

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Институт компьютерных и инженерных наук
Кафедра информационных и управляющих систем
Направление подготовки 09.04.04 – Программная инженерия
Направленность (профиль) образовательной программы Управление разработкой программного обеспечения

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой
_____ А.В. Бушманов
«__» _____ 2025 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему: Разработка telegram чат-бота с использованием нейросетей

Исполнитель студент группы 3105-ом1	_____	Л.И. Ткаченко
	(подпись, дата)	
Руководитель доцент, канд. физ-мат. наук	_____	В.В. Ерёмина
	(подпись, дата)	
Руководитель научного содержания программы магистратуры профессор, доктор техн. наук	_____	И.Е. Еремин
	(подпись, дата)	
Нормоконтроль инженер	_____	В.Н. Адаменко
	(подпись, дата)	
Рецензент доцент, канд. техн. наук	_____	С.Г. Самохвалова
	(подпись, дата)	

Благовещенск 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Институт компьютерных и инженерных наук
Кафедра информационных и управляющих систем

УТВЕРЖДАЮ

Зав. кафедрой

_____ А.В. Бушманов

«___» _____ 2025 г.

З А Д А Н И Е

К магистерской диссертации студента группы 3105-ом1

Ткаченко Леонида Ивановича

1. Тема магистерской диссертации: Разработка telegram чат-бота с использованием нейросетей

(Утверждено приказом от 06.03.2025 № 609-уч)

2. Срок сдачи студентом законченной работы (проекта): 10.06.2025

3. Исходные данные к магистерской диссертации: научные публикации, документация по библиотекам Python, учебная литература

4. Содержание магистерской диссертации (перечень подлежащих разработке вопросов): анализ современных подходов, проектирование архитектуры системы, разработка и обучение нейросетевой модели, интеграция с Telegram

5. Перечень материалов приложения (наличие чертежей, таблиц, графиков, схем, программных продуктов, иллюстративного материала и т.п.): _____

6. Рецензент магистерской диссертации: Самохвалова С.Г.,
доцент, канд. техн. наук

7. Дата выдачи задания 29.01.2025

Руководитель выпускной квалификационной работы: В.В. Еремина,
доцент, канд. физ-мат. наук

(фамилия, имя, отчество, должность, уч.степень, уч.звание)

Заявление принял к исполнению _____

РЕФЕРАТ

Магистерская диссертация содержит 88 страниц, 17 рисунок, 3 таблицы, 50 источников

РАЗРАБОТКА TELEGRAM-БОТА С ИСПОЛЬЗОВАНИЕМ НЕЙРОСЕТЕЙ

Целью данной выпускной квалификационной работы является разработка интеллектуального Telegram-бота, взаимодействующего с нейросетевой моделью для обработки пользовательских запросов в режиме диалога. Исследование направлено на построение эффективной архитектуры, обеспечивающей возможность интеллектуальной генерации ответов на естественном языке.

В рамках выполнения работы были решены следующие основные задачи:

- проведено исследование роли мессенджеров в современных коммуникационных системах, особенностей интеграции Telegram-ботов и требований к интерактивному взаимодействию с пользователями;
- изучены современные подходы к генерации текстов на основе нейросетевых моделей, включая архитектуры трансформеров и особенности работы API YandexGPT;
- разработана архитектура программного комплекса, обеспечивающего передачу пользовательского ввода, формирование запроса к модели, обработку ответа и обратную отправку пользователю через Telegram-бота;
- реализованы механизмы безопасности и устойчивости, в том числе контроль доступа (whitelist), обработка ошибок и сохранение истории диалога.

Разработка и внедрение нейросетевых моделей в структуру Telegram-бота позволяет создать удобный и функциональный инструмент для взаимодействия с пользователями. Полученные в ходе исследования результаты могут быть применены для создания персональных помощников, образовательных систем, корпоративных чат-ботов и других решений.

СОДЕРЖАНИЕ

Введение	6
1 Анализ коммерческой аналитики и методов прогнозирования спроса	8
1.1 Анализ предметной области	8
1.2 Объект исследования	10
1.3 Актуальность проводимого исследования	11
1.4 Существующие методы решения	15
2 Алгоритмическое решение исследуемой задачи и выбор программно-технической платформы	17
2.1 Алгоритмическая модель программного решения Telegram-бота на базе нейросетей	17
2.2 Профильное программное обеспечение для реализации нейросетевого чат-бота	24
2.3 Обоснование выбора программного обеспечения	31
2.3.1 Критерии выбора ПО	31
2.3.2 Анализ альтернатив	32
2.3.3 Описание выбранных инструментов	33
3 Программная реализация предлагаемого алгоритма решения задачи	40
3.1 Основные этапы практической разработки программного продукта	40
3.1.1 Модель жизненного цикла	40
3.1.2 Этапы разработки Telegram-бота	42
3.1.3 Архитектура разработанного решения	48
3.1.4 Выбор инструментов и технологий	51
3.1.5 Функциональные требования к системе	55
3.2 Практическая реализация информационной системы	59
3.2.1 Структура и архитектура программного кода	59
3.2.2 Системный промпт для ограничения области вопросов	62

3.2.3 Обработка команд и callback-запросов	64
3.2.4 Механизм авторизации пользователя	67
3.2.5 Обработка текстовых сообщений и ведение истории диалога	68
3.2.6 Вызов API YandexGPT с передачей контекста и системного промпта	73
3.2.7 Пример работы чат-бота и интерфейс пользователя	76
Заключение	82
Библиографический список	84

ВВЕДЕНИЕ

Современный этап развития информационного общества характеризуется стремительным нарастанием объема доступных данных, среди которых огромную долю составляют текстовые данные. Развитие интернета, социальных сетей, электронных библиотек, а также появление обширных корпоративных баз знаний привели к ситуации, при которой аналитики, исследователи, эксперты, научные работники и бизнес-аналитики оказываются перед необходимостью решения нетривиальных задач по автоматическому извлечению знаний из текстовой информации.

Однако повышение доступности и многообразия текстовых данных приводит к новым вызовам. При традиционных подходах комплексный анализ текстовых корпусов зачастую превращается в сложную, трудоёмкую и практически невыполнимую задачу при больших масштабах данных. Ручная обработка текстовой информации, особенно на естественных языках, быстро становится узким местом, замедляющим извлечение полезных знаний. В результате возникает острая потребность в автоматизации процессов анализа, систематизации и интерпретации текстовой информации, что стимулирует активное развитие прикладных и теоретических решений в области интеллектуального анализа текстов.

При этом объем и разнородность текстовых данных столь велики, что ручной анализ становится либо крайне трудоёмким, либо практически невозможным. В ответ на эту проблему сформировалась целая область исследований и решений, объединенных под общим знаменателем «интеллектуальный анализ текстов» (Text Mining, Natural Language Processing, Semantic Analysis). В число ключевых направлений данной области входят задачи тематического моделирования, извлечения ключевых слов, выделения именованных сущностей, анализа тональности, извлечения фактов, автоматической категоризации и кластеризации текстов, а также их дальнейшей визуализации, понимания и интеграции с иными системами.

Особую актуальность в наше время приобретают программные комплексы, интегрирующие в себе широкий спектр инструментов для анализа текстов. Такого рода решения становятся своеобразными «конструкторами» аналитики: они должны быть гибкими, модульными, легко расширяемыми, иметь понятную и доступную архитектуру для внедрения новых методов и подходов. Кроме того, они должны обладать удобными интерфейсами для конечных пользователей – аналитиков, исследователей и специалистов по обработке данных – предоставляя им средства визуализации, редактирования и настроек параметров анализа, а также интеграции с внешними информационными системами.

Настоящее исследование посвящено разработке концептуальной архитектуры, определению предметной области, формулировке требований к программному комплексу для комплексного анализа текстов. Мы рассмотрим существующие решения, определим объект исследования, проанализируем инструментарий и подходы к реализации. Кроме того, будет уделено внимание вопросам прав доступа, масштабируемости, макро- и микропроектированию такого ПО, а также практическим примерам его использования.

1 ПРЕДМЕТНАЯ ОБЛАСТЬ ПРОВОДИМОГО ИССЛЕДОВАНИЯ

1.1 Анализ предметной области

Современные информационные технологии и значительные объемы данных, циркулирующие в глобальных сетях, привели к возникновению масштабных текстовых массивов, нуждающихся в систематическом анализе. Ключевой тенденцией последних лет стало использование инструментов для автоматизированного семантического анализа, категоризации и извлечения скрытых закономерностей из текстовых данных. Предметная область, лежащая в основе данного исследования, охватывает методы интеллектуальной обработки естественного языка (NLP – Natural Language Processing), машинного обучения и информационного поиска, применяемые к большому количеству текстов.

Актуальность изучаемой области обусловлена стремительным ростом различных текстовых ресурсов: научные статьи, обзоры, медиаконтент, социальные сети, корпоративные базы знаний, техническая документация. Использование инструментов NLP (обработки естественного языка) позволяет не только автоматизировать рутинные задачи (классификация документов, извлечение ключевых слов, определение тональности отзывов), но и сформировать основу для глубокой аналитики (выделение скрытых тем, построение смысловых графов, выявление тенденций в больших корпусах текстов).

Другой важный аспект вышеописанной предметной области – это потребность в разработке удобных, масштабируемых и функциональных программных решений. Такие решения должны быть способны работать с разноплановыми данными, учитывать морфологические, синтаксические и семантические особенности языка, а также предоставлять исследователям и аналитикам инструменты для визуализации, интерактивной фильтрации, сложных запросов и интеграции с внешними системами.

Дополнительно следует отметить, что совершенствование подобной системы возможно не только на уровне алгоритмов, но и с точки зрения удобства

для конечного пользователя. В частности, встаёт вопрос о создании интуитивно понятных интерфейсов, позволяющих специалистам без глубоких знаний в области машинного обучения и вычислительной лингвистики запускать сложные аналитические процессы и получать результаты в удобном виде. Уже на ранних этапах проектирования комплексного решения необходимо уделять особое внимание вопросам UX/UI, а также средствам визуализации, которые упрощают восприятие выявленных закономерностей и трендов.

Важным направлением развития остаётся распределённая обработка, особенно при работе с гигантскими корпусами текстовых данных, собранными из корпоративных баз, социальных сетей и научных публикаций. Своевременная интеграция облачных решений и специализированных платформ, поддерживающих параллельные вычисления, делает возможным одновременную реализацию нескольких аналитических сценариев, снижая при этом общую нагрузку на инфраструктуру. Данный аспект особенно актуален для организаций, где анализ текстовых материалов связан с необходимостью оперативного реагирования на события в режиме реального времени, будь то мониторинг обратной связи пользователей или отслеживание ключевых тенденций на рынке.

Важным элементом архитектуры, на который обращают внимание исследователи и разработчики, является система прав доступа и разграничения ролей. Грамотная политика безопасности обеспечивает не только сохранность конфиденциальных данных, но и гибкие возможности масштабирования команды, в которой разные специалисты могут работать над своими задачами, не мешая друг другу. Например, аналитики могут иметь доступ лишь к результатам и настройкам предобработки, в то время как системные администраторы наделяются расширенными правами для настройки и обновления инфраструктуры.

Эволюция комплекса для интеллектуального анализа текстовых данных направлена не только на интеграцию новых алгоритмов, но и на систематическую проработку организационных, технических и интерфейсных аспектов. В конечном счёте такое целостное решение способно трансформировать

накопленные текстовые ресурсы в действенные знания, повышая конкурентоспособность бизнеса и способствуя прогрессу научных исследований, а открытость архитектуры и модульный принцип будут стимулировать дальнейшие инновации и эксперименты в области обработки естественного языка.

1.2 Объект исследования

Объектом исследования в данном случае является программный комплекс, предназначенный для автоматизированного тематического и семантического анализа больших текстовых корпусов. Этот комплекс должен включать в себя следующие ключевые компоненты:

- Модуль загрузки и предобработки данных: обеспечивает импорт данных из различных источников (локальных файлов, баз данных, веб-сервисов), проведение очистки текстов от служебных символов, нормализацию и лемматизацию.

- Модуль извлечения признаков и семантического анализа: выполняет выделение ключевых слов, автоматическую категоризацию, кластеризацию текстов, построение тематических моделей (LDA, NMF, тематическая сегментация) и анализ тональности.

- Модуль визуализации и интерпретации результатов: предоставляет интерфейсы для наглядного представления тематических карт, графов связей между концептами, диаграмм соотношения тематических кластеров.

- Модуль интеграции и расширяемости: позволяет подключать новые модели и библиотеки, интегрироваться с внешними системами аналитики и бизнес-логики.

Основная цель исследования – разработать и обосновать архитектуру такого программного комплекса, а также предложить методологию и инструментарий для его эффективной реализации. При этом особое внимание уделяется интеграции нейросетевых моделей обработки естественного языка в структуру Telegram чат-бота, что позволяет обеспечить интеллектуальную интерпретацию

пользовательских сообщений, адаптивное ведение диалога и автоматизированный анализ текстовых данных в реальном времени.

На нижеописанном рисунке 1 представлена общая концептуальная модель, детальная архитектура и специфика взаимодействия модулей будут рассмотрены в дальнейших главах.



Рисунок 1 – Упрощенная модель объектной области программного комплекса

1.3 Актуальность проводимого исследования

Современный этап развития информационного общества сопровождается беспрецедентным ростом объёмов текстовых данных, формируемых в научной, социальной, деловой и медийной сферах. Источниками таких данных являются социальные сети, корпоративные базы знаний, научные публикации, электронные библиотеки, форумы и новостные порталы. По мере увеличения объёмов информации возникает необходимость её систематизации, анализа и интерпретации. Однако традиционные методы обработки текстовых данных с применением ручного труда и классических статистических методов становятся крайне

неэффективными. Это приводит к появлению острой потребности в интеллектуальных программных системах.

Актуальность проводимого исследования обусловлена глобальными тенденциями цифровизации, в результате которых компании и исследовательские организации сталкиваются с постоянным приростом текстовой информации. Ручная обработка таких данных невозможна в силу их объёма, что приводит к необходимости внедрения технологий обработки естественного языка (NLP – Natural Language Processing) и методов машинного обучения. В частности, интеллектуальный анализ текстов (Text Mining) позволяет автоматизировать такие задачи, как выявление скрытых закономерностей, анализ тональности, выделение ключевых тем и категоризация документов. Это особенно важно в условиях современного бизнеса, где своевременное выявление трендов и анализ обратной связи от клиентов напрямую влияет на эффективность управленческих решений.

