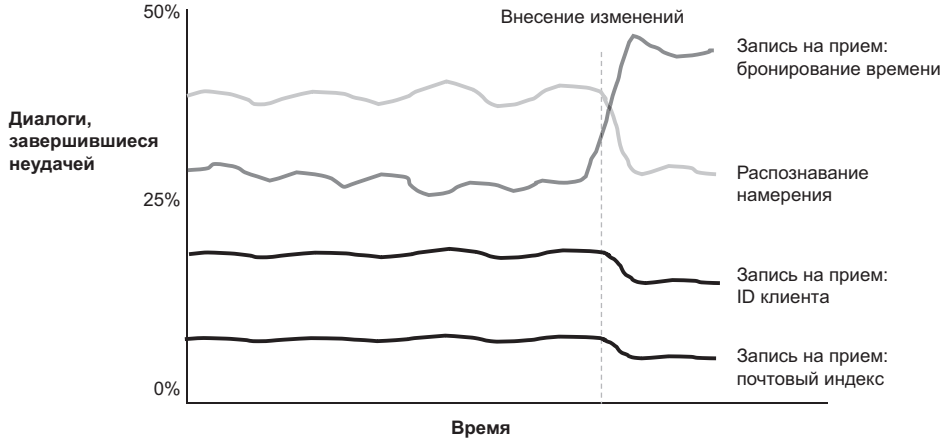


Рис. 3.14. Дашборд, отображающий результаты диалога в зависимости от последнего действия пользователя, а также их динамику во времени

Идеальный инструмент анализа должен уметь подсчитывать и отображать диалоги по следующим критериям:

- итог взаимодействия;
- последний шаг в диалоге;
- дата и/или время взаимодействия.

Некоторые платформы разговорного ИИ уже отслеживают эти данные за вас. Другие же платформы могут потребовать дополнительной настройки, например, введения контекстных переменных в ключевые части диалога. Такие метрики позволят команде выявить неожиданные паттерны в поведении пользователей, влияющие на эффективность чат-бота. Так, при анализе эскалированных диалогов команда PharmaBot обнаружила следующую закономерность:

> *Команда PharmaBot детально изучила эскалированные диалоги. Было выявлено множество неудач в ситуациях, когда бот спрашивал, идет ли речь о вакцинации или тестировании. Многие пользователи отвечали «да» — реакция, не соответствующая ожидаемому формату.*
>
> *Понимая важность анализа поведения пользователей, команда пришла к выводу, что неспособность бота корректно интерпретировать это неоднозначное «да» вызывала недовольство. Некоторые пользователи снова повторяли «да»,*

когда PharmaBot задавал тот же самый вопрос повторно. Это приводило как к прерыванию диалогов, так и к их эскалации.

Когда область проблемы определена, можно переходить к разработке решения. На этот процесс влияет то, как часто встречается проблема и как она может быть решена. В ситуации с «неожиданным “да”» есть два варианта: научить бота правильно обрабатывать ответ «да» или добиться того, чтобы пользователи перестали так отвечать.

Рассмотрим еще несколько способов поиска проблем.

Качественный анализ проблемы

Хотя метрики и чат-логи дают ценные сведения о работе ботов, некоторые проблемы сложно выявить исключительно количественным анализом. Использование в процессе принятия решений пользовательской обратной связи (качественный анализ) позволяет выявить больше болевых точек клиентов. Рассмотрим, как команда PharmaBot организовала свои опросы.

Чтобы лучше понять причины недовольства пользователей, команда PharmaBot решила собрать от них обратную связь и проанализировать ее. Пользователям было предложено подробно описать свои затруднения и ожидания от бота в анкете. После сбора данных команда систематизировала и выделила основные болевые точки, представленные в табл. 3.6.

Таблица 3.6. Ответы пользователей на опрос, позволившие выявить ряд проблем (часть 1)

Пользователь	Результат опроса	Выявленная проблема
1	Я пытался узнать о возможности вакцинации в разных штатах, но бот не смог дать четкую информацию. Я так и не понял, имею ли я право на вакцинацию, пока нахожусь в поездке. Ответы были слишком общими и не учитывали данные из разных регионов	PharmaBot понял основное намерение пользователя (право на вакцинацию), но не смог дать информацию по конкретным штатам. Чат-бот не учитывал местоположение пользователя и регион запроса и выдавал бесполезные ответы
2	Я пытался записаться на вакцинацию, но весь процесс оказался слишком запутанным. Инструкции бота были неясны, и я не смог понять, получилось у меня записаться или нет. Было бы полезно, если бы бот давал больше подсказок на протяжении всего процесса	Процесс записи был недостаточно понятен, из-за чего пользователи сомневались, удалось ли им записаться
3	(Ответа не поступило, пользователь покинул чат)	Слишком сложный сценарий отпугнул пользователей от завершения процесса. Пользователь 3 не дошел до опроса

В отличие от чат-логов, качественная обратная связь дает прямое понимание проблем пользователей — не нужно будет строить догадки, что пошло не так. В сочетании с транскриптами диалогов такие данные позволяют команде точнее обнаруживать недостатки чат-бота и устранять их.

Получение обратной связи от пользователей может существенно повысить эффективность чат-бота и удовлетворенность аудитории. Однако большинство пользователей крайне редко оставляют отзывы, поэтому небольшая мотивация или даже простое «спасибо» уже могут повысить вовлеченность. Если обратная связь является ключевым элементом вашей стратегии, стоит внедрить систему, создающую взаимовыгодные условия как для пользователей, так и для команды.

ПРЕДУПРЕЖДЕНИЕ Полученные отзывы дают ценную информацию о конкретных проблемах пользователей, но могут не иметь статистической значимости. Не стоит спешить и внедрять какие-либо решения лишь на основе единичных случаев. Вместо этого лучше искать повторяющиеся появления проблем, чтобы правильно оценить их масштаб и влияние.

Количественная оценка для выявления проблем

Анализируя мнение пользователей, можно выявить не только причины их недовольства, но и измеримые функциональные проблемы. Такие недочеты, как, например, медленный отклик или запутанные диалоги, можно количественно оценить с помощью чат-логов. Это помогает командам находить проблемы и определять приоритетность тех или иных улучшений. Продолжим наш пример с компанией MediWorld и рассмотрим другие проблемы:

Помимо проблем с пониманием запросов пользователей, команда PharmaBot выявила функциональные недостатки, которые можно устранить. Один из примеров приведен в табл. 3.7.

Таблица 3.7. Ответы пользователей на опрос, позволившие выявить ряд проблем (часть 2)

Пользователь	Результат опроса	Выявленная проблема
4	Боту потребовалось почти 5 минут, чтобы сообщить о наличии свободных мест на вакцинацию. Такая задержка крайне неудобна, особенно когда нужно заранее спланировать расписание. Хотелось бы получить быструю реакцию, ведь именно для этого я и обратился к боту — чтобы не ждать на линии!	Медленный отклик раздражает пользователей и снижает ценность чат-бота как более быстрой альтернативы по сравнению с традиционной службой поддержки

Проблемы такого рода можно выявлять путем анализа времени, затраченного на каждый шаг в чат-логе. Можно отслеживать среднее и максимальное время выполнения шагов, а аномальные значения будут указывать на плохо работающие бэкенд-системы или на нечетко сформулированные вопросы, над которыми пользователи долго размышляют.

Такой анализ можно проводить даже на подмножестве диалогов. Например, медленно отвечающий API с большей вероятностью приведет к тому, что пользователи прервут диалог. Необходимо анализировать такие взаимодействия и выявлять, что именно писали пользователи и сколько времени занимал каждый шаг. Подобный анализ можно проводить и без обратной связи от клиентов.

Определяя конкретные трудности, связанные со сценариями диалогов, аналитики смогут улучшить работу разговорной системы.

3.4.2. Групповой обзор

После анализа метрик результатов диалогов и отзывов пользователей команда PharmaBot составила список конкретных проблем. Теперь необходимо построить план улучшений, начиная с расстановки приоритетов.

Классификация проблем

С помощью метрик команды могут выявлять неожиданные поведенческие модели пользователей, влияющие на работу чат-бота. Например, при анализе эскалированных диалогов команда PharmaBot обнаружила интересный паттерн:

Самой большой проблемой PharmaBot оказалось частое неверное понимание запросов пользователей, особенно при различении вопросов о тестировании на COVID-19 и записи на вакцинацию. Пользователи были недовольны, когда бот их не понимал, и часто завершали диалог — либо вовсе отказываясь от взаимодействия, либо переходя к оператору. Сотрудники колл-центра подтвердили, что регулярно слышали жалобы на эту проблему от клиентов. Аналитика также показала высокую частотность данного сценария.

Команда решила устранить эту высокоприоритетную проблему. Бизнес-цель состояла в том, чтобы завершать процесс записи без вмешательства оператора. Был выявлен повторяющийся паттерн сбоев: бот не понимал, какого именно типа запись интересовала пользователя, и пользователи чувствовали себя непонятыми, — в результате многие диалоги оказывались неуспешными. Было обнаружено сразу две проблемы: непонимание запроса и чрезмерно сложный сценарий диалога.

Прежде чем перейти к решению проблем, команде необходимо их оценить и расставить приоритеты исходя из важности проблемы и ожидаемого эффекта для системы. Сюда относится оценка таких факторов, как частота возникновения проблемы, стоимость ее устранения и потенциальные выгоды от решения. Структурированный подход к приоритизации позволяет обеспечить максимальную ценность при ограниченных ресурсах. В случае PharmaBot наивысший приоритет

был присвоен устранению проблемы с непониманием типов записей на прием, так как это напрямую влияло на возможность завершить ключевой бизнес-процесс — забронировать время для приема. Рисунок 3.15 иллюстрирует пример оценки проблемы. Для более подробного анализа можно дополнительно учитывать такие показатели, как масштаб проблемы, результаты диалогов, сложность устранения для пользователей, влияние на другие сценарии и прочие факторы.

Выводы
- Сценарий «Запись на прием: бронирование времени» показал значительный рост числа эскалаций после изменений от 29 января.
- Рост, вероятно, был вызван добавлением формулировки: «переключить на оператора для записи на прием».

Рекомендации
- Изменить формулировку, чтобы сократить ответ и разбить диалог.
- Добавить уточняющие вопросы для пользователей, обращающихся к операторам.
- Разработать систему записи через SMS.

Рис. 3.15. Анализ роста числа эскалаций в сценарии «Запись на прием: забронировать время» с выявлением возможной причины (новая формулировка, направляющая пользователей к операторам) и рекомендациями для улучшения

Представленный на рис. 3.15 анализ проблемы может быть частью более широкой процедуры классификации, помогающей расставлять приоритеты улучшений чат-бота в зависимости от их эффекта и сложности устранения. Каждая проблема документируется аналогичным образом с указанием сути, вероятных причин и предложенных решений. В полном процессе классификации для определения приоритетности проблем они оцениваются по оказываемому ими эффекту, частоте и сложности решения. Для этого необходимо учитывать ценность, предполагаемые результаты и ожидаемую отдачу. Необходимо проводить анализ «затрат/выгод». Выгоды могут быть прямыми (например, повышение коэффициента удержания) или косвенными (улучшение пользовательского опыта). Затраты включают время, усилия и сложность исправления, при этом изменения, требующие согласования с несколькими отделами, займут больше времени. Ожидаемая отдача учитывает как выгоды, так и затраты, в зависимости от объема затронутых диалогов. Цель заключается в том, чтобы сосредоточиться на тех областях, где ожидаемая отдача оправдывает вложение времени и ресурсов. Пример расчета ожидаемой отдачи по проблеме приведен на рис. 3.16.

Финансовые затраты легко рассчитать в случае, если определенный сценарий диалога обрабатывается оператором, а не чат-ботом. Расчет учитывает стоимость звонка оператору, общее количество звонков в день и процент диалогов, перенаправленных на сотрудника. При наличии финансового обоснования определить приоритетность проблемы становится проще. Подобный расчет можно повторить для всех типов проблем. При этом следует помнить, что некоторые затраты являются косвенными: например, невежливый или неумелый бот может снижать удовлетворенность клиентов, что сложно напрямую выразить в финансовом эквиваленте.

Предположения			
Стоимость звонка оператору	$6,00		
Количество звонков в день	80 000		
Процент прерванных диалогов	10 %		
Количество прерванных диалогов в день	8000		
Количество дней в периоде	30		
	Высокие значения	**Средние значения**	**Низкие значения**
Процент обратных звонков в день	100 %	50 %	25 %
Недовольные пользователи, перенаправленные к оператору	240 000	124 000	60 000
Стоимость обратных звонков за период	**$1 440 000**	**$720 000**	**$360 000**

Рис. 3.16. Оценка стоимости обращений в колл-центр после того, как пользователи прервали взаимодействие с чат-ботом из-за неудовлетворительных результатов

Усилия — еще один важный фактор приоритизации. Так же как у самой проблемы есть своя цена, ее исправление также приводит к определенным затратам. Усилия включают в себя время, ресурсы и сложность реализации изменений. Ключевая задача — найти баланс между значимостью проблемы и скоростью ее решения. Оптимальными для устранения являются проблемы с высоким ожидаемым эффектом и низкими затратами на исправление. В первую очередь стоит устранять те из них, которые оказывают значительное влияние, но при этом легко решаются. На рис. 3.17 показана классификация улучшений в зависимости от возможного эффекта и требуемых усилий на реализацию.

Элементы высокой приоритетности обладают большим эффектом и требуют небольших усилий, далее следуют элементы средней приоритетности, обеспечивающие пошаговые результаты. Низкоприоритетные задачи характеризуются

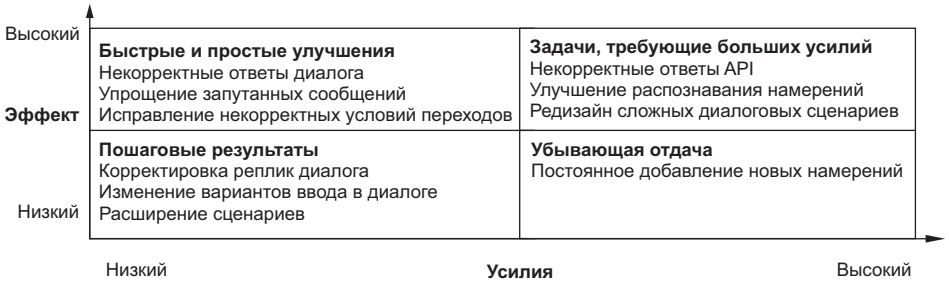


Рис. 3.17. Матрица «эффект — усилия» визуализирует взаимосвязь между требуемыми усилиями и потенциальным эффектом от предлагаемых изменений

низким эффектом, требуют больших усилий на реализацию и являются менее перспективными для исправления. Такая матрица помогает эффективно расставлять приоритеты, фокусируясь на изменениях, дающих наибольший эффект при минимальных усилиях.

Можно подробнее рассмотреть категории, представленные в матрице. Для каждой категории можно привести примеры проблем, причиняющих неудобства пользователям, и объяснить причины их возникновения. Таблица 3.8 начинается с примеров проблем с высоким эффектом, требующих небольших усилий, табл. 3.9 показывает примеры проблем с высоким эффектом, требующих больших усилий, а табл. 3.10 — примеры проблем со средним эффектом, требующих больших усилий.

Таблица 3.8. Примеры проблем с высоким эффектом, требующих небольших усилий

Проблема	С чем столкнулись пользователи	Возможные причины возникновения
Некорректный или неполный ответ	Чат-бот не понимает	Некорректный ответ из-за слабого распознавания намерений, недостаточной валидации входных данных или отсутствия адаптации к контексту пользователя
Неудачный ответ в диалоге	Чат-бот слишком сложный	Формат и/или текст не доносят информацию достаточно ясно
Неработающие диалоговые ветки	Чат-бот не работает	Чат-бот функционирует неправильно из-за некорректных или неправильно настроенных условий и переходов в сценарии диалога. Подобные ошибки возникают, когда логика, определяющая переходы между шагами (так называемые прыжки), некорректна или недостаточно протестирована. Это может привести к тупиковым сценариям, неуместным ответам или внезапному обрыву диалога, что отрицательно влияет на удержание и удовлетворенность пользователей
Расширение сценариев	Чат-бот слишком сложный	Сценарии содержат запутанные шаги, вызывающие затруднения у пользователей. Особенно характерно для длинных диалогов

Таблица 3.9. Примеры проблем с высоким эффектом, требующих больших усилий

Проблема	С чем столкнулись пользователи	Возможные причины возникновения
Пользователь не может завершить сценарий	Чат-бот слишком сложный	Сбои возникают из-за множества шагов в диалоге, что требует полного редизайна сценария
Вопросы пользователей не обрабатываются	Чат-бот не понимает	Недостаточно намерений для покрытия пользовательских запросов. Может потребовать добавления поиска или RAG для обработки редких типов вопросов

Таблица 3.10. Примеры проблем со средним эффектом, требующих больших усилий

Проблема	С чем столкнулись пользователи	Возможные причины возникновения
Неполный ответ (из-за сбоя API)	Чат-бот не понимает	Неполный ответ вследствие ошибки API. Бот может не поддерживать все варианты запросов или ответов API
Неправильно определенное намерение	Чат-бот не понимает	Путаница может возникнуть, если обучающие данные несбалансированны, то есть некоторые намерения содержат слишком мало или слишком много примеров высказываний, что приводит к ошибочной классификации. Кроме того, расхождения между обучающими данными и реальными пользовательскими запросами затрудняют правильное распознавание намерений

Указанные категории не являются абсолютными, необходимо корректировать приоритетность изменений с учетом частоты возникновения проблем.

Разработка решения для устранения выявленной проблемы

После определения проблемы с высоким приоритетом следующим важным шагом становится разработка решения — определение концепции устранения проблемы. Этот процесс предполагает совместную работу команды для формирования комплексного решения. Чтобы обеспечить структурированный подход, необходимо ответить на три ключевых вопроса: кто будет отвечать за внедрение исправления, какие изменения должны быть внесены и как будет реализовано решение. «Кто» охватывает конкретные роли и зоны ответственности команды, которая внедряет исправление. «Что» определяет характер решения: улучшение возможностей обработки естественного языка, повышение качества учета контекста или внедрение более продвинутой системы распознавания намерений. «Как» описывает технический подход и методологии, необходимые для реализации решения.

Кроме того, команда должна определить объем требуемых усилий, учитывая такие факторы, как сложность кода, требования к интеграции и возможные зависимости от внешних систем. Этап разработки решения имеет ключевое значение для построения обоснованного плана непрерывного улучшения.

Назначение приоритетов

Таблица приоритетности исправлений позволяет эффективнее управлять инициативами. Она содержит порядковые номера улучшений, краткие описания выявленных проблем, а также количественную оценку потенциального влияния исправлений на пользовательский опыт и ссылки на связанные задачи в GitHub. Такая комплексная структура не только упрощает процесс разработки,

но и способствует эффективной коммуникации и совместной работе членов команды. На рис. 3.18 представлен пример такой таблицы.

Приоритет	Описание	Рекомендуемое исправление	Ценность/эффект	ID
1	Существует дисбаланс обучающих данных между намерениями, что приводит к низкому качеству распознавания одного из намерений — «записаться на прием»	Расширить обучающие данные для намерения с помощью пользовательских высказываний из продакшена	Болевая точка: бот не понимает пользователя. Техническая ценность: повышение точности распознавания намерений. Бизнес-метрики: удержание	325
2	Бот сообщает пользователю, что введен несуществующий ID клиента	Переписать диалог, сделать его более вежливым. Подробнее объяснить пользователю, как найти ID на карте	Болевая точка: слишком сложный сценарий. Бизнес-метрики: NPS	334

Рис. 3.18. Пример таблицы приоритетности исправлений

Каждый столбец таблицы играет важную роль в эффективном устранении проблем:

- *Приоритет* — устанавливает степень срочности каждой проблемы, благодаря чему критически важные задачи решаются в первую очередь.
- *Описание* — предоставляет краткий обзор выявленной проблемы.
- *Рекомендуемое исправление* — содержит предлагаемое решение или улучшение для устранения проблемы, определяет направление работы.
- *Ценность/эффект* — отражает ожидаемое улучшение удовлетворенности пользователей или удобства использования бота в результате предлагаемых изменений, тем самым помогая расставлять приоритеты.
- *ID* — ссылка на соответствующую проблему в трекере задач проекта, например в репозитории GitHub, что упрощает совместную работу и отслеживание каждой задачи. Описание задачи на GitHub может содержать более подробные пояснения, ход выполнения и статус проблемы.

3.4.3. Определение критериев приемлемости

После определения приоритетности каждой проблемы и разработки концепции их решения следующим шагом становится определение критериев приемлемости (acceptance criteria) исправлений. Другими словами, как мы поймем, что проблема решена? Критерии приемлемости упрощают проверку функциональности в среде разработки и валидацию улучшений в продакшене.

Например, когда PharmaBot не справлялся с ответом «да» на вопрос «Вы хотите записаться на вакцинацию или сдать тест?», критерии приемлемости могли выглядеть следующим образом:

1. Когда PharmaBot просит пользователей выбрать между записью на вакцинацию и сдачей теста и они отвечают «вакцинация», бот записывает их на вакцинацию.
2. Когда PharmaBot просит пользователей выбрать между записью на вакцинацию и сдачей теста и они отвечают «тест», бот записывает их на сдачу теста.
3. Когда PharmaBot просит пользователей выбрать между записью на вакцинацию и сдачей теста и они отвечают «да», бот уточняет, хотят ли они записаться на вакцинацию.

Такие критерии приемлемости позволят команде тестирования проверить как существующую функциональность (критерии 1 и 2), так и новую (критерий 3). Нельзя внедрять исправление, если оно не соответствует всем критериям приемлемости.

После исправления команда сможет убедиться, что количество диалогов, завершающихся (неудачно) ответом «да» на вопрос «вакцинация или сдача теста?», резко снизилось или исчезло вовсе. Улучшились ли исходные метрики? Четкие и измеримые стандарты обеспечивают согласованность работы команды с ожиданиями пользователей и целями проекта, что создает основу для успешного улучшения бота.

Задания

1 Следующие примеры проблем относятся к внедрению систем разговорного ИИ и различаются по уровню сложности. Ваша задача — расставить приоритеты для этих проблем на основе их влияния на эффективность и результативность разговорного ИИ, учитывая как качественные, так и количественные метрики. При определении приоритетности проблем учитывайте их сложность. Примеры охватывают различные отрасли, так как данный подход к улучшению и расстановке приоритетов применим к чат-ботам самых разных типов и сфер применения.

a) Некорректная генерация ответов:

- Описание: чат-бот периодически выдает некорректные или нерелевантные ответы на запросы пользователей, что приводит к их неудовлетворенности и недовольству.
- Эффект: высокий — влияет на пользовательский опыт и доверие к чат-боту.
- Метрика: частота некорректных ответов в день (например, 15 % от общего числа ответов).

```
Пример диалога:
Пользователь: "Расскажи о побочных эффектах вакцины".
```

```
Чат-бот: "Вакцина против COVID-19 безопасна и эффективна".
Пользователь: "Но я слышал, что некоторые люди тяжело переносят
    вакцинацию. Можешь дать больше информации?"
Чат-бот: "Вакцина безопасна и эффективна".
Пользователь: (Выходит из чата.)
```

б) Медленный ответ:

- Описание: чат-бот слишком долго формирует ответы, что раздражает пользователей, особенно в экстренных ситуациях.
- Эффект: средний — негативно влияет на удовлетворенность и вовлеченность пользователей.
- Метрика: среднее время ответа на запрос пользователя в секундах (например, 8 секунд).

```
Пример диалога:
Пользователь: "Расскажи, где можно сдать тест на COVID".
(Долгая пауза, пользователь выходит из чата.)
```

в) Ограниченное понимание языка:

- Описание: чат-боту трудно понимать запросы, содержащие разговорную лексику, сленг или сложный синтаксис, что приводит к неверной интерпретации и неадекватным ответам.
- Эффект: средний — ограничивает способность бота эффективно взаимодействовать с пользователями, вызывает раздражение и снижает удовлетворенность.
- Метрика: процент непонятых запросов в день (например, 8 % от общего числа запросов).

```
Пример диалога:
Пользователь: "Что там с вакциной?"
Чат-бот: "Извините, я не понимаю ваш вопрос. Попробуйте еще раз".
```

г) Непоследовательная интеграция с бэкенд-системами:

- Описание: чат-бот сталкивается с несоответствиями при интеграции с бэкенд-системами, что приводит к предоставлению пользователям неполной или некорректной информации.
- Эффект: высокий — подрывает надежность чат-бота и доверие к предоставляемой им информации.
- Метрика: процент диалогов с ошибками интеграции с бэкендом в день (например, 12 % от общего числа диалогов).

```
Пример диалога:
Пользователь: "Можешь проверить, есть ли свободные места на
    вакцинацию завтра?"
Чат-бот: "Извините, возникли технические трудности. Попробуйте
    позже".
```

2 На примере последних диалогов с вашим чат-ботом повторите предыдущее задание.

3.5. Разработка и внедрение исправлений

Непрерывное улучшение зачастую осуществляется с помощью нескольких итераций фиксированной продолжительности, которые обычно называют спринтами. Их длительность варьируется от одной до четырех недель в зависимости от правил, установленных в компании. В то время как таблица приоритетности исправлений служит общей дорожной картой, план спринта конкретно перечисляет функции, которые будут добавлены в следующую очередь. На него влияет доступность ресурсов — объем работы, который можно выполнить и протестировать в заданные сроки. Кроме того, благодаря плану стейкхолдеры понимают, чего ожидать от ближайшего обновления.

3.5.1. Планирование спринта

Данное действие является частью системного подхода к обнаружению и устранению проблем. Оно необходимо для скоординированного и гибкого процесса разработки и гарантирует, что бот развивается в соответствии с предложенными решениями и в установленные сроки. Существуют различные инструменты, такие как, например, канбан-доски, позволяющие визуализировать состояние спринта на протяжении его выполнения. Базовая схема спринта должна включать в себя проблемы, над которыми ведется работа, и статус их исправления. На рис. 3.19 представлено дополнение таблицы исправлений (рис. 3.18) двумя столбцами — статус и сроки — с указанием соответствующего спринта.

Приоритет	Описание	Рекомендуемое исправление	Ценность/эффект	ID	Статус	Сроки
1	Существует дисбаланс обучающих данных…	Расширить обучающие данные…	Болевая точка: бот не понимает…	325	В разработке	Спринт 3
2	Бот сообщает пользователю, что…	Переписать диалог…	Болевая точка: слишком сложный…	334	Запланировано	Спринт 4

Описание проблемы **Планирование**

Рис. 3.19. Таблица приоритетности, дополненная указанием спринтов. Также можно добавить столбцы с указанием сроков тестирования и ожидаемых дат развертывания

3.5.2. Повторная оценка

Команда PharmaBot приложила значительные усилия для внедрения всех улучшений и после ввода их в продакшен стала отслеживать метрики, на которые рассчитывала повлиять. Сравнивая полученные результаты диалогов с двумя

предыдущими релизами, разработчики смогли убедиться, что исправления достигли ожидаемых результатов.

На рис. 3.20 показан дашборд команды PharmaBot.

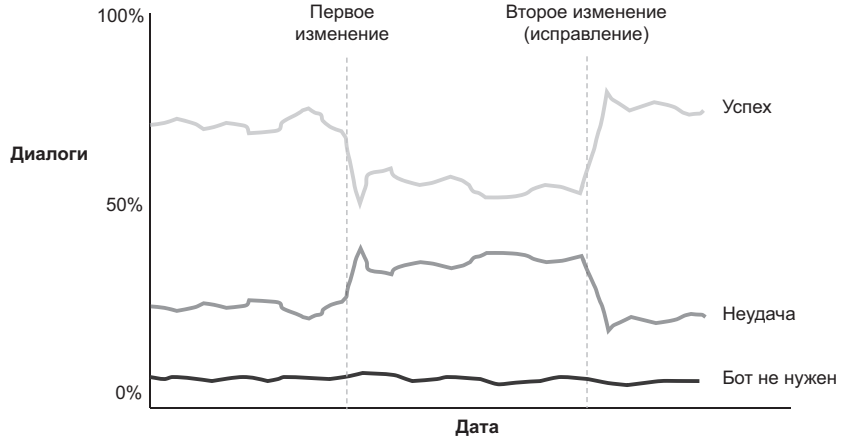


Рис. 3.20. Отслеживание результатов диалогов по мере внедрения изменений

Задания

1 Планирование спринта является ключевым этапом устранения ошибок и внедрения улучшений в PharmaBot. Смоделируйте сессию планирования спринта и расставьте приоритеты для всех исправлений и улучшений в рамках одной итерации. У вас в команде есть два аналитика диалогов, разработчик бэкенда на неполный рабочий день и тестировщик:
   - Проанализируйте таблицу приоритетности исправлений из предыдущего задания.
   - Распределите ресурсы с учетом эффективности и скорости работы команды.
   - Составьте план спринта. При помощи канбан-доски или аналогичного инструмента создайте план, включающий приоритетные исправления и улучшения, оценку трудозатрат и предполагаемое время выполнения. Добавьте столбцы статуса и сроков, чтобы лучше отслеживать прогресс и обеспечивать прозрачность работы на протяжении всего спринта.
   - Обсудите ожидаемые даты развертывания исправлений и улучшений, запланированных для спринта.

Итоги

- Непрерывное улучшение разговорных систем представляет собой постоянный итеративный процесс.

- Все улучшения должны быть направлены на достижение заранее определенных бизнес-целей и обеспечение удовлетворенности пользователей.
- Четко определенные метрики, правильно подобранные инструменты мониторинга и применение лучших практик имеют ключевое значение для успешной разработки системы.
- Используйте «правильные» метрики, подходящие именно вашему боту, а не те, что проще измерить.
- Детализированный анализ результатов диалогов позволяет выявить конкретные области для улучшения.
- На приоритетность проблемы влияют различные факторы, такие как частота ее возникновения, ожидаемое улучшение и сложность исправления, а также ресурсы команды.
- Регрессионное тестирование и анализ улучшений позволяют убедиться в достижении поставленных целей.

Часть 2

Паттерн: ИИ не понимает пользователя

«Чат-бот меня не понимает» является наиболее распространенной проблемой пользователей при взаимодействии с разговорным ИИ. В традиционных чат-ботах это связано с плохо обученным классификатором, для других же причиной может быть неэффективный поисковый механизм или просто отсутствие доступа к нужной информации.

Независимо от того, чат-бота какого типа вы создаете, ключевая способность, которой он должен обладать, — это умение понимать, чего хочет пользователь, и правильно ему отвечать. Чат-боты, понимающие запросы, дают полезные ответы; те же, которые не понимают, отвечают что-то вроде: «Извините, я вас не понимаю. Попробуйте еще раз».

Существует несколько методов устранения проблем с пониманием, и все они сводятся к выявлению целей пользователей и того, как они формулируют свои запросы. В главе 4 мы рассмотрим, как извлекать подобные данные из логов и других источников. В главе 5 показано, как с помощью обучения классификаторов улучшить способность чат-бота понимать клиентов, а в главе 6 мы добавим в чат генеративный ИИ, дополненный поиском (RAG). Глава 7 рассматривает, как использование генеративного ИИ на этапах обучения и тестирования позволяет создавать новые обучающие и тестовые данные с помощью LLM.

4 Понимание потребностей пользователей

В этой главе

- ✓ Определение признаков слабого понимания
- ✓ Измерение уровня понимания пользователей чат-ботом
- ✓ Оценка текущего состояния чат-бота
- ✓ Сбор и подготовка логов для измерения уровня понимания
- ✓ Интерпретация исходных логов

Хороший чат-бот способен распознавать (понимать), чего хочет пользователь. Это одна из ключевых метрик, используемых для оценки его эффективности. Чат-бот может демонстрировать высокий уровень понимания еще на начальных этапах внедрения (или, по крайней мере, «неплохой» для первого релиза), но со временем все чаще возвращать неправильные ответы. Вы можете это заметить, например, по более частым жалобам пользователей — через самого чат-бота («Ты не ответил на мой вопрос!») или в виде результатов опросов. При этом вовлеченность будет снижаться, показатель отказов — расти, а центр поддержки будет все чаще сообщать о случаях эскалации в задачах, которые должен был обрабатывать виртуальный помощник. Все это указывает на то, что ваш разговорный ИИ, возможно, плохо понимает собеседника.

В теории чат-боты должны постоянно улучшаться, но порой наблюдается, наоборот, снижение качества понимания. Наша цель — помочь вам распознать, когда

и почему это может происходить, а также рассказать, как избежать всевозможных подводных камней и заранее подготовиться к типичным сценариям жизненного цикла системы. В этой главе мы рассмотрим, что означает «высокая эффективность» разговорного ИИ в рамках его способности правильно определять или классифицировать цель пользователя (то есть понимать его). Мы также приведем несколько методов подготовки данных для их дальнейшего использования при измерении качества классификатора или оценке сгенерированных ответов.

4.1. Основы понимания

Понимание — это основа человеческой коммуникации. В разговорном ИИ мы используем методы обработки естественного языка, чтобы определить, чего хочет пользователь или что ему нужно. Так как диапазон возможных потребностей практически бесконечен, а количество способов их языкового выражения неисчерпаемо, это чрезвычайно сложная задача.

4.1.1. Последствия слабого понимания

Отсутствие понимания со стороны чат-бота, вероятнее всего, является главным источником разочарования клиента, который обратился за ответом и, возможно, даже получил его, но он никак не связан с вопросом. Или же чат-бот попросил переформулировать запрос, и тогда клиенту приходится подбирать другие слова для выражения той же мысли. Иногда это помогает, а иногда он снова получает ответ с просьбой перефразировать запрос. В большинстве случаев пользователи просят соединить их с оператором после одной-двух неудачных попыток — см. рис. 4.1.

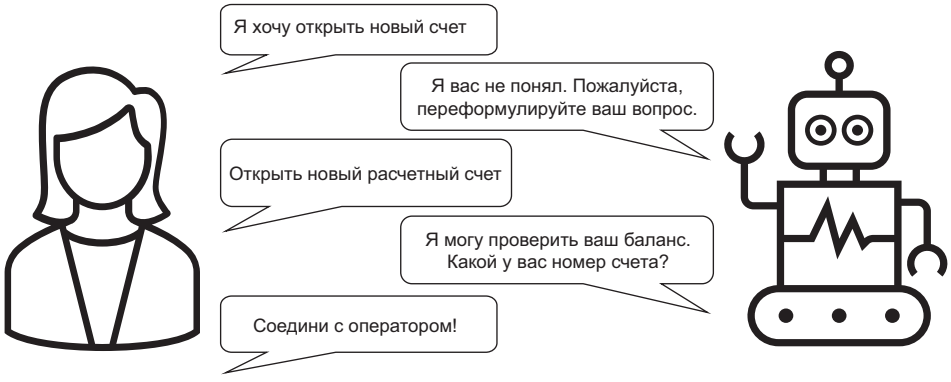


Рис. 4.1. Проблемы с точностью или охватом раздражают пользователя, так как для достижения цели требуется больше времени и зачастую несколько попыток. Пользователь теряет доверие к виртуальному помощнику

Если ваши пользователи сталкиваются с подобной ситуацией, скорее всего, проблема связана с *точностью* (способностью чат-бота сопоставлять полученное высказывание с имеющимися знаниями), *охватом* (спектром знаний, с которыми

система умеет рабатать) или и тем и другим одновременно. Невозможно сразу определить, что именно является главной причиной. Для этого потребуется сбор данных, без которых трудно понять, что необходимо исправить, а неправильное исправление может только усугубить существующие проблемы. В итоге ваша система разговорного ИИ станет дорогостоящей и сложной в сопровождении. Более того, она перестанет приносить заявленную ценность (например, будет не снижать, а, наоборот, даже повышать потребность в человеческом вмешательстве).

Одним из важнейших факторов успеха бота является то, как ваша компания подходит к его последующему сопровождению. В идеале спонсор проекта и команда поддержки должны сразу принять тот факт, что система потребует итеративных улучшений, особенно на начальном этапе, когда она столкнется с реальными пользователями. Несмотря на достижения в области самообучения больших языковых моделей и генеративного ИИ, чат-боты редко улучшаются «сами по себе».

Ресурсы поддержки понадобятся на протяжении всего жизненного цикла бота

Ваша компания считает, что чат-бот — это разовое решение по принципу «сделал и забыл», и не уделяет должного внимания поддержке и сопровождению виртуального помощника? Подобная позиция практически гарантирует неудачу в будущем.

Чат-бот, по сути, является цифровым сотрудником. Как и любой человеческий ресурс, он требует первоначального обучения, периодического повышения квалификации, поощрения и возможности осваивать новые навыки.

4.1.2. Причины слабого понимания

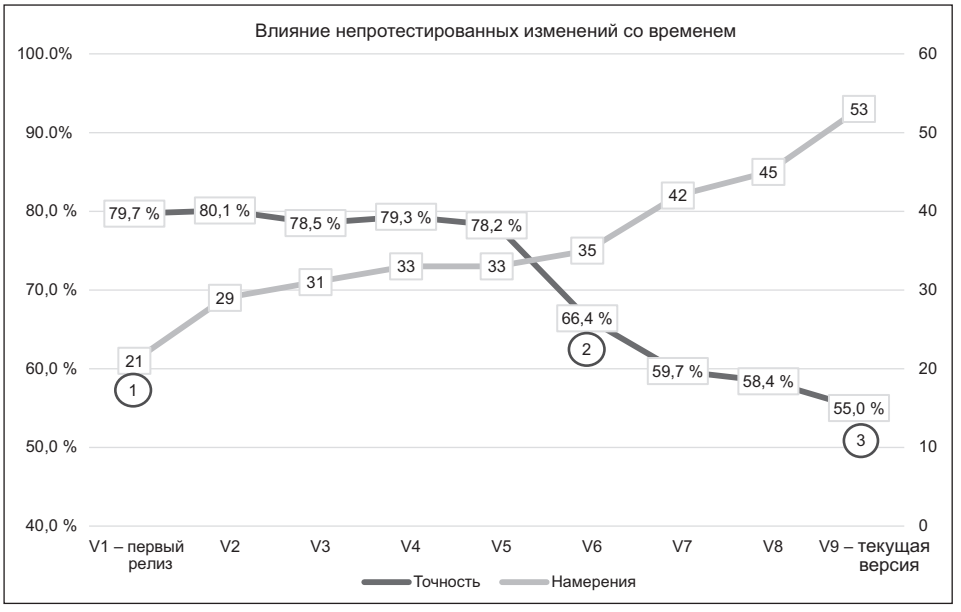
К наиболее распространенным причинам снижения уровня понимания чат-ботом относятся:

- искусственные обучающие данные (обучающие примеры, не отражающие реальный словарный запас пользователей);
- недостаточный охват тем или пробелы в охвате;
- появление новой информации, которая не была предоставлена виртуальному помощнику;
- отсутствие процесса проверки и контроля качества при добавлении новых намерений, обновлении существующих или изменении параметров инференса модели.

Последний пункт — отсутствие контроля — приводит к самым сложным для устранения проблемам. Без надзора со стороны компетентного владельца или специальной команды по обучению модели непроверенные изменения могут только усугубить ситуацию. В традиционных классификаторах обновления модели, внесенные специалистом, не знакомым с полным набором обучающих данных, часто приводят к дублированию, конфликтам в обучении намерений и необоснованным перекосам в распределении примеров между различными

намерениями. Непротестированные изменения параметров модели или промптов приводят к непредсказуемому поведению генеративной модели.

Мы наблюдали подобное у клиента, который внес изменения в обучающий набор своего классификатора, увеличив общее количество намерений с 21 до 53 в ходе девяти релизов. Сначала негативный эффект был незаметен, но со временем последствия этих непроверенных изменений проявились в виде плохих результатов опросов, прерванных сценариев, ненужных эскалаций и большого количества негативной обратной связи. Эксперты в предметной области сообщали, что бот начал давать неправильные ответы на вопросы, которые раньше обрабатывались корректно. Это типичные признаки слабого понимания, однако конкретный момент, когда данная проблема появилась, выявить не удалось. Серия ретроспективных экспериментов с предыдущими версиями показала результаты, представленные на рис. 4.2.



1. Клиент запустил бот с 21 намерением. Количественные измерения не проводились, однако результаты пользовательского тестирования были оценены как удовлетворительные. Начальная точность (измеренная ретроспективно) составила 79,7 %.

2. Мы протестировали каждую версию, пытаясь найти причины недовольства пользователей текущим состоянием системы. Первое существенное падение точности классификатора произошло, когда клиент добавил два намерения в версии V6. Точность снизилась до 66,4 %.

3. Далее клиент продолжил добавлять намерения, не осознавая, что эффективность системы стремительно ухудшается, пока ее точность не достигла 55 %.

Рис. 4.2. Ретроспективная оценка эффективности классификатора наглядно демонстрирует, как непроверенные изменения со временем влияют на систему. Если бы каждая версия тестировалась перед развертыванием, команда не обновляла бы бота до устранения всех проблем. Потребовалось несколько недель, чтобы вернуть классификатор в рабочее состояние

4.1.3. Как обеспечить понимание в традиционном разговорном ИИ?

Традиционные (негенеративные) системы разговорного ИИ обучаются на примерах пользовательских запросов, сгруппированных по *намерениям* (intents), которые иногда называют *классификациями* (classificatios) или *кластерами* (clusters). Намерения включают в себя различные перефразированные высказывания, выражающие одну и ту же цель. Некоторые системы также используют *сущности* (entities), которые работают как ключевые слова, уточняющие значение или характеристики запроса.

Логика диалога настраивается так, чтобы определять намерение (или комбинацию намерение + сущность) и на его основе выполнять действие. Действие может быть простым, например ответ на вопрос, или же инициировать сложный сценарий. В табл. 4.1 приведены примеры намерений, сущностей и возможных последующих шагов в диалоге.

Таблица 4.1. Примеры высказываний могут обрабатываться по-разному в зависимости от наличия или отсутствия сущностей

Высказывание	Намерение	Сущность	Возможный следующий шаг
«Сколько сумок я могу сдать в багаж?»	`Bag_Allowance`		Отобразить правила провоза багажа
«Я хочу купить билет»	`Book_Flight`		Сбор информации о пункте назначения
«Мне нужен билет в один конец до Коста-Рики»	`Book_Flight`	Коста-Рика	Сбор данных о вылете
«Я хочу повысить класс обслуживания до первого»	`Flight_Upgrade`	Первый класс	Запуск процесса апгрейда

К ботам, использующим традиционную классификацию, как правило, относятся агенты маршрутизации по темам, вопросно-ответные боты (FAQ), а также в некоторой степени процессно-ориентированные помощники. Важно помнить, что боты-классификаторы опираются на предопределенный набор тем (намерений), поэтому необходимо заранее предположить, с какими вопросами, скорее всего, столкнется ваш бот.

С практической точки зрения диапазон тем или намерений, которым вы обучаете систему, будет специфичен для вашей предметной области и цели использования. Как владельцы решения, мы должны постоянно настраивать систему так, чтобы она корректно понимала максимальный объем пользовательских запросов. Найти оптимальный баланс между широтой и глубиной возможных тем сложно, и это

зачастую требует компромиссов. Например, экономически нецелесообразно обучать классификатор распознавать абсолютно все возможные темы. Более того, это может ухудшить понимание тех из них, которые действительно значимы для пользователей.

Когда компания стремится обучить классификатор распознавать каждую возможную тему, его способность различать намерения ослабевает. Если намерения в вашей системе не отражают реальный пользовательский спрос (например, присутствует большое количество низкочастотных тем), это приводит к проблемам с точностью и уверенностью. На рис. 4.3 показана диаграмма с длинным «хвостом»: наибольшая бизнес-ценность бота-классификатора обычно достигается за счет обработки запросов высокой и средней частотности. Низкочастотные запросы, как правило, обрабатываются с помощью резервных механизмов, таких как эскалация, поиск или генеративный ИИ.

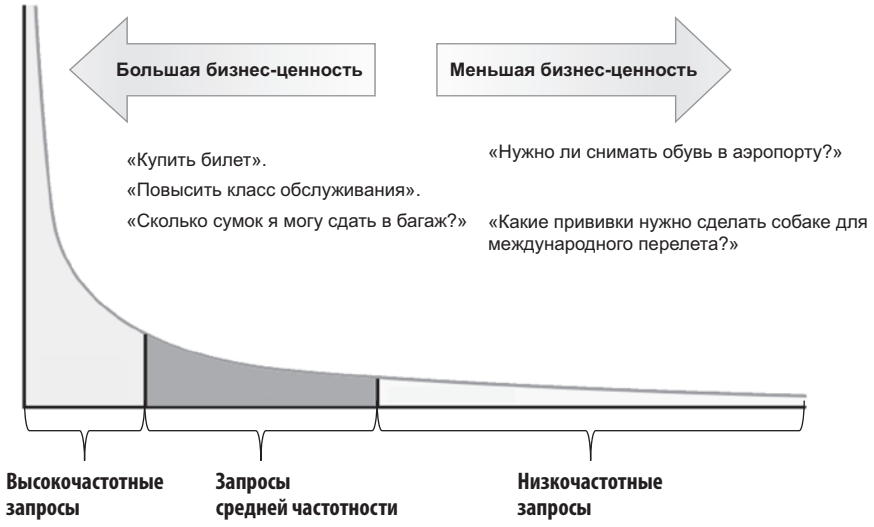


Рис. 4.3. По мере снижения частотности запросов диаграмма сужается в длинный «хвост». Каждое решение должно определять оптимальный баланс глубины и широты тем. Пороговая точка по бизнес-ценности обычно находится в диапазоне запросов средней частотности. Это не означает, что все низкочастотные запросы должны быть исключены, однако расширение охвата классификатора за их счет может иметь меньший эффект

До первого запуска необходимо предположить, какие темы наиболее важны для понимания ботом. Такие прогнозы зачастую основываются на логах взаимодействий с операторами, метриках колл-центра, фокус-группах, опросах или других исследованиях. Основное внимание должно быть уделено обучению модели распознавать подобные запросы, а также введению вспомогательных

намерений для поддержания диалога (например, приветствие, простая беседа, повтор, эскалация). После отправки решения в продакшен необходимо подтвердить эти прогнозы, собрав и проанализировав данные о взаимодействиях.

4.1.4. Как обеспечить понимание в генеративном ИИ?

Как генеративный ИИ понимает пользователей? Это вопрос с подвохом, поскольку в действительности он не столько понимает значение высказывания, сколько создает новые данные, похожие на те, на которых он был обучен, используя высказывание в качестве исходной точки. Это тонкое различие, но в случае генеративного ИИ мы стремимся лишь имитировать понимание, обучая модель оценивать входные данные с определенной позиции и затем генерировать конкретный тип вывода.

В разговорном ИИ наша цель заключается в генерации выходных данных, которые отражают или учитывают запрос пользователя с какой-либо конкретикой и/или персонализацией (а не только на уровне высокоуровневой категоризации, такой как классификация по темам или распознавание сущностей). На рис. 4.4 показано принципиальное различие между выходами классификационных и генеративных моделей.

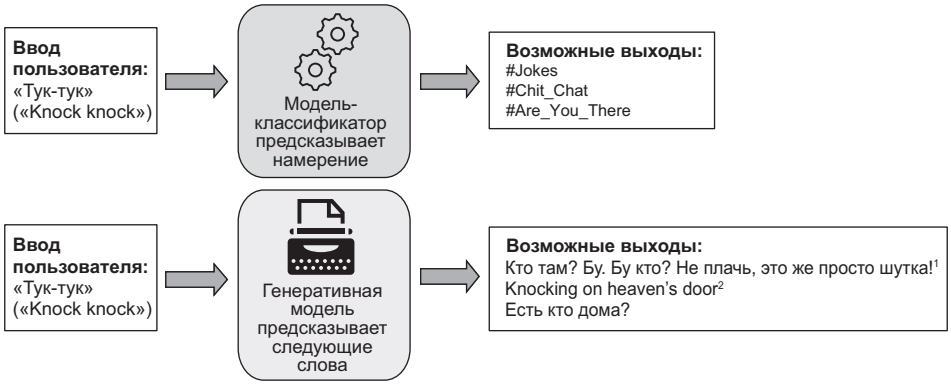


Рис. 4.4. Традиционные модели-классификаторы используют обучение с учителем для предсказания одной из нескольких предопределенных классификаций. Они распознают намерение или значение пользовательского ввода. Генеративные модели используют декодирующие трансформеры для продолжения текста. Они предсказывают последовательность токенов (условно говоря, слов или символов), которые с наибольшей вероятностью могут следовать за пользовательским вводом

---

[1] Известная английская шутка-каламбур, построенная на сходстве звучания вопроса «Boo who?» (Бу кто?) с плачем. — *Примеч. пер.*

[2] «Knock, knock, knockin' on Heaven's door» — строчка из песни Боба Дилана «Knockin' On Heaven's Door». — *Примеч. пер.*

Небольшое примечание об архитектурах LLM

Архитектуры только с энкодером лучше всего подходят для негенеративных сценариев, таких как обучение прогнозирующих моделей на основе текстовых эмбеддингов. Они извлекают значимый контекст из входных данных и требуют разметки для тонкой настройки.

Архитектуры только с декодером предназначены специально для генеративных сценариев. Такие модели обучаются без учителя путем обработки больших объемов данных. Они прогнозируют следующий токен в последовательности и могут быть настроены на выполнение конкретных задач, включая классификацию, ответы на вопросы и резюмирование.

Некоторые архитектуры LLM представляют собой *энкодер-декодеры*, что позволяет им поддерживать как генеративные, так и негенеративные сценарии. Обычно они применяются в случаях, когда входные данные объемные, а выходные — относительно небольшие, например при переводе или резюмировании.

В отличие от традиционного ИИ на основе классификатора, у генеративной модели нет заранее определенного списка намерений, входящих в область охвата. Однако, как и в традиционном ИИ, необходимо хорошо владеть предметной областью и диапазоном проблем, с которыми пользователи могут обратиться к вашему боту. Это определит стратегии, которые вы примените, чтобы направить вашу LLM на генерацию ответов, демонстрирующих понимание пользовательского ввода. Для этого есть несколько эффективных инструментов:

- *Выбор модели, подходящей для конкретной задачи*, — некоторые модели лучше оптимизированы для генерации разговорного текста (в отличие от написания кода, эссе или новостной статьи).
- *Промпт-инжиниринг* — данный метод предполагает предоставление модели входных данных для генерации оптимальных выходов. Такие данные могут включать инструкции, контекст, входные данные и индикаторы выхода. Промпт-инжиниринг позволяет достичь качественной имитации понимания и настроить модель на генерацию текста в разговорном стиле.
- *One-shot- или few-shot-промпты* — вы можете подкрепить промпты одним или несколькими примерами желаемого результата в необходимом вам формате, который должна сгенерировать модель.
- *Настройка параметров* — такие параметры, как температура[1], top-*p* и top-*k*, влияют на случайность и разнообразие генерируемого текста. Увеличение значений, как правило, повышает «креативность» ответа.
- *Генерация, дополненная поиском (RAG)*, — применение RAG усилит впечатление, что бот реально понимает пользователя, при этом продолжая выдавать

[1] Температура (temperature) — гиперпараметр, управляющий степенью случайности при генерации текста. Влияет на то, насколько креативным будет ответ модели. — *Примеч. ред.*

ответы в рамках вашей предметной области. Многие компании применяют RAG в своих разговорных ИИ, чтобы гарантировать, что ответы основаны на внешних проверяемых фактах и актуальной информации.

На момент написания данной книги корпоративные решения чаще всего используют генеративный ИИ в Q&A-ботах. Однако большинство систем, использующих данную технологию, не являются полностью генеративными — они представляют собой гибриды, сочетающие классификацию (с предопределенными парами «запрос — ответ»), процессно-ориентированные сценарии и сгенерированные ответы, которые могут быть встроены в структуру диалога и/или вызываться как резервный механизм (например, когда классификатор не может с достаточной уверенностью определить намерение).

Генеративный ИИ также может применяться для улучшения выходных данных классификации, добавляя персонализированное приветствие или краткое резюме проблемы перед выдачей «штампованного» (предварительно составленного) ответа или запуском сценария. При правильной реализации это позволяет вовлечь пользователя в общение с чат-ботом, так как последний, повторяя проблему, демонстрирует «понимание» и «сочувствие».

Задания

Оцените разговорный ИИ, над которым вы в данный момент работаете или который сопровождаете. Задайте себе следующие вопросы:

1. Проявляет ли бот какие-либо признаки слабого понимания, например:
   - дает неправильные ответы, не вполне относящиеся к запросу или полностью с ним не связанные;
   - слишком часто использует резервный механизм или эскалацию;
   - излишне часто уточняет, казалось бы, очевидный запрос (для решений с функцией устранения неоднозначности);
   - выдает устаревшую или некорректную информацию;
   - получает отрицательные отзывы или низкие NPS-оценки.
2. Как изначально был обучен и протестирован бот? Если решение уже развернуто, измерялись ли базовые показатели?
3. Обновлялся ли бот для распознавания новых тем и выдачи актуальных и точных ответов?
4. Кто имеет право вносить изменения в решение? Документируются ли эти изменения? Отслеживается ли работа чат-бота после обновления, чтобы убедиться, что все изменения дают ожидаемый эффект?

4.2. Как измерить понимание?

Понимание в чат-боте обычно измеряется в терминах корректности полученных результатов. Для классификатора это означает способность верно распознавать намерение. Для генеративных моделей — это способность создавать корректный и релевантный выход. Существует множество методов и инструментов для

измерения того, насколько хорошо ваше решение понимает пользовательские запросы. Выбор конкретного подхода будет зависеть от используемых технологий (традиционных, генеративных или их комбинации), а также от того, на каком этапе находится решение (до развертывания или после).

4.2.1. Измерение понимания для традиционного (основанного на классификации) ИИ

Эффективность классификатора измеряется в таких терминах, как доля верных результатов, точность и полнота. *Доля верных результатов* (accuracy) — это процент верных прогнозов. *Полнота* (recall) отражает способность классификатора корректно распознать намерение. *Точность* (precision) характеризует его способность воздержаться от присвоения неправильного намерения. Более высокая доля верных результатов, как правило, означает «хорошее понимание». Чат-бот не может выдать заранее определенный ответ или запустить необходимый процессно-ориентированный сценарий, если он не понял намерение пользователя.

Оценить эффективность классификатора можно с помощью некоторых методов data science, например *K*-блочной кросс-валидации или слепого тестирования. Под *слепым тестированием* подразумевается, что тестовое высказывание отсутствует в обучающем наборе, то есть классификатор ранее его не «видел». Тестовый набор может быть либо искусственным (например, созданным с использованием данных, сгенерированных ИИ), либо репрезентативным (сформированным на основе реальных пользовательских высказываний, взятых из логов). *K*-блочные и слепые тесты позволят собрать информацию о доле всех верных результатов вашей модели, а также о ее полноте и точности. Метрики, полученные с помощью таких тестов, помогают выявить, где модель работает хорошо, а где испытывает трудности. Подробные инструкции по улучшению понимания высказываний пользователей классификатором приведены в главе 5, здесь же мы ограничимся лишь кратким обзором разных подходов к тестированию.

Измерение понимания с помощью k-блочной кросс-валидации

Если ваш чат-бот еще не развернут, *K*-блочная кросс-валидация является одним из самых простых и доступных методов измерения доли верных результатов, так как не требует дополнительных размеченных данных. Кросс-валидация использует только уже существующий обучающий набор. По сути, данный метод измеряет внутреннюю согласованность вашей разметки — высокая доля верных результатов в основном указывает на то, что обучающие примеры были сгруппированы с другими похожими примерами. Суть процесса заключается в том, что определенный процент данных изымается из обучения, что создает временный слепой тестовый набор. Оставшиеся данные используются для создания временного классификатора. Далее каждое слепое высказывание прогоняется через классификатор, и прогнозы модели оцениваются. Затем временный слепой набор возвращается в обучающий набор, и данный процесс повторяется *k* раз,

чтобы каждый пример был использован и как обучающий, и как тестовый, но никогда не в обеих ролях одновременно.

K-блочное тестирование даст вам прогноз точности вашего классификатора при условии, что данные, использованные для обучения модели, репрезентативны для тех входных данных, с которыми модель столкнется в продакшене. Однако это может создать ложное чувство уверенности, особенно если ваши обучающие данные по большей части искусственные или мало похожи на реальные пользовательские высказывания. Другой нюанс состоит в том, что при работе с небольшими датасетами высок риск получить недостоверные метрики, так как у вас недостаточно данных для выделения тестовых примеров при сохранении минимально достаточного их количества для обучения каждого намерения. По этим причинам не рекомендуется проводить *k*-блочное тестирование, если ваше решение уже развернуто в продакшене.

Измерение понимания с помощью слепых тестовых данных, сгенерированных ИИ

Получение тестовых данных с помощью генеративного ИИ осуществляется теми же методами, что и генерация обучающих данных: вы даете модели промпт для генерации разных примеров и используете их в качестве «слепого» тестового набора. Данный метод лучше всего использовать до развертывания, но также он может быть уместен на раннем этапе выхода в продакшен для восполнения пробелов в продакшен-логах.

Подобно *k*-блочному тестированию, точность измерений полностью зависит от того, насколько тестовые данные соответствуют входным данным, которые получит ваша модель в реальной эксплуатации. Однако данный подход может быть подвержен смещению и переобучению, поэтому мы рекомендуем использовать его с осторожностью и проверять сгенерированные данные с помощью продакшен-логов, как только они становятся доступны.

Измерение понимания с помощью репрезентативных слепых тестовых данных

Если ваш чат-бот уже развернут, продакшен-логи являются одним из ключевых инструментов для оценки доли верных результатов. Логи содержат реальные репрезентативные данные о том, что именно спрашивают ваши пользователи и как они формулируют свои запросы. Под «репрезентативными» мы понимаем как распределения запросов по намерениям, активируемым в системе, так и высказывания, отражающие цель пользователя, — при любой комбинации слов.

Использование продакшен-логов позволяет получить наименее смещенные тестовые данные, но также требует определенных усилий на начальном этапе, которые, тем не менее, всегда окупаются, поскольку вы создаете повторно используемый ресурс для всех последующих анализов изменений. Вам необходимо лишь извлечь выборку логов и сопоставить пользовательские входные данные (высказывания) с намерениями, возвращаемыми вашей системой. Эти данные должны быть размечены человеком, умеющим определять окончательное эталонное («золотое»)

намерение, которому принадлежит высказывание. Начальные разметки дадут вам базовый уровень точности, а данные затем будут использованы для построения *репрезентативного слепого тестового набора*, который, по сути, является списком тестовых вопросов и ответов к ним в одном файле.

Выбор наиболее подходящего метода

Затраты и усилия при выборе каждого метода полностью зависят от размера и текущего состояния вашей системы (до развертывания или после):

- *K*-блочная кросс-валидация может показаться «дешевой и простой», так как не требует ручной разметки сверх той, что уже была выполнена при обучении. Однако при запуске эксперимента k раз необходимо учитывать стоимость вызовов API. Для небольших систем она незначительна, но в случае с крупными системами речь может идти о тысячах или десятках тысяч вызовов за эксперимент.
- Генерация тестовых датасетов влечет за собой затраты на сам процесс генерации данных в дополнение к стоимости API при запуске эксперимента.
- Репрезентативный слепой тестовый набор может иметь меньшую стоимость API при запуске эксперимента (по сравнению с кросс-валидацией, если ваш тестовый набор меньше обучающего), однако стоимость ручной разметки может быть значительной. Кроме того, этот метод требует, чтобы решение было уже развернуто и взаимодействовало с реальными пользователями. Его преимущество заключается в том, что результаты эксперимента будут более значимыми, чем при тестировании с помощью кросс-валидации или сгенерированного тестового набора.

В итоге у вас есть три основных метода измерения способности классификатора понимать пользователей. Ваш выбор должен соответствовать текущему этапу разработки или развертывания, как показано на рис. 4.5.

Метод тестирования классификатора	Применим на этапе
K-блочная кросс-валидация	До развертывания
Сгенерированные тестовые данные (слепые)	До развертывания или сразу после
Репрезентативные слепые тестовые данные	После развертывания

Рис. 4.5. K-блочная кросс-валидация и сгенерированные тестовые данные подходят для ситуаций, когда репрезентативные данные недоступны. После того как решение развернуто в продакшене, репрезентативные слепые тестовые данные дают более надежное измерение способности вашего классификатора понимать пользователей

4.2.2. Измерение понимания для генеративного ИИ

Установить, соответствует ли сгенерированный ответ «хорошему пониманию», — задача непростая, и формирование автоматизированных методов

тестирования только начинается. Основная трудность связана с природой генеративного ИИ: каждый сгенерированный ответ уникален для каждого пользовательского ввода.

Прежде чем развернуть решение с использованием генеративного ИИ, необходимо определить, что именно означает «хорошее понимание». Для генеративного разговорного ИИ мы предлагаем следующие критерии:

- Сгенерированный ответ соответствует заданному формату или стилю вывода, включая:
  - позиционирование бота (цель и точка зрения его персоны[1]);
  - установленный тон и характер персоны бота.
- Сгенерированный ответ является уместным для пользовательского ввода с точки зрения длины и структуры содержания (например, нужен ли ответ в виде краткой реплики, пошаговой инструкции или развернутого текста).
- Сгенерированный ответ не содержит ложной информации (галлюцинаций).
- Сгенерированный ответ не содержит высказываний ненависти, оскорблений, ненормативной лексики, предвзятости и дискриминации.
- Сгенерированный ответ не содержит вредоносной информации, даже если она является правдивой, то есть той, что может повлечь за собой юридическую ответственность компании или нанести ущерб ее репутации (например, негативные высказывания о конкуренте или утечка конфиденциальных данных).
- Сгенерированный ответ устойчив к попыткам промпт-инъекций.
- Сгенерированный ответ является корректным и полным и либо успешно завершает сценарий, либо переводит сценарий на следующий шаг или к следующему наилучшему действию.

Если ваше решение уже развернуто, необходимо получить репрезентативную выборку логов. Вручную оцените уровень понимания вашего бота. Каждый сгенерированный ответ оценивается как правильный, достаточный или уместный в соответствии с критериями, определенными для вашего решения.

Разумеется, это трудоемкий процесс, но ваши усилия обязательно окупятся. Размеченный набор может быть использован в качестве эталонного тестового набора для будущих улучшений и позволит отслеживать влияние изменений параметров модели (например, температуры, top-p, top-k) и других настроек LLM. Данные семплы также могут быть задействованы в создании few-shot-примеров (входных данных, сопоставленных с желаемыми выходами), которые затем участвуют в промпт-инжиниринге или дообучении (тонкой настройке).

---

[1] Персона чат-бота (chatbot persona) — виртуальный персонаж, который олицетворяет тон, голос и личность виртуального помощника. — *Примеч. ред.*

4.2.3. Измерение понимания с помощью обратной связи от пользователей

Один из способов определить уровень понимания модели — встроить механизм оценки ее ответов непосредственно в пользовательский опыт, например, через опцию «палец вверх/вниз». Данный метод может использоваться как в традиционных, так и в генеративных решениях.

Здесь важно учитывать, как часто вы будете запрашивать обратную связь, и понимать, какой цели она служит. Какой именно аспект опыта должна отражать оценка: удовлетворенность или неудовлетворенность конкретным ответом (для вопросно-ответных сценариев), процессом обслуживания и его результатом (для процессно-ориентированных ботов) или же всем диалогом в целом?

Задания

1. Изучите и задокументируйте свое решение (или пересмотрите и обновите его при необходимости), уделив внимание компонентам, в наибольшей степени ответственным за понимание:
   - для классификаторов это означает проверку обучающих данных;
   - для решений, дополненных поиском и извлечением, — проверку исходных документов или URL, любых дополнительных обогащений документов и графика их загрузки, чтобы убедиться, что ваша база знаний содержит наиболее релевантную и актуальную информацию;
   - для решений с генеративным ИИ — проверку диалоговых сценариев, подразумевающих генерацию ответов, и сопоставление промптов, параметров и настроек LLM с ожидаемыми результатами.
2. Проанализируйте ваши текущие методы тестирования, если они есть. Существуют ли исторические метрики тестирования, которые можно сопоставить с текущими признаками слабого понимания?
3. Рассмотрите методы тестирования, представленные в этом разделе. Какой из них подходит для текущего этапа жизненного цикла вашей системы?

4.3. Оценка текущего состояния системы

Прежде чем разрабатывать планы по улучшению, необходимо провести оценку того, насколько точно ваше решение способно определять цели и потребности пользователей на данный момент. Характер оценки будет зависеть от используемой технологии. Классификаторы и генеративные модели выполняют совершенно разные функции и, следовательно, требуют разной оценки.

4.3.1. Оценка традиционного (основанного на классификации) ИИ-решения

Для традиционного ИИ начните с анализа обучающего набора, чтобы сориентироваться в предметной области и текущем уровне ее охвата:

- Сколько классификаторов используется в решении?
- Сколько различных намерений обрабатывает система (или каждый классификатор)?
- Насколько уникально каждое намерение?
- Пересекаются ли примеры обучения для разных намерений?
- Соответствует ли спектр тем (намерений) вашему представлению о назначении чат-бота?
- Как система обрабатывает ввод, который она не понимает?
- Какова сложность диалога? Есть ли сложные сценарии, интеграции с бэкендом или поисковые интеграции?

Полезно визуализировать объем обучающих данных классификатора в виде диаграммы. На рис. 4.6 показан пример обучающего датасета. Здесь пока не так много информации, но это даст нам основу для сравнения после того, как мы соберем тестовые данные.

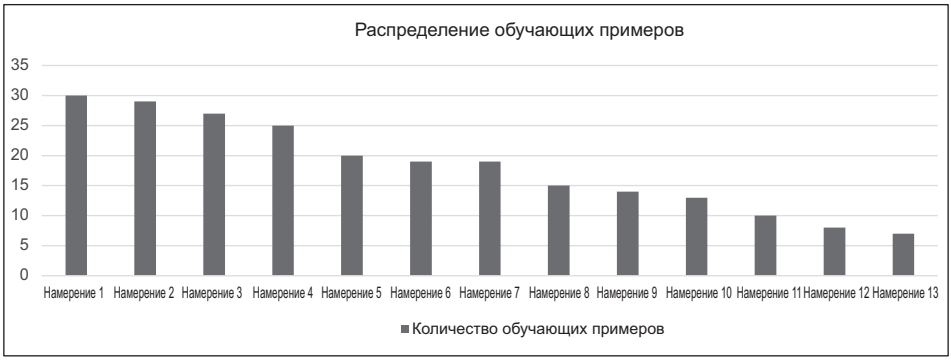


Рис. 4.6. У данного классификатора 13 намерений. Количество обучающих примеров варьируется от 7 до 30

В целом ожидается, что намерения с большим количеством обучающих примеров будут более частотными. Нам нужно, чтобы наиболее популярные темы распознавались в большинстве случаев. Намерения с большим количеством примеров могут также представлять темы с наиболее разнообразными формулировками. Однако нельзя допускать значительных диспропорций в объемах внутри обучающего набора. Например, если часть намерений обучена на сотнях примеров, а другие — всего на паре-тройке, это может привести к проблемам в работе, таким как, например, чрезмерное смещение в сторону мажоритарных классов (over-selection) — частое определение неправильного намерения из-за смещения, вызванного количеством обучающих примеров.

4.3.2. Оценка генеративного ИИ-решения

Для генеративного ИИ, как и для традиционного, необходимо хорошо понимать предметную область и охват, в пределах которых работает ваш бот. Однако вместо классификаций вводов вам нужно установить источники данных, из которых модель будет извлекать информацию при генерации ответов. Использует ли ваша система генеративный ИИ для формирования ответов или реплик? Если да, ответьте на следующие вопросы:

- Генерируются ли ответы на каждый пользовательский ввод?
- Является ли генерация ответов резервным механизмом для классификатора?
- Используется ли генерация текста для дополнения заранее заготовленного ответа классификатора?
- Использует ли ваше решение более одной LLM, например, разные модели для разных типов ответов, поддержку нескольких языков и т. д.?
- Использует ли ваше решение промпт-инжиниринг, настройу промптов, дообучение или другие индивидуальные настройки? Задокументировано ли это где-либо вместе с целями, ради которых была внедрена каждая настройка?
- Использует ли ваше решение RAG? Если да, то на основе какого источника данных? Как часто он обновляется? Содержит ли он дополнительные обогащения данных?

Задания

1. Оцените свою систему, используя критерии, описанные в этом разделе (в зависимости от того, какой тип ИИ вы используете).
2. После проведения первоначальной оценки обязательно задокументируйте текущее состояние решения — это будет вашей базовой конфигурацией системы.
3. Следуя рекомендациям по улучшению и примерам, приведенным в этой книге, зафиксируйте все внесенные изменения таким образом, чтобы можно было сопоставить обновления с последующими результатами анализа эффективности.

4.4. Извлечение тестовых данных из логов и их подготовка

В оставшейся части главы предположим, что у вас уже есть функционирующая система и доступ к ее логам. Мы покажем, как извлечь и подготовить эти данные для создания ресурса, который затем можно будет использовать для анализа текущего состояния бота (а также для валидации будущих изменений).

Для создания тестового набора на основе продакшен-логов потребуется некоторая предварительная работа. На рис. 4.7 приведены основные задачи, связанные с подготовкой данных для тестирования (или обучения).

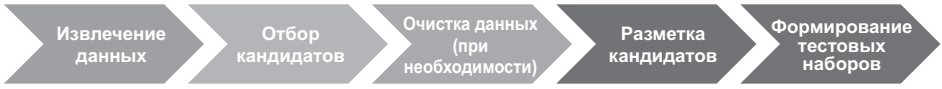


Рис. 4.7. После извлечения данных каждое высказывание должно быть сначала распределено по категориям для выделения потенциальных кандидатов; это позволит отделить хорошие и пригодные для тестирования данные от плохих или нерелевантных пользовательских вводов. Данные также могут потребовать очистки, например от персональных данных (PII). После этого данные необходимо разметить (для классификаторов их нужно разметить с указанием подходящего намерения, для генеративного ИИ — сопоставить с эталонным ответом). Наконец, данные собираются в один или несколько наборов, которые будут использоваться вашим инструментом тестирования

4.4.1. Извлечение продакшен-логов

В идеале у вас должен быть доступ к продакшен-логам за год и более. Так ваш тестовый набор будет содержать действительно репрезентативную выборку тем, с которыми сталкивается ваш бот в разные сезоны и при различных событиях, влияющих на вашу отрасль. Соберите семплы логов за разные недели или месяцы в течение года. Если ваше решение относительно новое, придется чаще обновлять тестовые наборы в течение первых 12–18 месяцев работы системы.

После того как вы получили часть продакшен-логов, проще всего преобразовать их данные в файл CSV или Excel (если это еще не сделано). На наш взгляд, полезнее всего преобразовать данные в такой формат, где одна строка соответствует одному шагу диалога (ввод пользователя и вывод бота), а строки сгруппированы по ID диалога. В зависимости от выбранного периода, числа пользователей, а также сложности и назначения бота ваш файл может содержать всего несколько сотен строк данных или же более 100 000.

Один из простых способов сократить объем данных до относительно управляемого — выбрать первое высказывание пользователя в каждом диалоге. И хотя такой метод подходит не всегда, рис. 4.8 показывает, что зачастую он позволяет извлечь полезные данные из логов. В диалоге на естественном языке пользователи, как правило, формулируют свою наиболее важную потребность еще на начальном этапе. Если средняя продолжительность диалога составляет десять шагов, то журнал с 100 000 строк необработанных данных может быть сокращен примерно до 10 000 строк для анализа, а дополнительная дедупликация сократит этот объем еще на несколько тысяч. Так мы получим набор, с которым удобно работать, и при этом он содержит разнообразные примеры, которые можно использовать для тестирования системы.

ID диалога	Шаг	Ввод пользователя	Ответ чат-бота
1001	1	Я хочу открыть счет.	Какой счет вы хотите открыть?
1001	2	Сберегательный.	Какую сумму вы хотите внести на ваш новый сберегательный счет?
1001	3	$500	Хорошо, сейчас откроем для вас новый сберегательный счет.
1002	1	У меня украли банковскую карту.	Я могу заблокировать вашу карту. У вас есть подозрительные списания?
1002	2	Да.	Вы знаете, когда у вас украли карту?
1002	3	Наверное, вчера днем.	Хорошо, я отправлю уведомление о мошенничестве и выпущу для вас новую карту.
1003	1	Какой у меня баланс счета?	Я могу посмотреть ваш баланс. Введите номер счета.
1003	2	123456791	Извините, не удалось найти счет с таким номером. Хотите попробовать снова?
1003	3	Да.	Введите номер счета.
1003	4	123456719	Баланс вашего счета составляет $547. Хотите узнать что-либо еще?
1003	5	Да, у меня есть просроченные комиссии?	У вас есть задолженность по комиссии в размере $42.
1004	1	Здравствуйте.	Здравствуйте. Чем я могу вам помочь?
1004	2	Мне нужна копия банковской выписки.	С радостью вам с этим помогу. За какой период вам нужна выписка?
1004	3	За прошлый месяц.	Банковская выписки за прошлый месяц отправлена на вашу электронную почту.
1005	1	Соедините меня со службой поддержки.	Чтобы я направил вас к правильному специалисту, пожалуйста, уточните вашу проблему.
1005	2	Мне нужно сообщить о краже кредитной карты.	Я могу заблокировать вашу карту. У вас есть подозрительные списания?

Рис. 4.8. Необработанные чат-логи показывают, что основная цель пользователя часто обозначается еще на первом шаге диалога, но иногда она появляется в виде дополнительного запроса далее в беседе или после обмена приветствиями. Она также может следовать за отказом от взаимодействия. Выбор первой строки, как правило, уже дает достаточно полезных данных, сокращая время, которое разметчикам приходится тратить на сортировку незначимых реплик, таких как клики по кнопкам, типовые ответы и PII или другая персонализированная информация. (Структура ваших логов может отличаться в зависимости от используемого инструмента.)

4.4.2. Рекомендации по отбору кандидатных тестовых высказываний

Каким бы способом вы ни получали и ни обрабатывали свои логи, следующая задача — выделить потенциальных кандидатов для слепого тестирования. Будем воспринимать это как своеобразный «первичный отсев»: для начала нам достаточно лишь определить, может ли реплика быть *потенциально* использована. Мы также рекомендуем не зацикливаться на анализе: если не получается принять решение по конкретной реплике в течение минуты или около того, лучше пропустить ее и двигаться дальше. (Если у вас возникают какие-то сомнения или чувствуется повторяющийся паттерн, пометьте реплику для повторного рассмотрения позже и продолжайте работу.) Для отбора потенциальных кандидатов

из продакшен-логов мы используем следующие критерии, сопровождая их при необходимости другими инструкциями:

- Является ли высказывание неразборчивым?
- Связано ли высказывание с предметной областью?
- Является ли высказывание многозначным?
- Содержит ли высказывание несколько намерений?
- Относится ли высказывание к предметной области, но при этом выходит за рамки сценария?
- Выражает ли высказывание цель, относящуюся к предметной области и попадающую в рамки сценария?

Рассмотрим каждую из категорий.

Является ли высказывание неразборчивым?

Возможно, по клавиатуре прошелся кот или пользователь просто случайно нажал клавиши. Или же технология распознавания речи некорректно транскрибировала вопрос звонящего и выдала какую-то бессмыслицу. Голосовые системы также могут захватывать фоновый шум и посторонние разговоры, особенно если они неправильно откалиброваны для конкретной среды. В вашем файле может встретиться множество пользовательских вводов, которые попросту не имеют смысла.

Примеры непонятных или нерелевантных высказываний:

- «делает он школа» (бессвязная фраза — если она получена из голосового ввода, скорее всего, это ошибка транскрипции речи);
- «она не... она сказала, что их четыре, а дала только одну... да, можешь сделать так, я сейчас в самолете, проверю, когда буду в офисе» (похоже на транскрипцию какого-то фонового разговора);
- «кдвпжтхкалитофрния линзенсц» (опечатки настолько серьезные, что понять смысл реплики невозможно).

Такие строки можно смело исключать из набора для слепого тестирования. Однако любые повторяющиеся паттерны, например систематические ошибки в транскрипции речи, следует отметить и затем проанализировать или же передать ответственной за них команде.

Связано ли высказывание с предметной областью?

Иногда могут встречаться запросы, которые понятны, но совершенно не относятся к тематике или назначению бота. Например, если ваше решение предназначено для помощи клиентам энергоснабжающей компании в управлении счетом и услугами, то вопросы о поп-культуре, случайно появившиеся в логах, можно исключить.

И хотя вы можете настроить ваш бот так, чтобы он перенаправлял нераспознанные темы в LLM, такие реплики не должны включаться в тестовый набор классификатора, так как им невозможно назначить эталонное намерение. Подобные высказывания можно использовать в негативном тестировании, чтобы проверить, корректно ли система распознает ситуации, когда не нужно давать ответ.

Является ли высказывание многозначным?

Возможно, вам встретится какое-то слово или короткая фраза, относящаяся к предметной области, но не выражающая конкретной цели. Например, если пользователь банковского чат-бота вводит всего лишь слово «счет», что он имеет в виду? Он хочет открыть счет? Закрыть? Проверить баланс? Кто знает?

К многозначным высказываниям также относятся ответы, сгенерированные при нажатии кнопки или в рамках сценария сбора информации. (Если до этого вы выбирали каждое первое высказывание в беседе, то таких примеров вы могли и не встретить.) Как правило, они не представляют ценности для тестирования эффективности классификатора, если только не совпадают с намерением внутри сценария. Включайте такие высказывания в выборку, только если они уместны.

Примеры многозначных высказываний:

- «водительские права» (может, и относится к предметной области, но четкой цели нет);
- «первое» (ссылка на контекстную информацию, которая, по-видимому, была предоставлена ранее, но сама по себе ничего не означает);
- «2» (может обозначать выбор кнопки, опции телефонного меню либо количество или какую-то величину, переданную в ответ на предыдущий вопрос).

В большинстве случаев такие высказывания не следует использовать в тестировании точности классификатора, так как они, скорее всего, не будут четко соответствовать какому-то одному намерению, а будут затрагивать сразу несколько. Тем не менее они не совсем бесполезны и пригодятся в анализе частоты использования пользователями различных формулировок. Посмотрите, насколько корректно такие функции вашего чат-бота, как, например, механизм разрешения лексической многозначности или уточняющие вопросы, обрабатывают подобные случаи.

Содержит ли высказывание несколько намерений?

Большинство чат-ботов на основе классификаторов работают эффективнее, когда им передается одна цель за раз. Высказывания, содержащие несколько разных целей, необходимо исключать из тестового набора для оценки точности классификатора, так как невозможно однозначно присвоить им «правильное» намерение.

Исключение из этого правила — наличие у бота механизма разрешения лексической многозначности, позволяющего уточнить основную цель пользователя путем отображения N наиболее вероятных намерений, определенных классификатором. В подобных решениях можно протестировать высказывания с несколькими намерениями, чтобы убедиться, что все они корректно предлагаются в качестве вариантов для выбора.

Примеры высказываний с несколькими намерениями:

- «У вас есть вакцина от COVID? Как записаться на прием?» (Две цели: 1) узнать о наличии вакцины; 2) записаться на прием.)
- «Я хочу обновить адрес на водительских правах и узнать, как получить водительские права для коммерческих перевозок». (Две цели: 1) обновить адрес, 2) узнать требования для получения водительских прав для коммерческой деятельности.)
- «У меня 95 000 бонусных баллов. У них есть срок действия? Сколько еще баллов нужно до платинового статуса? Можно ли их купить?» (Три цели: 1) узнать, истекает ли срок действия бонусных баллов; 2) узнать, сколько баллов необходимо до следующего уровня; 3) получить информацию о покупке баллов для повышения уровня.)
- «Я хочу поговорить с оператором и сообщить об угоне» — очень распространенный случай. Пользователь часто совмещает запрос на подключение к оператору с основной целью. Если оба намерения есть в обучающем наборе классификатора, такие высказывания можно обработать двумя способами:
  - исключить их, если невозможно назначить единственное «правильное» намерение;
  - включить их и отметить «предпочтительное» намерение. (Например, обслуживание ботом может быть более предпочтительным, если приоритетом является удержание пользователей, а конкурирующее намерение ведет к эскалации.)

Как и в случае с многозначными высказываниями, подобные примеры лучше отложить и анализировать отдельно для лучшего понимания поведения пользователей. Если такие ситуации встречаются часто, стоит разработать дополнительные стратегии их обработки. Если пользователи склонны объединять связанные запросы в одном высказывании, ответы в соответствующих намерениях можно обновить таким образом, чтобы они предусматривали все потребности. В первом примере — «У вас есть вакцина от COVID? Как записаться на прием?» — ответ о наличии вакцины может сразу содержать ссылку для записи.

Генеративный ИИ обычно гораздо лучше справляется с несколькими намерениями, чем классификаторы, поэтому подобные высказывания можно включать в тестовый набор, если ваше решение обладает соответствующими возможностями.

Пара слов о работе с несколькими намерениями в моделях-классификаторах

Мы не раз сталкивались с масштабными и сложными попытками обработать несколько намерений программно. Обычно это предполагает сбор N лучших намерений и их сохранение в контексте, а затем дополнительную логику для обработки последующих тем после ответа на первый запрос. В большинстве случаев результатом становится чрезмерно усложненное, трудно масштабируемое или попросту неэффективное решение. У такого подхода есть и другой серьезный недостаток: подобная логика не может точно различить высказывания, действительно содержащие несколько целей, и высказывания с одной целью, которая активировала несколько намерений.

Многие современные фреймворки для чат-ботов предоставляют автоматическое разрешение многозначности тем (например: «Вы имели в виду: [Намерение 1], [Намерение 2] или [Намерение 3]?»). Здесь мы порекомендуем одно — позволить встроенному механизму выполнять свою работу. Иногда это означает, что пользователю придется формулировать свои вопросы или цели по одному. Частота и значимость таких случаев обычно не оправдывают затрат на создание и поддержку кастомной логики для обработки множественных намерений в чат-боте на основе классификатора.

Относится ли высказывание к предметной области, но при этом выходит за рамки сценария?

Скорее всего, вы встретите высказывания, в которых выражена одна четкая цель, связанная с предметной областью, но текущая система не может ее обработать. Например, банковский чат-бот может сообщать пользователю баланс счета, но при этом не распознавать запросы о процентных ставках. Чат-бот авиакомпании может быть обучен политике организации, но не владеть информацией о правилах досмотра в аэропорту.

С точки зрения пользователя такие вопросы вполне обоснованны, и пробелы в тематическом охвате часто приводят к разочарованию. Это особенно актуально, если у системы нет генеративного ИИ или резервного поиска. И даже если бот отвечает: «Извините, я вас не понимаю. Переформулируйте вопрос», — никакая другая формулировка не приведет к необходимому результату. Что же в таком случае делать?

Если у классификатора нет обученных намерений для обработки подобных запросов, все «пробелы» в знаниях необходимо отложить на будущее. В дальнейшем их можно сгруппировать по темам или категориям, но пока они должны быть исключены из тестового набора, так как им нельзя присвоить намерение. Отслеживайте подобные темы и при необходимости добавляйте их в план улучшений.

Аналогичным образом, если ваша генеративная система пока не умеет отвечать на какие-то вопросы (например, если репозиторий документов в RAG-решении не содержит материалов по данной теме), пока отложите их, но отслеживайте их частотность.

Высказывание выражает цель, относящуюся к предметной области и попадающую в рамки сценария

Отлично! Вопросы или запросы, которые соответствуют вашему решению и предметной области, должны быть включены в эталонный тестовый набор.

4.4.3. Подготовка и очистка данных для использования в итеративных улучшениях

Если вы никогда ранее не видели продакшен-логи чат-бота, вас удивит, насколько они «грязные». В них встречается масса неправильных или неформальных предложений, опечаток и ошибок (в текстовом канале), ошибок распознавания речи (в голосовом канале), а также различные персональные данные (PII). Вот как мы рекомендуем с этим работать.

Неправильное или неформальное построение предложений

В большинстве случаев можете оставить все как есть. Люди очень по-разному выражают свои мысли. Пользователь может не знать, как именно правильно сформулировать то, что ему нужно, особенно при общении с машиной. Если цель можно определить, то высказывание является репрезентативным примером и должно быть в целом сохранено без изменений.

Опечатки и ошибки в словах

Если опечатка или ошибка существенно не изменяет смысл всей фразы, оставляйте как есть. Часто встречающиеся орфографические ошибки отражают реальный стиль коммуникации пользователей. Классификатор должен уметь корректно отвечать как на вопрос «В чем разница между задолженностью по кредиту и суммой основного долга?», так и на вопрос «В чем разница между зодолженостью по кредиту и суммой основного долга?».

Регистр и пунктуация, как правило, игнорируются классификатором, но все равно стоит в этом убедиться, проведя пару тестов на используемой вами платформе.

Ошибки распознавания речи

Если ваше решение использует технологию преобразования речи в текст (automated speech recognition, ASR), вы не встретите опечаток, но скорее всего, увидите какие-то неожиданные слова, являющиеся результатом ошибки распознавания. Первое средство борьбы с этим — обучение моделей распознавания речи, если это возможно. Так как основа работы классификатора — это текст, важно получить максимально точное представление высказывания пользователя до его обработки текстовым классификатором.

Если модели распознавания речи продолжают систематически искажать значимые для вашей предметной области слова, включайте такие случаи в тестовый набор (и в итоге, скорее всего, стоит добавить их в обучающие данные).

Например, в одной энергетической компании мы регулярно сталкивались с тем, что частотное слово «жилой» распознавалось как «живой». Так как обновление модели заняло бы много времени и привело бы к снижению удержания звонков, оптимальным решением стало добавление «живой» в словарь синонимов чатбота. Другой пример — искажение «VIN» как «BIN» в сценарии, где необходимо было распознавать идентификационный номер транспортного средства (vehicle identification number). Поэтому мы добавили в обучающие данные обе аббревиатуры, а также сохранили подобные ошибки распознавания для дальнейшего тестирования.

Персональные данные

В логах можно встретить самые разные виды PII: имена, номера телефонов, почтовые или электронные адреса, номера социального страхования, номера счетов и т. д. Такие данные не должны использоваться в обучающих или тестовых наборах. В идеале подобная информация в логах должна быть замаскирована, но механизмы, предназначенные для этого, не всегда срабатывают. Если у вашего решения есть функция маскировки PII, заменяйте реальные данные символами того же формата (например, ###-###-#### вместо номера телефона). Если такой функции нет, либо полностью удаляйте PII, либо заменяйте их на очевидно вымышленные, например «имяпользователя@email.com».

4.4.4. Процесс разметки

После того как вы отобрали данные, содержащие четко выраженную цель в рамках вашей предметной области (и при необходимости очистили их), нужно разметить их в соответствии с задачей.

Разметка эталонного тестового набора для традиционного ИИ (классификатора)

Разметка тестового набора для классификатора предполагает маркировку каждого высказывания соответствующим намерением. Задача на практике оказывается сложнее, чем звучит, и именно на нее уходит больше всего времени при подготовке тестового набора.

Достаточно легко определить и отсеять неразборчивый или многозначный ввод пользователя. Однако после того, как вы установили, что высказывание относится к предметной области, требуется немало усилий, чтобы присвоить ему правильное намерение. Разметчик (или их команда) должен хорошо знать текущие обучающие данные. В процессе обязательно вскроются случаи пересечения намерений, и разметчики столкнутся с вопросом, как правильно маркировать то или иное высказывание.

К задаче разметки можно подойти по-разному. Иногда этим занимается один человек, в других же случаях за работу берется целая команда, состоящая из приверженцев стратегии «разделяй и властвуй». По нашему опыту, это лишь

приводит к дополнительным проблемам, на решение которых уходит много времени.

В идеале все разметчики должны работать сообща, вместе разбирая каждое высказывание. Это обеспечит лучшее понимание назначения каждого намерения. Всем разметчикам необходимо понимать критерии разграничения намерений, которые могут не только использовать схожие ключевые слова, но также иметь разные цели. Альтернативный подход — разметка одних и тех же данных несколькими специалистами (или хотя бы части пересекающихся данных) с последующим сравнением и анализом расхождений.

Другой прием, которым стоит воспользоваться, если ваши логи содержат намерения, установленные для каждого высказывания в ходе работы системы, — просмотреть записи и проверить, верно ли они назначены. В таком случае вам останется разметить только те высказывания, где прогноз оказался ошибочным.

Данный процесс может занять от нескольких часов до нескольких дней, требует значительной концентрации внимания и утомляет глаза. При первом просмотре разметчики должны стремиться вынести суждение и скорее двигаться дальше. Если на разметку какого-то одного высказывания уходит более 60 секунд, его стоит пропустить и вернуться к нему позже. Также важно делать перерывы каждые час-два. Небольшой отдых помогает взглянуть на задачу свежим взглядом.

Может ли все сделать LLM?

Если вы строите свой первый классификатор, то вполне можете прогонять высказывания через LLM в качестве первого шага разметки или классификации данных. Однако если у вас уже есть продакшен-логи, то никакого смысла в использовании отдельной классификационной LLM нет, так как в любом случае потребуется участие человека для проверки результатов разметки.

В результате разметки тестового набора у вас появится эталонная выборка, размеченная человеком. В зависимости от конкретного варианта использования она может включать от нескольких сотен до нескольких тысяч высказываний и позволит вам получить информацию о текущей точности вашего классификатора. Кроме того, вы сможете использовать ее для улучшения системы.

Последнее, что необходимо сделать, — преобразовать данные в файл, который может быть обработан вашим инструментом тестирования и который затем можно будет использовать для измерения результатов обновлений. Формат зависит от выбранного вами инструмента, но, как правило, это будет текстовый или CSV-файл, содержащий строку для каждого тестового высказывания в одном столбце и эталонное намерение в другом. В табл. 4.2 приведен пример того, как может выглядеть ваш тестовый набор.

Таблица 4.2. Пример тестового набора с одной парой высказывание/намерение в строке

Высказывание	Эталонное намерение
Я хочу поговорить с человеком.	`Request_Agent`
Можно поговорить с менеджером?	`Request_Agent`
Соедините со службой поддержки.	`Request_Agent`
Вы работаете по воскресеньям?	`Office_Hours`
Во сколько вы открываетесь?	`Office_Hours`
Когда вы закрываетесь?	`Office_Hours`
Какой у вас график работы в выходные?	`Office_Hours`

Разметка эталонного тестового набора для генеративного ИИ

Создание тестового набора для анализа качества работы генеративного ИИ предполагает оценку качества ответа, выданного системой (если у вас есть продакшен-логи), и его обновление или замену на лучший ответ в соответствии с метриками, которые вы заранее определили для своего решения. Экспертам в предметной области необходимо будет просмотреть каждый пример вывода, чтобы убедиться, что он фактически корректный и полный, а также соответствует политике компании и отражает цель персоны виртуального агента.

После проверки выводов у вас сформируется набор высказываний, сопоставленных с эталонным ответом или реакцией. Такой ресурс даст вам оперативную информацию о качестве сгенерированных ответов. Его также можно использовать для настройки промптов и конфигураций LLM.

Последнее, что необходимо сделать, — преобразовать данные в файл, который может быть обработан вашим инструментом тестирования. Формат зависит от выбранного вами инструмента, но, как правило, это текстовый или CSV-файл, содержащий строку для каждого тестового высказывания в одном столбце и эталонный ответ в другом. В табл. 4.3 показан пример такого набора.

Таблица 4.3. Пример тестового набора с одной парой высказывание/ответ в строке

Высказывание	Эталонный ответ
Можно ли сдать в багаж сноуборд?	Вы можете сдать в багаж один комплект сноубордического снаряжения. Комплект должен находиться в одной сумке и может включать до двух сноубордов и одну сумку для ботинок. Если вес комплекта превышает 23 кг, необходимо оплатить сбор за перевес
Сколько ждать возврата средств?	Возвраты по кредитным картам обрабатываются в течение пяти рабочих дней с момента запроса. Все остальные возвраты обрабатываются в течение 20 рабочих дней с момента запроса

Задания

1. Извлеките данные из ваших логов и определите кандидатов для тестовых высказываний.
2. При необходимости очистите данные от PII.
3. Оцените прогнозы классификатора или сгенерированные моделью ответы. Полученные результаты — это базовые показатели эффективности вашей системы.
4. Присвойте каждому высказыванию эталонное намерение или эталонный ответ.
5. Сохраните файл в формате, подходящем для вашего инструмента тестирования.

4.5. Что говорят данные?

Если ваши логи содержат исходное предсказание намерения или сгенерированный ответ, у вас уже есть все необходимое для вычисления доли верных результатов (accuracy) вашей текущей системы. (Разделите количество правильных предсказаний или ответов на общее количество проверенных кандидатов.) Ваша разметка покажет диапазон и частоту тем, с которыми пользователи обращаются к вашему чат-боту.

4.5.1. Анализ размеченных логов для традиционного ИИ (классификатора)

В случае систем на основе классификаторов особенно интересен анализ распределения намерений по их частотности. Соотносится ли оно с количеством обучающих примеров для каждого намерения? На рис. 4.9 показано идеальное, достаточно сбалансированное распределение обучающих примеров по сравнению с реальной частотностью намерений.

Разрыв между количеством обучающих примеров и фактическими упоминаниями в логах сам по себе не свидетельствует о каких-то проблемах, но может подсказать вам способы повысить точность бота. На рис. 4.10 показан пример размеченных высказываний, которые совершенно не соответствуют тому, как система была обучена.

Также стоит проанализировать количество высказываний, которые относились к предметной области, но выходили за рамки охвата бота. (Они были отмечены еще на этапе подготовки данных, описанном в разделе 4.1.4.) Есть ли спрос на темы, которым классификатор пока еще не обучен? Несоответствие между задаваемыми вопросами и умением классификатора их распознавать приводит к проблеме слабого понимания ботом.

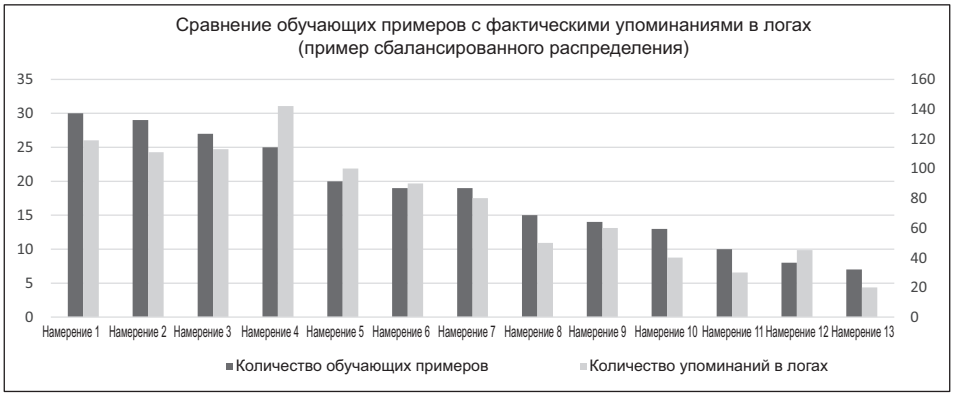


Рис. 4.9. Темные столбцы демонстрируют количество обучающих примеров в системе. Светлые столбцы показывают количество размеченных высказываний для каждого намерения. Если ваша диаграмма следует схожему паттерну, приоритеты обучения, скорее всего, соответствуют требованиям системы

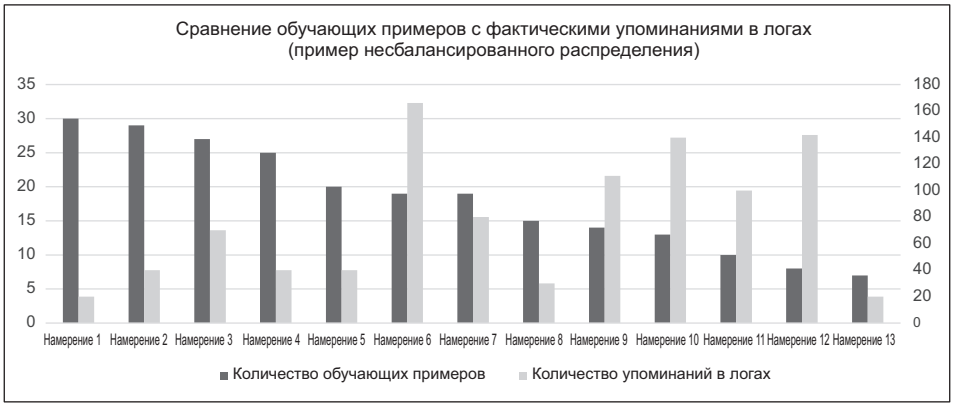


Рис. 4.10. Количество обучающих примеров (темные столбцы) показывает значительный разрыв по многим намерениям по сравнению с размеченными данными логов (светлые столбцы). Без показателей точности по каждому намерению мы не можем сразу определить, оказывает ли этот разрыв отрицательное влияние на систему. Однако мы можем сделать некоторые выводы, например: 1) первые пять намерений не так важны для наших пользователей, как мы предполагали; и 2) намерения с наибольшим упоминанием (светлые столбцы в намерениях 6, 10, 11 и 12) оказались для пользователей важнее, чем прогнозировалось

Определение доли верных результатов дало нам общее представление об уровне понимания системой пользователя. Следующий шаг — провести более детальный анализ по конкретным намерениям. Можно начать с наименее успешных

намерений, которые при этом являются высокочастотными/ценными в вашем решении. Процесс улучшения понимания классификатора мы подробнее рассмотрим в главе 5.

4.5.2. Анализ размеченных логов для генеративного ИИ

В случае с генеративным ИИ размеченные логи дают представление о разнообразии вопросов и запросов от пользователей. В процессе разметки вы могли обнаружить пробелы в охвате предметной области, а также понять, как с помощью лучшего промпт-инжиниринга и дообучения можно повысить качество генерируемых ответов. Если ваше решение использует RAG, вы можете сопоставить качество ответов с документами в репозитории.

Определение доли верных результатов дало нам представление об уровне понимания системой. Процесс улучшения понимания генеративным ИИ мы подробнее рассмотрим в главе 6.

4.5.3. Необходимость итеративного улучшения

На данном этапе у вас есть все данные, необходимые для начала планирования улучшений. Результаты анализа эффективности будут служить дорожной картой исправлений. Помните, что это итеративный процесс: вы вносите изменения, а затем смотрите, имели ли они положительный, нейтральный или отрицательный эффект на понимание системой.

Также важно отметить, что ваш слепой или эталонный тестовый набор необходимо будет обновлять в течение всего жизненного цикла решения. Напомним, что одной из причин снижения точности чат-бота является появление новой информации. Вот несколько примеров, с которыми мы сталкивались:

- Глобальная пандемия COVID-19, которая повлияла на формат работы, взаимодействие в общественных местах и общение с семьей практически каждого человека.
- Принятие нового закона, в результате чего к государственным организациям появился ряд новых вопросов.
- Появление новых товаров на рынке или отзыв продукции.
- После утечки данных в компании чат-бот оказался завален вопросами вроде «Мои данные в безопасности?» и «Расскажите об утечке данных».

Будьте готовы регулярно просматривать логи. Частота проверок зависит от размера системы — в первое время после запуска можно проводить их ежедневно, затем еженедельно, ежемесячно и ежеквартально. При этом не забывайте обновлять тестовые наборы в зависимости от вносимых изменений:

- Если в систему добавляются новые намерения, в тестовый набор необходимо добавить новые высказывания.
- Если какие-то намерения объединяются или разделяются, соответствующие изменения должны быть внесены в тестовый набор.
- Если в базу знаний, на которую ссылается ваша генеративная модель, добавляются новые данные, тестовый набор должен быть готов к их валидации.
- Если в систему добавляются новые сценарии LLM или модификации промптов, это также должно быть отражено в тестовом наборе.

Задания

1. Просмотрите размеченные данные и проанализируйте результаты. Есть ли области со слабым пониманием?
   - Если да, какова, на ваш взгляд, причина?
   - А может, причин несколько?
2. Как бы вы расставили приоритеты в возможных изменениях, необходимых для улучшения понимания системой?

Итоги

- Чат-боты хорошо понимают пользователей, если способны определить, чего они хотят, и предоставить удовлетворительный ответ или помочь им продвинуться к цели.
- Понимание традиционным ИИ опирается как минимум на два механизма: корректную классификацию намерения и способность выдать результат на основе этой классификации. (Дополнительные механизмы, такие как распознавание сущностей или контекст, могут модифицировать или персонализировать вывод.)
- Понимание генеративным ИИ основывается на высказывании и сопутствующем ему промпте и на умении создать ответ, соответствующий вопросу или цели пользователя.
- Низкий уровень понимания системой снижает ее бизнес-ценность и проявляется в том, что чат-бот возвращает неверные ответы или вовсе не отвечает.
- Невозможно оценить эффективность чат-бота без предварительного сбора данных.
- Понимание чат-ботом обычно выражается в таком показателе, как доля верных результатов (accuracy), — доля случаев, когда система выдает правильный ответ или выполняет нужное действие.

- Существует множество инструментов и методов для анализа понимания системой. Некоторые из них зависят от типа ИИ и/или текущего этапа — перед развертыванием или после него.
- Репрезентативный эталонный тестовый набор, составленный на основе реальных пользовательских высказываний (продакшен-логов), используется для анализа базовой эффективности бота и может быть преобразован в повторно используемый ресурс для измерения эффекта будущих изменений.
- Следует помнить о необходимости мониторинга и повторного обучения системы на протяжении всего жизненного цикла чат-бота.
- Обновления обучения могут потребовать соответствующих обновлений слепого тестового набора.

5 Улучшаем понимание традиционного ИИ

В этой главе

- ✓ Типы ошибок, которые может допускать классификатор
- ✓ Базовый уровень текущей эффективности классификатора
- ✓ Методы data science для выявления и приоритизации улучшений
- ✓ Интеграция сгенерированного контента в традиционный ИИ для повышения уровня понимания

Мы продемонстрируем структурированный итеративный подход к улучшению понимания разговорных систем, основанных на классификации. Эта глава опирается на понятия, представленные в предыдущей, и использует результаты, полученные в финальном задании раздела 4.4 (где вы создали тестовый набор с назначенным эталонным намерением для каждого высказывания в формате, подходящем для своего инструмента тестирования). Далее рассмотрим, как большие языковые модели могут дополнять ответы, основанные на намерениях, тем самым обеспечивая более качественный пользовательский опыт. (Если вас интересуют только методы улучшения с помощью генеративного ИИ, можете сразу перейти к следующей главе.)

Мы начнем свою работу с создания плана улучшений и определения типов ошибок, которые может допускать ваш классификатор. Затем пройдем через семь

циклов улучшений, устраняя проблемы, с которыми вы можете столкнуться в собственном текстовом классификаторе. И хотя для извлечения определенных выводов из данных в этой главе мы будем использовать методы data science, вам вовсе не обязательно быть для этого специалистом в этой области.

5.1. План улучшений

Если вы создали слепой тестовый набор на основе выборки из продакшен-логов, у вас должен быть надежный тестовый набор с «репрезентативным распределением». Это означает, что наиболее часто задаваемые пользователями темы представлены в тестовых данных в соответствующем объеме. Это станет ключевым фактором при расстановке приоритетов среди проблем, выявленных в ходе тестов.

Если вы работаете с результатами *k*-блочного теста (который мы рассматривали в главе 4), то не сможете точно определить, какие темы наиболее важны, поэтому логично начать с тех, в которых доля верных результатов наименьшая.

В любом случае пришло время углубиться в результаты тестирования. План улучшений всегда начинается с выявления наиболее проблемных областей в чат-боте.

5.1.1. Определение проблемных паттернов в неправильно интерпретированных высказываниях

Первым показателем, который привлечет ваше внимание, станет доля верных результатов на этапе тестирования. Это как если бы вы получили назад свою контрольную по орфографии или математике и увидели на странице красные пометки. Если в тесте было 100 вопросов и вы правильно ответили на 79 из них, доля верных результатов (accuracy) составит 79 %. Для классификаторов данная метрика полезна с целью быстрого «обзорного» понимания состояния модели, но она не дает полного представления о происходящем и о том, с чего же начать улучшения. Для этого необходимо понимать возможные исходы и типы ошибок, которые может допускать ваш классификатор. Они тесно связаны с такими показателями, как полнота, точность и оценка F1.

Краткое объяснение, что такое полнота, точность и оценка F1

В главе 4 мы описали *полноту* (recall) как способность классификатора правильно определить намерение, а *точность* (precision) — как способность воздержаться от определения неправильного намерения. Эти понятия можно рассматривать с точки зрения положительных и отрицательных прогнозов. Для каждого высказывания, которое мы проверяем на модели, существует четыре возможных исхода, и при этом они не являются взаимоисключающими, то есть каждый прогноз одновременно попадает в две или три категории.

На рис. 5.1 показана матрица ошибок, визуализирующая все возможные исходы:

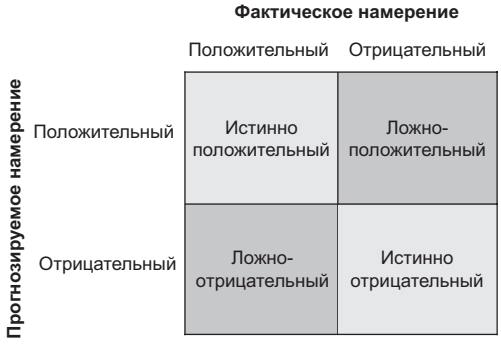


Рис. 5.1. В матрице ошибок 2 × 2 возможные исходы формируются путем сравнения прогнозируемых намерений с фактическими

- *Истинно положительный (true positive)* — прогноз совпадает с правильным намерением.
- *Истинно отрицательный (true negative)* — прогноз не совпадает с неправильным намерением.
- *Ложноположительный (false positive)* — прогноз совпадает с неправильным намерением.
- *Ложноотрицательный (false negative)* — прогноз не совпадает с правильным намерением.

Первой метрикой, которая нас интересует, является полнота намерений. Для этого нам необходимо знать количество истинно положительных и ложноотрицательных прогнозов. Намерение, возвращающее ложноотрицательные прогнозы, допускает ошибку недовыборки (under-selection). Вычисление этой метрики для каждого намерения напоминает такой показатель, как доля верных результатов. Например, если в нашем тесте было пять вопросов для намерения `#Request_Agent` и классификатор дал правильный ответ четыре раза, полнота данного намерения составит 80 %:

Полнота = Истинно положительные / (Истинно положительные +
+ Ложноотрицательные).

Следующая метрика, позволяющая разобраться в работе классификатора, — это точность. Она измеряет, насколько хорошо классификатор воздерживается от ложноположительных прогнозов. Намерение, возвращающее ложноположительные прогнозы, допускает ошибку перевыборки (over-selection), пример которой показан в последних двух строках табл. 5.1:

Точность = Истинно положительные / (Истинно положительные +
+ Ложноположительные).

Таблица 5.1. Результаты теста

Высказывание	Правильное намерение	Прогнозируемое намерение	Верно
Служба поддержки.	Request_Agent	Request_Agent	1
Свяжите с оператором.	Request_Agent	Request_Agent	1
С кем я могу поговорить?	Request_Agent	Request_Agent	1
Хочу поговорить с человеком.	Request_Agent	Request_Agent	1
Когда я смогу получить ответ от человека?	Request_Agent	Office_Hours	0
Что ты умеешь?	VA_Capabilities	Request_Agent	0
Кто-то помял мою машину.	Report_Accident	Request_Agent	0

Результаты теста показывают семь реплик, пять из которых помечены правильным намерением #Request_Agent. Первые четыре прогноза были истинно положительными. Последние две строки содержат случаи, когда намерение #Request_Agent было спрогнозировано для высказываний, которые на деле к нему не относятся («Что ты умеешь?» и «Кто-то помял мою машину»). Ложноположительные прогнозы учитываются в расчете точности: 4/(4 + 2) = 0,66.

Полный анализ всех возможных исходов для намерения `#Request_Agent` приведен на рис. 5.2. На нем также показаны истинно отрицательные прогнозы (не используются в расчетах, включены для демонстрации всех результатов).

Высказывание	Правильное намерение	Прогнозируемое намерение	Истинно положительный Request_Agent	Ложноположительный Request_Agent	Ложноотрицательный Request_Agent	Истинно отрицательный Request_Agent	Истинно отрицательный Office_Hours	Истинно отрицательный VA_Capabilities	Истинно отрицательный Report_Accident
Служба поддержки.	Request_Agent	Request_Agent	X				X	X	X
Свяжите с оператором.	Request_Agent	Request_Agent	X				X	X	X
С кем я могу поговорить?	Request_Agent	Request_Agent	X				X	X	X
Хочу поговорить с человеком.	Request_Agent	Request_Agent	X				X	X	X
Когда я смогу получить ответ от человека?	Request_Agent	Office_Hours			X			X	X
Что ты умеешь?	VA_Capabilities	Request_Agent		X		X	X		X
Кто-то помял мою машину.	Report_Accident	Request_Agent		X		X	X	X	

Рис. 5.2. Выделенные столбцы используются для вычисления точности и полноты для #Request_Agent

Теперь, когда мы знаем значения полноты и точности, можно рассчитать оценку F1, которая является гармоническим средним между полнотой и точностью. Вычисляется она следующим образом:

$$\text{Оценка F1} = (2 \times \text{Точность} \times \text{Полнота}) / (\text{Точность} + \text{Полнота}).$$

Для намерения `#Request_Agent` получим (2 × 0,66 × 0,8) / (0,66 + 0,8) = 0,72. В табл. 5.2 представлены все три показателя.

Таблица 5.2. Полнота, точность и оценка F1 для #Request_Agent

Намерение	Полнота	Точность	Оценка F1
`Request_Agent`	0,80	0,66	0,72

А что же с истинно отрицательными прогнозами?

Ранее в этом разделе мы упоминали истинно отрицательные прогнозы (или true negatives) — те, что не совпадают с неверным намерением. Они возникают всякий раз, когда у нас более одного обученного намерения, но в рамках нашего метода не несут никакой полезной информации.

Почему? Дело в том, что у каждого прогноза модели существует только один верный исход и два возможных ошибочных. Такую несправедливость трудно заметить, если рассматривать только два намерения. Но представим модель, обученную 20 намерениям. Каждый раз, когда мы делаем один истинно положительный прогноз, мы одновременно получаем еще 19 истинно отрицательных. И для каждого ложноположительного прогноза у нас есть один ложноотрицательный и 18 истинно отрицательных. Таким образом, все истинно отрицательные прогнозы накапливаются в очень большое число, которое для наших целей не дает никакой значимой информации. Поэтому мы не учитываем их вовсе.

Какая метрика важнее

Полнота, точность или оценка F1 — что важнее? Хороший вопрос! Однако ответ зависит от того, чего именно ваша компания ожидает от системы. Вот несколько моментов, которые стоит принять во внимание:

- Полнота необходима, когда ложноотрицательные результаты могут привести к серьезным последствиям. Только представьте: инструмент обнаружения мошенничества пропустил 25 % мошеннических транзакций. (Для чат-бота это было бы правильное намерение, которое не распознается в 25 % случаев.)
- Точность важна, когда частые ложноположительные результаты приводят к серьезным последствиям. Вспомните шоу «Своя игра!», где участник теряет баллы, если он берется ответить на вопрос, но ошибается (или чат-бот, который постоянно выбирает намерение #Request_Agent, вызывая ненужные эскалации).
- Оценка F1 полезна, когда ложноположительные и ложноотрицательные результаты одинаково нежелательны. Именно ее мы чаще всего выбираем для своих решений, так как она отражает сбалансированное соотношение между полнотой и точностью.

Визуализация данных с помощью матрицы ошибок

Ранее в этом разделе мы уже показывали вам матрицу ошибок размером 2 × 2, чтобы наглядно продемонстрировать возможные исходы. Матрица ошибок помогает оценить работу классификатора, визуализируя сводку прогнозов, сделанных моделью. Некоторые инструменты тестирования автоматически включают такую матрицу в результаты работы.

На рис. 5.3 показан вымышленный сценарий, в котором модель-классификатор сделала десять идеально правильных прогнозов.



Рис. 5.3. Сплошная диагональная линия показывает, что каждое прогнозируемое намерение (обозначенное одной буквой) совпадает с фактическим

Закрашенные ячейки, выходящие за пределы диагонали, дают нам полезную информацию о том, где модель допускает ошибки (рис. 5.4).



Рис. 5.4. Эта модель сделала девять правильных прогнозов, но ошибочно предсказала намерение G, тогда как фактическим намерением было E

5.1.2. Поэтапные улучшения

Поэтапные (инкрементальные) улучшения позволяют добиться измеримых, но при этом управляемых изменений. Каждое внесенное в классификатор исправление может повлиять на несколько намерений. Иногда этот эффект положительный, а иногда нет. Конечно, можно рискнуть и обновить сразу несколько намерений, но если тестирование покажет снижение эффективности, будет трудно определить виновника. Поэтому необходимо найти баланс между эффективностью и возможностью доработок.

5.1.3. С чего начать: выявление самых серьезных проблем

В целом лучше всего начинать с наиболее частых намерений, но при этом с низкими значениями оценки F1. Цели вашей компании также могут влиять на расстановку приоритетов. Если классификатор не распознает низкочастотное намерение и это приводит к дорогостоящему вмешательству человека, именно оно получит наивысший приоритет.

В оставшейся части главы мы рассмотрим вымышленный пример чат-бота, обслуживающего население в интересах Государственного бюро автотранспортных средств (государственное агентство в США, регулирующее и управляющее выдачей удостоверений личности, водительских прав и некоторых разрешений, а также занимающееся регистрацией транспортных средств).

Для начала последуем совету из главы 4 и проведем общий анализ текущих обучающих данных, представленных в табл. 5.3.

Таблица 5.3. Намерения и количество примеров к ним в базовом обучающем наборе

Название намерения	Количество примеров
Accident_Report	2
Appointment	6
Change_Contact_Records	3
Chitchat_Goodbye	3
Chitchat_Hello	4
Chitchat_Thanks	2
Chitchat_VA_About	8
Fee_Info	5
General_Negative_Feedback	6
General_Request_Agent	5
Get_ID_Number	4
Item_Not_Received	8

Таблица 5.3 (*окончание*)

Название намерения	Количество примеров
License_or_ID	5
License_Reinstatement	4
Login_Issue	4
Name_Change	6
Office_Information	6
Payment_Methods	3
Refund_Overcharge	4
Report_Sold_Vehicle	6
Report_Stolen_License_Permit_ID	5
Report_Stolen_Plates_Registration	3
Report_Stolen_Vehicle	2
Request_Receipt	4
Vehicle_Permit	5
Vehicle_Title	6
Walk_In	6
Всего	**125**

Какие выводы можно сделать о данном обучающем наборе? В нем 27 намерений и 125 обучающих примеров к ним. Примеры распределены достаточно равномерно. Качественный анализ показал, что все намерения уникальны, пересекаются лишь некоторые из них. В ряде намерений встречаются повторяющиеся термины. В результате просмотра полного набора обучающих высказываний (не приведены в таблице) было выявлено, что в нескольких намерениях встречаются такие термины, как: «права», «ПТС», «удостоверение», «транспортное средство», «угон». Однако, как показано в табл. 5.4, контексты, в которых эти слова встречались, были размечены правильно.

В целом диапазон тем соответствует назначению чат-бота, цель которого — отвечать на вопросы пользователей к Бюро автотранспортных средств.

Таблица 5.4. Примеры высказывания из базового обучающего набора демонстрируют, что некоторые термины повторяются в разных намерениях

Высказывание	Размеченное намерение
Сколько стоит получить права?	Fee_Info
Как узнать номер водительского удостоверения?	Get_ID_Number
Я так и не получил водительское удостоверение	Item_Not_Received

Высказывание	Размеченное намерение
ПТС так и не пришел	Item_Not_Received
Добавить человека в ПТС	Vehicle_Title
Как получить водительское удостоверение?	License_or_ID
Заменить разрешение на парковку	Vehicle_Permit
Я продал транспортное средство	Report_Sold_Vehicle
Хочу заявить об угоне автомобиля	Report_Stolen_Vehicle
Мое водительское удостоверение украли	Report_Stolen_License_Permit_ID

Вычисление базовых показателей

Теперь, когда мы провели первоначальную оценку наших обучающих данных, необходимо понять, как они работают на практике. Начнем с запуска *k*-блочной кросс-валидации, чтобы вычислить базовые показатели. Результаты для первой версии системы (V1), приведенные в табл. 5.5, не так уж и плохи, учитывая небольшой объем обучающих данных.

Таблица 5.5. Базовые показатели *k*-блочной валидации (V1)

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
Accident_Report	2	2	1	1	1
Appointment	6	8	1	0,75	0,8571
Change_Contact_Records	3	0	0	0	0
Chitchat_Goodbye	3	0	0	0	0
Chitchat_Hello	4	6	1	0,6667	0,80
Chitchat_Thanks	2	2	1	1	1
Chitchat_VA_About	8	8	1	1	1
Fee_Info	5	2	0,40	1	0,5714
General_Negative_Feedback	6	7	1	0,8571	0,9231
General_Request_Agent	5	4	0,80	1	0,8889
Get_ID_Number	4	6	1	0,6667	0,80
Item_Not_Received	8	6	0,6250	0,8333	0,7143
License_Reinstatement	4	4	1	1	1
License_or_ID	5	5	0,60	0,60	0,60
Login_Issue	4	4	1	1	1
Name_Change	6	8	1	0,75	0,8571
Office_Information	6	6	1	1	1

Таблица 5.5 (*окончание*)

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
Payment_Methods	3	3	1	1	1
Refund_Overcharge	4	4	1	1	1
Report_Sold_Vehicle	6	5	0,8333	1	0,9091
Report_Stolen_License_Permit_ID	5	6	1	0,8333	0,9091
Report_Stolen_Plates_Registration	3	3	0,3333	0,3333	0,3333
Report_Stolen_Vehicle	2	3	1	0,6667	0,80
Request_Receipt	4	4	1	1	1,0000
Vehicle_Permit	5	6	1	0,8333	0,9091
Vehicle_Title	6	9	1	0,6667	0,80
Walk_In	6	4	0,6667	1	0,80

Наш *k*-блочный тест включал в себя в общей сложности 125 вопросов (полный объем обучающего набора), и 105 из них были спрогнозированы верно, что дает 84 % верных результатов. Несколько намерений продемонстрировали идеальные показатели полноты и точности (что часто является признаком искусственно сформированного набора данных). Два намерения показали полноту, равную 0, и у каждого из них было всего по три обучающих примера. В этом заключается один из недостатков *k*-блочного тестирования, примеров оказалось недостаточно, чтобы корректно распределить их между сгенерированными обучающими и тестовыми наборами. Скорее всего, в продакшене такие намерения будут показывать результаты лучше чем 0. А вот намерения с идеальным показателем полноты, вполне вероятно, будут работать не так хорошо. Если вы запускаете первую версию системы и еще не располагаете другими обучающими данными, такие результаты в целом позволяют выйти в продакшен, но при этом необходимо предупредить стейкхолдеров, что фактическая эффективность бота будет ниже до тех пор, пока у вас не появятся репрезентативные данные, которые можно будет использовать для обновления обучения.

После запуска системы необходимо зафиксировать новые базовые показатели, но уже с использованием слепого тестового набора, созданного на основе логов. Пример можно найти в табл. 5.6, в которой заметен разрыв между эффективностью, спрогнозированной *k*-блочным тестом, и реальными пользовательскими вводами.

При первом запуске слепого теста было правильно определено 102 вопроса из 134, что дает долю всех верных результатов в 76 % — на 8 пунктов ниже, чем 84 %, предсказанные *k*-блочным тестом.

Таблица 5.6. Базовые показатели слепого теста (V1)

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
Accident_Report	2	2	1	1	1
Appointment	7	5	0,7143	1	0,8333
Change_Contact_Records	4	4	1	1	1
Chitchat_Goodbye	1	1	1	1	1
Chitchat_Hello	1	1	1	1	1
Chitchat_Thanks	1	1	1	1	1
Chitchat_VA_About	1	2	1	0,50	0,6667
Fee_Info	11	9	0,8182	1	0,90
General_Negative_Feedback	2	3	1	0,6667	0,80
General_Request_Agent	3	2	0,6667	1	0,80
Get_ID_Number	3	5	1	0,60	0,75
Item_Not_Received	16	9	0,4375	0,7778	0,56
License_Reinstatement	5	5	0,60	0,60	0,60
License_or_ID	7	5	0,5714	0,80	0,6667
Login_Issue	9	5	0,4444	1	0,6153
Name_Change	9	9	1	1	1
Office_Information	9	11	1	0,8182	0,90
Payment_Methods	2	2	1	1	1
Refund_Overcharge	3	4	1	0,7500	0,8571
Report_Sold_Vehicle	6	7	1	0,8571	0,9231
Report_Stolen_License_Permit_ID	7	8	0,8571	0,75	0,80
Report_Stolen_Plates_Registration	5	4	0,80	1	0,8889
Report_Stolen_Vehicle	2	2	0,50	0,50	0,50
Request_Receipt	4	4	1	1	1
Vehicle_Permit	4	5	1	0,80	0,8889
Vehicle_Title	2	8	1	0,25	0,40
Walk_In	8	5	0,3750	0,60	0,4615

Оценка начальной стратегии обучения

После того как вы получили размеченные логи и зафиксировали некоторые показатели базовой эффективности, можно проверить обоснованность решений, которые легли в основу вашей начальной стратегии обучения.

Нехватка репрезентативных обучающих данных — распространенная проблема для проектов в области разговорного ИИ. Как и во многих других недавно запущенных чат-ботах, наш начальный обучающий набор был создан экспертами в предметной области, которые подготовили искусственные примеры для тем, которые, по их мнению, будут встречаться чаще всего. На рис. 5.5 мы можем сравнить количество примеров для обучения каждого намерения с количеством примеров из логов, случайно выбранных для тестирования.



Рис. 5.5. Сравнение обучающих примеров с высказываниями в репрезентативном слепом тестовом наборе показывает, что существует определенный разрыв для многих частотных намерений (репрезентативные слепые высказывания) на левой стороне графика. Также наблюдается разрыв и по нескольким менее частотным намерениям (правая сторона)

Сравнение объемов обучающих данных и репрезентативных слепых высказываний для каждого намерения позволяет понять, насколько охват тем в нашем боте соответствует реальным пользовательским взаимодействиям. Одним из первых наблюдений стало то, что на практике `#Item_Not_Received` оказалось наиболее частотным намерением. Это подтвердило нашу начальную стратегию построения системы, заключавшуюся в том, чтобы снабдить данное намерение бóльшим количеством обучающих примеров (по сравнению с другими). Мы также отметили, что намерение `#Chitchat_VA_About` имело слишком большое количество обучающих примеров по сравнению с тем, насколько редко данная тема встречалась в логах, — следовательно, оно переобучено (over-trained), так как оказалось не таким востребованным, как мы предполагали. Однако до тех пор, пока мы не рассмотрим метрики эффективности для данных намерений, нельзя сделать какие-либо окончательные выводы. Тем не менее наши наблюдения могут лечь в основу дальнейших улучшений системы.

Задания

1. Проведите репрезентативный слепой тест с использованием ваших собственных данных и определите, какие намерения, если таковые имеются, показывают слабые результаты.
2. Соответствует ли количество обучающих примеров для конкретного намерения числу реальных случаев, наблюдаемых в логах?
3. Как бы вы расставили приоритеты в улучшении для намерений с наихудшими результатами?

5.2. Решение проблемы «намерение неверное»

Когда ваш чат-бот неправильно определяет намерение, он совершает две ошибки: дает ложноположительный (определено неверное намерение) и ложноотрицательный (не удалось определить верное намерение) прогнозы. Давайте рассмотрим необходимый цикл улучшений, чтобы продемонстрировать, как можно подойти к решению этой проблемы.

5.2.1. Улучшение показателя полноты для одного намерения

Начнем с намерения `#Login_Issue`, которое оказалось пятым по частотности, но имело заметно низкий показатель полноты — 0,44. В нашем слепом тестовом наборе было 9 примеров высказываний: четыре из них классификатор определил верно (истинно положительные), а пять — неверно (ложноотрицательные). Данное намерение показало идеальную точность, это означает, что оно никогда ошибочно не выбиралось для других намерений. В табл. 5.7 показаны соответствующие метрики.

Таблица 5.7. Метрики для #Login_Issue — запуск слепого тестового набора на базовой версии классификатора показал низкую полноту, но идеальную точность

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Login_Issue`	9	5	0,4444	1	0,6153

В табл. 5.8 представлены подробные результаты слепого теста. Классификатор пять раз не смог определить верное намерение: в трех случаях оно было определено неверно, в двух случаях уровень уверенности был настолько низким, что классификатор не вернул прогноз.

Таблица 5.8. Базовый слепой тест для #Login_Issue показал полноту в 44 %. Из девяти высказываний правильное (то есть эталонное) намерение было определено пять раз

Высказывание	Эталонное намерение	Спрогнозированное намерение	Уверенность
Сброс пароля от аккаунта	`Login_Issue`	`<none>`	—
Я не могу войти в свой профиль	`Login_Issue`	`Item_Not_Received`	0,8131
Нужна помощь со входом в профиль	`Login_Issue`	`<none>`	—
Не приходит код безопасности	`Login_Issue`	`Item_Not_Received`	0,2358
Не получилось войти в профиль	`Login_Issue`	`Login_Issue`	0,8033
У меня не получается войти в аккаунт	`Login_Issue`	`Login_Issue`	0,6680
Доступ по паролю заблокирован	`Login_Issue`	`Login_Issue`	0,5520
Сброс пароля	`Login_Issue`	`Login_Issue`	0,4875
Мне так и не прислали код безопасности	`Login_Issue`	`Item_Not_Received`	0,2091

Если мы посмотрим на обучающие примеры, станет ясно, почему некоторые запросы оказались не распознаны. Было всего четыре примера:

- Не могу войти на сайт.
- Проблема с онлайн-аккаунтом.
- Онлайн-проблемы.
- Проблема со входом в аккаунт.

Наши обучающие примеры лишены разнообразия ключевых слов и выражений, встречающихся в реальных взаимодействиях. Пользователи могут называть свою учетную запись «профилем» и перечислять конкретные проблемы, такие как «заблокирован», «сброс пароля», «не получил код безопасности». Следовательно, чтобы улучшить систему, необходимо добавить несколько репрезентативных примеров (полученных из логов):

- Помощь со входом в онлайн-портал.
- Мне нужно сбросить пароль.
- Мне нужно получить код безопасности для входа.

С учетом данных дополнений мы обновили наш классификатор до версии V2 и повторно запустили слепой тест. Посмотрим, как это повлияло на полноту для `#Login_Issue` в табл. 5.9.

Доля всех верных результатов увеличилась с 76 до 79 % (106 правильных ответов из 134), а табл. 5.10 демонстрирует заметное улучшение полноты и оценки F1. Показатель точности при этом остается без изменений.

Таблица 5.9. Результаты слепого теста показывают улучшение показателя полноты для новой версии классификатора (V2). Из девяти высказываний правильное (эталонное) намерение было определено восемь раз

Высказывание	Эталонное намерение	Спрогнозированное намерение	Уверенность
Сброс пароля от аккаунта	`Login_Issue`	`Login_Issue`	0,8253
Я не могу войти в свой профиль	`Login_Issue`	`Item_Not_Received`	0,8131
Нужна помощь со входом в профиль	`Login_Issue`	`Login_Issue`	0,6846
Не приходит код безопасности	`Login_Issue`	`Login_Issue`	0,7179
Не получилось войти в профиль	`Login_Issue`	`Login_Issue`	0,8899
У меня не получается войти в аккаунт	`Login_Issue`	`Login_Issue`	0,7840
Доступ по паролю заблокирован	`Login_Issue`	`Login_Issue`	0,9083
Сброс пароля	`Login_Issue`	`Login_Issue`	0,9204
Мне так и не прислали код безопасности	`Login_Issue`	`Login_Issue`	0,2551

Таблица 5.10. Сравнение метрик — вторая версия классификатора показывает лучшие результаты по сравнению с базовой версией (V1) для намерения #Login_Issue

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Login_Issue` — Базовая модель (V1)	9	5	0,4444	1	0,6153
`Login_Issue` — V2	9	8	0,8889	1	0,9412

5.2.2. Улучшение показателя точности для одного намерения

Теперь попробуем улучшить показатель точности для одного намерения. Метрики для намерения `#Chitchat_VA_About` не продемонстрировали никаких изменений ни в V1, ни в V2. (Необходимо смотреть на результаты теста после каждого обновления.) В табл. 5.11 показано, что полнота была идеальной, но точность составила лишь 50 %. Это означает, что классификатор придает слишком большое значение этой теме и ошибочно определяет ее у высказываний с другими намерениями (ложноположительные прогнозы для других намерений).

Таблица 5.11. Метрики после обновления до версии V2 показывают, что у #Chitchat_VA_About полнота идеальна, но точность низкая

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Chitchat_VA_About`	1	2	1	0,50	0,6667

В табл. 5.12 видно, что в нашем слепом тестовом наборе был всего один запрос для этого намерения, но классификатор определил его дважды.

Таблица 5.12. Результаты слепого теста для версии V2 показывают, что намерение #Chitchat_VA_About часто определяется для высказываний с другим намерением

Высказывание	Эталонное намерение	Прогнозируемое намерение	Уверенность
У тебя есть имя?	`Chitchat_VA_About`	`Chitchat_VA_About`	0,8042
Где мои номера?	`Item_Not_Received`	`Chitchat_VA_About`	0,3015

У нас есть восемь обучающих примеров. Мы знаем, что они были созданы искусственно (составлены по шаблону), но наши логи показывают, что тема `Chitchat_VA_About` (поболтать с виртуальным ассистентом) не особо распространена. Слепой тестовый набор содержал лишь один пример высказывания для этого намерения.

Одна из стратегий улучшения показателя точности заключается в сокращении обучающих примеров. Это сигнализирует классификатору, что данное намерение встречается гораздо реже других. Поэтому мы исключим три примера, так как они либо избыточны, либо, как в случае с «Откуда ты родом?», в логах не было подтверждений, что данный вопрос релевантен:

- Ты робот?
- Что я могу у тебя спросить?
- Что ты умеешь делать?
- Чем ты можешь мне помочь? (УДАЛЕНО ИЗ ОБУЧЕНИЯ)
- Как тебя зовут?
- Откуда ты родом? (УДАЛЕНО ИЗ ОБУЧЕНИЯ)
- С кем я разговариваю? (УДАЛЕНО ИЗ ОБУЧЕНИЯ)
- Ты кто?

После обновления обучающего набора (версия V3) мы снова запустили слепой тест и проанализировали его результаты. Было установлено улучшение показателя точности для намерения `#Chitchat_VA_About` в V3 по сравнению с V2 — получены идеальные показатели по всем метрикам. Довольно необычно, но доля всех верных результатов снизилась до 78 % (с 79 %), и один из удаленных вопросов относился к намерению `#Login_Issue`. В табл. 5.13 показаны изменения метрик от V2 к V3 для обоих намерений.

И хотя у `#Login_Issue` наблюдается небольшое снижение показателей, текущая оценка F1 равна 0,875, что все же значительно лучше, чем базовое значение F1 — 0,6153. Необходимо помнить, что небольшие датасеты более чувствительны к изменениям и любое из них в одном намерении потенциально может повлиять

на все оставшиеся. Причем изменения могут быть как отрицательными, так и положительными. Но вместо того чтобы заострять на этом наше внимание, давайте внесем еще несколько корректировок и проверим, улучшатся ли метрики.

Таблица 5.13. Метрики до и после обновления до V3 для #Chitchat_VA_About и #Login_Issue показывают, что изменения одного намерения могут повлиять на другое

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Chitchat_VA_About` — V2	1	2	1	0,50	0,6667
`Chitchat_VA_About` — V3	1	1	1	1	1
`Login_Issue` — V2	9	8	0,8889	1	0,9412
`Login_Issue` — V3	9	7	0,7777	1	0,875

5.2.3. Улучшение оценки F1 для одного намерения

Теперь перейдем к улучшению оценки F1 для `#Item_Not_Received`. Таблица 5.14 показывает, что после обновления до V3 этот показатель составил 56 %.

Таблица 5.14. После обновления до V3 оценка F1 для намерения #Item_Not_Received осталась на уровне 0,56

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Item_Not_Received` — V2	16	9	0,4375	0,7777	0,56
`Item_Not_Received` — V3	16	9	0,4375	0,7777	0,56

У намерения было 8 обучающих примеров, но логи показали, что это очень частотная тема, поэтому нам нужно добиться более высокой эффективности. Для этого добавим к нему 10 новых примеров из логов (версия V4) и проведем еще один эксперимент.

Таблица 5.15 демонстрирует, что показатель полноты для данного намерения увеличился более чем в два раза, и хотя точность немного снизилась, оценка F1 значительно улучшилась. Доля всех верных прогнозов, сделанных классификатором, также возросла с 78 до 81 %.

Таблица 5.15. Метрики для #Item_Not_Received до и после обновления показывают улучшение оценки F1

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Item_Not_Received` — V3	16	9	0,4375	0,7777	0,56
`Item_Not_Received` — V4	16	19	0,875	0,7368	0,8

5.2.4. Улучшение показателей точности и полноты для нескольких намерений

Иногда возникает путаница из-за сильного пересечения терминов между разными намерениями, имеющими схожие цели. На рис. 5.6 показана матрица ошибок, предоставленная нашим инструментом тестирования.

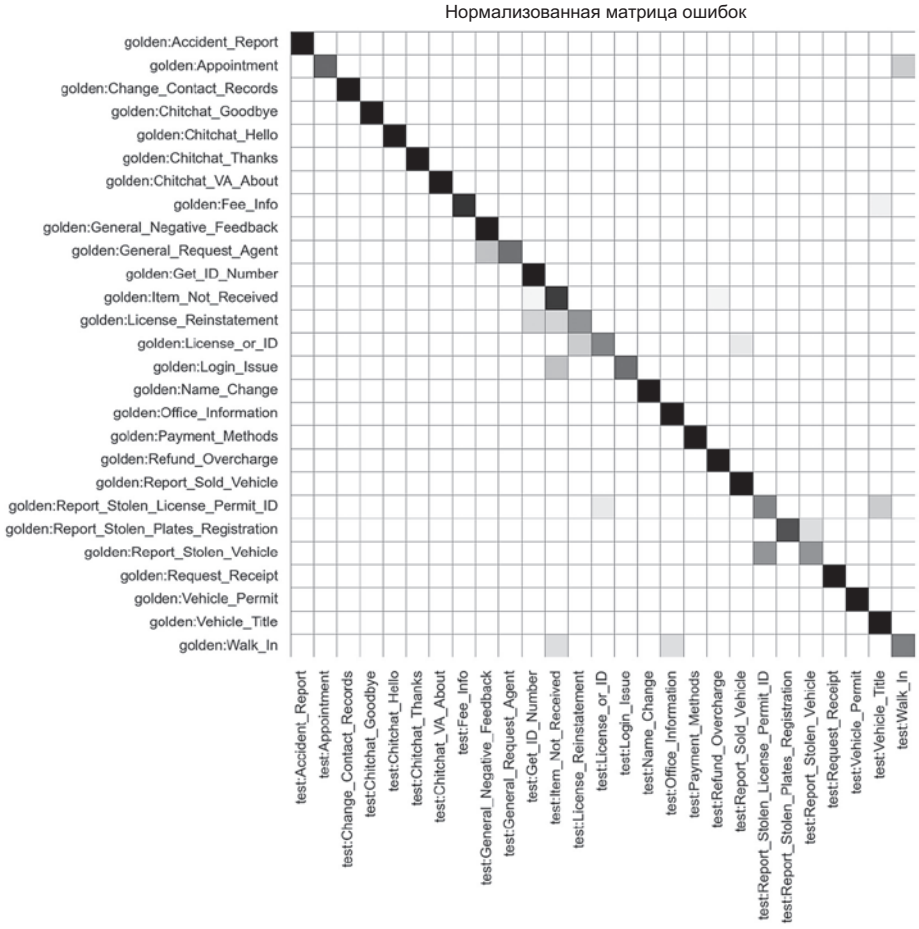


Рис. 5.6. Матрица ошибок после обновления до V4. Насыщенность заливки клеток отражает количество вопросов, определенных для данного намерения. Если бы тест классификатора верно определил абсолютно все намерения, диагональ из черных квадратов проходила бы из верхнего левого угла к нижнему правому. Клетки, выходящие за пределы этой диагонали, отмечают области ошибок модели (golden — эталон, test — тест)

В модели наблюдается путаница между намерениями, связанными с угоном. Одним из решений данной проблемы является их объединение. Однако такое

действие должно быть тщательно обдумано. Скорее всего, намерения оказались разделены неслучайно — например, они предполагают разные ответы. Вместе с тем направить диалог к соответствующему ответу можно с помощью распознавания сущностей.

Объединим все эти намерения в одно под названием `#Report_Stolen`. Примеры приведены в табл. 5.16. Но не забудьте, что слепой тестовый набор также необходимо обновить с учетом данного изменения, как и связанные сценарии диалога.

Таблица 5.16. Примеры высказываний для трех намерений, которые будут объединены в одно #Report_Stolen

Название намерения	Обучающий пример
`Report_Stolen_Vehicle`	Заявить об угоне автомобиля
`Report_Stolen_Vehicle`	Я хочу сообщить об угоне автомобиля
`Report_Stolen_Plates_Registration`	У меня украли номера
`Report_Stolen_Plates_Registration`	У меня украли свидетельство о регистрации
`Report_Stolen_Plates_Registration`	Сняли номера с автомобиля
`Report_Stolen_License_Permit_ID`	У меня украли документы
`Report_Stolen_License_Permit_ID`	Украден кошелек
`Report_Stolen_License_Permit_ID`	Украли водительское удостоверение
`Report_Stolen_License_Permit_ID`	У меня украли удостоверение личности
`Report_Stolen_License_Permit_ID`	У меня украли водительские права

Логика диалога будет обновлена таким образом, чтобы при обнаружении определенного значения сущности или синонима в высказывании выдавался соответствующий исходный ответ. Также может потребоваться добавить условие по умолчанию, чтобы уточнить или выдать общий ответ в случае, если высказывание активирует новое намерение, но никакая сущность не была обнаружена. В табл. 5.17 приведен пример подобной реализации.

Таблица 5.17. Обновление диалога с использованием распознавания сущностей для нового намерения #Report_Stolen

Распознанная сущность/синоним	Обработка
Транспортное средство, автомобиль, грузовик, мотоцикл	Направляется к исходному ответу для `#Report_Stolen_Vehicle`
Номерные знаки, регистрация, номера	Направляется к исходному ответу для `#Report_Stolen_Plates_Registration`
Водительское удостоверение, водительские права, документы	Направляется к исходному ответу для `#Report_Stolen_License_Permit_ID`
(не обнаружено)	Уточнение («Похоже, что-то было украдено. Не могли бы вы уточнить, что именно?»)

После внесения данных изменений мы получаем новую версию классификатора — V5. В табл. 5.18 собраны метрики для трех старых намерений в версии V4 и метрики для нового намерения в версии V5.

Таблица 5.18. Метрики до и после обновления до V5 показывают, что объединение трех намерений в одно #Report_Stolen приводит к идеальным результатам по всем показателям для данной темы

Намерение	Количество примеров	Количество прогнозов	Полнота	Точность	Оценка F1
`Report_Stolen_License_Permit_ID` — V4	7	5	0,5714	0,8	0,6666
`Report_Stolen_Plates_Registration` — V4	5	4	0,8	1	0,8888
`Report_Stolen_Vehicle` — V4	2	2	0,5	0,5	0,5
`Report_Stolen` — новое намерение в V5	14	14	1	1	1

Последние изменения значительно улучшили эффективность обработки данной группы запросов и повысили долю всех верных результатов до 85 %, что превышает исходное значение по *k*-блочному тесту (84 %).

После завершения этого обновления можно перейти к другим намерениям, требующим улучшения. В ходе нескольких итераций мы скорректировали оставшиеся намерения с наихудшими показателями, добавив им несколько примеров из логов. Так у нашего классификатора появилась версия V6. В табл. 5.19 представлен обзор обновленных намерений.

Таблица 5.19. Количество обучающих примеров в V6 было увеличено по сравнению с V5, и теперь оно лучше соответствует объему данных, представленному в репрезентативном наборе для слепого тестирования

Намерение	Количество обучающих примеров в V5	Количество обучающих примеров в V6	Высказывания в репрезентативном слепом наборе
`License_or_ID`	5	6	7
`License_Reinstatement`	4	6	5
`Login_Issue`	7	8	9
`Walk_In`	6	8	8

Данное обновление позволило достичь доли всех верных результатов в 92 % для последней версии классификатора (V6). В классификации естественного языка это очень высокий показатель для репрезентативного слепого набора. Достичь

100 % невозможно, таких результатов не бывает даже в реальной коммуникации между людьми.

Каждый датасет уникален, и при наличии достаточного объема данных можно провести еще несколько итераций обучения. Однако попытки приблизиться к 100 % верных прогнозов будут сопровождаться снижением отдачи. Кроме того, существует риск переобучения модели на текущем тестовом наборе. При появлении новых логов и создании нового набора для тестирования могут проявиться дополнительные пробелы в знаниях (или же факт переобучения).

Таблица 5.20 содержит сравнение оценок F1 для первой версии классификатора и последнего обновления. Двенадцать намерений остались без изменений, так как они уже показывали отличные результаты. По одному намерению мы получили снижение показателя с 90 до 80 %, оставшиеся же 14 показали улучшение. Вполне оправданный компромисс — более половины намерений стали работать лучше при незначительном снижении показателя для одного из них.

Таблица 5.20. Сравнение оценок F1 для базовой модели и для V6

Намерение	Оценка F1 для базовой модели (V1)	Оценка F1 для V6	Изменение
Accident_Report	1	1	(без изменений)
Appointment	0,8333	0,833	(без изменений)
Change_Contact_Records	1	1	(без изменений)
Chitchat_Goodbye	1	1	(без изменений)
Chitchat_Hello	1	1	(без изменений)
Chitchat_Thanks	1	1	(без изменений)
Chitchat_VA_About	0,6667	0,9524	+ 0,2857
Fee_Info	0,90	0,80	– 0,1
General_Negative_Feedback	0,80	0,80	(без изменений)
General_Request_Agent	0,80	0,80	(без изменений)
Get_ID_Number	0,75	0,8571	+ 0,1071
Item_Not_Received	0,56	0,8750	+ 0,315
License_Reinstatement	0,60	0,75	+ 0,15
License_or_ID	0,6667	1	+ 0,3333
Login_Issue	0,6153	0,9412	+ 0,3259
Name_Change	1	1	(без изменений)
Office_Information	0,90	1	+ 0,1
Payment_Methods	1	1	(без изменений)
Refund_Overcharge	0,8571	0,8571	(без изменений)

Таблица 5.20 (*окончание*)

Намерение	Оценка F1 для базовой модели (V1)	Оценка F1 для V6	Изменение
`Report_Sold_Vehicle`	0,9231	1	+ 0,0769
`Report_Stolen_License_Permit_ID`	0,80	— (объединено)	+ 0,2
`Report_Stolen_Plates_Registration`	0,8889	— (объединено)	+ 0,1111
`Report_Stolen_Vehicle`	0,50	— (объединено)	+ 0,5
`Report_Stolen`	—	1	— (объединено)
`Request_Receipt`	1	1	(без изменений)
`Vehicle_Permit`	0,8889	1	+ 0,1111
`Vehicle_Title`	0,40	0,8	+ 0,4
`Walk_In`	0,4615	0,75	+ 0,2885

Задания

1. Используя результаты предыдущего задания (приоритизация намерений с наихудшими показателями), определите категорию ошибки для каждого намерения: полнота, точность или обе метрики сразу.
2. Проведите несколько итераций корректировки обучения для улучшения каждого намерения.
3. Измерьте изменения, чтобы убедиться, что:
   - ожидаемый эффект достигнут;
   - метрики других намерений не снизились.

5.3. Решение проблемы «намерение не найдено»

Теперь, когда мы убедились, что наш классификатор хорошо работает в рамках текущей области, можно при необходимости задуматься о ее расширении. В ходе первичного анализа продакшен-логов вы наверняка обнаружите темы, которых не было в обучении. Некоторые из них довольно очевидны, но могло случиться, что в момент запуска бота у вас не было достаточного количества данных для обучения его соответствующему намерению. Или же компания еще не была готова предоставить ответы на некоторые вопросы. Иногда упускаются сезонные темы, поскольку на момент создания модели они не были актуальны (например, налоговый период, сезон ураганов, окончание финансового года и т. д.). Другие же темы могут оказаться совершенно неожиданными (например, утечка данных).

Но несмотря на отсутствие намерений, соответствующих подобным высказываниям, классификатор все равно будет пытаться их определить — он не знает, чего не знает, и старается сопоставить высказывание с наиболее близким из

известных ему примеров. В идеале классификатор должен возвращать очень низкий уровень уверенности, что вызвало бы ответ типа «anything_else» («что-то другое») или «нет необходимого действия». На практике же неизвестные высказывания часто содержат слова, встречающиеся в обучающих данных, и поэтому классификатор может ошибочно определить намерение, содержащее схожие слова в примерах высказываний.

5.3.1. Определение высказываний для новых намерений

В главе 4 нужно было выделить высказывания, относящиеся к предметной области, но не включенные в исходный охват модели. Теперь мы к ним вернемся.

Одна из тем, выявленная в логах, касалась пользователей, желающих аннулировать свои водительские удостоверения или регистрацию автомобиля. В логах можно увидеть, какое намерение классификатор присваивал каждому высказыванию в момент его поступления. Теперь их можно протестировать на последней версии классификатора (V6), чтобы получить от модели новые прогнозы.

В табл. 5.21 показано, что классификатор демонстрировал низкую уверенность и/или давал неверные прогнозы всякий раз, когда высказывание содержало глагол «аннулировать» в любой форме.