С ростом числа цифровых платформ и социальных сетей анализ текстовой информации становится критически важным инструментом для мониторинга общественного мнения, выявления фейковых новостей и борьбы с дезинформацией. Компании используют интеллектуальный анализ текстов для изучения предпочтений потребителей, обработки отзывов и формирования маркетинговых стратегий. В научной сфере автоматизированный анализ текстов позволяет исследователям эффективно обрабатывать большие массивы публикаций, выявлять научные тренды и находить релевантные источники для своих исследований.

Несмотря на широкий выбор программных средств и библиотек для анализа текстов (таких как NLTK, SpaCy, Gensim, HuggingFace Transformers), существующие решения часто имеют ряд ограничений:

- Фрагментарность функционала: многие инструменты ориентированы на выполнение отдельных задач (например, только тематическое моделирование или только анализ тональности), что требует дополнительных усилий по их интеграции.

- Ограниченная масштабируемость: некоторые библиотеки плохо справляются с анализом больших корпусов текстов, что особенно актуально для корпоративного уровня.

- Сложности интеграции: отсутствие единых стандартов интерфейсов затрудняет объединение различных моделей и модулей в единый программный комплекс.

- Высокие требования к ресурсам: современные модели на основе трансформеров (например, BERT, GPT) требуют значительных вычислительных мощностей, что повышает порог входа для малых и средних организаций.

Таким образом, возникает необходимость создания комплексного программного решения, которое будет не только интегрировать существующие подходы и инструменты, но и обеспечивать их удобную реализацию, расширяемость и гибкость использования для различных практических задач.

С научной точки зрения актуальность работы заключается в разработке методологии проектирования архитектуры программного комплекса, интегрирующего современные подходы к анализу текстов на естественном языке. Исследование включает не только теоретический анализ существующих методов (тематическое моделирование, выделение сущностей, анализ тональности, использование трансформеров), но и их практическую реализацию в рамках модульной архитектуры программного комплекса. Особое внимание уделяется вопросам масштабируемости, гибкости и унифицированности интерфейсов для подключения новых моделей и инструментов.

Современный рынок информационных технологий уже предлагает ряд решений, призванных упростить обработку и анализ текстовых данных, однако отсутствие единого стандарта и несовместимость некоторых библиотек приводят к дополнительным издержкам при построении комплексных систем. Одновременно с этим, потребности бизнеса и научной сферы стимулируют создание инструментов, позволяющих быстро адаптироваться к специфике конкретной

предметной области и оперировать большими массивами данных без ущерба для скорости и точности анализа.

Кроме того, в последнее время возрастает внимание к технологиям распределённой обработки, которая даёт возможность параллельно обрабатывать большие массивы текстов, существенно сокращая время получения результатов и повышая пропускную способность системы. Такое масштабирование позволяет системам интеллектуального анализа текстов охватывать огромные объёмы информационных ресурсов, будь то база документов крупной корпорации, поток сообщений в социальных сетях или коллекция научных статей в определённой области знаний.

Актуальным направлением остаётся также разработка пользовательских интерфейсов и инструментов визуализации, позволяющих специалистам без значительных знаний в области машинного обучения эффективно работать с системой. Именно такие средства повышают доступность аналитических возможностей и существенно упрощают восприятие сложных метрик и моделей.

Таким образом, исследование комплексной методологии для создания программного комплекса, сочетающего современные NLP-технологии, распределённые вычислительные ресурсы и удобные пользовательские инструменты, представляется своевременным и востребованным. Разработка подобной архитектуры позволит обеспечить гибкость, масштабируемость и единые стандарты интеграции, что в конечном счёте даст возможность организациям и исследователям существенно повысить качество и скорость извлечения знаний из больших массивов текстовой информации.

Одним из приоритетных аспектов в развитии подобных систем является расширение их функциональных возможностей за счёт применения методов адаптации к конкретным предметным областям. Использование специализированных корпусов текстов и соответствующих лексиконо-семантических ресурсов позволяет повысить точность и релевантность результатов, учитывая узкоспециализированную терминологию и концепты. Такие подходы особенно

востребованы в медицине, праве, финансовой аналитике и других высокоспециализированных сферах, где общие NLP-модели могут давать неточные результаты из-за контекстных нюансов и специфических понятий, отсутствующих в общем языковом наборе данных. Кроме того, адаптация моделей под конкретную предметную область способствует улучшению качества взаимодействия с пользователем, так как позволяет чат-ботам и другим интеллектуальным системам предоставлять более релевантные и содержательные ответы, учитывающие специфику терминологии и логику отраслевых процессов.

1.4 Существующие решения

В настоящий момент на рынке и в научном сообществе существуют разнообразные инструменты и платформы для анализа текстовых данных. Рассмотрим некоторые из них:

- NLTK (Natural Language Toolkit) – широко используемая библиотека на языке Python, содержащая базовые инструменты для работы с естественным языком: токенизация, стемминг, лемматизация, POS-теггинг (Part-of-Speech tagging), выделение именованных сущностей. Преимущества: обширная документация, большое сообщество, богатый набор классических NLP-инструментов. Недостатки: недостаток встроенных современных трансформерных моделей, сложность интеграции с крупными системами, ограниченная масштабируемость.

- SpaCy – высокопроизводительная библиотека NLP на Python, ориентированная на скорость и современную архитектуру. Поддерживает быстрый POS-теггинг, выделение именованных сущностей, синтаксический разбор. Преимущества: высокая скорость, удобный API, поддержка языковых моделей для различных языков. Недостатки: ограниченный набор готовых модулей для тематического анализа, тональность не выделена как ключевая функциональность, сложное подключение дополнительных алгоритмов.

- Gensim – библиотека для тематического моделирования и семантического анализа текстов. Предоставляет реализацию LDA (Latent Dirichlet

Allocation), Word2Vec и других моделей. Преимущества: ориентация на тематическое моделирование, эффективная работа с большими корпусами. Недостатки: узкая специализация на тематическом анализе, отсутствие комплексного подхода (нет встроенных инструментов для визуализации, ограниченные средства предварительной обработки).

- HuggingFace Transformers – набор современных библиотек и моделей для продвинутой обработки естественного языка, включающий BERT, GPT и другие трансформерные архитектуры. Преимущества: доступ к новейшим достижениям в NLP, простота интеграции предобученных моделей. Недостатки: высокая вычислительная сложность, требования к аппаратным ресурсам, необходимость дополнительной инфраструктуры для визуализации и анализа результатов.

- Коммерческие платформы и облачные сервисы (Google Cloud Natural Language, AWS Comprehend, Azure Cognitive Services) – предоставляют разнообразные инструменты для анализа текстов, интегрируемые с другими сервисами. Преимущества: масштабируемость, техническая поддержка, удобство развертывания в облаке. Недостатки: зависимость от внешних провайдеров, возможные ограничения по кастомизации, часто высокая стоимость.

На рынке уже присутствует множество решений, но большинство из них ориентировано либо на определённый узкий аспект анализа (например, только тематическое моделирование, извлечение именованных сущностей или классификация текстов), либо требует значительных усилий по интеграции разнородных компонентов, что усложняет их практическое применение в реальных проектах. Кроме того, сложным остается вопрос унификации интерфейсов, масштабируемости, обеспечения прав доступа и гибкости адаптации инструментов под конкретные нужды пользователя. Большинство готовых решений представляют собой либо облачные сервисы с ограниченными возможностями кастомизации, либо библиотеки, требующие существенных затрат на разработку собственной

инфраструктуры, особенно в части поддержки диалоговой логики, авторизации, логирования, мониторинга и обучения моделей.

Также немногие из существующих платформ предлагают нативную интеграцию с мессенджерами, такими как Telegram, что требует дополнительных работок для адаптации под API конкретного канала коммуникации.

На рисунке 2 показано, что исследуемый комплекс должен выступать в роли центрального звена, консолидирующего возможности различных библиотек и сервисов, и предоставлять единый удобный интерфейс для пользователя.

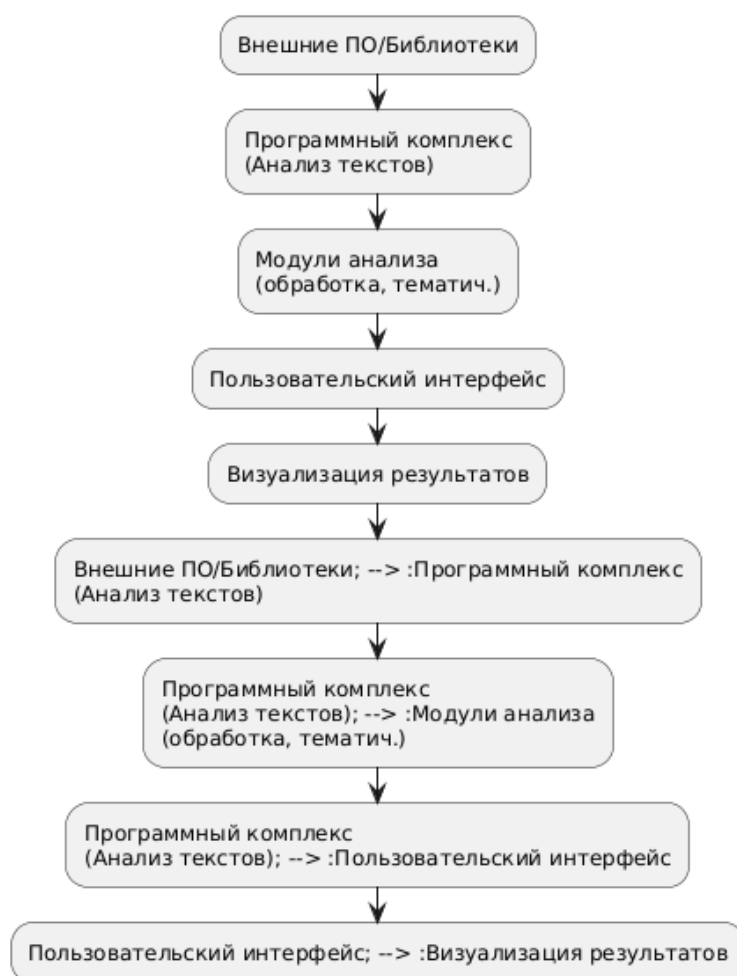


Рисунок 2 – Пример интеграции внешних библиотек для анализа текстов

Таким образом, предметная область исследования – это разработка интегрированного ПО для интеллектуального анализа текстовых данных, обеспечивающего функциональность от предобработки до визуализации и интеграции.

Объект исследования – многофункциональный программный комплекс, сочетающий в себе возможности тематического моделирования, тонального анализа, выделения сущностей и других инструментов NLP. Существующие решения дают нам богатый набор инструментов, но требуют создания концептуально новой архитектуры, ориентированной на модульность, расширяемость и удобство использования.

Существующие на сегодняшний день решения предлагают широкий выбор специализированных инструментов и библиотек, однако они зачастую изолированы друг от друга и не обеспечивают целостного, удобного в использовании подхода. Это обуславливает необходимость разработки новой архитектуры, основанной на принципах модульности, масштабируемости и открытости, что позволит адаптировать программный комплекс под конкретные задачи аналитики в различных предметных областях — от маркетинга и образования до юриспруденции и государственного управления.

Разрабатываемая система должна учитывать современные вызовы цифровой обработки информации, обеспечивать устойчивую работу с разноформатными источниками текстов, а также предусматривать возможности для интеграции с внешними API и базами данных. Такой подход способствует созданию универсального и практико-ориентированного инструмента, востребованного как в научной среде, так и в прикладных проектах.

2 АЛГОРИТМИЧЕСКОЕ РЕШЕНИЕ ИССЛЕДУЕМОЙ ЗАДАЧИ И ВЫБОР ПРОГРАММНО-ТЕХНИЧЕСКОЙ ПЛАТФОРМЫ

2.1. Алгоритмическая модель программного решения Telegram-бота на базе нейросетей

Разработка интегрированного решения для взаимодействия с нейросетевой моделью посредством Telegram-бота базируется на всестороннем анализе функциональных требований и архитектурных особенностей системы. Представленная ниже структура демонстрирует, как концептуальные решения переходят в конкретную реализацию, обеспечивая модульность, масштабируемость и устойчивость разрабатываемого продукта.

Прежде всего, для наглядного отображения ключевых функциональных сценариев используется диаграмма прецедентов. Она иллюстрирует распределение ролей и сценариев взаимодействия между пользователями, администраторами и интеграционными модулями системы. В рамках этой модели выделяются основные участники: конечный пользователь, Telegram-бот как интерфейсный посредник, нейросетевая модель как интеллектуальный обработчик запросов, а также администратор системы, ответственный за управление, настройку и мониторинг.

Среди ключевых прецедентов со стороны пользователя можно выделить: отправку текстового запроса, получение ответа на основе семантической интерпретации, запрос справочной информации, участие в интерактивных сценариях (например, анкетирование, подбор рекомендаций), а также отправку обратной связи. Администратор системы, в свою очередь, имеет доступ к функциям настройки параметров нейросетевой модели, обновления обучающих данных, просмотра логов, настройки прав доступа и контроля качества обслуживания.

Как показано на рисунке 3, система предусматривает обработку запросов, регистрацию действий, настройку параметров и управление доступом, что формирует основу функциональной модели решения.

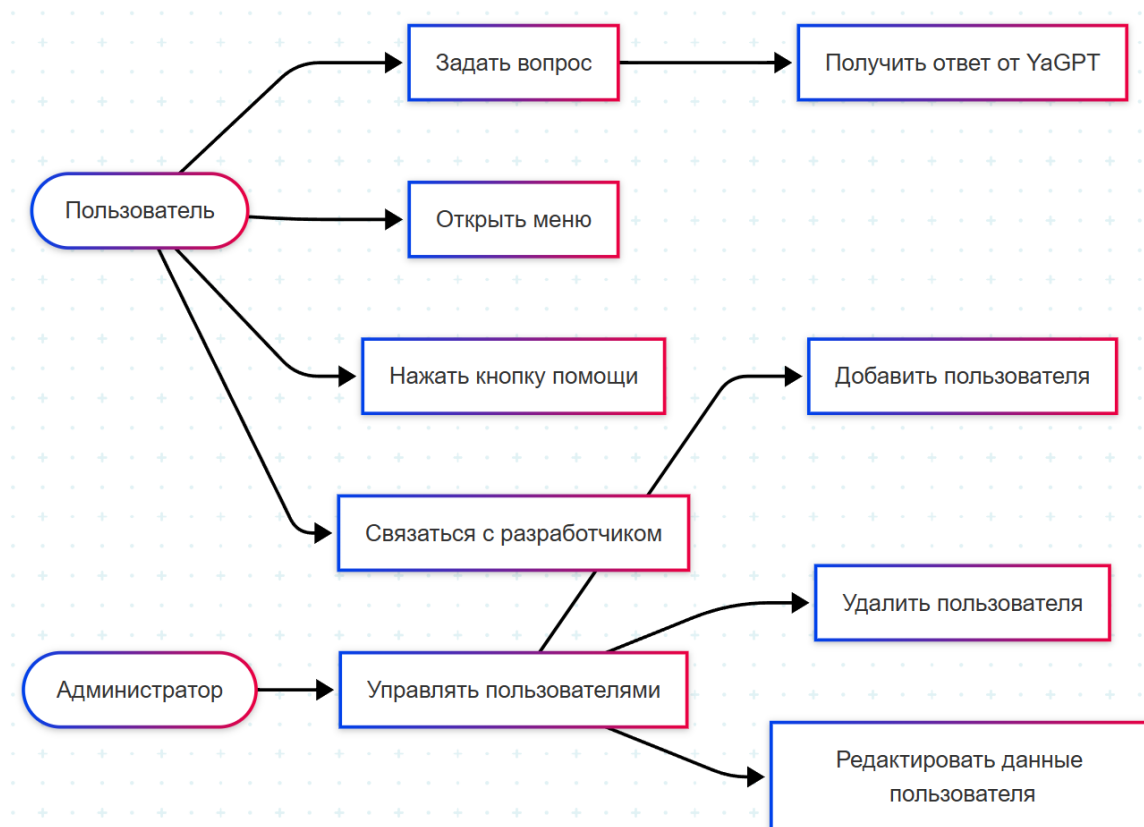


Рисунок 3 – Диаграмма прецедентов

Функциональные сценарии системы:

- обработка запросов – бот должен принимать текстовые запросы от пользователя и генерировать ответ на основе нейросетевой модели;
- обратная связь – бот должен отправлять уведомления о статусе обработки запроса;
- обучение модели – администратор должен иметь возможность запускать процесс обучения нейросети через бот;
- регистрация действий пользователя – система должна записывать все взаимодействия пользователей для анализа;
- настройка параметров взаимодействия – Пользователь должен иметь возможность изменять настройки взаимодействия с ботом (например, язык общения, частоту уведомлений);
- обновление нейросети – система должна предоставлять возможность администратору загружать обновления для нейросетевой модели;

- управление доступом – система должна поддерживать различные уровни доступа для пользователей;
- мониторинг работы нейросети – администратор должен иметь возможность получать отчеты о работе нейросети.

Далее представлена диаграмма последовательностей, отражающая динамику обработки входящего запроса. Этот графический материал демонстрирует порядок передачи данных между компонентами системы – от первичного приема запроса до формирования и отправки ответа пользователю.

Рисунок 4 наглядно демонстрирует, как последовательное взаимодействие между модулями обеспечивает корректную обработку запроса и синхронную работу системы. В частности, диаграмма иллюстрирует этапы, начиная с первичной проверки входящего сообщения, его валидации и записи в лог, что позволяет гарантировать целостность данных на начальном этапе обработки. Такой подход способствует снижению вероятности возникновения ошибок на последующих этапах.

После первичной обработки запрос поступает в модуль интеграции с внешней языковой моделью, где происходит формирование запроса на основе сохраненного контекста диалога. На данном этапе особое внимание уделяется корректной упаковке данных и настройке параметров генерации, что критически важно для получения релевантного ответа от нейросетевой модели. Диаграмма последовательностей демонстрирует, как информация переходит от одного модуля к другому, обеспечивая прозрачность и предсказуемость работы системы.

Заключительный этап, представленный на диаграмме, описывает процесс формирования итогового ответа и его передачу обратно пользователю. Этот этап включает не только получение результата от нейросетевой модели, но и выполнение дополнительных проверок, обеспечивающих корректное отображение информации в клиентском приложении. Такая синхронизация между модулями позволяет системе оперативно реагировать на изменения в запросах и адаптировать выводимый ответ к текущему контексту диалога.

Таким образом, представленная диаграмма последовательностей служит не только визуальным подтверждением логики работы системы, но и инструментом для анализа и оптимизации процесса обработки запросов, выявляя критические участки и обеспечивая высокую надежность конечного решения.

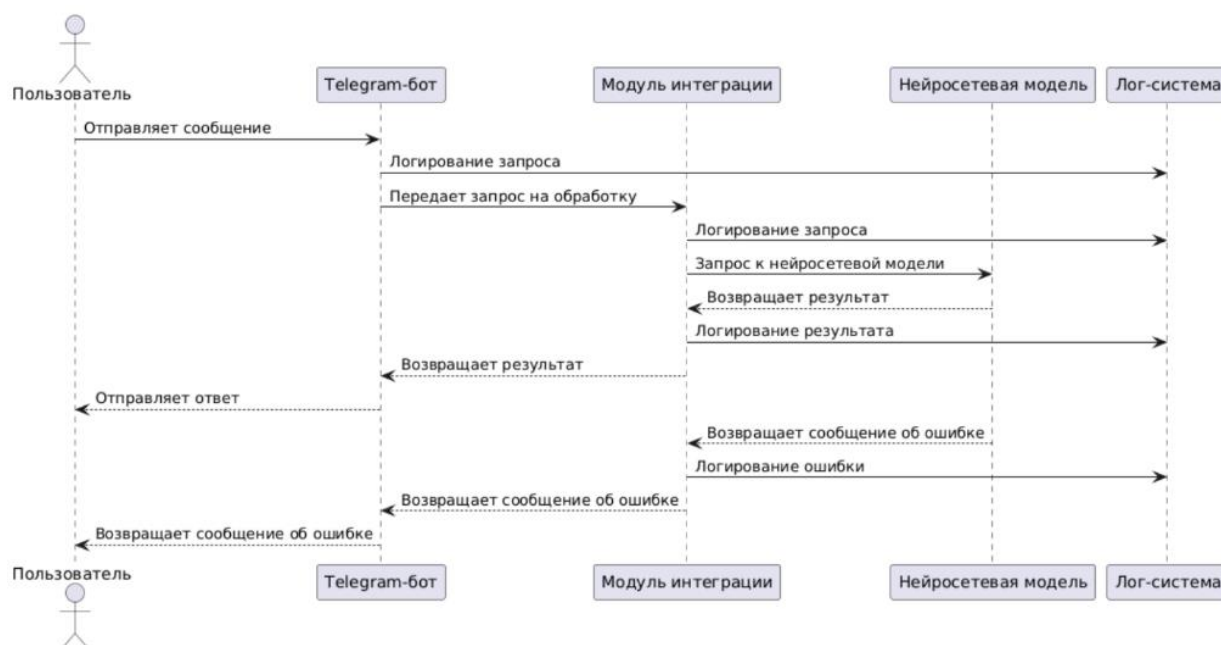


Рисунок 4 – Диаграмма последовательностей

Чтобы более детально отразить внутреннюю логику переходов между состояниями системы, необходимо рассмотреть динамику состояний в процессе обработки запроса, включая последовательность активации компонентов, изменение контекста и условия завершения каждого этапа.

Рисунок 5 демонстрирует, как система переходит от состояния ожидания запроса к состоянию обработки данных, а затем – к формированию и отправке ответа или обработке ошибок. Также представленная ниже диаграмма описывает последовательные состояния нейросетевой модели в процессе обработки и генерации ответа на запрос, поступивший через модуль интеграции.

Нижеописанная диаграмма имеет следующую структуру:

- Начальное состояние – модель ожидает получения запроса от модуля интеграции. На этом этапе модель готова к принятию новых данных и запуску дальнейшей обработки, но пока не выполняет вычислительных операций.
- Состояние "Получение запроса" – модель получает запрос на обработку. На данном этапе запрос анализируется, извлекаются необходимые параметры и определяется, какие именно операции предстоит выполнить.
- Состояние "Обработка данных" – выполняется основная обработка данных с привлечением нейросетевых алгоритмов. Модель применяет соответствующие архитектуры и алгоритмы машинного обучения для извлечения признаков.
- Состояние "Формирование результата" – модель подготавливает ответ в соответствии с запросом, используя результаты вычислений с предыдущего шага. На этом этапе формируется итоговый текст, числовые показатели или иные данные, необходимые для ответа.
- Состояние "Отправка результата" – модель отправляет сформированный результат обратно в модуль интеграции. Этот модуль затем передаёт ответ конечному пользователю или другой системе.
- Состояние "Ошибка обработки" – если в процессе обработки данных возникают ошибки (например, некорректный формат входных данных, сбой в вычислительных ресурсах или внутренние ошибки модели), модель переходит в это состояние.
- Конечное состояние – завершение работы с запросом. На этом этапе модель готова принять новый запрос и снова перейти в начальное состояние ожидания.

Таким образом, цикл обработки запроса формирует замкнутую и логически завершённую структуру взаимодействия, где конечное состояние служит не просто завершением диалога, а точкой готовности к следующему циклу коммуникации. Такая архитектура обеспечивает стабильную работу системы, гибкость

диалогов и возможность масштабирования сценариев взаимодействия без потери целостности пользовательского опыта.

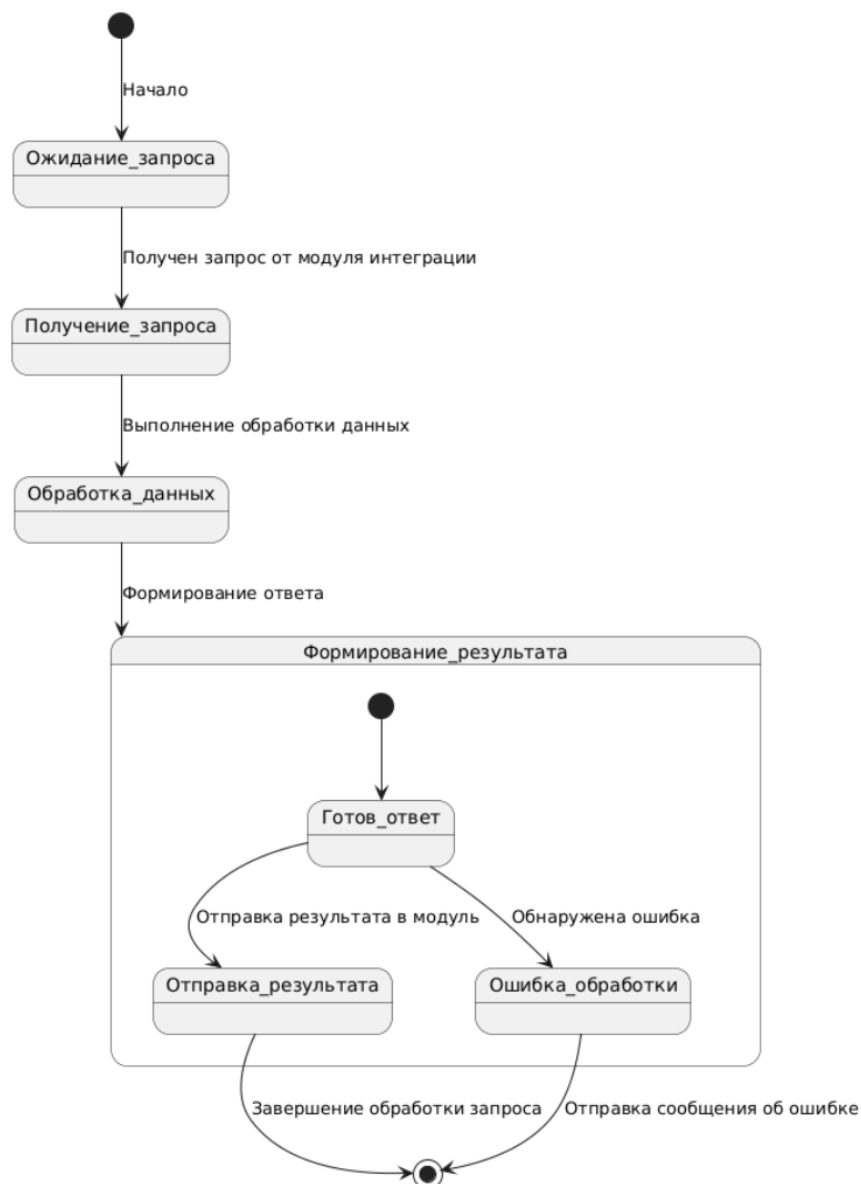


Рисунок 5 – Диаграмма состояний, отражающая ответ нейросетевой модели на запрос

Предлагаемая методика анализа текстовых данных основывается на принципах модульности, расширяемости и интеграции разнородных инструментов. Для решения поставленной задачи предлагается последовательный пайплайн, включающий следующие основные стадии:

- Сбор и подготовка данных: на этом этапе происходит интеграция различных источников текстовых данных: локальных файлов, корпоративных баз знаний, открытых репозиториях, веб-API. Предусматриваются механизмы обновления, фильтрации и предварительной сортировки данных.
- Предобработка текстов: включает очистку текстового материала от служебных символов, HTML-разметки, эмодзи и другого шумового контента, а также выполнение операций лемматизации, стемминга, нормализации. Применяются языковые морфологические анализаторы и фильтрация стоп-слов, что обеспечивает стандартизацию входных данных для дальнейших аналитических процедур.
- Лингвистический и семантический анализ: на данном этапе к подготовленным данным применяются инструменты для выделения именованных сущностей (Named Entity Recognition, NER), частеречной разметки (POS-теггинга), определения синтаксической структуры предложений и контекстно-зависимой семантики. Затем реализуется тематическое моделирование (LDA, NMF, LSI), анализ тональности и построение векторных представлений текста с помощью трансформерных моделей (например, BERT, RoBERTa). Результатом этого этапа является формирование структурированных семантических признаков, необходимых для последующих шагов.
- Аналитическая интеграция и расширяемость: предусматривается возможность подключения новых инструментов, модулей, моделей и алгоритмов без существенной перестройки основной архитектуры. Реализуемые API обеспечивают взаимодействие с внешними системами бизнес-аналитики, базами знаний и инструментами машинного обучения.
- Визуализация и предоставление результатов пользователю: Итоговые данные представляются в удобном графическом интерфейсе с применением интерактивных дашбордов, облаков слов, тематических карт, графов связей между сущностями. Пользователь получает возможность настраивать параметры

анализа, фильтровать результаты, комбинировать различные визуальные представления, что расширяет аналитические возможности системы.