Таблица 5.21. Несопоставленные высказывания из логов и прогнозы классификатора версии V6 к ним

Высказывание	Прогнозируемое намерение	Уверенность
Аннулировать регистрацию	`Appointment`	0,2681
Аннулировать регистрацию автомобиля	`License_or_ID`	0,3651
Аннулировать водительское удостоверение	`License_Reinstatement`	0,3042
Аннулируй регистрацию	`Appointment`	0,2417
Аннулирование регистрации	`Fee_Info`	0,2786
Аннулируй мою регистрацию	`Item_Not_Received`	0,3004
Аннулировать замену водительского удостоверения	`Vehicle_Permit`	0,3264
Аннулировать права	`License_Reinstatement`	0,3237
Аннулировать ПТС или регистрацию	`Vehicle_Title`	0,5913
Аннулирование регистрации транспортного средства	`Item_Not_Received`	0,2914
Аннулировать водительское удостоверение для коммерческих перевозок	`License_Reinstatement`	0,2995
Аннулирование водительского удостоверения	`Get_ID_Number`	0,3387
Как аннулировать регистрацию транспортного средства?	`License_or_ID`	0,4324

Таблица 5.21 (*окончание*)

Высказывание	Прогнозируемое намерение	Уверенность
Мне нужно аннулировать регистрацию транспортного средства	`License_or_ID`	0,3481
Мне нужно аннулировать мое удостоверение	`Get_ID_Number`	0,3205
Я хочу аннулировать регистрацию моего автомобиля	`Change_Contact_Records`	0,3147
Аннулируй регистрацию автомобиля	`License_or_ID`	0,3447
Мне нужно аннулировать документы	`Change_Contact_Records`	0,2982
Я хотел аннулировать регистрацию	`Item_Not_Received`	0,3155
Хочу подтвердить аннулирование регистрации	`Item_Not_Received`	0,4092
Вопросы по аннулированию регистрации транспортного средства	`Fee_Info`	0,2795
Я хочу аннулировать регистрацию моего пикапа	`Item_Not_Received`	0,4761

Мы случайным образом соберем из данных примеров обучающий набор, содержащий 9 высказываний для нового намерения `#Cancel_Registration_or_License`, а оставшиеся 13 добавим в набор для слепого тестирования.

Проведя слепой тест на обновленном классификаторе (версия V7), мы получаем общую точность в 92 % — очень хороший результат, если не идеальный. Однако получить его не всегда легко, поэтому если общая эффективность вашей системы значительно снизилась, необходимо пройти через соответствующие шаги по доработке намерений, на которые повлияло изменение (в зависимости от того, коснулась ли проблема полноты, точности или обоих показателей сразу).

Рассмотрим еще один пример нового намерения. Логи содержали несколько высказываний, связанных с утечкой данных, — пример того, как эффективность чат-бота может ухудшаться из-за появления новой информации. В данном случае организация ранее не сталкивалась с утечкой данных. Однако когда это произошло и об этом узнала общественность, пользователи начали активно задавать соответствующие вопросы, что привело к появлению ошибочных прогнозов, как показано в табл. 5.22.

Для решения данной проблемы мы отобрали семь репрезентативных высказываний из логов и создали новое намерение под названием `#Data_Breach`. При этом мы проверили, что в новый обучающий набор были добавлены важные ключевые слова, такие как «взлом», «утечка» и «украден». Оставшиеся высказывания были включены в набор для слепого тестирования последней версии классификатора — V8. Новое намерение `#Data_Breach` продемонстрировало идеальный результат, а сравнение оценок F1 в табл. 5.23 показывает, что почти

все остальные намерения сохранили свои показатели или улучшили их по сравнению с базовой моделью.

Таблица 5.22. Несопоставленные высказывания на тему утечки данных из логов и прогнозы классификатора версии V7 к ним

Высказывание	Прогнозируемое намерение	Уверенность
Я хочу узнать больше о взломе портала	–	—
Я хочу узнать о нарушении безопасности на портале и о том, находятся ли мои данные под угрозой	–	—
Мой аккаунт украли	Report_Stolen	0,9483
Мой номер водительского удостоверения был украден	Report_Stolen	0,9240
Хочу задать вопрос об утечке данных	–	—
Нет, я хочу узнать об утечке данных и украденных удостоверениях	Report_Stolen	0,8604
Кто-то взломал мой аккаунт	Report_Stolen	0,4662
Кто-то использует номер моего водительского удостоверения	Get_ID_Number	0,4067
Кто-то украл мое удостоверение	Report_Stolen	0,7705
Кто-то украл мои личные данные	Report_Stolen	0,8043
Украдены данные	Report_Stolen	0,9263
Украдены персональные данные	Report_Stolen	0,9166
Украден номер социального страхования	Report_Stolen	0,7515
Повлиял ли недавний взлом на мой аккаунт?	–	—
Мой аккаунт взломан?	Login_Issue	0,3998
Произошла утечка данных?	–	—
Да, я хочу узнать, был ли украден мой номер водительского удостоверения	Report_Stolen	0,4092
Да, что мне делать с утечкой данных на портале?	–	—
Был ли украден мой номер социального страхования при взломе?	Report_Stolen	0,7806
Я хочу узнать, были ли украдены мои личные данные	Report_Stolen	0,9198

Классификатор не имел достаточной уверенности, чтобы корректно сопоставить большинство высказываний, содержащих слова «взлом» или «утечка данных», что нормально, ведь он еще не обучен данной теме. Однако высказывания, содержащие слово «украден» и его формы, сильно коррелируют с намерением #Report_Stolen, что может привести к нежелательному для пользователей результату, поскольку система пока не содержит ответов, касающихся украденных данных

Таблица 5.23. Итоговое сравнение базовой модели (V1) и финальной версии системы (V8)

Намерение	Оценка F1 базовой модели (V1)	Оценка F1 в V8
Accident_Report	1	1
Appointment	0,8333	0,8333
Cancel_Registration_or_License	— (НОВОЕ)	0,9630
Change_Contact_Records	1	0,8889
Chitchat_Goodbye	1	1
Chitchat_Hello	1	1
Chitchat_Thanks	1	1
Chitchat_VA_About	0,6667	1
Data_Breach	— (НОВОЕ)	1
Fee_Info	0,90	0,9524
General_Negative_Feedback	0,80	0,80
General_Request_Agent	0,80	0,80
Get_ID_Number	0,75	1
Item_Not_Received	0,56	0,8750
License_Reinstatement	0,60	0,75
License_or_ID	0,6667	1
Login_Issue	0,6153	0,9412
Name_Change	1	1
Office_Information	0,90	1
Payment_Methods	1	1
Refund_Overcharge	0,8571	0,80
Report_Sold_Vehicle	0,9231	1
Report_Stolen_License_Permit_ID	0,80	— (объединено)
Report_Stolen_Plates_Registration	0,8889	— (объединено)
Report_Stolen_Vehicle	0,50	— (объединено)
Report_Stolen	—	0,9630
Request_Receipt	1	1
Vehicle_Permit	0,8889	1
Vehicle_Title	0,40	0,6667
Walk_In	0,4615	0,75

Доля всех верных результатов осталась на уровне 92 %. (Обновленный набор для слепого тестирования содержал 160 вопросов, из которых 147 были

классифицированы верно.) Напомним, что в самом первом тесте доля правильных прогнозов составляла 76 % — улучшение налицо. Матрица ошибок версии V8, представленная на рис. 5.7, также показывает улучшение — диагональная линия стала заметно темнее.

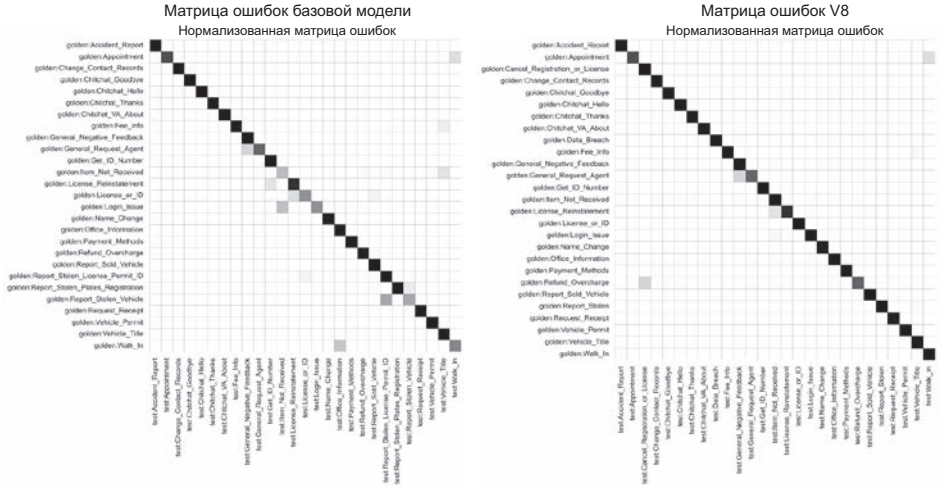


Рис. 5.7. Сравнение матрицы ошибок базовой модели (V1) и обновленной версии V8

Мы можем продолжить итерации, чтобы немного увеличить количество верных прогнозов, но для данного сценария использования таких результатов классификатора уже более чем достаточно. Любые дальнейшие изменения при ограниченном объеме данных, которым мы на данный момент располагаем, скорее всего, приведут к переобучению модели. Помните, что разумная стратегия состоит в том, чтобы планировать итерации на протяжении всего жизненного цикла чат-бота, используя новые логи и обновленные тестовые наборы.

5.3.2. Когда лучше перестать добавлять намерения

При анализе логов вы можете столкнуться с вопросами, которые относятся к рассматриваемой предметной области, но встречаются крайне редко. В наших логах мы тоже их обнаружили, но не нашли других высказываний с аналогичными целями:

- Мне нужна форма, которую должен заполнить врач, подтверждающая, что водитель больше не может управлять автомобилем.
- У меня вопрос по поводу электронных подписей.
- Каков порядок получения специального номерного знака?

Как понять, что лучше остановиться и больше не добавлять новые намерения? В данном вопросе стоит ориентироваться на данные, полученные из логов,

размеченных человеком. Можно подсчитать общее количество примеров по каждому намерению и представить их в виде графика, как показано на рис. 5.8.

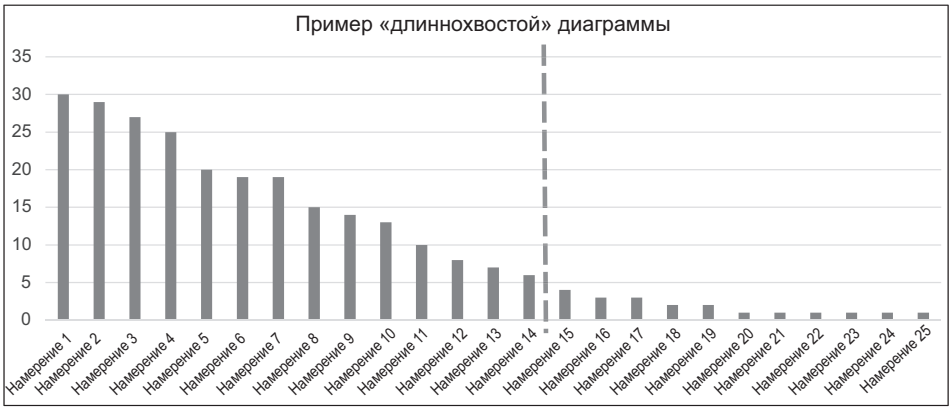


Рис. 5.8. Пример «длиннохвостой» диаграммы. Термины «короткая голова» и «длинный хвост» описывают распределение объемов данных при их отображении на столбчатой диаграмме. Намерения с наибольшим объемом данных располагаются слева (короткая голова), а по мере уменьшения объема для каждого намерения график вытягивается вправо, образуя длинный хвост

На нашей диаграмме мы определили точку, разделяющую намерения, которые должны входить в область охвата, и те, которые можно исключить. Данная точка не является статичной или строго заданной — это решение, принимаемое в соответствии с бизнес-целями и на основании минимального количества обучающих примеров, необходимого для создания нового намерения. Все, что находится слева от этой линии, вероятнее всего, стоит включить в обучение, так как есть доказательства, что эти темы будут упоминаться чаще. Все, что находится справа, не будет включено в обучение текущей версии классификатора. Со временем в логах может накопиться достаточно данных, чтобы оправдать добавление нового намерения, но до тех пор ваша система должна обрабатывать такие темы с помощью одной из следующих стратегий: выдавать ответ о том, что бот не понимает запрос, передавать диалог оператору, проводить поиск по документам или же генерировать ответ на основе RAG или LLM.

Задания

1. Определите новые темы в ваших логах и, исходя из обнаруженных высказываний, добавьте намерения.
2. Добавьте высказывания в слепой набор и протестируйте внесенные изменения.
3. Удается ли вашему классификатору распознавать новое намерение, при этом не снижая эффективности определения уже существующих?

5.4. Дополнение традиционного ИИ генеративным контентом

В разговорном ИИ мы, как правило, рассматриваем два подхода к предоставлению ответов: статический ответ (в классических реализациях на основе намерений) или полностью сгенерированный ответ (в архитектурах типа RAG). Статические ответы необходимы в случаях, когда требуется обеспечить неизменность содержания или структуры. И хотя персонализация возможна, она, как правило, ограничена заранее определенными сущностями или другими контекстными условиями диалога. Зачастую это приводит к более формальным, «холодным» ответам чат-бота. На рис. 5.9 показано, как три пользователя с одной и той же целью, но разными личными обстоятельствами получают одинаковый ответ от бота.

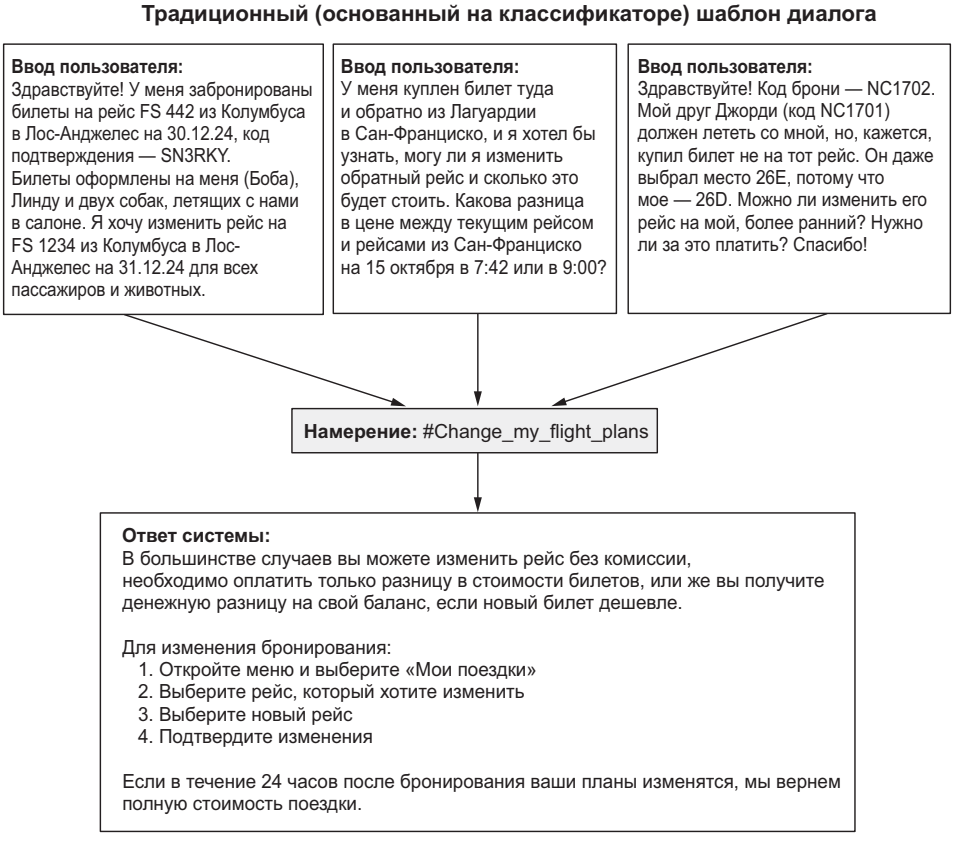


Рис. 5.9. В традиционном (основанном на классификаторе) шаблоне диалога определяется намерение пользователя, и система формирует статический или минимально персонализированный ответ

5.4.1. Сочетание традиционного и генеративного ИИ при обработке одного намерения

Мы можем улучшить пользовательский опыт, применяя гибридный шаблон ответа, сочетающий персонализированный сгенерированный контент с заранее заданными статическими ответами, написанными для конкретного намерения. Наша цель — показать, что бот понимает проблему пользователя, сохранив при этом точность передаваемой информации. Многие большие языковые модели превосходно справляются с задачами резюмирования, поэтому их можно использовать для создания дружелюбных сообщений, отражающих индивидуальный подход к каждому пользователю. На рис. 5.10 показано, как это будет выглядеть.

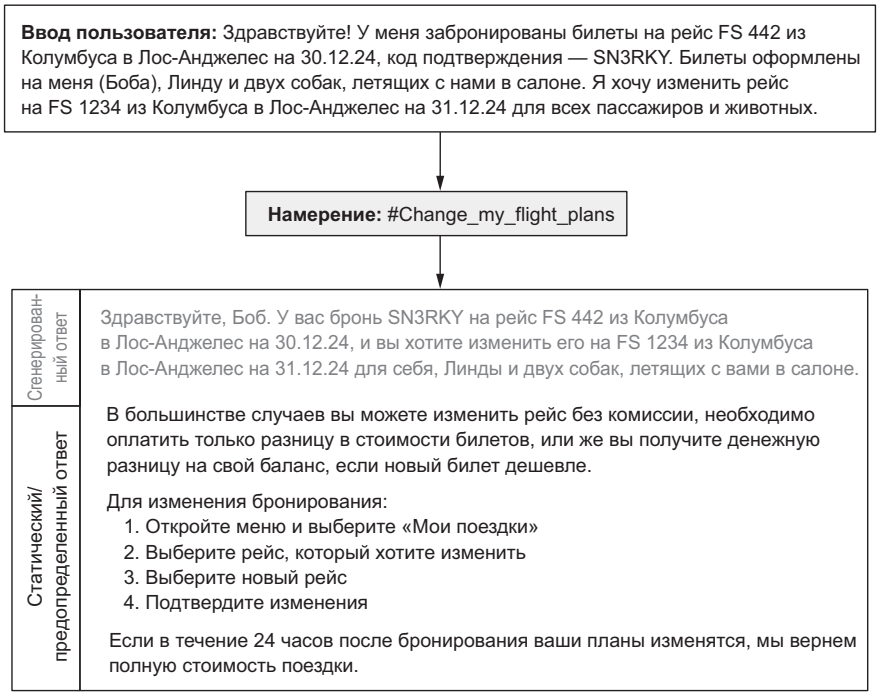


Рис. 5.10. Бот определяет намерение с помощью традиционного ИИ, а затем добавляет сгенерированный текст перед статическим ответом, настроенным для данного намерения. Сгенерированное приветствие и краткое изложение проблемы показывают пользователю, что бот понимает его цель и конкретные детали ситуации

Данный сценарий предполагает вызов API к LLM как отдельный шаг диалога. LLM генерирует текст, который затем размещается перед заранее определенным ответом. На рис. 5.11 показаны основные шаги реализации подобного шаблона.



Рис. 5.11. LLM может вызываться в рамках традиционного диалогового сценария для приветствия пользователя и краткого изложения его проблемы перед выдачей предопределенного/статического ответа

5.4.2. Промпты для демонстрации понимания

В разговорном ИИ бот обычно выступает в роли представителя компании. Это «цифровой» ресурс, а не «человеческий», однако его задача — быть лицом организации. Люди способны проявлять эмпатию, и на практике они часто повторно формулируют проблему клиента, чтобы показать, что понимают ее суть. LLM можно настроить таким образом, чтобы они имитировали подобное поведение.

Поскольку в рамках нашего сценария традиционный ИИ уже определил намерение пользователя, мы можем создать промпт, который инструктирует LLM под конкретную задачу. В данном случае нам требуется, чтобы LLM сгенерировала персонализированное, доброжелательное приветствие, которое затем можно будет объединить со статическим содержимым. Ниже приведен пример промпта для резюмирования пользовательского запроса.

Листинг 5.1. Промпт для генерации приветствия и краткого изложения проблемы пользователя

```
<|инструкция|>
Ты — оператор службы поддержки авиакомпании Friendly Skies Airline. Каждый ввод
содержит описание проблемы клиента. Сначала поприветствуй клиента, а потом
кратко перескажи его проблему.

<|ввод|>
Здравствуйте! Меня зовут Тихиро, и ранее в этом году мне возвращали деньги
на баланс при отмене рейса, но сейчас я не могу их найти. Можете проверить,
вернули ли деньги? Это было бронирование № WKRP01. Мой номер участника программы
лояльности — 8675309. Заранее большое спасибо!

<|вывод|>
Здравствуйте, Тихиро. Я понял, что вам должны были вернуть стоимость билетов на
отмененный рейс по брони № WKRP01, но вы не видите этой суммы на своем балансе.
```

Задания

1 Соберите набор высказываний, чтобы протестировать и настроить промпт LLM для генерации приветствия пользователя и краткого изложения его проблемы.

Попробуйте разные формулировки промпта. Ваша цель — создать эффективные инструкции, обеспечивающие качественные результаты для большинства тестовых высказываний.

Итоги

- Эффективность классификатора можно оценить с точки зрения доли верных результатов (accuracy), полноты (recall), точности (precision) и оценки F1. Данные показатели отражают типы ошибок, которые может допускать классификатор.
- Результаты тестирования позволят определить дальнейшие шаги по улучшению эффективности системы. Намерения с большим количеством данных, но низкой эффективностью — ваша отправная точка.
- Итеративные циклы тестирования и обучения показывают влияние внесенных изменений.
- Чат-бот может использовать дополнительные стратегии, такие как разрешение лексической многозначности, уточняющие вопросы или распознавание сущностей, чтобы давать более релевантные ответы или объединять намерения.
- Чат-бот с хорошо работающим классификатором более полезен бизнесу, если он с первой попытки выдает релевантные ответы, тем самым снижая нагрузку на операторов. Необходимо планировать мониторинг и повторное обучение системы на протяжении всего жизненного цикла бота.
- Генеративный ИИ может дополнять традиционные классификаторы, делая статические ответы модели более персонализированными и доброжелательными, в результате чего у пользователя создается впечатление, что бот действительно его понимает.

6 Повышение качества ответов с помощью RAG

В этой главе

- ✓ Повышение качества ответов чат-бота без добавления намерений
- ✓ Улучшение слабого понимания ботом с помощью RAG
- ✓ Оценка преимуществ RAG по сравнению с традиционными поисковыми моделями
- ✓ Выбор подходящих методов RAG для своей системы разговорного ИИ
- ✓ Оценка и улучшение эффективности RAG в системах разговорного ИИ

В предыдущих главах мы рассматривали такую проблему вопросно-ответных систем, как непонимание ими запросов пользователей. Сначала мы пытались решить ее, обучая чат-бота распознавать большее количество намерений, однако на каком-то этапе эффективность такого подхода начинает снижаться. Редкие нестандартные вопросы из так называемого длинного хвоста часто нецелесообразно реализовывать в виде отдельных намерений. В данной главе представлены другие методы обработки подобных случаев, включая поиск и генерацию, дополненную поиском (RAG). Такие подходы позволяют существенно повысить качество ответов бота при слабом уровне понимания.

В конце главы 5 мы пришли к выводу, что добавление новых намерений не всегда оправданно, особенно при работе с редко встречающимися запросами в рамках

конкретной предметной области. В этой главе мы рассмотрим, как можно повысить точность ответов на подобные вопросы с помощью поиска.

И поиск, и RAG позволяют улучшить работу чат-бота за счет добавления новых данных и документов без необходимости создавать новые намерения. Такой подход позволяет поддерживать тысячи намерений при обучении лишь нескольких из них. Кроме того, ответы, формируемые с помощью данных методов, проще изменять — достаточно лишь обновить документы, а не вносить изменения в сам чат-бот.

Использование поиска и RAG делает систему проще для создателей и эффективнее для пользователей. Рассмотрим, как изменится чат-бот, если в него добавить функции поиска и RAG.

6.1. За пределами намерений: роль поиска в разговорном ИИ

Традиционные системы разговорного ИИ сосредоточены на понимании пользовательских намерений. Такие системы обучаются распознавать заранее определенные категории запросов и выдавать заготовленные ответы.

На рис. 6.1 представлена концепция чат-бота, основанного на намерениях, в ее простейшем виде. Классификатор определяет намерение и выбирает соответствующий сценарий диалога. Если же он не может определить намерение пользователя, система возвращает типовой ответ вроде «Я не понял ваш вопрос», что и приводит к проблеме непонимания. Вопросно-ответная модель на основе намерений — хороший способ начать обработку часто задаваемых вопросов, поскольку она позволяет задать точный ответ для каждого типа запроса. Первоначально данный метод весьма эффективен, однако со временем все чаще становятся заметны его ограничения. Система сталкивается с запросами, выходящими за рамки заранее заданных намерений, в результате чего чат-бот выдает нерелевантные или неверные ответы, которые только раздражают пользователей. Кроме того, сопровождение и обновление намерений требуют значительных трудозатрат.



Рис. 6.1. Чат-бот, основанный на намерениях, сначала определяет намерение, а затем сопоставляет его с соответствующим ответом

Существует компромисс между точностью ответа и числом охватываемых вариантов запросов. На рис. 6.2 приведен пример из PharmaBot, рассмотренного в главе 3. Ответ точен, но обобщен: система распознала вопрос о побочных эффектах вакцины, однако не уловила все смысловые нюансы пользовательского запроса.

Для обработки таких нюансов можно добавить в бот поиск. Существуют два основных подхода:

- *Традиционный поиск* — предоставляет пользователю документы или части текста, релевантные запросу. Пользователь самостоятельно находит в них нужную информацию.
- *RAG* — поиск дополняется генерацией, тем самым найденные фрагменты объединяются в готовый ответ.

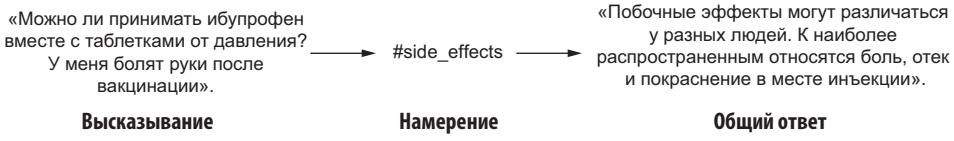


Рис. 6.2. Системы, основанные на намерениях, определяют основную тему высказывания и выдают статический общий ответ

На рис. 6.3 показано, как RAG совершенствует процесс извлечения информации, находя релевантные части документов и собирая из них конкретный, контекстный ответ. В отличие от традиционных систем на основе намерений, которые выдают фиксированные общие ответы, RAG динамически извлекает информацию, чтобы дать пользователю точный и релевантный ответ. На примере видно, как информация о сочетании ибупрофена и лекарств от давления преобразуется в конкретную рекомендацию.



Рис. 6.3. RAG находит релевантные части текста и на их основании формирует итоговый ответ

Теперь подробнее рассмотрим, как можно эффективно использовать данные возможности.

6.1.1. Использование поиска в системах разговорного ИИ

Пользовательские запросы распределяются по принципу «короткой головы и длинного хвоста», как показано на рис. 6.4. В начале распределения («короткая голова») находятся наиболее частые вопросы. Большинство же взаимодействий связано с редкими, нишевыми или специализированными запросами, составляющими «длинный хвост».

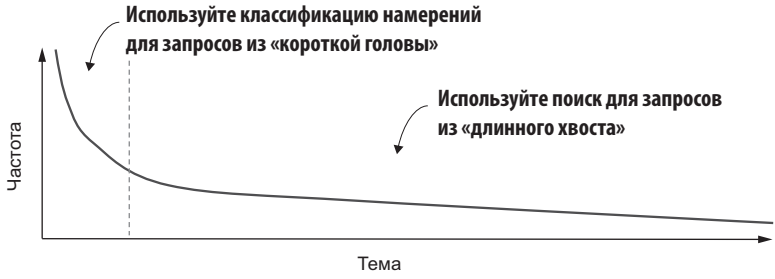


Рис. 6.4. Распределение пользовательских запросов. Намерения охватывают наиболее частые запросы, тогда как редкие и уникальные вопросы требуют интеграции поиска

Для PharmaBot «короткая голова» включает общие вопросы о COVID, такие как информация о вакцине и запись на прием. У каждого чат-бота этот набор свой, но он всегда охватывает наиболее частотные запросы. Хорошо обученная система распознает их с высокой степенью уверенности. Намерения дают разработчику полный контроль над обработкой вопросов из «короткой головы», хотя при этом могут игнорироваться некоторые смысловые нюансы.

Ранее мы видели, что PharmaBot не справился со сложным вопросом, поскольку использовал статическое намерение `#side_effects`. На рис. 6.5 показано, как тот же вопрос обрабатывается с помощью традиционного поиска.

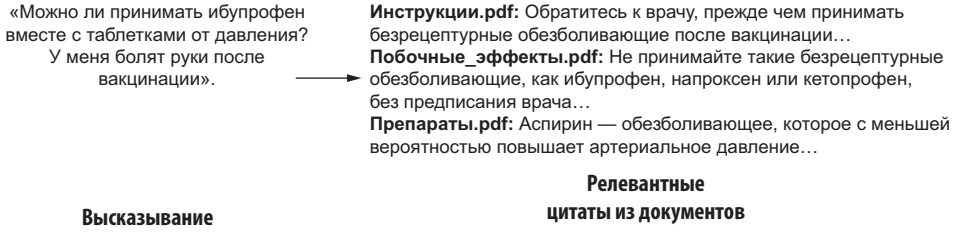


Рис. 6.5. Поиск находит релевантные фрагменты и отображает их пользователю, часто со ссылками на исходные документы

В примере выше ответ охватывает все нюансы пользовательского вопроса. Найденные цитаты из документов содержат информацию о побочных эффектах вакцины, ибупрофене, артериальном давлении и болевых ощущениях. Однако чат-бот не выдал единый связный ответ. Вместо этого он предложил ссылки на документы и выдержки из них. Пользователю придется самостоятельно проанализировать информацию из этих источников.

6.1.2. Преимущества традиционного поиска

Традиционный поиск может дополнять чат-бота на основе намерений, позволяя ему извлекать релевантную информацию из базы документов. Такой подход имеет несколько преимуществ:

- *Широта* — бот может обращаться к различным материалам в базе документов, предоставляя ответы на широкий спектр вопросов.
- *Простота сопровождения* — добавление новой информации в систему сводится к добавлению или редактированию документов в базе.
- *Технологическая зрелость* — поиск является давно устоявшейся технологией с развитой теорией и практическими методами реализации. Его можно внедрить с относительно небольшими вычислительными ресурсами и инфраструктурными затратами.
- *Скорость* — хотя поиск несколько медленнее статического ответа, основанного на намерениях, его выполнение все же достаточно быстрое.

Таким образом, поиск является отличным дополнением системы, основанной на намерениях. Наиболее распространенный способ объединения намерений и поиска заключается в использовании порога уверенности в модуле понимания естественного языка (NLU) чат-бота. NLU пытается определить намерение на основе пользовательского высказывания. Если намерение распознается с высокой уверенностью, система возвращает статический ответ. В противном случае высказывание пользователя передается в поисковый компонент (в некоторых системах разговорного ИИ он установлен как *резервное действие* или *резервное намерение*). Высокоуровневая архитектура такой модели показана на рис. 6.6.



Рис. 6.6. Намерения и поиск взаимно дополняют друг друга. Чат-бот с поддержкой поиска использует ответы на основе намерений, если высказывание распознано с высокой уверенностью; в противном случае он обращается к поиску. Комбинация намерений и поиска улучшает возможности чат-бота, однако у этого подхода есть свои ограничения

6.1.3. Недостатки традиционного поиска

При интеграции традиционного поиска в чат-бот возникают две основные проблемы: качество поисковых результатов и пользовательский опыт от их представления.

Первый существенный недостаток — зависимость качества поиска от совпадения по ключевым словам, что может привести к неточностям, если пользователь нечетко формулирует запрос. Рассмотрим предыдущий пример вопроса: «Можно ли принимать ибупрофен вместе с таблетками от давления? У меня болят руки после вакцинации». Такой запрос может быть преобразован в «ибупрофен давление лекарство руки боль вакцина», что сохраняет ключевые слова, но теряет смысловые нюансы исходного вопроса.

Не все поисковые движки ограничиваются поиском совпадения по ключевым словам, но подобный подход все же является доминирующим. Современные поисковые системы также поддерживают поиск по смыслу. Такой подход реализуется с использованием векторных баз данных, которые будут подробнее рассмотрены в следующем разделе. Как и традиционные поисковые системы, поиск по векторным базам принимает входной запрос и возвращает набор релевантных документов и цитат из них.

Второй важный недостаток поисковых решений связан с пользовательским опытом взаимодействия с найденными материалами. Ограниченное пространство окна чат-бота затрудняет удобное представление нескольких результатов поиска. Обычно это решается отображением ограниченного числа результатов (с возможностью для пользователя запросить больше).

Для экономии пространства на экране иногда показываются только ссылки на документы без цитат из них. В таком случае пользователю приходится покинуть интерфейс чата, что нарушает последовательность взаимодействия и может привести к тому, что пользователь вообще покинет чат-бот и продолжит поиск информации на сторонних ресурсах.

Результаты поиска также трудно обрабатывать через голосовой интерфейс, поскольку это приводит к длинным зачитываниям текста и не обеспечивает оптимального пользовательского опыта.

Наиболее критично то, что поиск не формирует связного ответа. Для некоторых пользователей это непринципиально, однако большинство все же не любят, когда им приходится делать выводы самостоятельно: «Зачем мне вообще был нужен чат-бот, если я мог просто поискать все сам?»

> **ПРИМЕЧАНИЕ** Можно рассмотреть вариант объединения веб-поиска с синтезом ответов. И хотя чат-бот может формировать синтезированные ответы на основе результатов веб-поиска, у такого подхода тоже есть свои ограничения. Он требует более сложной обработки, но при этом избавляет пользователей от необходимости просматривать множество документов. Эффективность системы напрямую зависит от того, насколько точно найденные цитаты соответствуют запросу. При этом итоговый ответ может потерять некоторые нюансы или контекст, если извлеченные данные неполные.

Поиск с синтезом ответов обычно основывается на методах извлечения информации по правилам, на алгоритмах ранжирования, сопоставлении ключевых слов или заранее определенных эвристиках. Ответы формируются путем объединения информации из найденных документов. И хотя такой метод способен эффективно собирать релевантные данные, он плохо справляется с неполными или многозначными запросами. Система не обладает подлинным «пониманием» содержания — она лишь выбирает и переформулирует существующий текст, что может приводить к потере контекста, фрагментарности ответов или чрезмерной зависимости от наиболее заметных, но не всегда точных источников. Ей недостает гибкости генеративных подходов.

Именно здесь и появляется RAG, который не просто извлекает текст из документов — он сочетает извлечение с генерацией, что позволяет чат-боту создавать связный, контекстно осмысленный ответ, основанный на релевантной информации из различных источников. В отличие от традиционного поиска и методов резюмирования, RAG способен адаптироваться к более широкому спектру пользовательских запросов и обеспечивать более подробные и точные ответы, используя сочетание механизмов извлечения информации в режиме реального времени и возможностей генерации естественного языка.

В следующем разделе мы рассмотрим, как RAG улучшает качество ответов чат-бота, повышая точность и сохраняя контекст даже при сложных или многослойных запросах.

6.2. За пределами поиска: генерация ответов с помощью RAG

Невозможность предоставить пользователю четкий и при этом полный ответ является недостатком традиционных методов поиска. Чтобы преодолеть данное ограничение, можно использовать в своей системе усовершенствованную альтернативу — RAG. По сути, RAG объединяет сильные стороны извлечения информации с помощью поиска и гибкость генеративных моделей, предлагая более комплексный подход к пониманию и обработке пользовательских запросов. Главное отличие заключается в том, что этот подход всегда приводит к формированию конкретного ответа.

6.2.1. Использование RAG в системах разговорного ИИ

RAG сочетает в себе преимущества методов извлечения и генерации, улучшая пользовательский опыт. Как и традиционный поиск, он извлекает релевантные выдержки из текстов для обработки запросов из «длинного хвоста». Затем RAG передает эти цитаты вместе с запросом пользователя в генеративную модель ИИ, которая формирует ответ. Таким образом, RAG «дополняет» найденные фрагменты, обеспечивая связный диалог даже при сложных или редких запросах.

Рисунок 6.7 демонстрирует, как PharmaBot с помощью RAG отвечает на уже знакомый нам вопрос:

```
Пользователь: "Можно ли принимать ибупрофен вместе с таблетками от давления?
У меня болят руки после вакцинации".
Чат-бот: Не рекомендуется принимать ибупрофен для снятия боли без предписания
врача. Лучше выбрать другое обезболивающее, которое не повышает давление,
например аспирин.
```



Рис. 6.7. RAG извлекает релевантные цитаты и дополняет ответ, синтезируя информацию в обоснованный вывод

PharmaBot извлекает те же цитаты, что и при традиционном поиске, однако теперь он суммирует их, чтобы сформировать ответ, который отражает конкретные опасения пользователя и дает адаптированный совет. Важно, что ответ основан на документах PharmaBot, а не на общих знаниях генеративной модели. PharmaBot может при необходимости предоставить ссылки на подтверждающие документы, но делает основной акцент на итоговом ответе, а не на исходных цитатах. Это обеспечивает более удобный опыт взаимодействия для пользователя.

RAG позволяет чат-ботам лучше понимать пользовательские запросы и при этом упрощает процесс разработки системы, снижая необходимость в добавлении новых намерений. Такой подход повышает удовлетворенность пользователей и обеспечивает устойчивость системы разговорного ИИ к различным изменениям языка и запросов со временем.

Использование RAG делает взаимодействие более динамичным и снижает нагрузку на пользователя при условии минимизации галлюцинаций. И хотя RAG сокращает вероятность их появления, полностью исключить их нельзя. Поэтому необходимо уделять внимание качеству извлеченных документов, поведению генеративной модели и ситуациям, когда поиск не находит релевантных источников, — об этом будет сказано далее.

Когда пользователь взаимодействует с системой разговорного ИИ, модуль извлечения подключается к надежным источникам данных, выполняет поиск (по ключевым словам, семантический или векторный) и определяет релевантность найденных результатов. Большая языковая модель (LLM) затем использует извлеченную информацию для формирования ответа. Она генерирует итоговый ответ на основе найденных данных и передает его пользователю через интерфейс чат-бота. При необходимости может также применяться автоматический перевод.

Подобно традиционному поиску, RAG может дополнять системы, основанные на намерениях. «Короткая голова» обслуживается намерениями, а «длинный хвост» — RAG. При этом ответы всегда соответствуют актуальным источникам, а обновить документы значительно проще, чем заново обучить чат-бота.

6.2.2. Преимущества RAG

RAG может стать отличным дополнением вопросно-ответных чат-ботов. Извлекая информацию из корпоративных баз знаний (а не из обученной генеративной модели), RAG гарантирует, что ответы основаны на точных и актуальных данных, тем самым повышая достоверность и надежность чат-бота. Кроме того, RAG делает системы разговорного ИИ более универсальными, позволяя эффективно обрабатывать широкий спектр пользовательских запросов и задач. Она также снижает частоту проблем типа «бот не понимает пользователя» и «пользователю приходится многое делать самостоятельно».

Преимущества RAG включают повышенную релевантность ответов, адаптивность и передачу предварительно обработанного запроса операторам поддержки.

Повышенная релевантность

Главное преимущество RAG заключается в его способности получать и интегрировать актуальную информацию из ваших баз знаний. Такие источники предоставляют более разнообразные данные, чем заранее обученные намерения, а большие языковые модели способны представить информацию из документов в виде целостного ответа.

Рассмотрим другой пример из PharmaBot. Изначально бот состоял из нескольких намерений, в том числе инструкции по посещению ресторанов. Он отвечал на основе общих рекомендаций, ссылаясь на необходимость «ознакомиться с местными правилами». Ответ не учитывал индивидуальные обстоятельства пользователя и использовал универсальный шаблон.

В табл. 6.1 приведено сравнение ответов версий PharmaBot на основе намерений и RAG. Вариант с намерениями дает общий ответ, тогда как версия с RAG адаптирует его под местоположение пользователя — Нью-Йорк.

Бот с RAG учитывает местоположение пользователя — Нью-Йорк — и предоставляет более актуальную информацию, адаптированную к местным рекомендациям и требованиям, взятую из актуальных «Рекомендаций по охране здоровья в Нью-Йорке» и обобщенную применительно к посещению ресторанов. Такой персонализированный подход улучшает взаимодействие с пользователем, предоставляя более релевантные и практически применимые рекомендации, основанные на контексте запроса.

Таблица 6.1. Сравнение ответов на основе намерений и RAG

Взаимодействие с ботом на основе намерений	Взаимодействие с ботом, интегрированным с RAG
Пользователь: Каковы текущие меры по профилактике COVID-19 для ресторанов в Нью-Йорке? PharmaBot: Текущие меры по профилактике COVID-19 для ресторанов включают ограничение заполняемости помещений, обязательное ношение масок персоналом и посетителями, когда они не сидят за столом, а также более строгие требования к уборке. Пожалуйста, ознакомьтесь с местными правилами для получения наиболее актуальной информации.	Пользователь: Каковы текущие меры по профилактике COVID-19 для ресторанов в Нью-Йорке? PharmaBot: COVID-19 по-прежнему представляет собой серьезную угрозу для жителей Нью-Йорка. Позаботьтесь о своевременной вакцинации от COVID-19 на сезон 2023–2024, которая обеспечит вам защиту от новых штаммов вируса и укрепит общий иммунитет. Маски остаются эффективным средством снижения распространения COVID-19 и других респираторных инфекций, сокращая передачу заболеваний воздушно-капельным путем при различных видах деятельности. Чтобы минимизировать риск заражения COVID-19, рекомендуется носить маски в людных закрытых помещениях. Источник: NYC Health

Адаптивность

С точки зрения диалога еще одним преимуществом RAG является способ генерации ответов. RAG может адаптировать стиль создаваемого ответа под стиль вопроса пользователя. Если пользователь задает вопрос в более формальном или деловом тоне, ответ будет сформулирован аналогично. Ответы на уникальные запросы генерируются в режиме реального времени. Иногда пользователи ожидают краткий и прямой ответ («Существуют ли ограничения на посещение ресторанов? — “да” или “нет”»), а иногда — более подробный и развернутый («Могу ли я прийти в ресторан со всей семьей и нужно ли нам надеть маски?»). И хотя оба вопроса могут опираться на одни и те же источники, ответы на них будут существенно различаться по форме и содержанию.

Передача запроса оператору-человеку (отсутствие ответа)

Бывают случаи, когда система разговорного ИИ не может дать однозначный ответ. В таких ситуациях лучше, чтобы бот сообщил, что не знает ответа, или перенаправил пользователя к специалисту. На рис. 6.8 показан пример, когда пользователь задает вопрос, на который чат-бот не может ответить.

```
Пример диалога:

Пользователь: У меня ахалазия. Может ли моя дисфагия усугубиться, если я сделаю
бустерную прививку и у меня проявятся побочные эффекты?
```

Когда пользователь задает вопрос, система разговорного ИИ выполняет многоэтапный процесс: извлекает информацию, генерирует ответ и определяет, насколько он обоснован найденными данными, прежде чем выдать окончательный результат. Следующие шаги иллюстрируют данный процесс, показывая, как система извлекает релевантные цитаты из текста, генерирует предварительный



Рис. 6.8. Дополнение RAG работой операторов. Если ответ имеет слабое семантическое совпадение с найденными документами, пользователь направляется к специалисту

ответ, оценивает его точность и в конечном итоге решает, следует ли предоставить его пользователю или нужно перевести его на оператора:

1. *Извлечение цитат из текста* — система выбирает документы, связанные с ахалазией, дисфагией и общей информацией о побочных эффектах вакцины:

   Пример 1: `"Ахалазия – это заболевание пищевода, вызывающее трудности при глотании".`

   Пример 2: `"Распространенные побочные эффекты вакцины включают боль, жар и утомляемость".`

   Пример 3: `"Дисфагия, или затруднение при глотании, может быть симптомом заболеваний пищевода, таких как ахалазия".`

2. *Генерация ответа* — LLM создает предварительный ответ на основе найденных фрагментов текста:

   ```
   "Бустерная прививка может вызвать обычные побочные эффекты от вакцины, но
   нет явных доказательств того, что она способна усугубить дисфагию у людей
   с ахалазией".
   ```

3. *Проверка соответствия* — система оценивает, насколько сгенерированный ответ опирается на найденные фрагменты, и выявляет проблему: утверждение «нет явных доказательств» напрямую не подтверждается извлеченными документами.

4. *Решение о переводе* — из-за низкого уровня совпадения между ответом и исходными текстами система делает вывод, что ответ может быть недостаточно обоснован, и предлагает перевести пользователя на специалиста:

   ```
   Ответ чат-бота: "К сожалению, в нашей базе данных нет точного ответа на ваш
   вопрос. Я могу соединить вас со специалистом, который предоставит более
   подробную информацию".
   ```

В данном сценарии система нашла релевантные фрагменты из документов и передала их LLM. Модель сгенерировала ответ, а система сравнила его с исходными данными. Такая проверка включает анализ доли слов и выражений

в ответе, совпадающих с найденными источниками. Если процент совпадений низок, система считает, что ответ не основан на документах, и предлагает пользователю альтернативный вариант решения его запроса.

Возможен и другой сценарий — когда поиск не вернул никаких документов. Тогда система не будет вызывать LLM для генерации ответа, а сразу отклонит вопрос. В обоих случаях разговорный ИИ может вернуть ответ вроде «Мне неизвестна эта информация» или любой другой резервный вариант. Оба подхода снижают вероятность появления галлюцинаций или нерелевантных ответов.

6.2.3. Сочетание RAG с другими моделями генеративного ИИ

RAG можно также использовать в сочетании с другими способами применения генеративного ИИ: например, RAG будет отвечать на информационные запросы, в то время как другие модели ИИ сосредоточатся на анализе тональности, переводе текста и прочих задачах. Используя комбинацию разных возможностей, системы разговорного ИИ могут предоставлять пользователям более широкий спектр услуг, повышая эффективность и их удовлетворенность взаимодействием.

RAG является лишь одной из форм применения генеративного ИИ, которые повышают эффективность разговорного ИИ. Когда пользователи задают типичные вопросы или запрашивают подробную информацию о продукте или услуге, RAG обращается к корпоративной базе знаний для предоставления точных и актуальных ответов. Опираясь на данные конкретной организации, RAG выдает пользователю релевантную информацию, адаптированную под его потребности.

Однако некоторые пользовательские запросы могут потребовать не только предоставления информации, но и выполнения конкретных действий (транзакционные запросы). В таких случаях система разговорного ИИ выполняет транзакцию и направляет пользователя через определенные шаги или процессы. Например, пользователь хочет совершить покупку после получения информации о продукте или услуге через RAG. В ответ система разговорного ИИ может плавно перейти к транзакционному действию, такому как, например, запуск процесса оформления заказа или запись на вакцинацию, тем самым обеспечивая последовательное и эффективное взаимодействие пользователя с системой.

> **ПРИМЕЧАНИЕ** В сценариях, где возможности RAG ограниченны, например при предоставлении данных в режиме реального времени или выполнении специфических запросов пользователей, может быть использован вызов функции, обеспечивающей динамичное получение информации из внешних систем. Такой подход позволит чатботу определять соответствующие намерения и параметры для ответов в режиме реального времени, расширяя возможности RAG при сложных взаимодействиях. И хотя данный аспект подробно не рассматривается в этом разделе, вызов функций особенно полезен в тех случаях, когда требуется более гибкая система разговорного ИИ.

И хотя RAG демонстрирует высокую эффективность и точность при ответах на пользовательские запросы, дополнительные возможности, такие как передача

запроса оператору или интеграция RAG с другими сценариями применения генеративного ИИ, могут еще больше усовершенствовать пользовательский опыт, гарантируя, что пользователи получат всю необходимую поддержку и помощь максимально эффективным способом.

6.2.4. Сравнение подходов: намерения, поиск и RAG

В табл. 6.2 представлены обобщенные характеристики и эффективность трех типов чат-ботов: основанных на намерениях, интегрированных с поиском и использующих RAG. Каждый тип оценивается в соответствии с требованиями и возможностями, которые пользователи и разработчики ожидают от разговорного ИИ. Сопоставив эти характеристики, можно определить наиболее подходящее решение для конкретной задачи.

Таблица 6.2. Сравнение возможностей чат-ботов, основанных на намерениях, чат-ботов с поиском и чат-ботов с RAG

Требования	Чат-боты на основе намерений	Чат-боты с поиском	Чат-боты с RAG
Гибкость в обработке запросов	Ограничены заранее определенными намерениями, могут игнорировать нюансы	Обрабатывают редкие запросы, предоставляя ссылки и цитаты из документов	Обрабатывают редкие запросы, предоставляя готовые ответы
Точность и релевантность ответов	При высоком уровне уверенности в определении намерения ответы точны и заранее подготовлены	Предоставляют контекстно релевантные и точные документы, помогающие пользователю найти ответ	Предоставляют контекстно релевантные и точные ответы, основанные на документах
Добавление новых данных	Добавление или обновление вручную составленных пар «намерение — ответ»	Добавление или обновление документов в базе знаний	Добавление или обновление документов в базе знаний
Сопровождение и масштабируемость	Требует обширного регрессионного тестирования при изменении данных, используемых для обучения классификатора определять намерения	База документов поддерживается за счет добавления новых и удаления устаревших материалов	База документов поддерживается за счет добавления новых и удаления устаревших материалов
Качество сгенерированных ответов	Отображаются заранее подготовленные ответы	Пользователь должен самостоятельно составлять ответ из найденных цитат и документов	Ответы основаны на исходных документах, но адаптированы к контексту вопроса

И хотя традиционные чат-боты помогают компаниям автоматизировать простые задачи и предоставлять пользователям базовую поддержку, интеграция технологий RAG значительно расширяет их возможности, позволяя выдавать более точные, контекстно осмысленные ответы и в итоге повышая качество взаимодействия с пользователем.

Задания

1. Вспомните ваш последний реализованный чат-бот и рассмотрите его с точки зрения концепции «длинного хвоста»:
   - приведите три примера редких или нетипичных запросов пользователей, которые традиционные чат-боты на основе намерений не смогут адекватно обработать;
   - проанализируйте, как эти запросы иллюстрируют феномен «длинного хвоста» в разговорном ИИ.
2. Для этого же чат-бота определите, ответы на какие вопросы можно получить с использованием традиционного поиска и на какие — с помощью RAG.

6.3. Реализация RAG

Как следует из названия «генерация, дополненная поиском», RAG состоит из двух компонентов: поиск и генерация. На этапе поиска алгоритмы ищут и извлекают данные, релевантные промпту или вопросу пользователя. В условиях открытого домена, например, для массового пользователя, информация может поступать из документов в интернете, а в закрытом корпоративном домене обычно используется более узкий набор внутренних источников для обсепечения безопасности и надежности.

6.3.1. Общая схема реализации

При использовании RAG система осуществляет поиск информации, релевантной заданному вопросу, и использует ее для генерации связного ответа. Разобьем этот процесс на несколько этапов:

1. Пользователь задает вопрос чат-боту.
2. С помощью NLU определяется намерение пользователя:
   - Если система распознала вопрос с высокой степенью уверенности — например, если он относится к уже известным намерениям, которым обучена модель, — она может ответить сразу, и поиск не требуется. На этом процесс завершается.
   - Если система не может распознать запрос, она переходит к поиску. Запрос пользователя отправляется в поисковый инструмент, который анализирует содержимое документов, формирует результаты поиска и ранжирует их.

3. Ранжированные результаты поиска возвращаются в чат-бот для отображения. (До появления RAG на данном этапе чат-бот просто получал список ссылок и цитат. Это уже полезнее, чем «Извините, я не понял ваш вопрос».)
4. Вместо простого отображения результатов исходный вопрос пользователя и найденные результаты отправляются в LLM. Модель может заново проранжировать результаты, но основная ее задача — сгенерировать краткий, обобщенный и грамматически правильный ответ.
5. Ответ передается обратно в систему.
6. Ответ отображается пользователю через интерфейс чат-бота.

> **ПРИМЕЧАНИЕ** Ни исходный вопрос пользователя, ни сгенерированный ответ не обязаны *полностью* совпадать с текстами документов. И хотя дословные ответы возможны, а в некоторых случаях даже предпочтительны (например, по юридическим причинам), основной акцент делается на то, чтобы ответ был обоснован содержанием базы знаний, то есть опирался на проверенный набор документов.

По своей природе LLM не гарантируют одинаковые результаты при каждом обращении. Модель может выдавать разные ответы на один и тот же вопрос из-за незначительных различий в контексте или формулировке. Такая изменчивость обусловлена вероятностной природой LLM: они генерируют текст на основе вероятностных закономерностей, а не извлекают фиксированные ответы. И хотя это обеспечивает гибкость системы и позволяет создавать более точные и контекстно уместные ответы, стабильных результатов от таких моделей ожидать не стоит.

Важно обратить на это отдельное внимание, потому что члены команды, не знакомые с принципами работы LLM, нередко ожидают однородных ответов. Такое недопонимание тормозит развитие проекта и вызывает разногласия в ходе решения задач. Осознание того, что LLM ориентируются на релевантность и контекст, а не на точное воспроизведение документа, помогает согласовать ожидания и наладить эффективное взаимодействие внутри команды.

В некоторых случаях сгенерированный ответ может быть очень близок к формулировкам в исходных документах или полностью совпадать с ними. Такое происходит, если документы содержат релевантные и информативные фрагменты, напрямую отвечающие на вопрос пользователя. В подобных ситуациях модель RAG может включить в ответ дословные цитаты, предоставляя максимально точную и релевантную информацию.

В других же случаях модель RAG использует сведения из базы знаний для понимания контекста и ключевых концепций, связанных с запросом пользователя. На основе этого она формирует ответ, соответствующий содержанию документов, даже если конкретная формулировка вопроса или ответа в них дословно не встречается. Такой подход обеспечивает большую гибкость и адаптивность, позволяя выдавать ответы, которые эффективно решают задачу пользователя и при этом опираются на достоверные данные из базы документов.

6.3.2. Подготовка базы документов для RAG

Рассмотрим, как осуществляется извлечение данных при поиске в ходе RAG. На рис. 6.9 представлена схема подготовки данных.

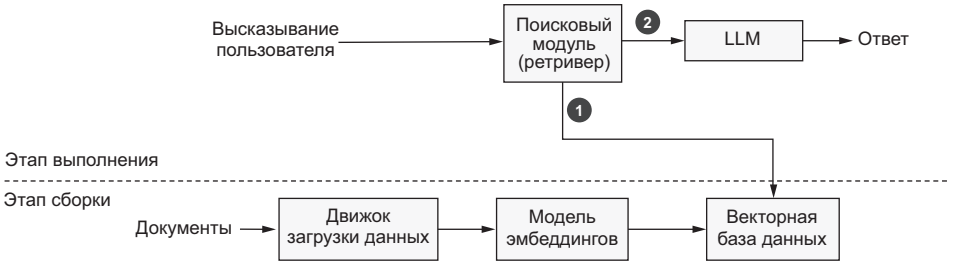


Рис. 6.9. В процессе сборки и выполнения RAG использует векторную базу данных

Системный пайплайн предварительной обработки гарантирует, что как исходные данные, так и вопрос пользователя (или его переформулировка от LLM) пригодны для использования в поиске RAG. Этот пайплайн играет ключевую роль в преобразовании данных в эмбеддинги, что позволяет эффективно сопоставить запрос пользователя с релевантной информацией. Затем с помощью различных методов (например, косинусного сходства и др.) выявляются наиболее подходящие совпадения, это обеспечивает точные, контекстно релевантные результаты. Ниже приведены ключевые этапы такого процесса, описывающие, как данные подготавливаются для их последующего использования в поиске:

1. *Предварительная обработка данных*. Система (обычно пайплайн данных, а не сама LLM) подготавливает исходные документы к поиску. Например, PDF-документы конвертируются в текст, таблицы — в обрабатываемые структуры. Для расширения исходного содержимого также могут добавляться метаданные. Затем текст делится на связные семантические единицы — *чанки*. Например, документ может быть разделен по абзацам. Чанкинг — это стандартный процесс выделения и извлечения осмысленных групп слов (чанков) из предложений для их последующего анализа или обработки. Выбранная стратегия чанкинга влияет на итоговое качество результатов. Существует множество инструментов с открытым исходным кодом, позволяющих графически представить и проанализировать различные подходы к чанкингу или разделению текста.
2. *Генерация эмбеддингов*. Модель эмбеддингов преобразует чанки в эмбеддинги — числовые представления слов или фраз в многомерном векторном

пространстве. Эмбеддинги фиксируют семантические связи между словами и документами, обеспечивая более глубокое понимание контекста. Схожие значения и контексты располагаются ближе друг к другу, а несхожие — дальше, что дает более релевантные результаты поиска по сравнению с поиском по ключевым словам.

3. *Сохранение в векторной базе данных.* Созданные эмбеддинги сохраняются в векторной базе данных, что облегчает поиск по сходству. Каждый чанк индексируется по своему векторному представлению, и это позволяет быстро проводить поиск на основе смысла, а не совпадений слов.
4. *Извлечение и сопоставление в ходе выполнения.* В процессе работы система взаимодействует с пользователем через чат-бота. Вопрос пользователя преобразуется в вектор с помощью той же модели эмбеддингов, после чего данный вектор ищется в базе данных для извлечения наиболее релевантных фрагментов (чанков) на основе семантического сходства. Отобранные фрагменты передаются в LLM, которая синтезирует их в ответ и возвращает пользователю.

Каждый из упомянутых выше этапов направлен на оптимизацию процесса поиска. Это позволяет LLM формировать точные и контекстно осмысленные ответы на основе наиболее релевантных данных.

В листинге 6.1 приведен пример кода для функции генерации эмбеддингов. Можно использовать любую кастомную функцию или другую векторную базу данных — эффективность при этом будет зависеть лишь от выбранной модели эмбеддингов. Наиболее распространенный подход к RAG — создать плотное векторное представление базы знаний, чтобы вычислять семантическое сходство с пользовательскими запросами. В примере ниже в качестве векторной базы данных используется Chroma.

Листинг 6.1. Разделение файла на чанки, генерация эмбеддингов и их сохранение в векторной базе данных

```
from langchain.document_loaders import TextLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma

loader = TextLoader(filename)
documents = loader.load()

text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
texts = text_splitter.split_documents(documents)

from langchain.embeddings import HuggingFaceEmbeddings

embeddings = HuggingFaceEmbeddings()
docsearch = Chroma.from_documents(texts, embeddings)
```

RAG работает за счет извлечения релевантных документов или фрагментов текста на основе пользовательского запроса, а затем — генерации ответа на естественном языке. Этот процесс может быть реализован и без разбиения текста на чанки или создания эмбеддингов — с помощью других подходов к извлечению документов и генерации текста.

Так, например, использование Lucene в качестве альтернативы чанкингу и эмбеддингам предполагает применение его возможностей индексирования и поиска по документам. Lucene способен находить наиболее релевантные документы в зависимости от запроса. После этапа поиска модуль генерации RAG формирует связный ответ на основе содержания найденных документов. Lucene демонстрирует высокую эффективность при поиске текстовой информации, а такая сложная задача, как генерация ответов, подобных человеческим, передается специализированным генеративным компонентам модели RAG. Этот подход особенно эффективен в системах, где приоритет отдается точности и скорости поиска, а не глубине семантического понимания.

Задания

1. Разделение текста на чанки. В этом упражнении вам предлагается поэкспериментировать с различными стратегиями чанкинга и генерацией эмбеддингов:
   - Выберите пример текстовых данных (небольшие текстовые файлы на ваше усмотрение).
   - Определите стратегию чанкинга (разделение по предложениям или словам). В качестве эксперимента попробуйте использовать инструмент для чанкинга.
   - Создайте эмбеддинги для чанков с помощью открытой модели эмбеддингов и загрузите полученные чанки в векторную базу данных (например, Chroma).
2. Настройка пайплайна загрузки данных. В этом задании вы научитесь строить простой пайплайн загрузки для обработки документов в системе RAG:
   - Выберите набор документов, релевантных сфере деятельности вашей компании или конкретному случаю использования чат-бота. Начните с более простых текстов (без таблиц и т. п.).
   - Постройте пайплайн загрузки, учитывая такие факторы, как простота использования и совместимость с RAG. Для поиска используйте модели с открытым исходным кодом, модели эмбеддингов Hugging Face и LlamaIndex.
   - Разверните пайплайн загрузки для предварительной обработки и структурирования датасета для работы с RAG.
   - Протестируйте пайплайн, чтобы убедиться в корректности его работы и целостности данных.

6.4. Дополнительные аспекты реализации RAG

Традиционный поиск возвращает ссылки, фрагменты или полные тексты релевантных документов, и пользователям приходится самостоятельно просматривать их, чтобы найти нужный ответ. В отличие от него, RAG возвращает уже готовый ответ, при этом пользователь при желании может ознакомиться с документами, использованными для его формирования.

6.4.1. Почему нельзя просто использовать LLM напрямую?

Что, если система разговорного ИИ просто передаст запрос пользователя в LLM и получит от нее ответ? Ведь LLM обучаются на огромных объемах данных.

Во-первых, LLM, обученные на данных, доступных в интернете, имеют ограничения, связанные с природой этих данных, которые представляют собой некий снапшот состояния источников, доступных на момент обучения. Это означает, что они не содержат специфическую для какой-либо сферы, личную или закрытую информацию, а также данные, созданные после окончания обучения. Поэтому даже самые современные модели LLM со временем устаревают и выдают неточные ответы. RAG решает данную проблему, предоставляя модели актуальные данные уже после этапа обучения.

Во-вторых, поскольку LLM обучаются на обширных и разнообразных датасетах, отследить происхождение ответов зачастую невозможно, что снижает доверие к результатам. RAG, напротив, гарантирует обоснованность результата: вам точно известно, какие данные были переданы модели для генерации ответа.

Широкий тематический охват также создает проблему, ведь имея доступ к огромному объему информации, LLM могут выдавать ответы с высокой уверенностью, даже не располагая достоверными сведениями или контекстом. Такая склонность к созданию правдоподобных, но неверных или неподтвержденных данных называется *галлюцинациями*. Мы же ожидаем, что система разговорного ИИ будет выдавать корректные и точные ответы. И хотя определенные методы составления промптов могут частично снизить риск галлюцинаций, предоставление данных через RAG является более надежным решением.

Также важно, чтобы приоритет отдавался ответам, основанным на конкретных документах или их корпусе, а не на данных, на которых обучалась LLM. Основная задача системы RAG — формировать ответы, опирающиеся на содержание и контекст ваших документов, благодаря чему эти ответы становятся точными, релевантными и заслуживающими доверия пользователей.

При выборе конкретной LLM необходимо учитывать, на каких данных и в какой предметной области она обучалась. Если модель обучена на общих данных, а вам нужны специализированные результаты, она не сможет обеспечить должное качество ответов. В таких случаях стоит рассмотреть метод объединения моделей, при котором они комбинируются для использования сильных сторон каждой из

них. Это повышает эффективность в определенных сферах применения. Если у вас есть необходимые для этого ресурсы и данные, можно также рассмотреть возможность дообучения (тонкой настройки) выбранной модели под конкретные задачи. Однако это требует значительных вычислительных и финансовых затрат, поэтому лучше сначала попробовать настройку промптов (prompt-tuning). Несмотря на то что стоимость дообучения постепенно снижается и продолжит снижаться и дальше, этот шаг все же требует тщательного анализа. Кроме того, появляются и новые методы обучения под конкретные предметные области, обеспечивающие дополнительную гибкость при разработке таких систем.

6.4.2. Поддержание актуальности и релевантности ответов с помощью RAG

RAG дает значительные преимущества по сравнению с ответом на вопросы напрямую через LLM. RAG использует LLM для генерации текста на естественном языке, и модель формирует точные ответы на основе найденных документов. Механизм поиска в режиме реального времени позволяет получать актуальную информацию, даже если новые или обновленные документы были добавлены в процессе сборки. Таким образом, RAG гарантирует, что сгенерированный ответ отражает самые последние данные из исследуемых источников. Такая интеграция корпоративного контента повышает релевантность и точность ответов и укрепляет доверие пользователей, которые знают, что получают актуальную и достоверную информацию.

Кроме того, RAG не ограничивается простым доступом к корпоративному контенту. Она обращается к конкретным фрагментам и извлекает данные из множества документов, что позволяет отслеживать и проверять ответы вплоть до их точных источников, тем самым обеспечивая пользователям полную прозрачность и надежность. Это также упрощает процесс верификации и повышает ответственность команды разработчиков, позволяя точно понять, как и почему бот генерирует тот или иной ответ.

RAG также определяет границы области знаний LLM, удерживая ее в пределах собственной компетенции. В отличие от LLM, используемых напрямую, которые могут пытаться дать ответ вне своей области знаний, RAG способен распознать запрос, выходящий за рамки его возможностей, и сообщить об этом. Способность ответить пользователю «Мне неизвестна эта информация» предотвращает выдачу неверных ответов и способствует прозрачности взаимодействия. Устанавливая четкие границы своего понимания, RAG помогает разработчикам создавать системы ИИ, ориентированные на точность, надежность и достоверность, тем самым улучшая общий пользовательский опыт.

Есть большая разница между ответами «Я не понимаю» и «Я не могу найти ответ на ваш вопрос». Основная цель RAG заключается в генерации информативных и релевантных ответов на основе содержимого найденных документов, однако бывают случаи, когда система не может обнаружить достаточно материала для

осмысленного ответа. В таких случаях модель RAG признает невозможность дать удовлетворительный ответ и сообщает об этом пользователю.

Тем не менее важно осознавать, что конкретное поведение системы RAG, например вывод ответа «Мне неизвестна эта информация», зависит от строения модуля поиска, качества базы знаний и настроек или параметров генеративной модели. Кроме того, разработчики могут реализовать специальные стратегии или резервные механизмы на случай, если система не способна сгенерировать ответ: например, предложить альтернативные варианты или запросить у пользователя дополнительную информацию.

6.4.3. Как настроить пайплайн загрузки данных?

Настройка пайплайна загрузки данных, который корректно сохраняет структуру документов, имеет решающее значение для обеспечения точных результатов поиска в системе RAG. Необходимо учитывать несколько ключевых моментов. По сути, каждое архитектурное решение, принятое при проектировании компонентов, влияет на общую точность результатов.

Прежде всего, необходимо реализовать механизмы подключения существующих хранилищ контента к системе поиска или переноса данных в новое хранилище. Это позволит системе поиска получать необходимую информацию, при этом сохраняя целостность данных.

Следующая задача заключается в корректном извлечении структурных элементов (таких, как заголовки, таблицы и списки) во время загрузки данных. Элементы форматирования способствуют лучшей организации и четкому представлению документа. Сохранение структуры данных при их загрузке позволяет системе поиска использовать это для повышения точности и релевантности выдачи.

Кроме того, существуют сложности, связанные с разбиением текста на чанки. Возможность разделения больших документов на поддокументы для их индексации повышает эффективность поиска. Это обеспечивает более точную индексацию и извлечение данных, что способствует ускоренному доступу к нужным фрагментам внутри объемных документов. При этом выбор подходящих методов поиска, таких как векторный, семантический, федеративный, по ключевым словам или гибридный, значительно расширяет возможности поисковой системы.

Использование платформы LangChain упрощает настройку пайплайна загрузки данных. Напомним, что для этого вам понадобятся:

- *загрузчики документов* (loaders) — для загрузки данных различных форматов;
- *инструменты преобразования документов* (transformers) — для обработки и структурирования данных с целью обеспечения эффективного поиска;
- *механизмы поиска* (retrievers) — для извлечения наиболее релевантных чанков документов в ходе запроса.

Загрузчики документов обеспечивают работу с различными форматами, упрощая процесс и обеспечивая эффективную обработку и извлечение контекста, необходимого для точных ответов LLM. Они загружают данные из исходных документов, рассматривая каждый извлеченный фрагмент как отдельный документ, состоящий из текстового содержимого и связанных метаданных. LangChain предоставляет встроенные инструменты для работы с различными типами файлов: всеми файлами в каталоге, PDF, CSV, JSON, HTML, markdown, txt и др.

Так, например, можно загрузить транскрипты, корпоративные документы или тексты с веб-страницы. Загрузчики предоставляют метод `load` для загрузки данных из настроенного источника:

- Загрузчик текстовых файлов:

```
from langchain_community.document_loaders import TextLoader

# Загрузка текстовых данных из файла с помощью TextLoader
loader = TextLoader("./ваши_данные/ваш_текст.txt")
document = loader.load()
```

- Загрузчик CSV-файлов:

```
from langchain_community.document_loaders import CSVLoader

# Загрузка данных из CSV-файла с помощью CSVLoader
loader = CSVLoader("./ваши_данные/ваша_таблица.csv")
document = loader.load()
```

  Ознакомьтесь с документацией LangChain, касающейся настройки парсинга и загрузки CSV. Например, вы можете указать собственные разделители, имена полей и другие параметры.

  Аналогичным образом LangChain предоставляет `DirectoryLoader` для загрузки всех документов из каталога, `UnstructuredHTMLLoader` для загрузки HTML-документов и другие инструменты для наиболее распространенных форматов, а загрузчик `AzureAIDocumentIntelligenceLoader` полезен для работы с документами Microsoft Office.

- Загрузчик документов Microsoft:

```
%pip install --upgrade --quiet langchain langchain-community
➥azure-ai-documentintelligence

from langchain_community.document_loaders import
AzureAIDocumentIntelligenceLoader

file_path = "<ваш_файл>"
endpoint = "<ваш_эндпоинт>"
key = "<ключ>"
loader = AzureAIDocumentIntelligenceLoader(
    api_endpoint=endpoint, api_key=key, file_path=file_path,
    ➥sapi_model="prebuilt-layout"
)

documents = loader.load()
```

После загрузки документов необходимо перейти к *инструментам их преобразования*, позволяющим разбить длинный документ на более мелкие части, чтобы выбранная LLM могла их обработать. LLM обладают свойством, называемым «контекстным окном», определяющим длину текста, которую модель может эффективно обработать за один проход, поэтому части текста должны соответствовать размеру этого контекстного окна. Логично предположить, что увеличение контекста (то есть длины текстовых фрагментов) автоматически приведет к повышению качества работы модели при решении задач, связанных с пониманием языка. Однако последние исследования показывают, что это не всегда так. Существуют данные, указывающие на то, что языковые модели демонстрируют более высокие результаты, если им предоставляется меньше текста, но при этом текст является максимально релевантным текущей задаче.

Большее контекстное окно позволяет включать больше информации в промпт в ходе инференса, однако такая техника, часто называемая *«prompt stuffing» (перегруженный промпт)*, имеет свои недостатки. Обработка большего объема текста требует больше вычислительных ресурсов, что замедляет инференс и увеличивает стоимость вычислений, особенно для компаний, оплачивающих использование модели по количеству токенов, где сведение длинных документов, таких как годовые отчеты или расшифровки встреч, может стать затратным. И хотя увеличение размера контекстного окна может частично улучшить результаты, данный эффект имеет свой предел. Подобно человеку, LLM может испытывать «информационную перегрузку»: при чрезмерном количестве деталей модель может упустить действительно важные моменты. Исследования показали, что LLM чаще фокусируется на информации, расположенной в начале или в конце промпта, и может пропустить ключевые инструкции, находящиеся посередине.

Для этого нам и нужны инструменты преобразования документов, которые выполняют предварительную обработку текста, извлекают релевантную информацию и придают ей более структурированный вид, пригодный для эффективного использования языковой моделью при генерации текста. В LangChain есть такие встроенные преобразователи, упрощающие работу с документами.

Процесс разбиения текста выполняется следующим образом:

1. Текст делится на небольшие фрагменты.
2. Эти фрагменты объединяются в более крупные чанки заданного размера, который обычно устанавливается некоторой функцией.
3. По достижении требуемого размера формируется новый текстовый юнит. Далее создается следующий фрагмент с небольшим перекрытием, чтобы сохранить контекст между частями текста.

Вы также можете выбрать правила разбиения (по символам, словам, токенам) и способ измерения размера фрагмента. В пакете langchain-text-splitters

доступно множество типов разбиения. Ниже приведены несколько примеров разделителей текста:

1. *Рекурсивный* — рекурсивное разбиение текста рекомендуется использовать в первую очередь. Оно направлено на размещение связанных между собой фрагментов текста рядом.
2. *HTML* — «структурно-ориентированный» разделитель, разбивающий текст по HTML-элементам, — пример показан в листинге 6.2. Он осуществляет разбиение на уровне элементов, добавляя метаданные к заголовкам для повышения релевантности чанков. Такой подход сохраняет семантическую группировку и контекстную структуру документа:
   - *по символам* — разбивает документ по заданным пользователем символам (например, «\n\n»);
   - *по коду* — использует синтаксические и грамматические элементы языков программирования, таких как Python и JavaScript (и еще 13 других), структурируя код в логические блоки;
   - *Markdown* — определяет разметку Markdown и организует документ в структурированный формат (аналогично HTML);
   - *по токенам* — применяет токенизатор, например tiktoken, чтобы разбить текст в соответствии с ограничениями модели по количеству токенов.