– Управление доступом и безопасность: Значимым аспектом методики является четко определенная система разграничения прав. Каждому пользователю могут быть назначены роли с различными уровнями полномочий. Это обеспечивает защиту конфиденциальных данных, контроль за изменениями в настройках анализа и прозрачный аудит действий.

Приложение А демонстрирует общую логику работы комплекса, распределяя функциональные уровни в соответствии с их ролью в общем пайплайне. Общая схема решения задачи предусматривает последовательную трансформацию данных от неструктурированного текстового массива к осмысленной и наглядной информационной модели, доступной для анализа и принятия решений.

В вышеописанном тексте был отражен комплексный подход к реализации системы обработки текстовых данных с использованием Telegram-бота и нейросетевой модели. В нем последовательно рассматриваются ключевые этапы – от сбора и подготовки данных до лингвистического анализа, аналитической интеграции, визуализации результатов и обеспечения безопасности посредством разграничения прав доступа. Представленные диаграммы (прецедентов, последовательностей и состояний) иллюстрируют не только архитектурные решения, но и динамику взаимодействия модулей системы, позволяя увидеть целостную картину процесса обработки запросов. Такой модульный и масштабируемый подход создаёт прочную основу для дальнейшей адаптации и развития решения в условиях изменяющихся требований и роста нагрузки.

2.2. Профильное программное обеспечение для реализации нейросетевого чат-бота

Реализация авторской методики базируется на интеграции проверенных, широко распространённых программных решений, позволяющих обеспечить высокую эффективность, надежность и масштабируемость разрабатываемой системы. Ключевым элементом данного подхода является использование внешней

языковой модели YandexGPT, которая выполняет роль интеллектуального ядра для обработки естественного языка и генерации осмысленных ответов. Интеграция с YandexGPT осуществляется посредством REST API, что обеспечивает стандартизированный, гибкий и масштабируемый обмен данными между системой и внешним сервисом. Такой подход позволяет формировать запросы, включающие как текущее сообщение пользователя, так и накопленный контекст диалога, что существенно повышает релевантность и связность генерируемых ответов.

Особое внимание в методике уделяется возможностям YandexGPT по интерпретации сложных запросов. Эта языковая модель, обученная на обширных корпусах текстов, демонстрирует высокое качество генерации, способна учитывать контекст предыдущих сообщений и адаптировать стиль ответа в зависимости от специфики диалога. Поддержка настраиваемых параметров, таких как «температура» генерации, максимальное число токенов, выбор конкретной модели и прочие настройки, позволяет тонко регулировать поведение нейросети в зависимости от текущих задач. Это открывает перспективы для дальнейшей адаптации системы: в будущем модуль интеграции может быть легко заменён или дополнен альтернативными решениями, такими как модели из семейства HuggingFace Transformers или OpenAI GPT, что обеспечивает максимальную гибкость архитектуры и дает возможность оперативно реагировать на изменения требований или рыночных условий.

Для обеспечения взаимодействия с конечными пользователями в разработке системы используется библиотека Python-telegram-bot, выступающая связующим звеном между Telegram-ботом и основной логикой приложения. Эта библиотека предоставляет широкий спектр функциональных возможностей: она реализует маршрутизацию входящих сообщений и команд, поддержку встроенных клавиатур, inline-кнопок и callback-запросов, что значительно упрощает создание интуитивно понятного пользовательского интерфейса. Применение асинхронных функций в рамках библиотеки позволяет обеспечить высокую

отзывчивость системы даже при пиковых нагрузках, что критически важно для оперативной обработки запросов в реальном времени. Благодаря этому конечный пользователь получает мгновенный отклик, а система сохраняет высокую производительность при параллельном обслуживании множества пользователей.

Инструменты для сетевого взаимодействия играют не менее важную роль в реализации методики. Библиотека Requests, широко используемая в экосистеме Python, обеспечивает надежный механизм выполнения HTTP-запросов к внешним API, таким как YandexGPT. Requests позволяет не только формировать запросы с необходимыми параметрами, но и обрабатывать коды состояния ответов сервера, что важно для своевременного выявления ошибок, сетевых сбоев или некорректных данных. Возможность перехвата и обработки ошибок в режиме реального времени обеспечивает устойчивость системы и минимизирует риск потери данных, что особенно важно в условиях постоянного обмена информацией между компонентами системы.

Для повышения прозрачности работы приложения и оперативного выявления возможных проблем внедрена система журналирования с использованием встроенного модуля Logging. Грамотно настроенное логирование позволяет фиксировать все ключевые события: от приема входящих сообщений и передачи запросов к внешним сервисам до получения ответов и обработки ошибок. Такая детальная запись событий является не только инструментом для отладки, но и базой для анализа производительности системы. Логи помогают выявить потенциальные узкие места, оптимизировать процесс обработки запросов и обеспечить своевременное реагирование на любые неполадки. В перспективе накопленные журналы могут стать основой для построения системы мониторинга и автоматизированного аудита, что дополнительно повысит надежность и безопасность системы.

Вопросы безопасности и контроля доступа решаются посредством реализации простого, но эффективного механизма вайтлиста, который ограничивает доступ к функционалу бота только для определенного круга пользователей.

Несмотря на свою базовую природу, данный подход позволяет значительно снизить риск несанкционированного использования ресурсоемких функций, обеспечивая защиту от злоумышленников и предотвращая перегрузку системы. В будущем существующую схему можно расширить до полноценной системы аутентификации и авторизации с использованием современных протоколов, таких как OAuth2 или JWT. Это позволит не только усилить контроль доступа, но и интегрировать систему с корпоративными решениями по управлению безопасностью, что особенно актуально при обработке конфиденциальных данных и взаимодействии с внешними сервисами.

Дополнительно, внедрение многофакторной аутентификации и ролевой модели доступа создаст гибкую структуру разграничения полномочий пользователей, что повысит общую устойчивость системы к внутренним и внешним угрозам.

Архитектурное решение системы представлено в виде модульной структуры, где каждый компонент выполняет строго определенную функцию. Основными модулями являются Telegram-бот, модуль интеграции с языковой моделью, система логирования и механизм контроля доступа. Такая структура обеспечивает высокую масштабируемость, поскольку позволяет добавлять новые функциональные блоки или заменять отдельные компоненты без необходимости полной переработки системы. Модульный подход также способствует упрощению разработки, тестирования и поддержки, позволяя параллельно работать над различными частями системы. Важным преимуществом архитектуры является возможность динамического масштабирования с помощью контейнерных платформ, таких как Docker или Kubernetes, что позволяет распределять нагрузку и оперативно реагировать на изменения в количестве запросов.

Дополнительно система предусматривает интеграцию с внешними базами данных, такими как PostgreSQL или MongoDB, а также с системами полнотекстового поиска, например, Elasticsearch. Это обеспечивает долговременное хранение истории диалогов, что является критически важным для аналитики и

последующего улучшения качества ответов. Долговременное хранение данных позволяет вести подробный анализ поведения пользователей, выявлять закономерности и настраивать систему для оптимальной работы в различных сценариях. Более того, возможность интеграции с BI-системами и аналитическими платформами, такими как Grafana и Kibana, открывает широкие возможности для визуализации статистики использования, оперативного мониторинга и прогнозирования нагрузки.

Для реализации кэширования и ускорения доступа к часто запрашиваемым данным в системе используется Redis – высокопроизводительное хранилище ключ-значение, работающее в оперативной памяти. Его применение позволяет существенно сократить время отклика при обработке повторяющихся запросов, снизить нагрузку на основную базу данных и обеспечить быстрый обмен данными между компонентами системы. Redis также может быть задействован для хранения временных сессий, промежуточных результатов обработки и токенов авторизации, что способствует повышению общей масштабируемости и отказоустойчивости информационной системы.

Важной составляющей комплексного решения является возможность адаптации системы к динамично меняющимся требованиям. Используемые компоненты легко взаимозаменяемы и могут быть дополнены более мощными и специализированными модулями в зависимости от задач. Например, при необходимости повышения качества генерации текста можно заменить или дополнить YandexGPT альтернативными моделями, что позволяет добиться максимальной точности и релевантности ответов. Такая гибкость архитектуры является важным конкурентным преимуществом, позволяющим быстро внедрять инновации и адаптироваться к новым технологическим вызовам.

Общий обзор используемого профильного программного обеспечения демонстрирует, что реализуемая методика опирается на надежные и проверенные инструменты, отвечающие высоким требованиям к производительности, безопасности и удобству использования. Комплексный подход, сочетающий в себе

интеграцию внешней языковой модели, использование эффективных средств для сетевого взаимодействия, надежное логирование, а также гибкую систему контроля доступа, создает прочную основу для дальнейшего развития программного продукта. Такая архитектура не только обеспечивает высокое качество обработки запросов, но и позволяет оперативно реагировать на изменения, масштабировать систему и интегрировать ее с современными сервисами и аналитическими платформами.

Таким образом, реализация авторской методики представляет собой всесторонне проработанное решение, способное удовлетворить потребности как текущего этапа разработки, так и будущего масштабирования, и модернизации. Применение современных технологий и инструментов, а также модульный подход к построению архитектуры, гарантируют высокую надежность, гибкость и адаптивность системы. Это позволяет не только решать поставленные задачи на высоком уровне, но и создавать условия для дальнейшего расширения функциональности, интеграции с новыми сервисами и удовлетворения растущих требований пользователей в динамично меняющемся цифровом пространстве. Кроме того, предложенный подход обеспечивает технологическую независимость и устойчивость решения к изменениям как в пользовательских предпочтениях, так и в программной и аппаратной среде.

2.3 Обоснование выбора программного обеспечения

2.3.1 Критерии выбора ПО

При разработке системы на основе Telegram-бота с использованием нейросетевой модели необходимо учитывать ряд критериев для обоснованного выбора технологий. Ключевыми параметрами являются гибкость, производительность, масштабируемость и совместимость решений с другими компонентами системы. Эти качества определяют способность системы адаптироваться к изменяющимся требованиям, эффективно обрабатывать возрастающие нагрузки и беспрепятственно интегрироваться в существующую ИТ-инфраструктуру. Кроме того, важны такие аспекты, как надежность, безопасность, простота

поддержки и наличие сообщества разработчиков. В литературе предлагается оценивать технологии на основе совокупности факторов, включая размер и сложность проекта, скорость разработки, доступность специалистов, устойчивость к отказам, тренды развития и др. Опираясь на эти критерии, производится объективный выбор инструментов, позволяющий обеспечить баланс между производительностью системы и затратами на её развитие. В совокупности, системный подход к выбору технологического стека позволяет не только оптимизировать текущую реализацию, но и заложить прочную основу для дальнейшего масштабирования и модернизации системы.

2.3.2 Анализ альтернатив

На этапе анализа альтернатив рассматривались различные инструменты, способные удовлетворить указанным критериям. Для каждого компонента системы были проанализированы несколько решений. Разработка Telegram-бота могла быть реализована с использованием различных Python-фреймворков (например, Aiogram, Telethon), однако они сравнивались по гибкости и простоте реализации. Обработка HTTP-запросов к внешним API могла выполняться либо встроенными средствами Python (модуль `urllib`), либо асинхронными клиентами (`aiohttp`, `HTTPX`) для повышения производительности. В части нейросетевой модели основными альтернативами выступали использование внешнего API (например, OpenAI GPT-3) либо развертывание собственной модели. Хранение данных могло быть организовано как с помощью реляционных СУБД (PostgreSQL, MySQL), так и нереляционных хранилищ (NoSQL). Для логирования и мониторинга сравнивались традиционные методы (простое логирование в файл) с продвинутыми стеками наблюдаемости (ELK/EFK-стек, системы Application Performance Monitoring). Наконец, способы развертывания рассматривались от прямого запуска на сервере до контейнеризации с оркестрацией (Docker Compose, Docker Swarm, Kubernetes). По каждому направлению выбор пал на конкретные инструменты, удовлетворяющие критериям проекта лучше альтернатив – причины этого выбора подробно изложены ниже.

2.3.3 Описание выбранных инструментов

В качестве основного фреймворка для разработки бота выбрана библиотека `python-telegram-bot`. Данное решение продиктовано ее высокой гибкостью и богатым функционалом для работы с Telegram Bot API.

Библиотека предоставляет интуитивно понятный интерфейс, благодаря чему упрощается реализация команд и обработчиков сообщений; это делает ее подходящим выбором как для начинающих, так и для опытных разработчиков.

`Python-telegram-bot` активно поддерживается сообществом и регулярно обновляется, что гарантирует устойчивость проекта в долгосрочной перспективе.

Альтернативные решения (например, `Asyncio`-фреймворк `Aiogram`) также рассматривались, однако были отклонены из-за меньшей распространенности и немного более высокой сложности освоения. Для выполнения HTTP-запросов к внешнему сервису выбран модуль `Requests`, являющийся де-факто стандартом для HTTP-клиента в Python. Библиотека `Requests` интуитивно понятна и проста в использовании, имеет превосходную документацию и широкое сообщество пользователей.

Хотя существуют более производительные или асинхронные клиенты (HTTPX, `Aiohttp`), `Requests` был предпочтителен ввиду достаточной производительности для задач бота и простоты интеграции. Нейросетевая модель интегрируется через `YandexGPT API` – облачный сервис от Яндекса для генерации текста. Выбор сделан в пользу `YandexGPT` по причине его ориентации на русскоязычный контент и удобной интеграции через `Yandex Cloud`. Данный API предоставляет возможности по созданию интеллектуальных чат-ботов, генерации и суммаризации текстовой информации в бизнес-приложениях.

`YandexGPT` доступен в двух режимах: асинхронная модель для сложных задач, требовательных к ресурсам, и упрощенная модель для ответов в режиме реального времени.

Это позволяет достичь баланса между качеством ответа и быстродействием системы, в зависимости от контекста запроса. Альтернативой было

использование зарубежных моделей (например, OpenAI GPT-3/4), однако учитывая вопросы локализации, стоимости и требований к защите данных, YandexGPT стал оптимальным выбором.

Для хранения данных, таких как записи диалогов или настройки пользователей, выбрана реляционная СУБД PostgreSQL. PostgreSQL – одна из наиболее развитых открытых СУБД, сопоставимая по возможностям с коммерческими решениями

Она обеспечивает полное соблюдение свойств ACID, что гарантирует надежность и целостность данных при обработке транзакций. Кроме того, PostgreSQL поддерживает широкий набор типов данных (включая JSON) и сложные составные запросы, что дает гибкость в эволюции структуры данных и реализации сложной логики. Альтернативы вроде SQLite (слишком ограничен в масштабируемости) или MySQL/MariaDB (менее строгая соблюдаемость стандартов SQL, более бедные возможности JSON-хранения) были признаны менее предпочтительными. Для ведения журналов событий используется встроенный модуль logging языка Python. Он предоставляет гибкую систему логирования с разграничением уровней важности сообщений, конфигурацией форматирования и возможностью вывода в разные цели (файлы, консоль, удаленные серверы).

В отличие от примитивного вывода через print, использование стандартизированного логгера упрощает отладку и сопровождение: логи могут централизованно собираться и анализироваться, а уровень детализации легко настраивается без изменения кода. Таким образом, связка PostgreSQL и системы логирования составляет надежный фундамент для долговременного хранения и отслеживания информации, необходимой для работы бота и аналитики его работы.

Дополнительно в архитектуре системы используется Redis – высокопроизводительное хранилище данных в формате ключ-значение, работающее в оперативной памяти. В отличие от реляционных СУБД, Redis оптимизирован под молниеносный доступ к временным или часто запрашиваемым данным. Он идеально подходит для хранения пользовательских сессий, кэширования промежуточных

результатов, токенов аутентификации и других данных, актуальных в течение ограниченного времени. Такое распределение нагрузки между PostgreSQL и Redis позволяет добиться как надежного долговременного хранения, так и высокой скорости отклика системы в условиях интенсивного взаимодействия с пользователями. Кроме того, благодаря поддержке структурированных типов данных, таких как списки, множества и хэши, Redis может использоваться не только для кэширования, но и для реализации очередей задач, механизма блокировок или счётчиков, что делает его важным компонентом масштабируемой и отказоустойчивой архитектуры.

Важным требованием к разрабатываемой системе является возможность наблюдать за ее работой в реальном времени и ретроспективно анализировать поведение. С этой целью интегрированы инструменты мониторинга: Grafana и Kibana. Grafana выбрана как открытая платформа визуализации метрик и построения информационных панелей. Она позволяет в наглядной форме отображать показатели производительности (например, количество обрабатываемых ботом сообщений в единицу времени, латентность ответов API, загрузку процессора/памяти контейнеров и т.д.) и тем самым оперативно выявлять узкие места. Grafana гибко подключается к различным источникам данных (в частности, ее легко интегрировать с системами сбора метрик типа Prometheus) и благодаря большому сообществу имеет множество готовых решений для визуализации. В дополнение к метрикам, для углубленного анализа логов системы используется Kibana – компонент стека Elastic. Kibana представляет собой мощный инструмент визуализации и анализа данных, позволяющий создавать интерактивные дашборды и отчеты по поступающим логам. В связке с хранилищем Elasticsearch данный инструмент обеспечивает быстрый поиск по журналам работы бота (например, по идентификатору сессии или ключевым словам ошибок) и построение аналитических графиков на основе событий. Альтернативой могла быть проприетарная система мониторинга либо упрощенное решение на основе простого просмотра лог-файлов, однако сочетание Grafana и Kibana (как открытых

и масштабируемых инструментов) предоставляет максимально гибкую и масштабируемую систему наблюдаемости. Это решение позволяет оперативно обнаруживать неполадки, отслеживать статистику использования бота и принимать обоснованные решения по оптимизации системы на основе данных. Контейнеризация и оркестрация.

Для обеспечения переносимости и масштабируемости приложения применены технологии контейнеризации. Сервис Telegram-бота упакован в контейнер с использованием Docker. Благодаря универсальности и простоте использования Docker стал стандартом в современном IT-ландшафте. Контейнеризация позволяет изолировать окружение приложения вместе со всеми зависимостями, что гарантирует идентичное поведение на этапах разработки, тестирования и в промышленной эксплуатации. Docker-образы обеспечивают быстрое развертывание и обновление приложения, а также более эффективное использование ресурсов по сравнению с традиционными виртуальными машинами. Для управления контейнеризированным приложением и его масштабирования используется система оркестрации Kubernetes. Kubernetes на сегодняшний день является де-факто стандартом оркестрации контейнеров и существенно упрощает развертывание, масштабирование и управление контейнеризированными приложениями

С помощью Kubernetes развернутое решение может автоматически масштабироваться при увеличении нагрузки (добавляя реплики бота), обеспечивать отказоустойчивость (перезапуск контейнера в случае сбоя) и облегчит управление конфигурацией через декларативные манифесты. Альтернативные подходы, такие как ручное управление контейнерами или использование менее функциональных оркестраторов, уступают Kubernetes по уровню автоматизации и поддержки со стороны сообщества. Таким образом, применение Docker и Kubernetes в связке гарантирует, что разработанная система сможет гибко развиваться и масштабироваться без существенной переработки архитектуры.

Kubernetes также упрощает управление конфигурацией посредством декларативных манифестов, что позволяет централизованно контролировать

состояние всего кластера и вносить изменения в конфигурацию без необходимости ручного вмешательства в каждый контейнер. Это обеспечивает гибкость в настройке системы, облегчает обновление компонентов и делает процесс администрирования более прозрачным. Альтернативные подходы, такие как ручное управление контейнерами или использование менее функциональных оркестраторов, уступают Kubernetes по уровню автоматизации, интеграции с облачными сервисами и поддержке со стороны глобального сообщества разработчиков.

Кроме того, использование Docker и Kubernetes в связке гарантирует, что разработанная система сможет гибко развиваться и масштабироваться без существенной переработки архитектуры. Такая интеграция позволяет не только оперативно реагировать на изменения в требованиях пользователей, но и эффективно оптимизировать использование вычислительных ресурсов. Контейнеризованное приложение легко переносимо между различными облачными платформами, что предоставляет возможность выбора наиболее оптимального инфраструктурного решения и снижает затраты на эксплуатацию. Поскольку оба инструмента являются открытыми проектами с огромным сообществом, разработчики могут рассчитывать на регулярные обновления, улучшения функциональности и оперативное решение возникающих проблем.

В контексте описанного подхода, на рисунке 6 приведена схема, демонстрирующая, каким образом контейнеризация и оркестрация позволяют гибко управлять сервисом Telegram-бота.

На схеме представлена последовательность обмена сообщениями между основными компонентами архитектуры: пользователем, Telegram-ботом, внешним API нейросетевой модели и подсистемами хранения/мониторинга. Процесс начинается с отправки пользователем сообщения боту, после чего бот (развернутый в Docker-контейнере и работающий под управлением Kubernetes) принимает сообщение через Telegram API и обрабатывает его с помощью библиотеки `python-telegram-bot`. Затем бот формирует запрос к внешнему сервису YandexGPT API (с использованием библиотеки `Requests`) для генерации ответа

на основе нейросети. Получив ответ, бот сохраняет необходимые данные (например, факт обращения или полученный результат) в базе данных PostgreSQL и отправляет сгенерированный ответ пользователю в чат. Параллельно с этим, ключевые события (вызовы API, ошибки, время ответа) фиксируются в системе Logging. Инструменты мониторинга Grafana и Kibana используют эти данные: Grafana отображает метрики работы (например, время ответа на запрос пользователя), тогда как Kibana позволяет анализировать логи (например, содержание ошибок или частоту обращений). Представленная диаграмма наглядно подтверждает обоснованность выбора архитектуры и ПО: каждый компонент выполняет специализированную роль, а их совместное использование обеспечивает требуемую гибкость, производительность и масштабируемость системы.

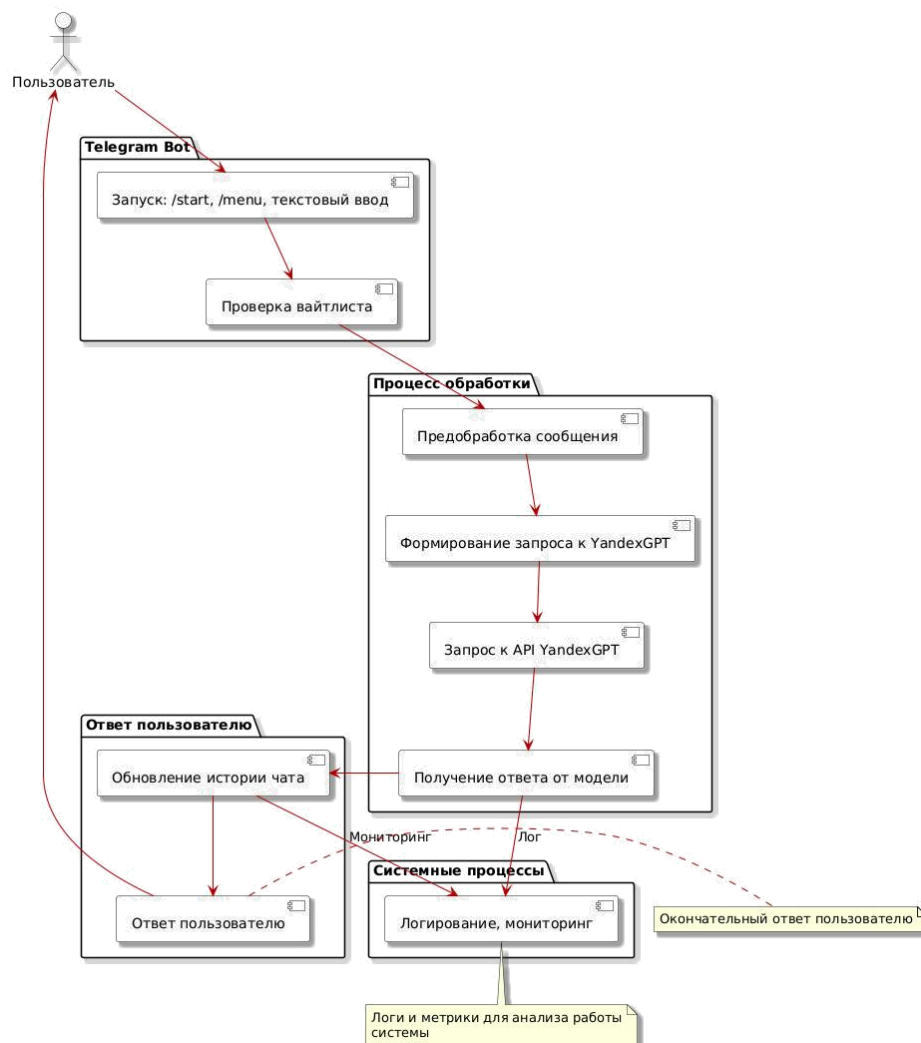


Рисунок 6 – Структурная схема обработки запроса в Telegram-боте на базе YandexGPT

Во второй главе диссертационного исследования представлена комплексная методика проектирования системы, основанной на использовании Telegram-бота и внешней языковой модели. Проведён анализ основных функциональных сценариев, что позволило обосновать выбор ключевых компонентов архитектуры и продемонстрировать логику взаимодействия модулей через диаграммы прецедентов, последовательностей и состояний. В результате сформирована структурированная модель решения, отражающая этапы от предобработки входящих сообщений и формирования запросов к нейросетевой модели до получения итогового ответа и его визуализации пользователю.

Проведённый обзор профильного программного обеспечения (включая библиотеки Python-telegram-bot и Requests, а также YandexGPT API, Docker и Kubernetes) подтвердил целесообразность применения современных инструментов, обеспечивающих гибкость и масштабируемость системы. Выделены основные преимущества модульного подхода и контейнеризации, что позволяет оперативно реагировать на изменения в требованиях, повышать отказоустойчивость и упрощать процесс интеграции с внешними сервисами. Тем самым вторая глава демонстрирует обоснованность выбранных технологических решений и закладывает фундамент для реализации системы в соответствии с поставленными целями и требованиями к качеству.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРЕДЛАГАЕМОГО АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ

В данной главе описывается практическая реализация разработанного чат-бота для мессенджера Telegram с использованием нейросетевой модели YandexGPT (Lite). Особенностью решения является то, что тематика ответов бота ограничена вопросами, связанными с Амурским государственным университетом (АМГУ). Ниже приведены основные этапы жизненного цикла программного продукта, принятые архитектурные и технологические решения, выбор инструментов, формулировка функциональных требований, а также детали программной реализации: структура кода, обработка команд и сообщений, интеграция с API YandexGPT. Приводятся фрагменты кода наиболее важных компонентов и пример работы чат-бота с иллюстрациями.

3.1 Основные этапы практической разработки программного продукта

Создание программного продукта (чат-бота) потребовало прохождения ряда этапов, от планирования до тестирования и развёртывания. Ниже рассматриваются модель жизненного цикла проекта и конкретные стадии разработки, архитектурное проектирование, выбор технологий, а также определение требований к системе.

3.1.1 Модель жизненного цикла проекта

При разработке программного обеспечения важно определить модель жизненного цикла – последовательность стадий и работ от зарождения идеи до эксплуатации и сопровождения системы. Существует несколько классических моделей жизненного цикла: водопадная (каскадная) модель, итеративная и инкрементальная разработка, гибкие методологии (Agile, Scrum, экстремальное программирование) и др. В российской практике нередко ориентируются на стандарт ГОСТ Р ИСО/МЭК 12207–2010, задающий стадии жизненного цикла и процессы на каждой из них. Каскадная модель предполагает строгую

последовательность этапов – от сбора требований до сопровождения – без возврата к предыдущим фазам, тогда как современные гибкие методологии допускают итеративное развитие и постоянную адаптацию по мере получения обратной связи.