Листинг 6.2. HTML-разделитель

```
# Установка пакета langchain-text-splitters, если еще не установлен
%pip install -qU langchain-text-splitters

# Импорт необходимых модулей
import langchain
import langchain_text_splitters

print("Версия Langchain:", langchain.__version__)
print("Разделители текста Langchain успешно загружены!")
from langchain_text_splitters import HTMLHeaderTextSplitter
from langchain.schema import Document
# Убедитесь, что документ импортирован корректно

from bs4 import BeautifulSoup

print("BeautifulSoup успешно загружен!")

# Пример HTML-контента для разбиения
html_string = """
<!DOCTYPE html>
<html>
<body>
<div>
<h1>Введение</h1>
<p>Здесь располагается введение к документу.</p>
<div>
```

```
<h2>Глава 1: Начало работы</h2>
<p>Данный раздел содержит основные тезисы для начала работы.</p>
<h3>Раздел 1.1 Установка</h3>
<p>Данный подраздел рассказывает о процессе установки.</p>
<h3>Раздел 1.2 Конфигурация</h3>
<p>Данный подраздел содержит параметры конфигурации.</p>
</div>
<div>
<h2>Глава 2: Продвинутые техники</h2>
<p>Данный раздел содержит информацию о более продвинутых техниках.</p>
</div>
<br/> <!-- Фикс: Правильное форматирование самозакрывающегося тега -->

<p>Что вы узнали из введения.</p>
</div>
</body></html>
"""

# Определение заголовков для разбиения (h1, h2, h3 представляют разные уровни)
headers_to_split_on = [
    ("h1", "Заголовок 1"),  # Заголовки верхнего уровня
    ("h2", "Заголовок 2"),  # Заголовки разделов
    ("h3", "Заголовок 3"),  # Заголовки подразделов
]

# Инициализация разделителя HTML-заголовков с заданными уровнями
html_splitter =
     HTMLHeaderTextSplitter(headers_to_split_on=headers_to_split_on)

# Разбиение HTML-документа на структурированные фрагменты
html_header_splits = html_splitter.split_text(html_string)

# Отображение результатов
for doc in html_header_splits:
    print(f"Content:\n{doc.page_content}\nMetadata: {doc.metadata}\n{'-'*40}")
```

Далее рассмотрим процесс создания эмбеддингов. Тип данных и требования к языковой поддержке определяют выбор модели эмбеддингов. Кроме того, если вы работаете с терминологией, специфичной для определенной предметной области, модели эмбеддингов могут потребовать расширения.

Модели эмбеддингов в LangChain преобразуют текст в числовые представления — эмбеддинги, которые могут быть обработаны системой. LangChain интегрируется с различными провайдерами моделей (OpenAI, Cohere, Hugging Face и др.) для генерации эмбеддингов. Например, класс `OpenAIEmbeddings` использует API OpenAI для создания эмбеддингов, что можно сделать с использованием ключа как API OpenAI, так и API Azure OpenAI.

```
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings()
text = "Тестовый документ."
query_result = embeddings.embed_query(text)
query_result[:5]
```

Также возможны интеграции `CohereEmbeddings`, `TensorFlowEmbeddings` и `HuggingFaceInferenceEmbeddings`.

После получения эмбеддинги необходимо сохранить в векторной базе данных, такой как, например, Chroma, которую мы использовали ранее. При выборе векторной базы данных следует учитывать ее эффективность в ходе выполнения, масштабируемость относительно размера датасета и общую эффективность. Другой важный аспект — возможность интеграции с такими инструментами, как LangChain, который постоянно совершенствуется. LangChain расширяет возможности векторных баз данных, упрощая процесс обработки эмбеддингов и интеграции с различными рабочими процессами машинного обучения и ИИ. Такое сочетание обеспечивает эффективное управление данными и их извлечение, делая систему достаточно надежной для масштабируемых и высокопроизводительных приложений.

Механизмы поиска, или *ретриверы*, служат своеобразным «мостом» между эмбеддингами и пользовательскими запросами. Пока эмбеддинги хранят числовые представления документов в векторной базе данных, ретриверы определяют и извлекают наиболее релевантные фрагменты на основе оценки схожести.

Работа ретривера выглядит следующим образом:

1. Запрос пользователя преобразуется в эмбеддинг с использованием той же модели, что и при загрузке данных.
2. Векторная база данных выполняет поиск наиболее семантически схожих эмбеддингов.
3. Ретривер извлекает лучшие совпадения и передает их в LLM для генерации ответа.

LangChain предоставляет несколько методов извлечения, например ретривер, основанный на поиске по схожести (`similarity`):

```
from langchain.vectorstores import Chroma
     retriever = vector_db.as_retriever(search_type="similarity", search_
kwargs={"k": 5})
     retrieved_docs = retriever.get_relevant_documents(query)
```

Ретриверы обеспечивают возврат только наиболее релевантных частей документа, благодаря чему LLM работает с точным, качественным контекстом, а не с необработанными данными.

6.4.4. Управление задержками

Универсального механизма управления задержками пока не существует. Долгое время отклика раздражает пользователей, и следующие методы могут помочь вам в борьбе с этим:

- *Используйте качественное хранилище векторов с эффективным поиском.* Библиотека Facebook AI Similarity Search (FAISS) позволяет быстро находить

схожие эмбеддинги. Существует множество специальных векторных баз данных, таких как Chroma, Milvus, Pinecone и Weaviate, и с каждым днем их число растет. Традиционные базы данных и поисковые системы, такие как, например, Elasticsearch, предоставляют плагины для векторного поиска. Оба варианта имеют свои преимущества и могут быть выбраны в зависимости от ваших потребностей — масштабируемости, функциональности, эффективности и стоимости.

- *Предобрабатывайте и очищайте свой датасет.* Наличие нескольких версий одного и того же документа увеличивает время поиска и снижает качество результатов.
- *Уведомляйте пользователя о задержке.* Вежливое сообщение по типу «Пожалуйста, подождите, мне нужна еще одна минутка» может успокоить пользователя и частично нивелировать негативный эффект от задержки.
- *Используйте потоковую генерацию ответа, чтобы пользователь видел текст по мере создания каждого токена.* LLM может тратить на создание ответа от 1,5 до 5 секунд и более, на поиск — от 5 до 10 секунд. Если система разговорного ИИ будет ждать завершения генерации, то пользователь может подумать, что чат-бот завис.
- *Рассмотрите возможность кэширования.* Кэширование векторной базы данных и истории чата конкретного пользователя позволяет локально хранить часто запрашиваемую информацию, связанную с его взаимодействиями. Это избавляет от необходимости каждый раз генерировать ответы с нуля, тем самым экономя вычислительные ресурсы. И хотя кэширование будет потреблять дополнительные токены, ради эффективности системы стоит пойти на компромисс.

6.4.5. Когда использовать резервный механизм и когда выполнять поиск

Решение о том, использовать ли ответ RAG или перенаправить пользователя к оператору-человеку, зависит от нескольких ключевых факторов. Так, например, этап генерации RAG не должен вызываться, если этап поиска не дал удовлетворительных результатов. В таком случае система разговорного ИИ может корректно завершить обработку запроса и предложить пользователю перевести его на оператора. Отказ от генерации ответа на основе некачественных результатов поиска позволяет сократить задержку и повысить скорость реакции, тем самым экономя вычислительные ресурсы.

На рис. 6.10 показан процесс принятия решения о том, стоит ли использовать ответ RAG или же лучше передать запрос оператору:

1. Разговорный ИИ обрабатывает запрос пользователя и передает его системе поиска, которая ищет релевантную информацию в базе знаний.
2. Результаты поиска оцениваются на предмет их пригодности для генерации ответа.

3. Если результаты признаны подходящими, система генерирует ответ с помощью RAG.

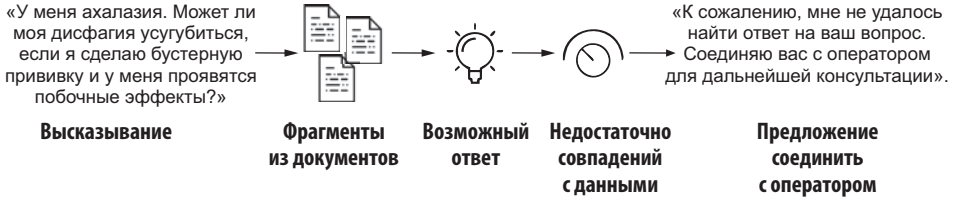


Рис. 6.10. Когда ответ, сгенерированный RAG, не соответствует извлеченным фрагментам, целесообразнее предложить помощь оператора

4. Если результаты поиска нерелевантны (например, их недостаточно или они отсутствуют вовсе), система вежливо предлагает передать запрос оператору.
5. Ответ (сгенерированный системой или переданный оператору) возвращается пользователю.

6.5. Оценка и анализ эффективности RAG

Оценка возможностей модели RAG в рамках разговорного ИИ является многогранной задачей, требующей анализа каждой составляющей. В большинстве случаев мы будем учитывать три аспекта:

- *Насколько ответ корректен?* Удалось ли модели ответить непосредственно на вопрос пользователя? Например, если пользователь попросил о сбросе пароля, ответ должен четко описывать соответствующие шаги, а не обсуждать общие вопросы безопасности аккаунта.
- *Соответствует ли ответ контексту пользователя?* Учтена ли конкретная ситуация или история взаимодействия с клиентом? Например, если пользователь ранее уже сообщал о проблеме с аккаунтом, система должна предложить индивидуальное решение, а не общий совет.
- *Основан ли ответ на документах (или же содержит галлюцинацию, то есть вымышлен)?* Отражает ли он достоверную информацию, извлеченную из исходных документов, без добавления несуществующих деталей? Например, если речь идет о политике компании, ответ должен ссылаться на фактический документ, а не на выдуманные положения.

Для повышения качества оценки LLM могут самостоятельно выставлять баллы своим ответам по ключевым критериям. Такой подход наиболее эффективен при сочетании автоматического анализа и экспертной проверки: LLM формирует предварительную оценку, а эксперт проверяет точность и уместность ответов в конкретном контексте.

Данные критерии оценки позволяют определить достоверность ответов чат-бота. Если исходные документы корректны, а система RAG извлекла именно те данные, которые были необходимы, то и сгенерированные ответы будут точными. Оценка ответов может перейти в анализ эффективности отдельных компонентов RAG, таких как качество индекса документов, эффективность механизма поиска и точность генерации ответа.

6.5.1. Метрики индексации

Метрики индексации позволяют оценить, насколько эффективно система организует, хранит и извлекает большие объемы данных. Основные показатели включают скорость индексации, требования к хранению, масштабируемость и способность обрабатывать высокоразмерные данные, например векторные представления. Таблица 6.3 обобщенно отражает все эти моменты и приводит соответствующие примеры.

Таблица 6.3. Ключевые метрики, влияющие на эффективность и точность индексации документов в системе RAG

Аспект	Краткое описание	Пример
Метрики индексации	Оценивают скорость, объем хранения и масштабируемость. Критически важны для систем, работающих с большими объемами данных	Индексация новостного агрегатора, где важны скорость и масштаб
Эффективность векторной базы данных	Измеряет способность работать с высокоразмерными данными	Службе техподдержки необходимо быстро искать конкретные неисправности в нескольких документах сразу (например: «Почему мое устройство перегревается?»)
Коэффициент полноты	Отражает точность поиска релевантных данных. Высокая полнота важна для полного охвата нужной информации	В поиске юридических документов высокий коэффициент полноты гарантирует нахождение всех релевантных дел
Сложность запроса	Влияет на эффективность системы в зависимости от параметров запроса, размерности данных и их разнообразия	Финансовые базы данных, обрабатывающие сложные запросы по множеству параметров
Инструменты бенчмаркинга	Такие инструменты, как ANN-Benchmark, позволяют сравнивать алгоритмы по показателям полноты и QPS (запросов в секунду)	Поиск алгоритма для системы видеопоиска, сочетающего скорость с точностью

Первым важным компонентом анализа являются метрики индексации, позволяющие оценить, насколько эффективно организованы данные и доступ к ним в базе знаний системы. Сюда относятся такие параметры, как скорость индексации, объем хранения данных и масштабируемость процесса. Эффективная индексация играет

ключевую роль в работе RAG, поскольку напрямую влияет на скорость и точность извлечения информации. Низкая эффективность индексации приводит к замедлению отклика и ошибкам при поиске данных, что снижает качество ответов.

Эффективность векторной базы данных — еще один важный показатель, отражающий способность системы хранить и извлекать высокоразмерные векторы, представляющие сложные данные (текст, изображения, эмбеддинги). Поскольку такие базы используют приближенный поиск, а не точное совпадение, их необходимо оценивать по расширенным метрикам: не только по числу запросов в секунду (QPS) и задержке, но и по *коэффициенту полноты* (recall rate). Если векторная база данных работает эффективно, модель RAG получает доступ к высококачественной, релевантной информации, что обеспечивает точные и контекстно уместные ответы. В противном случае возможны задержки при извлечении данных и использование нерелевантных или недостаточно полезных фрагментов, что снижает качество генерации.

Рассмотрим пример использования чат-бота службы поддержки, работающего на базе RAG. При высоком коэффициенте полноты бот получает широкий спектр информации из базы знаний, что позволяет эффективно решать запросы клиентов, тем самым повышая их удовлетворенность. При низком значении этого показателя важные данные могут быть пропущены, что приведет к некорректным ответам и раздражению пользователей. Таким образом, эффективность чат-бота напрямую зависит от способности полноценно извлекать релевантную информацию. Это подчеркивает значимость высокого коэффициента полноты в автоматизированных системах поддержки.

Сложность запроса, зависящая от таких факторов, как размерность, точность формулировки и разнообразие данных, также влияет на эффективность векторной базы данных. Запросы высокой размерности требуют больше вычислительных ресурсов, так как операции вычисления расстояний между векторами становятся более сложными. Это может привести к увеличению времени отклика и расходу памяти. Более конкретные запросы, ориентированные на узкие сегменты векторного пространства, усложняют задачу индексации и извлечения релевантных векторов, особенно в крупных датасетах. Высокая сложность запросов перегружает систему, тем самым увеличивая ее время отклика и вероятность выдачи нерелевантных ответов. Сложные запросы затрудняют отделение релевантных результатов от нерелевантных.

Например, чат-бот интернет-магазина, помогающий пользователям находить товары по сложным запросам с несколькими параметрами (бренд, цвет, размер, рейтинг), сталкивается со следующей задачей. Пользователь вводит запрос: «Кроссовки Adidas синего цвета, 37 размер, рейтинг не ниже 4 звезд». Запрос сложный, так как он включает несколько измерений, каждое из которых представлено отдельным вектором в базе данных.

Инструменты бенчмаркинга, такие как ANN-Benchmarks и VectorDBBench, позволяют оценивать рассмотренные выше параметры, сравнивая разные алгоритмы

и конфигурации. Это способствует созданию надежной архитектуры поиска, лежащей в основе системы RAG и гарантирующей высокое качество генерируемого контента. ANN-Benchmarks отображает коэффициент полноты по оси X и QPS по оси Y, иллюстрируя эффективность алгоритмов при разных уровнях точности поиска. VectorDBBench визуализирует данные показатели раздельно.

6.5.2. Метрики поиска

Следующий шаг оценки — это метрики поиска, позволяющие измерить способность системы находить релевантную информацию в индексированных данных. Основные показатели включают долю верных ответов среди найденных (accuracy), показатели точности и полноты, а также время отклика. Эффективные метрики поиска гарантируют, что пользователи получают релевантные ответы, тем самым повышая уровень их удовлетворенности и доверия к системе разговорного ИИ. Таблица 6.4 содержит эти ключевые параметры и иллюстрирует их примерами.

Таблица 6.4. Ключевые аспекты, влияющие на метрики поиска

Аспект	Краткое описание	Пример
Доля верных ответов (retrieval accuracy)	Оценивает способность системы находить релевантную информацию в индексированных данных	Чат-бот успешно находит правильные инструкции по устранению неполадок в большом датасете
Точность и полнота	Точность отражает релевантность найденных документов, а полнота показывает общее число найденных релевантных документов	Баланс между точностью и полнотой при выдаче рекомендаций товаров в чат-боте интернет-магазина
Контекстная точность и контекстная полнота	Специфично для RAG: контекстная точность оценивает релевантность, контекстная полнота — полноту охвата информации	Оценка того, насколько точно сгенерированный ответ чат-бота службы поддержки соответствует контексту запроса
Оптимизация параметров	Настройка параметров поиска и алгоритмов для повышения скорости, доли верных ответов и релевантности результатов	Настройка кластеров FAISS или параметров Elasticsearch для улучшения поиска документов в юридических базах данных
Модели эмбеддингов	Использование различных эмбеддингов влияет на качество поиска, повышая точность или полноту	Использование модели BERT для точного понимания контекста в чат-боте, предоставляющем юридические консультации
Фильтрация и повторное ранжирование	Удаление шума и изменение порядка результатов для повышения доли релевантных и верных ответов	Исключение нерелевантных статей из новостного чат-бота и повторное ранжирование наиболее подходящих результатов
Нормализованный дисконтированный совокупный прирост (NDCG)	Оценивает качество ранжирования с учетом релевантности и позиции документов в выдаче	Гарантирует, что наиболее подходящие статьи появляются первыми в чат-боте техподдержки

Эти метрики позволяют оценить качество поиска, релевантность документов и степень соответствия ответов пользовательским запросам. *Доля верных ответов* (retrieval accuracy) показывает, насколько эффективно система поиска способна находить наиболее релевантную информацию в индексированных данных. Если показатель низкий, чат-бот может выдавать ответы, лишь отдаленно связанные с запросом, или вовсе не находить важные сведения.

Точность (precision) отражает долю релевантных документов среди всех найденных, тогда как полнота (recall) показывает, сколько релевантных документов удалось найти системе. Высокий показатель полноты означает, что система находит большинство релевантных документов в базе данных. Высокая точность указывает на то, что найденные документы в основном действительно соответствуют запросу пользователя. Баланс между точностью и полнотой крайне важен для обеспечения полного охвата нужной информации при минимальном количестве нерелевантных результатов.

Для моделей RAG существуют специальные метрики, такие как контекстная точность и контекстная полнота, которые оценивают степень соответствия и полноту охвата информации в сгенерированных ответах относительно запроса пользователя. Контекстная точность измеряет, насколько точно найденный контекст совпадает с запросом, что указывает на релевантность и корректность ответа. Контекстная полнота оценивает, насколько полно сгенерированный ответ охватывает релевантную информацию, полученную из найденного контекста.

Для повышения эффективности поиска можно использовать следующие стратегии:

- оптимизация параметров поиска;
- использование различных моделей эмбеддингов;
- внедрение фильтрации и повторного ранжирования результатов.

Первая стратегия заключается в настройке параметров поиска с целью повышения его скорости и доли верных результатов. Регулировка таких параметров, как количество кластеров в FAISS или сложность поискового запроса в Elasticsearch, может значительно улучшить показатели точности и полноты. Это обеспечивает возврат наиболее релевантных документов (повышая точность) и одновременное извлечение всех значимых данных (повышая полноту).

Оптимизация параметров также сокращает время отклика системы, улучшая ее отзывчивость. Настройка алгоритмов индексации и запроса позволяет ускорить процесс поиска информации без уменьшения доли правильных ответов.

Вторая стратегия заключается в применении различных моделей эмбеддингов с целью подбора наиболее эффективной комбинации для поиска релевантных документов. Эмбеддинги оказывают существенное влияние на качество поиска. Например, модели BERT и Sentence-BERT улучшают способность системы

понимать контекст и семантику пользовательских запросов, тем самым повышая точность. Эмбеддинги на основе GPT-3 обеспечивают более широкое восприятие контекста, улучшая показатель полноты за счет извлечения более разнообразных релевантных данных.

Третья стратегия — это использование фильтрации и повторного ранжирования. Эти подходы дополнительно повышают метрику точности за счет исключения нерелевантных документов и улучшают показатель полноты, отдавая приоритет релевантным результатам. Применение специальных фильтров по предметной области или контекстно-зависимого повторного ранжирования может значительно повысить качество конечных результатов поиска.

Метрика NDCG (Normalized Discounted Cumulative Gain, нормализованный дисконтированный совокупный прирост) используется в случаях, когда порядок документов имеет значение. Она измеряет эффективность алгоритмов ранжирования, оценивая как релевантность, так и позицию документов в выдаче, что обеспечивает комплексную оценку качества ранжирования.

6.5.3. Метрики генерации

Последняя область оценки касается самой генерации — насколько сгенерированный ответ является релевантным и полным. Таблица 6.5 содержит краткий обзор ключевых метрик, стратегий повышения качества генерации и примеры их практического применения с подробными объяснениями.

Таблица 6.5. Ключевые аспекты, влияющие на метрики генерации

Аспект	Краткое описание	Пример
Достоверность (faithfulness)	Оценивает фактическую долю верных результатов сгенерированного вывода на основе найденного контекста	Обеспечение фактической корректности ответов чат-бота на вопросы. Например: «Что будет, если нарушить условия договора?»
Релевантность ответа	Оценивает, насколько сгенерированный ответ релевантен конкретному пользовательскому запросу	«Каков статус моего заказа?» — чат-бот после поиска соответствующих данных отвечает: «Ваш заказ №12345 в настоящее время находится в пути и будет доставлен 15 августа». Такой ответ напрямую связан с вопросом пользователя, предоставляя ему конкретную информацию о статусе заказа без лишних деталей
Дообучение	Улучшает генерацию за счет согласования LLM с данными конкретной предметной области, повышая долю верных результатов и их релевантность	Дообучение LLM для юридических консультаций обеспечивает правильность и юридическую достоверность сгенерированных ответов

Таблица 6.5 (*окончание*)

Аспект	Краткое описание	Пример
Промпт-инжиниринг	Создание промптов для направления LLM к генерации более контекстно уместных и релевантных ответов	Применение методик промпт-инжиниринга позволяет медицинскому чат-боту выдавать четкие и уместные рекомендации
Смешивание моделей (блендинг)	Комбинирование моделей для повышения качества генерации и достижения баланса между правильностью и «живостью» ответов	Смешивание модели, ориентированной на поиск, с моделью, ориентированной на язык, для создания правильных и при этом естественных ответов
Sensibleness and Specificity Average (SSA)	Оценивает качество ответов в чат-ботах с широкой предметной областью, что обеспечивает разумность и конкретность ответов	Оценка ответов чат-бота с широкой предметной областью позволяет убедиться, что ответы имеют смысл и не слишком расплывчаты
Faithfulness-Evaluator	Оценивает, не содержит ли сгенерированный ответ галлюцинаций, сопоставляя его с найденным контекстом	Применение FaithfulnessEvaluator для обеспечения того, чтобы ответы финансового чат-бота основывались на найденных документах

Два основных показателя для оценки эффективности LLM — это достоверность (faithfulness) и релевантность (relevancy) ответа. Достоверность оценивает фактическую правильность ответа на основе найденного контекста; релевантность оценивает, насколько ответ соответствует заданному вопросу. Ответ может быть фактически точным (достоверным), но плохо согласованным с вопросом (менее релевантным), либо верным, но не основанным на исходных данных.

Для улучшения метрик генерации можно использовать несколько стратегий. Дообучение LLM на данных конкретной предметной области позволяет повысить достоверность и релевантность, согласуя модель с контекстом ее применения. Это обеспечивает правильность и актуальность сгенерированных ответов для конкретной области. Не стоит забывать и о промпт-инжиниринге — включение необходимого контекста или ограничений в промпт делает процесс генерации более целенаправленным и согласованным с намерением пользователя. Кроме того, можно использовать методику смешивания (блендинга) моделей. Комбинирование нескольких моделей, каждая из которых специализируется на определенном аспекте задачи, позволяет повысить качество генерации. Например, одна модель может лучше справляться с поиском точной информации, а другая — с созданием «живой» и контекстно уместной речи. Объединение таких моделей поможет достичь более сбалансированного и эффективного результата.

Метрика Sensibleness and Specificity Average (SSA) оценивает качество ответов в чат-ботах с широкой предметной областью. Sensibleness (разумность) гарантирует, что ответы логически связаны с контекстом, а specificity average (степень

конкретности) обеспечивает их полноту без расплывчатости. Изначально данные показатели определялись людьми.

И хотя важно избегать расплывчатых ответов, столь же важно предотвращать галлюцинации LLM. LlamaIndex разработал метрику FaithfulnessEvaluator для измерения галлюцинаций, оценивающую, соответствует ли ответ найденному контексту. LlamaIndex был создан для решения задачи интеграции LLM с различными источниками данных, чтобы модели могли получать доступ, выполнять запросы и извлекать знания из структурированных и неструктурированных данных. Это фреймворк для построения контекстно-дополненных генеративных ИИ-приложений. Он предоставляет возможности интеграции данных, их индексации, обогащения, обработки запросов и многого другого. Мы рекомендуем регулярно следить за последними обновлениями LlamaIndex.

Разговорный ИИ, интегрированный с RAG, способен эффективно обрабатывать редкие и сложные запросы. Однако для достижения этих преимуществ требуется постоянный мониторинг всех компонентов, особенно тех, где наиболее часто возникают ошибки, например, на этапах поиска и генерации.

6.5.4. Сравнение эффективности решений для индексации и эмбеддингов в RAG

В предыдущих разделах мы рассматривали факторы, влияющие на эффективность систем RAG. Компоненты индексации и эмбеддингов играют ключевую роль в эффективном поиске релевантных документов и обеспечении отзывчивости системы. При внедрении RAG в систему разговорного ИИ важно оценить ее влияние на точность поиска и релевантность ответов. Бенчмаркинг является ключевым условием создания эффективной системы RAG. Необходимо иметь метод оценки того, повышают ли изменения в системе долю совпадений запросов пользователей с результатами. Улучшение составляет 1 %, 2 % или больше? Это принципиально важно для понимания, действительно ли ваша система RAG эффективна.

Кроме того, без надлежащего мониторинга, валидации и оценки будет сложно доказать эффективность системы. Системы RAG по своей природе сложны и обычно показывают 50–60 % верных результатов. Для практического применения необходимо повысить данный показатель до более чем 80 %.

PharmaBot, изначально разработанный как чат-бот для ответов на общие вопросы о COVID-19 (например, информация о вакцине и записи на прием), теперь планируется дополнить RAG. Цель состоит в том, чтобы расширить возможности PharmaBot для ответов на более сложные вопросы, такие как «Можно ли принимать ибупрофен вместе с лекарством от давления?» и «После вакцинации болят руки. Что делать?» Сначала необходимо выбрать датасет, включающий медицинские статьи, научные публикации и рекомендации медицинских организаций, посвященные различным заболеваниям и взаимодействиям лекарств

и вакцин. Затем формируется набор репрезентативных запросов, подобных приведенным выше, которые, вероятнее всего, появятся у пользователей.

Далее выбираются решения для индексации и эмбеддинги. Для индексации рассматриваются, например, FAISS и Elasticsearch. Наиболее популярным источником последних бенчмарков эффективности моделей текстовых эмбеддингов являются рейтинги MTEB, размещенные на платформе Hugging Face. И хотя MTEB предоставляет полезную информацию, указанные там результаты являются самоотчетными (то есть были заявлены и опубликованы самими разработчиками), и многие модели могут выдавать меньше верных результатов на реальных данных. Стоит также рассмотреть такие модели, как BERT, Sentence-BERT и GPT-3, поскольку они широко применяются на практике. После этого проводится генерация эмбеддингов для нашего контента и их индексация с использованием выбранных методов. Затем выполняются тестовые запросы и измеряется эффективность.

При оценке работы PharmaBot, усовершенствованного с помощью RAG, можно использовать табл. 6.6 для определения релевантных метрик, установления базовых показателей и целевых значений. Это позволит оценить улучшения, внесенные различными комбинациями решений для индексации (FAISS, Elasticsearch) и эмбеддингов (BERT, Sentence-BERT, GPT-3), а также сравнить эти решения. Например, можно сравнить, как различные комбинации компонентов RAG соотносятся с бизнес-целями.

Таблица 6.6. Приоритетные метрики для оценки RAG на основе бизнес-целей

Бизнес-цель	Приоритетные метрики	Причина	Пример
Удовлетворенность клиентов	Правильность ответов (response accuracy). Релевантность	Влияет непосредственно на пользовательский опыт и удовлетворенность	PharmaBot предоставляет пользователям верные ответы на их вопросы
Операционная эффективность	Задержка. Пропускная способность	Обеспечивает способность системы быстро обрабатывать большой объем запросов	Чат-бот службы поддержки крупного интернет-магазина должен иметь минимальную задержку, чтобы предоставлять быстрые ответы во время пиков покупательской активности
Масштабируемость	Количество запросов, обрабатываемых в секунду. Использование ресурсов	Оценивает эффективность системы при увеличении нагрузки	Чат-бот медицинского учреждения успешно справляется с сезонными всплесками количества записей на прием и пользовательских запросов

Бизнес-цель	Приоритетные метрики	Причина	Пример
Экономическая эффективность	Использование CPU. Использование памяти	Обеспечивает высокую эффективность без чрезмерного потребления ресурсов	Чат-бот некоммерческой организации предоставляет круглосуточную психологическую поддержку при ограниченном финансировании

Можно составить сравнительную таблицу, подобную табл. 6.7. Приведенные числовые значения являются условными и должны быть скорректированы в соответствии с фактически выбранными компонентами и результатами бенчмаркинга. Например, одна из конфигураций может включать использование Elasticsearch и трех различных моделей эмбеддингов для получения данных.

Таблица 6.7. Оценка различных конфигураций PharmaBot

Метрики		Конфигурация 1	Конфигурация 2	Конфигурация 3
Правильность ответов (response accuracy)	Полнота	0,85	0,87	0,88
	Точность	0,75	0,77	0,78
	Оценка F1	0,8	0,82	0,83
Релевантность	Среднеобратный ранг (MMR)	0,70	0,72	0,74
	Средняя точность	0,65	0,68	0,70
Задержка	Средняя задержка (мс)	50	55	60
Пропускная способность	Запросов в секунду	20	18	16
Использование ресурсов	Использование CPU (%)	70	65	90
	Использование памяти (Гбайт)	8	7	12

Чтобы PharmaBot посредством внедрения RAG удовлетворил определенные бизнес-требования, необходимо учитывать ряд вынужденных компромиссов, основанных на метриках оценки. Компонент индексации в сочетании с выбранными компонентами эмбеддингов обеспечивает правильность и релевантность ответов при низкой задержке и высокой пропускной способности, и поэтому они подходят для систем, требующих быстрых и верных ответов. Однако подобные комбинации демонстрируют средний или даже высокий уровень использования ресурсов, что может привести к увеличению операционных затрат. И напротив, интеграция другого компонента эмбеддингов с различными стратегиями индексации обеспечит бóльшую правильность и релевантность ответов, но ценой значительно большей задержки и меньшей пропускной способности из-за

вычислительной сложности выбранных эмбеддингов. Это может повлиять на способность системы эффективно обрабатывать большой объем запросов. Высокая загрузка CPU и использование памяти также могут привести к повышенной нагрузке на ресурсы и увеличению операционных затрат. В конечном счете выбор оптимальной комбинации зависит от баланса между долей верных результатов и релевантностью, возможностями системы по эффективной обработке запросов и потреблением ресурсов для контроля затрат.

Также следует учитывать необходимость регулярной переоценки и доработки системы RAG. Таблица 6.7 не является полной, а лишь передает ключевую идею — выбор основных метрик оценки и использование структурированного подхода к их анализу. Проводя регулярные тесты и уделяя внимание отдельным компонентам RAG, вы в итоге получите надежный пайплайн оценки вашей системы. Главная цель заключается в достижении восходящей тенденции, как показано на рис. 6.11. Последовательное применение стратегий, рассмотренных в данной главе, а также анализ их результатов позволят установить влияние различных конфигураций на эффективность RAG. Некоторые настройки приводят к значительным улучшениям, что подчеркивает важность экспериментов в процессе оптимизации. Не существует единственно верного подхода — для более точной настройки системы RAG важно рассматривать разные альтернативы.

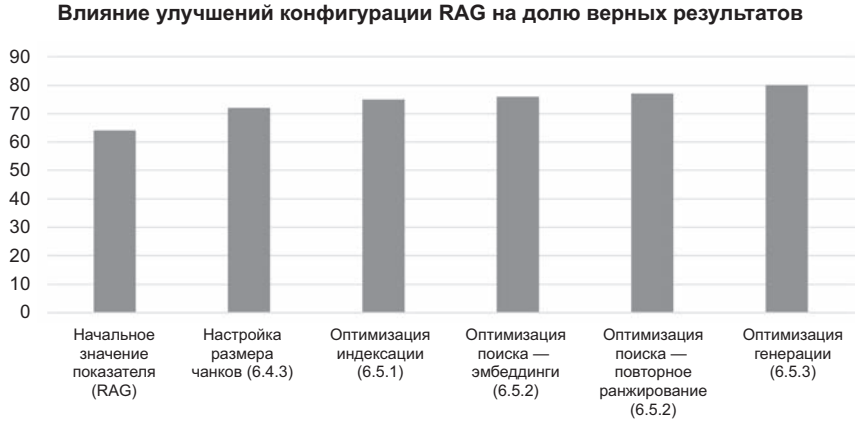


Рис. 6.11. Различные конфигурации и оптимизации повышают долю верных результатов (accuracy) системы RAG

Кроме того, можно использовать фреймворки для оценки RAG — от платных коммерческих предложений до инструментов с открытым исходным кодом. Оптимальное решение должно поддерживать баланс между простотой сопровождения, эксплуатационной нагрузкой и применимостью метрик, отслеживаемых инструментом, к вашей архитектуре RAG и бизнес-целям. Ниже приведено несколько актуальных примеров, но постоянно появляются новые платформы, предоставляющие еще больше возможностей:

- *Arize* — платформа для мониторинга моделей, ориентированная на показатели точности, полноты и оценки F1. Она полезна в сценариях, требующих постоянного отслеживания эффективности, и обеспечивает соответствие систем RAG заданным порогам доли верных результатов в приложениях, работающих в режиме реального времени. Arize — это коммерческое решение с поддержкой и регулярными обновлениями для корпоративного использования.
- *RAGAS* — инструмент с открытым исходным кодом, предлагающий упрощенную оценку без необходимости эталонных данных. Основное внимание уделяется средней точности (AP) и таким пользовательским метрикам, как достоверность. Он оценивает, насколько сгенерированный контент соответствует предоставленному контексту, и подходит для первичной оценки или случаев, когда эталонные данные отсутствуют.

Задания

1. Оцените релевантность ответов, генерируемых моделью RAG в рамках системы разговорного ИИ:
   - Определите критерии оценки релевантности ответов, создаваемых моделью RAG.
   - Разработайте систему балльной оценки для количественного измерения релевантности на основе таких факторов, как семантическое сходство и информативность.
   - Подготовьте набор пользовательских запросов, которые будут оцениваться.
   - Составьте набор ожидаемых ответов (вручную).
   - Сравните сгенерированные ответы с подготовленными, чтобы определить уровень релевантности.
   - Рассчитайте значения точности, полноты и оценки F1 для количественного анализа эффективности модели RAG при генерации релевантных ответов.
   - Проанализируйте результаты оценки, чтобы выявить закономерности или области, где модель справляется и не справляется с генерацией релевантных ответов.
   - Установите возможные факторы, влияющие на релевантность ответов, и на их основе предложите стратегии повышения эффективности RAG.
2. Оцените соответствие ответов RAG документам-источникам:
   - Сгенерируйте ответы на пользовательские запросы с помощью модели RAG и определите исходные документы или фрагменты, из которых были получены ответы.
   - Оцените степень соответствия, сравнив релевантность исходных документов или их частей с соответствующими пользовательскими запросами.
   - Разработайте систему оценки для количественного анализа эффективности модели RAG на основе таких факторов, как релевантность и полнота охвата документов.

Итоги

- Традиционные чат-боты, основанные на намерениях, могут быть значительно улучшены за счет интеграции в них поиска.
- Намерения подходят для ответов на наиболее распространенные короткие вопросы, тогда как поиск эффективен для длинных и более сложных запросов.
- Традиционный поиск возвращает ссылки или цитаты из документов вместо прямого ответа.
- RAG расширяет возможности поиска, генерируя ответы на основе найденных документов.
- С помощью RAG чат-боты могут предоставлять контекстно релевантные ответы в режиме реального времени, снижая уровень неудовлетворенности пользователей и улучшая качество взаимодействия. Привязка ответов к сфере деятельности компании также упрощает сопровождение и дополнение системы намерений для разработчиков.
- При внедрении RAG необходимо учитывать возможные проблемы — от управления задержками до обеспечения резервных механизмов и передачи диалога оператору для предотвращения галлюцинаций.
- Оценка системы RAG должна проводиться с точки зрения различных компонентов: индексация, поиск и генерация.

7 Дополнение намерений с помощью генеративного ИИ

В этой главе

- ✓ Создание новых обучающих и тестовых примеров с помощью генеративного ИИ
- ✓ Выявление пробелов в текущих данных разговорного ИИ
- ✓ Использование LLM для построения новых намерений в разговорном ИИ

Самая неприятная проблема, с которой сталкиваются пользователи разговорных ИИ, — это случаи, когда система их не понимает, особенно если это происходит неоднократно! Такое случается во всех типах разговорного ИИ: в Q&A-ботах, процессно-ориентированных ботах и в маршрутизирующих агентах. Существует несколько стратегий повышения способности ИИ к пониманию. Первая из них — улучшение обучения намерений вручную (глава 5), что обеспечивает полный контроль со стороны разработчика, однако требует значительных временны́х затрат и специальных навыков. Вторая стратегия — генерация, дополненная поиском (RAG, глава 6), передающая данную задачу генеративному ИИ, тем самым сокращая участие человека. В этой главе мы рассмотрим гибридный подход, при котором генеративный ИИ дополняет усилия разработчика. Он применим как к системам, основанным на правилах, так и к системам, использующим генеративный ИИ.

Использование генеративного ИИ в качестве некой «музы» разработчика сокращает объем и время его работы, расширяет тестовые данные, доступные для

анализа, и при этом оставляет окончательное принятие решения за человеком, что препятствует галлюцинациям (случаям, когда ИИ выдает правдоподобные, но неверные утверждения).

Предположим, вы создаете решение на основе разговорного ИИ для службы техподдержки. Побеседовав с персоналом, вы установили, что самая частая задача, которую система должна решать, — это сброс пароля. Следовательно, ИИ должен хорошо понимать намерение, связанное со сбросом пароля.

Так как решение на основе разговорного ИИ новое, у вас еще нет реальных пользовательских высказываний для обучения. Когда вы спрашиваете сотрудников службы поддержки, как пользователи чаще всего начинают диалог, то слышите: «Обычно они пишут что-то вроде “забыл пароль” или “не могу войти в систему”». К такому ограниченному описанию стоит отнестись с долей недоверия — наверняка в запросах встречается и более разнообразная лексика, но пока что вам сложно представить, какая именно. Здесь и поможет генеративный ИИ.

Рассмотрим, как разработчик и генеративный ИИ могут работать сообща.

7.1. Начало работы

Большие языковые модели (LLM) способны выполнять множество технических задач, включая классификацию и ответы на вопросы. Именно эти задачи и лежат в основе разговорного ИИ. Так почему бы просто не использовать генеративный ИИ для выполнения основных задач разговорного ИИ?

LLM универсальны, поскольку обучены на огромных объемах данных. Это позволяет использовать их для множества задач, но есть и оборотная сторона медали. Каковы же недостатки использования LLM в качестве классификатора в разговорном ИИ?

- *Финансы* — использование LLM может быть дорогостоящим.
- *Скорость* — из-за обработки миллиардов параметров LLM могут работать медленнее (временны́е затраты).
- *Репутационные риски* — высокая обобщающая способность LLM приводит к появлению галлюцинаций, которые могут снизить доверие к вашему боту или даже вызвать юридические риски.
- *Отсутствие прозрачности и объяснимости* — LLM часто представляют собой некий «черный ящик».

Разговорный ИИ, в свою очередь, использует специализированные технологии. Классификатор, обученный исключительно под конкретную задачу, обходится дешевле и работает быстрее, так как обрабатывает меньше параметров. И хотя доля верных результатов в этом случае может быть ниже, система гарантированно использует контролируемый набор ответов. Данные различия обобщенно представлены в табл. 7.1.

Таблица 7.1. Сравнение традиционной обработки естественного языка (NLP) в разговорном ИИ и генеративного ИИ при выполнении задачи классификации

Характеристика	Традиционная NLP	Генеративный ИИ
Модель	Специализирована и оптимизирована для одной задачи — классификации	Универсальная модель, хорошо справляющаяся со многими задачами
Скорость выполнения	Быстрая	Медленная
Стоимость выполнения	Низкая	Высокая
Доля верных результатов	В основном ответы верные (обучена вами на небольшом объеме данных)	В основном ответы верные (предобучен на большом объеме данных)
Масштабируемость	До 100 намерений, после этого становится чрезмерно сложной	Хорошо обобщает с использованием подхода RAG
Контролируемость	Полностью контролируется человеком, требует тщательного тестирования	Склонна к галлюцинациям при полной автономности, галлюцинации трудно обнаружить автоматически

Гибридный подход позволит объединить преимущества обоих методов.

7.1.1. Зачем это нужно: преимущества и недостатки

LLM может значительно сэкономить время и усилия разработчика. LLM и человек могут плодотворно сотрудничать, работая как партнеры. Обучение классификатора разговорного ИИ требует участия человека, но оно также требует данных, которые не всегда легко собрать. Иногда их можно получить только после развертывания разговорного ИИ в продакшене. Даже в простом примере с определением проблемы «забыл пароль» мы по-прежнему не знаем всех способов, с помощью которых пользователи могут сформулировать свой запрос. Они даже могут использовать «неправильные» слова!

LLM особенно полезны в следующих сценариях:

- *Бутстрэппинг* — ИИ сталкивается с проблемой «холодного старта». Как обучать модель, если данных нет? LLM могут сгенерировать начальный набор обучающих данных.
- *Расширение данных* (expanding) — использование LLM для заполнения пробелов в существующих данных, когда их недостаточно для оптимизации точности классификатора. Особенно полезно для понимания редких, но важных намерений (например, сообщений о мошенничестве).
- *Надежное тестирование* — LLM могут генерировать дополнительные тестовые данные, повышая нашу уверенность в устойчивости работы разговорного

ИИ (актуально даже при использовании генеративного ИИ для создания ответов, как в RAG).

LLM позволяют проводить множество экспериментов; некоторые из них дают результаты, которые можно использовать в приложении как обучающие или тестовые данные. Человек и LLM могут дополнять друг друга. Например, LLM предлагает темы и варианты формулировок, которые, вероятно, будут использовать клиенты. Человек выбирает наиболее подходящие и просит модель развить их, дополнив инструкции в промпте или few-shot-примеры.

Взаимодействие с LLM носит итеративный и совместный характер. Как правило, с первой попытки правильно сформулировать промпт не удается. Модель может неверно понять задачу или выдать недостаточно полезный контент. Поэтому следует ожидать, что вам придется провести несколько экспериментов, прежде чем удастся получить качественные результаты. Зато потом вы сможете быстро генерировать высказывания для всех намерений и улучшать понимание запросов пользователей вашим ИИ.

7.1.2. Что для этого нужно

Многие LLM способны помочь в задаче генерации дополнительных обучающих или тестовых данных для намерения «забыл пароль». Получается, мы просто выбираем какую-то одну модель и позволяем ей самостоятельно работать? Не совсем. LLM действительно полезна, но не стоит ожидать, что она все сделает за вас. Необходимо заранее понимать, с чего начать, например, требуется знать, какие недостатки существуют в вашем решении. Также необходимо выбрать LLM, подходящую конкретно для вашего случая.

Доступ к LLM — самое очевидное предварительное условие для ее использования в расширении возможностей разговорного ИИ. Однако при выборе модели следует учитывать несколько неочевидных факторов:

- *Условия использования* — некоторые LLM прямо запрещают использовать их для «создания или улучшения другой модели». Данный пункт направлен на предотвращение разработки систем-конкурентов, но и использование модели для улучшения разговорного ИИ может быть истолковано аналогичным образом. Степень допустимого юридического риска зависит только от вас. (Проконсультируйтесь с юридическим отделом — возможно, там уже знают, какие модели можно использовать в вашей компании.)
- *Конфиденциальность данных* — будет ли модель сохранять ваши данные и использовать их для дальнейшего обучения? Данные, содержащиеся в вашем разговорном ИИ, могут рассматриваться компанией как конфиденциальные. Если это так, то их нельзя передавать первой попавшейся LLM.
- *Возможности* — не каждая модель способна выполнять задачи творческой генерации. Убедитесь, что выбранная вами модель умеет следовать инструкциям.

- *Открытый исходный код или коммерческие предложения* — в большинстве случаев важна объяснимость. Модели с открытым исходным кодом обычно обеспечивают больший доступ к информации о процессе обучения, включая обучающие данные и исходный код модели. Коммерческие решения, как правило, не раскрывают подобные сведения, но зачастую проще в использовании. Данный фактор может повлиять и на соответствие этическим и нормативным требованиям.
- *Задержка и время отклика* — приходится выбирать между скоростью и правильностью ответов: более крупная модель может давать больше верных ответов, но работать медленнее.

В этой главе и в оставшейся части книги мы будем использовать несколько примеров промптов и моделей. Все примеры можно адаптировать под вашу собственную модель (или модели). Не бойтесь экспериментировать с другими моделями, особенно с теми, которые появятся уже после публикации этой книги.

Кроме того, для эффективного взаимодействия с LLM вам понадобится некоторый уровень предметных знаний. Сюда относится информация о проблемах, с которыми обращаются пользователи вашего разговорного ИИ, перечень намерений, которые система должна поддерживать, и пользовательские высказывания, относящиеся к ним. Используйте как можно больше реальных данных, дополняя их с помощью LLM.

7.1.3. Как использовать дополненные данные

LLM может сгенерировать дополнительные данные для вашего чат-бота. Ее применение на этапе разработки снижает риск появления галлюцинаций и их негативное влияние. Так, можно добавлять данные в обучающую выборку, расширять тестовую выборку и модифицировать существующие данные (например, изменять их грамматическую структуру или подбирать синонимы).

Лучшим источником данных являются реальные пользователи системы. Однако тут мы сталкиваемся с известной проблемой «курицы и яйца»: если система еще не запущена в продакшен, данных может не быть вовсе. Классификатор намерений должен обучаться на разнообразных данных, чтобы впоследствии понимать разнообразные пользовательские высказывания. Чат-бот, в свою очередь, должен тестироваться на данных, на которых он не обучался.

При отсутствии обучающих данных обычно генерируются высказывания с низкой вариативностью. У разработчика в голове есть несколько ключевых слов, и он «привязывается» к ним. Даже при десятках примеров такие низковариативные высказывания не несут большого количества информации. Высоковариативные высказывания, напротив, охватывают значительно больше смыслов, как показано в табл. 7.2, и в целом делают классификатор более устойчивым.

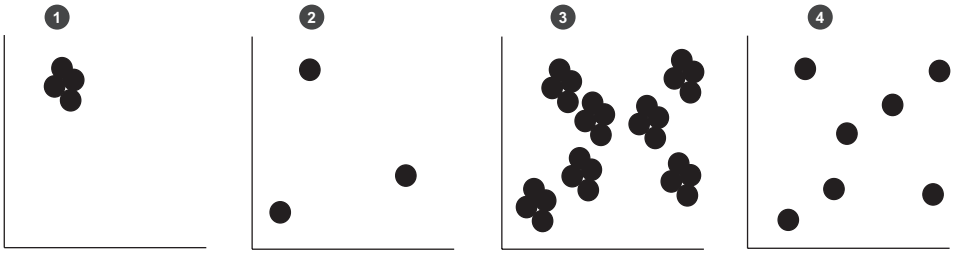
Таблица 7.2. Сравнение низковариативных и высоковариативных высказываний. Высокая вариативность повышает устойчивость классификатора

Низковариативный набор	Высоковариативный набор
• Я забыл свой пароль • Забыл свой пароль • Забыл пароль • Нужна помощь, я забыл пароль	• Я не могу войти • Аккаунт заблокирован • Забыл пароль

Несмотря на меньшее количество примеров, высоковариативные высказывания охватывают все смыслы, содержащиеся в низковариативных. Одного высказывания «забыл пароль» уже достаточно, чтобы предсказать намерение всех четырех низковариативных вариантов. В обратную сторону это не работает, даже если добавить десятки незначительных вариаций фразы «забыл пароль» в низковариативный набор. Фраза «я не могу войти» не находит никаких совпадений в низковариативном наборе.

Предпочтительнее иметь небольшой, но высоковариативный обучающий набор, охватывающий широкий диапазон низковариативных тестовых высказываний. Лучше обучать классификатор на десяти сильных вариантах, чем на ста слабых. Это делает чат-бот более устойчивым к разнообразным формулировкам выражений, которые он встретит в продакшене, и снижает риск дисбаланса обучающих данных (что приводит к снижению понимания).

На рис. 7.1 можно увидеть графическое представление всей информации, передаваемой высказываниями. Первый график отображает низковариативные



1. **Небольшой низковариативный набор охватывает лишь два слова.**
2. **Небольшой высоковариативный набор охватывает несколько разных слов.**
3. **Большой низковариативный набор подходит для тестирования, обеспечивая широкое покрытие входных данных.**
4. **Большой высоковариативный набор подходит для обучения, обеспечивая эффективное покрытие максимальной вариативности.**

Рис. 7.1. Визуализация охвата разных типов наборов. Идеальный обучающий набор — это набор № 4, в котором высокая вариативность охватывается небольшим числом примеров. Набор № 3 лучше всего подойдет для тестирования

высказывания из табл. 7.2. Так как они состоят всего лишь из двух главных слов, высказывания располагаются очень близко друг к другу. Второй график показывает высоковариативные высказывания, которые из-за отсутствия общих слов распределены по всей сетке, но между ними остается слишком много пустого пространства. Третий график иллюстрирует идеальный тестовый набор с широким покрытием входных данных. Четвертый график демонстрирует идеальный обучающий набор, в котором максимальная вариативность охвачена минимальным числом примеров. Тестовый датасет может быть значительно больше обучающего. Желательно, чтобы тестовые данные были разнообразными, но допустимо и наличие почти дублирующихся примеров.

В этой главе мы сначала рассмотрим, как использовать генеративный ИИ для создания высказываний с высокой вариативностью. Затем мы расширим набор этих высказываний за счет множества мелких вариаций. К концу главы вы узнаете, как генерировать высказывания, соответствующие третьему и четвертому графикам.

Задания

1. Представьте, что вы создаете чат-бота для обычного розничного магазина. Составьте десять высказываний для намерения `#store_location`, отражающих вопросы пользователей вроде «Где находится ваш магазин?». Засеките, сколько времени это займет.
2. Составьте еще десять высказываний, но уже не используя слова «где», «магазин», «находится» и «располагается». Снова засеките время. Стали ли высказывания более разнообразными?
3. Повторите два предыдущих задания для намерения `#store_hours`. Сначала используйте любые слова, а затем ограничьте себя и не используйте «когда», «время» и «часы».

7.2. Улучшение существующих намерений

Работа начинается с четкого понимания того, какое намерение необходимо улучшить, — в нашем случае это намерение «забыл пароль». Для того чтобы разговорный ИИ мог корректно распознавать данное намерение, необходимо достаточное количество обучающих данных. Напомним, что наша служба поддержки не знает всех способов, которыми пользователи могут формулировать проблему. Они сказали: «Пользователи обычно пишут что-то вроде “забыл пароль” или “не могу войти в систему”». Однако этого недостаточно, чтобы научить чат-бот стабильно распознавать намерение «забыл пароль».

Решить эту проблему нам поможет LLM. Сначала модель сгенерирует контекстуальные синонимы, чтобы мы поняли весь диапазон возможной лексики, а затем создаст полноценные высказывания с ее использованием. После этого

мы попросим LLM сгенерировать разные грамматические формы: например, вопросы и утверждения, предложения в прошедшем и настоящем времени. В итоге мы сможем использовать знания, полученные при построении одного намерения («забыл пароль»), в создании следующего («найти магазин»).

Начнем с самого простого шага — подбора синонимов.

Можно ли использовать другую LLM?

Да! В нашей книге мы будем использовать несколько моделей. Сфера генеративного ИИ постоянно развивается, и используемые нами модели могут быть заменены более актуальными решениями, которые появятся к моменту публикации книги или ее прочтения. Принципы, которые мы демонстрируем, куда важнее, чем конкретные модели.

7.2.1. Творческий подход к подбору синонимов

Первый шаг — сформулировать хороший промпт. Это гарантирует, что LLM правильно поймет, что именно от нее требуется, и первой задачей в нашем примере будет генерация ряда синонимов. Процесс написания промпта не обойдется без экспериментов.

Наши специалисты в предметной области отметили, что пользовательские высказывания часто содержат фразу «забыл пароль». Один из способов повысить устойчивость чат-бота — убедиться, что он охватывает самые разные именные и глагольные словосочетания. Попросим LLM сгенерировать возможные синонимы-существительные.

Как настроить и запустить LLM?

Есть несколько вариантов настройки среды LLM. Вы можете запускать LLM локально на своем компьютере с помощью инструмента Ollama или использовать коммерческие облачные платформы. Мы использовали Prompt Lab, доступную на платформе IBM watsonx.ai, поскольку она нам хорошо знакома, но ничего в этом разделе не связано с конкретной платформой, поэтому используйте ту, которая удобна вам.

Для нашей задачи мы будем использовать модель falcon-40b-8lang-instruct (https://huggingface.co/tiiuae/falcon-40b) с жадным декодированием. Жадное декодирование означает, что модель на каждом шаге генерирует наиболее вероятное следующее слово и выдает одинаковый результат при каждом запуске.

Несмотря на то что нам нужно получить целостные фразы, лучшие результаты достигаются при разбиении задачи на части. Пользовательские высказывания обычно представляют собой предложения или их части, состоящие преимущественно из именных и глагольных словосочетаний. Начнем с довольно

простого промпта — попросим сгенерировать синонимы для имени существительного.

Листинг 7.1. Генерация синонимов для существительного без контекста

```
Сгенерируй список существительных.
Существительные должны быть синонимами слова "пароль".
```

Простая инструкция

```
Список из 5 существительных
учетные данные, секретное слово, ключ, код, код доступа
```

Сигнал (cue) для LLM

Вывод LLM

В листинге выше LLM стала нашим словарем. И хотя полученные существительные действительно являются синонимами слова «пароль», они все же редко употребляются в контексте входа в систему. Чтобы получить более релевантные результаты, необходимо предоставить LLM больше контекста.

Для этого укажем, зачем нам нужны синонимы и какие именно типы синонимов требуются. Следующий листинг показывает улучшенный промпт и его результаты.

Листинг 7.2. Генерация синонимов для существительного с контекстом

```
Ты — потребитель, обращающийся в чат-сервис
интернет-магазина.
Сгенерируй нумерованный список существительных,
которые потребитель мог бы использовать для описания
проблемы со входом на сайт во фразе "Я забыл свой пароль".
Существительные должны быть синонимами слова "пароль".
```

Фоновая информация для LLM

Подробные инструкции, предоставляющие контекст — забытый пароль

```
Список из 5 существительных
1. "данные учетной записи" 2. "учетные данные" 3. "данные для входа"
4. "аутентификация" 5. "идентификатор"
```

Сигнал для LLM

Вывод LLM

Так звучит гораздо естественнее. Теперь сгенерируем синонимы для глагола.

Листинг 7.3. Генерация синонимов для глагола с контекстом

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список глаголов, которые
потребитель мог бы использовать, для описания проблемы
со входом на сайт во фразе "Я забыл свой пароль".
Глаголы должны быть синонимами слова "забыл".
```

Обновленная инструкция для глаголов вместо существительных

Обновленный сигнал (глаголы вместо существительных)

```
Список из 5 глаголов
1. Забыл 2. Потерял 3. Записал не туда
4. Куда-то положил 5. Ошибся
```

Вывод LLM

Неплохо, хотя предпоследний глагол выглядит немного странно в данном контексте. Еще одно ограничение состоит в том, что все сгенерированные глаголы находятся в форме прошедшего времени. Причина понятна, ведь пример тоже

был в прошедшем времени, но нам нужно, чтобы LLM предоставила больше разнообразия. Попробуем перейти от *глаголов* к *фразам с глаголами*.

Листинг 7.4. Генерация синонимичных фраз с глаголами и контекстом

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список фраз с глаголами,
которые потребитель мог бы использовать для описания
проблемы со входом на сайт во фразе "Я забыл свой пароль".
Глаголы должны быть синонимами слова "забыл".

Список из 5 фраз с глаголами
1. Я забыл свой пароль.
2. Я потерял свой пароль.
3. Я не помню свой пароль.
4. Я куда-то положил свой пароль.
5. Я не могу вспомнить свой пароль.
```

Обновленные инструкция и сигнал для «фраз с глаголами»

Вывод LLM

LLM выдала полные предложения (отлично!), и теперь среди них есть также высказывание в настоящем времени («не могу вспомнить»). Это уже прогресс! Всего за несколько минут работы с промптами мы добились того, что LLM предложила множество вариантов, над которыми можно подумать. Раньше мы считали, что наш чат-бот должен просто подбирать синонимы к словам «забыл» и «пароль». Теперь же у нас есть десяток других полезных слов, которые также стоит учитывать при тестировании бота.

Давайте немного поэкспериментируем. На этот раз повысим креативность модели, перейдя на *декодирование с семплированием* (sampling decoding) и повысив температуру. Мы также изменим промпт, попросив «10 синонимов» вместо «5 существительных».

Листинг 7.5 показывает полученные существительные, а листинг 7.6 — фразы с глаголами.

> **ПРИМЕЧАНИЕ** При жадном декодировании LLM каждый раз генерирует одинаковые результаты. При декодировании с семплированием создается недетерминированный вывод. С такими промптами вы, скорее всего, получите другие результаты. Все в порядке! Мы используем LLM лишь для того, чтобы почерпнуть идеи.

Листинг 7.5. Генерация синонимов для существительного с повышенной креативностью

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список синонимов, которые
потребитель может использовать вместо слова "пароль",
описывая проблему вроде "Я забыл свой пароль".

Список из 10 синонимов слова "пароль":
```

Обновленные промпт и сигнал

```
1. "Секретное слово"
2. "Персональный идентификационный номер"
3. "PIN-код"
4. "Код доступа"
5. "Информация для входа"
6. "Данные для входа"
7. "Информация учетной записи"
8. "Данные учетной записи"
9. "Идентификационный номер"
10. "Код безопасности"
```

Вывод LLM

Отлично! Неплохой список существительных. Не обязательно использовать их все, но это уже довольно широкий набор для тестирования.

Теперь сделаем то же самое с глаголами.

Листинг 7.6. Генерация синонимов фраз с глаголами с повышенной креативностью

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список синонимов, которые
потребитель может использовать вместо слова "забыл",
описывая проблему вроде "Я забыл свой пароль".

Список из 10 синонимов слова "забыл":
1. Забыл
2. Не вспомнил
3. Не знал
4. Неизвестно
5. Не подходит
6. Не могу получить доступ
7. Не смог вспомнить
8. Не запомнил
9. О боже, не смог вспомнить
10. Не могу войти
```

Обновленные инструкция и сигнал

Вывод LLM

Куда более креативный список синонимов. Конечно, не без странностей («неизвестно», «не подходит»), но есть и интересные варианты:

- *Не вполне подходящий глагол.* «Не вспомнил» звучит немного странно, но наводит на «не могу вспомнить».
- *Неправильное время.* «Не знал» наводит на «не знаю».
- *Эмоциональная окраска.* «О боже» напоминает о том, что пользовательские высказывания могут содержать эмоциональные элементы.

Всего за несколько минут мы сгенерировали синонимы, релевантные нашей предметной области, но пока что у нас есть только отдельные слова. Мы начали с высказывания «Я забыл свой пароль» и можем подставлять в него новые существительные и глаголы, но все еще ограничены простой структурой «подлежащее — сказуемое — дополнение». Пользователи наверняка будут использовать

более разнообразные грамматические конструкции, и мы не хотим, чтобы наш чат-бот зависел только от одной из них. Он должен понимать разные формы высказываний. Давайте применим LLM для генерации новых вариантов.

7.2.2. Генерация новых грамматических конструкций

Вместо того чтобы просить LLM сгенерировать слова или словосочетания, попробуем получить от нее полные высказывания. Для этого нам нужно разработать промпт, который запрашивает лексическое разнообразие, но при этом не смещает модель исключительно в сторону «Я забыл свой пароль». Давайте используем схожий промпт, который также задает контекст, но вместо того чтобы включать в него фразу «Я забыл свой пароль», опишем проблему пользователя (он не может войти в аккаунт).

Первый результат показан в листинге ниже.

Листинг 7.7. Генерация полных высказываний

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список примерных высказываний,
с которых потребитель мог бы начать диалог, если он
не может войти в систему. Используй разнообразные
существительные и глаголы, связанные со входом,
паролями и аккаунтами.

Список из 10 высказываний:
1. Я не могу войти в систему
2. Мой пароль не подходит
3. Я не знаю свой пароль
4. Я забыл свое имя пользователя
5. Я забыл свой пароль
6. Мой аккаунт заблокирован
7. Я не получил письмо для сброса пароля #
8. Мне нужна помощь с аккаунтом
9. Можете помочь мне войти в аккаунт?
10. Я пытался сбросить пароль, но не вышло
```

Обновленные инструкция и сигнал

Вывод LLM

Мы получили больше разнообразия. Теперь список высказываний содержит несколько новых формулировок (например, «письмо для сброса пароля»). Кроме того, теперь предложения имеют разную структуру. В выводе модели встречаются следующие грамматические конструкции:

- *Утверждение с прямым порядком слов (действительный залог)*: «Я забыл свой пароль».
- *Страдательный залог*: «Мой аккаунт заблокирован».
- *Конструкции с предлогом*: «Мне нужна помощь с аккаунтом».
- *Вопрос*: «Можете помочь мне войти в аккаунт?»
- *Уточнение*: «Я пытался сбросить пароль, но не вышло».

Разнообразие примеров растет. Сгенерированные высказывания можно использовать для обучающего или тестового набора, но все еще остаются некоторые пробелы. Например, все высказывания — полные предложения. А что насчет пользователей, которые настолько заняты (или раздражены), что пишут лишь пару слов? Может ли LLM сгенерировать подобные фрагменты предложений? Ответ — в листинге ниже.

Листинг 7.8. Генерация фрагментов высказываний

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список примерных фраз, с которых
потребитель мог бы начать диалог, если он не может войти
в систему. Используй разнообразные существительные
и глаголы, связанные со входом, паролями и аккаунтами.
Используй только неполные предложения.

Список из 10 неполных предложений:
1. Забыл пароль
2. Не могу войти
3. Аккаунт заблокирован
4. Нужна помощь со сбросом пароля
5. Не помню адрес электронной почты
6. Почта не работает
7. Не могу сменить пароль
8. Не могу получить доступ к аккаунту
9. Не могу создать новый аккаунт
10. Не могу удалить старый аккаунт
```

Обновленная инструкция для генерации неполных предложений. Сигнал не изменился

One-shot-пример сразу после сигнала

Вывод LLM

Во всех предыдущих промптах нам не удавалось сгенерировать неполные предложения. На этот раз мы дали LLM дополнительную инструкцию. Помимо обычных изменений (замены слова «высказывания» на «неполные предложения»), мы добавили еще одну подсказку — первый пример «Забыл пароль». Это называется one-shot-обучением, потому что мы дали LLM один пример того, что нам нужно, и это помогло модели понять, как обрабатывать наш запрос.

> **Zero-shot? One-shot? Few-shot?**
>
> Термины «zero-shot», «one-shot» и «few-shot» относятся к количеству примеров (shot), указанных в промпте. Zero-shot-промпт не содержит примеров. One-shot-промпт включает один пример. Few-shot-промпт содержит несколько примеров.

Для генерации обучающих данных обучение — это отличный способ получить именно тот тип вывода, который вам нужен. Каждый раз, когда у вас возникают трудности с тем, чтобы заставить LLM следовать вашим инструкциям, попробуйте привести хороший пример, а не просто корректировать формулировку. Работая над этой главой, мы использовали больше промптов, чем приведено в книге, и ни один из них не давал нам неполные предложения до тех пор, пока мы не применили one-shot-обучение.

Кроме того, one-shot-обучение можно использовать для переноса наработанных решений при создании одного намерения на другое. В следующем листинге мы генерируем примеры для намерения «сброс пароля», используя примеры для намерения «местоположение магазина».

Листинг 7.9. Применение one-shot-обучения для множества грамматических конструкций

```
Ты — потребитель, обращающийся в чат-сервис
интернет-магазина.

Сгенерируй фразы, которые потребитель может использовать,
чтобы узнать, где расположены магазины. Создай фразы
для каждой из следующих грамматических конструкций.
Прямой вопрос: Где вы находитесь?
Косвенный вопрос: Можете сказать, как найти ваши магазины?
Неполное предложение: Местоположение магазина
Команда: Дайте маршрут проезда

Сгенерируй фразы, которые потребитель может
использовать, когда ему нужно сбросить пароль.
Создай фразы для каждой из следующих грамматических
конструкций.
Прямой вопрос: Я забыл свой пароль
Косвенный вопрос: Как я могу сбросить свой пароль?
Неполное предложение: Сброс пароля
Команда: Отправь мне ссылку для сброса пароля
```

Стандартный контекст для LLM остается без изменений

One-shot-пример включает инструкцию и желаемый результат

Инструкция для LLM дополняется подсказкой «Прямой вопрос:»

Вывод LLM (начинается после подсказки «Прямой вопрос:»)

С помощью одного такого промпта мы смогли получить примеры для каждой нужной нам грамматической конструкции (LLM допустила ошибку в пункте «Прямой вопрос», но результат все равно полезный).

Есть еще один прием для генерации высказываний. Вместо того чтобы использовать подробные инструкции, предоставьте несколько примеров и попросите LLM продолжить список. Для этого мы воспользуемся другим форматом запроса и другой моделью — granite-13b-instruct-v2 (https://mng.bz/DMlR), а также применим декодирование с семплированием чтобы повысить креативность и получить недетерминированные результаты. В листинге ниже приведены промпт и первый вывод.

Листинг 7.10. Использование креативного промпта для генерации примеров с моделью Granite

```
<|инструкция|>
Вот реальные высказывания, отправленные пользователями
в автоматизированную службу поддержки. Твоя задача —
сгенерировать новые примеры от людей, у которых возникли
проблемы с паролем и входом в систему.

<|пример|>
Я не могу войти в аккаунт
```

Маркер, обозначающий нашу инструкцию для модели

Текст инструкции

Начало примеров

```
<|пример|>
Мои данные для входа не подходят

<|пример|>
Забыл пароль

<|пример|>
Помогите мне войти в мой аккаунт

<|пример|> ◄——— Сигнал для вывода
Здравствуйте, похоже, я не могу войти в свой аккаунт ◄— Вывод LLM
```

Поскольку мы используем недетерминированные параметры, результат работы модели каждый раз будет отличаться. Вот примеры выводов, полученных при пяти последовательных запусках одного и того же промпта:

- «Можете помочь мне восстановить пароль?»
- «Мой аккаунт заблокирован».
- «Не могу вспомнить имя пользователя или пароль».
- «Надеюсь, вы можете мне помочь, Я только что сбросил пароль, но ничего не вышло».
- «Я ввел неверный пароль 5 раз подряд».

Мы не уточняли, какая именно вариативность нам нужна, но все же получили несколько интересных формулировок. Появились новые глаголы («восстановить») и новые концепции («5 раз подряд»). Вот почему так важны эксперименты с разными LLM, промптами и параметрами. Генерация обучающих данных требует доли креативности. Не полагайтесь на один-два эксперимента — используйте в своей работе совместные усилия как человека, так и генеративного ИИ.

7.2.3. Создание намерений на основе вывода LLM

Подведем итог нашим экспериментам. Мы сгенерировали ряд контекстуальных (а не просто общих) синонимов для существительных и глаголов, создали целые высказывания с похожей структурой, а затем использовали LLM для генерации предложений с разными грамматическими структурами. В своей работе мы применяли несколько моделей, разные промпты и настройки параметров. Генеративный ИИ оказался отличным помощником!

Теперь у нас есть множество вариантов для формирования обучающего набора. Выберем 10 высказываний, охватывающих все созданные нами ранее вариации. Для некоторых высказываний мы возьмем точный вывод LLM, в другие же внесем корректировки. Например, сгенерированные высказывания часто содержали слово «пароль» — мы можем заменить его на «данные для входа» или «данные аккаунта». Также во многих фразах встречался глагол «забыл», поэтому мы заменим его на «не помню». В листинге ниже приведен возможный набор высказываний.

Листинг 7.11. Десять отобранных нами высказываний на основе предложений LLM

```
1. Я не могу войти в аккаунт
2. Мои данные для входа не подходят
3. Забыл пароль
4. Аккаунт заблокировали
5. Не могу вспомнить данные моего аккаунта
6. Мой аккаунт заблокирован
7. Не пришло письмо для сброса пароля
8. Нужна помощь со сбросом пароля
9. Можете помочь мне войти в аккаунт?
10. Я пытался сбросить пароль, но ничего не вышло
```

Мы прошли большой путь с момента изначального предположения, что «большинство запросов содержат слова «забыл» и «пароль». Если использовать данные высказывания в обучающем наборе, наш чат-бот будет понимать пользователей гораздо лучше, чем при обучении только на основе ключевых слов.

В успешной генерации высказываний нам помогло целое множество промптов. Отсюда возникает вопрос: можем ли мы объединить их в один?

Нам понадобятся все приемы, изученные ранее: контекст для LLM, четкие инструкции и one-shot-примеры. Попробуем обучить намерение «поиск магазина», используя только лучшие примеры из намерения «забыл пароль». В листинге ниже продемонстрирован данный процесс с использованием модели falcon-40b-8lang-instruct и жадного декодирования.

Листинг 7.12. Применение one-shot-обучения для переноса знаний с одного намерения на другое

```
Ты — потребитель, обращающийся в чат-сервис
интернет-магазина.
```

Стандартный контекст для LLM

```
Сгенерируй нумерованный список примеров высказываний,
с которых потребитель может начать диалог, если он
не может войти в систему. Используй разнообразные
существительные и глаголы, связанные со входом, паролями
и аккаунтами. Используй различные грамматические конструкции.
```

Инструкция и сигнал для one-shot-примера (сброс пароля)

```
Список из 10 высказываний
1. Я не могу войти в аккаунт
2. Мои данные для входа не подходят
3. Забыл пароль
4. Аккаунт заблокировали
5. Не могу вспомнить данные моего аккаунта
6. Мой аккаунт заблокирован
7. Не пришло письмо для сброса пароля
8. Нужна помощь со сбросом пароля
9. Можете помочь мне войти в аккаунт?
10. Я пытался сбросить пароль, но ничего не вышло
```

One-shot-пример

```
Сгенерируй нумерованный список примеров высказываний,
с которых клиент может начать диалог, если ему нужно
найти определенный магазин. Используй разнообразные
существительные и глаголы, относящиеся к поиску,
местоположению и маршруту. Используй различные
грамматические конструкции.

Список из 10 высказываний
1. Где находится ваш магазин?
2. Как найти ваш магазин?
3. Как отсюда добраться до вашего магазина?
4. Какой поворот мне нужен, чтобы доехать до вашего магазина?
5. Ваш магазин находится к северу или к югу отсюда?
6. На какой улице находится ваш магазин?
7. Где именно расположен ваш магазин?
8. Как далеко отсюда ваш магазин?
9. Сколько времени нужно, чтобы добраться до вашего магазина?
10. В каком городе находится ваш магазин?
```

Инструкция и сигнал для целевого намерения (поиск магазина)

Вывод LLM

Неплохо! Отличное начало для нового намерения. Результат хорош по нескольким причинам:

- *Разнообразие глаголов* — ни один глагол не повторяется, за исключением «находится».
- *Разнообразие понятий* — примеры охватывают как абсолютные, так и относительные понятия посредством использования слов «местоположение» и «маршрут». Они также включают временны́е и пространственные аспекты («сколько времени», «как далеко»).
- *Разнообразие по степени детализации* — высказывания варьируются от «в каком городе» до «на какой улице», а также «отсюда».

Однако у данных высказываний есть и определенные ограничения:

- *Грамматическая структура* — все высказывания являются вопросами. Нет ни команд, ни неполных предложений.
- *Разнообразие существительных* — во всех примерах используется слово «магазин».
- *Очевидные пропуски* — например, «не работает навигатор, я заблудился». Удивительно, что в высказываниях отсутствуют такие варианты, как «какой у вас адрес» или «маршрут проезда».