В контексте данной работы разработка чат-бота велась одним разработчиком и в сжатые сроки, поэтому был применён комбинированный подход. Изначально проект планировался и анализировался по классической схеме (как в каскадной модели), но в процессе реализации использовались принципы итеративной разработки. Это означало, что разработка шла циклично: добавлялась часть функциональности – затем сразу проводилось тестирование и при необходимости вносились правки, после чего переходили к следующей итерации. Такой подход, вдохновлённый Agile, позволил гибко реагировать на изменения и улучшать систему на ходу. Например, базовая версия бота с простым ответом была создана и протестирована, затем постепенно усложнялась: добавлялся системный промпт, механизм авторизации, контекстный диалог. Принципы экстремального программирования и визуализация задач в духе Kanban также нашли применение. В результате удалось эффективно организовать жизненный цикл проекта чат-бота с минимальными ресурсами, сочетая плановость классического подхода и адаптивность Agile.

Как показано на таблице 1, комбинированный подход сочетает в себе преимущества итеративного метода и строгости каскадной модели.

Таблица 1 – Сравнение моделей жизненного цикла разработки ПО

Характеристика	Каскадная	Инкрементальная	Agile	Комбинированная
Последовательность этапов	Да	Частично	Нет	Частично
Итеративность	Нет	Да	Да	Да
Подходит для единоличной разработки	Частично	Да	Да	Да
Возможность оперативного изменения требований	Нет	Да	Да	Да

3.1.2 Этапы разработки Telegram-бота

Процесс разработки информационной системы включал ряд этапов, каждый из которых имел свои цели и результаты. Ниже перечислены ключевые стадии жизненного цикла создания Telegram-бота с интеграцией нейросети:

Предпроектный анализ. На этом этапе проводился сбор и анализ требований к системе. Определялись цели проекта – разработать чат-бот, способный отвечать на вопросы пользователей на естественном языке с помощью модели YandexGPT. Учитывая специфику задачи, была обозначена предметная область бота: информация об Амурском государственном университете (история, структуры, учебные направления и т.д.). Выявлялись потребности целевых пользователей, например необходимость круглосуточного автоматизированного консультирования по вопросам, связанным с университетом. Сформулировано общее видение продукта. Также проводилось изучение аналогов (существующих AI-ботов информационной направленности), возможностей Telegram Bot API и сервиса YandexGPT. Результатом этапа стал предварительный перечень функций бота и ограничений (например, доступ к боту только у авторизованных лиц), а также критерии успешности проекта.

Проектирование. На этапе проектирования была разработана архитектура системы и продуманы способы реализации требований. Определено, что бот будет реализован на Python в виде асинхронного приложения, взаимодействующего с Telegram через Bot API и использующего облачный API YandexGPT для генерации ответов. Спроектирована схема взаимодействия между основными компонентами системы: пользовательский интерфейс (чат в Telegram), серверная логика бота и внешнее AI-сервис (модель YandexGPT). Также была продумана структура программного кода: выделены основные модули (инициализация бота, обработчики команд, модуль работы с API YandexGPT и др.). Отдельно были разработаны алгоритмы обработки сообщений пользователя, включая проверку авторизации по whitelist и использование системного промпта для ограничения

тематики ответов нейросети. На этапе проектирования осуществлён и выбор технологий и библиотек. На рисунке 7 представлена диаграмма компонентов.

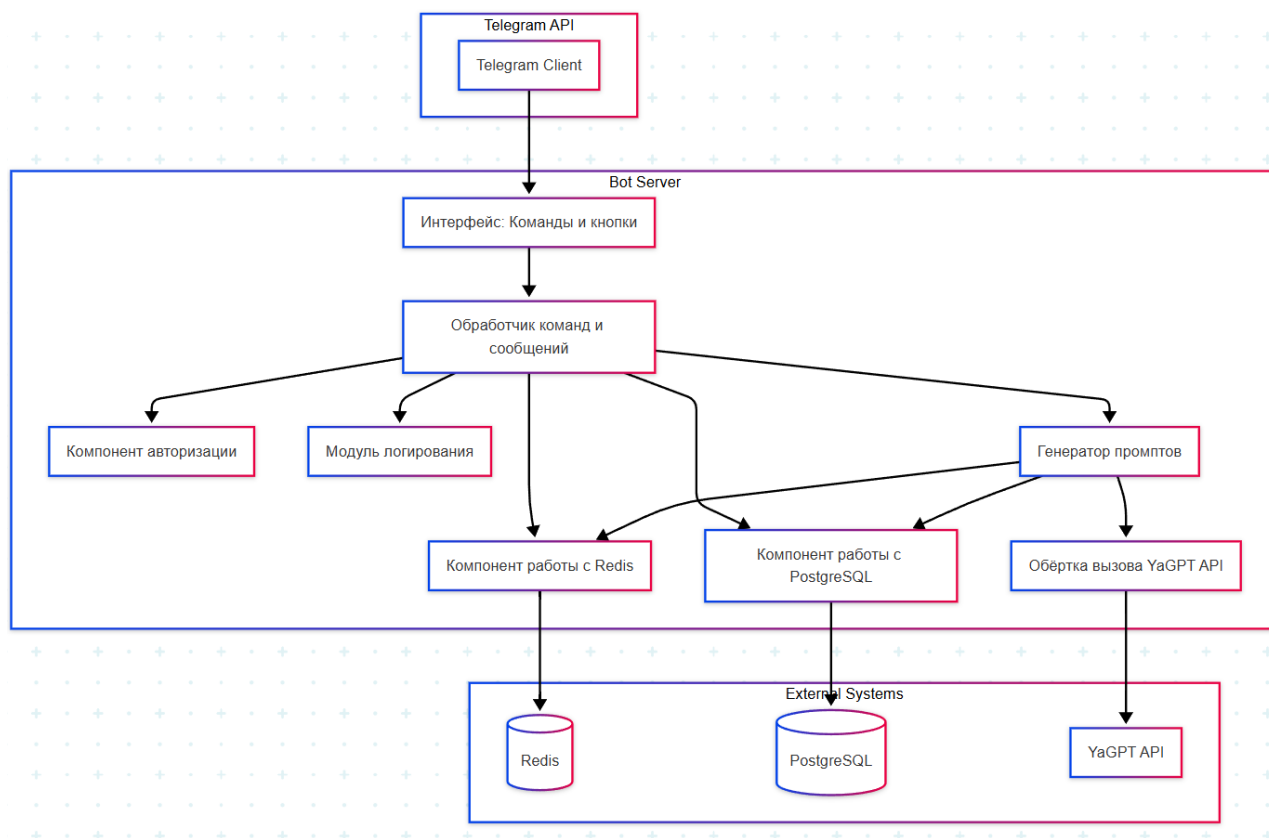


Рисунок 7 – Диаграмма компонентов

Данная диаграмма отражает архитектуру программного комплекса чат-бота, построенного с использованием Telegram API, языковой модели (YaGPT API), а также внешних хранилищ данных – Redis и PostgreSQL. Структура разделена на два уровня: Bot Server и External Systems.

Верхний уровень (Bot Server) представляет собой логический сервер, реализующий бизнес-логику обработки пользовательских сообщений, команд и взаимодействия с внешними сервисами.

Входной канал: Telegram Client. Взаимодействие с пользователем осуществляется через Telegram API, где клиент Telegram принимает входящие сообщения и команды, передавая их в интерфейс бота.

Команды и кнопки. Это интерфейс взаимодействия пользователя с ботом, включающий команды и элементы управления.

Обработчик команд и сообщений. Центральный компонент, отвечающий за маршрутизацию входящих запросов. Он направляет сообщения к соответствующим модулям, в зависимости от их типа и логики обработки.

Компонент авторизации. Отвечает за проверку прав доступа пользователей, регистрацию и верификацию, если требуется ограниченный доступ к функциям бота.

Модуль логирования. Служит для регистрации событий, ошибок и другой диагностической информации. Использует встроенные возможности языка Python для ведения логов.

Компонент работы с Redis. Предназначен для быстрой работы с временными или часто запрашиваемыми данными: хранение пользовательских сессий, кэширование результатов, счётчики и т.д.

Компонент работы с PostgreSQL. Отвечает за долговременное хранение структурированных данных, таких как история диалогов, настройки пользователей, логика работы с транзакциями.

Генератор промптов. Формирует запросы (промпты) к языковой модели YaGPT в зависимости от входящих данных, истории сообщений и контекста.

Обёртка вызова YaGPT API. Обеспечивает взаимодействие с внешним языковым API, инкапсулируя детали сетевого вызова и обработки ответов от модели.

Нижний уровень (External Systems) содержит внешние сервисы, с которыми взаимодействует серверная часть:

- Redis – хранилище временных и промежуточных данных.
- PostgreSQL – СУБД для хранения постоянных данных.
- YaGPT API – языковая модель, предоставляющая интеллектуальные ответы на основе полученного промпта.

Предложенная архитектура (динамика взаимодействия) удовлетворяет основным требованиям проекта: обеспечивается диалоговый режим через Telegram, используется нейросетевая модель для генерации ответов

(YandexGPT), контролируется доступ пользователей и сохраняется контекст диалога. После завершения проектирования можно переходить к реализации и настроить систему согласно разработанной спецификации (практические аспекты реализации рассматриваются в разделе 3.2).

Реализация. Данный этап заключался в непосредственном программировании чат-бота в соответствии с разработанным проектом. Кодирование велось на языке Python. Сначала был создан основной файл приложения (например, `main.py`), в котором инициализируется бот, задаются ключевые настройки (токены, списки доступа, промпты) и регистрируются обработчики событий (команд и сообщений). Затем были реализованы функции-обработчики для основных команд (`/start`, `/menu` и др.), а также обработчик произвольных текстовых запросов от пользователя. В процессе реализации особое внимание уделялось интеграции с API YandexGPT: написаны функции формирования запроса (в формате JSON, содержащего системный промпт, историю диалога и новый вопрос пользователя) и обработки ответа от API (извлечение сгенерированного текста и отправка его пользователю). Также в код были внедрены проверки авторизации пользователя (сопоставление его ID со списком разрешённых) и механизмы ведения истории диалога (сохранение предыдущих сообщений для учета контекста). Реализация велась с регулярным локальным тестированием компонентов (например, проверялось, что бот корректно отвечает на команду `/start` даже без обращения к ИИ, что формат отправляемого в YandexGPT запроса соответствует спецификации API). По мере написания кода проект размещался в системе контроля версий (Git), что позволяло отслеживать изменения и при необходимости откатывать неудачные решения. Как видно из Рисунка 7 – Блок-схема алгоритма обработки текстовых сообщений чат-ботом, после получения сообщения происходит последовательная проверка авторизации, извлечение текста, формирование запроса с учетом истории диалога и системного промпта, а затем вызов API YandexGPT для генерации ответа.

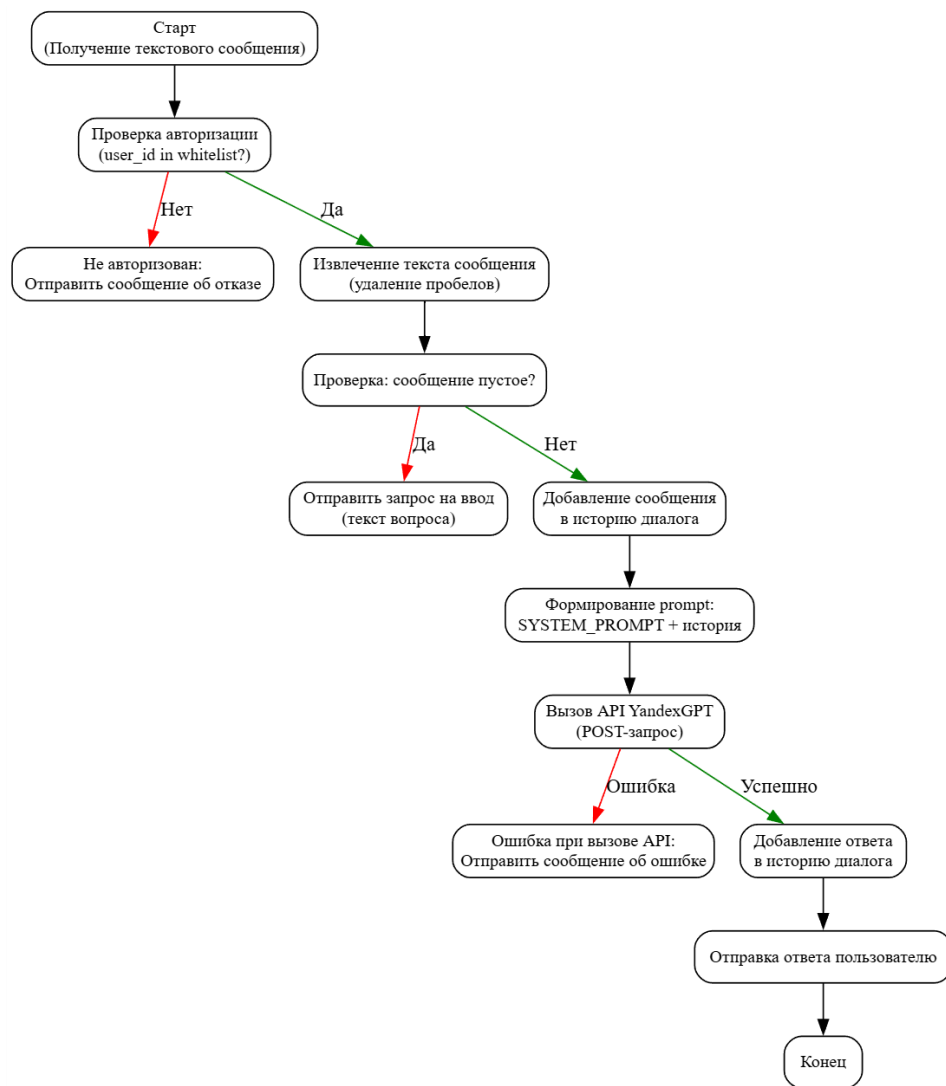


Рисунок 8 – Процесс обработки сообщений

Тестирование. После того, как базовая функциональность была готова, проводилось тестирование чат-бота. Тестирование шло в несколько этапов. Во-первых, модульное тестирование ключевых функций – например, функции формирования JSON-запроса к YandexGPT, функции обработки команды /start (проверка, что бот отправляет приветствие только авторизованным пользователям). Во-вторых, интеграционное тестирование – запуск всего бота и проверка полного цикла работы: от приёма сообщения через Telegram до получения ответа от YandexGPT и отправки результата пользователю. Бот проверяли на различных сценариях: сообщение от авторизованного пользователя vs от неавторизованного (проверка отказа в обслуживании), пустой или очень длинный запрос,

последовательность нескольких вопросов (проверка сохранения контекста разговора), использование команд меню и нажатие кнопок. Также измерялось время ответа: как правило, генерация занимала 2–5 секунд, что соответствовало требованию. Проверялась устойчивость к ошибкам: например, симулировалась недоступность API YandexGPT – бот должен был не «упасть», а вернуть пользователю сообщение об ошибке. Выявленные баги и несоответствия требованиям исправлялись итеративно: код дорабатывался и повторно тестировался. Благодаря гибкому подходу с повторяющимися итерациями, реализация и тестирование шли параллельно: новые возможности сразу проверялись в работе.

Развёртывание и ввод в эксплуатацию. После успешного тестирования чат-бот был развёрнут в целевом окружении для опытной эксплуатации. Поскольку бот работает локально, развёртывание заключалось в установке и запуске Python-приложения на выбранной машине (например, на сервере в университете или персональном компьютере) и обеспечении его непрерывной работы. Были настроены параметры конфигурации: токен Telegram Bot API, полученный у BotFather, ключ API YandexGPT, список разрешённых пользователей (whitelist) и другие настройки. Бот был запущен в режиме постоянного опроса сервера Telegram (long polling) для приёма новых сообщений. В ходе опытной эксплуатации контролировалась стабильность работы: бот запускался и останавливался несколько раз, проверялось автоматическое восстановление соединения при временных сбоях сети, анализировались логи на отсутствие исключений. По завершении этого этапа бот был признан готовым к постоянной работе. Определённые пользователи (добавленные в whitelist) получили ссылку на бота в Telegram и смогли начать с ним диалог, что фактически ознаменовало ввод системы в эксплуатацию.

После прохождения всех перечисленных этапов разработанный программный продукт – Telegram-бот с нейросетевой моделью – был готов к решению поставленных задач. Далее, в разделе 3.1.3 описываются архитектурные решения,

принятые при проектировании и реализации бота, а в разделе 3.1.5 формулируются требования к его функциональности.

3.1.3 Архитектура разработанного решения

Разработанный чат-бот построен по архитектуре, сочетающей событийно-ориентированную обработку сообщений и клиент-серверное взаимодействие с внешним AI-сервисом. В основе лежит асинхронное однопоточное приложение на Python, функционирующее локально и использующее API Telegram для приёма и отправки сообщений. Бот выступает в роли клиента по отношению к двум внешним системам: к серверам Telegram (через Telegram Bot API) и к сервису YandexGPT (через API генеративной модели). Иначе говоря, архитектуру можно представить как трёхзвенную схему: пользователь – сервер чат-бота – облачный AI-сервис. Пользователь взаимодействует с ботом через приложение Telegram, серверная часть бота осуществляет всю логику, а для получения интеллектуальных ответов привлекается облачная модель.

Основные составляющие архитектуры чат-бота включают следующие компоненты:

- Интерфейс с Telegram: обеспечивается через Telegram Bot API. Бот подключается к серверам Telegram и получает от них обновления (новые сообщения) либо путём периодического опроса (long polling), либо через механизм вебхуков. В данном проекте реализован long polling ввиду простоты настройки при локальной работе (нет необходимости запускать публичный веб-сервер для вебхуков). Telegram Bot API передаёт боту объекты обновлений (Update), содержащие входящие сообщения пользователя, на которые бот должен реагировать.

- Модуль обработки команд и сообщений: логика приложения, реализованная в коде на Python. Этот модуль решает, как реагировать на ту или иную входящую информацию. Он распознаёт сервисные команды (сообщения, начинающиеся с символа «/», например /start или /menu) и произвольные текстовые запросы. Для команд предусмотрены обработчики, выполняющие вспомогательные действия: приветствие пользователя, вывод меню с кнопками и пр. Для

обычных текстовых сообщений предназначен отдельный обработчик, реализующий основную функцию бота – формирование запроса к нейросети и получение ответа. Обработчик текстовых сообщений также отвечает за ведение контекста диалога (сохранение истории) и применение системного промпта перед обращением к модели.

– Модуль интеграции с YandexGPT: компонент, отвечающий за взаимодействие с API YandexGPT. Он получает от основного логического модуля сформированный запрос пользователя (включая историю предыдущих сообщений и системное сообщение-промпт), подготавливает HTTP-запрос в требуемом формате (JSON) к облачному сервису и отправляет его. После получения ответа (сгенерированного моделью текста) данный модуль обрабатывает его (извлекает нужные данные из JSON) и возвращает результат в основной модуль для последующей отправки пользователю. Схема на рисунке 2 демонстрирует, как происходит обмен данными между Telegram, серверной логикой и API YandexGPT.



Рисунок 9 – Блок-схема алгоритма обработки текстовых сообщений чат-ботом

– Механизм авторизации (контроль доступа): компонент, гарантирующий, что бот отвечает только определённым кругом пользователей. Реализован через хранение списка допустимых пользователей (whitelist) с уникальными идентификаторами Telegram. При каждом новом обращении бот проверяет ID отправителя: если пользователь не найден в списке разрешённых, то основной функционал ему не предоставляется. В нашем случае неавторизованному пользователю бот отправляет сообщение об отказе в обслуживании и игнорирует дальнейшие запросы. Таким образом обеспечивается ограничение доступа (например, если бот предназначен для использования только сотрудниками или студентами вуза). Механизм авторизации можно гибко настроить, меняя

содержимое whitelist (например, загрузкой из отдельного конфигурационного файла без изменения кода программы). Как показано на Рисунке 9, после проверки входящего запроса бот анализирует user_id и позволяет доступ только тем, кто входит в список разрешённых пользователей.

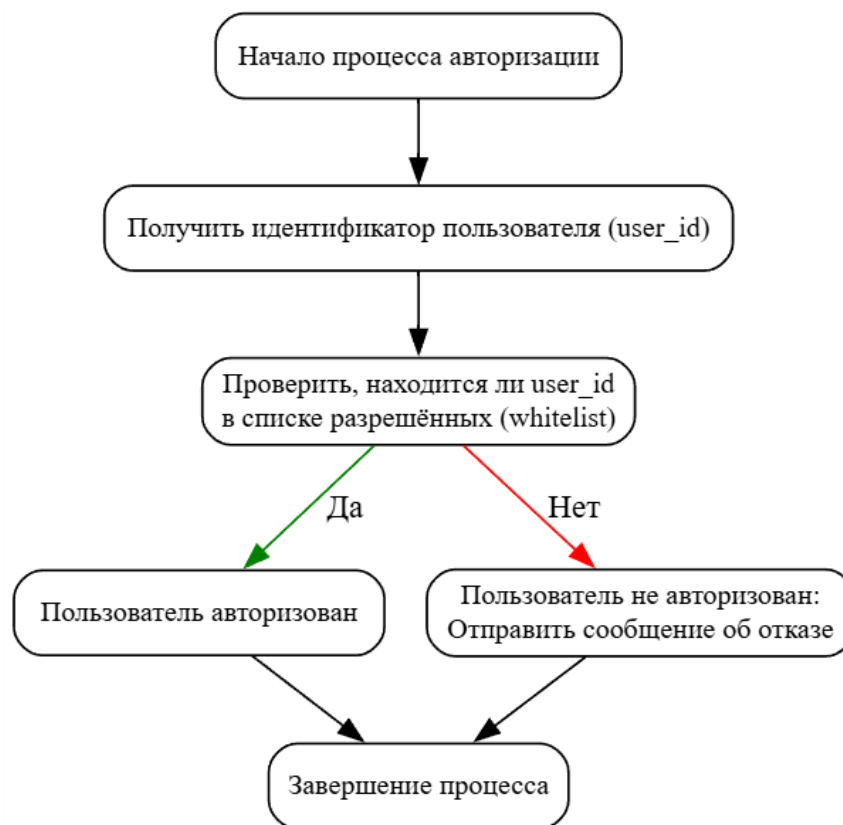


Рисунок 10 – Блок-схема механизма авторизации по whitelist

– Хранилище контекста диалога: используется для сохранения истории сообщений в текущей сессии общения с каждым пользователем. В рамках проекта хранение реализовано в оперативной памяти (например, в виде структуры данных dict в Python, где ключ – идентификатор чата или пользователя, а значение – список последних N сообщений диалога). Эта история включается в каждый запрос к YandexGPT, чтобы модель могла учитывать предыдущие вопросы и ответы, поддерживая связный разговор. Объём сохраняемой истории может быть ограничен (например, последние 10 сообщений) во избежание превышения лимитов длины prompt у модели.

Выбранная архитектура является асинхронной: это означает, что бот способен одновременно обрабатывать несколько событий (сообщений) без блокировки основного потока. Библиотека `python-telegram-bot` поддерживает `asyncio`, что позволяет, например, отправлять запрос к YandexGPT в фоне и параллельно принимать новые входящие сообщения. Асинхронность особенно важна, так как генерация ответа моделью может занимать сотни миллисекунд или несколько секунд в зависимости от вопроса, и неблокирующее поведение бота обеспечивает отзывчивость интерфейса.

Бот развёрнут локально, однако благодаря архитектуре Telegram это не ограничивает его географически: пользователи могут обращаться к боту из любого места, Telegram берёт на себя доставку сообщений по сети. Для доступа к YandexGPT необходима интернет-связь и действующий ключ Yandex Cloud API.

Таким образом, спроектированная архитектура удовлетворяет основным требованиям проекта: она обеспечивает диалоговый режим работы (через привычный мессенджер Telegram), использование возможностей нейросетевой модели (через YandexGPT), ограничение доступа (через `whitelist`), учёт контекста диалога и асинхронную обработку запросов для эффективности. Перед переходом к выбору инструментов и технологий, на Рисунке 10 представлена диаграмма развёртывания компонентов чат-бота, демонстрирующая, как локальное приложение взаимодействует с внешними сервисами.

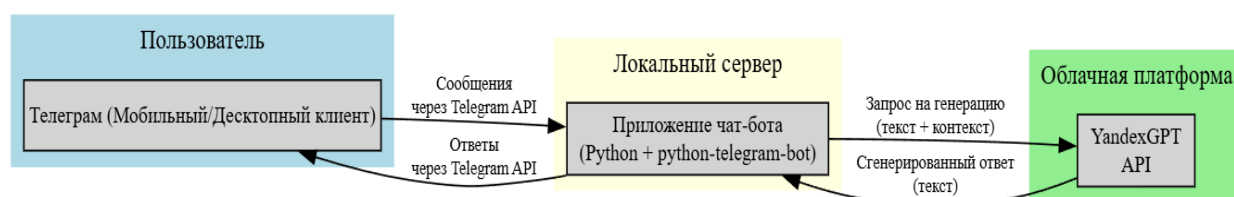


Рисунок 11 – Диаграмма развёртывания

3.1.4 Выбор инструментов и технологий

Для реализации описанного решения были выбраны современные инструменты, обеспечивающие надёжную и удобную разработку чат-ботов, а также

простую интеграцию с AI-моделями. Выбор технологий обосновывался критериями скорости разработки, совместимости с требуемым функционалом и наличия поддержки со стороны сообщества. Ключевые используемые средства перечислены ниже:

- Язык программирования Python. В качестве основного языка выбран Python (версии 3.9+). Python предоставляет высокоуровневые средства для работы с сетью, поддержки асинхронного программирования (`asyncio`), а также богатую экосистему готовых библиотек для самых разных задач. Для создания бота Python удобен тем, что существуют популярные библиотеки для работы с Telegram Bot API и HTTP-запросами. Кроме того, Python широко используется в сфере искусственного интеллекта, что упростило интеграцию с AI-сервисами. Динамическая типизация и интерактивная среда Python ускорили цикл разработки и отладки по сравнению с компилируемыми языками.

- Telegram Bot API и библиотека `python-telegram-bot`. Для взаимодействия с платформой Telegram выбрана библиотека `python-telegram-bot` (использована актуальная версия 20.x). Эта библиотека предоставляет удобный модуль `telegram.ext` с реализацией ключевых компонентов: класса приложения `Application`, диспетчера сообщений, обработчиков команд (`CommandHandler`), обработчиков текстовых сообщений (`MessageHandler`), фильтров (например, `filters.TEXT` для фильтрации текстовых сообщений) и др. Библиотека поддерживает асинхронный режим работы и совместима с новейшими версиями Telegram Bot API. Выбор пал на `python-telegram-bot` благодаря ее зрелости, подробной документации и активному сообществу. Альтернативами могли быть другие Python-библиотеки (например, `Aiogram` или `Telebot`), однако `python-telegram-bot` показалась наиболее подходящей из-за встроенной поддержки `asyncio` и удобной архитектуры (концепция `Application/Handlers`). С помощью данной библиотеки реализация бота свелась к регистрации необходимых обработчиков и вызову метода `Application.run_polling()`, после чего библиотека сама управляет получением обновлений от Telegram и передачей их в обработчики.

– HTTP-клиент `requests`. Для обращения к веб-сервису YandexGPT используется библиотека `requests` – популярный Python-модуль для выполнения HTTP-запросов. Хотя `requests` является синхронной (блокирующей) библиотекой, ее использование оправдано простотой: вызов внешнего API происходит внутри асинхронной функции бота, и, благодаря особенностям реализации `python-telegram-bot`, одновременная обработка других событий не страдает (библиотека может выполнять сетевые запросы в отдельных потоках, или же разработчик мог бы обернуть вызов в `executor`). Альтернативой мог выступить асинхронный HTTP-клиент (например, `aiohttp`), однако для целей демонстрации функциональности выбор сделан в пользу `requests` как более простого средства. С помощью этой библиотеки формируется HTTP POST-запрос к API YandexGPT с необходимыми заголовками и телом (JSON), а затем обрабатывается ответ.