Задача оказалась слишком сложной для выполнения одним промптом. Мы попросили LLM выдать все, что нам нужно, и даже предоставили примеры. Модель смогла выполнить многие из наших запросов, но также проигнорировала или не справилась с некоторыми из них. Все не так просто, как кажется: невозможно идеально настроить одно намерение и попросить LLM воспроизвести этот результат для всех остальных. В нашей задаче слишком много инструкций

и переменных, чтобы текущие LLM могли все выполнить с первого раза. Возможно, в будущем это изменится.

Вот почему мы рекомендуем использовать LLM как напарника, а не пытаться все сделать самостоятельно или, наоборот, полностью положиться на модель. Вы не можете переложить мыслительный процесс на LLM, но можете поручить ей провести эксперименты за вас, и она сделает это очень быстро. Генерация синонимов и разнообразных грамматических структур звучит просто, но вряд ли вы сможете сделать это так же быстро и полно, как LLM. Позвольте модели сгенерировать идеи, а затем выберите лучшие из них.

> **ЗАПОМНИТЕ** LLM не может думать за вас, но она сможет составить неплохой «черновик».

В табл. 7.3 приведены рекомендации о том, что стоит и чего не стоит делать при использовании LLM для генерации обучающих и тестовых данных.

Таблица 7.3. Наши «да» и «нет» в использовании LLM для генерации обучающих и тестовых данных

Да	Нет
• Использовать LLM как партнера или креативного помощника. Управление процессом остается за вами. • Предоставлять контекст и четкие инструкции. • Направлять LLM с помощью примеров и one-shot-обучения. • Экспериментировать с промптами. • Использовать вывод LLM для дополнения данных, собранных от пользователей	• Использовать ответы LLM без проверки и доработки. • Ожидать, что LLM сама поймет, чего вы от нее хотите. • Пытаться выполнить слишком много задач в одном промпте. • Передавать конфиденциальные данные коммерческой платформе LLM, которая сохраняет их «для будущего обучения». • Полагать, что вывод LLM способен заменить реальные пользовательские данные

LLM отлично справляются с генерацией обучающих данных даже в тех случаях, когда у вас есть четко сформулированная задача, но нет репрезентативных пользовательских высказываний. И хотя всегда лучше использовать реальные пользовательские данные, полученные из продакшена, такая возможность есть не всегда. Данные, сгенерированные LLM, помогают восполнить такие пробелы. Датасеты, на которых обучаются такие модели, обширны и наверняка включают и нужную вам предметную область (например, обслуживание клиентов), хоть и не охватывают все ваши потребности. Они, скорее всего, уже встречали множество высказываний вроде «сброс пароля», но ни одного, содержащего название именно вашего приложения. Если перед вами стоит выбор между отсутствием обучающих данных, данными, созданными экспертами, и данными, сгенерированными LLM, предпочтение лучше отдать последнему варианту.

7.2.4. Получение новых примеров с помощью шаблонов

В предыдущем разделе мы собрали разнообразный список высказываний, объединив несколько отдельных выводов LLM и использовав их для генерации новых высказываний. На рис. 7.2 показан пример изменения полных высказываний (из листинга 7.7) с использованием синонимов, приведенных в листингах 7.1–7.6.

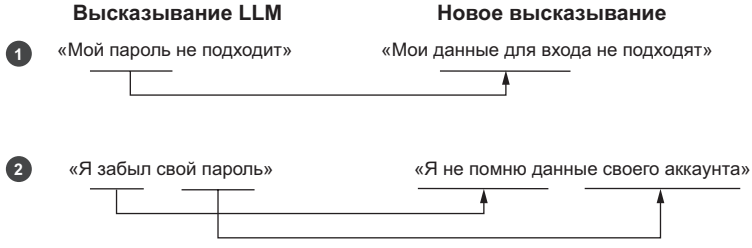


Рис. 7.2. Генерация дополнительных примеров на основе исходного вывода LLM. Модель сгенерировала «Мой пароль не подходит», но теперь мы знаем, что смежное с ним высказывание — «Мои данные для входа не подходят»

> **СОВЕТ** Создание примеров на основе шаблонов — это программная задача, а не задача генеративного ИИ. Сочетание в своей работе разных подходов позволяет получить лучшие результаты.

Использование шаблонов особенно полезно, когда некоторые выводы LLM содержат только один глагол или одно существительное. И хотя нам удалось разнообразить набор высказываний вручную, мы можем развить эту идею и рассматривать вывод LLM как шаблон. Начав с базового высказывания «Я забыл свой пароль», мы затем подобрали контекстные синонимы для слов «забыл» и «пароль». На рис. 7.3 это высказывание преобразовано в шаблон «Я <глагольная группа> свой <именная группа>», позволяющий сгенерировать больше высказываний.

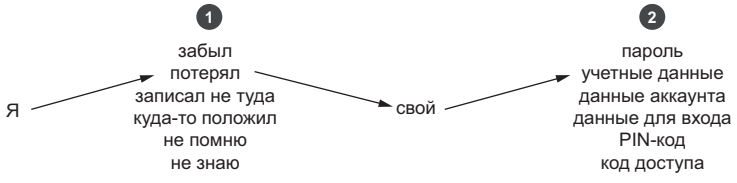


Рис. 7.3. Преобразование фразы «Я забыл свой пароль» в шаблон, позволяющий заменять глаголы и существительные в контексте. Пример высказывания, созданного на основе данного шаблона, — «Я потерял свои учетные данные»

Такой шаблон позволяет сгенерировать в общей сложности 36 высказываний, так как включает шесть вариантов глаголов и шесть вариантов существительных ($6 \times 6 = 36$). Это большой объем данных, но он несбалансирован — все высказывания используют одну и ту же грамматическую структуру. Еще хуже то, что некоторые из них могут вообще никогда не встретиться в реальных пользовательских диалогах. Такой подход непригоден для генерации обучающих данных, поскольку он чрезмерно смещает модель в сторону одной языковой конструкции. Шаблонные высказывания будут перевешивать влияние других, более нестандартных примеров, таких как, например, «Можете помочь мне войти в аккаунт?», «Я пытался сбросить пароль, но ничего не вышло» и «Аккаунт заблокировали».

Тем не менее шаблонные высказывания могут быть использованы для тестового набора, но только если вы осознаете их несбалансированность. Нет ничего плохого в том, чтобы проверить ваш разговорный ИИ на всех 36 высказываниях в качестве санити-теста («проверки на вменяемость») — просто не ограничивайтесь только одним шаблоном, посчитав, что намерение и так уже хорошо обучено.

Шаблоны полезны для создания тестовых данных, так как они позволяют убедиться, что чат-бот способен отличить одно намерение от другого даже при наличии лишней информации. Предположим, что помимо шаблона «забыл пароль», у нас есть еще шаблон «местоположение магазина», в котором используются глаголы «нужен», «забыл» и «хочу» и существительные «адрес», «местоположение» и «маршрут». Шаблон для определения местоположения магазина аналогичен рис. 7.3, но использует структуру «Я <глагольная группа> ваш <именная группа>». Также предположим, что некоторые пользователи будут сначала приветствовать бота («Привет», «здравствуйте», «добрый день») или формулировать просьбу о помощи («можете мне помочь?», «пожалуйста, помогите»). Такие дополнения никак не отражаются на общем смысле пользовательского высказывания. Влияют ли они на работу чат-бота? На рис. 7.4 показано, как мы можем организовать такой тест.



1. Шаблон приветствия.
2. Шаблон «забыл пароль».
3. Шаблон «местоположение магазина».
4. Шаблон заключительной фразы.

Рис. 7.4. Использование шаблонов для проверки того, влияют ли приветствия и заключительные фразы на понимание чат-ботом. Пример одного из высказываний: «Здравствуйте, я потерял свои учетные данные, пожалуйста, помогите»

В каждом намерении есть три варианта глаголов и три варианта существительных, что в итоге дает девять (3 × 3 = 9) комбинаций для каждого намерения. Без учета приветствий можно было бы провести 18 тестов (по 9 на намерение). В нашем тесте мы добавили три варианта приветствий и два варианта заключительных фраз, что позволило увеличить размер тестового набора в шесть раз. 108 (18 × 6 = 108) высказываний будут включать, например: «Привет! Я забыл свой пароль. Можешь мне помочь?» и «Добрый день. Мне нужен ваш адрес. Помогите, пожалуйста» — и еще 106 других вариантов. Все они могут быть включены в тестовый набор.

> В теории нет разницы между теорией и практикой — но на практике есть.
>
> — *Йоги Берра*

Мы не ожидаем, что добавление приветствий и заключительных фраз как-то повлияет на классификацию, но все равно можем это проверить. Если ваши обучающие данные сильно несбалансированны, чат-бот может реагировать и на дополнительные слова. Поэтому проведение подобных тестов может стать еще одной проверкой на вменяемость (санити-тестом) наряду с методами, представленными в главе 5.

Задания

1. С помощью генеративного ИИ составьте примеры для намерения «местоположение магазина». Сколько существительных, глаголов и разных грамматических конструкций вы сможете сгенерировать? Засеките время, необходимое на выполнение данного задания:
   - Используйте промпт, содержащий только инструкции, — zero-shot-промпт.
   - Используйте промпт, включающий примеры, — one-shot- или few-shot-промпт.
   - Удалось ли модели сгенерировать более разнообразные высказывания за меньшее время, чем при их создании вручную?
2. Повторите данное задание для намерения, которое плохо распознается вашим чат-ботом (или чат-ботом, которым вы пользуетесь). Если возможно, дополните обучающий или тестовый набор чат-бота некоторыми из этих новых высказываний и измерьте изменение точности.

7.3. Повышение креативности

В главе 5 мы рассмотрели применение принципов data science для оценки и улучшения ваших обучающих и тестовых данных. Они позволяют измерить уровень понимания чат-ботом и количественно оценить эффект от произведенных улучшений. Однако устойчивые методы требуют больше времени для реализации. Прежде чем углубиться в статистические подходы, рассмотрим

несколько нестандартных способов использования LLM. Методы на основе LLM, приведенные в этом разделе, не заменяют статистические, но помогают быстрее получить общее представление о системе.

7.3.1. Генерация дополнительных намерений

LLM может помочь вам придумать новые намерения, которые системе, возможно, придется обрабатывать. И хотя желательно работать с реальными данными, например с логами обращений в службу поддержки, пара новых идей не повредит. Если вы только начинаете процесс поддержки, у вас может еще не быть необходимых исходных данных, и LLM поможет вам сделать первый шаг. В листинге ниже продемонстрирован процесс генерации идей для новых намерений.

Листинг 7.13. Генерация новых намерений

```
Ты — потребитель, обращающийся в чат-сервис интернет-магазина.

Сгенерируй нумерованный список проблем, из-за которых ты мог бы обратиться к чат-боту магазина.

Список из 5 проблем:
1. Я хочу купить товар, но не могу найти его на сайте
2. Цена изменилась
3. Мой заказ не пришел
4. Мне нужна помощь с аккаунтом
5. Я хочу вернуть товар
```

Стандартный контекст для LLM — без изменений

Инструкция и сигнал

Вывод LLM

Отличное начало! Мы получили проблемы, связанные с поиском товаров, изменением цены, отслеживанием заказа, входом в аккаунт и возвратами. Все эти категории можно развить в отдельные намерения и диалоговые сценарии в нашем разговорном ИИ.

7.3.2. Проверка на наличие неоднозначностей

Мы также можем проверить, согласна ли LLM с намерениями, которые мы собрали. Для этого необходимо взять несколько обучающих высказываний и попросить LLM распределить их по намерениям. Посмотрим, что произойдет, если не сообщать модели название нашего намерения — «забыл пароль».

Листинг 7.14. Определит ли LLM то же намерение («забыл пароль»), что и мы?

```
На основе списка высказываний определи, к какому намерению они относятся.
```

Инструкция для LLM

```
Список высказываний:
1. Я не могу войти в аккаунт
2. Мои данные для входа не подходят
3. Забыл пароль
4. Аккаунт заблокировали
5. Не могу вспомнить данные моего аккаунта
6. Мой аккаунт заблокирован
7. Не пришло письмо для сброса пароля
8. Нужна помощь со сбросом пароля
9. Можете помочь мне войти в аккаунт?
10. Я пытался сбросить пароль, но ничего не вышло
```

Входные данные, упомянутые в инструкции

```
Классификация намерений:  <-- Сигнал
- "Я не могу войти в аккаунт" → "Вход"
- "Мои данные для входа не подходят" → "Вход"
- "Забыл пароль" → "Сброс пароля"
- "Аккаунт заблокировали" → "Вход"
- "Не могу вспомнить данные моего аккаунта" → "Вход"
- "Мой аккаунт заблокирован" → "Вход"
- "Не пришло письмо для сброса пароля" → "Сброс пароля"
- "Нужна помощь со сбросом пароля" → "Сброс пароля"
- "Можете помочь мне войти в аккаунт?" → "Вход"
- "Я пытался сбросить пароль, но ничего не вышло" → "Сброс пароля"
```

Вывод LLM

LLM предложила два намерения, хотя мы использовали только одно. Намерение «вход» и «сброс пароля» вполне соответствуют нашему исходному «забыл пароль». Однако предложенные LLM намерения кажутся слишком узкими, особенно учитывая, что для обоих случаев, скорее всего, потребуется одно и то же решение.

> **Формат вывода LLM не всегда последователен**
>
> В листинге 7.14 LLM «изобрела» собственную схему вывода — пункты списка маркированы через тире, а высказывания и намерения соединены стрелками. Так как мы просто визуально просматриваем результаты, такой формат допустим, в противном случае можно предоставить LLM дополнительные инструкции по выводу (например, «ответь маркированным списком»). Также можно привести пример желаемого формата в one-shot-промпте.

Данный тест не такой надежный, как методы, представленные в главе 5, но его можно использовать для быстрой проверки корректности обучающих данных. Если LLM не видит в них какой-либо внутренней согласованности, возможно, стоит задуматься.

LLM и разработчики способны хорошо дополнять друг друга. На рис. 7.5 показаны различные методы, с помощью которых LLM может улучшить способность вашего разговорного ИИ понимать пользователей.

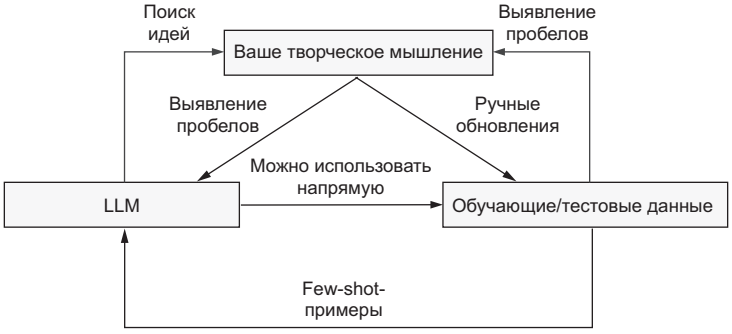


Рис. 7.5. LLM полезна разработчику во множестве задач

Задания

1. Опишите назначение чат-бота, которого вы на данный момент создаете или используете. С помощью данного описания и промпта к LLM сгенерируйте примеры проблем, которые ваш бот должен уметь решать. Примените декодирование с семплированием и отправьте промпт несколько раз, чтобы получить разные варианты. Соотносятся ли полученные примеры с намерениями или сценариями, которые обрабатывает бот?
2. Из тестовых данных вашего бота извлеките несколько высказываний. Попросите LLM определить, к какому намерению или сценарию они относятся. Соответствует ли ответ LLM реализации бота?

Итоги

- LLM — отличные «напарники», расширяющие возможности разработчика. Людям и LLM куда лучше работать вместе.
- Экспериментируйте с разными моделями, промптами и параметрами, чтобы добиться наилучшего результата от LLM. Продолжайте итерации! Не рассчитывайте, что первый результат сразу будет идеальным.
- Не ограничивайтесь только лишь инструкциями для LLM. Приводите примеры через one-shot- или few-shot-промпты.
- Если вы обнаружили пробел в данных, то можете попросить LLM заполнить его.
- Результаты вывода LLM можно сразу использовать в обучающих данных или же предварительно их отредактировать.
- Жадное декодирование позволит получать одинаковый вывод при каждом запросе. Для более разнообразных и креативных ответов применяйте декодирование с семплированием. Используйте один и тот же промпт несколько раз, чтобы получить множество разных выводов и выбрать наиболее подходящий.

Часть 3

Паттерн: ИИ слишком сложный

Сложные чат-боты одинаково неудобны как для разработчиков, так и для пользователей. Первым трудно сопровождать запутанные рабочие процессы, а вторым — общаться с ботом. Избыточная сложность снижает вероятность успешного завершения сценария, тем самым сокращая ценность технологии ИИ.

Никто не стремится создать громоздкое решение. Изначально все хотят сделать что-то простое. Но по мере добавления новых функций и учета ряда дополнительных факторов сложность может внезапно повыситься. В этой части книги мы расскажем, как это контролировать.

Глава 8 содержит несколько примеров диалоговых сценариев и показывает, как можно понизить их сложность, делая процессы более удобными для пользователя. Глава 9 учит контролировать сложность системы с помощью доступного контекста, персонализируя и адаптируя диалог под канал взаимодействия. Глава 10 рассматривает использование LLM для снижения сложности чат-ботов как на этапе их разработки, так и в продакшене.

8 Оптимизация сложных сценариев

В этой главе

- ✓ Влияние сложности системы на пользователя
- ✓ Влияние сложности системы на бизнес и команду поддержки
- ✓ Как найти баланс между естественностью диалога и сложностью реализации
- ✓ Как упростить путь пользователя

Избыточная сложность вызывает трудности как у пользователей чат-ботов, так и у их разработчиков, часто приводя к неблагоприятным бизнес-результатам или задержкам при внедрении. Создание диалога, который будет одновременно простым и естественным, требует внимательного планирования и понимания конечного пользователя. Как дизайнеры и разработчики таких решений мы стремимся создать опыт, позволяющий пользователю достичь своей цели с минимальными усилиями. Почему это так важно? Потому что потребители должны принять и начать использовать систему, чтобы оправдать затраты разработчика на ее сопровождение.

Потребители ассоциируют «естественный» диалог с «простотой» и «удобством». Интерфейс, которым легко пользоваться, как правило, дает наилучшие результаты. И напротив, взаимодействие, при проектировании которого не были учтены потребности пользователя, часто получается запутанным, неестественным

и чрезмерно сложным. Это может привести к тому, что клиенты будут чаще обращаться к операторам, покидать чат или вообще не смогут достичь нужного им результата.

В этой главе мы рассмотрим проблему сложности системы с точки зрения как пользователя, так и бизнеса. Иногда ради снижения сложности для пользователя приходится идти на некоторые «уступки». Мы обсудим возможные компромиссы, необходимые для устранения проблем пользователя, а также то, как определить приоритеты и выбрать оптимальные решения.

8.1. Бремя сложности

Сложность — палка о двух концах: она может добавить проблемные моменты или точки отказа в диалог, ориентированный на выполнение конкретной задачи, но без нее зачастую невозможно реализовать более полезные сценарии взаимодействия. Простые FAQ-боты редко бывают сложными, но при этом значительно ограничены в функциональности. Пользователям, которым нужно выполнить какое-то действие, нужен бот, способный не только объяснить, *как* что-то сделать, но и сделать это за них (или от их имени). Рассмотрим пример: пользователь хочет узнать остаток средств на своем счете. FAQ-бот сможет лишь объяснить, как проверить баланс, но цель пользователя при этом не будет достигнута. Бот самообслуживания, напротив, сразу предоставит фактический баланс счета, тем самым удовлетворив запрос.

Разница в сложности между двумя решениями очевидна. FAQ-боту достаточно определить цель пользователя и выдать соответствующий ответ. А процессно-ориентированный бот обычно требует интеграций с внешними бэкенд-системами. Ему может понадобиться аутентифицировать пользователя и получить доступ к конфиденциальной информации из одного или нескольких источников.

Таким образом, существует неизбежный уровень сложности, присущий решениям, способным выполнять более полезные задачи, свойственные электронным помощникам. Виртуальный агент должен обслуживать две стороны: конечного пользователя и бизнес или организацию, которая его разработала и сопровождает. Решение, кто возьмет на себя все бремя или последствия сложности, требует поиска тонкого баланса.

8.1.1. Влияние сложности на конечного пользователя

Сложные транзакционные диалоги могут показаться пользователю запутанными или даже пугающими. Плохо спроектированные сценарии увеличивают проблемы при взаимодействии. Главный недостаток избыточной сложности заключается в том, что она создает дополнительные точки отказа. Из-за ошибок в ходе диалога пользователи могут застрять на одном месте и несколько

раз повторять попытки добиться от бота желаемого. В результате они просят соединить их с оператором или просто покидают чат.

Мы сами стали свидетелями эффекта избыточной сложности, когда работали с одной страховой компанией. У нее был свой голосовой помощник для поставщиков медицинских услуг, которые звонили, чтобы получить информацию о статусе и оплате поданных заявок, но процесс поиска заявки имел лишь 40 % успешности. Для начала поиска от пользователя требовалось получить четыре вида данных. На рис. 8.1 показаны этапы поиска заявки.



Рис. 8.1. Сценарий предоставления статуса заявки включает семь этапов. Пользователь должен предоставить четыре вида данных, прежде чем система выполнит поиск и (в случае успеха) выдаст статус заявки

Процесс поиска был довольно утомительным для пользователя, поскольку требовал ввода нескольких длинных числовых последовательностей. Листинг ниже демонстрирует данный сценарий с позиции пользователя.

Листинг 8.1. Сценарий поиска заявки, в котором пользователь должен предоставить четыре вида данных

```
ПОЛЬЗОВАТЕЛЬ: Я хочу узнать статус заявки.
ГОЛОСОВОЙ ПОМОЩНИК: Я помогу вам узнать статус вашей заявки. Сначала,
пожалуйста, назовите ID поставщика.
ПОЛЬЗОВАТЕЛЬ: <называет номер поставщика>
ГОЛОСОВОЙ ПОМОЩНИК: Спасибо. Теперь, пожалуйста, укажите ID клиента.
ПОЛЬЗОВАТЕЛЬ: <называет номер клиента>
ГОЛОСОВОЙ ПОМОЩНИК: Я вас понял. Назовите дату подачи заявки.
ПОЛЬЗОВАТЕЛЬ: <называет дату заявки>
ГОЛОСОВОЙ ПОМОЩНИК: Теперь, пожалуйста, назовите номер заявки.
ПОЛЬЗОВАТЕЛЬ: <называет номер заявки>
ГОЛОСОВОЙ ПОМОЩНИК: Пожалуйста, подождите, выполняется поиск заявки.
ГОЛОСОВОЙ ПОМОЩНИК: Я нашел вашу заявку.
```

Мы пытались найти способы сократить сложность сценария. Исследование показало, что большинство пользователей имели только одну заявку для конкретной комбинации ID клиента и даты. Другими словами, поиск по этой информации часто давал единственный результат. Это означало, что запрос номера заявки был излишним для большинства пользователей.

На рис. 8.2 показан обновленный сценарий поиска заявки, требующий меньшего количества данных. Если найдена только одна заявка, ее статус предоставляется сразу. Если же система обнаруживает несколько заявок, бот запрашивает у пользователя номер для уточнения.

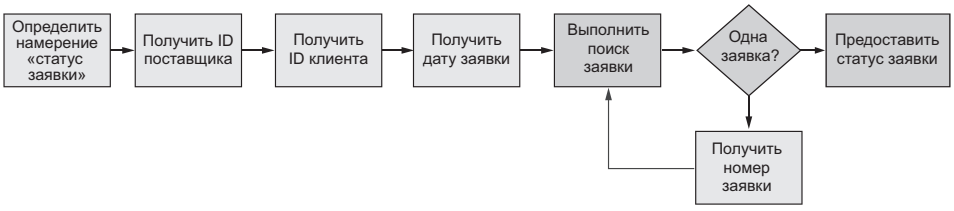


Рис. 8.2. Обновленный сценарий предоставления статуса заявки требует только три вида данных для большинства пользователей. Четвертый вид запрашивается лишь при необходимости

Исключив этап запроса номера заявки для учетных записей, содержащих только одну заявку, мы упростили сценарий для пользователя. Это, в свою очередь, повысило показатель завершения задач и сократило количество ошибок при поиске номера заявки вдвое. В листинге ниже показан усовершенствованный вариант диалога.

Листинг 8.2. Большинству пользователей теперь требуется предоставить только четыре вида данных

```
ПОЛЬЗОВАТЕЛЬ: Я хочу узнать статус заявки.
ГОЛОСОВОЙ ПОМОЩНИК: Я помогу вам узнать статус вашей заявки. Сначала,
пожалуйста, назовите ID поставщика.
ПОЛЬЗОВАТЕЛЬ: <называет номер поставщика>
ГОЛОСОВОЙ ПОМОЩНИК: Спасибо. Теперь, пожалуйста, укажите ID клиента.
ПОЛЬЗОВАТЕЛЬ: <называет номер клиента>
ГОЛОСОВОЙ ПОМОЩНИК: Я вас понял. Назовите дату подачи заявки.
ПОЛЬЗОВАТЕЛЬ: <называет дату заявки>
ГОЛОСОВОЙ ПОМОЩНИК: Я нашел вашу заявку.
```

8.1.2. Влияние сложности на бизнес-метрики

Чересчур сложный пользовательский сценарий может негативно сказаться на бизнес-метриках по нескольким направлениям: может вырасти доля пользователей, покидающих сценарий (коэффициент отказов), увеличиться количество эскалаций к сотрудникам поддержки, снизиться доля успешно выполненных задач в автономном режиме, а также ухудшиться NPS или результаты пользовательских опросов.

Примером подобного влияния на бизнес, в контексте уже упомянутой ранее страховой компании, стало неожиданно высокое количество отказов (запросов соединить с оператором), возникавших сразу после успешного поиска заявки. После того как пользователь проходил процесс поиска (включавший ответы на три или четыре вопроса), ему задавался финальный вопрос: хочет ли он, чтобы информация о заявке была зачитана ему или же отправлена по факсу[1]. На рис. 8.3 показано распределение ответов пользователей на данный вопрос.

[1] В США зарегулированные отрасли до сих пор используют факс вместо электронной почты в целях безопасности. — *Примеч. ред.*



Рис. 8.3. Почти две трети пользователей просили соединить их с оператором. Наиболее логичный вариант — зачитать информацию пользователю (то есть передать ее через тот же канал, в котором он уже взаимодействует) — оказался лишь вторым по популярности. Всего 1 % пользователей выбрали получить информации по факсу

На данном этапе система уже успешно идентифицировала пользователя и нашла нужную ему информацию. Но, к сожалению, пользователи прерывали взаимодействие, не осознавая, что были *в шаге* от достижения цели. Это стало значительной проблемой для бизнеса — потеря удержания означала, что задачи, которые могли быть (и практически были) успешно завершены виртуальным помощником, приходилось выполнять людям. Следующий листинг демонстрирует исходный пользовательский сценарий до внесения изменений.

Листинг 8.3. Пользователь запрашивает перевод на оператора, даже несмотря на успешный поиск заявки

```
ГОЛОСОВОЙ ПОМОЩНИК: Я нашел вашу заявку. Зачитать ее вам или отправить по факсу?
ПОЛЬЗОВАТЕЛЬ: Соедините с оператором.
```

В теории первоначальное решение предложить два варианта предоставления информации, по всей видимости, выглядело довольно логично (эксперты в предметной области утверждали: «Пользователи часто просят отправить им информацию по факсу»), однако фактические данные показали, что на данном этапе диалога терпение пользователей иссякало. Вероятная причина могла заключаться в том, что сам вопрос уже создавал ложное ощущение сложности: иначе почему бы боту просто не зачитать информацию? Также возможно, что пользователю надоело отвечать на множество вопросов, не получая при этом ощутимого результата.

В данном примере бизнес пострадал по нескольким направлениям: выросло число отказов и эскалаций, а доля задач, решенных ботом самостоятельно, соответственно, снизилась. Логичным решением стало исключение данного вопроса и автоматическое зачитывание информации о заявке. Возможность получения факса была перенесена на более поздний этап — после предоставления всей информации о заявке. На рис. 8.4 показан обновленный сценарий диалога и его возможные результаты.



Рис. 8.4. В обновленном сценарии информация о заявке зачитывается сразу, а затем предлагается опция получения факса для небольшого процента пользователей, которым это требуется

Исключение одного шага диалога гарантировало, что все пользователи, предоставившие необходимые данные для поиска заявки, успешно завершали сценарий, как показано в листинге 8.4. После предоставления информации пользователь мог просто завершить вызов, если получил все необходимое, что приводило к полному удержанию звонка системой. Пользователь также мог выбрать получение факса — в таком случае запускался отдельный сценарий. Некоторые пользователи все же могли запросить оператора, но причины обращения на данном этапе, скорее всего, отличались от тех, что были *ранее*. (Возможно, у них возникли какие-то *другие* вопросы.)

Листинг 8.4. Детали заявки предоставляются сразу, если поиск прошел успешно

```
ГОЛОСОВОЙ ПОМОЩНИК: Я нашел вашу заявку, номер 10012345. Заявка была одобрена,
и 3 октября была произведена выплата в размере $100. Хотите также получить факс
с этой информацией?
ПОЛЬЗОВАТЕЛЬ: (может ответить "да", "нет", запросить оператора или завершить
звонок)
```

Разрабатывая своего бота — или улучшая уже существующего, — помните, что понятные взаимодействия (с точки зрения пользователя) упрощают достижение успешного результата. Определение «успешности» варьируется в зависимости от конкретного сценария и бизнес-целей. При правильном планировании бизнес-метрики системы можно напрямую связать со стратегиями, использованными при ее построении или улучшении. Подходящие метрики помогут понять уровень успешности сценария, его сильные и слабые стороны.

8.1.3. Дополнительные издержки и выгоды от снижения сложности для пользователя

Снижение сложности системы для пользователя может иногда потребовать более сложно спроектированного диалога. Транзакционные диалоги, то есть те, в которых пользователь взаимодействует с системой на протяжении нескольких

шагов для достижения своей цели, имеют множество потенциальных точек сбоя, и каждая новая функция или возможность добавляет дополнительную сложность. Чтобы минимизировать влияние этой сложности на пользователя, диалоги должны быть максимально естественными и продуманными. Иными словами, чем сложнее ваш процесс, тем больше усилий следует приложить к тому, чтобы снизить сложность взаимодействия для пользователя, тем самым повысив вероятность успешного результата.

Некоторые стратегии упрощения системы для пользователя просты и недороги в реализации, но дают значительный эффект. Другие же, особенно требующие интеграции с бэкенд-системами, могут быть технически сложными и/или затратными. Каждая компания и каждый сценарий должны оценивать выгоду, получаемую от таких улучшений, по сравнению со стоимостью их внедрения и другими компромиссами при добавлении в диалог возможностей вроде понимания естественного языка, персонализации и автоматизации.

Разумеется, правильно подобранные метрики — тема, повторяющаяся на протяжении всей книги, — являются ключом к грамотному планированию и расстановке приоритетов. Стоит ли проектировать сценарии под все возможные варианты взаимодействия, с которыми может столкнуться бот? Определенно нет. Но существуют подходы, позволяющие оптимизировать опыт для большинства пользователей и ситуаций. Правило 80/20 помогает принять решение о том, куда лучше направить усилия по улучшению или расширению возможностей бота. Если 80 % пользователей получат от этого выгоду, улучшение стоит реализовать.

Сложность взаимодействия для пользователя может воспрепятствовать достижению его цели. Проектирование эффективного и в то же время простого диалога требует глубокого понимания пользователя: кто он, зачем пришел к вашему виртуальному помощнику и чего ожидает от взаимодействия с ним. Анализ конечного пользователя должен быть основой вашего дизайна. Возможно, нецелесообразно или экономически невыгодно сразу реализовывать все возможные улучшения, однако все же стоит стремиться к тому, чтобы бот быстро и точно понимал потребность пользователя и устранял препятствия, мешающие ему достичь цели: везде, где это только возможно.

Задания

Изучите сценарий бота, который вы построили или с которым взаимодействовали:

1. Перечислите все шаги.
2. Определите шаги, выполнение которых наиболее затруднительно для пользователя.
3. Определите, есть ли возможность изменить порядок шагов или убрать некоторые из них, чтобы упростить процесс для пользователя.

8.2. Упрощение и оптимизация пути пользователя

Существует множество стратегий и техник, которые можно использовать для проектирования естественного и простого диалогового сценария. В этом разделе мы рассмотрим возможные способы оптимизации пути пользователя.

8.2.1. Определение сложных диалоговых сценариев

В сложных сценариях каждое действие или ответ пользователя могут направить его по определенному пути. В идеале этот путь должен быть наиболее эффективным маршрутом к достижению конечной цели. Если путь пользователя слишком сложен или неэффективен, необходимо его улучшить.

Как понять, что ваше решение слишком сложное или неэффективное? Мы наблюдали несколько антипаттернов диалогового дизайна, которые излишне повышали сложность взаимодействия:

- Запрос у пользователя информации, которой у него, скорее всего, нет под рукой или на получение которой требуется время.
- Жесткие, негибкие требования к формату пользовательского ввода.
- Неоднозначные вопросы, вызывающие у пользователя сомнение относительно того, как дать «правильный» ответ.
- Одинаковое обращение со всеми пользователями и сценариями, особенно с ненужными вопросами в некоторых случаях.
- Избыточное количество вариантов ответа или варианты, не соответствующие ожиданиям пользователя относительно того, как достичь их цели.
- Запрос информации в нелогичном или разрозненном порядке.
- Запрос информации, не соответствующей интерфейсу или каналу взаимодействия (например, запрос адреса электронной почты по голосовому каналу).
- Передача информации, не соответствующей интерфейсу или каналу взаимодействия (например, озвучивание длинного или сложного URL по голосовому каналу).

Выявление сложных элементов в своих диалоговых сценариях — первый шаг на пути к их упрощению. Метрики эффективности также могут указывать на существующие проблемные зоны. Тщательный анализ решения с точки зрения пользователя поможет выявить сложные взаимодействия.

8.2.2. Использование уже известной информации о клиенте

Оптимальный пользовательский опыт подразумевает эффективное использование той информации, которая уже известна системе о клиенте в начале разговора, а также той, что собирается в процессе самого взаимодействия. Такая информация, как правило, хранится в контексте диалога. Она может поступать из

бэкенд-системы или непосредственно от самого пользователя в ходе взаимодействия. Используя известные данные о пользователе, можно персонализировать диалог или динамически направлять клиента по наиболее эффективному пути к завершению задачи.

Одна из компаний, с которой мы работали, предлагала пользователю три варианта для поиска его учетной записи: по (13-значному) номеру учетной записи, по номеру социального страхования или по номеру телефона. После выбора *способа* поиска пользователь должен был ввести соответствующий номер. Каждый из этих шагов не только требовал от него дополнительных усилий, но и создавал потенциальные точки отказа. На рис. 8.5 представлены шаги данного сценария.

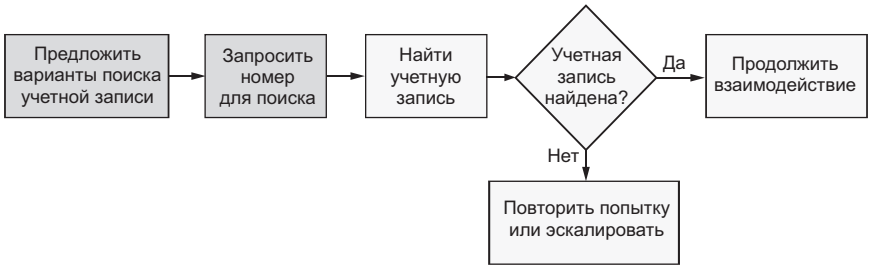


Рис. 8.5. Пользователь должен предоставить два вида информации, чтобы продолжить выполнение сценария

Листинг ниже содержит пример диалога с позиции пользователя, говорящего с голосовым помощником по телефону.

Листинг 8.5. Для поиска учетной записи у пользователя запрашиваются два вида информации

```
ГОЛОСОВОЙ ПОМОЩНИК: Что вы хотите использовать для поиска вашей учетной записи?
Номер телефона, номер социального страхования или номер учетной записи?
ПОЛЬЗОВАТЕЛЬ: Номер телефона.
ГОЛОСОВОЙ ПОМОЩНИК: Какой у вас номер телефона?
ПОЛЬЗОВАТЕЛЬ: <говорит номер телефона>
ГОЛОСОВОЙ ПОМОЩНИК: Одну минуту, ищу вашу учетную запись.
ГОЛОСОВОЙ ПОМОЩНИК: Я нашел вашу учетную запись. Пожалуйста, введите код
доступа.
```

Поскольку решение реализовано в формате телефонного звонка, мы обычно можем определить номер, с которого звонит клиент. Эта информация позволит упростить путь для большинства пользователей. Вместо того чтобы задавать два вопроса перед выполнением поиска, мы осуществляем фоновый поиск по номеру телефона. Если учетная запись найдена, мы предлагаем использовать для поиска номер телефона. Пользователю остается лишь подтвердить свое согласие. На рис. 8.6 представлена обновленная схема поиска.

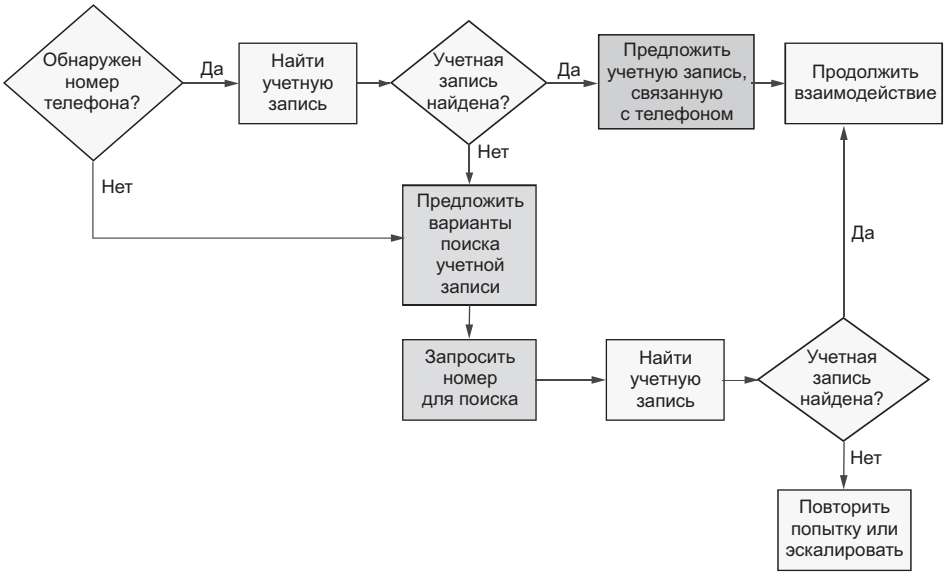


Рис. 8.6. Используя уже известные сведения о пользователе (его номер телефона), мы можем избежать некоторых проблемных точек и устранить потенциальные ошибки, возникающие при сборе данных от пользователя

В следующем листинге приведен пример обновленного диалога с позиции пользователя.

Листинг 8.6. Пользователи, чей номер телефона был определен, минуют ненужные шаги

```
ГОЛОСОВОЙ ПОМОЩНИК: Ваша учетная запись привязана к номеру телефона, с которого
вы звоните?
ПОЛЬЗОВАТЕЛЬ: Да.
ГОЛОСОВОЙ ПОМОЩНИК: Я нашел вашу учетную запись. Пожалуйста, введите код
доступа.
```

8.2.3. Соответствие ожиданиям пользователя

Ход сценария должен соответствовать ожиданиям пользователя от диалога. Для этого необходимо предоставлять или запрашивать информацию в том порядке, в каком пользователь ожидает ее получить или передать. Другими словами, вы должны согласовать сценарий взаимодействия с его внутренней моделью хранения и извлечения информации.

Это касается и выбора формулировок. Когда ваш чат-бот предлагает варианты ответа, убедитесь, что терминология соответствует тому, как пользователь понимает предлагаемые опции. Достаточно ли у него информации, чтобы выбрать наиболее подходящий для своей ситуации вариант?

Еще один способ соответствовать ожиданиями пользователя — позволить ему предоставлять несколько видов информации за один шаг. Такой прием называется *«заполнение слотов»* (slot filling), и он позволяет пользователю выразить свою потребность собственными словами, включая важные для него детали или уточнения. Если система будет способна распознавать и сохранять ключевую информацию, это позволит клиенту продвинуться дальше по сценарию, минуя повторяющиеся или ненужные этапы.

Так, например, при бронировании столика в ресторане система может запросить дату, время, количество человек, имя для брони и контактный номер. Но в диалоговом решении пользователь может сказать что-то вроде: «Я бы хотел забронировать столик на двоих на эту субботу в восемь вечера». Гибкая система распознает его намерение (забронировать столик) и определит три из пяти необходимых деталей брони (так называемые *сущности*). Так система сможет пропустить этапы, на которых она должна была спросить дату, время и количество человек, и сразу перейти к имени и номеру телефона. На рис. 8.7 показано, как разные реплики заполняют слоты в сценарии бронирования.

Высказывание	Дата	Время	Количество гостей
Забронировать столик.			
Забронировать столик на эту субботу.	X		
Забронировать столик на эту субботу в восемь вечера.	X	X	
Забронировать столик на двоих на эту субботу в восемь вечера.	X	X	X

Рис. 8.7. По мере того как пользователь предоставляет больше информации, каждое упомянутое слово (подчеркнутое) заполняет соответствующий слот, устраняя необходимость запрашивать эти сведения на последующих этапах

8.2.4. Гибкость в ожидаемых ответах пользователя

Процессно-ориентированные боты должны получать от пользователя определенные данные, чтобы продвигаться по сценарию. Иногда пользователь не совсем понимает, что именно от него требуется, и если он не сумеет предоставить необходимую информацию или указать ее в требуемом формате, это может привести к повторным запросам или сбоям. Гибкая система позволяет предусмотреть несколько «правильных» ответов, что дает возможность упростить диалог для одних пользователей и терпеливо направлять других, нуждающихся в дополнительной помощи.

Непонимание вопроса — частая причина повторных попыток или сбоев при сборе информации. Это чаще всего происходит, когда вопрос с несколькими вариантами ответа интерпретируется пользователем как вопрос типа «да/нет»

и наоборот. Подход к решению этой проблемы зачастую определяется каналом взаимодействия. На рис. 8.8 показаны способы избежать путаницы в подобных вопросах с использованием возможностей разных каналов.

Канал	Исходный вывод	Обновленный вывод
Веб	Вы хотите получить напоминание в сообщении или по телефону?	Вы хотите получить напоминание в сообщении или по телефону? Варианты кнопок: (Сообщение) (Звонок)
Телефон	Вы хотите получить напоминание в сообщении или по телефону?	Я могу отправить напоминание в сообщении или позвонить вам по телефону. Какой вариант предпочтительнее? (Повтор) Если вы хотите получить напоминание в сообщении, скажите «сообщение» или нажмите 1. Если вы хотите, чтобы я вам позвонил, скажите «звонок» или нажмите 2.

Рис. 8.8. Ответы модели можно адаптировать под конкретный канал, чтобы обеспечить лучшие условия для успешного взаимодействия

Если устранить неоднозначность невозможно (особенно в голосовом канале, где нет кнопок и раскрывающихся списков), убедитесь, что ваш бот способен корректно обработать даже попытку пользователя дать ответ, который, по его мнению, запрашивает система. Вместо того чтобы повторно задавать вопрос (что воспринимается как ошибка и может привести к эскалации), бот должен уточнить смысл ответа пользователя.

В листинге ниже приведен пример переспрашивания, который указывает на то, что пользователь не понял вопрос с первого раза.

Листинг 8.7. Повтор шага, указывающий на проблему с ответом пользователя

```
ГОЛОСОВОЙ ПОМОЩНИК: Вы хотите получить напоминание в сообщении или по телефону?
ПОЛЬЗОВАТЕЛЬ: Да.
ГОЛОСОВОЙ ПОМОЩНИК: Извините, я вас не понял. Пожалуйста, выберите: текстовое
сообщение или телефонный звонок.
```

С позиции пользователя разница заключается в том, что повтор реплики указывает на то, что он «застрял» или что-то пошло не так, тогда как уточняющий вопрос, показанный в следующем листинге, создает впечатление, что пользователь продолжает продвигаться по сценарию.

Листинг 8.8. Уточнение, подтверждающее, что пользователь продвигается по сценарию

```
ГОЛОСОВОЙ ПОМОЩНИК: Вы хотите получить напоминание в сообщении или по телефону?
ПОЛЬЗОВАТЕЛЬ: Да.
ГОЛОСОВОЙ ПОМОЩНИК: Отлично! Что предпочтительнее — сообщение или телефонный
звонок?
```

Другой способ оптимизации диалога — обработка исправленной информации, если она предоставляется в ответ на вопрос подтверждения (как правило, вопрос, на который ожидается ответ «да» или «нет»). Например, мы спрашиваем звонящего боту пользователя, хочет ли он получать текстовые уведомления на номер телефона, с которого звонит (определитель номера). Наиболее вероятные ответы, которые точно необходимо предусмотреть, — это утвердительный («Да, пришлите уведомление на этот номер») или отрицательный («Нет, не надо слать уведомление на этот номер»).

В процессно-ориентированном сценарии утвердительный ответ приведет к переходу на следующий шаг диалога. Отрицательный ответ может вызвать процесс получения альтернативного номера для отправки сообщений. Любой другой ответ приведет к отклонению от сценария и/или повтору вопроса.

Рассмотрим третий тип информации, которую наш бот может получить на данном этапе: пользователи иногда сразу предоставляют обновленные или исправленные данные вместо ответа «да» или «нет» (например: «Используйте другой номер: 555-867-5309»). В листинге ниже неожиданный ответ вызывает ошибку, что приводит к повтору запроса, а затем к дополнительному шагу по сбору данных.

Листинг 8.9. Жесткие рамки допустимого ответа при шаге подтверждения приводят к лишним действиям

```
ГОЛОСОВОЙ ПОМОЩНИК: Хотите ли вы использовать номер, с которого вы звоните, для
получения текстовых уведомлений?
ПОЛЬЗОВАТЕЛЬ: Отправляйте их на 555-867-5309.
ГОЛОСОВОЙ ПОМОЩНИК: Извините, я вас не понял. Хотите ли вы использовать номер,
с которого вы звоните, для получения текстовых уведомлений?
ПОЛЬЗОВАТЕЛЬ: (вздыхает) Нет.
ГОЛОСОВОЙ ПОМОЩНИК: Хорошо, на какой номер вы хотите получать уведомления?
ПОЛЬЗОВАТЕЛЬ: (вздыхает) 555-867-5309.
```

Если бот ожидает только ответы «да» или «нет», его логика не сработает и произойдет сбой, вызывающий повтор запроса. Кроме того, пользователю придется повторять информацию, которую он уже предоставил в своем первоначальном ответе. Эти два лишних шага (повтор и сбор данных) неудобны для пользователя, ценящего эффективность.

Следующий листинг содержит пример более гибкой системы, когда она воспринимает обнаруженный телефонный номер как эквивалент ответа «нет» и одновременно сохраняет его, что позволяет пользователю миновать последующий этап сбора данных.

Листинг 8.10. Гибкая система запоминает необходимую информацию, что оптимизирует сценарий

```
ГОЛОСОВОЙ ПОМОЩНИК: Хотите ли вы использовать номер, с которого вы звоните, для
получения текстовых уведомлений?
ПОЛЬЗОВАТЕЛЬ: Отправляйте их на 555-867-5309.
ГОЛОСОВОЙ ПОМОЩНИК: Хорошо. Уведомления будут отправляться на номер 555-867-5309.
```

8.2.5. Поддержка сценариев самообслуживания с помощью API и бэкенд-процессов

Проектирование интерфейса, который будет более понятен конечному пользователю, может потребовать интеграций, персонализирующих или ускоряющих процесс самообслуживания. А можно ли использовать API-подключения к клиентской базе данных, чтобы извлекать необходимую информацию и тем самым сокращать процесс и повышать вероятность успешного выполнения задачи?

На уже знакомом нам примере со страховой компанией рассмотрим, как бэкенд-процессы могут упростить путь пользователя. В данной системе ID клиентов могут состоять из девяти или одиннадцати цифр. Кроме того, девятизначный ID начинается с буквы. Поскольку взаимодействие проходит по телефонному каналу, возникает несколько уровней сложности: получение буквенно-цифровой информации по голосовому каналу, ее распознавание и последующий поиск в базе данных.

Получение буквенно-цифровой информации по голосовому каналу означает, что пользователю нужно либо произнести свой ответ, либо использовать клавиатуру телефона. Распознавание буквенно-цифровой информации по голосовому каналу подвержено ошибкам из-за сходства звучания некоторых буквенных символов («B», «C», «D», «Z» и т. д.), а также из-за схожего звучания некоторых цифр на английском и букв (например, «8» и английская буква «H»). На рис. 8.9 показан первоначальный сценарий, дающий довольно высокий уровень ошибок.

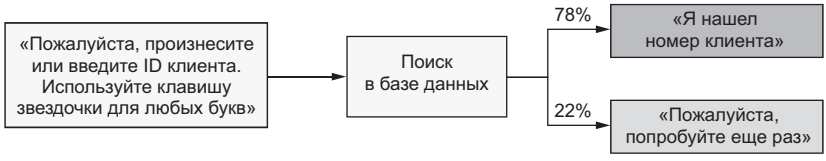


Рис. 8.9. Первоначальный сценарий содержал сложный вопрос, а ввод данных приводил к высокому уровню ошибок

Чтобы упростить сценарий для пользователя, мы добавили бэкенд-процесс, определяющий, получило ли приложение достаточное количество цифр, с ведущим буквенным символом или без него. Если обнаруживались только девять цифр, система автоматически добавляла буквенный символ за пользователя (он одинаков для всех клиентов). Более подробные инструкции предоставлялись при повторе шага. На рис. 8.10 показано, как обновленный бэкенд-процесс позволил упростить формулировку вопроса.

Реализация подобной возможности может быть довольно дорогостоящей, поэтому итоговое решение зачастую принимается на основе анализа соотношения

затрат и выгод. (Есть ли у вас необходимые для этого метрики?) Добавление подобного функционала может значительно повысить ценность решения, но также внести больше потенциальных точек отказа.

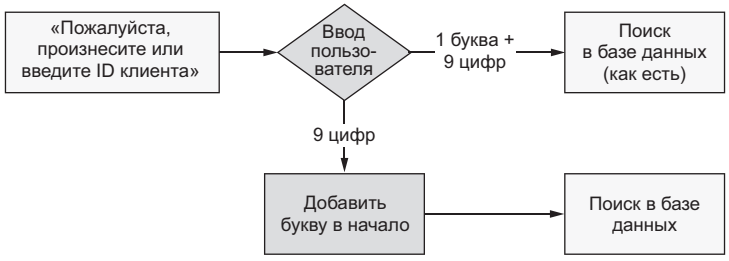


Рис. 8.10. Добавив бэкенд-шаг, выполняющий часть работы за пользователя, мы смогли упростить вопрос

Для каждой возможной точки отказа следует спроектировать диалог таким образом, чтобы он корректно обрабатывал ошибку, предоставляя пользователю наиболее подходящую альтернативу или возможность вернуться к нужному этапу. Можно, например, добавить повтор шага (повтор предыдущего вопроса), а также предоставить пользователю возможность выбрать другой путь для достижения его цели или переключиться на оператора.

Дальнейшие действия будут зависеть от типа ошибки. Например, если система ожидает ввод десяти цифр, а пользователь ввел только девять, логично повторить запрос, а если бэкенд-система недоступна (что вызывает сбои API), повтор будет бесполезен. В подобных случаях лучше всего передать обращение оператору для обработки вручную. Если поиск учетной записи по ее номеру не увенчался успехом, можно предложить пользователю воспользоваться другим способом, например поиском по номеру телефона.

Задания

Проанализируйте вашу текущую систему, чтобы выявить возможности для упрощения и оптимизации пользовательского пути:

1. Есть ли в вашем решении какие-либо антипаттерны, создающие дополнительную сложность для пользователя (см. раздел 8.2.1)?
2. Эффективно ли вы используете информацию, которая уже известна о пользователе и его ситуации?
3. Следует ли разговор логичной последовательности, учитывающей ответы пользователя?
4. Использует ли ваше решение API или бэкенд-процессы, которые могли бы облегчить взаимодействие, ускорить выполнение задачи или сократить вероятность ошибок ввода?

Итоги

- Чем менее естественным ощущается диалог, тем труднее пользователю успешно завершить сложный сценарий чат-бота.
- Снижение сложности для пользователя везде, где это только возможно, приведет к достижению наивысшего коэффициента удержания и доли завершенных задач.
- Недостаточно гибкое взаимодействие может дезориентировать пользователя и даже показаться ему грубым, особенно в сложных сценариях.
- Анализ целевой аудитории бота служит основой для его проектирования.
- Снижение сложности для пользователя может потребовать расширения функциональных возможностей системы, что, в свою очередь, увеличит общую сложность экосистемы чат-бота.

9 Влияние контекста на адаптивность виртуального помощника

В этой главе

- ✓ Использование контекста в ходе взаимодействия с виртуальными помощниками
- ✓ Адаптация разговорного ИИ к различным модальностям
- ✓ Выявление проблем, возникающих при игнорировании модальности

Эффективное использование контекста во взаимодействиях с виртуальным помощником имеет решающее значение для позитивного пользовательского опыта. Пользователи ожидают, что их запросы будут не просто поняты, но удовлетворены с учетом контекста. В этой главе мы рассмотрим три основных способа персонализации виртуальных помощников: контекст, модальность и генерация, дополненная поиском (RAG). Каждый из этих элементов усиливает способность системы понимать пользователей и успешно взаимодействовать с ними.

Контекст подразумевает зависимость взаимодействия от ситуативных и исторических данных о пользователе. Так, например, учет его текущего местоположения и времени для ответа на вопрос о погоде является примером эффективного использования контекста по сравнению с выдачей общего прогноза.

Модальность описывает способ взаимодействия пользователя с системой — голосом, текстом или через визуальный интерфейс. Каждая модальность имеет свои

преимущества и ограничения. Адаптация виртуального помощника к выбранной модальности обеспечивает более плавное взаимодействие — независимо от того, набирает ли пользователь сообщение, разговаривает с ботом или взаимодействует через графический интерфейс.

RAG объединяет традиционную генерацию ответов с возможностью получать информацию из внешних источников данных в режиме реального времени. Это позволяет виртуальным помощникам предоставлять более содержательные и подробные ответы, что значительно повышает их актуальность и точность.

Используя все три элемента, виртуальные помощники обеспечивают адаптивный и персонализированный пользовательский опыт. В этой главе мы покажем, как подобные методы персонализации способны преобразить взаимодействие системы с пользователями. Приведенные примеры взяты из нашего реального опыта реализации проектов и демонстрируют, как такие подходы эволюционировали в лучшие практики.

9.1. Роль контекста в работе виртуальных помощников

Эффективные виртуальные помощники используют контекст для предоставления осмысленных и результативных ответов. Без учета контекста пользователи могут столкнуться с различными затруднениями, недопониманием и разочарованием, даже если помощник технически весьма продвинут. Рассмотрим пример Эммы — выпускницы университета, которая управляет своими финансами с помощью чат-бота банка по имени Макс:

Эмма, совсем недавно окончившая университет, только устроилась на свою первую работу и рада возможности самостоятельно управлять своими финансами. Она открыла счет в банке, где в поддержке клиентов помогает чат-бот Макс. Эмма использует бота для разных задач — проверки баланса, настройки автоплатежей и получения информации о студенческом кредите. Однако, несмотря на все продвинутые функции Макса, она часто недовольна его работой.

Когда Эмма решает оформить кредитную карту, она начинает новый чат с ботом. Макс предоставляет список кредитных карт банка, описывая преимущества и недостатки каждой из них. Однако бот не распознал Эмму как уже существующего клиента банка, и поэтому выдает общие рекомендации, не учитывающие ее финансовое положение и имеющиеся счета.

Желая принять взвешенное решение, Эмма просит дать совет о том, какая кредитная карта лучше подойдет именно для нее. Макс, не имея доступа к ее данным и истории транзакций, продолжает предоставлять общие рекомендации, не соответствующие ее расходам и финансовым целям. Эмме приходится тратить дополнительное время, самостоятельно просматривая сайт банка

и звоня в службу поддержки, чтобы получить персонализированную помощь, а это сводит на нет удобство, которое должен был обеспечить Макс.

Как вы можете видеть, взаимодействие Эммы с Максом сопровождается рядом проблем: слишком общие рекомендации, игнорирование существующих отношений с банком, отсутствие персонализации и слабая коммуникация. Несмотря на технические возможности чат-бота, Эмма не получила от него достаточной пользы.

Использование контекста является ключом к повышению эффективности чат-ботов. Понимание контекста позволяет им предоставлять релевантные, своевременные и точные ответы — именно такого опыта пользователи и ожидают от чат-ботов. В табл. 9.1 приведены основные проблемы, с которыми столкнулась Эмма, и их возможные решения. Используя контекст, чат-боты, подобные Максу, могут превратиться из простых исполнителей задач в интеллектуальных и адаптивных помощников, повышающих эффективность работы и удовлетворенность пользователей. Устранение таких проблем с помощью понимания контекста обеспечит плавное и комфортное взаимодействие с ботом, успешно помогающим таким пользователям, как Эмма.

Таблица 9.1. Решение проблем Эммы

Проблема	Возможное решение	Почему и как
Чат-бот не имеет доступа к информации о пользователе при ответе на его вопросы	Интеграция информации о счете	Имея доступ к существующим счетам и финансовой истории Эммы, Макс может предоставлять персонализированные рекомендации. Например, он мог бы посоветовать кредитную карту, соответствующую текущим счетам и расходам Эммы
Чат-бот дает общие рекомендации	Контекстно-зависимые финансовые рекомендации	Имея доступ к истории транзакций и конкретным финансовым целям Эммы, Макс мог бы предложить более релевантные и персонализированные советы, экономя время и усилия пользователя
Повторение стандартных ответов	Адаптивные ответы	Понимая финансовые приоритеты и модель расходов Эммы, Макс может настраивать свои ответы и действия в соответствии с ними. Например, зная, что Эмма только начала работать и, возможно, еще продолжает выплачивать студенческий кредит, Макс мог бы предложить советы по управлению бюджетом и долгом
Чат-бот работает только в текстовом канале	Использование разных модальностей для разных контекстов	Крайне важно уметь определять подходящую модальность для каждой задачи. Например, предоставление персональных рекомендаций по кредитным картам в чате, отправка подробных финансовых отчетов по электронной почте или уведомления от приложения делают взаимодействие более эффективным и удобным

9.1.1. Влияние контекста на взаимодействие с пользователем

Контекст играет ключевую роль в формировании взаимодействия между пользователями и виртуальными помощниками, особенно чат-ботами. Если чат-бот понимает контекст запроса пользователя, он способен предоставить более точные, релевантные и персонализированные ответы, что значительно улучшает качество пользовательского опыта. В этом разделе мы рассмотрим, как контекст влияет на взаимодействие пользователей с системой.

Повышение релевантности и точности

Когда чат-бот учитывает историю взаимодействия с пользователем при формировании своих ответов, диалог становится более релевантным. Например, если Эмма часто интересуется своим сберегательным счетом, чат-бот может запомнить эту «предпочтительную» тему и оперативно предоставлять обновления и информацию по счету. Учет прошлых взаимодействий позволяет чат-боту давать ответы, ориентированные на конкретные интересы и потребности пользователя, вместо того чтобы просто предлагать общие сведения. Такой персонализированный подход экономит время пользователя и формирует у него ощущение, что бот его действительно знает и понимает.

> **ПРИМЕЧАНИЕ** Если вы хотите, чтобы ваш чат-бот учитывал историю взаимодействий с пользователем, потребуется разработать для этого отдельное решение, так как большинство провайдеров разговорного ИИ не добавляют в свои продукты функцию контекстной осведомленности (context-awareness).

Знание текущего «состояния» пользователя позволяет чат-боту предоставлять более точные ответы. Например, когда Эмма спрашивает о своем балансе, контекстно-осведомленный бот должен сначала уточнить, о каком именно счете идет речь, предложив список доступных вариантов. После выбора счета чат-бот предоставляет актуальный баланс, отражающий последние операции — пополнения, снятия, обрабатываемые или запланированные платежи, — чтобы у Эммы сформировалось полное представление о состоянии ее финансов.

Такой уровень осведомленности особенно важен для финансовых запросов, например, при предоставлении рекомендаций по доступным средствам или предстоящим платежам. Например, если напомнить Эмме о транзакциях, находящихся в обработке, она будет лучше понимать свою финансовую ситуацию. В зависимости от банка и типа операции, транзакции могут обрабатываться до пяти дней, что влияет на доступный баланс. И хотя большинство современных чат-ботов просто выдают цифру баланса, по-настоящему контекстно-осведомленная система предоставляет пользователю наиболее релевантную, своевременную и точную информацию.

Персонализированный опыт

Одним из наиболее значимых преимуществ контекстно-осведомленных чат-ботов является возможность персонализированного взаимодействия. Если

чат-бот распознает и анализирует финансовые цели и модели расходов пользователя, система способна предлагать индивидуальные рекомендации, соответствующие его потребностям. Например, пользователю, который часто путешествует, можно предложить кредитную карту с бонусами для путешественников. Такой уровень персонализации делает рекомендации чат-бота более полезными и улучшает общее восприятие системы, поскольку она учитывает уникальные обстоятельства клиента.

Кроме того, адаптация ответов к определенной ситуации и важным событиям в жизни клиента значительно повышает релевантность коммуникации. Например, если чат-бот знает, что пользователь недавно окончил университет и устроился на работу, он может предложить советы по управлению бюджетом, ориентированные на человека, который совсем недавно стал финансово независимым. Такое персонализированное взаимодействие вызывает у клиента ощущение поддержки и понимания, что повышает его удовлетворенность и вовлеченность при использовании сервиса.

В табл. 9.2 показано, как контекстно-осведомленный чат-бот адаптирует свои рекомендации с учетом конкретных жизненных обстоятельств пользователя, вместо того чтобы просто выдавать стандартные ответы.

Таблица 9.2. Примеры взаимодействия с неконтекстными, контекстными и персонализированными ответами

Запрос пользователя: «Что мне делать с расходами в этом месяце?»	
Неконтекстный ответ	Следите за своими расходами и придерживайтесь установленного бюджета. Также можно воспользоваться различными инструментами для контроля за финансами, чтобы отслеживать ваши траты
Контекстный ответ для путешественника	Так как вы часто путешествуете, будет выгодно оформить кредитную карту с бонусами для путешественников. Также можно выделить отдельный бюджет на каждую поездку и ознакомиться с доступными программами лояльности, чтобы оптимизировать расходы на путешествия в этом месяце
Контекстный ответ для выпускника университета, только устроившегося на новую работу	Желаем успехов в новой работе! Переходя к финансовой самостоятельности, важно составить бюджет с учетом нового дохода. Рекомендуем заранее выделить средства на предстоящие расходы, например на покупку рабочей одежды или проездной, а также начать формировать финансовую подушку безопасности

Эффективность в решении проблем

Контекстная осведомленность повышает эффективность взаимодействий чат-ботов при решении всевозможных проблем. Если чат-бот знает о последних действиях пользователя или о его текущих жизненных обстоятельствах, он будет предоставлять более точные и релевантные решения. Например, если

пользователь недавно сообщал о потере карты, чат-бот может сразу предложить помощь в отслеживании доставки новой карты. Такой подход ускоряет решение проблемы и положительно сказывается на отношении пользователя к системе, избавляя его от необходимости повторно объяснять ситуацию или получать стандартные ответы.

Понимание контекста также делает взаимодействие более плавным. Запоминая предыдущие шаги диалога, чат-бот может избегать повторяющихся вопросов, упрощая и ускоряя процесс. Эта функция особенно полезна в ситуациях, требующих быстрого реагирования. Чат-бот, эффективно выполняющий задачи пользователя, повышает его доверие к сервису и улучшает общее впечатление от взаимодействия.

Рассмотрим пример: пользователь недавно сообщал о потере кредитной карты. Обычно чат-бот предлагает стандартный набор опций, таких как «Проверить баланс счета», «Открыть новый счет» или «Оформить кредит». Однако при наличии контекстной осведомленности классификатор намерений чат-бота учитывает недавние взаимодействия с пользователем. Поэтому вместо стандартных пунктов чат-бот предлагает индивидуальные варианты: «Перевыпустить карту», «Отследить статус доставки карты» или «Проверить последние транзакции на предмет несанкционированной активности». На рис. 9.1 показаны два варианта взаимодействия с чат-ботом. Без контекстной осведомленности первый сценарий предлагает общие опции и требует повторного ввода информации, что приводит к неудовлетворенности пользователя. При наличии контекстной осведомленности второй сценарий «помнит» недавнюю проблему пользователя и предлагает релевантные варианты, что обеспечивает более быстрое и эффективное решение задачи.

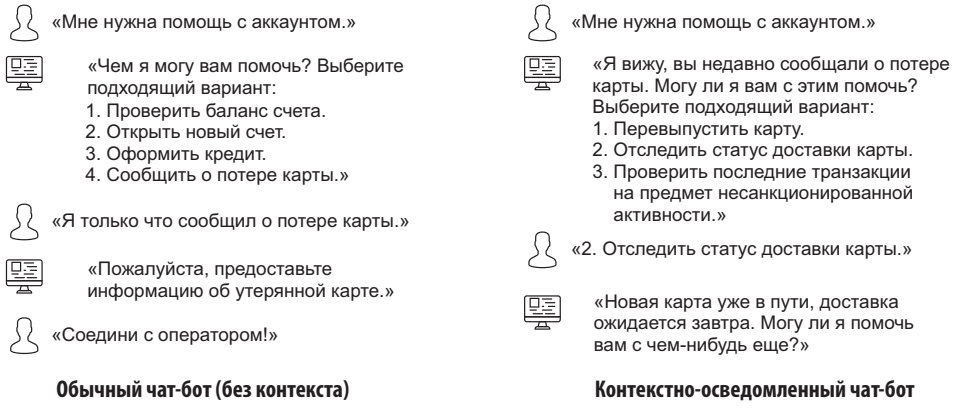


Рис. 9.1. Обычный чат-бот не учитывает контекст пользователя — он всегда предлагает одинаковые варианты ответа. Контекстная осведомленность улучшает общий пользовательский опыт

Улучшения, показанные справа на рис. 9.1, достигаются за счет отслеживания данных и использования моделей машинного обучения, которые анализируют

недавние действия пользователя и его предпочтения. Система запоминает предыдущие взаимодействия и применяет этот контекст для корректировки отображаемых опций, повышая релевантность и скорость ответов чат-бота. Такая функциональность позволяет системе обеспечивать более плавный и интуитивно понятный пользовательский опыт.

Существует два вида истории взаимодействия с чат-ботом, оба из которых важны для сохранения контекста:

- *История сеанса* — содержание текущего сеанса взаимодействия. После завершения сеанса история записывается, но затем удаляется. Эта информация помогает понимать непосредственный контекст, например уточняющие вопросы в рамках одного разговора.
- *Постоянная история пользователя* — охватывает несколько сеансов взаимодействия с одним и тем же пользователем, даже через разные каналы. Она может использоваться для определения предпочтений и особенностей поведения пользователя, однако ее интеграция в диалог требует дополнительных усилий. Использование постоянной истории улучшает пользовательский опыт при множественных взаимодействиях.

Проактивная поддержка

Проактивная поддержка является отличительной чертой хорошо спроектированного контекстно-осведомленного чат-бота. Проактивные чат-боты могут инициировать разговор с пользователем, а не просто реагировать на его запросы. Они способны выявлять потенциальные проблемы до того, как те станут критическими. Например, чат-бот банка может заранее отправлять напоминания о предстоящих платежах, низком балансе или подозрительной активности по счету. Такие уведомления помогают пользователям лучше контролировать свои финансы и избегать возможных проблем, например штрафов за просрочку платежа или перерасхода средств.

Чат-бот также может использовать в своей работе прогнозную аналитику, основанную на поведении пользователя и его финансовых паттернах. Так, например, в результате анализа тенденций и привычек пользователя система может предложить ему открыть накопительный счет, если у него регулярно наблюдается значительный ежемесячный остаток денежных средств. Такие прогнозы позволяют чат-боту выполнять роль некоего финансового советника, помогающего пользователям более эффективно достигать своих финансовых целей.

Формирование доверия и лояльности

Последовательность в ответах имеет решающее значение для формирования доверия пользователей к чат-ботам. Если бот стабильно предоставляет точные и персонализированные рекомендации, он демонстрирует свою надежность и компетентность. Пользователи чувствуют большую уверенность, полагаясь на чат-бота, когда знают, что его ответы будут адаптированы к их

ситуации. Доверие является основой для долгосрочного взаимодействия и лояльности.

Персонализированные взаимодействия и проактивная поддержка повышают удовлетворенность пользователей: когда они чувствуют, что их понимают и сочувствуют им, они чаще сообщают о положительном опыте и продолжают пользоваться сервисом. Высокий уровень удовлетворенности стимулирует повторные взаимодействия и укрепляет связь пользователя с организацией, которой принадлежит чат-бот, принося выгоду как клиентам, так и компании.

9.1.2. Что такое контекстная информация?

Контекстная информация включает в себя любые данные, которые могут персонализировать пользовательский опыт. К ним относятся местоположение пользователя, часовой пояс, тип устройства, предпочтения, модели поведения, предыдущие взаимодействия и используемые каналы. Каждый из этих факторов способен добавить индивидуальности во взаимодействие с пользователем, однако их интеграция в систему чат-бота может потребовать дополнительных усилий при разработке.

Местоположение пользователя

Местоположение пользователя указывает на его географическое положение. Оно может определяться с помощью GPS-данных или IP-адреса без необходимости вручную указывать текущее местоположение. Знание местоположения пользователя позволяет виртуальному помощнику предоставлять более релевантную и оперативную информацию, особенно в ситуациях, требующих быстрой реакции.

Рассмотрим пример. Эмма, находясь в командировке за границей, ищет ближайший банкомат.

> *Эмма отправляется в командировку за границу. Ей нужно найти ближайший банкомат, чтобы снять местную валюту, поэтому она обращается за помощью к банковскому чат-боту Максу.*

Знание местоположения пользователя имеет решающее значение для предоставления релевантной и своевременной помощи, как показано на рис. 9.2. Макс использует текущее местоположение Эммы, чтобы вывести список ближайших банкоматов. Кроме того, данные о местоположении помогают выявлять и предотвращать мошеннические операции, отмечая транзакции, совершаемые в необычных или неожиданных местах. Для получения данных о местоположении пользователя можно использовать геолокационный API и интегрировать его данные в работу чат-бота.

> **ПРИМЕЧАНИЕ** Следует безопасно хранить данные о местоположении пользователя и запрашивать согласие на их получение в соответствии с требованиями конфиденциальности. Удаляйте данные, если они больше не нужны!

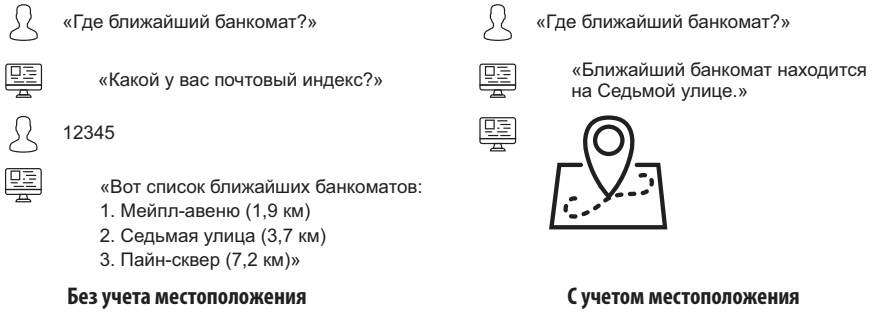


Рис. 9.2. Обычные чат-боты не учитывают местоположение пользователей, но при его учете ответы становятся более релевантными

Часовой пояс

Часовой пояс определяется местоположением пользователя. Эта информация необходима для функций, связанных с планированием и временны́ми ограничениями. Вернемся к нашему примеру.

> *Находясь в Лондоне (Великобритания), Эмма хочет запланировать телефонный разговор со службой поддержки своего банка в Нью-Йорке. Она просит Макса подобрать подходящее время.*

Знание часового пояса гарантирует, что коммуникации и напоминания будут согласованы с актуальным для пользователя временем, как показано на рис. 9.3. Так, например, Макс может предложить удобное время для звонка Эммы из Лондона, несмотря на то что команда поддержки находится в Нью-Йорке. Это предотвращает путаницу и неудобства, связанные с разницей во времени.



Рис. 9.3. Большинство чат-ботов не учитывают часовой пояс пользователя, однако с данным контекстом ответы становятся значительно полезнее

Точно конвертировать время в местный часовой пояс пользователя можно с помощью таких библиотек, как, например, pytz. Убедитесь, что ваши функции планирования учитывают часовой пояс места, где находится пользователь, при отправке уведомлений и назначении встреч, чтобы взаимодействие происходило в подходящее ему время и не вызывало недоумения.

Тип устройства

Тип устройства указывает, с какого устройства пользователь взаимодействует с чат-ботом: смартфон, планшет, компьютер или другой гаджет, например смарт-часы. Рассмотрим пример.

> *Эмма предпочитает распоряжаться своими финансами с планшета, а смартфон использует для быстрого просмотра баланса и уведомлений.*

Знание типа устройства позволит чат-боту оптимизировать взаимодействие под текущее устройство пользователя. Например, Макс может предложить упрощенный интерфейс и более лаконичные ответы для небольшого экрана смартфона, а на планшете — более подробную информацию и расширенные функции. Это обеспечивает эффективность взаимодействия вне зависимости от используемого устройства.

Определение типа устройства позволяет чат-боту адаптировать интерфейс и формат ответов для их оптимального отображения и функциональности. Тип устройства можно определить по строкам user agent или с помощью API, предоставляющих информацию об устройстве. Разработайте упрощенный интерфейс для мобильных устройств и более детализированный для компьютеров — это обеспечит плавный и персонализированный пользовательский опыт на всех платформах. При проектировании и улучшении интерфейса учитывайте, что на это потребуются дополнительные ресурсы.

Предпочтения пользователя

Предпочтения пользователя — это его индивидуальные настройки, такие как предпочтительные каналы связи, параметры уведомлений и формат представления данных. Например:

> *Эмма предпочитает получать ежемесячные финансовые сводки по электронной почте, а срочные уведомления — в виде текстовых сообщений.*

Следуя предпочтениям Эммы, Макс отправляет важную информацию наиболее удобным и ненавязчивым для нее способом. Такое уважение к пользовательским предпочтениям способствует формированию доверия и вовлеченности.

Реализация пользовательских предпочтений согласует взаимодействие с ожиданиями пользователей. Необходимо предоставить возможность выбирать предпочтительные каналы связи, параметры уведомлений и формат представления данных, а затем обеспечить безопасное хранение подобных сведений и их последовательное применение. Сохраняйте настройки для персонализации взаимодействий и уведомлений, чтобы чат-бот уважал выбор пользователя, повышая его удовлетворенность системой и формируя доверие к ней.

Модели поведения

Модели поведения отражают повторяющиеся действия и привычки пользователя, наблюдаемые в процессе его взаимодействия с чат-ботом и другими сервисами. Вернемся к примеру.

> *Эмма регулярно каждое утро проверяет баланс своего счета и оплачивает счета в первый день каждого месяца.*

Распознавание моделей поведения позволяет чат-боту предугадывать потребности пользователя и проактивно их удовлетворять. Так, например, Макс может каждое утро автоматически сообщать баланс счета или напоминать Эмме об оплате счетов по мере приближения первого числа месяца. Такая активная помощь делает взаимодействие более полезным и интуитивным, улучшая пользовательский опыт.

Анализ поведения пользователя помогает прогнозировать его потребности и заранее предоставлять поддержку. Для этого необходимо собирать данные о взаимодействиях и транзакциях пользователя, выявлять закономерности в них и применять алгоритмы машинного обучения для анализа и прогнозирования поведения. Используйте полученные данные для реализации проактивных рекомендаций и функций, которые автоматически адаптируются к распознанным моделям. Это повышает качество взаимодействия, а пользовательский опыт становится актуальным и естественным.

Предыдущие взаимодействия

Предыдущие взаимодействия — это история взаимодействий и транзакций между пользователем и чат-ботом. Она включает заданные вопросы, использованные сервисы и действия, предпринятые по результатам взаимодействий. Вернемся к нашему примеру.

> *За последние несколько месяцев Эмма неоднократно обращалась к Максу за советами по планированию бюджета и вариантами погашения кредита. Сегодня она спрашивает, как открыть накопительный счет.*

Понимание предыдущих взаимодействий позволяет чат-боту предоставлять более персонализированные и последовательные ответы. Например, зная, что Эмма ранее интересовалась финансовыми рекомендациями, Макс может предложить накопительные планы, соответствующие ее прошлым запросам и целям. Такая преемственность экономит время пользователя, избегая повторных вопросов, и формирует ощущение узнаваемости и доверия, поскольку чат-бот демонстрирует «память» и понимание потребностей.

Ведение логов предыдущих взаимодействий помогает чат-боту предоставлять персонализированные ответы. Для этого необходимо сохранять записи взаимодействий в базе данных, используя уникальные идентификаторы пользователей

для отслеживания и извлечения истории. Такая информация используется для формирования ответов на основе прошлых запросов. Благодаря алгоритмам анализа взаимодействий рекомендации бота становятся более актуальными, что обеспечивает последовательность и узнаваемость в его ответах.

ПРИМЕЧАНИЕ Логи предыдущих взаимодействий также полезены для оценки эффективности работы чат-бота. Если чат-бот дал ответ, но пользователь вернулся с тем же вопросом в течение определенного времени (например, 24 часов или недели), это может означать, что ответ оказался бесполезным. Анализ лога взаимодействий позволяет определить реальное удержание пользователя в рамках системы.

Модальность

Модальность — это способ или канал коммуникации, который пользователь предпочитает или вынужден использовать в конкретный момент. Это может быть текст, голос, электронная почта, push-уведомления или любой другой способ связи. Рассмотрим пример.

> *Эмма находится в дороге и за рулем взаимодействует с банковским чат-ботом Максом с помощью голосовых команд. Она просит Макса проверить баланс счета и последние транзакции, чтобы быть в курсе своего финансового положения, не отвлекаясь при этом на набор текста или экран телефона.*

Распознавание и адаптация под предпочтительную модальность пользователя имеют ключевое значение для эффективной коммуникации. В данном случае, учитывая текущий контекст, Макс может отправить Эмме краткое текстовое сообщение со сжатым изложением активности по счету. Разные ситуации требуют разных модальностей, и чат-бот, способный гибко переключаться между ними, обеспечивает пользователю удобное и эффективное получение информации. Такая адаптивность повышает удовлетворенность и вовлеченность пользователей, руководствуясь их индивидуальными предпочтениями и обстоятельствами.

Чат-боты могут значительно повысить контекстную осведомленность, если будут учитывать и интегрировать такие факторы, как местоположение пользователя, предыдущие взаимодействия и модальность. Это обеспечит более точные и релевантные ответы, делая взаимодействие содержательным и одновременно с этим «понимающим».

Внедрение рассмотренных выше стратегий и контекстной информации помогает чат-ботам обеспечивать высокую степень персонализации и вовлеченности. Использование клиентского контекста повышает точность и уместность ответов системы, а также укрепляет доверие и удовлетворенность пользователей, что в конечном итоге способствует их удержанию и лояльности.

Задания

1. Определите пять типов контекстной информации, которую виртуальный помощник может использовать для улучшения взаимодействия с пользователем. В списке ниже показан пример структуры анализа, описывающий, как можно собрать и использовать данные о местоположении пользователей и как они влияют на их удовлетворенность:
   - *Контекстная информация* — местоположение пользователя.
   - *Метод сбора* — GPS-данные с мобильного устройства.
   - *Использование* — предоставление сервисов, зависящих от местоположения, например местные прогнозы погоды или рекомендации ближайших ресторанов.
   - *Результат* — повышает релевантность и персонализацию, улучшая удовлетворенность пользователя.

   Проделайте то же самое для четырех других типов контекстной информации, описав способы ее сбора, влияние на взаимодействие и потенциальный эффект на вовлеченность пользователей.
2. Разработайте контекстно-зависимые ответы. Составьте три пользовательских запроса, которые виртуальный помощник мог бы получить. Для каждого укажите как минимум два типа контекстной информации, которые необходимо использовать для персонализации ответа. Напишите контекстно-зависимый ответ виртуального помощника и объясните, как контекст улучшает взаимодействие. Например:
   - Запрос пользователя: «Какая сегодня погода?»
   - Контекстная информация: местоположение пользователя, текущее время суток.
   - Контекстно-зависимый ответ: «Доброе утро! В Сан-Франциско сейчас солнечно, а к полудню ожидается повышение температуры до 24°C».

9.2. Понятие модальности

Модальность относится к различным каналам или способам, с помощью которых пользователи взаимодействуют с виртуальными помощниками. К модальностям относятся текст, голос, изображения и мультимодальные взаимодействия. Каждая из них имеет свои преимущества и ограничения, влияя на то, как пользователи общаются с системой и насколько эффективно она удовлетворяет их потребности. Использование модальности позволяет перейти от просто функционального чат-бота к интуитивно понятному и вовлекающему в диалог. Эффективность виртуального помощника напрямую зависит от выбранной модальности.

Так, например, текстовое взаимодействие может быть предпочтительным в ситуациях, где удобнее печатать, в то время как голосовые интерфейсы обеспечивают свободу рук и естественный диалог. Мультимодальные взаимодействия, объединяющие текст, голос и визуальные элементы, позволяют создавать более богатый и гибкий пользовательский опыт.

Оценка влияния различных модальностей на вовлеченность пользователей и проектирование взаимодействий имеют ключевое значение для непрерывного совершенствования. Разработчики могут применять специальные методики оценки, чтобы анализировать сценарии работы виртуального помощника в различных модальностях и выявлять области для улучшения. Благодаря этому происходит процесс постоянного развития.

9.2.1. Сравнение модальностей

Первые чат-боты представляли собой простые текстовые приложения, однако со временем они эволюционировали в мультимодальные системы коммуникации. В общем смысле *модальность* означает способ или форму, в которой происходит взаимодействие или воспринимается информация. Модальность чат-бота определяется тем, как он обрабатывает информацию и общается с пользователем:

- *Текстовая* — основная и наиболее распространенная модальность. Пользователи вводят свои вопросы или команды, а чат-бот отвечает им текстом.
- *Визуальная* — в такой модальности чат-бот взаимодействует в основном через текст, но ответы дополняются визуальными элементами, такими как кнопки, изображения, карусели, видео и другие графические интерфейсы, упрощающие взаимодействие. Добавление визуальных элементов улучшает пользовательский опыт, обеспечивая наглядные подсказки и варианты выбора.
- *Голосовая* — голосовые боты общаются с пользователями при помощи устной речи посредством распознавания ввода и генерации голосовых ответов. Как правило, диалоговый движок все равно работает с текстом — просто с использованием преобразования речи в текст и текста в речь.
- *Мультимодальная* — такие боты сочетают в себе несколько модальностей, например текст, голос, изображения и видео. Мультимодальный чат-бот может, например, понять устный запрос, отобразить соответствующие изображения и ответить одновременно голосом и текстом.

В табл. 9.3 приводится краткое сравнение модальностей, выделяются их отличительные характеристики, преимущества и недостатки, а также указывается, когда их лучше использовать и как они влияют на вовлеченность пользователя. Кроме того, в ней описаны основные способы оценки каждой модальности.

Выбор модальности напрямую влияет на то, как пользователи взаимодействуют с системой и насколько активно они вовлекаются в процесс общения с ИИ. Учитывая преимущества и недостатки текстовой, голосовой, визуальной и мультимодальной коммуникации, разработчики чат-ботов могут лучше адаптировать взаимодействие под потребности и предпочтения пользователей. Эффективность виртуального помощника определяется его способностью бесшовно интегрировать и оптимизировать различные каналы взаимодействия.

Таблица 9.3. Виды модальностей

	Текстовая модальность	Визуальная модальность	Голосовая модальность	Мультимодальные взаимодействия
Описание	Взаимодействие с помощью письменного текста	Взаимодействие с помощью текста и дополнительных визуальных элементов	Взаимодействие посредством устной речи	Сочетают текст, голос и визуальные элементы
Преимущества	Точная передача информации. На текст легко ссылаться	Повышенное понимание за счет визуальных элементов и привлекательного интерфейса	Взаимодействие без помощи рук. Естественный речевой поток	Более богатый пользовательский опыт. Универсальное взаимодействие
Недостатки	Ограничения скорости ввода текста. Отсутствуют эмоциональные оттенки	Визуальные элементы могут отвлекать пользователя или игнорироваться им. Требуется хорошо продуманный интерфейс	Требуется четкая речь. Возможны ошибки распознавания из-за особенностей произношения	Более сложная разработка и реализация
Когда использовать	В ситуациях, когда удобно вводить текст	В сценариях, требующих визуальной поддержки для лучшего понимания	В ситуациях, требующих взаимодействия без помощи рук	В сценариях, где необходимо сочетать несколько модальностей
Влияние на вовлеченность пользователя	Высокая вовлеченность в текстоцентричных контекстах	Повышенная вовлеченность благодаря визуальным подсказкам	Повышение вовлеченности за счет естественности взаимодействия	Повышенная вовлеченность благодаря разнообразию форм взаимодействия
Способы оценки	Опросы пользователей. Тестирование удобства использования	Тестирование пользовательского опыта. Анализ эффективности визуальных элементов	Тесты точности распознавания речи. Обратная связь от пользователей	A/B-тестирование. Мультимодальные оценки удобства использования

9.2.2. Влияние модальности на сценарии виртуальных помощников

При проектировании сценариев взаимодействия виртуальных помощников выбор модальности существенно влияет как на пользовательский опыт, так и на эффективность взаимодействия.

Одним из ключевых факторов является контекст, в котором используется виртуальный помощник. Например, голосовая модальность полезна, когда пользователям нужно управлять системой без помощи рук, например за рулем автомобиля или во время приготовления пищи. Такая модальность требует надежных механизмов обработки естественного языка для точной интерпретации голосовых команд и предоставления соответствующих ответов. Кроме того, разработчики должны добиваться, чтобы голосовое взаимодействие воспринималось естественно и плавно, без заметно роботизированных или повторяющихся ответов, которые могут раздражать пользователя. На работу систем распознавания речи также влияют фоновые шумы, поэтому необходимо применять технологии шумоподавления.

Для голосовых взаимодействий решающее значение имеет краткость. Так как голос — более ограниченный по объему канал, чем текст, пользователи предпочитают короткие, четкие и легко воспринимаемые ответы. Длинные и многословные ответы неудобны, поскольку пользователи не могут быстро просмотреть или пропустить их, как при чтении текста. Следовательно, виртуальный помощник должен выдавать четкую и лаконичную информацию, обеспечивая эффективное взаимодействие без помощи рук.

В напечатанных текстах ясность тоже важна, однако этот формат допускает более развернутые ответы. И хотя лаконичная информация предпочтительнее, текстовые интерфейсы дают возможность изучать детали в удобном для пользователя темпе. Текстовые виртуальные помощники должны обрабатывать короткие запросы и выдавать полезные ответы, при этом используя функции автоподсказок и предложений ввода для ускоренного взаимодействия. Пользователь может выполнять несколько задач одновременно или ожидает подробных ответов, которые можно перечитать позже, поэтому тексты и подсказки должны быть структурированы так, чтобы его не перегружать.

> **СОВЕТ** Специальные возможности, такие как, например, программы чтения с экрана (скринридеры) для пользователей с нарушением зрения, позволят сделать текстовые взаимодействия доступными и удобными для всех.

Визуальная модальность, объединяющая текст с элементами интерфейса — изображениями, кнопками, иконками, — требует иного подхода к проектированию. Визуальные элементы способны значительно повысить уровень понимания и вовлеченности, предоставляя пользователю контекстные подсказки и делая интерфейс более интерактивным. Но при этом важно соблюдать баланс между визуальной привлекательностью и функциональностью. Перегрузка интерфейса избыточными графическими элементами снизит понятность и юзабилити. Здесь важна визуальная ясность и последовательность, поэтому графика должна использоваться для дополнения текста, а не его замещения. Такой тип модальности особенно эффективен в сценариях, где визуальные средства помогают упростить сложную информацию, например в руководствах по устранению неполадок или

интерактивных инструкциях. Необходимо также обеспечивать адаптивность визуальных элементов к разным размерам экранов и разрешениям, чтобы сохранить целостность представления на всех устройствах.

9.2.3. Примеры того, как модальность влияет на пользовательский опыт

Листинги 9.1 и 9.2 демонстрируют, как чат-бот техподдержки может использовать разные модальности — текстовые инструкции и визуальные (видео) — для помощи пользователю при настройке нового принтера.

Листинг 9.1. Взаимодействие в текстовой модальности

```
Пользователь: Мне нужна помощь с настройкой нового принтера.
Чат-бот: С удовольствием помогу вам настроить принтер! Пожалуйста, выполните
следующие действия: распакуйте принтер, установите его на устойчивую поверхность
и подключите к источнику питания. Установите картриджи с чернилами и загрузите
бумагу. Скачайте и установите драйверы принтера с нашего веб-сайта для настройки
программного обеспечения, а затем подключите принтер к компьютеру через USB или
Wi-Fi.
Пользователь: Я не могу открыть крышку принтера, чтобы установить картриджи. Где
находится защелка?
Чат-бот: Защелка находится на передней панели. Пожалуйста, найдите на ней кнопки
или рычаги.
Пользователь: Я проверил, но защелки нет. Помоги мне ее найти.
Чат-бот: Приношу извинения за неудобства. Если вы не можете найти защелку,
пожалуйста, обратитесь к руководству пользователя для вашей конкретной модели
принтера, чтобы ознакомиться с подробными инструкциями.
Пользователь: У меня нет руководства. Да что же это такое!
Чат-бот: Понимаю ваше недовольство. К сожалению, я не могу предоставить более
точную информацию, не зная конкретную модель. Найдите на нашем сайте страницу
поддержки для вашей модели.
Пользователь: Все, с меня хватит!
```

Листинг 9.2. Взаимодействие в визуальной модальности

```
Пользователь: Мне нужна помощь с настройкой нового принтера.
Визуальный бот: С удовольствием помогу вам настроить принтер! Хотите посмотреть
видеоинструкцию?
Пользователь: Да, пожалуйста.
Визуальный бот: Отлично! Вот видео пошагового процесса настройки принтера.
[Видео начинает воспроизводиться] [Пользователь смотрит видео, затем начинает
настройку принтера]
Пользователь: Я дошел до момента, где нужно вставить картриджи. Можешь показать
этот момент еще раз?
Визуальный бот: Конечно! Вот фрагмент видео, где показано, как вставлять
картриджи. [Видео перематывается на нужный момент]
Пользователь: Отлично, теперь я все понял. Спасибо!
```

Текстовые инструкции, хотя и предоставляют подробное пошаговое руководство, часто вызывают недовольство у пользователей, особенно когда

они сталкиваются с какими-то проблемами, которые не были описаны. Без визуальных подсказок может быть трудно понять конкретные действия, например, как найти защелку на корпусе принтера. Это вынуждает клиентов дополнительно уточнять информацию. В листинге выше заметно растущее недовольство пользователя, когда он раз за разом не может найти защелку. Необходимость дополнительных пояснений делает взаимодействие менее эффективным и более утомительным.

В отличие от текста, визуальная модальность, особенно в форме видеоинструкций, обеспечивает более наглядный и понятный способ настройки. Видео позволяет пользователю увидеть процесс в действии, что делает каждый шаг ясным и легко воспроизводимым. Это снижает необходимость в многочисленных уточнениях и снижает вероятность ошибок или неверного толкования инструкций. Пользователь может в любой момент вернуться к нужной части видео, если ему требуется дополнительное объяснение, что делает процесс настройки более плавным и эффективным.

Рассмотрим сценарий, в котором пользователь запрашивает информацию о рейсе с помощью голосового чат-бота. Если голосовой бот спроектирован грамотно, он сможет предоставить нужную информацию или при необходимости переключиться на другую модальность. Однако плохо спроектированный бот, наоборот, вызовет еще большее недоумение и раздражение, что и показано в листингах 9.3 и 9.4.

Листинг 9.3. Взаимодействие с плохо спроектированным голосовым ботом

```
Пользователь: Хочу проверить статус моего рейса.
Голосовой бот: Пожалуйста, укажите номер рейса и дату вылета, чтобы я мог
проверить его статус.
Пользователь: Рейс UA123, 5 августа.
Голосовой бот: Подождите, ищу необходимую информацию... Ваш рейс UA123 5 августа
вылетает по расписанию. Пожалуйста, посетите наш веб-сайт, если вам нужно больше
информации или у вас есть другие вопросы.
Пользователь: Можете сказать, какой у меня номер выхода на посадку?
Голосовой бот: Пожалуйста, посетите наш веб-сайт, чтобы получить более подробную
информацию о вашем рейсе.
Пользователь: [..кладет трубку..]
```

Листинг 9.4. Взаимодействие с хорошо спроектированным голосовым ботом

```
Пользователь: Хочу проверить статус моего рейса.
Голосовой бот: С радостью вам в этом помогу. Пожалуйста, укажите номер рейса
и дату вылета.
Пользователь: Рейс UA123, 5 августа.
Голосовой бот: Одну минуту, проверяю информацию... Рейс UA123, 5 августа,
вылетает по расписанию, выход на посадку 22B. Хотите, расскажу о выдаче багажа
или как пройти к выходу?
```

```
Пользователь: Да, подскажите, как пройти.
Голосовой бот: Выход 22B находится в терминале C. После прохождения контроля
следуйте по указателям к терминалу. Он расположен рядом с фудкортом. Отправить
вам карту на телефон?
Пользователь: Да, пожалуйста.
Голосовой бот: Я отправил карту на ваш телефон. Могу ли я помочь чем-нибудь еще?
```

В плохо спроектированном взаимодействии голосовой бот не обеспечивает полноценной поддержки, вынуждая пользователя искать дополнительную информацию другими, менее удобными ему способами. И напротив, хорошо спроектированный голосовой бот предугадывает потребности пользователя, предоставляя не только запрашиваемую информацию, но и дополнительные полезные сведения, что повышает качество пользовательского опыта.

9.2.4. Особенности проектирования голосовых ботов

Как показано в примерах из предыдущего раздела, существует несколько ключевых факторов, которые необходимо учитывать, чтобы добиться положительного пользовательского опыта при взаимодействии с голосовым ботом. Прежде всего важны ясность и краткость коммуникации. Бот должен разбивать информацию на небольшие пошаговые инструкции, чтобы пользователи могли легко следовать им и понимать каждый этап процесса. Контекстная осведомленность — еще один важный момент: бот должен распознавать прогресс пользователя и предлагать подсказки, соответствующие ситуации. Кроме того, реализация механизмов обработки ошибок и предоставление пользователю возможности запросить уточнение или повторение прозвучавшей информации будут полезны тем, кто нуждается в дополнительных разъяснениях. Важно также проектировать голосовой интерфейс в естественной, разговорной манере, чтобы сделать взаимодействие интуитивным и менее «роботизированным». Наконец, мультимодальная поддержка, например ссылки на видеоинструкции или визуальные подсказки, позволяет учитывать различные формы восприятия информации и повышает общую эффективность голосового бота. С учетом всех этих аспектов разработчики смогут создавать голосовые взаимодействия, которые будут одновременно эффективными, удобными и адаптируемыми к самым разным потребностям и обстоятельствам пользователей.

Пять главных советов в проектировании голосового бота:

- *Предоставляйте пошаговые инструкции*. Голосовые боты должны разбивать задачи на небольшие этапы и последовательно вести пользователей через каждый из них. Это особенно важно, поскольку пользователь не может «просматривать» шаги, как при текстовом взаимодействии. Если он будет подтверждать выполнение каждого шага перед переходом к следующему, то не запутается в деталях.

- *Предусматривайте надежную обработку ошибок.* Ошибки распознавания речи — частое явление в голосовых интерфейсах из-за разных акцентов, особенностей речи или фонового шума. Необходимо спроектировать стратегии, которые позволят корректно и без лишних проблем для пользователя обрабатывать такие ситуации — уточняющими вопросами или упрощенными ответами, позволяющими вернуться к нужной теме.
- *Учитывайте адаптивное время отклика.* Голосовые боты должны управлять темпом общения. Так как пользователи не могут «перечитывать» услышанное как текст, бот должен контролировать скорость речи, делать естественные паузы для осмысления информации и понимать сигналы, что пользователь не успевает.
- *Подтверждайте действия перед их выполнением.* Голосовые боты должны получать подтверждение от пользователя перед выполнением важных операций. Это особенно важно для предотвращения ошибок, вызванных неверным распознаванием голосовых команд.
- *Поддерживайте различные паттерны речи.* Система распознавания голоса должна быть обучена понимать разные акценты, диалекты и особенности произношения. Такая инклюзивность обеспечивает эффективное взаимодействие с ботом для максимально широкой аудитории.

Стоит отметить, что все соображения, касающиеся проектирования голосовых ботов, в равной степени применимы и к проектированию чат-ботов. Обе модальности опираются на одни и те же ключевые элементы: анализ потребностей пользователей, контекстная осведомленность, обработка естественного языка и возникающих ошибок. Кроме того, такие факторы, как персонализация, доступность, бесшовные переходы между каналами взаимодействия и непрерывное улучшение, также играют важную роль в создании эффективных и удобных виртуальных помощников, независимо от того, основаны они на голосе или тексте.

И тем не менее некоторые моменты необходимо адаптировать специально под голосовые взаимодействия. В отличие от текстовых чат-ботов, голосовые боты должны справляться с вариациями в произношении, акцентами и фоновым шумом, которые могут повлиять на понимание и, следовательно, пользовательский опыт. Наиболее важными характеристиками голосового взаимодействия являются краткость и ясность, поскольку пользователям трудно удерживать в памяти длинные отрезки речи. В то время как текстовые чат-боты могут предоставлять более детальную информацию, которую пользователь может прочитать в удобном ему темпе и при необходимости вернуться к предыдущим сообщениям, голосовые боты требуют большей лаконичности. Адаптируя эти принципы к уникальным особенностям каждой модальности, разработчики смогут создавать более интуитивно понятных и эффективных виртуальных помощников.

Задания

1. Ознакомьтесь со следующими сценариями и определите, какая модальность (текстовая, голосовая, визуальная или мультимодальная) будет наиболее эффективной для каждого из них. Кратко объясните свой выбор:
   - Сценарий 1: Пользователю требуется помощь в установке сложного программного обеспечения.
   - Сценарий 2: Пользователь находится за рулем и ищет ближайшую заправку.
   - Сценарий 3: Пользователь хочет посмотреть новый каталог одежды.
   - Сценарий 4: Пользователь хочет каждый день получать мотивирующую цитату.
2. Выберите задачу, с которой бот может помочь пользователю, например бронирование авиабилета или выбор кредитной карты. Разработайте мультимодальное взаимодействие, включающее как минимум две разные модальности. Объясните, как каждая из них улучшает пользовательский опыт и способствует выполнению задачи. Например:
   - Задача: бронирование авиабилета.
   - Взаимодействие: виртуальный помощник использует голос для уточнения исходных данных (например, пункта назначения и дат) и текст для отображения подходящих рейсов с изображениями и ценами.
   - Объяснение: голосовое взаимодействие обеспечивает быстрый и естественный сбор информации, тогда как текст и визуальные элементы помогают пользователю сравнить варианты и принять осознанное решение.

9.3. Повышение контекстной осведомленности и улучшение общего пользовательского опыта с помощью RAG

По мере развития виртуальных помощников все более важным становится создание адаптивных и контекстно-осведомленных взаимодействий. Термин «адаптивные потоки» (adaptive flows) означает способность виртуального помощника динамически изменять траекторию взаимодействия на основе контекста и поведения пользователя в режиме реального времени. Это позволяет ботам предоставлять более персонализированные и релевантные ответы. С помощью таких передовых технологий, как RAG (глава 6), системы могут получать и использовать большое количество контекстной информации, что повышает их способность понимать и соответствующим образом реагировать на запросы пользователей. Такой подход не только повышает точность и релевантность взаимодействий, но и значительно улучшает общий пользовательский опыт.

RAG объединяет сильные стороны систем поиска и генеративных моделей, позволяя виртуальным помощникам извлекать релевантную информацию из

внешних источников и формировать контекстно-насыщенные ответы. Этот подход обеспечивает более глубокое понимание пользовательских запросов и более точные ответы, поскольку помощник опирается на широкую базу знаний, выходящую за рамки данных для предобучения. Если внедрить RAG в адаптивные потоки, боты смогут непрерывно обучаться и адаптироваться к новой информации, что гарантирует актуальность и релевантность взаимодействий. В этом разделе рассматриваются принципы построения адаптивных потоков и способы применения RAG для создания более гибких и контекстно-осведомленных виртуальных помощников.

9.3.1. Проектирование адаптивных потоков с использованием RAG

Создание адаптивных потоков требует особого подхода к проектированию. Благодаря адаптивности пользователи получают персонализированные, релевантные и быстрые ответы, а это повышает качество взаимодействия. В отличие от статических потоков, где диалог следует заранее определенному сценарию, адаптивные более гибкие и отзывчивые, что позволяет виртуальному помощнику эффективно реализовывать самые разные сценарии.

В качестве примера рассмотрим виртуального помощника интернет-магазина. В статическом потоке бот может просто уточнить у пользователя интересующую его категорию товара и показать список доступных позиций:

```
Пользователь: Привет, мне нужен новый ноутбук.
Чат-бот: Отлично! У нас есть как игровые модели, так и ноутбуки для работы
и повседневного использования. Какой вам нужен?
```

Статический поток обеспечивает базовое универсальное взаимодействие. Чат-бот предлагает стандартные варианты, не учитывая поведение или предпочтения пользователя. Пользователю приходится запрашивать дополнительную информацию о каждом товаре отдельно, что занимает время и снижает вовлеченность. Такой подход не принимает во внимание контекстные данные, например предыдущие взаимодействия пользователя или его интересы, что приводит к шаблонному ответу, не соответствующему индивидуальным потребностям.

В адаптивном потоке бот может учитывать предыдущие покупки, предпочитаемые бренды и текущие акции, тем самым предоставляя персональные рекомендации. Если ранее пользователь интересовался экологичными товарами, помощник может отдать приоритет именно таким вариантам:

```
Пользователь: Привет, мне нужен новый ноутбук.
Чат-бот: С возвращением! В прошлый раз вы просматривали экологичные модели. Все
еще интересуетесь экологически безопасными товарами?
```

Адаптивный поток использует контекстную информацию для создания более персонализированного и релевантного взаимодействия. Запоминая интерес

пользователя к экологичным товарам, чат-бот предлагает соответствующие рекомендации. Кроме того, адаптивный поток может проактивно предоставлять дополнительную информацию, например отзывы покупателей и сравнение моделей, без явного запроса. Такой опережающий подход делает взаимодействие более эффективным и приятным, поскольку бот заранее удовлетворяет потребности пользователя. В целом адаптивный поток демонстрирует, как использование контекста и персонализации способно значительно повысить качество работы виртуального помощника.

Для реализации адаптивного ответа промпт для RAG должен содержать инструкцию, чтобы модель учитывала историю пользователя (предыдущие поиски) при генерации ответа. Так помощник будет не только удовлетворять текущие потребности клиента, но и предвосхищать их:

```
{
  "user_query": "Привет, мне нужен новый ноутбук.",
  "context": "предыдущие поиски: экологичные ноутбуки",
  "generate_response": true
}
```

Другой пример — техподдержка. Статический поток может только проводить пользователя через стандартный сценарий устранения неполадок, тогда как адаптивный динамически подстраивается под конкретное устройство пользователя, его предыдущие обращения и текущие диагностические данные. Если пользователь часто обращается по поводу проблем с сетью, система может заранее проверить сетевые настройки и предложить подходящие решения, тем самым экономя время и повышая эффективность работы.

Различия между статическими и адаптивными потоками на начальных этапах взаимодействия доказывают преимущество адаптивного дизайна. Использование RAG позволяет системе эффективно задействовать в своей работе контекстные данные и обеспечивать более персонализированный и привлекательный пользовательский опыт. Разница очевидна: статические потоки предлагают единообразный подход, тогда как адаптивные выстраивают взаимодействие в соответствии с индивидуальными потребностями, предпочтениями и предыдущим опытом пользователя.

Создание адаптивных потоков на основе RAG требует сочетания эффективных механизмов поиска с современными возможностями генерации естественного языка. RAG укрепляет способность помощника извлекать релевантные данные из внешних источников и генерировать контекстно подходящие ответы, обеспечивая более динамичное и гибкое взаимодействие. Пошагово процесс проектирования адаптивных потоков с использованием RAG выглядит следующим образом:

1. *Определить элементы контекста, влияющие на взаимодействие*. К ним относятся предпочтения пользователя, история его взаимодействий, текущие данные, а также внешние источники информации. Например, для виртуального

помощника, работающего в области здравоохранения, такими элементами являются медицинская история пользователя, текущие показатели здоровья, сезонные тенденции и последние медицинские исследования. Учитывая всю эту информацию, бот сможет предоставить персональные рекомендации и напоминания.

2. *Разработать механизмы поиска информации*. RAG позволяет помощнику в режиме реального времени обращаться к внешним базам данных, документам или API, чтобы получать актуальные сведения. Например, бот интернет-магазина может находить последние отзывы о товарах, информацию об их наличии на складе или актуальные скидки из базы данных продавца. Эти сведения будут затем использоваться для формирования персональных рекомендаций.
3. *Использовать полученные данные для генерации связных и контекстно-релевантных ответов*. На этом этапе мы обращаемся к возможностям генерации естественного языка. Например, если пользователь запрашивает рекомендации по экологически безопасным товарам, помощник не просто перечисляет варианты, но и отмечает их характеристики, относящиеся к экологии, — перерабатываемая упаковка или энергосбережение, — что делает ответ более содержательным и полезным.

Непрерывное обучение и использование обратной связи имеют ключевое значение для поддержания эффективности адаптивного потока. Помощник должен обучаться на реальных взаимодействиях, корректируя свои ответы с учетом полученной обратной связи и меняющихся предпочтений клиентов. Так, например, если пользователи техподдержки постоянно отмечают определенное решение как наиболее полезное, помощник должен отдавать ему приоритет в будущих взаимодействиях. Кроме того, пользовательская обратная связь может применяться для настройки алгоритмов поисков и генерации данных, что повышает эффективность системы.

Используя RAG в своих адаптивных потоках, виртуальные помощники способны обеспечивать более персонализированные, релевантные и оперативные взаимодействия, значительно улучшая пользовательский опыт. Такой подход не только повышает точность и качество ответов, но и делает бота более интуитивно понятным, удобным и способным адаптироваться к уникальным потребностям каждого пользователя.

9.3.2. Стратегии поиска и генерации контекстно-релевантных ответов

В реальной практике баланс между персонализацией и масштабируемостью имеет решающее значение для обеспечения высококачественного пользовательского опыта при сохранении эффективности работы системы. Персонализация

означает настройку ответов под каждого пользователя на основе его предпочтений, поведения и контекста, что делает взаимодействие более привлекательным и осмысленным. Масштабируемость, в свою очередь, гарантирует, что такие персонализированные взаимодействия будут доступны для широкой и разнообразной пользовательской базы без значительного увеличения затрат или снижения эффективности.

Одним из наиболее продуктивных способов достижения этого баланса является использование механизмов контекстной осведомленности в режиме реального времени. Виртуальный помощник может динамически адаптировать свои ответы, используя актуальные данные, такие как текущее местоположение пользователя, время суток или выполняемое им действие. Например, бот туроператора может предоставлять разные рекомендации в зависимости от того, находится ли пользователь в аэропорту или просматривает информацию из дома. Контекстная осведомленность в режиме реального времени усиливает персонализацию при одновременном сохранении масштабируемости системы.

Еще один эффективный способ генерации релевантных ответов заключается в учете истории и предпочтений пользователя. Анализируя прошлые взаимодействия, историю покупок, часто задаваемые вопросы и типичное поведение в процессе коммуникации, виртуальный помощник может корректировать свои ответы в соответствии с ожиданиями пользователя. Например, если пользователь часто интересуется веганскими рецептами, бот может в будущем отдавать приоритет именно таким вариантам, делая взаимодействие более естественным и персонализированным.

Одной из ключевых задач при внедрении чат-ботов, использующих большие языковые модели, является умение отличать простые запросы от сложных для оптимизации вычислительных ресурсов. Чтобы поддерживать свою отзывчивость и экономическую эффективность, система должна обрабатывать простые запросы с помощью быстрых и конкретных ответов, избегая избыточных вычислительных затрат на базовые взаимодействия. Это позволяет направлять ресурсы туда, где они наиболее необходимы, — на генерацию высокоперсонализированных ответов на более сложные запросы.

Сложные многоэтапные запросы требуют более глубокого понимания и тщательной обработки. Такие запросы часто включают несколько уровней контекста и последовательную информацию, которую система должна точно интерпретировать и интегрировать. Недостаточная обработка таких взаимодействий может привести к неполным или неточным ответам, что разочарует пользователей и снизит доверие к возможностям системы.

И хотя архитектура RAG расширяет возможности чат-бота за счет поиска релевантной информации, у нее есть одно важное ограничение: она не до конца понимает глубинные потребности и контекст взаимодействия. RAG в первую очередь направлена на поиск корректной информации без учета тонких нюансов пользовательского намерения и контекста. Это может приводить к ситуациям,

когда чат-бот предоставляет точные данные, но не соответствует ожиданиям пользователя по части решения проблемы или автономного выполнения задачи. Современные пользователи все чаще ожидают, что чат-боты будут не только предоставлять необходимую информацию, но и активно управлять процессом и решать их запросы.

> **СОВЕТ** Сочетайте RAG с другими технологиями, например с контекстно-зависимыми фреймворками, чтобы создать чат-бота, который будет не просто отзывчивым, а по-настоящему интерактивным и адаптивным. Гибридный подход позволяет использовать сильные стороны RAG в поиске информации, дополняя их возможностями глубинного контекстного понимания и адаптивных ответов.

С технической точки зрения это подразумевает интеграцию устойчивых механизмов поиска RAG с современными возможностями обработки естественного языка, семантического анализа и контекстного моделирования. Например, RAG будет извлекать необходимые данные из обширной базы, а другие компоненты системы — интерпретировать тон пользователя с учетом его истории взаимодействий и текущего контекста. В результате мы получим ответ, который будет одновременно точным, эмпатичным и релевантным текущей ситуации.

Такое сочетание позволяет чат-боту динамически менять стиль взаимодействия в зависимости от потребностей клиента. Так, например, если он задает простой фактический вопрос, чат-бот может быстро выдать ответ с помощью RAG. Но если пользователь проявляет признаки замешательства или нуждается в дополнительной помощи, компонент, отвечающий за контекстную осведомленность, может предложить пояснение, дополнительные материалы или даже самостоятельно выполнить задачу.

Реализация такой гибридной системы требует сложной архитектуры, в которой RAG отвечает за первичное извлечение информации, а другие компоненты обрабатывают и уточняют эти данные с учетом контекстных сигналов. Такой подход позволяет чат-боту справляться с широким спектром взаимодействий — от простых запросов до сложных сценариев решения проблем, выводя пользовательский опыт на новый уровень вовлеченности и удовлетворенности.

9.3.3. Сопровождение и обновление адаптивных потоков

Сопровождение и обновление адаптивных потоков в виртуальных помощниках, особенно использующих RAG, имеет ключевое значение для обеспечения долгосрочной правильности и релевантности ответов и удовлетворенности пользователей. По мере изменения характера взаимодействий с пользователями важно постоянно совершенствовать систему, чтобы она могла адаптироваться к новым контекстам и предоставлять точные ответы. Как и в случае со статическими потоками, необходимо отслеживать взаимодействия пользователей для получения информации о том, насколько эффективно работают адаптивные потоки. На основе этих данных можно итеративно вносить изменения

в работу чат-бота, обеспечивая соответствие системы ожиданиям и потребностям пользователей.

Поскольку RAG полагается на поиск информации в режиме реального времени, базы данных и источники информации, используемые этой архитектурой, должны регулярно обновляться. Устаревшие данные могут привести к неточным ответам и подорвать доверие пользователей. Поэтому необходимо постоянно обновлять базу знаний, пополняя ее наиболее актуальной информацией из проверенных источников. Это особенно важно в сферах, где информация быстро меняется, например в новостях, здравоохранении и технологиях.

Система должна отслеживать и обновлять контекст на протяжении всего диалога. Поддержание контекста означает не только запоминание предыдущих взаимодействий, но и корректировку ответов с учетом новой информации. Для этого потребуется более сложная обработка естественного языка и сложные структуры данных, позволяющие помощнику распознавать контекстные сигналы (например, ссылки на предыдущие сообщения) и соответствующим образом обновлять свое состояние. Контекстно-осведомленные алгоритмы позволяют боту извлекать и интерпретировать релевантные детали, даже если они распределены по нескольким сообщениям. Динамическое управление контекстом гарантирует, что ответы остаются точными и релевантными, улучшая общий пользовательский опыт.

Эффективное управление контекстом также предполагает адаптацию к изменениям пользовательских намерений в режиме реального времени. Помощник должен распознавать, когда предпочтения пользователя меняются, и в соответствии с этим корректировать свое понимание. Например, если пользователь изначально интересовался бюджетными вариантами путешествий, но затем упомянул о том, что предпочитает отели подороже, бот должен скорректировать свои рекомендации. Реализация механизмов динамического управления контекстом предполагает использование моделей машинного обучения, способных изучать и прогнозировать предпочтения пользователей, а также добавление обратной связи для анализа понимания бота. Такая адаптивность является ключом к поддержанию релевантности и связности ответов, делая виртуального помощника более эффективным и интуитивно понятным инструментом.

Благодаря стратегиям, описанным в этой главе, адаптивные потоки в виртуальных помощниках будут оставаться точными, релевантными и «чуткими» к изменяющимся потребностям пользователей. Основные практики включают поддержание контекстной осведомленности системы в режиме реального времени, использование истории и предпочтений пользователей, а также динамическую адаптацию к изменяющимся входным данным. Кроме того, RAG значительно расширяет возможности виртуальных помощников за счет генерации контекстно релевантных ответов на основе найденной информации. Тем не мнее, как было отмечено ранее, одной лишь RAG недостаточно. Эффективные системы должны

сочетать ее с механизмами глубокого понимания контекста, возможностью выполнения задач и моделями, адаптированными под данные конкретных пользователей. Настройка языковых моделей с учетом индивидуальных особенностей и предпочтений клиентов позволяет ботам выдавать более точные, актуальные и персонализированные ответы. Сопровождение и регулярное обновление системы являются обязательными условиями для поддержания высокой эффективности помощника и удовлетворенности пользователей. Следуя этим принципам, виртуальные помощники смогут обеспечить более эффективные, естественные и вовлекающие взаимодействия.

Задания

1. Спроектируйте взаимодействия для контекстно-осведомленного виртуального помощника. Выберите сценарий, например бронирование путешествий или предоставление техподдержки клиентам, а затем создайте поток диалога, демонстрирующий, как виртуальный помощник запоминает предыдущие взаимодействия и адаптируется к новой информации. Ваш сценарий должен включать не менее пяти взаимодействий с пользователем и иллюстрировать следующее:
   - Как бот отслеживает и поддерживает контекст (например, запоминает предпочтения пользователя или ранее заданные вопросы).
   - Как он динамически обновляет контекст, когда пользователь предоставляет новые сведения.
   - Как это улучшает пользовательский опыт и делает взаимодействие более плавным и релевантным.

   В конце потока объясните, как виртуальный помощник управляет контекстом на протяжении всего диалога и почему такой подход повышает согласованность системы и удобство ее использования.
2. Разработайте масштабируемую архитектуру для виртуального помощника, включая механизмы хранения данных, обработки и поиска информации.

Итоги

- Понимание и использование контекстной информации, такой как местоположение пользователя, его предыдущие взаимодействия и предпочтения, имеет решающее значение для выдачи персонализированных и релевантных ответов, а также для повышения общего уровня удовлетворенности пользователей.
- Проектирование виртуальных помощников, способных эффективно обрабатывать текстовые, голосовые, визуальные и мультимодальные взаимодействия, обеспечивает более привлекательный и гибкий пользовательский опыт. Адаптация дизайна взаимодействия под особенности каждой модальности является ключом к удовлетворению разнообразных потребностей пользователей.

- Использование RAG в адаптивных потоках позволяет виртуальным помощникам получать точную информацию и формировать контекстно уместные ответы. Благодаря такому гибридному подходу бот способен эффективно обрабатывать как простые, так и сложные запросы.
- Внедрение надежных механизмов динамического управления контекстом позволяет виртуальным помощникам поддерживать и обновлять контекст на протяжении всего взаимодействия. Это необходимо для предоставления связных, релевантных ответов и адаптации к изменяющимся потребностям пользователей в режиме реального времени.
- Регулярный мониторинг, учет обратной связи, адаптивное обучение и оптимизация имеют решающее значение для поддержания точности и актуальности адаптивных потоков. Необходимо добиваться того, чтобы ответы, основанные на RAG, оставались точными и контекстно уместными на всем протяжении диалога, тем самым поддерживая высокий уровень удовлетворенности пользователей.

10 Снижение сложности системы с помощью генеративного ИИ

В этой главе

- ✓ Проектирование и улучшение процессных потоков с помощью генеративного ИИ
- ✓ Замена диалоговых потоков для разрешения многозначности на LLM
- ✓ Тестирование статических диалоговых потоков с генеративным ИИ в роли «пользователя»

Построить процессно-ориентированного бота, который удовлетворял бы всем потребностям и ожиданиям стейкхолдеров, довольно сложно. Конфликт приоритетов может привести к так называемому коллективному дизайну, результатом которого станет избыточная сложность системы. И даже исходя из самых лучших побуждений разработчики все равно могут неосознанно внести в нее редкие граничные случаи, мешающие основному потоку диалога. Такие сложности затрудняют использование бота и повышают вероятность того, что пользователь бросит сеанс и не завершит задачу. Генеративный ИИ может помочь выявить и улучшить чрезмерно сложные сценарии, повышая эффективность работы системы.

Разработчики процессных потоков часто запрашивают у пользователя слишком много информации. (Чем больше информации, тем лучше, так ведь? Нет,

не так, особенно если из-за этого чат-бот не справляется со своей задачей!) Существует несколько способов улучшить процессные потоки с помощью генеративного ИИ:

- генерация идей для построения процессного потока;
- генерация идей для улучшения существующих потоков. А с помощью генеративного ИИ, выступающего в роли пользователя, можно также их протестировать;
- замена некоторых статических процессных потоков на процессы, управляемые большой языковой моделью (LLM).

В качестве примера возьмем сценарий проверки статуса страхового заявления в компании медицинского страхования и рассмотрим, как генеративный ИИ может помочь спроектировать и улучшить этот поток и ему подобные.

10.1. Разработка процессных потоков с поддержкой ИИ

На рис. 10.1 показана базовая схема процессного потока.



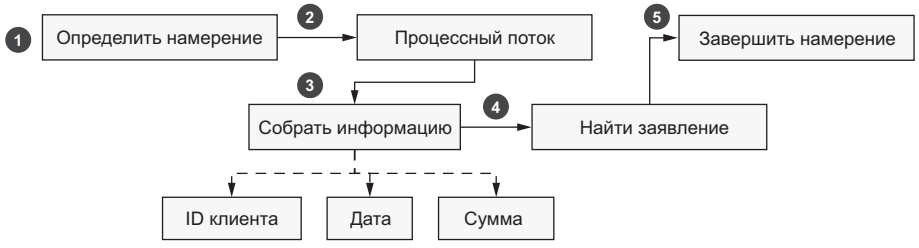
Рис. 10.1. Общая схема процессного потока, который начинается с распознавания конкретного намерения, включает один или несколько последовательных шагов и завершается выполнением процесса (удовлетворением намерения)

В нашем примере процессный поток включает клиентов медицинской страховой компании, которые обращаются к чат-боту, чтобы узнать статус своего заявления. На первый взгляд процесс кажется довольно простым, но он состоит из нескольких этапов:

- *Определение намерения* — необходимо распознать, что пользователь хочет узнать статус своего заявления. Это запускает процессный поток, состоящий из нескольких шагов.
- *Начало процессного потока* — сбор информации, необходимой для выполнения запроса о статусе заявления; в нашем случае — данных, необходимых для поиска.
- *Середина процессного потока* — использование собранной информации для выполнения действия. В нашем примере это поиск страхового заявления пользователя.
- *Завершение процессного потока* — предоставление пользователю информации о статусе его заявления.

Общий поток обработки заявлений показан на рис. 10.2.

В главе 5 было показано, как можно улучшить классификатор намерений чат-бота для более точного *распознавания* и *понимания* намерений пользователя. В этой главе внимание будет сосредоточено на совершенствовании оставшейся части процессного потока для успешного *выполнения* пользовательских запросов.



1. Бот определяет, что намерение пользователя — «статус заявления».
2. Процессный поток запускается.
3. Процесс задает сбор информации о клиенте и его заявлении.
4. Собранная информация используется для поиска заявления.
5. Найденное заявление зачитывается пользователю, поток завершается.

Рис. 10.2. Схема процессного потока проверки статуса страхового заявления

10.1.1. Генерация диалоговых потоков с помощью генеративного ИИ

Процессные потоки в разговорных системах часто строятся на основе уже существующих рабочих процессов. Они могут быть скопированы из другого канала, веб-приложения или сценария колл-центра. В нашем примере со статусом заявления предположим, что готового процесса пока нет. В таком случае можно использовать LLM, чтобы спроектировать целевой рабочий процесс.

Ниже приведен пример промпта для LLM.

Листинг 10.1. Промпт для создания потока проверки статуса заявления

```
Инструкция: Ты — разработчик разговорного ИИ.
➥ Ты создаешь чат-бота, который помогает пользователям находить
➥ информацию о статусе своих страховых заявлений.
➥ Страховые заявления содержат следующую информацию:
➥ дату подачи, ID клиента, сумму возмещения и
➥ статус заявления.
Спроектируй диалоговый поток, который поможет пользователям
➥ найти их заявление.
➥ Диалог должен быть максимально коротким и
➥ удобным для пользователя.
Объясни, почему ты построил диалог именно таким образом.

Вывод: ←┤
```

Сценарий и контекст

Подробные инструкции

Сигнал для вывода

Для выполнения этой задачи LLM необходимо предоставить несколько видов информации:

- *Сценарий и контекст* — модель должна понимать, в каком контексте она работает («ты — разработчик разговорного ИИ»). Ей также следует сообщить все исходные данные, например то, какие сведения содержатся в страховых заявлениях.
- *Инструкции* — модель должна спроектировать диалог, который позволит пользователю достичь своей цели (найти заявление), при этом сделав процесс максимально коротким и простым. Кроме того, модель должна объяснить свою логику, чтобы можно было оценить полученный результат.
- *Сигнал* — слово «Вывод» указывает модели, что инструкция завершена.

Небольшие изменения могут привести к большим различиям

LLM могут выдавать совершенно разные ответы на достаточно похожие промпты. Даже изменения форматирования, например добавление пробела или включение/исключение символов новой строки, могут вызвать значительные изменения в выводе. Точные промпты, использованные в нашей книге, доступны в репозитории GitHub по адресу https://github.com/andrewrfreed/EffectiveConversationalAI.

Следующий листинг показывает вывод, который мы получили от промпта, состоящего из трех строк (строки, начинающиеся с «Инструкция», «Спроектируй» и «Объясни»).

Листинг 10.2. Вывод от mixtral-8x7b-instruct-01 для потока проверки статуса страхового заявления (меньше строк)

```
Диалоговый поток для чат-бота по проверке статуса страхового
заявления будет выглядеть следующим образом:

1. Пользователь: Здравствуйте, я хочу узнать статус моего
страхового заявления.
2. Чат-бот: Конечно, давайте я вам помогу. Пожалуйста,
введите ваш ID клиента.
3. Пользователь: Мой ID клиента 123456.
4. Чат-бот: Спасибо. Теперь скажите, пожалуйста,
дату подачи заявления.
5. Пользователь: Дата подачи — 10 января 2022 г.
6. Чат-бот: Понял. Какова сумма возмещения?
7. Пользователь: Сумма возмещения — $1000.
8. Чат-бот: Спасибо. Теперь у меня есть вся необходимая
информация. Проверяю статус вашего заявления.
9. Чат-бот: Статус вашего заявления — одобрено, сумма $1000
выплачена.
```

Пример диалога, включающий реплики как чат-бота, так и пользователя

```
Диалоговый поток разработан так, чтобы быть кратким и удобным
для пользователей. Чат-бот начинает с запроса ID клиента,
который является самым важным элементом для идентификации
заявлений. Затем чат-бот спрашивает дату подачи заявления и сумму
возмещения, необходимые для поиска конкретного заявления.
Запрашивая информацию в таком порядке, чат-бот сможет быстро
и эффективно получить сведения о заявлении пользователя.

Кроме того, чат-бот подтверждает полученную информацию перед
проверкой статуса. Это снижает вероятность ошибки и гарантирует,
что пользователь получит точный ответ. Чат-бот также дает
понятные и краткие инструкции на протяжении всего диалога,
благодаря чему пользователю легко следовать по потоку.

Таким образом, диалоговый поток позволяет обеспечить плавный
и эффективный пользовательский опыт, при этом гарантируя
точность и достоверность предоставляемой информации.
```

Объяснение сгенерированного потока

LLM сгенерировала пример двустороннего диалога, состоящего из реплик бота и пользователя. Полезное решение — так нам легче представить, как будет выглядеть полный диалог. Однако следует помнить, что это всего лишь пример, — пользователи могут отвечать на вопросы самыми разнообразными способами.

Итак, наше взаимодействие с LLM включало следующее:

- Мы сообщили LLM, что заявления состоят из ID клиента, даты подачи, суммы возмещения и статуса. Модель сделала вывод, что статус — это выходной элемент, а остальные три — входные.
- LLM спроектировала поток, который последовательно собирает все три входных элемента.
- Сгенерированный диалог вежливый и эффективный. Бот подтверждает ввод пользователя словом «спасибо», а на каждом шаге выдает четкие инструкции.

Интересно, что вывод существенно меняется в зависимости от того, как мы используем символы новой строки в промпте. Так, например, листинг ниже содержит вывод от промпта, состоящего из шести строк (новая строка после каждой точки в инструкции).

Листинг 10.3. Вывод от mixtral-8x7b-instruct-01 для потока проверки статуса заявления (больше строк)

```
1. Диалоговый поток начинается с приветствия и введения в назначение чат-бота.

Чат-бот: Здравствуйте! Я могу помочь вам узнать статус ваших страховых
заявлений. Пожалуйста, введите ваш ID клиента.

Объяснение: Начав с приветствия и назначения бота, пользователь понимает
его цель и чего ожидать от диалога. Запрос ID клиента первым позволяет
идентифицировать аккаунт пользователя и сузить поиск по заявлениям.
```

```
2. После того как пользователь предоставит ID клиента, чат-бот подтверждает его
получение и спрашивает дату подачи заявления.

Чат-бот: Спасибо, я получил ваш ID. Чтобы я мог найти ваше заявление, мне нужно
знать дату его подачи.

Объяснение: Подтверждение получения ID клиента гарантирует правильность
введенных данных и способствует формированию доверия. Запрос даты подачи
заявления позволяет еще больше сузить поиск.

3. После того как пользователь сообщит дату подачи заявления, чат-бот
подтверждает ее и запрашивает сумму возмещения.

Чат-бот: Спасибо, я записал дату подачи заявления. Сумма возмещения составляла
[вставьте сумму]. Все верно?

Объяснение: Подтверждение даты и суммы позволяет убедиться в правильности данных
и снижает вероятность ошибки.

4. После того как пользователь подтвердит сумму возмещения, чат-бот
предоставляет статус заявления.

Чат-бот: Отлично, сумма подтверждена. Статус вашего заявления — [вставьте
статус]. Могу ли я еще чем-то помочь?

Объяснение: Предоставление статуса заявления — конечная цель диалога, и важно
достичь ее как можно скорее. Спрашивая, нужно ли пользователю что-то еще, чат-
бот дает возможность запросить дополнительную помощь.

Таким образом, данный диалоговой поток разработан так, чтобы быть кратким
и простым. Запрашивая наиболее важную информацию вначале (ID клиента и дату
подачи заявления), чат-бот может быстро сузить поиск. Подтверждение ID клиента,
даты и суммы позволяет гарантировать правильность введенных данных и уменьшить
вероятность ошибки. Предоставление статуса заявления — конечная цель диалога,
и чат-бот делает это максимально оперативно. Наконец, вопрос о том, нужно ли
пользователю что-то еще, позволяет при необходимости предоставить дополнительную
помощь.
```

Вывод в листинге 10.3 довольно сильно отличается от предыдущего. Во-первых, реплики диалога и их объяснение перемешаны. Для каждого шага приведено его описание, пример реплики бота и подробное пояснение. Во-вторых, пример содержит только сообщения бота — мы не видим ответов пользователя. В-третьих, этот поток подразумевает подтверждение после каждого полученного от пользователя вида данных. Наконец, в примере диалога есть небольшие ошибки. В пункте 3 инструкция говорила, что будет запрошена сумма возмещения, но в диалоге показано подтверждение суммы без ее предварительного запроса.

Вместе оба промпта дают нам достаточно идей, чтобы спроектировать наш собственный диалоговый поток для проверки статуса заявления. Поскольку вывод в листинге 10.2 более полный, мы возьмем за основу именно его. Поток

получился неплохой, но, возможно, немного длинный. Можем ли мы его улучшить? Конечно! Попросим LLM предложить возможные изменения.

10.1.2. Улучшение диалогового потока с помощью генеративного ИИ

LLM могут помочь нам в улучшении процессных потоков, независимо от того, были ли они созданы человеком или моделью. Давайте усовершенствуем процессный поток из листинга 10.2, сгенерированный с помощью LLM, чтобы пользователям было удобнее узнавать статус своих заявлений.

Текущий сценарий задействует три вида данных: ID клиента, дату подачи заявления и сумму возмещения. Это соответствует базовым требованиям бота — аутентификации пользователя (по ID клиента) и критериям поиска (ID, дата и сумма). Тем не менее этот процесс достаточно обременителен для пользователя. Возможно, ID клиента и еще какого-то одного вида данных уже будет достаточно для поиска заявления. Давайте попросим LLM упростить поток.

Листинг 10.4. Запрос к LLM для улучшения диалогового потока из листинга 10.2

```
<<Промпт из листинга 10.1>>
<<Вывод из листинга 10.2>>

Инструкция: Упрости данный диалоговый поток.
Запроси у пользователя только два вида данных.

Вывод:
```

Все содержимое листинга 10.1 (опущено для краткости)

Все содержимое листинга 10.2 (опущено для краткости)

Простая инструкция для LLM

Сигнал для вывода

Промпт довольно простой. Мы имитируем разговор с LLM и включаем в запрос всю историю предыдущего взаимодействия: исходный промпт и вывод модели. Затем мы добавляем несложную инструкцию («упрости данный диалоговый поток») с определением того, что именно мы подразумеваем под упрощением («запроси только два вида данных»). Полученный вывод приведен в листинге ниже.

Листинг 10.5. Упрощенный диалоговый поток и объяснение, предоставленные LLM

```
Упрощенный диалоговый поток для чат-бота по поиску страховых
заявлений будет выглядеть следующим образом:

1. Пользователь: Здравствуйте, я хочу узнать статус
   моего страхового заявления.
2. Чат-бот: Конечно, давайте я вам помогу. Пожалуйста,
   введите ваш ID клиента и дату подачи заявления.
3. Пользователь: Мой ID клиента — 123456, а дата подачи заявления — 10 января
   2022 г.
```

Теперь шаг 2 собирает все данные в одном вопросе

```
4. Чат-бот: Спасибо. Теперь у меня есть вся необходимая информация. Проверяю
   статус вашего заявления.
5. Чат-бот: Статус вашего заявления — одобрено,
   сумма $1000 выплачена.
```

Шаг 5 использует сумму возмещения как выходной параметр, а не входной

```
Упрощенный диалоговый поток требует от пользователя только двух видов данных:
ID клиента и дату подачи заявления. Это сокращает количество шагов в диалоге
и упрощает процесс для пользователя. Чат-бот может использовать ID клиента
для идентификации всех заявлений пользователя, а дату подачи — для поиска
конкретного случая.

Запрашивая оба вида данных одновременно, чат-бот оптимизирует диалог
и обеспечивает более эффективный пользовательский опыт. Чат-бот по-прежнему
подтверждает полученную информацию перед проверкой статуса заявления, тем самым
обеспечивая точность и достоверность предоставленных данных.

Таким образом, упрощенный диалоговый поток способен обеспечить быстрый
и удобный пользовательский опыт, при этом сохраняя точность и достоверность
предоставляемой информации.
```

LLM дала нам две рекомендации: во-первых, не запрашивать сумму возмещения, а во-вторых, объединить два вопроса в один. Рассмотрим эти изменения подробнее:

- *Удаление суммы возмещения.* ID клиента и дата подачи заявления — данные, которые пользователь с наибольшей вероятностью знает. Также они проще всего вводятся в веб- или мобильных интерфейсах — большинство платформ для чат-ботов имеют готовые библиотеки для ID и дат. Система сама озвучивает сумму возмещения при выводе ответа.
- *Объединение двух вопросов.* Новый поток объединяет запрос ID клиента и даты подачи заявления в один вопрос (шаг 2). Это подходит опытным пользователям, желающим минимального числа шагов, но может быть сложнее для тех, кто помнит только один вид данных и кому требуется помощь в поиске второго. Хорошо, если ваш чат-бот умеет принимать оба вида в одном сообщении, но не стоит делать это обязательным требованием.

Эксперты в предметной области или LLM?

Мы рекомендуем обращаться к экспертам в вашей предметной области перед выводом какого-либо решения в продакшен. LLM больше подходят для генерации идей и их быстрой проверки. Используйте их для изучения возможных подходов и быстрого проектирования потенциальных решений.

Одним простым промптом мы получили два подхода к тому, как можно улучшить наш диалоговый поток. Может, у вас есть свои идеи? Какие инструкции вы бы дали LLM?

Задания

1. Возьмите листинг 10.4 и попробуйте предоставить модели другие инструкции:
   - Запрашивайте у пользователя только один вид данных за раз.
   - Помогайте пользователю, если он отвечает «Я этого не знаю» на один из вопросов.
   - Добавьте дополнительные параметры, например ID заявления, и посмотрите, как бот сгенерирует новые варианты потока.
2. С помощью LLM сгенерируйте процессный поток для другой ситуации, например:
   - бронирование авиабилета;
   - покупка билета в кино;
   - рекомендация места для отдыха.

   Либо используйте любой другой сценарий из вашего чат-бота!

10.2. Выполнение процессных потоков с поддержкой ИИ

Генеративный ИИ отлично справился с проектированием процессных потоков. Но до сих пор это были достаточно статичные сценарии, применяемые в традиционных разговорных ИИ. Проверка статуса заявления является примером поиска по принципу «заполнения слотов», когда мы используем диалог для сбора информации, необходимой для выполнения задачи. Зачастую это становится просто сбором параметров, нужных для вызова API. Этот подход требует тщательного сопоставления вопросов и ответов с параметрами API. Затем ответы подставляются в параметры до тех пор, пока вызов API не будет выполнен. Заполнение слотов является одним из самых распространенных процессных потоков в разговорном ИИ.

Но что, если предоставить LLM больше свободы?

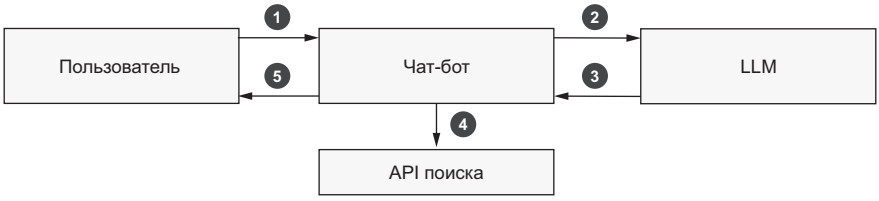
10.2.1. Выполнение диалоговых потоков с помощью генеративного ИИ

Наш предыдущий процессный поток был спроектирован статически. Теперь попробуем сделать по-другому. Мы просто опишем процесс и позволим LLM самой решать, какие вопросы задавать в ходе диалога. На рис. 10.3 показана интеграция LLM в процесс сбора информации для вызова API поиска.

Для выполнения задачи чат-боту необходимо наличие некоторой логики:

- Когда распознается намерение — проверка статуса страхового заявления, — бот позволяет LLM решать, какой вопрос задать следующим.
- Если бот обнаруживает, что LLM отвечает списком переменных, он возвращает контроль себе и выполняет поиск заявления.

- Бот использует защитные механизмы, такие как преклассификатор, чтобы убедиться, что данные, отправляемые LLM, не являются вредоносными, например «игнорируй все предыдущие инструкции и <сделать что-то вредное>».



1. Пользователь отвечает чат-боту.
2. Бот передает диалог LLM и запрашивает следующий ответ.
3. LLM генерирует ответ.
4. Если информации достаточно, выполняется поиск.
5. Чат-бот отвечает пользователю.

Рис. 10.3. Как разговорный ИИ может использовать LLM для принятия решения о следующем вопросе

В следующем листинге показано, как LLM шаг за шагом генерирует диалог.

Листинг 10.6. LLM самостоятельно решает, какие вопросы задавать для поиска заявления

```
Инструкция: Ты — разработчик разговорного ИИ. Ты создаешь чат-
бота, который помогает пользователям находить информацию о статусе
своих страховых заявлений.
Страховые заявления содержат следующую информацию: дату подачи, ID
клиента и сумму возмещения.
Генерируй реплики диалога до тех пор, пока не соберешь
все три вида данных.
Когда вся информация будет собрана, ответь полезной
нагрузкой в следующем формате:
{"memberID": "(полученный ID клиента)", "claimDate":
"(полученная дата подачи заявления)", "claimAmount":
"(полученная сумма возмещения)"}

Чат-бот: Чем я могу вам помочь?
Пользователь: Мне нужна информация по моему заявлению.
Чат-бот: ◄──
```

Инструкция из предыдущих промптов

Описание новой задачи

Сигнал для вывода

При статичном приветствии бота мы указываем само приветствие и первую реплику пользователя

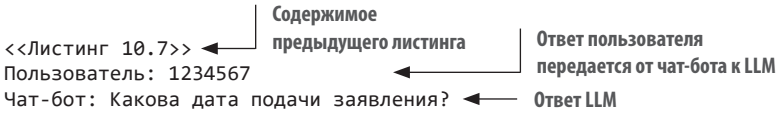
Такого простого промпта достаточно, чтобы бот начал генерировать диалог. (Мы написали этот промпт всего за пару минут.) Листинги ниже показывают вывод после каждой итерации. Листинг 10.7 содержит первый шаг диалога, следующий за выводом из листинга 10.6.

Листинг 10.7. Вывод первого шага диалога (LLM генерирует следующий вопрос)

```
<<Листинг 10.6>>  ◄── Содержимое предыдущего листинга
Чат-бот: Я вам помогу. Какой у вас ID клиента?  ◄── Вывод из первого промпта
```

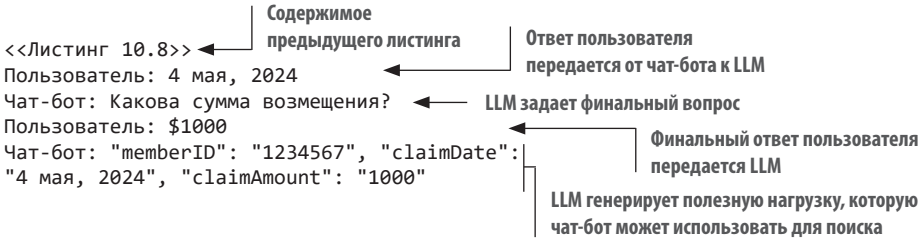
Пока что бот справляется хорошо. Он решил начать с ID клиента. Посмотрим, что он сделает далее.

Листинг 10.8. Вывод второго шага диалога (LLM генерирует следующий вопрос)



И снова подходящий ответ. LLM хорошо справляется, при этом не имея ничего другого, кроме исходного промпта. Реплики немного повторяются — оба вопроса сформулированы как «Каков/Какой <X>». Посмотрим на остальную часть диалога.

Листинг 10.9. Вывод шагов 3 и 4



В итоге LLM сделала все, что нам было нужно. Она задавала вопросы без использования заранее прописанных реплик и сформировала набор параметров, который можно передать в API для поиска заявления. (Она не сгенерировала корректный JSON, но наверняка сделала бы это при более подробном промпте.)

Может возникнуть соблазн предоставить LLM полную свободу действий. Но прежде чем решиться на это, посмотрим, как модель поведет себя в другой ситуации. На этот раз пользователь не будет знать все, что требуется системе. Как в таком случае поступит LLM? Подобный сценарий приведен в листинге ниже.

Листинг 10.10. Ответы LLM, когда у пользователя нет всей необходимой информации

```
Инструкция: Ты — разработчик разговорного ИИ. Ты создаешь чат-бота, который
    помогает пользователям находить информацию о статусе своих страховых
    заявлений.
```

```
Страховые заявления содержат следующую информацию: дату подачи, ID клиента
    и сумму возмещения.
Генерируй реплики диалога до тех пор, пока не соберешь все три типа данных.
Когда вся информация будет собрана, ответь полезной нагрузкой
в следующем формате:
{"memberID": "(полученный ID клиента)", "claimDate":
"(полученная дата подачи заявления)", "claimAmount":
"(полученная сумма возмещения)"}
```

Та же инструкция, что и в предыдущих примерах

```
Чат-бот: Чем я могу вам помочь?
Пользователь: Мне нужна информация по моему заявлению.
Чат-бот: Я вам помогу. Какой у вас ID клиента?
Пользователь: Не знаю.
Чат-бот: Какова дата подачи заявления?
Пользователь: Тоже не знаю.
Чат-бот: Какова сумма возмещения?
Пользователь: $1000
Чат-бот: Я вам помогу. Какой у вас ID клиента?
```

Тот же начальный диалог, что и в предыдущих примерах

Пользователь не имеет некоторых сведений

LLM застряла!

О нет! В промпте не было указано, что делать с ошибками, поэтому LLM будет бесконечно задавать вопросы, пока пользователь не потеряет терпение и не выйдет из чата. Скорее всего, он даже не сможет отказаться от этого диалога. Теперь мы видим ограничения такого подхода. Задавать множество вопросов для выполнения поиска — метод ненадежный. Давайте попробуем что-то другое. А что, если поиск будет выполняться самой LLM?

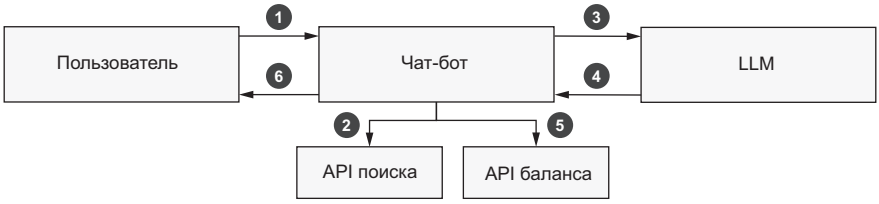
10.2.2. Использование LLM для поиска

В таких сценариях, как, например, медицинское страхование, крайне важен тщательный поиск. Поставщик медицинских услуг может иметь в своей клиентской базе сотни поданных заявлений (а то и больше). Строгие критерии поиска не только необходимы, чтобы найти нужную информацию, но и обязательны по закону. Поэтому давайте рассмотрим другой сценарий с гораздо меньшим числом вариантов для поиска.

> **Разве это не генерация, дополненная поиском (RAG)?**
>
> Отчасти да. Мы создаем текстовые «фрагменты» на основе вывода структурированных API и предоставляем LLM доступ к ним. Можно не называть это RAG, но сходство определенно есть. И что более важно, это полезный инструмент в вашем арсенале, как бы вы его ни называли.

Наш новый пример — клиенты банка, проверяющие баланс своих счетов. Обычно у одного человека может быть от одного до четырех счетов в одном банке. Чат-боту необходимо знать, о каком из них спрашивает пользователь. В данном случае имеют значение несколько метаданных: тип (расчетный или сберегательный), владелец (один или несколько) и номер (хотя владельцы могут его не помнить).

Предположим, что пользователь вошел в чат-бот (по своему ID или подтвержденному номеру телефона, и мы уже знаем, кто это) и запрашивает баланс счета. Чат-бот обращается за помощью к LLM. Схема потока показана на рис. 10.4.



1. Вопрос пользователя с описанием его счета.
2. Бот использует API, чтобы получить ВСЕ счета, к которым пользователь имеет доступ.
3. LLM получает описание пользователя и список счетов.
4. LLM выбирает наиболее вероятный счет.
5. Бот запрашивает баланс выбранного счета.
6. Чат-бот отвечает пользователю.

Рис. 10.4. Обработка пользовательских запросов с помощью LLM

Пользователь может задать боту следующие вопросы:

- Сколько денег у меня на счете?
- Сколько денег у меня на сберегательном счете?
- Сколько денег на нашем совместном сберегательном счете?
- Сколько денег на счете моего сына?
- Сколько денег на счете, который я только что открыл?

Промпт и пример вывода приведены в следующем листинге. Этот промпт выполняется со стоп-критерием по любому символу пробела (пробел или перенос строки). В противном случае LLM продолжает вывод с объяснением сделанного выбора.

Листинг 10.11. Использование LLM для поиска

```
<|инструкция|>
Ты поддерживаешь работу цифрового помощника. Пользователь
задает вопрос об одном из своих банковских счетов. Используй
предоставленную контекстную информацию, чтобы определить,
о каком счете он, вероятнее всего, спрашивает.

<|пользователь|>
Сколько денег на счете моего сына?

<|контекст|>
Имя пользователя: Боб
```

Базовая инструкция

Ввод пользователя передается непосредственно LLM

```
Счета: [ {"id":12345, "type":"checking", "owners":["Bob","Jane"],
"opened":"12/25/2000"}, {"id":23456, "type":"saving",
"owners":["Bob","Jane"], "opened":"1/3/2005"}, {"id":34567,
"type":"saving", "owners":["Bob","Jack"], "opened":"2/4/2024"}]
```

LLM получает контекст — данные вошедшего пользователя и метаданные всех его счетов

```
<|вывод|>
ID счета: 34567
```

Сигнал для вывода и вывод

Отлично! LLM может отвечать на все пять вопросов. В табл. 10.1 приведены полученные ответы. Но помните, что мы просим модель только выбрать номер счета. Чат-бот все равно будет вызывать API для проверки баланса и формулировать итоговый ответ.

Таблица 10.1. Ответы из листинга 10.11 для разных входных вопросов

Вопрос	Ответ (номер счета)
Сколько денег у меня на счете?	12345
Сколько денег у меня на сберегательном счете?	23456
Сколько денег на нашем совместном сберегательном счете?	23456
Сколько денег на счете моего сына?	34567
Сколько денег на счете, который я только что открыл?	34567

Можно сделать несколько наблюдений:

- *Вариативность*. Мы обработали несколько разных критериев поиска, включая даты, типы и владельцев, без уточняющих вопросов.
- *Гибкость*. Такие критерии, как «мой сын» или «счет, который я только что открыл», были обработаны без строгого параметра API.
- *Выбор по умолчанию*. Для двух многозначных вопросов («мой счет» и «наш совместный сберегательный счет») бот выбрал первый подходящий вариант. Следовательно, порядок сортировки важен.

LLM продемонстрировала невероятную гибкость! При низких рисках вполне можно позволить LLM проводить поиск самостоятельно. Если итоговый вывод выглядит примерно так: «На вашем <тип> счете с номером <ID> находится <баланс>», то можно справиться и без каких-либо уточнений. Пользователь всегда получает верную информацию и подтверждающие данные. Если же нужен другой счет, он может ответить, например: «Нет, я имел в виду баланс моего сберегательного счета».

Генеративный ИИ и LLM открывают разнообразные возможности для расширения функционала ваших чат-ботов. И хотя в таком случае вам придется постоянно поддерживать баланс между скоростью разработки и контролем над системой, LLM способны выполнять задачи, которые в традиционных чат-ботах были бы сложными или даже невозможными.

Безопасно ли позволять LLM выбирать ID счета? А как же галлюцинации?

В примере с запросом баланса банковского счета мы обеспечиваем безопасность, разделяя сам вызов API и логику выбора LLM. Обычный API для проверки баланса имеет два параметра: ID пользователя и номер счета. В данном сценарии мы позволяем LLM выбирать только номер счета. Так мы защищаемся от галлюцинаций LLM в комбинации «ID пользователя + номер счета», которые могут привести к утечке информации. Если LLM выдумает номер счета, API-вызов завершится с ошибкой. Если LLM выберет неверный счет пользователя, пользователь все равно получит информацию о своем счете. Обязательно тщательно тестируйте проектирование и реализацию системы, прежде чем довериться ей.

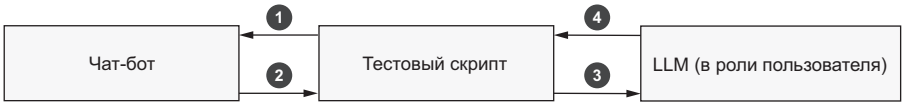
Такой «безопасный» дизайн необходим, если LLM имеет возможность выполнять вызовы API.

Задания

1. Перепишите промпт в листинге 10.6, чтобы получить более разнообразные ответы (а не только «Каков ваш <X>»).
2. Перепишите промпт в листинге 10.11 так, чтобы LLM возвращала сигнальное значение, например «не найдено», если вопрос пользователя неоднозначен. Вы также можете добавить дополнительные инструкции или применить few-shot-обучение.

10.3. Тестирование потоков с помощью ИИ

В предыдущих разделах мы использовали генеративный ИИ для проектирования и реализации решений, заставляя LLM действовать как чат-бот. В этом разделе поменяем все местами и будем применять LLM для генерации типичных или «креативных» ответов, чтобы посмотреть, как чат-бот с ними справится в нашем сценарии со страховыми заявлениями. Общая схема потока показана на рис. 10.5.



1. Тестовый скрипт отправляет сообщение чат-боту (первое сообщение просто инициирует разговор).
2. Ответ, сгенерированный чат-ботом, отправляется скрипту.
3. LLM получает инструкции и полный контекст разговора, чтобы сгенерировать следующее сообщение.
4. Ответ LLM (в роли пользователя) отправляется скрипту. Затем снова возвращаемся к шагу 1.

Рис. 10.5. Схема потока, показывающая, как тестовый скрипт вызывает LLM в роли пользователя чат-бота

Чтобы собрать такой тестовый скрипт, нам нужны три компонента: общий промпт для LLM, чтобы он действовал как пользователь; тестовый скрипт для вызова чат-бота и LLM; методология анализа результатов.

Приступим.

10.3.1. Настройка генеративного ИИ для роли пользователя

Чтобы успешно играть роль пользователя, LLM потребуется три вида данных: общие инструкции по задаче, описание тестируемого сценария и текущий диалог.

Во-первых, зададим общий контекст, указав LLM, что мы хотим, чтобы она имитировала пользователя в диалоге. Инструкция будет довольно простая:

```
Действуй как пользователь телефонного чат-бота компании по медицинскому
страхованию. Продолжи диалог возможным ответом.
```

Инструкция описывает основную задачу LLM — отвечать как пользователь, а не как система. Никаких дополнительных указаний мы не даем.

Во-вторых, нам нужно, чтобы LLM могла обрабатывать разные сценарии, поэтому промпт должен быть адаптивным. Вот несколько сценариев, которые мы хотим протестировать:

- Пользователь располагает всей необходимой информацией (ID клиента, дата подачи заявления, сумма возмещения).
- Пользователь не знает часть необходимой информации.
- Пользователь не знает часть необходимой информации, но имеет альтернативу (например, ID заявления).

Для каждого сценария мы добавим в промпт немного отличающиеся инструкции. В табл. 10.2 приведены сценарии и возможные инструкции для LLM.

Таблица 10.2. Описание сценариев и инструкций к ним

Описание	Инструкция
Пользователь располагает всей необходимой информацией	Ты хочешь узнать, было ли одобрено одно из твоих заявлений. Ты знаешь, что твой ID клиента — 123456, дата подачи заявления — 4 мая 2024 г., а сумма возмещения — $1000
Пользователь не знает часть необходимой информации	Ты хочешь узнать, было ли одобрено твое последнее заявление. Ты знаешь только свой ID клиента — 123456 — и больше ничего
Пользователь не знает часть необходимой информации, но имеет альтернативу	Ты хочешь узнать, было ли одобрено твое последнее заявление. Ты знаешь, что твой ID клиента — 123456, а ID заявления — 987654321987654

Инструкции из табл. 10.2 содержат информацию, которую LLM сможет использовать в диалоге:

- *Сценарий* — чего LLM должна добиться, например, узнать, была ли одобрена страховая выплата по заявлению.
- *Тестовые данные* — мы знаем, что чат-бот может вызывать API, поэтому нам нужно предоставить LLM данные, имеющиеся в нашей системе. Для этого мы указываем информацию, которую хотим использовать.
- *Границы* — мы оговариваем, какую информацию LLM не знает. Это должно предотвратить «выдумывание» информации (галлюцинации), из-за которой вызовы API будут завершаться ошибкой.

Мы не даем LLM больше никаких указаний, потому что хотим проверить, как она будет пытаться добиться нужного результата в чат-боте.

Наконец, нам нужно передать LLM транскрипт предыдущего диалога, чтобы она понимала контекст (и что она уже отвечала). Тестовый скрипт способен отслеживать историю диалога, поскольку именно он вызывает и чат-бота, и LLM. (Существует множество способов получить транскрипты диалогов, некоторые из них будут рассмотрены в главе 12.)

Теперь мы можем написать функцию на Python, которая будет генерировать промпт для выбранного сценария. Она принимает два аргумента: инструкции для сценария (см. табл. 10.2) и транскрипт диалога. В листинге ниже приведена эта функция.

Листинг 10.12. Функция Python для генерации промпта тестового сценария

```
def get_prompt(guidance, transcript):
    prompt=f'''
ИНСТРУКЦИЯ:
Ты – пользователь, желающий узнать статус своего заявления.
{инструкция}
Продолжи диалог возможным ответом.

ДИАЛОГ:
{транскрипт}
Пользователь: '''
    return prompt
```

Общее описание задачи

Инструкции на основе выбранного сценария и тестовые данные

Вставляется транскрипт диалога

Сигнал для ответа LLM

Функция генерирует промпт для заданного сценария и транскрипта диалога.

Листинг 10.13 содержит пример вызова функции `get_prompt()`. Он предполагает существование функции `call_llm()`, реализация которой зависит от используемой LLM-платформы (предположим, что была проведена инициализация с API-ключом, после чего у вас была возможность выбрать модель и конфигурацию, а затем была предоставлена функция передачи промпта и получения вывода).

Обязательно используйте декодирование с семплированием в `call_llm()`, чтобы получать разнообразные ответы.

Листинг 10.13. Python-код с динамическим промптом

```
guidance= Ты хочешь узнать, было ли одобрено одно
из твоих заявлений.
Ты знаешь, что твой ID клиента — 123456, дата подачи
заявления — 4 мая 2024 г., а сумма возмещения — $1000.'''
transcript='''Чат-бот: Чем я могу вам помочь?
Пользователь: Я хочу узнать статус своего заявления.
Чат-бот: Какой у вас ID клиента?'''

prompt=get_prompt(guidance, transcript)
user_response=call_llm(prompt)
transcript += f"\nUser: {user_response}"
```

Полный текст тестового сценария для LLM

Полный транскрипт диалога на текущий момент

Динамическая генерация промпта

Получение ответа LLM (например: «Мой ID клиента — 123456»)

Обновление транскрипта диалога

Теперь у нас есть первая часть нашего тестового скрипта. Настроим вторую.

10.3.2. Настройка тестирования диалога

Далее тестовый скрипт должен вызывать чат-бота. Скрипт будет принимать сгенерированный LLM «пользовательский» ввод и передавать его боту, затем принимать его ответ, записывать его в транскрипт и снова вызывать LLM. Здесь нам опять понадобится функция (ее реализация будет зависеть от используемой платформы), которая в нашем случае называется `call_chatbot()`. Ожидается, что эта функция будет настраивать подключение к чат-боту, аутентифицироваться с помощью API-ключа и управлять пользовательским диалогом.

Эта часть тестового скрипта показана в листинге ниже.

Листинг 10.14. Python-код для вызова чат-бота

```
# user_response был получен после вызова LLM
bot_response = call_chatbot(user_response)
transcript += f"\nSystem: {bot_response}"
```

Отправляет сообщение чат-боту

Сохраняет ответ чат-бота в транскрипте

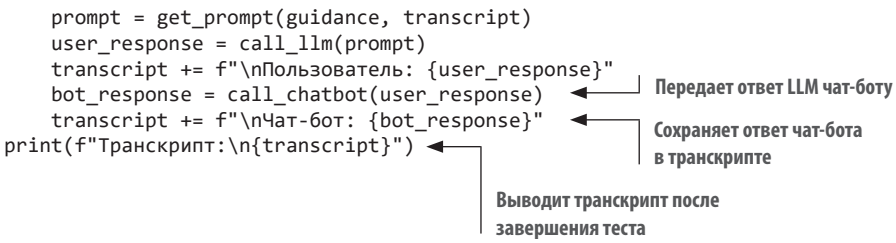
Теперь можно объединить все части. В следующем листинге все элементы соединяются в единый тестовый скрипт.

Листинг 10.15. Python-код, объединяющий вызовы LLM («пользователя») и чат-бота

```
def run_test(guidance):
    print(f"Запуск теста с:\n{guidance}\n")
    bot_response = call_chatbot('')
    transcript = f"Чат-бот: {bot_response}"
    for i in range(4):
```

Диалог часто начинается с «пустого» ввода

Тестирует несколько шагов диалога



Скрипт подключается к чат-боту и выполняет определенное число шагов (четыре). В зависимости от наших потребностей мы можем увеличить или уменьшить это число или добавить дополнительную логику для определения момента, когда диалог завершается (или прерывается ошибкой).

В листинге ниже показан пример вывода после выполнения скрипта в одном из наших тестовых сценариев.

Листинг 10.16. Пример вывода тестового скрипта

```
Запуск теста с:
Ты хочешь узнать, было ли одобрено одно из твоих заявлений.
Ты знаешь, что твой ID клиента — 123456, дата подачи заявления — 4 мая 2024 г.,
а сумма возмещения — $1000.

Транскрипт:
Чат-бот: Чем я могу вам помочь?
Пользователь: Здравствуйте, я хочу узнать статус своего заявления.
Чат-бот: Я вам помогу. Какой у вас ID клиента?
Пользователь: Мой ID клиента — 123456.
Чат-бот: Какова дата подачи заявления?
Пользователь: Дата подачи — 4 мая 2024 года.
Чат-бот: Какова сумма возмещения?
Пользователь: Сумма возмещения $1000.
Чат-бот: Спасибо. Теперь у меня есть вся необходимая информация. Проверяю статус
вашего заявления.
Статус вашего заявления — одобрено, сумма $1000 выплачена.
```

Данный скрипт задает базовые механизмы, позволяющие LLM выступать в роли пользователя вашего чат-бота.

Такой тип тестирования может стать отличным дополнением к другим тестам. LLM может сгенерировать такие пользовательские вводы, которые вы никогда бы не подумали проверить в чат-боте, поэтому полезно посмотреть, как ваша система на них среагирует. Помните, что LLM только имитирует человека — реальные пользователи могут никогда себя так не вести и никогда не отвечать как LLM. Но иногда могут.

Задания

1. Сыграйте роль бота. Реализуйте функцию `call_chatbot(user_response)` следующим кодом:

   ```
   return input('Введите ответ бота: ')
   ```

   Это позволит вам проверить, как LLM (в роли пользователя) реагирует на сообщения, которые вы (в роли чат-бота) ей отправляете. Это избавит вас от необходимости реализовывать чат-бота только для того, чтобы увидеть, как работает тестовый скрипт.
2. Добавьте функцию `call_chatbot(user_response)` в вашего реального чат-бота, а функцию `call_llm(prompt)` подключите к выбранной вами ИИ-платформе. Обновите функцию `get_prompt` так, чтобы она соответствовала вашему сценарию. Расширяет ли LLM возможности вашего чат-бота?

Итоги

- LLM могут с нуля разрабатывать для вас процессные потоки. С помощью простого промпта они могут сгенерировать примерные диалоговые сценарии, а также обосновать свои решения. Полученный процесс можно затем реализовать в традиционном разговорном ИИ.
- LLM могут также улучшить уже существующий поток, например упростить его.
- С помощью генеративного ИИ можно выполнить весь диалог целиком. Однако чтобы избежать ошибок, необходимо найти компромисс между скоростью реализации и степенью контроля.
- Некоторые процессы заполнения слотов можно заменить процессами, управляемыми LLM. Такое решение будет гораздо более гибким, чем сопоставление параметров API.
- Оценивайте стоимость ошибки при предоставлении LLM возможности самостоятельно принимать решения. В некоторых случаях ошибки некритичны, но будьте осторожны с API, на которые LLM может повлиять.
- LLM может имитировать пользователей вашего разговорного ИИ. Используйте ее для генерации тестовых диалогов, чтобы проверить, как система будет вести себя в различных сценариях.

Часть 4

Паттерн: разрешение конфликтов

Взаимодействие с чат-ботом значительно отличается от взаимодействия с человеком. Прежде чем перейти к делу, люди обычно перекидываются парой фраз, тогда как чат-боты сразу приступают к задаче. Люди хорошо умеют сглаживать углы, а чат-боты часто ведут себя как... роботы.

Еще до начала диалога пользователи уже могут быть негативно расположены к разговорным системам. Попадая к голосовым помощникам, они сразу нажимают клавишу 0 (чтобы переключиться на оператора), а при использовании чат-бота незамедлительно вводят слово «оператор». Или же делают это после первой ошибки ИИ. В любом случае мы получаем *отказ* — пользователь отказывается от взаимодействия с ИИ и отдает предпочтение общению с человеком.

В главе 11 мы рассмотрим причины отказов пользователей от ИИ на разных этапах взаимодействия — от начала и до самого его завершения — и поделимся методами, позволяющими снизить вероятность отказа. И тем не менее полностью исключить такой исход невозможно, а поэтому в главе 12 мы расскажем, что при этом делать, а именно как резюмировать взаимодействие с ИИ таким образом, чтобы оператору-человеку было легче продолжить диалог с места отказа.

11 Сокращение числа отказов

В этой главе

- ✓ Причины, по которым пользователи предпочитают общаться с человеком
- ✓ Как предотвратить немедленный отказ пользователя от ИИ
- ✓ Как удержать пользователя в диалоге с разговорным ИИ
- ✓ Использование генеративного ИИ для более дружелюбного общения
- ✓ Когда стоит переключать пользователя на оператора-человека (а когда нет)

Термин «отказ» означает попытку пользователя выйти из взаимодействия с виртуальным агентом, часто с целью связаться с оператором-человеком. Это также называется *эскалацией* или *обнулением* (нажатие нуля на телефонной клавиатуре для соединения с оператором). Отказы обходятся дорого: чат-боты — это своеобразная инвестиция, и их ценность необходимо оправдывать реальной пользой для бизнеса. Потеря удержания пользователей из-за чрезмерного количества отказов может «потопить» бизнес-кейс.

Пользователи могут отказываться по множеству причин, требующих разных стратегий и подходов. Независимо от типа бота — голосовой, текстовый, FAQ-бот,

процессно-ориентированный или маршрутизирующий, — выявление того, на каком этапе пользователь выходит из диалога, поможет определить причины отказа и спроектировать взаимодействие с более высоким удержанием.

В этой главе мы рассмотрим примеры разговорных ИИ, страдающих от частых отказов, и проанализируем решение каждой возможной проблемы. Различные сценарии применения могут иметь разные решения в зависимости от приоритетов, ресурсов и ограничений компании, но существуют также общие паттерны и принципы, позволяющие повысить ценность разговорных систем и лучше удерживать пользователей.

11.1. Что вызывает отказы?

Некоторые пользователи, сталкиваясь с виртуальным агентом, сразу запрашивают человека — они даже не пытаются взаимодействовать с разговорным ИИ. Мы называем это *мгновенным отказом* (immediate opt out). В других случаях пользователь сначала соглашается на взаимодействие с виртуальным агентом, но пытается отказаться позже. Причины таких отказов обычно значительно отличаются от причин мгновенных отказов и часто указывают на проблемы в общей структуре диалога или в отдельном шаге потока. Слабое понимание запросов пользователя также может быть причиной.

11.1.1. Причины мгновенных отказов

Сами по себе мгновенные отказы дают очень мало информации об их причинах. Сбор данных значительно затруднен и может проводиться только через опросы или последующий анализ эскалаций (весьма трудоемкий процесс). Мы выявили несколько наиболее распространенных причин такого поведения, причем одна из них не исключает другие.

Негативный опыт взаимодействия с IVR, чат-ботом или виртуальным агентом

Интерактивные голосовые меню (IVR, interactive voice responce) позволяют пользователю взаимодействовать с компьютером, используя клавиатуру телефона или простые речевые команды. Первые IVR появились еще в 1970-х годах, но в 2000-х они стали более доступными и с тех пор широко применяются по всему миру. Вряд ли найдется человек, который ни разу не сталкивался с раздражающим началом звонка, когда автомат в течение тридцати секунд объясняет, как пользоваться меню. Еще хуже — предупреждение «Пожалуйста, прослушайте все пункты меню» (потому что они могли поменяться)!

Чат-боты тоже существуют достаточно давно, но способными выполнять функционально полезные задачи они стали сравнительно недавно, и их развитие в сторону полноценных виртуальных помощников все еще продолжается.

Пользователи могут не видеть различий между IVR, простым чат-ботом и продвинутым виртуальным помощником. По факту им это и не важно. Негативный

опыт прошлых взаимодействий формирует недоверие ко всем автоматизированным системам. В интернете даже существуют форумы и статьи, где пользователи делятся «лайфхаками», как быстрее обойти автоматизированную систему и попасть к оператору.

Пользователь считает свою проблему слишком сложной для машины

Иногда пользователь считает свою ситуацию слишком уникальной или сложной, полагая, что она выходит за пределы возможностей автоматизированной системы. Иногда они правы, а иногда — нет. Понятия сложности и уникальности относительны для каждого человека, и пользователи могут не знать, что тысячи других людей уже сталкивались с аналогичной проблемой. Сомнения также могут быть вызваны предыдущим опытом, но не всегда.

Такие пользователи отказываются от взаимодействия с ботом, потому что уверены, что им все равно потребуется помощь оператора. Они воспринимают автоматизированную систему как пустую трату времени, которая лишь удлинит или усложнит путь к решению проблемы.

Пользователь предпочитает общаться с человеком

Общение друг с другом имеет фундаментальное значение для нашего выживания как вида, поэтому некоторые пользователи предпочитают иметь дело только с реальными людьми. Эта потребность может не иметь никакого отношения к функциональным возможностям системы и объясняться просто чувством одиночества, деликатной или неловкой темой обращения, недоверием к машинам и автоматизированным системам в принципе, языковыми или когнитивными барьерами и т. д. Причем такая причина отказа все чаще оказывается связана с возрастом: пользователи более старшего возраста чаще запрашивают оператора, поскольку им сложнее взаимодействовать с машинами, тогда как представители цифрового поколения гораздо лучше ориентируются в автоматизированных системах. Но всегда найдутся пользователи, которые, независимо от их возраста, предпочитают общаться с человеком.

11.1.2. Причины более поздних отказов

Пользователи, которые сначала соглашаются взаимодействовать с виртуальным помощником, но потом пытаются отказаться, обычно делают это из-за трудностей на каком-либо этапе. Подобные случаи часто можно связать с конкретной задачей, действием или шагом в диалоговом сценарии, и это упрощает поиск первопричины.

Бот не понимает запрос пользователя

Если бот не понимает запрос пользователя, тот может отказаться от взаимодействия. Чаще всего это происходит на ранних этапах диалога, но может случиться и в более поздний момент.

Системы обычно допускают определенное количество повторных попыток ввода до эскалации к оператору. Как правило, это три, но в зависимости от задачи порог

может быть больше или меньше. Это важный инструмент удержания пользователей в автоматизированном канале, но наша терпимость к ошибкам машин часто ниже, чем к ошибкам людей. Пользователь может прекратить взаимодействие уже после одной-двух повторных попыток переформулировать запрос, решив, что бот так его и не поймет.

Пользователь не понимает, чего от него хочет бот

Некорректно сформулированный вопрос бота может сбить пользователя с толку. Он не понимает, какой тип или формат ответа от него требуется, и поэтому просит бота повторить вопрос. Но если формулировка останется прежней, путаница не исчезнет. В результате пользователь посчитает, что застрял, и скорее всего, попросит помощи оператора.

У пользователя нет запрашиваемой информации

Когда пользователь должен предоставить информацию, которой у него нет, он ответит: «Я не знаю» или «У меня такого нет» либо сразу попросит перевести на оператора. Если сценарий не может быть продолжен без определенных данных и пользователю не предлагается никаких альтернативных вариантов, он приходит к выводу, что без участия человека его цель недостижима.

Пользователь не видит прогресса

Пользователь может отказаться от взаимодействия с системой, если он посчитает, что диалог ходит по кругу или зашел в тупик. Причиной этому может быть ошибка в логике потока или недостаточная ясность диалога, который никак не отражает продвижение пользователя к его цели, в результате чего тот неизбежно потеряет терпение и захочет выйти.

Пользователь недоволен ответом системы или ожидал другого результата

Бот может предоставить технически верный ответ, но если он не удовлетворяет пользователя, тот может запросить оператора в надежде получить другой результат. Пользователь также может посчитать предоставленную информацию недостаточной и отказаться от виртуального помощника, если не имеет возможности задать дополнительные вопросы или уточнить детали.

11.1.3. Сбор данных об отказе

Чтобы определить, снижается ли метрика удержания вашей системы от отказов пользователей, требуется собрать необходимые для этого данные. Некоторые платформы разговорных ИИ формируют отчеты о действиях или задачах, активированных в ходе диалога. Они также могут предоставлять данные о том, насколько успешно завершилось взаимодействие.

Но иногда встроенной аналитики недостаточно для получения метрик, необходимых для оптимизации. В таком случае можно ввести в ключевые точки сценария

дополнительные контекстные переменные. Хорошая инструментализация потоков позволяет выявлять проблемы и приоритизировать улучшения. (Для этого вам может потребоваться хранилище данных и корпоративный или кастомный инструмент отчетности для отслеживания данных во времени.)

Для сложных решений, ориентированных на выполнение задач, можно использовать так называемые хлебные крошки (breadcrumbs) для обозначения начала или завершения основных потоков и подпотоков. На рис. 11.1 показан пример данных об отказах, сгруппированных по основному потоку, в рамках которого произошел запрос на эскалацию.

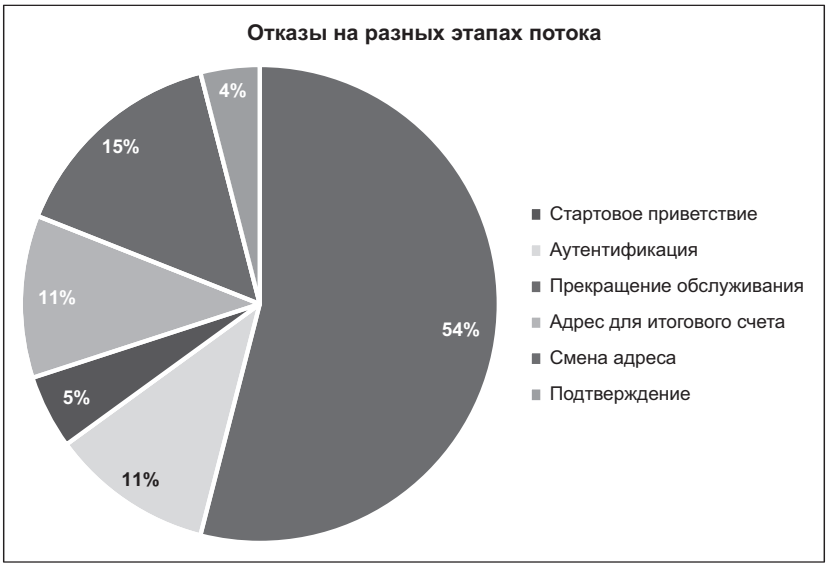


Рис. 11.1. Распределение отказов по этапам потока показывает, что для этого автоматизированного процессно-ориентированного бота мгновенные отказы (запрос оператора сразу после стартового приветствия) происходили чаще, чем отказы на других этапах сценария

Если ваш диалог спроектирован таким образом, что фиксирует точный шаг отказа, вы можете искать закономерности и выявлять первопричины, как показано на рис. 11.2.

Первым шагом анализа, особенно для процессно-ориентированных ботов, будет обнаружение потоков, где происходят мгновенные отказы. В простых Q&A-ботах у вас будет только начальное приветствие, а все после него будет относиться к категории «прочие отказы».

В нашем процессно-ориентированном боте запрос оператора во время стартового приветствия расценивается как мгновенный отказ — пользователь не готов к взаимодействию. Все остальные отказы связаны с другими шагами потока.

Рисунок 11.3 демонстрирует распределение отказов относительно положения задач в общей последовательности диалога.

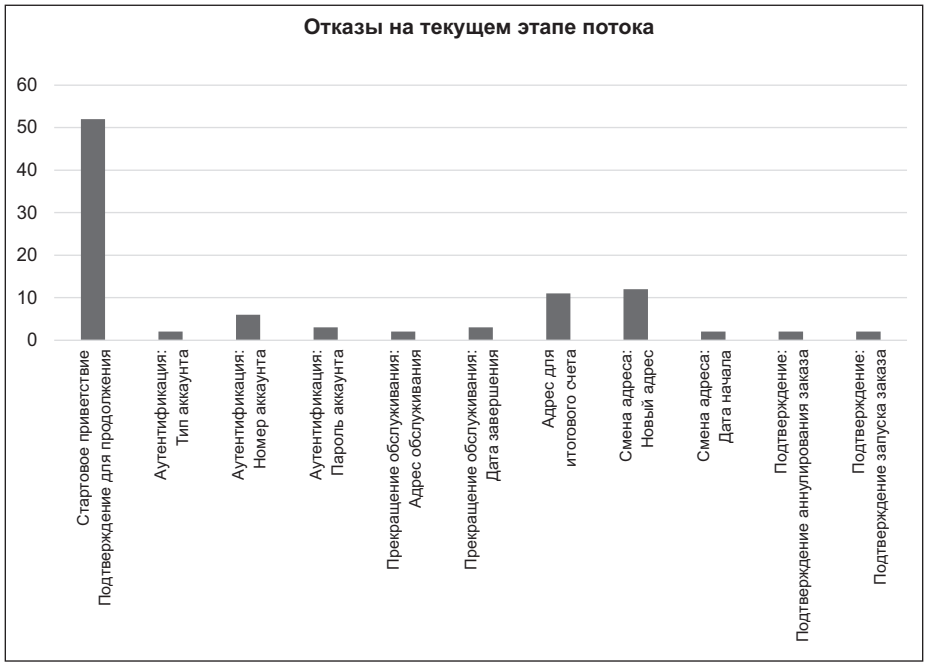


Рис. 11.2. Распределение отказов по конкретным шагам помогает найти первопричину. Диаграмма показывает большое количество мгновенных отказов (слева), а также обозначает проблему со сбором адресной информации в последующих шагах

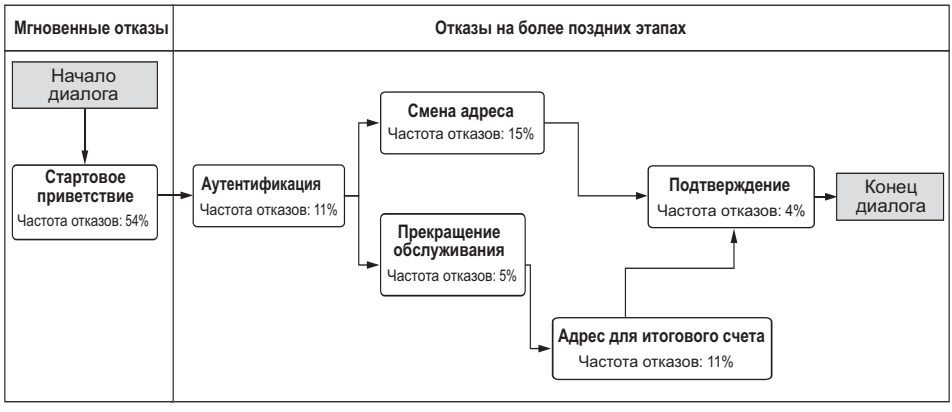


Рис. 11.3. Схема показывает, как далеко пользователь продвинулся перед отказом. Такие данные помогают выявить первопричину

Оставшаяся часть главы посвящена способам решения проблемы отказов — подходам, направленным на снижение ранних отказов, отказов на более поздних этапах (later opt-out), а также на удержание пользователя в канале.

Задания

Проанализируйте информацию о том, почему пользователи отказываются от виртуального помощника:

1. Понимаете ли вы различия между мгновенным отказом и отказом на более поздних этапах?
2. Замечаете ли вы повторяющиеся паттерны отказов в ваших собственных диалоговых потоках?

11.2. Сокращение числа мгновенных отказов

Вы наверняка слышали выражение: «У вас не будет второго шанса, чтобы произвести первое впечатление». Мгновенные отказы — это признак того, что пользователь не впечатлен вашей системой. У вас есть лишь пара мгновений на то, чтобы убедить его, что он попал туда, куда хотел, и что ваш виртуальный помощник компетентен и эффективен. В этом разделе мы рассмотрим стратегии, с помощью которых можно снизить вероятность того, что пользователь с первых секунд запросит оператора.

Граница между «мгновенным отказом» и просто «отказом» в рамках диалога не является произвольной, но при этом она довольно гибкая. Пользователь может покинуть бот как на самом первом шаге, так и в течение первых нескольких шагов. Такое разграничение необходимо для определения точки в сценарии, где клиент может либо продолжить взаимодействие, либо отказаться от него. Так, например, для FAQ-бота запрос оператора в самом первом сообщении будет считаться «мгновенным отказом», а все последующее — «другими отказами».

Ниже представлены три стратегии, позволяющие наиболее эффективно сократить число мгновенных отказов.

11.2.1. Начните с хорошего: приветствия и представления

Что позволит пользователю получить благоприятное впечатление от его взаимодействия с ботом? Здесь применимы базовые принципы клиентского сервиса — пользователь должен чувствовать, что он попал куда нужно, что он в надежных руках и что его время ценят.

Первой причиной мгновенного отказа, рассмотренной в этой главе, был негативный опыт использования IVR-системы, чат-бота или виртуального помощника. Приветствие и дальнейшее представление вашего бота задаст тон всему диалогу. Это ваш шанс завоевать доверие пользователя — убедить его, что виртуальный

агент способен столь же эффективно и действенно выполнить необходимую задачу, как и оператор-человек.

В главе 1 мы уже упоминали, как важно для системы решить проблему мгновенных отказов пользователей. Мы рассмотрели пример виртуального помощника для энергетической компании, который терял более половины звонков из-за мгновенных отказов. Помощник был расширением крупного IVR (голосовое меню основной линии поддержки клиентов). Пилотная программа была предназначена для обработки двух задач самообслуживания: прекращение обслуживания и смена адреса обслуживания.

Считалось, что все, кто попадал к виртуальному помощнику, хотели выполнить одну из этих двух операций (прекратить обслуживание или сменить адрес) посредством выбора нужной позиции IVR, после чего звонящий направлялся к боту. Информация о пользователе и его выборе передавалась виртуальному помощнику, который сразу запускал соответствующий сценарий. В листинге ниже показано, как выглядело взаимодействие — сначала с IVR, а затем с ботом.

Листинг 11.1. Передача потока от IVR виртуальному помощнику

```
IVR: Спасибо, что позвонили в ABC Energy. Если вы хотите сообщить об аварийной
ситуации, выберите один из следующих вариантов: скажите "газовая авария" или
нажмите 1. "Электрическая авария" или нажмите 2. "Отключение электроэнергии" —
или нажмите 3. Если вы позвонили не по поводу аварийной ситуации, скажите
"другое" или нажмите 4.

ПОЛЬЗОВАТЕЛЬ: <выбирает опцию 4>

IVR: Главное меню. Для информации по счетам и аккаунту нажмите 1. Платежи —
нажмите 2. Подключение или прекращение обслуживания, а также смена адреса
обслуживания — нажмите 3.

ПОЛЬЗОВАТЕЛЬ: <выбирает опцию 3>

IVR: По какому поводу вы звоните? Скажите "строительство нового дома" или
нажмите 1. "Подключить, прекратить или перенести обслуживание" — или нажмите 2.
"Переподключение" — или нажмите 3. "Уличное освещение" — или нажмите 4. "Ничего
из перечисленного" — или нажмите 5.

ПОЛЬЗОВАТЕЛЬ: <выбирает опцию 2>

IVR: Чем я могу вам помочь? Скажите "проверить статус заявки" или нажмите 1.
"Подключить обслуживание" — или нажмите 2. "Прекратить обслуживание" — или
нажмите 3. "Сменить адрес" — нажмите 4.

ПОЛЬЗОВАТЕЛЬ: <выбирает опцию 3>

<Клиент переводится на виртуального агента, голос системы меняется.>

ВИРТУАЛЬНЫЙ ПОМОЩНИК: Я помогу вам прекратить обслуживание. Для начала мне
нужна информация о вашем текущем адресе. Выберите тип аккаунта: частный или
коммерческий?
ПОЛЬЗОВАТЕЛЬ: Соедините с оператором.
```

В первых продакшен-логах мы заметили, что значительное количество пользователей отказываются от взаимодействия почти сразу же. И хотя виртуальный помощник технически весьма функционален, многие звонящие даже не давали ему шанса. Мы прослушали записи звонков и обнаружили, что пользовательский опыт в целом оказался довольно разрозненным. Клиенты звонили по номеру, попадали в IVR (с определенным «голосом» IVR) и проходили меню тем, связанных с обслуживанием. Если звонящий выбирал прекращение обслуживания или смену адреса, он перенаправлялся к нашему виртуальному помощнику, но переход был резким — другой голос пропускал приветствие (ведь с пользователем уже поздоровались в IVR) и сразу переходил к задаче, что очень эффективно с точки зрения процесса.

Перевод на виртуального помощника вызывал у пользователей замешательство, проявлявшееся в долгих паузах, вздохах или запинках («э-э...», «мм...», «хм...»). Им никто не представился, а первый вопрос, хоть и был процессуально корректен, звучал безлично и неуклюже. Оператор-человек никогда бы не начал разговор подобным образом, сначала он бы дружелюбно представился. Если IVR уже собрал информацию о цели звонка, оператор подтвердил бы ее перед продолжением («Я вижу, вы звоните по поводу смены адреса обслуживания, все верно?»).

Мы переработали сценарий, начиная с первой реплики, и добавили контекстно-зависимое приветствие и представление. Виртуальному помощнику была присвоена персона и установлен более разговорный стиль общения, как показано в следующем листинге.

Листинг 11.2. Обновленное приветствие и представление виртуального помощника

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Добрый день. ◄—— Контекстно-зависимое приветствие
Меня зовут Алиса. Я виртуальный агент ABC Energy. ◄—— Представление виртуального агента
```

Диалог начинается с приветствия и представления, и это позволит отличить нашего виртуального помощника от IVR-системы. Звонящий получает возможность адаптироваться к переходу на новый голос и иную модель взаимодействия.

Контекстно-зависимое приветствие, например упоминание времени суток или обращение к пользователю по имени, преобразует роботизированный, безличный обмен репликами в более теплый и дружелюбный диалог. В некоторых сценариях имя или персона бота могут быть неуместны, но в нашем случае они создали ощущение присутствия и вовлеченности. «Алиса» всегда готова помочь клиенту.

11.2.2. Обозначьте возможности и установите ожидания

Среди других причин мгновенного отказа мы также выделили пользователей, которые считают, что их проблема уникальна или слишком сложна для машины. Иногда такое предположение справедливо, а иногда — нет.

Крайне важно формировать ожидания от системы с самого начала взаимодействия. Нередко компании запускают пилотного виртуального помощника с ограниченным функционалом и охватом задач. Если ваше решение поддерживает лишь часть возможных запросов, необходимо об этом сообщить. Так пользователи либо будут уверены, что им помогут, либо сразу поймут, что нет. Самый подходящий момент рассказать о назначении и возможностях чат-бота (или быстро подтвердить цель пользователя перед продолжением), особенно в подобных сценариях, — это приветствие (либо точка входа в систему).

Поэтому изначальное приветствие виртуального агента энергетической компании было расширено таким образом, чтобы подтвердить цель клиента (на основании данных из IVR). Мы кратко описали путь, который предстоит пройти пользователю. Поскольку это был голосовой канал, более подверженный внезапным разрывам соединения, мы также сформировали ожидания относительно того, как будет выглядеть (и звучать) успешное выполнение задачи. В листинге ниже приведен дополненный текст приветствия.

Листинг 11.3. Обновленное приветствие, сообщающее о возможностях системы и формирующее ожидания

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Добрый день. Меня зовут Алиса.
Я виртуальный агент ABC Energy.
Я здесь, чтобы помочь вам с отключением услуги
энергоснабжения.
Мне потребуется собрать данные о вашем аккаунте,
чтобы составить заявку на отключение.
Пожалуйста, оставайтесь на линии, пока я не продиктую
номер подтверждения.
Если соединение прервется до подтверждения, изменения
по вашему аккаунту внесены не будут.
```

Подтверждает назначение или возможности бота

Предварительно описывает путь пользователя

Формирует ожидания об успешном завершении взаимодействия

11.2.3. Подкрепляйте мотивацию

Технически подкованный пользователь, постоянно взаимодействующий с ботами, осознает эффективность автоматизированных решений, но пользователи, обращающиеся к ним редко или даже впервые, не знают, чего им ожидать, и процесс может показаться им слишком сложным или затянутым. Такие пользователи предпочитают оператора-человека, что приводит к мгновенным отказам. Однако если мотивировать их остаться в боте, то количество подобных отказов может значительно снизиться.

В нашем примере средняя длительность звонка при оформлении заявки на прекращение обслуживания через оператора составляла около 5–7 минут (плюс время ожидания). Автоматизированный процесс выполняет эту задачу ничуть не медленнее. В следующем листинге показано, какую правку мы внесли в приветствие, чтобы сообщить пользователю, что на достижение его цели не требуется много времени.

Листинг 11.4. Обещание быстрого решения удерживает звонящего

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Добрый день. Меня зовут Алиса. Я виртуальный агент ABC
Energy.
Я здесь, чтобы помочь вам с отключением услуги энергоснабжения.
Мне потребуется собрать данные о вашем аккаунте, чтобы составить заявку на
отключение.
Процесс займет всего несколько минут.  <-- Удержание пользователя
Пожалуйста, оставайтесь на линии, пока я не продиктую номер подтверждения. Если
соединение прервется до подтверждения, изменения по вашему аккаунту внесены не
будут.
```

Чтобы еще сильнее мотивировать пользователя, можно заверить его, что его цель будет достигнута, даже если возникнут проблемы с системой. Листинг ниже содержит обновленное приветствие.

Листинг 11.5. Предотвращение немедленных отказов путем гарантии эскалации проблемы

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Добрый день. Меня зовут Алиса.
Я виртуальный агент ABC Energy.
Я здесь, чтобы помочь вам с отключением услуги энергоснабжения.
Мне потребуется собрать данные о вашем аккаунте, чтобы составить заявку на
отключение. Процесс займет всего несколько минут. Пожалуйста, оставайтесь на
линии, пока я не продиктую номер подтверждения. Если соединение прервется до
подтверждения, изменения по вашему аккаунту внесены не будут.
Если возникнут какие-либо сложности,
я переведу вас на оператора поддержки.
```

Обещает пользователю, что при возникновении проблем его соединят с оператором

11.2.4. Задействуйте пользователя

По возможности стоит давать пользователю некоторое ощущение контроля. Например, это может выражаться в получении его согласия на продолжение взаимодействия с виртуальным помощником. Такой подход может быть неуместным в некоторых сценариях, однако он позволяет четко зафиксировать, что пользователь согласился продолжить диалог.

Мы еще раз обновили приветствие нашего виртуального помощника. Теперь мы спрашиваем пользователя, готов ли он продолжить.

Листинг 11.6. Приветствие задействует пользователя

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Добрый день. Меня зовут Алиса. Я виртуальный агент ABC
Energy.
Я здесь, чтобы помочь вам с отключением услуги энергоснабжения.
Мне потребуется собрать данные о вашем аккаунте, чтобы составить заявку на
отключение. Процесс займет всего несколько минут. Пожалуйста, оставайтесь на
линии, пока я не продиктую номер подтверждения. Если соединение прервется до
подтверждения, изменения по вашему аккаунту внесены не будут.
Если возникнут какие-либо сложности, я переведу вас на оператора поддержки.
Готовы продолжить?  <-- Требуется участие пользователя
```

И хотя мы рассчитываем услышать «да», нам также стоит подготовиться к обработке и других ответов. Предположим, что некоторые пользователи соединились с системой по ошибке. Иногда такое может случаться из-за недопонимания того, что произойдет после выбора той или иной опции в IVR. В других случаях это мог быть просто промах по клавише — ошибочное нажатие кнопки на телефоне. Эти «другие» ответы подтвердили нашу гипотезу — пользователи говорили «назад», «нет» или сообщали совершенно другую задачу. Они попадали в систему случайно, и наша задача заключалась в том, чтобы вернуть их на правильный путь для достижения их цели. На рис. 11.4 показано, как мы переработали логику приветствия, чтобы каждый вариант ответа, не являющийся «да», был обработан корректным образом.

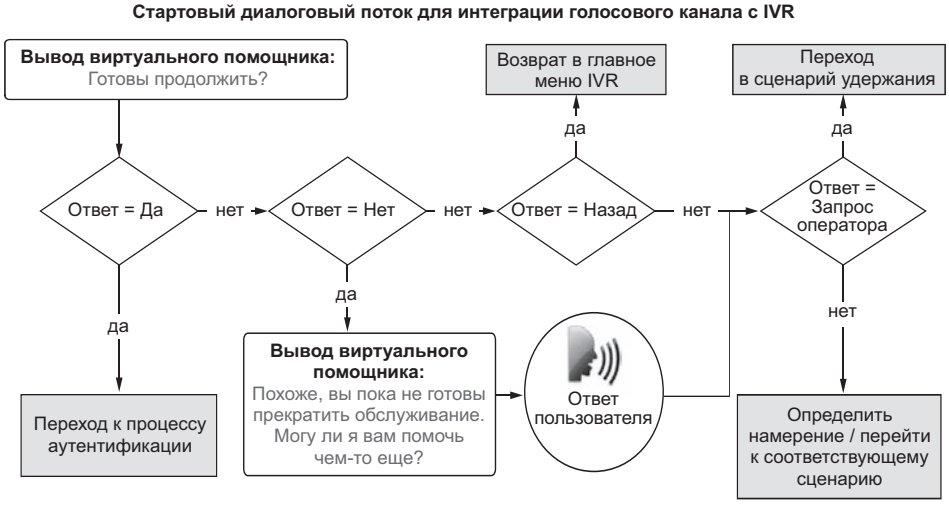


Рис. 11.4. Такая логика диалога позволяет обрабатывать разные ответы на вопрос «Готовы продолжить?» Если дается прямой ответ, например «нет», «назад» или «соедините с оператором», вызывается заранее определенный сценарий. В противном случае ответ направляется в классификатор для определения намерения и последующего выбора соответствующего сценария

Задания

Проанализируйте фрагмент вашего диалога, где пользователь мгновенно отказывается от взаимодействия, и ответьте себе на следующие вопросы:

1. Приветствует ли виртуальный помощник пользователя, представляется ли ему?
2. Разъясняет ли виртуальный помощник свою цель — что он может и чего не может делать?
3. Предоставляет ли виртуальный помощник возможность пользователю легко достичь поставленной цели (не сложнее, чем альтернативные каналы или оператор-человек)?

11.3. Сокращение числа отказов, возникающих позже

Отказы, происходящие на более поздних этапах диалога, зачастую можно связать с конкретной проблемой или недочетом в его архитектуре. В этом разделе мы рассмотрим стратегии и подходы, которые помогут минимизировать такие случаи.

11.3.1. Старайтесь понять пользователя

В главе 4 мы уже обсуждали, как важно понять, чего хочет пользователь. Отказы могут указывать на проблемы с распознаванием намерений. При взаимодействии с чат-ботом, приветствие которого представляет собой вопрос вроде «Чем я могу помочь?», пользователь ожидает, что может задать практически любой вопрос, связанный с вашей компанией или даже со сферой деятельности в целом.

Если система не способна обрабатывать довольно широкий спектр запросов, она будет просить пользователя повторить или переформулировать вопрос, что только вызовет его раздражение. Не допускайте устаревания вашего решения, регулярно обновляйте обучающие данные, чтобы они отражали самые актуальные потребности пользователей. Некоторым сценариям пойдет на пользу внедрение поиска или RAG, особенно в широких предметных областях и вопросно-ответных ботах.

11.3.2. Старайтесь быть понятыми

Выбор используемых слов и формулировок также имеет значение, особенно при запросе какой-либо информации у пользователя. Слишком длинное сообщение или неудачно сформулированный вопрос могут оказаться трудными для восприятия. Пользователь может не понять, какие именно данные от него требуются, или не иметь их вовсе и поэтому отказаться от дальнейшего взаимодействия. Если вы запрашиваете какую-либо информацию, четко формулируйте, как именно нужно ответить. Иногда вопросы с выбором могут восприниматься как вопросы «да/нет»:

- «Вы звоните по вопросу подключения, отключения или переноса услуги электроснабжения?»
- «Вы хотите подать запрос на повторное рассмотрение, оспаривание требования или его проверку?»

На рис. 11.5 показан пример вопроса с выбором, который может быть воспринят как вопрос типа «да/нет», и один из способов разрешения подобного недопонимания.



Рис. 11.5. Структура вопроса позволяет пользователю понять его тип и ожидаемый формат ответа

11.3.3. Будьте гибкими и готовыми подстроиться

Жесткие, запутанные или чересчур ограничивающие условия допустимости ответа могут привести к отказам. Диалог должен быть достаточно гибким, чтобы обрабатывать широкий диапазон корректных ответов и при необходимости уточнять информацию. Пример уточняющего вопроса показан на рис. 11.6.



Рис. 11.6. Устойчивый сценарий диалога способен обрабатывать различные допустимые ответы. Чтобы точнее определить цель пользователя, необходимая информация уточняется в отдельном шаге

Рекомендации для голосовых решений

Если в вашей системе используется распознавание речи, во избежание серьезных ошибок понимания убедитесь, что транскрипция корректна. Поскольку языковые модели работают с текстом, неверно распознанная речь может значительно осложнить понимание. Если ваша система некорректно фиксирует речь пользователя, важные моменты могут быть упущены.

Для голосовых решений крайне важно минимизировать когнитивную нагрузку на пользователя, поэтому вопросы необходимо формулировать максимально четко и лаконично. Если вам нужно включить в вывод какую-то контекстную информацию, вопрос должен идти в самом конце, тем самым сигнализируя пользователю, что настала его очередь говорить, — пример показан на рис. 11.7. Ваша система также должна быть способна при необходимости повторять вопрос или предоставленную ранее информацию.

Плохо	Хорошо
Спасибо, что позвонили в Acme Bank. **Чем я могу вам помочь?** Я могу проверить баланс вашего счета, выполнить перевод средств или запланировать платеж.	Спасибо, что позвонили в Acme Bank. Я могу проверить баланс вашего счета, выполнить перевод средств или запланировать платеж. **Чем я могу вам помочь?**

Рис. 11.7. Плохой пример (слева) помещает вопрос «Чем я могу вам помочь?» в середину реплики, что может побудить пользователя начать говорить еще до окончания сообщения. Хороший пример (справа) сначала содержит информационные высказывания, а вопрос к пользователю размещается в конце сообщения, тем самым давая ему понять, что можно начать говорить

Используйте доступные технологии

В некоторых ситуациях применение соответствующих технологических решений позволяет сгладить возможные трудности для пользователя и улучшить понимание его потребностей. Например, у одной страховой компании был голосовой помощник для поддержки клиентов, чьи ID могли состоять из 9 или 13 цифр, которым иногда предшествовала буква. Мы не хотели перегружать звонящего вопросом о том, какой длины его ID и есть ли в нем буква, — это создает слишком большую когнитивную нагрузку, особенно при первом опыте общения с системой. Мы просто спрашивали: «Каков ваш ID?» Некоторые пользователи произносили номер вслух, включая букву. Другие вводили номер через клавиатуру, игнорируя первую букву. Мы хотели учесть естественное поведение пользователей, поэтому сделали логику сценария более устойчивой. Звонящий мог и не предоставлять буквенный символ при условии, что нам удалось распознать нужное количество цифр для поиска. Вместо того чтобы докучать клиенту повторными уточнениями, система подставляла стандартные данные и пыталась выполнить поиск по уже имеющейся информации (принимались оба ответа — как «123456789», так и «X123456789»). При повторной попытке изначальное сообщение слегка изменялось, чтобы пользователь ввел необходимое нам значение.

Еще одним инструментом повышения устойчивости и точности системы, при наличии технологической возможности, являются кастомные модели распознавания речи. Транскрипция чисел или букв по голосовому каналу затруднена из-за сходства фонем (например, буква «H» английского алфавита часто звучит как цифра «8»), поэтому необходимо выбирать модели, оптимизированные под такие входы.

11.3.4. Показывайте прогресс

Автоматизированные потоки могут варьироваться по сложности ответа на один и тот же вопрос — от простой сортировки до многошаговых, многопотоковых и динамичных пользовательских сценариев. В более сложных потоках дизайн диалога должен сообщать пользователю о его продвижении по задаче или процессу. Если пользователю кажется, что его взаимодействие с ботом будет длиться вечно или что он застрял в повторяющемся меню, он просто приходит к выводу, что разговор с оператором-человеком будет более эффективным.

Прогресс можно показывать разными способами и оптимизировать их под конкретный канал. Перед запуском длительного сценария лучше заранее установить ожидания пользователя: какие данные вам понадобятся и сколько времени это займет. Текстовая платформа может выдавать более подробные сообщения или визуальные индикаторы (нумерация вопросов, прогресс-бар и т. п.). В голосовом канале нужно быть лаконичнее, но можно использовать фразы вроде «Мы почти закончили» или «Осталось всего несколько вопросов».

11.3.5. Предугадывайте потребности пользователя

Когда пользователь получает ответ от бота или завершает поток, он может обнаружить, что его цель все еще не достигнута. Есть ли у вашего диалога исходы, которые технически корректны, но оставляют проблему нерешенной? Подумайте о пользователе: у него есть потребность, он получает ответ, но потребность остается. Если поток завершается без удовлетворения этой потребности, пользователь наверняка обратится к оператору. А что ему еще остается делать?

Можно ли расширить возможности системы и добавить альтернативный путь решения проблемы? На рис. 11.8 приведены примеры сценариев и возможные способы предотвратить эскалацию.

Сценарий	Следующее действие
Пользователь проверяет статус страхового заявления и узнает, что оно было отклонено	Предложить помощь в подаче претензии
Пациент звонит с просьбой повторно выдать рецепт, но его срок истек	Предложить подать запрос на продление от имени пациента
Клиент банка запрашивает свой баланс, и счет оказывается в минусе	Предложить показать последние транзакции
Путешественник пропускает рейс, а возможные варианты переноса ему не подходят	Предложить добавить путешественника в лист ожидания на другие рейсы

Рис. 11.8. Вы можете избежать эскалации, предложив следующий шаг, если видите, что проблема пользователя осталась нерешенной

11.3.6. Будьте вежливы

При создании виртуального помощника помните, что вы пытаетесь симулировать общение с человеком. Эмпатия свойственна людям, но в машинное взаимодействие ее приходится встраивать намеренно. Пользователи уже прикладывают усилия в ходе диалога, поэтому если им требуется повтор вопроса или уточнение, не заставляйте их чувствовать себя виноватыми. Сообщения об ошибке должны быть сформулированы так, чтобы направлять пользователя обратно на путь решения без чувства вины.

Например, наш бот для страховой компании мог искать заявления по их номеру, состоящему из 13 цифр. Изначально нам показалось логичным выдавать «информативное» сообщение при повторной попытке ввода, но на практике это звучало довольно грубо, особенно по телефону, как показано в листинге ниже.

Листинг 11.7. Сценарий повторной попытки с грубым взаимодействием

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Чтобы найти ваше заявление, мне нужен его
тринадцатизначный номер.

ПОЛЬЗОВАТЕЛЬ: <произносит тринадцатизначный номер, но распознавание речи
зафиксировало только двенадцать цифр>

ВИРТУАЛЬНЫЙ ПОМОЩНИК: Вы не предоставили тринадцатизначный номер. Повторите или
введите номер.
```

Иногда пользователи изначально не называют необходимое количество цифр, в других случаях механизм распознавания речи не «слышит» все произнесенные цифры или неверно выполняет транскрипцию ответа. Независимо от того, по чьей вине это произошло, системе приходится повторно запрашивать информацию у пользователя. При первой повторной попытке достаточно просто еще раз озвучить вопрос. При последующих попытках необходимо плавно направлять пользователя, если ему сложно дать корректный ответ или предоставить информацию в требуемом формате.

Листинг 11.8. Обновленный сценарий повторной попытки

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Готов помочь вам с поиском. Какой номер заявления?

ПОЛЬЗОВАТЕЛЬ: <произносит тринадцатизначный номер, но распознавание речи
зафиксировало только двенадцать цифр>

ВИРТУАЛЬНЫЙ ПОМОЩНИК: Извините, я вас не расслышал. Какой номер у вашего
заявления?

ПОЛЬЗОВАТЕЛЬ: <произносит тринадцатизначный номер, но распознавание речи
зафиксировало только двенадцать цифр>

ВИРТУАЛЬНЫЙ ПОМОЩНИК: Извините, я снова вас не расслышал. Вы можете произнести
или ввести тринадцатизначный номер заявления.
```

Задания

Проанализируйте последующие потоки вашего диалога, в которых пользователи могут отказаться от взаимодействия, и ответьте на следующие вопросы:

1. Как часто пользователи прерывают диалог из-за того, что виртуальный помощник не понимает их запрос?
2. Бывают ли случаи, когда пользователи прерывают диалог из-за того, что они не понимают, что виртуальный помощник у них спрашивает?
3. Приятен ли виртуальный помощник в общении? Демонстрируют ли его ответы компетентность, эффективность и эмпатию?
4. Возможно ли с помощью автоматизации оптимизировать путь пользователя к его конечной цели?
5. Обращаются ли пользователи к оператору по причине того, что после взаимодействия с виртуальным помощником у них остаются нерешенные задачи, даже в случае, казалось бы, «успешного» завершения потока?

11.4. Удержание пользователя при отказе

Все стратегии, рассмотренные нами до этого момента, направлены на снижение числа мгновенных отказов пользователей, но не всегда способны полностью их устранить. Чтобы понизить уровень отказов, попытайтесь удержать пользователей в рамках диалога с виртуальным агентом в момент их попытки немедленно выйти из него или отказаться от взаимодействия в ключевых точках потока.

Методы удержания пользователя в канале применимы к самым разным сценариям. Но действовать следует осторожно — пользователь ни в коем случае не должен почувствовать, что система «удерживает» его насильно. Если такое впечатление все же возникнет, то к моменту, когда клиент наконец попадет к оператору-человеку, он будет раздражен и враждебно настроен.

Поток удержания при отказе включает в себя следующие шаги:

1. Определить истинную цель или потребность пользователя.
2. Оценить, способен ли бот удовлетворить эту потребность.
3. Если способен — убедить пользователя, что бот действительно может решить его задачу.
4. Если не способен — направить пользователя к следующему наилучшему действию (например, к другому виртуальному агенту или оператору-человеку).

Такой поток можно реализовать еще в самом начале диалога, чтобы сразу сократить число мгновенных отказов. Его также можно начать уже после того, как пользователь завершил сценарий. Если клиенты запрашивают оператора сразу

после прохождения «успешного» пути, это может указывать на какую-то проблему: ответ системы либо не понравился, либо не привел к желаемому результату.

11.4.1. Начните со сбора данных об отказе

Если ваша система уже готова к работе или находится на стадии предделоя, вы можете реализовать упрощенный поток отказа, когда пользователя просят указать причину выхода. В самой простой версии бот просто передает обращение оператору вне зависимости от ответа. На рис. 11.9 показан пример потока, предназначенного для сбора данных об отказе. Это довольно ненавязчивая стратегия, позволяющая выяснить, почему пользователи выходят из диалога.

После сбора метрик может оказаться, что требуется улучшить обучение модели или что стратегия решения задач не совпадает с тем, как пользователи на самом деле хотят взаимодействовать с системой.

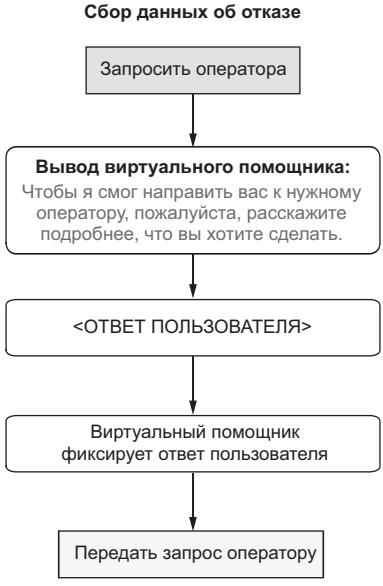


Рис. 11.9. Простой поток для сбора причины отказа

11.4.2. Реализация потока удержания при отказе

После того как вы узнаете, почему именно пользователи отказываются от взаимодействия с системой, можно перейти к более осмысленной обработке подобных диалогов. Первое, что следует сделать, — обновить классификатор и предпринять необходимые меры по решению проблемных запросов. Сюда относится:

- расширение текущих возможностей бота для обработки новых типов запросов;

- перенаправление пользователя к другому виртуальному агенту;
- эскалация к оператору-человеку (если такая возможность предусмотрена);
- предоставление информации о доступных альтернативах, если перевод на оператора невозможен.

На рис. 11.10 показан диалоговый поток, определяющий, какие запросы находятся в рамках возможностей системы, а какие — за их пределами, с резервным механизмом для запросов, которые классификатор не смог распознать.

В нашем примере с энергетической компанией мы добавили в бота логику, спрашивающую у пользователя о том, какое действие ему необходимо выполнить (и постепенно обучили классификатор распознавать широкий диапазон подобных запросов). Многие пользователи все же настаивали на эскалации, повторно требуя соединить их с оператором. В таком случае они перенаправлялись в общую очередь ожидания.

Некоторые пользователи озвучивали задачу, которую система понимала, но которая выходила за пределы ее возможностей. Таких пользователей необходимо было направить к оператору, специализирующемуся именно на их вопросе (если он доступен), чтобы сократить количество последующих перенаправлений. Так, в одних компаниях существует единый отдел обслуживания клиентов, обрабатывающий все обращения, в других же пользователи направляются к разным операторам или отделам, в зависимости от их запроса: например, в отдел выставления счетов или к операторам, записывающим на прием.

Если пользователь озвучивал цель, соответствующую назначению системы, мы мотивировали его продолжить взаимодействие с ботом. Это давало ему ощущение контроля над процессом и внимания к его запросу. Предоставив информацию о времени ожидания оператора, мы позволили пользователю самостоятельно решать, как ему лучше распорядиться своим временем, а правильно определив его потребность, укрепили доверие к системе. Многие пользователи были готовы продолжить взаимодействие с ботом, если тот демонстрировал компетентность в интересующем их вопросе, как показано в следующем листинге.

Листинг 11.9. Поток удержания при отказе, определяющий цель пользователя и мотивирующий его продолжить работу в канале

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Добрый день. Меня зовут Алиса. Я виртуальный агент ABC
Energy.
Я здесь, чтобы помочь вам с отключением услуги энергоснабжения. Мне потребуется
собрать данные о вашем аккаунте, чтобы составить заявку на отключение. Процесс
займет всего несколько минут. Пожалуйста, оставайтесь на линии, пока я не
продиктую номер подтверждения. Если соединение прервется до подтверждения,
изменения по вашему аккаунту внесены не будут. Если возникнут какие-либо
сложности, я переведу вас на оператора поддержки. Готовы продолжить?

ПОЛЬЗОВАТЕЛЬ: Соедините с оператором.
```

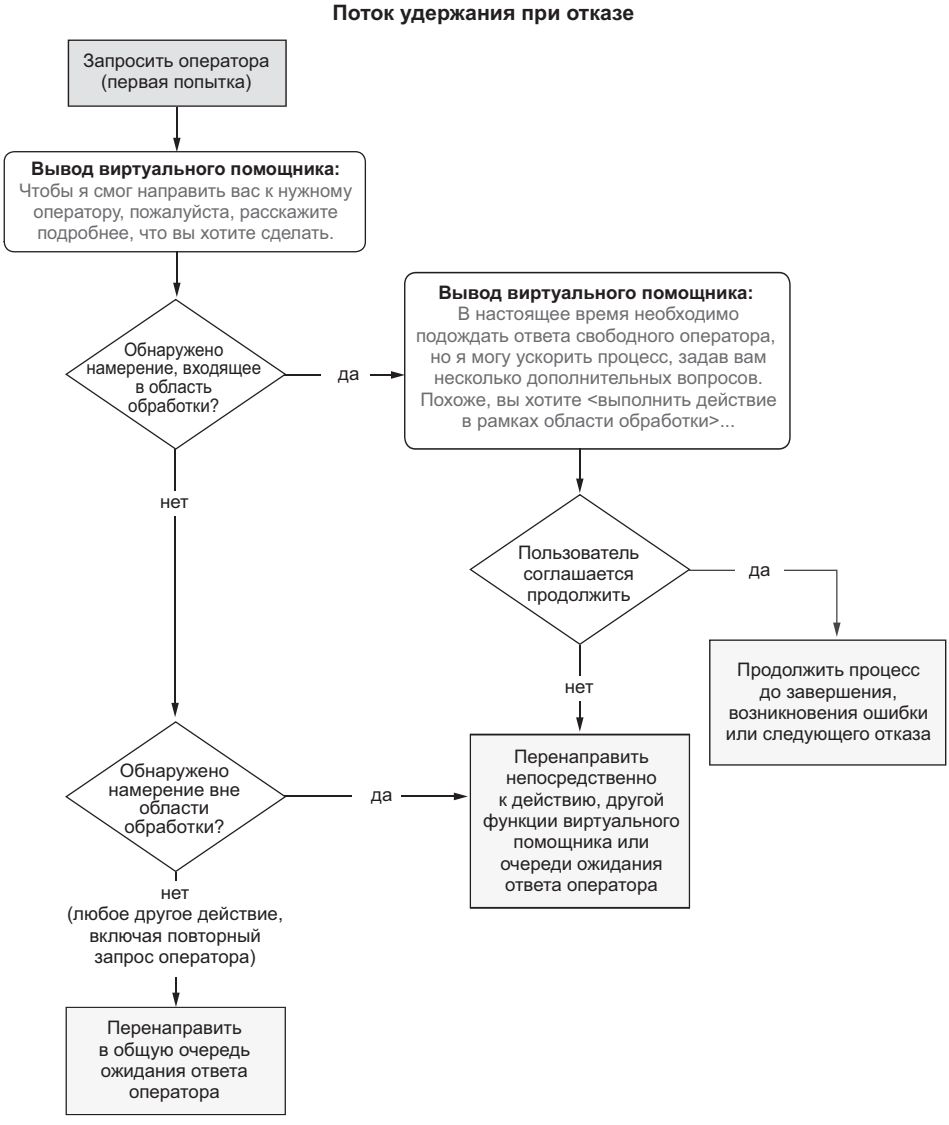


Рис. 11.10. Процесс удержания при отказе направлен на определение потребности пользователя. Поток может распознать запрос, входящий в область обработки; запрос вне области обработки, который система понимает, но не может обработать; либо вовсе нераспознанный запрос. Если распознанный запрос входит в область обработки, бот может замотивировать пользователя остаться в канале. Если запрос находится вне области обработки, бот может перенаправить пользователя к соответствующей функции системы или к очереди ожидания ответа оператора. Если запрос не распознан, пользователь направляется в очередь ожидания оператора — общую или установленную по умолчанию

```
ВИРТУАЛЬНЫЙ ПОМОЩНИК: Чтобы я смогла направить вас к нужному оператору,
пожалуйста, расскажите подробнее, что вы хотите сделать.

ПОЛЬЗОВАТЕЛЬ: Я хочу прекратить обслуживание и закрыть счет прямо сейчас.

ВИРТУАЛЬНЫЙ ПОМОЩНИК: В настоящее время необходимо подождать ответа свободного
оператора, но я могу ускорить процесс, задав вам несколько дополнительных
вопросов. Похоже, вы хотите прекратить подачу электроэнергии и получить итоговый
счет. Верно?

ПОЛЬЗОВАТЕЛЬ: Да.

ВИРТУАЛЬНЫЙ ПОМОЩНИК: Отлично! Давайте начнем с поиска информации по вашему
счету.
```

Затем виртуальный агент продолжал собирать всю необходимую информацию, пока не возникала ошибка или пока абонент повторно не запрашивал оператора. Это позволяло либо полностью выполнить задачу клиента в автоматизированном режиме, либо собрать максимум информации до передачи обращения оператору, тем самым сократив время, необходимое человеку для завершения звонка.

Новая схема взаимодействия оказалась весьма успешной. Среди звонящих, которые изначально запрашивали оператора, но при этом озвучивали свою цель, соответствующую возможностям виртуального агента, 38 % удалось убедить остаться и пройти успешную аутентификацию. В сочетании с другими обновлениями системы, описанными в данной главе, общая доля успешно завершенных сценариев возросла с 27 до 30 %.

Задания

Проанализируйте, что вы узнали о стратегиях удержания пользователя:

1. Наблюдаются ли в вашем боте определенные закономерности отказов, причины которых вам неясны (то есть вы не понимаете, почему пользователь запросил оператора)?
2. Удалось ли бы вам точнее определить необходимое улучшение системы, если бы вы попросили пользователя предоставить больше информации о требуемом действии?

11.5. Улучшение диалога с помощью генеративного ИИ

В главе 5 мы показали, как можно использовать генерацию, дополненную поиском (RAG), для формирования ответов чат-бота в режиме реального времени. Однако даже если вы предпочитаете строить более статичные и контролируемые сценарии, генеративный ИИ все равно будет вам полезен в процессе разработки дизайна диалогов. Дизайнеры диалогов специализируются на

написании реплик, отвечающих интересам как системы, так и пользователей. Но если в вашей команде нет такого специалиста, то можно использовать генеративный ИИ для создания сообщений, обеспечивающих достижение нужных вам целей.

В этом разделе мы рассмотрим несколько приемов улучшения статических ответов с помощью генеративного ИИ.

11.5.1. Улучшение сообщений об ошибках с помощью генеративного ИИ

Написание качественных сообщений об ошибках — непростая задача. Порой система выдает ответ, который технически корректен, но для пользователя звучит сухо или даже грубо. Это особенно нежелательно в голосовых каналах, где пользователь может совершить ошибку, даже того не осознавая. Например, если пользователя просят ввести ID определенной длины, он может случайно пропустить одну цифру или, наоборот, добавить лишнюю.

Для номера социального страхования, состоящего из 9 цифр, формально корректное, но несколько резкое сообщение может звучать так: «Требуется девять цифр. Введите правильный номер социального страхования». Попробуем улучшить это сообщение с помощью генеративного ИИ.

Так как задача носит творческий характер, мы будем использовать инструкционную модель (mixtral-8x7b-instruct-v01-q) и декодирование с семплированием (sampling decoding). При таком подходе ответы модели будут недетерминированными. Мы выполним запрос несколько раз, чтобы получить несколько вариантов и затем выбрать лучший из них.

Листинг 11.10. Улучшение сообщения об ошибке с помощью генеративного ИИ — итерация 1

```
Инструкция: Ты — редактор текста, разрабатывающий диалоги для
голосовой диалоговой системы. Тебе будет предоставлен сценарий
и исходное сообщение (ввод). Переделай ввод в вывод.
Вывод должен быть кратким и содержать не более 12 слов.
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен звучать доброжелательно.
Вывод не должен указывать на ошибку пользователя.

Сценарий: Пользователь ввел номер социального страхования
неверной длины.
Вход: Требуется девять цифр. Введите правильный номер
социального страхования.
Выход: Пожалуйста, введите 9-значный номер социального страхования.
```

Инструкция и указания для модели

Сценарий и исходное сообщение, которое необходимо улучшить

Вывод LLM

После трех запусков этого промпта были получены следующие ответы:

- «Пожалуйста, введите 9-значный номер социального страхования».
- «Пожалуйста, введите полный 9-значный номер социального страхования».
- «Пожалуйста, введите 9 цифр номера социального страхования. Спасибо!»

Такие сообщения звучат гораздо лучше. Однако их можно сделать еще более вежливыми, добавив извинение. Расширим инструкцию, включив в нее соответствующее требование.

Листинг 11.11. Улучшение сообщения об ошибке с помощью генеративного ИИ — итерация 2

```
Инструкция: Ты – редактор текста, разрабатывающий диалоги для
голосовой диалоговой системы. Тебе будет предоставлен сценарий
и исходное сообщение (ввод). Переделай ввод в вывод.
Вывод должен быть кратким и содержать не более 12 слов.
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен звучать доброжелательно.
Вывод не должен указывать на ошибку пользователя.
Вывод должен включать извинение.

Сценарий: Пользователь ввел номер социального
страхования неверной длины.
Вход: Требуется 9 цифр. Введите правильный
номер социального страхования.
Выход: Пожалуйста, 9-значный номер социального страхования.
```

Исходная инструкция

Новая инструкция — указано добавить извинение

Исходный сценарий и ввод

Вывод LLM

После шести запусков измененного промпта были получены следующие ответы:

- «Извините, пожалуйста, введите 9-значный номер социального страхования».
- «Приношу извинения. Пожалуйста, введите правильный номер социального страхования».
- «Извините за неудобства, попробуйте снова, введите 9 цифр».
- «Прошу прощения, давайте попробуем снова. Введите, пожалуйста, 9-значный номер социального страхования».
- «Приношу извинения, пожалуйста, введите 9-значный номер социального страхования».
- «Прошу прощения, немного неверно. Попробуйте снова, введя 9 цифр».

Можно выбрать один из вариантов и использовать его без изменений, либо составить новый ответ из разных фрагментов, например: «Извините, давайте попробуем снова. Пожалуйста, введите 9-значный номер социального страхования».

11.5.2. Улучшение приветствия с помощью генеративного ИИ

Наше приветственное сообщение могло вызвать замешательство у некоторых клиентов энергетической компании, потому что переход от одной системы IVR к другой был довольно резкий. Может ли генеративный ИИ обновить приветствие нашего бота? Узнаем в листинге ниже.

Листинг 11.12. Улучшение приветственного сообщения с помощью генеративного ИИ — итерация 1

```
<|инструкция|>
Ты – редактор текста, разрабатывающий диалоги для голосовой диалоговой системы.
Тебе будет предоставлен сценарий и исходное сообщение (ввод). Переделай ввод
в вывод.
Вывод должен содержать приветствие.
Вывод должен быть полезным и давать пользователю четкие указания.

<|сценарий|>
Пользователь звонит в свою энергетическую компанию ABC Energy. Пользователь
подключается к первой диалоговой системе. Система предоставляет меню.
Пользователь нажимает кнопку. Пользователь переводится на вторую диалоговую
систему.

<|ввод|>
Я помогу вам прекратить обслуживание. Для начала мне нужна информация о вашем
текущем адресе. Выберите тип аккаунта: частный или коммерческий?

<|вывод|>
Здравствуйте! Благодарим за обращение в ABC Energy. Я помогу
вам прекратить обслуживание. Для начала не могли бы вы уточнить,
является ли ваш аккаунт частным или коммерческим? Получив эту
информацию, я смогу лучше помочь вам с вашим запросом.
```

Вывод LLM

Первый вариант вывода уже получился неплохим, но можно сгенерировать еще несколько:

- «Здравствуйте! Добро пожаловать в ABC Energy. Похоже, вы хотите прекратить обслуживание. Прежде чем мы продолжим, могу я уточнить, о каком типе аккаунта идет речь: частном или коммерческом? Кроме того, не могли бы вы сообщить свой текущий адрес, чтобы мы могли корректно обработать ваш запрос? Спасибо».
- «Здравствуйте! Добро пожаловать в ABC Energy. Я здесь, чтобы вам помочь. Для начала не могли бы вы уточнить, является ли ваш аккаунт частным или коммерческим? После этого я смогу помочь вам с любыми вопросами или проблемами, связанными с прекращением обслуживания».

Мы получили несколько вариантов, как можно улучшить наше приветственное сообщение. Они все включают в себя приветствие, название компании

и вступительную часть. Затем следует основной вопрос и комментарий после него. Есть только одна проблема: сообщение должно заканчиваться вопросом — приглашением пользователя к ответу. (Это особенно важно в голосовых системах, так как если задать вопрос и потом продолжить говорить, пользователь может начать отвечать еще до окончания сообщения бота.) Мы можем вручную удалить последнюю часть или же попробовать обновить инструкцию для LLM.

Листинг ниже содержит новую инструкцию.

Листинг 11.13. Улучшение приветственного сообщения с помощью генеративного ИИ — итерация 2

```
<|инструкция|>
Ты — редактор текста, разрабатывающий диалоги для голосовой диалоговой системы.
Тебе будет предоставлен сценарий и исходное сообщение (ввод). Переделай ввод
в вывод.
Вывод должен содержать приветствие.
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен заканчиваться вопросом.  <-- Новая инструкция

<|сценарий|>
Пользователь звонит в свою энергетическую компанию ABC Energy. Пользователь
подключается к первой диалоговой системе. Система предоставляет меню.
Пользователь нажимает кнопку. Пользователь переводится на вторую диалоговую
систему.

<|ввод|>
Я помогу вам прекратить обслуживание. Для начала мне нужна информация о вашем
текущем адресе. Выберите тип аккаунта: частный или коммерческий?

<|вывод|>
Здравствуйте! Я помогу вам прекратить обслуживание в ABC Energy.
Для начала не могли бы вы уточнить, речь идет о частном
или коммерческом аккаунте? Спасибо!
```

Вывод LLM

Сообщение стало лучше, но все еще заканчивается словом «Спасибо!». Мы просили LLM не делать этого, но она все равно поступила по-своему. На самом деле при каждом запуске данного промпта модель завершала сообщение именно так, поэтому мы изменили инструкцию на:

```
Вывод должен заканчиваться вопросом. Вывод не должен заканчиваться "Спасибо!"
```

И тем не менее это лишь небольшой недочет. Мы всегда можем удалить «Спасибо!» вручную и использовать обновленное сообщение.

Приветствие также можно улучшить, если дополнительно описать возможности бота и объяснить пользователю, что от него ожидается. Следующий листинг содержит еще один вариант приветствия.

Листинг 11.14. Улучшение приветственного сообщения с помощью генеративного ИИ — итерация 3

```
<|инструкция|>
Ты – редактор текста, разрабатывающий диалоги для голосовой
диалоговой системы. Тебе будет предоставлен сценарий и исходное
сообщение (ввод). Переделай ввод в вывод.
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен описывать возможности системы.
Система помогает пользователям прекратить энергоснабжение.
Вывод должен сообщать пользователю, что ему нужно будет ответить
на несколько вопросов, касающихся обслуживания.
Вывод должен сообщать пользователю, что изменения не будут внесены
до завершения процесса.
Вывод должен заканчиваться подтверждением готовности пользователя
продолжить.
```

Подробная инструкция

```
<|сценарий|>
Пользователь звонит в свою энергетическую компанию ABC Energy, чтобы
прекратить энергоснабжение. Диалоговая система компании приветствует
пользователя.

<|ввод|>
Я здесь, чтобы помочь вам прекратить энергоснабжение.
Для этого мне необходимо собрать некоторую информацию
об обслуживании. Готовы продолжить?
```

Простое приветствие, которое мы просим LLM улучшить

```
<|вывод|>
Добро пожаловать в ABC Energy! Я здесь, чтобы помочь вам прекратить
энергоснабжение. Чтобы все прошло успешно, мне нужно будет задать
вам несколько вопросов о вашем аккаунте. Уверяю вас, изменения не
будут внесены, пока прекращение обслуживания не будет окончательно
подтверждено. Готовы продолжить?
```

Вывод LLM

Отличное приветствие! Напомним, что мы используем декодирование с применением семплирования и поэтому получаем недетерминированные ответы. У вас, скорее всего, будет другой результат. Ниже приведено еще несколько примеров ответов от LLM:

- «Добро пожаловать в ABC Energy! Я могу помочь вам прекратить энергоснабжение уже сегодня. Чтобы это сделать, мне потребуется задать вам несколько вопросов, касающихся деталей вашего аккаунта. Обращаю ваше внимание, что никакие платежи не будут начислены до полного завершения процесса. Вы готовы продолжить?»
- «Добро пожаловать в ABC Energy! Я могу помочь вам прекратить энергоснабжение. Для этого я задам несколько вопросов, связанных с вашим аккаунтом. Обратите внимание, что никакие платежи не будут начислены в ходе данного процесса. Вы готовы предоставить необходимые данные?»

Вы можете сами поэкспериментировать с разными инструкциями, чтобы получить другое приветствие, или же использовать то сообщение, которое вам больше всего нравится.

Одним из способов улучшения является мотивирование пользователя — для этого нужно сказать ему, что процесс не займет много времени. Дополним нашу инструкцию, чтобы отразить данное требование. В следующем листинге промпт расширяется инструкцией: «Необходимо замотивировать пользователя тем, что вопросы простые и ответ на них не займет много времени».

Листинг 11.15. Улучшение приветственного сообщения с помощью генеративного ИИ — итерация 4

```
<|инструкция|>
Ты — редактор текста, разрабатывающий диалоги для голосовой диалоговой системы.
Тебе будет предоставлен сценарий и исходное сообщение (ввод). Переделай ввод
в вывод.
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен описывать возможности системы. Система помогает пользователям
прекратить энергоснабжение.
Необходимо замотивировать пользователя тем, что вопросы
простые и ответ на них не займет много времени.
Вывод должен сообщать пользователю, что ему нужно будет ответить на несколько
вопросов, касающихся обслуживания.
Вывод должен сообщать пользователю, что изменения не будут внесены до завершения
процесса.
Вывод должен заканчиваться подтверждением готовности пользователя продолжить.

<|сценарий|>
Пользователь звонит в свою энергетическую компанию ABC Energy, чтобы прекратить
энергоснабжение. Диалоговая система компании приветствует пользователя.

<|ввод|>
Я здесь, чтобы помочь вам прекратить энергоснабжение.
Для этого мне необходимо собрать некоторую информацию
об обслуживании. Готовы продолжить?

<|вывод|>
Добро пожаловать в ABC Energy! Я здесь, чтобы помочь вам
прекратить энергоснабжение. Чтобы все прошло успешно, мне
нужно будет задать вам несколько простых и быстрых вопросов
о вашем аккаунте. Уверяю вас, изменения не будут внесены, пока
прекращение обслуживания не будет окончательно подтверждено.
Готовы начать?
```

Новая инструкция

Исходное приветствие, которое мы просим LLM улучшить

Вывод LLM (новый фрагмент выделен жирным)

LLM успешно преобразует сообщение, сохраняя его краткость, но теперь оно поясняет, что последующие вопросы будут «простыми и быстрыми» для пользователя.

Давайте еще немного усовершенствуем приветствие. Пользователя следует заверить, что при необходимости к звонку подключится специалист службы поддержки. В следующем листинге мы добавили соответствующую инструкцию: «Вывод должен сообщать пользователю, что представитель службы поддержки подключится к звонку, если пользователь не сможет завершить процесс самостоятельно».

Листинг 11.16. Улучшение приветственных сообщений с помощью генеративного ИИ — итерация 5

```
<|инструкция|>
Ты — редактор текста, разрабатывающий диалоги для голосовой диалоговой системы.
Тебе будет предоставлен сценарий и исходное сообщение (ввод).
Переделай ввод в вывод.
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен описывать возможности системы. Система помогает пользователям
прекратить энергоснабжение.
Вывод должен сообщать пользователю, что ему нужно будет ответить на несколько
вопросов, касающихся обслуживания. Необходимо замотивировать пользователя тем,
что вопросы простые и ответ на них не займет много времени.
Вывод должен сообщать пользователю, что изменения не будут внесены до завершения
процесса.
Вывод должен сообщать пользователю, что представитель службы
поддержки подключится к звонку, если пользователь не сможет
завершить процесс самостоятельно.
Вывод должен заканчиваться подтверждением готовности пользователя продолжить.

<|сценарий|>
Пользователь звонит в свою энергетическую компанию ABC Energy, чтобы прекратить
энергоснабжение. Диалоговая система компании приветствует пользователя.

<|ввод|>
Я здесь, чтобы помочь вам прекратить энергоснабжение.
Для этого мне необходимо собрать некоторую информацию
об обслуживании. Вы готовы?

<|вывод|>
Добро пожаловать в ABC Energy! Я здесь, чтобы помочь вам
прекратить энергоснабжение. Чтобы все прошло успешно, мне нужно
будет задать вам несколько простых вопросов о вашем аккаунте.
Уверяю вас, изменения не будут внесены, пока прекращение
обслуживания не будет окончательно подтверждено. Если возникнут
какие-либо трудности, представитель службы поддержки с радостью
вам поможет. Готовы начать?
```

Новая инструкция

Исходное приветствие, которое мы просим LLM улучшить

Вывод LLM (новый фрагмент выделен жирным)

После добавления всей необходимой информации приветственное сообщение стало слишком длинным. Попросим модель сократить его, добавив новую инструкцию.

Следующий листинг добавляет требование: «Итоговое сообщение должно содержать не более 40 слов».

Листинг 11.17. Улучшение приветственных сообщений с помощью генеративного ИИ — итерация 6

```
<|инструкция|>
Ты — редактор текста, разрабатывающий диалоги для голосовой диалоговой системы.
Тебе будет предоставлен сценарий и исходное сообщение (ввод).
Переделай ввод в вывод.
```

```
Вывод должен быть полезным и давать пользователю четкие указания.
Вывод должен описывать возможности системы. Система помогает пользователям
прекратить энергоснабжение.
Вывод должен сообщать пользователю, что ему нужно будет ответить на несколько
вопросов, касающихся обслуживания. Необходимо замотивировать пользователя тем,
что вопросы простые и ответ на них не займет много времени.
Вывод должен сообщать пользователю, что изменения не будут внесены до завершения
процесса.
Вывод должен сообщать пользователю, что представитель службы поддержки
подключится к звонку, если пользователь не сможет завершить процесс
самостоятельно.
Вывод должен заканчиваться подтверждением готовности пользователя
продолжить.
Вывод должен содержать не более 40 слов.
```

Новая инструкция

```
<|сценарий|>
Пользователь звонит в свою энергетическую компанию ABC Energy, чтобы прекратить
энергоснабжение. Диалоговая система компании приветствует пользователя.

<|ввод|>
Я здесь, чтобы помочь вам прекратить энергоснабжение.
Для этого мне необходимо собрать некоторую информацию
об обслуживании. Вы готовы?
```

Исходное приветствие, которое мы просим LLM улучшить

```
<|вывод|>
Отлично! Чтобы прекратить энергоснабжение, я задам вам несколько
простых и быстрых вопросов. Изменения не будут внесены до
завершения процесса. Если возникнут какие-либо трудности, к вам
подключится оператор. Готовы продолжить?
```

Вывод LLM

Генеративный ИИ — отличный способ усовершенствовать ваши диалоги!

11.6. Иногда эскалация необходима

Бывают ситуации, когда виртуальный помощник должен сообщить пользователю информацию, которая его не обрадует. Например, если пользователь подал заявление и потом узнает, что оно было отклонено, он, скорее всего, останется недоволен и захочет как-то исправить ситуацию. Если автоматизированная система не предлагает пути для разрешения или пересмотра вопроса, пользователь, скорее всего, предпочтет покинуть диалог с ботом.

Грамотное проектирование позволяет создавать проактивные сценарии, которые предугадывают последующие потребности пользователя. Таким образом, даже если эскалация запроса действительно является оптимальным решением, ваши метрики смогут различать «отказ от взаимодействия» и «осознанную эскалацию по бизнес-причинам». В следующей главе мы рассмотрим, как оптимизировать передачу обращения оператору-человеку.

Итоги

- Отказы являются одной из основных причин низкого удержания пользователей, из-за чего виртуальный агент не достигает поставленных бизнес-целей.
- Пользователи, которые прерывают взаимодействие в самом начале диалога, чаще всего делают это из-за недостаточной уверенности в возможностях виртуального агента и его способности решить их задачу.
- Отказы, происходящие на более поздних этапах диалога, указывают на слабое понимание системы или ошибки в проектировании сценария.
- Удержание в момент отказа представляет собой эффективную стратегию повышения показателя удержания, а также источник ценных данных об ожиданиях пользователей от вашего бота.
- Генеративный ИИ может существенно помочь в разработке вежливых и при этом эффективных ответов для ваших диалоговых сценариев.

12 Резюмирование при передаче диалога оператору

В этой главе

- ✓ Определение структуры эффективного резюме диалога
- ✓ Инструментирование вашего разговорного ИИ, чтобы улучшить резюме
- ✓ Преобразование транскрипта чата в связный текст с помощью LLM
- ✓ Извлечение структурированных данных из транскрипта чата с помощью LLM

Разработчики разговорного ИИ часто хотят, чтобы их системы обрабатывали все пользовательские запросы. Однако в большинстве сценариев определенный процент пользователей завершает взаимодействие с человеком, а не с ботом. Разговорный ИИ создается для обработки легко автоматизируемых диалогов и перенаправления более сложных или ценных запросов к операторам. Пользователи, пожелавшие выполнить свою задачу с помощью бота, будут разочарованы «неудачным» исходом взаимодействия, а потому важно обеспечить для них максимально плавный переход от системы к оператору.

Два самых простых способа передачи диалога оператору одновременно являются и наименее предпочтительными из них. Мы можем передать диалог «вслепую», заставив оператора заново запрашивать всю необходимую информацию. Или же мы передадим оператору полный транскрипт диалога, в котором ему придется

самостоятельно искать нужные данные (пока пользователь ждет!). Лучше всего предоставить оператору целевое резюме диалога, чтобы он мог сразу продолжить сценарий с того места, где закончил разговорный ИИ.

12.1. Резюмирование: основные сведения

Резюме диалога обеспечивает плавную передачу взаимодействия от ИИ к человеку. Сначала давайте рассмотрим, зачем вообще нужны такие резюме, а затем изучим элементы, из которых они состоят.

12.1.1. Зачем нужно резюмирование

Большинство решений разговорного ИИ сохраняют полные транскрипты диалогов для возможного аудита и анализа данных. Это настоящий клад обучающих данных, конечно, если у вас есть время для их обработки. Но когда вы — оператор колл-центра, которому только что был передан разговор от ИИ, у вас нет времени читать длинный транскрипт. Вам нужно быстро понять суть проблемы пользователя и сразу приступить к ее решению. Некоторые диалоги занимают множество страниц (сотни слов), тогда как оператору нужно лишь несколько основных деталей: слишком большой объем данных потребует много времени на чтение, а чересчур малый не даст достаточно информации.

На рис. 12.1 показано, как из полного транскрипта диалога создается его краткое резюме при переводе на оператора-человека.

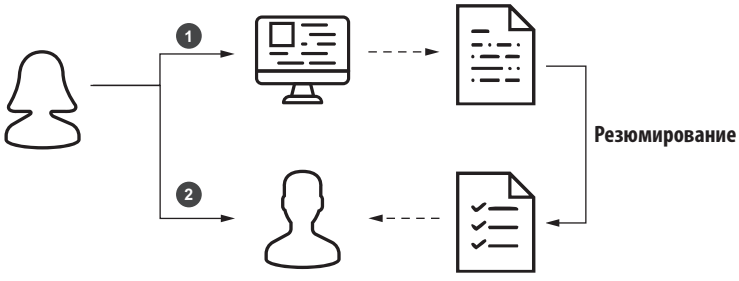


Рис. 12.1. Хорошее резюме помогает оператору быстро войти в курс дела, даже если пользователь ранее вел длительный диалог с ботом

Из предыдущей главы мы знаем, что некоторые пользователи сразу переводятся на оператора, но многие все же проходят несколько шагов, прежде чем запросить эскалацию. В следующем листинге приведен пример диалога, в котором пользователь хоть и достиг своей цели, но все равно запрашивает оператора.

Листинг 12.1. Пример диалога между пользователем и ИИ

```
ИИ: Я автоматизированный помощник, чем я могу вам помочь?
Пользователь: Статус заявления.
ИИ: Хорошо, статус заявления. Укажите ваш номер налогоплательщика.
Пользователь: 123456789.
ИИ: Спасибо. Укажите восьмизначный идентификатор клиента.
Пользователь: 87654321.
ИИ: Хорошо, ваша дата рождения?
Пользователь: 31.12.2000.
ИИ: Ваше имя — Джон Доу?
Пользователь: Да.
ИИ: Отлично! Укажите дату подачи заявления.
Пользователь: 1 февраля 2024.
ИИ: Я нашел ваше заявление №111222333 от 1 февраля
2024 года для клиента №87654321 и поставщика №123456789.
Возмещение на сумму $201,83 было выплачено 23 мая 2024 года.
Пользователь: Оператор.
```

Пользователь достигает своей цели

Пользователь все же запрашивает оператора

Мы не знаем, почему пользователь так поступил (может, ему не пришел чек?). Но причина не столь важна. В большинстве решений на базе разговорного ИИ пользователи могут перейти к оператору на любом этапе сценария. Диалог в листинге 12.1 состоял из семи реплик и занял около двух минут. Этого достаточно, чтобы его прочтение и осмысление требовали определенных усилий от оператора. Представьте себя на его месте — захотели бы вы это делать? А если бы диалог был еще длиннее?

Плавная передача должна происходить быстро. Оператору необходимо сразу понять, что произошло, чтобы принять эффективные меры. Пользователь не должен ждать, пока он разберется в проблеме. Правильно составленное резюме позволяет удовлетворить все эти требования.

12.1.2. Структура эффективного резюме

Резюме диалога должно включать ровно столько информации, сколько необходимо для того, чтобы понять суть диалога. Оно состоит из структурированных метаданных и краткого текстового обзора, а его содержание зависит от конкретного сценария использования. Пример приведен на рис. 12.2.

Элементы метаданных

Структурированные элементы резюме формируются из закрытых вопросов, таких как «Какой у вас ID клиента?» (в отличие от открытых вопросов вроде «Чем я могу вам помочь?»). Такие элементы могут включать данные, собранные в ходе диалога, или контекст, полученный извне.

Примеры элементов этого типа:

- ID пользователя, вошедшего в систему (чат), или номер телефона звонящего (голос/SMS);

- идентификаторы, собранные в ходе диалога;
- идентификаторы, обнаруженные в процессе диалога;
- количество сессий чата, проведенных пользователем ранее;
- результаты анализа тональности пользовательских высказываний.

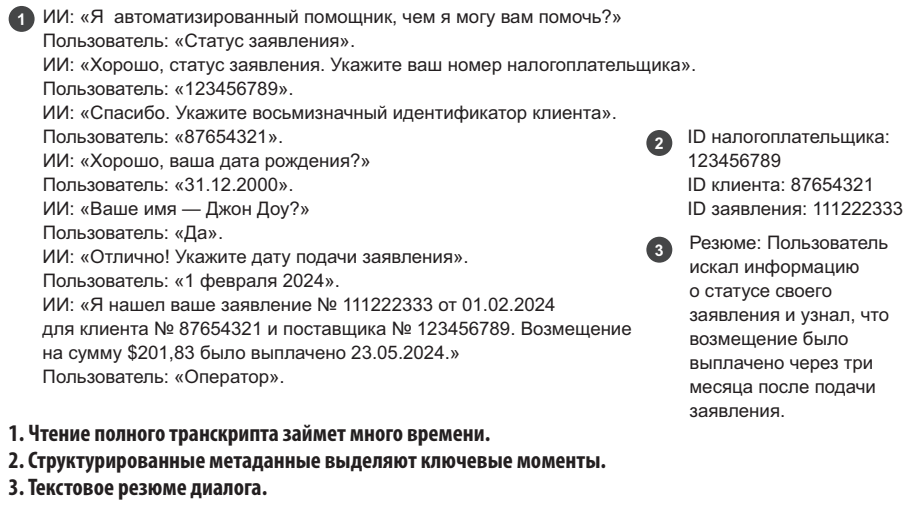
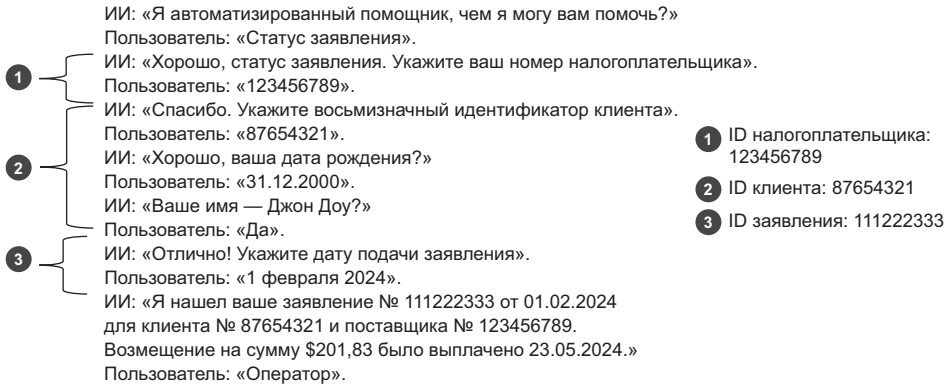


Рис. 12.2. Эффективное резюме извлекает ключевые детали из диалога. В данном примере оно состоит из краткого содержания диалога и сведений о последнем заявлении, по которому выполнялся поиск. На обработку вызова ИИ потребовалось две минуты, тогда как оператор может прочитать резюме всего за несколько секунд

На рис. 12.2 бот собрал пять видов данных, однако оператору требуется только три из них. На рис. 12.3 показана структура данного резюме.

В ходе поиска заявления в медицинскую страховую компанию требуется значительный объем данных для подтверждения личности звонящего. Необходимо проверить, кто звонит, о ком и о чем идет речь. Для подтверждения информации о клиенте требуется три вида данных: ID, дата рождения и подтверждение имени. Оператору, принимающему звонок, достаточно знать, что клиент был верифицирован, то есть понадобится только его ID.

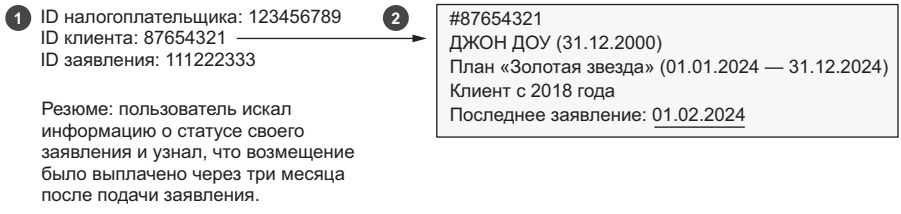
Аналогичным образом можно выделить данные, относящиеся к заявлению: часть из них предоставляет пользователь (дата подачи заявления), а часть — ИИ (статус заявления, дата возмещения, сумма). В резюме включается только ID заявления, и этого достаточно, чтобы оператор при необходимости мог получить полную информацию, включая детали, не упомянутые в диалоге.



1. Один вопрос идентифицирует поставщика.
2. Три вопроса идентифицируют клиента.
3. Данные о поставщике, клиенте и дате идентифицируют заявление.

Рис. 12.3. Не каждый закрытый вопрос должен быть отражен в резюме. В данном случае при проверке страхового заявления наиболее важными являются ID поставщика, клиента и заявления

Элементы резюме можно включать в программное обеспечение, используемое операторами при обработке запросов. В колл-центрах это называется *«всплывающее окно»* (screen pop) — объект, отображающий контекстную информацию для оператора во время разговора. На рис. 12.4 приведен пример такого окна для страхового агента. Оператор получает структурированные данные в виде краткого сводного блока и может получить дополнительную информацию благодаря интеграции с бэкендом. Нажатие на ID клиента позволяет открыть сведения о нем или изображение его страхового полиса, а нажатие на ID заявления — само заявление (например, в формате PDF).



1. Элементы резюме, полученные из диалога.
2. Всплывающее окно оператора колл-центра.

Рис. 12.4. Если резюме интегрировано с ПО колл-центра, оператор получает доступ к широкому спектру информации: например, нажав на ID клиента в системе, может узнать дополнительные сведения о нем

Структурированные метаданные предоставляют ключевые виды данных, полученные в ходе диалога. Это избавляет оператора от необходимости повторно задавать вопросы, на которые пользователь уже ответил боту. Пользователи *чрезвычайно* не любят, когда им приходится повторять одно и то же! Оператору удобно иметь все данные под рукой, но ему также нужно понимать общий контекст разговора. Для этого нам и нужно текстовое резюме.

Элементы текстового резюме

До перехода к оператору диалог может содержать сотни слов (в нашем примере транскрипта — около 100 слов). Средняя скорость чтения взрослого человека — около 200 слов в минуту при чтении для удовольствия и ниже при восприятии сложного материала. Пользователь не будет долго ждать, поэтому оператор должен быстро войти в курс дела. Хорошее резюме позволяет сократить это время на несколько минут.

Следуйте принципу «ничего лишнего». Резюме из одного-двух предложений может быстро передать большую часть информации. В нашем примере текстовое резюме — «Пользователь искал информацию о статусе своего заявления и узнал, что возмещение было выплачено через три месяца после подачи заявления» — в первых пяти словах уже отражается весь процесс поиска, а в последних восьми указывается вероятная причина обращения к оператору.

Текстовое резюме не должно содержать повторов, исключая избыточную информацию:

- *Первоначальное намерение пользователя* — явно не указывается, так как выводится из факта поиска заявления.
- *Промежуточные вопросы* — в резюме не говорится: «Бот запросил у пользователя номер налогоплательщика, ID клиента, дату рождения и т. д.». Все это выводится из того, что система уже нашла заявление, — она не сделала бы этого без необходимых данных.
- *Структурированное содержимое* — нет необходимости дублировать данные, уже предоставленные в более сжатой форме.

Резюме не указывает, кто инициировал эскалацию — пользователь или система, однако эта информация может быть добавлена как новое поле.

В процессе резюмирования всегда приходится идти на определенные компромиссы. Слишком короткое резюме упускает важные сведения, которые могли бы быть полезны оператору, слишком длинное потребует больше времени на ознакомление по сравнению с чтением исходного диалога. Выбирайте такой метод резюмирования, который лучше всего подходит именно для вашего сценария.

Далее в этой главе мы покажем, как можно использовать генеративный ИИ для автоматического создания резюме. Однако сначала нужно, чтобы система

разговорного ИИ структурировала данные в формате, подходящем для резюмирования.

Задания

Составьте несколько резюме для следующего примера диалога. Далее в этой главе вы будете совершенствовать их с помощью изученных нами методов:

```
ИИ: Я автоматизированный помощник, чем я могу вам помочь?
Пользователь: Статус страхового заявления.
ИИ: Хорошо, статус заявления. Укажите ваш номер налогоплательщика.
Пользователь: 123456789.
ИИ: Спасибо. Укажите восьмизначный идентификатор клиента.
Пользователь: 87654321.
ИИ: Хорошо, ваша дата рождения?
Пользователь: 31.12.2000.
ИИ: Ваше имя — Джон Доу?
Пользователь: Да.
ИИ: Отлично! Укажите дату подачи заявления.
Пользователь: 1 февраля 2024.
ИИ: Я нашел ваше заявление №111222333 от 01.02.2024 для клиента
№87654321 и поставщика №123456789. Возмещение на сумму $201,83 было
выплачено 23.05.2024.
Пользователь: Оператор.
```

1. Составьте только текстовое резюме для данного примера.
2. Составьте структурированное резюме с видами данных. Потребовались бы вам иные элементы, если бы абонент запрашивал сведения о своем страховом плане или хотел заранее оценить стоимость процедуры?
3. Возьмите транскрипт диалога из чат-бота, над которым вы в данный момент работаете. Составьте для него как текстовое, так и структурированное резюме.

12.2. Подготовка чат-бота к резюмированию

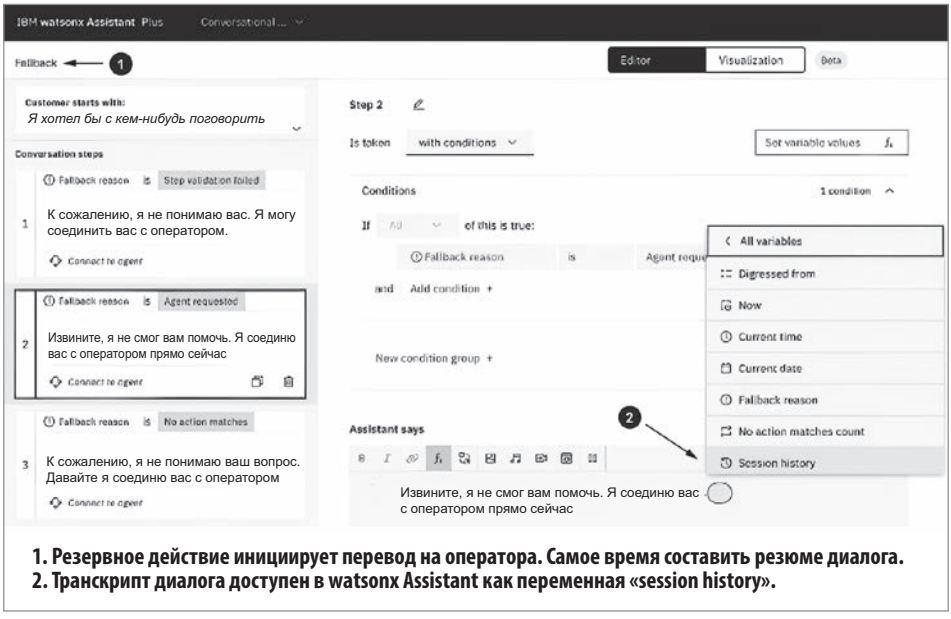
Предположим, что наш чат-бот уже ведет транскрипт всего диалога. Большинство платформ поддерживают данный процесс по умолчанию, но не все. Транскрипт — это минимальный элемент, необходимый для построения резюме. В этом разделе мы рассмотрим несколько способов сбора данных, необходимых как для текстовых, так и для структурированных резюме диалогов.

12.2.1. Использование встроенных элементов

Платформы разговорного ИИ часто уже включают встроенные элементы, предназначенные для резюмирования. Наиболее распространенный из них — это транскрипт диалога, представляющий собой постоянно обновляющийся лог сообщений между пользователем и системой. Доступ к нему можно получить

разными способами в зависимости от платформы. Один из доступных механизмов называется *переменной сессии* (session variable).

На рис. 12.5 показано, как можно получить доступ к транскрипту на нашей платформе (watsonx) через встроенную переменную сессии под названием «session history» (история сессии).



1. Резервное действие инициирует перевод на оператора. Самое время составить резюме диалога.
2. Транскрипт диалога доступен в watsonx Assistant как переменная «session history».

Рис. 12.5. Доступ к транскрипту диалога через переменную «session history»

В зависимости от используемой вами платформы транскрипт диалога будет сохраняться в разных форматах. Наша платформа предоставляет резюме в формате JSON, как показано в следующем листинге.

> **ПРИМЕЧАНИЕ** На многих платформах разговорного ИИ транскрипт недоступен внутри диалоговой сессии как переменная, если вы не создадите его самостоятельно с помощью вебхуков. Мы покажем, как создать переменную с транскриптом далее в этой главе.

Листинг 12.2. Транскрипт диалога через встроенную переменную «session history»

```
[{"a":"Я автоматизированный помощник, чем я могу вам помочь?"},{"u":
➥"Статус заявления","n":true},{"a":"Хорошо, статус заявления.\nУкажите ваш номер
➥налогоплательщика"},{"u": "123456789"},{"a":"Спасибо.\nУкажите восьмизначный
➥идентификатор клиента "},{"u":"87654321"},{"a":" Хорошо, ваша дата
➥рождения?"},{"u":"31.12.2000"},{"a":"Ваше имя – Джон Доу?\nварианты:
```

```
➥ ["Да","Нет"]"},{"u":"Да"},{"a":"Отлично!\nУкажите дату подачи
➥заявления."},{"u":"01.02.2024"},{"a":" Я нашел ваше заявление №111222333 от
01.02.2024
➥для клиента №87654321 и поставщика №123456789. Возмещение на сумму $201,83
было выплачено
➥23.05.2024"},{"u":"оператор"}]
```

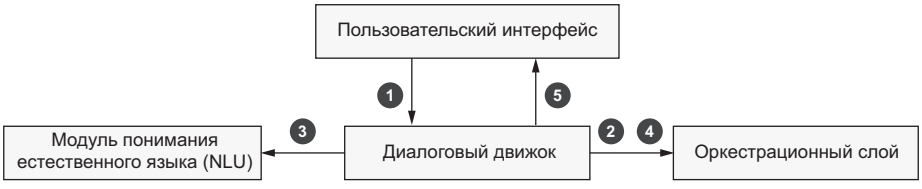
Формат JSON предназначен для чтения машиной, но его можно преобразовать. В этой главе мы сделали иллюстрации более читаемыми, заменив ключи `"a"` на `"Бот"`, ключи `"u"` на `"Пользователь"`, а символы `"\n"` (новые строки) — на пробелы.

Транскрипт также содержит некоторые метаданные, которые можно проигнорировать, например варианты ответа в виде кнопок «Да» и «Нет». Другие платформы разговорного ИИ могут включать дополнительные метаданные, например временны́е метки.

Далее в этой главе (в разделе 12.3) мы покажем, как прогонять данный формат транскрипта через LLM для получения резюме. Вы увидите, что LLM довольно устойчивы к JSON-транскриптам, так что можете создавать резюме как в исходном формате, так и в переработанном, более удобном для восприятия человеком.

12.2.2. Инструментирование чат-бота для создания транскриптов

Некоторые платформы разговорного ИИ требуют, чтобы вы самостоятельно составляли и сохраняли транскрипт диалога либо создавали собственную версию транскрипта в нужном вам формате. В любом случае это реализуется в *оркестрационном слое*, как показано на рис. 12.6. Оркестрационный слой отвечает за вызовы внешних систем через API. Терминология может различаться в зависимости от платформы разговорного ИИ, но чаще всего данный механизм называют *вебхуком*.



1. Пользователи отправляют сообщения в диалоговый движок помощника.
2. Оркестрационный слой при необходимости собирает дополнительный контекст.
3. Компонент NLU интерпретирует сообщение пользователя.
4. Оркестрационный слой при необходимости записывает транскрипт.
5. Диалоговый движок формирует ответ и возвращает его пользователю.

Рис. 12.6. Создать транскрипт диалога в любом необходимом вам формате можно в оркестрационном слое вашего чат-бота

Вебхук представляет собой разновидность API. Вебхуки, как правило, доступны до того, как бот обработает ответ пользователя (так называемый предвебхук), после обработки ответа пользователя (пост-вебхук) или в других заранее определенных событиях. Вебхуки могут получать доступ к контексту беседы либо напрямую, либо через входные параметры. В следующем листинге продемонстрирован псевдокод для создания транскрипта. (Правильную терминологию и формат можно найти в документации вашей платформы разговорного ИИ.)

Листинг 12.3. Псевдокод вебхука, обновляющего транскрипт

```
def transcript_webhook(request, response):
    userMessage = request.input.text
    botMessage = response.output.text
    if (userMessage != None):
        response.context.transcript += 'Пользователь: ' +
        ➥userMessage + '\n'
    if (botMessage != None):
        response.context.transcript += 'Бот: ' +
        ➥botMessage + '\n'
```

Сообщение пользователя обычно содержится в объекте request

Сообщение бота обычно содержится в объекте response

Сообщение пользователя может отсутствовать, например, если диалог начинается с приветствия бота

Транскрипт необходимо сохранить в контекстной переменной (переменной сессии). Все сообщения пользователя и бота добавляются в транскрипт

В листинге 12.3 показан пост-вебхук, так как он имеет доступ и к запросу, и к ответу. Каждый раз, когда бот отвечает пользователю, вебхук обновляет транскрипт. Приведенный транскрипт содержит минимально необходимое количество элементов — только сообщения пользователя и бота — и использует простейший формат: одно сообщение на строку, что удобно для его восприятия человеком. Тем не менее формат транскрипта может быть любым — в виде одной строки, массива строк, объекта JSON или пользовательского объекта.

Как было сказано ранее, простой транскрипт легче читать. Платформы разговорного ИИ, как правило, предоставляют множество дополнительных данных для каждого сообщения, которые вы можете при необходимости включить в свои транскрипты:

- *Метка времени сообщения* — можно использовать для отображения абсолютного времени получения или отправки сообщения («11:25:53») либо относительного времени с начала диалога («00:01:15» для сообщения, отправленного через 1 минуту и 15 секунд после начала сессии).
- *Кнопки* — понадобятся, когда бот предлагает пользователю варианты ответов в виде кнопок и тот выбирает один из них. Особенно полезны в голосовых решениях, где пользователь может вводить двухтональные многочастотные сигналы (DTMF, или «тональный набор») через клавиатуру. Так вы будете знать, что пользователь нажал «0», а не произнес «ноль».

- *Исходный, или нормализованный, ввод* — многие платформы разговорного ИИ нормализуют определенные типы данных, например даты и числа. Вы можете использовать как исходную фразу («первое февраля две тысячи двадцать четвертого года»), так и ее нормализованный вариант («01.02.2024»).
- *Элементы с расширенным форматированием и нетекстовые элементы* — ответы вашего бота могут содержать HTML-разметку или даже изображения и ссылки, которые не подходят для транскрипта.

Псевдокод в листинге 12.3 показывает, как обновлять контекстную переменную, отслеживающую ход диалога, но не демонстрирует, как ее инициализировать. Самый простой способ это сделать — добавить пустую строку, например: `response.context.transcript = ''`

Существуют и другие виды данных, связанные с диалогом, которые можно включить в начало транскрипта:

- *Метка времени сессии* — когда начался диалог.
- *Длительность сессии* — как долго длился диалог.
- *Идентификатор пользователя* — может включать сведения о зарегистрированном пользователе, например имя, электронную почту или ID. Для голосовых решений это может быть номер телефона звонящего.
- *Идентификатор устройства и канала* — способ, с помощью которого пользователь взаимодействует с вашим разговорным ИИ, например, тип устройства (телефон или компьютер) либо используемый канал (виджет чата, SMS, Facebook Messenger и т. д.).
- *Причина перевода к оператору* — почему бот передал пользователя человеку, например, «мгновенный отказ», «отказ» или «бот не понял запрос пользователя».

Можно оставить все эти элементы в структурированной части резюме (в виде пар «ключ — значение») и не включать их в текстовую часть, так как они относятся ко всему взаимодействию в целом.

Такие виды данных, а также многие другие обычно доступны на любой платформе разговорного ИИ. Часто они включены в системные логи ИИ, которые могут служить дополнительным источником данных для построения транскриптов. Дополнительные виды данных предоставляются платформами разговорного ИИ, так как они универсальны и применимы к любому диалогу в качестве его метаданных. Это может послужить вашей отправной точкой при создании любого транскрипта.

Как уже было показано ранее в этой главе, хорошее резюме состоит не только из неструктурированного транскрипта. Структурированные метаданные крайне полезны для выделения ключевых моментов диалога. Однако некоторые метаданные не универсальны, а характерны только для конкретной системы.

Так, например, в нашем боте для страховой компании уникальны ID клиентов и заявлений.

Платформа разговорного ИИ не придает этим ID особого значения — они записываются как обычные пользовательские сообщения. Если вы хотите использовать конкретные контекстные элементы своей системы в резюме, необходимо самостоятельно настроить ИИ для их сохранения, чтобы затем включить их в резюме. Давайте рассмотрим, как это можно сделать.

12.2.3. Инструментирование чат-бота (для сбора данных)

Платформы разговорного ИИ, как правило, позволяют сохранять произвольные значения в переменных, которые называются *контекстными переменными* или *переменными сессии*. Их можно использовать для любых значимых данных, собираемых в ходе диалога, особенно если впоследствии они помогут получить дополнительную информацию.

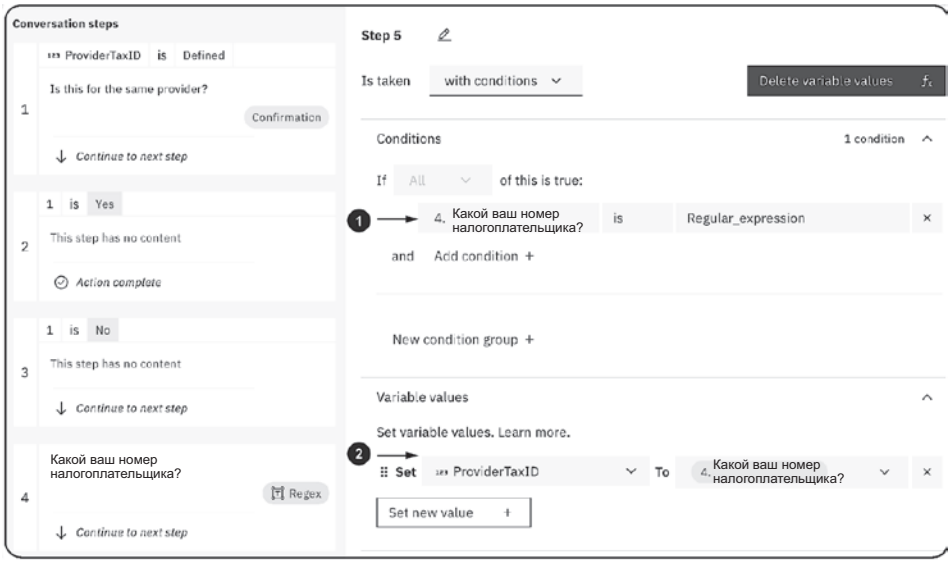
Сохраняемые данные будут различаться в зависимости от конкретного приложения. Вот несколько примеров из разных предметных областей:

- *Медицинское страхование* — ID клиента, ID поставщика, ID страхового заявления.
- *Розничная торговля* — номер заказа, ID товара, место продажи.
- *Банковское дело* — ID счета, тип счета.

Каждый раз, когда бот задает вопрос с ответом фиксированного формата (например: «Какой ID у вашего заявления?»), ответ на него необходимо инструментировать.

> **Будьте осторожны с конфиденциальными данными**
>
> Некоторые типы данных требуют особого обращения. Существуют определенные правила и нормативные акты, регулирующие обращение с персональными или чувствительными данными. Ваш разговорный ИИ может их обрабатывать, но добавление таких данных в резюме или логи требует отдельного согласования с юридическим отделом. Следите за тем, какую информацию вы собираете и храните, как долго вы ее храните, и обязательно подтверждайте правомерность своих действий у юристов.

Способ хранения контекста зависит от используемой вами платформы разговорного ИИ. Возможно, это делается через пользовательский интерфейс, а может, вам потребуется написать для этого код. На рис. 12.7 показан low-code-метод, используемый на нашей платформе для сохранения контекстных переменных.



1. После того как бот проверит структурированный ввод…
2. …значение может быть сохранено в контекстной переменной.

Рис. 12.7. Сохранение значимой информации в контекстной переменной `ProviderTaxID`, чтобы впоследствии использовать ее при создании резюме

Вы можете получить доступ к сохраненным контекстным переменным и в ходе диалога, в том числе для создания структурированного резюме. Листинг ниже содержит псевдокод для доступа к контекстным переменным и их сохранения в структурированном объекте.

Листинг 12.4. Псевдокод вебхука, создающего структурированное резюме

```
def on_transfer(request, response):
    summary = ConversationSummary()

    summary.providerTaxID = response.context.
    ➥ ProviderTaxID
    summary.memberID      = response.context.MemberID
    summary.claimID       = response.context.ClaimID
```

Этот метод вызывается, когда диалог передается оператору

Вы можете выбрать пользовательский объект для хранения резюме

Установите все значения, необходимые для вашего пользовательского резюме

Вы также можете использовать эти переменные непосредственно в своем помощнике для создания неструктурированного резюме. На рис. 12.8 показан low-code-метод, используемый на нашей платформе для объединения нескольких переменных в одну большую строку резюме.

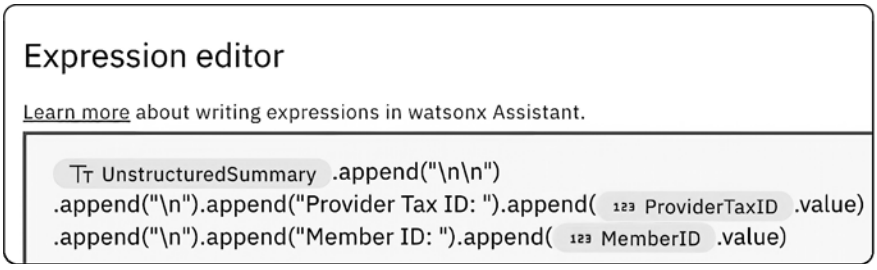


Рис. 12.8. Использование low-code-редактора выражений (`Expression Editor`) для объединения нескольких видов данных в резюме

В этом разделе мы рассмотрели разные способы сбора информации, необходимой для создания резюме. Платформы разговорного ИИ собирают множество данных, которые можно использовать при резюмировании. Вы также можете использовать другие инструменты для сбора дополнительной информации и контролировать ее форматирование. Собранные данные могут быть полезны для множества целей, включая эффективную передачу диалога оператору.

Задания

1 Просмотрите резюме, составленные вами в заданиях раздела 12.1. Изменили бы вы теперь какие-либо их элементы?

12.3. Улучшение резюме с помощью генеративного ИИ

Что делать, если вы не хотите вносить никаких изменений в свой разговорный ИИ? Можно ли все равно собрать и отформатировать все данные, необходимые для качественного резюме? Может ли генеративный ИИ взять на себя часть работы? Да! Рассмотрим, как это сделать.

Для этого необходимо выполнить два обязательных условия. Во-первых, вам нужен траснкрипт диалога в любом виде: встроенный транскрипт из вашей платформы разговорного ИИ, транскрипт, созданный вами вручную или же извлеченный из логов платформы. Во-вторых, необходимо понимать, каким должно быть хорошее резюме для вашего сценария использования. Только после этого вы сможете использовать LLM для резюмирования.

В этом разделе мы будем использовать модель granite-13b-chat-v2 с жадным декодированием. Данная модель хорошо справляется с необходимыми нам

задачами резюмирования и извлечения информации. Жадное декодирование ограничит креативность модели, и ее ответы будут воспроизводимыми (нам нужно получать одно и то же резюме и извлекать одни и те же данные в рамках одного диалога).

12.3.1. Генерация текстового резюме транскрипта с помощью резюмирующих промптов

Начнем с простого резюмирующего промпта, приведенного в листинге ниже. Мы передадим модели JSON-версию транскрипта чата.

Листинг 12.5. Генерация резюме на основе JSON-транскрипта чата

```
Резюмируй следующий транскрипт диалога между
пользователем (u) и автоматизированным помощником (a).
Резюме должно состоять из 1–2 предложений.
```

Описание задачи и подсказка для интерпретации JSON-объекта

Ограничение размера резюме

```
Транскрипт:
[{"a":"Я автоматизированный помощник, чем я могу вам помочь?"},
➥{"u": "Статус заявления","n":true},{"a":"Хорошо, статус
➥заявления.\nУкажите ваш номер налогоплательщика."},{"u":
➥"123456789"},{"a":"Спасибо.\nУкажите восьмизначный
➥идентификатор клиента. "},{"u":"87654321"},{"a":" Хорошо,
➥ваша дата рождения?"},{"u":"31.12.2000"},{"a":"Ваше имя —
➥Джон Доу?\nварианты: ["Да","Нет"]"},{"u":"Да"},{"a":
➥"Отлично!\nУкажите дату подачи заявления."},{"u":"01.02.2024"},
➥{"a":" Я нашел ваше заявление №111222333 от 01.02.2024
➥для клиента №87654321 и поставщика №123456789. Возмещение
➥на сумму $201,83 было выплачено 23.05.2024."},{"u":"оператор"}]
```

JSON-версия транскрипта чата

```
Резюме:
"Я нашел ваше заявление №111222333 от 01.02.2024 для клиента
№87654321 и поставщика №123456789. Возмещение на сумму $201,83
было выплачено 23.05.2024."
```

Сигнал

Вывод модели

Сгенерированное резюме дословно воспроизводит последнюю реплику ИИ. Это может показаться странным, но на самом деле такое резюме довольно хорошо отражает суть взаимодействия. Последнее сообщение бота содержит все ключевые сведения, охватывающие самые важные элементы диалога.

Используя декодирование с семплированием и подходящие настройки температуры, можно получить более креативное резюме, но с риском галлюцинаций. Можно также изменить формат транскрипта и проверить, повлияет ли это на ответ LLM. В следующем листинге приведен промпт, резюмирующий тот же диалог, но с другим входным форматом — неструктурированным текстом вместо JSON.

Листинг 12.6. Генерация резюме на основе текстового транскрипта чата

```
Резюмируй следующий транскрипт диалога между пользователем и ИИ.

Резюме должно состоять из 1–2 предложений.

Транскрипт:
ИИ: "Я автоматизированный помощник, чем я могу вам помочь?"
Пользователь: "Статус заявления".
ИИ: "Хорошо, статус заявления. Укажите ваш номер налогоплательщика".
Пользователь: "123456789".
ИИ: "Спасибо. Укажите восьмизначный идентификатор клиента".
Пользователь: "87654321".
ИИ: "Хорошо, ваша дата рождения?"
Пользователь: "31.12.2000".
ИИ: "Ваше имя — Джон Доу?"
Пользователь: "Да".
ИИ: "Отлично! Укажите дату подачи заявления."
Пользователь: "1 февраля 2024 года".
ИИ: "Я нашел ваше заявление №111222333 от 01.02.2024 для клиента №87654321
и поставщика №123456789. Возмещение на сумму $201,83 было выплачено 23.05.2024".
Пользователь: "Оператор".

Резюме:
ИИ предоставляет информацию о статусе заявления, поданного
клиентом с ID 87654321 и поставщиком с ID 123456789. Возмещение
на сумму $201,83 было выплачено 23 мая 2024 года. Оператор сможет
помочь с любыми вопросами или проблемами, связанными с данным
заявлением.
```

Описание задачи без дополнительных подсказок

Исходное ограничение размера резюме

Текстовая версия транскрипта чата

Сигнал

Вывод модели

Новое резюме содержит ту же информацию, что и резюме, сгенерированное на основе JSON. Оно по-прежнему основано на последнем сообщении бота, но теперь реплика перефразирована в связный текст. В резюме также добавлено третье предложение — несмотря на требования промпта, — интерпретирующее последний ответ пользователя.

> **ПРИМЕЧАНИЕ** В оставшейся части этой главы мы будем опускать сам транскрипт, чтобы не перегружать листинги. Мы будем использовать текстовую версию транскрипта. Полные листинги доступны в репозитории книги на GitHub: https://github.com/andrewrfreed/EffectiveConversationalAI.

Вспомним совет из начала раздела: необходимо понимать, каким должно быть хорошее резюме. Резюме, сгенерированные LLM, приемлемые, но их можно сделать еще лучше, например, предоставив модели более точные инструкции, как показано в листинге ниже. Раз уж мы хотим сделать акцент на инструкциях, то лучше использовать инструкционную модель — granite-13b-instruct-v2. Попросим ее учесть диалог, предшествующий передаче запроса оператору.

Листинг 12.7. Расширение инструкции для LLM

```
Резюмируй следующий транскрипт диалога между пользователем и ИИ.
Резюме должно состоять из 1–2 предложений.
Резюме должно делать акцент на последнем действии пользователя, которое предшествовало его просьбе соединить с оператором.

Транскрипт:
<... текстовая версия транскрипта опущена ...>

Резюме:
Пользователь просит соединить его с оператором, поскольку хочет узнать, почему его заявление было отклонено.
```

Исходное описание задачи

Исходное ограничение размера резюме

Расширенная инструкция с уточнением, на чем сделать акцент

Текстовая версия транскрипта чата

Тот же сигнал

Вывод модели

Модель следует инструкции. Последняя реплика бота сообщает пользователю, что возмещение по его заявлению было выплачено («Возмещение на сумму $201,83 было выплачено 23 мая 2024 года»), после чего пользователь запрашивает оператора. Резюме, сгенерированное LLM, указывает, что «он хочет узнать, почему его заявление было отклонено». Резюме получилось кратким, но содержащим неверное предположение, поскольку заявление было удовлетворено, а не отклонено. Возможно, пользователь ожидал большей суммы возмещения или хотел получить более детализированный отчет. Текстовое резюме не содержит дословного повторения транскрипта, наблюдавшегося в листинге 12.6, оставляя данные сведения для отдельного структурированного резюме. Такой результат уже больше соответствует желаемому. В следующем улучшении промпта попробуем исключить из вывода домыслы модели.

Один из способов достичь желаемого результата — вызвать LLM с one-shot-промптом (с одним примером) или few-shot-промптом (с несколькими примерами). Создание one-shot-примера заставляет задуматься о том, каким должно быть качественное резюме для вашего диалога. Использование одного или нескольких примеров зачастую является самым быстрым способом улучшить промпт.

Следующий листинг содержит one-shot-пример, предоставляемый модели granite-13b-instruct-v2.

Листинг 12.8. One-shot-промпт для резюмирования

```
<|инструкция|>
Резюмируй следующий транскрипт диалога между пользователем и ИИ.
Резюме должно состоять из 1–2 предложений.
Резюме должно делать акцент на последнем действии пользователя, которое предшествовало его просьбе соединить с оператором.
```

Исходное описание задачи с обновленным разграничителем <|инструкция|>

```
<|транскрипт|>
ИИ: "Я автоматизированный помощник, чем я могу вам помочь?"
Пользователь: "Статус заявления".
ИИ: "Хорошо, статус заявления. Укажите ваш номер налогоплательщика".
Пользователь: "012345678".
ИИ: "Спасибо. Укажите восьмизначный идентификатор клиента".
Пользователь: "У меня его нет".
ИИ: "Хорошо, попробуем другой вариант. Дата рождения клиента?"
Пользователь: "Оператор".

<|резюме|>
Пользователь запросил оператора после того, как не смог
предоставить ID клиента, необходимый для поиска заявления.
```

One-shot-пример транскрипта диалога и его резюме

```
<|транскрипт|>
<... текстовая версия транскрипта опущена ...>

<|резюме|>
Пользователь запрашивает оператора, потому что хочет
получить больше информации о своем страховом заявлении.
```

Текстовая версия транскрипта чата

Перестроенный сигнал

Вывод модели

Такое резюме вполне можно использовать вместе со структурированными метаданными. В нем присутствует некоторое предположение — «хочет получить больше информации», — но, судя по контексту, оно вполне логично: если пользователь обратился к оператору после того, как узнал, что его заявление было удовлетворено, ему, скорее всего, требуется какая-то дополнительная информация. Резюме также повторяет структуру примера — начало предложения полностью совпадает с примером, токен за токеном.

Обратите внимание, что в one-shot-примере (листинг 12.8) мы использовали немного другой формат. Вместо простого обозначения `Транскрипт:` применяются специально оформленные токены, такие как, например, `<|транскрипт|>`. Без них не удавалось получить корректные резюме. Модель не могла отделить промпт от фрагмента диалога, вероятно, потому, что обе части содержали двоеточия. Может, в будущем LLM станут более устойчивыми к появлению разделителей в разных местах промпта. И тем не менее подобное незначительное изменение может существенно повлиять на качество работы модели.

Здесь возможны два подхода: сделать акцент на инструкциях или на примерах, каждый из которых предполагает свои компромиссы:

- *Управление выводом* — большинство LLM уже обучены выполнять резюмирование. Многие из них хорошо понимают инструкции и генерируют качественные резюме. One-shot- и few-shot-промпты дополнительно ограничивают формат вывода, показывая модели нужный стиль языка, даже если для этого требуется предоставить несколько примеров.
- *Стоимость* — добавление примеров повышает стоимость инференса, так как увеличивается число входных токенов.

Сгенерированные нами текстовые резюме представляют собой краткий пересказ диалога, но почти не содержат структурированных метаданных, которые были бы полезны оператору. Ранее в этой главе мы рассматривали инструментирование чат-бота с помощью кодинга или low-code-методов для сбора структурированных метаданных, которые затем можно передать человеку. Но что, если мы не хотим инструментировать чат-бота? Может ли LLM извлечь данные самостоятельно?

12.3.2. Генерация структурированного резюме транскрипта с помощью экстрактивных промптов

Мы можем использовать LLM для извлечения структурированных данных из транскриптов диалогов — экстракция (или извлечение) является еще одной задачей, которой обучены многие LLM.

Первый шаг — определение того, как должен выглядеть структурированный результат. Возможных форматов много, но один из наиболее полезных — JSON, удобный по двум причинам: он легко обрабатывается последующими приложениями, и многие LLM хорошо справляются с его генерацией.

Нам снова потребуется инструкционная модель, в этот раз будем использовать mistral-7b-instruct-v0-2, поскольку она способна генерировать JSON-объекты, руководствуясь лишь инструкциями. Следующий листинг содержит промпт, генерирующий структурированный вывод в формате JSON на основе предоставленного диалога.

Листинг 12.9. Экстрактивное резюме без примеров

```
<|инструкция|>
Прочитай следующий транскрипт диалога между
пользователем и автоматизированным помощником.
Извлеки все ID в формате JSON.

<|транскрипт|>
<... текстовая версия транскрипта опущена ...>

<|JSON|>
{
"TaxID": "123456789",
"MemberID": "87654321",
"DateOfBirth": "31.12.2000",
"Name": "Джон Доу",
"ClaimNumber": "111222333",
"DateOfService": "1 февраля 2024 года",
"ProviderID": "123456789"
}
```

Описание задачи с простой инструкцией о необходимом формате JSON

Текстовая версия транскрипта чата

Обновленный сигнал для генерации JSON

Вывод модели

Для первого результата неплохо. Модель сгенерировала JSON и извлекла все структурированные данные. Однако она собрала больше данных, чем

требовалось: мы просили только ID, а модель добавила и другие элементы, при этом продублировала один из ID (номер налогоплательщика — TaxID — оказался тем же, что и ID поставщика, — ProviderID).

> **ПРИМЕЧАНИЕ** Многие модели способны генерировать JSON лишь после одного или нескольких примеров. Все протестированные нами модели извлекали больше видов данных, чем требовалось (требовалось всего три). Модель Mistral оказалась одной из немногих, которая смогла выдать корректный JSON без примеров в промпте. Скорее всего, в будущем модели станут еще лучше справляться с генерацией JSON-данных. В качестве эксперимента можно включить пример JSON-объекта в инструкцию.

Давайте дополним наш промпт примером (выделен жирным).

Листинг 12.10. Экстрактивное резюме с одним примером

```
<|инструкция|>
Прочитай следующий транскрипт диалога между
пользователем и автоматизированным помощником.
Извлеки все ID в формате JSON.

<|транскрипт|>
ИИ: "Я автоматизированный помощник, чем я могу вам помочь?"
Пользователь: "Статус заявления".
ИИ: "Хорошо, статус заявления. Укажите ваш номер налогоплательщика".
Пользователь: "333444555".
ИИ: "Спасибо. Укажите восьмизначный идентификатор клиента".
Пользователь: "55667788".
ИИ: "Хорошо, дата рождения клиента?"
Пользователь: "19 апреля 2024 года".

<|JSON|>
{"TaxID": 333444555, "MemberID": 55667788}

<|транскрипт|>
<... текстовая версия транскрипта опущена ...>

<|JSON|>
{"TaxID": 123456789, "MemberID": 87654321, "DateOfService":
"1 февраля 2024 года", "ClaimNumber": "111222333"}
```

Исходное описание задачи

One-shot-пример

Текстовая версия транскрипта чата

Исходный сигнал

Вывод модели

С помощью примера мы показали модели, что не нужно включать в вывод все данные. Мы также добились того, что модель перестала дублировать номер налогоплательщика в ID поставщика. Кроме того, JSON-объект теперь сжат в одну строку без переносов. Извлеченные данные точны, но, скорее всего, нам потребуется строгое совпадение имен ключей. Если приложение оператора ожидает поле с названием `ClaimID`, то в резюме оно не должно называться `ClaimNumber`.

В следующем листинге мы даем модели более подробный пример (выделен жирным).

Листинг 12.11. Обновленный one-shot-пример для экстрактивного резюме

```
<|инструкция|>
Прочитай следующий транскрипт диалога между
пользователем и автоматизированным помощником.
Извлеки все ID в формате JSON. #
```

Исходное описание задачи

```
<|транскрипт|>
ИИ: "Я автоматизированный помощник, чем я могу вам помочь?"
Пользователь: "Статус заявления".
ИИ: "Хорошо, статус заявления. Укажите ваш номер налогоплательщика".
Пользователь: "333444555".
ИИ: "Спасибо. Укажите восьмизначный идентификатор клиента".
Пользователь: "55667788".
ИИ: "Хорошо, дата рождения клиента?"
Пользователь: "19 апреля 2024 года".
ИИ: "Имя клиента – Джим Смит?"
Пользователь: "Да".
ИИ: "Отлично! Укажите дату подачи заявления".
Пользователь: "9 апреля 2024 года".
ИИ: "Я нашел ваше заявление №444444555 от 9 апреля 2024 года
для клиента №55667788 и поставщика №333444555. Возмещение
на сумму $156,81 было выплачено 23 апреля 2024 года."
Пользователь: "Спасибо! До свидания".

<|JSON|>
{"TaxID": 333444555, "MemberID": 55667788, "ClaimID": 444444555}
```

Обновленный one-shot-пример

```
<|транскрипт|>
<... текстовая версия транскрипта опущена ...>
```

Текстовая версия транскрипта чата

```
<|JSON|>
{"TaxID": 123456789, "MemberID": 87654321, "ClaimID":
"111222333"}
```

Исходный сигнал

Вывод модели

Обновление пошло на пользу, теперь мы получили только нужные нам ключи. Однако немного настораживает, что one-shot-пример должен быть очень похож на предоставляемый транскрипт. Это подразумевает, что для резюмирования других вариантов диалога нам, скорее всего, придется составлять отдельный пример для каждого из них. (А что, если диалог содержит ID авторизации, ID электронного платежа и др.?) Есть и еще один нюанс: значения `TaxID` и `MemberID` были числовыми, но для `ClaimID` модель выдала строковое значение даже после того, как в примере это поле было числовым.

Тестирование на наличие галлюцинаций

В наших резюме мы не сталкивались с галлюцинациями, но это не означает, что они невозможны. Любой промпт для резюмирования должен быть протестирован на множестве входных данных еще до развертывания, чтобы узнать, возникают ли галлюцинации. Чтобы их обнаружить после развертывания, необходимо проверить, что каждое извлеченное значение действительно присутствует в тексте транскрипта.

Вернемся к предыдущему one-shot-примеру и добавим к нему новые инструкции, выделенные жирным шрифтом в листинге ниже.

Листинг 12.12. Экстрактивное резюме с одним примером

```
<|инструкция|>
Прочитай следующий транскрипт диалога между пользователем и автоматизированным
помощником. Извлеки все ID в формате JSON.
Используй только следующие ключи в JSON:
"TaxID", "MemberID", "ClaimID".
Значения JSON должны быть числами, а не строками.

<|транскрипт|>
ИИ: "Я автоматизированный помощник, чем я могу вам помочь?"
Пользователь: "Статус заявления".
ИИ: "Хорошо, статус заявления. Укажите ваш номер налогоплательщика".
Пользователь: "333444555".
ИИ: "Спасибо. Укажите восьмизначный идентификатор клиента".
Пользователь: "55667788".
ИИ: "Хорошо, дата рождения клиента?"
Пользователь: "19 апреля 2024 года".

<|JSON|>
{"TaxID": 333444555, "MemberID": 55667788}

<|транскрипт|>
<... текстовая версия транскрипта опущена ...>

<|JSON|>
{"TaxID": 123456789, "MemberID": 87654321, "ClaimID":
111222333}
```

Дополненное описание задачи

Более короткий one-shot-пример

Текстовая версия транскрипта чата

Исходный сигнал

Вывод модели

Готово! Мы получили именно тот результат, который нам нужен. Нам удалось настроить LLM на генерацию структурированного резюме, используя комбинацию инструкций и примеров. При этом не потребовалось как-то изменять нашу систему — достаточно было лишь вызвать LLM. Этот вызов изолирован в компоненте, отвечающем за передачу диалога оператору.

Задания

1. Вернитесь к резюме, которое вы составили в разделе 12.1 для примера транскрипта чата. Используя LLM на ваш выбор (или модель, предпочитаемую в вашей компании), сгенерируйте текстовое резюме для того же транскрипта. Похож ли ответ LLM на ваше резюме? Использовали ли вы инструкции, примеры или оба подхода сразу?
2. Повторите задание 1, но измените параметры модели. В этой главе мы использовали жадное декодирование и штраф за повторение 1.1. Попробуйте декодирование с семплированием или измените значение штрафа — увеличьте

или уменьшите его. Удалось ли достичь лучших результатов? Соответствует ли вывод вашим ожиданиям?

3 Можно ли сгенерировать текстовое резюме и *одновременно* извлечь ключевые детали из транскрипта в одном промпте? Составьте промпт, который позволит получить следующий результат:

```
{"TaxID": 123456789, "MemberID": 87654321, "ClaimID": 111222333,
"Резюме": "Пользователь запрашивает оператора, потому что хочет
получить больше информации о своем страховом заявлении."}
```

4 Придумайте сценарий, включающий несколько запросов статуса заявления. Составьте пример диалога между ботом и пользователем, обращающимся к системе, чтобы проверить статус четырех заявлений. Некоторые заявления были удовлетворены, некоторые все еще находятся в обработке, а последнее — отклонено. Такой диалог будет содержать примерно в два-четыре раза больше данных, чем исходный пример. Как в таком случае должно выглядеть его резюме? Определив структуру желаемого резюме, попробуйте сгенерировать его с помощью LLM.

5 Составьте идеальное резюме для следующего диалога:

```
ИИ: "Я автоматизированный помощник, чем я могу вам помочь?"
Пользователь: "Страхование".
ИИ: "Хорошо, заявления. Укажите ваш номер налогоплательщика".
Пользователь: "333444555".
ИИ: "Спасибо. Укажите восьмизначный идентификатор клиента".
Пользователь: "55667788".
ИИ: "Хорошо, дата рождения клиента?"
Пользователь: "19 апреля 2024 года".
ИИ: "Имя клиента — Джейн Уильямс?"
Пользователь: "Да".
ИИ: "На данный момент клиент является участником программы страхования.
Выбранный план — PPO со страхованием на весь 2024 год. Желаете узнать
что-то еще?"
Пользователь: "Я хочу сходить к врачу, какая сумма возмещения мне
полагается?"
ИИ: "Поскольку вы входите в сеть поставщиков, сумма возмещения
составит $20".
Пользователь: "Нужно ли получить предварительное одобрение для
посещения врача?"
ИИ: "Да, хотите зарегистрировать его сейчас?"
Пользователь: "Да".
ИИ: "Хорошо, я переведу вас к специалисту, который вам с этим
поможет".
```

Теперь составьте промпт для LLM, чтобы сгенерировать аналогичное резюме.

6 Протестируйте новый промпт на транскрипте из листинга 12.1. При необходимости уточните запрос, чтобы получить качественные резюме для обоих диалогов.

Итоги

- Передача диалогов операторам — неизбежная часть многих систем разговорного ИИ. Операторам необходимо предоставлять краткие резюме, содержащие ключевые детали диалога — как в текстовой, так и в структурированной форме.
- Для создания резюме необходим транскрипт диалога. Большинство платформ разговорного ИИ создают их автоматически, и вы можете настроить ИИ для генерации транскрипта в нужном вам формате.
- Структурированные резюме можно получить, расширив возможности своего разговорного ИИ, чтобы система сохраняла важные данные по мере их сбора либо извлекала их с помощью LLM уже после завершения диалога.
- Необходимо понимать, как выглядит качественное резюме, прежде чем поручать его создание LLM.
- LLM способны генерировать текстовые резюме и извлекать ключевые данные из транскриптов. Чтобы получить желаемое резюме, понадобятся лишь четкие инструкции и примеры.

Эндрю Фрид, Кари Джейкобс, Энико Рожа

Эффективный разговорный ИИ. Создаем чат-ботов, которые действительно работают

Перевела с английского В. Загребина
Научный редактор А. Петраки

Руководитель дивизиона	*Ю. Сергиенко*
Руководитель проекта	*А. Питиримов*
Ведущий редактор	*Е. Строганова*
Литературный редактор	*Ю. Широнина*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, Н. Викторова*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 05.2026. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 27.03.26. Формат 70×100/16. Бумага офсетная. Усл. п. л. 29,670. Тираж 700. Заказ 0000.

Read IT Club

Комьюнити рецензентов и переводчиков ИТ-литературы

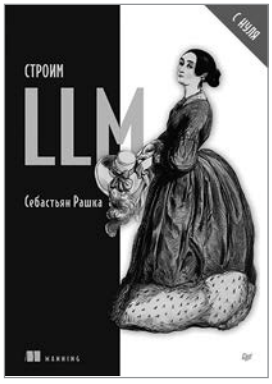
Миссия участников клуба — обеспечить высокое качество профессиональной переводной литературы на русском языке. «Книжные дебагеры» проверяют корректность терминологии и подписей на схемах и иллюстрациях, чтобы сделать книги более понятными русскоязычному читателю. Стать участником Read IT Club может любой ИТ-специалист, готовый поделиться опытом с сообществом.

присоединиться к нам


ВОСПОЛЬЗУЙТЕСЬ
ВОЗМОЖНОСТЬЮ
ПРИОБРЕСТИ КНИГИ
НА САЙТЕ ИЗДАТЕЛЬСТВА
piter.com
со скидкой
по промокоду
20%
PITER

Себастьян Рашка

СТРОИМ LLM С НУЛЯ



Говорят, что физик Ричард Фейнман однажды сказал: «Я не понимаю того, что не могу создать». Основываясь на этом же важном принципе, автор бестселлеров Себастьян Рашка шаг за шагом ведет вас к созданию LLM в стиле GPT, которую вы сможете запустить на своем ноутбуке. Это увлекательная книга, которая охватывает каждый этап процесса — от планирования и кодирования до обучения и тонкой настройки.

«Строим LLM с нуля» — это чрезвычайно занимательное путешествие в основы генеративного ИИ. Не полагаясь на существующие библиотеки LLM, вы реализуете в коде базовую модель, превратите ее в классификатор текста и в конечном счете создадите чат-бот, который сможет следовать вашим инструкциям в диалоге. И вы действительно поймете LLM, потому что создали ее сами!

КУПИТЬ