– Облачный AI-сервис YandexGPT. В качестве ядра генерации ответов используется модель YandexGPT Lite, доступная через API Yandex Cloud. YandexGPT – это большая языковая модель (из семейства YaLM 2.0), предоставленная компанией Яндекс для внешних разработчиков в 2023 году. Сервис имеет несколько вариантов моделей; в частности, существует YandexGPT Lite и YandexGPT Pro, отличающиеся объемом и качеством (Pro – более мощная версия, требующая больше ресурсов). Для данного проекта выбран YandexGPT Lite как достаточный по качеству ответов и более экономичный по стоимости использования. В рамках образовательного проекта запросы к YandexGPT Lite укладываются в предоставляемую квоту. Преимущество YandexGPT для отечественных разработок в том, что сервис доступен из России без ограничений (в отличие от API OpenAI, который может иметь ограничения по регионам), а также более низкая стоимость запросов. Интеграция с YandexGPT выполнена через REST API: бот отправляет HTTPS-запрос на определённый эндпоинт Yandex Cloud и получает ответ в формате JSON.

– Дополнительные утилиты Python. Помимо перечисленного, в проекте используются стандартные модули Python: `asyncio` (для управления

асинхронными задачами и запуска обработчиков), logging (для ведения журнала работы бота, что облегчает отладку), а также модуль json для разбора ответа от YandexGPT. Управление зависимостями осуществлялось с помощью менеджера пакетов pip и виртуального окружения (venv), что позволило изолировать необходимые библиотеки. Код проекта оформлен в соответствии со стилевым стандартом PEP 8, чтобы облегчить чтение и сопровождение.

Выбор указанных инструментов был сделан после анализа альтернатив. Например, рассматривались другие библиотеки для Telegram, но python-telegram-bot предпочтительнее из-за преимуществ, описанных выше. В качестве AI-модели мог использоваться OpenAI ChatGPT API, однако (как отмечалось) YandexGPT оказался более подходящим по доступности и затратам, кроме того, учитывалась тематика проекта и возможность тонкой настройки (системного промпта) под русскоязычный контекст АМГУ. Таким образом, финальный стек технологий (Python + python-telegram-bot + requests + YandexGPT) оптимально соответствует поставленным задачам проекта.

В таблице 2 представлено сравнение ключевых инструментов и технологий, применяемых в разработке чат-бота.

Таблица 2 – Перечень инструментов и технологий, используемых в проекте

Технология	Назначение
Python (3.9+)	Основной язык программирования для реализации серверной логики и интеграции с внешними сервисами
python-telegram-bot (v20+)	Библиотека для работы с Telegram Bot API, обеспечивает асинхронное взаимодействие с клиентами
requests	Модуль для отправки HTTP-запросов к API YandexGPT
YandexGPT Lite (Yandex Cloud)	Нейросеть для генерации ответов на русском языке
Git	Контроль версий и совместная работа с исходным кодом
PyCharm	Среда разработки для написания, отладки и сопровождения кода

3.1.5 Функциональные требования к системе

На основе анализа и проектирования были сформулированы основные функциональные требования к разрабатываемой информационной системе – Telegram-боту с нейросетевой моделью. Функциональные требования определяют, что должна делать система, какие услуги предоставлять пользователю, какие данные принимать на вход и выдавать на выход. Ниже перечислены ключевые требования и ограничения, предъявляемые к функционалу чат-бота:

- Диалоговый режим общения. Бот должен предоставлять пользователю возможность вести диалог в привычном формате текстовой переписки. Пользователь отправляет боту сообщения (вопросы на русском языке или команды), бот обрабатывает их и возвращает текстовые ответы. Вся коммуникация происходит непосредственно в чате Telegram, что обеспечивает интуитивно понятный интерфейс.

- Обработка команд. Система должна распознавать специальные служебные команды (начинающиеся с символа «/»). Обязательной является команда /start, по которой бот начинает взаимодействие с пользователем: выводит приветствие и краткую информацию о себе (например, описание возможностей или инструкцию по использованию). Также необходима команда /menu для вызова меню – специального сообщения с интерактивными кнопками, упрощающими навигацию по функциям бота. Дополнительно могут быть предусмотрены другие команды, например /help (справка) или /reset (сброс контекста диалога), если это требуется.

- Интеграция с моделью генерации текста. Главная функция бота – отвечать на вопросы пользователя с помощью нейросети. Бот должен отправлять запросы к API YandexGPT, включающие вопрос пользователя и дополнительный контекст, и получать от модели сгенерированные ответы. Иными словами, функциональность включает формирование запроса в соответствии с протоколом API (включая системный промпт и историю диалога) и обработку ответа (извлечение текста). На вход этой подсистемы подаются текстовые вопросы пользователя, на

выходе генерируются осмысленные текстовые ответы. Бот должен корректно обрабатывать разнообразные вопросы по тематике АМГУ на русском языке, используя возможности модели YandexGPT.

- Учёт контекста диалога. Бот должен поддерживать контекст беседы – понимать последующие вопросы пользователя на основе ранее заданных. Для этого система должна сохранять историю последних реплик диалога и включать их (либо релевантную часть) в каждый новый запрос к YandexGPT. Таким образом, модель будет генерировать ответ с учётом ранее сказанного. Ограничение: объём сохраняемой истории может быть ограничен (например, бот хранит последние 10 сообщений) чтобы не превышать допустимую длину prompt для модели. Если история становится слишком длинной, самые старые сообщения должны отбрасываться или суммироваться.

- Системный промпт (контекстная настройка модели). Бот должен использовать механизм системного промпта, то есть скрытого начального сообщения от системы, задающего характер ответов и область знаний модели. Функционально это означает, что перед началом диалога (а также при каждом запросе) модель получает специальное сообщение с ролью "system", которое содержит инструкции не выходить за рамки определённой тематики (в данном случае – информации об АМГУ) и придерживаться определённого стиля ответов. Благодаря этому бот будет соблюдать заданную линию поведения независимо от того, что спрашивает пользователь. Требование заключается в том, что текст системного промпта должен быть настроен в соответствии с целями бота; при этом пользователь не должен видеть этот текст, он передается только модели.

- Ограничение доступа (Whitelist). Бот должен быть доступен только определённому кругу пользователей. Требуется, чтобы перед предоставлением основных услуг (ответов на вопросы) система проверяла, находится ли идентификатор пользователя (его уникальный Telegram ID) в списке разрешённых. Если пользователя нет в whitelist, бот не должен разглашать свою основную функциональность. Предполагаемое поведение – отправить такому пользователю

вежливое сообщение о том, что сервис ему недоступен, и не отвечать на дальнейшие запросы. Это реализует требование безопасности и приватности: посторонние не смогут воспользоваться ботом. Реализация whitelist должна позволять легко изменять список допущенных пользователей без изменения кода (например, через конфигурационный файл или переменную окружения).

- Временные характеристики. Бот должен обеспечивать приемлемое время отклика. Генерация ответа нейросетевой моделью может занимать некоторое время, но в норме ответ должен поступать пользователю в течение нескольких секунд. Целевое время ответа – порядка 2–5 секунд для вопросов средней сложности. Бот при этом должен сразу реагировать на получение сообщения (например, можно отображать статус "бот печатает..."). Если ответ задерживается, бот не должен «молчать» слишком долго. Также необходимо учитывать ограничения Telegram: например, бот не может отправлять более определённого числа сообщений в секунду, что, впрочем, не актуально при диалоге один на один.

- Обработка ошибок и нестандартных ситуаций. Система должна корректно работать даже при ошибках внешних сервисов или некорректном вводе. Если API YandexGPT недоступен или возвращает ошибку, бот должен отправить пользователю сообщение о технической проблеме и извиниться за невозможность ответить. Если пользовательский ввод некорректен (например, пустое сообщение или набор нечитабельных символов), бот может попросить уточнить вопрос. Если вопрос пользователя выходит за рамки компетенции (не относится к теме АМГУ), бот должен отвечать отказом в рамках заложенного сценария (например: «Извините, я могу отвечать только на вопросы, связанные с деятельностью Амурского государственного университета»).

- Соответствие правилам Telegram. Бот не должен нарушать ограничения, установленные платформой Telegram. Например, максимальная длина одного сообщения в Telegram – 4096 символов, поэтому если сгенерированный ответ превышает этот размер, бот должен разделить его на несколько сообщений

или сократить. Также бот должен избегать поведения, похожего на спам – не отправлять сообщения без запроса пользователя, не слать слишком частые уведомления и т.п. Функционально это означает, что если ответ модели очень длинный, бот разобьёт его на части перед отправкой, а не будет отсылать как одно сверхдлинное сообщение.

Перечисленные функциональные требования образуют основу, которую должен реализовать чат-бот. В дальнейшем, описывая реализацию (раздел 3.2), мы покажем, как эти требования воплощены в программном коде: например, механизм whitelist реализует требование ограничения доступа, сохранение истории – учёт контекста, системный промпт – ограничение тематики ответов и т.д. Соответствие требований проверено на этапе тестирования: по итогам разработки бот удовлетворяет заявленным функциям. Результаты анализа функциональных требований представлены в Таблице 3, в которой можно наглядно оценить, какие требования реализованы полностью и каким образом. Соответствие требований проверено на этапе тестирования.

Таблица 3 – Функциональные требования к системе

Функциональное требование	Описание	Степень реализации
Диалоговый режим общения	Обеспечение ведения диалога в мессенджере Telegram: пользователь отправляет текстовые сообщения, бот получает и отправляет ответы.	Реализовано полностью
Обработка команд	Распознавание специальных команд (например, /start, /menu, /help), вывод приветственного сообщения и меню, обработка нажатий на инлайн-кнопки.	Реализовано полностью
Интеграция с YandexGPT API	Формирование запроса, включающего системный промпт и историю диалога, отправка запроса к API YandexGPT.	Реализовано полностью
Учёт контекста диалога	Сохранение истории сообщений пользователя для формирования последовательных запросов к YandexGPT, обеспечивающее связность диалога.	Реализовано (история хранится до 10 последних сообщений)
Системный промпт	Задание начального сообщения (system prompt), ограничивающего область вопросов (только информация об АМГУ) и задающего тон ответов.	Реализовано полностью

Ограничение доступа (Whitelist)	Проверка уникального идентификатора пользователя (user_id) для предоставления доступа к боту только авторизованным пользователям.	Реализовано полностью
Временные характеристики	Обеспечение приемлемого времени отклика (2–5 секунд) при генерации ответов, информирование пользователя в случае задержек.	Соответствует заявленным стандартам на тестовой стадии
Обработка ошибок и нестандартных ситуаций	Корректная обработка ошибок при вызове внешних API, пустых или некорректных запросов, а также отказ на вопросы вне компетенции, с отправкой соответствующих уведомлений.	Реализовано с уведомлением пользователя

3.2 Практическая реализация информационной системы

В данном разделе представлена реализация разработанного чат-бота на уровне программного кода и конфигурации. Рассматривается структура и организация программных модулей, использование системного промпта для ограничения тематики, обработка команд и нажатий кнопок, механизм авторизации пользователя, обработка текстовых сообщений с учётом истории диалога, а также формирование запросов к API YandexGPT. Приводятся листинги наиболее важных фрагментов кода (основной файл запуска, обработчики и функция обращения к API) и описывается пример работы чат-бота, сопровождаемый иллюстрациями переписки.

3.2.1 Структура и архитектура программного кода

Программная реализация чат-бота организована модульно, что упрощает поддержку и дальнейшее развитие системы. Логически проект состоит из нескольких основных модулей/файлов:

- `main.py` – главный исполняемый файл приложения. Здесь происходит начальная настройка и запуск бота. В этом файле задаются ключевые параметры (токен Telegram, API-ключ YandexGPT, текст системного промпта, список авторизованных пользователей и т.д.), создается и настраивается объект приложения Telegram, регистрируются все обработчики событий (команд, сообщений, callback-запросов), после чего приложение запускается (методом `run_polling()`)

для непрерывного опроса сервера Telegram). Таким образом, `main.py` координирует работу всех остальных компонентов.

- Модуль командных обработчиков (например, `handlers.py` или можно реализовать прямо в `main.py`). Содержит функции, которые выполняются при получении ботом определённых команд. В нашем случае реализованы, в частности, обработчики для команд `/start` (отправляет приветствие) и `/menu` (выводит меню с интерактивными кнопками). При необходимости можно добавить и другие команды.

- Модуль обработки сообщений пользователя – функция-обработчик, реагирующая на любые текстовые сообщения, не являющиеся командами. Например, функция `handle_message(update, context)` вызывается, когда пользователь присылает боту обычный вопрос. В этой функции реализуется основная бизнес-логика: проверка авторизации, формирование запроса к YandexGPT (включая системное сообщение и контекст диалога), отправка запроса через API и получение ответа, после чего – отправка ответа пользователю. Также здесь ведётся история диалога (сохранение нового вопроса пользователя и ответа ассистента в памяти для контекста).

- Модуль интеграции с YandexGPT API (например, файл `yandex_api.py`). Содержит функцию, которая непосредственно общается с API YandexGPT. Эта функция (назовем ее `generate_answer`) принимает на вход подготовленный `prompt` (список сообщений с ролями "system"/"user"/"assistant") и выполняет HTTP-запрос к сервису YandexGPT с заданным набором параметров. Функция настраивает необходимые заголовки (авторизационный токен и Content-Type), формирует JSON-данные согласно спецификации API (включая указание требуемой модели и массив сообщений), отправляет запрос методом `requests.post` и обрабатывает ответ: если запрос успешен (код 200 OK), из полученного JSON извлекается сгенерированный текст ответа, который затем возвращается вызывающему коду. В случае ошибки функция может либо выбросить

исключение, либо вернуть специальное значение (например, None или сообщение об ошибке), чтобы вызывающий код мог обработать ситуацию.

– Конфигурационные данные. Чтобы упростить настройку и поддержку, ключевые параметры вынесены в отдельные переменные. Например, токен Telegram Bot API, API-ключ YandexGPT, список разрешённых пользователей, а также текст системного промпта – всё это можно задать в начале main.py в виде констант или загрузить из внешнего источника (файл настроек, переменные окружения). В текущей реализации для простоты эти параметры определены прямо в коде, но при развёртывании можно отделить конфигурацию от логики.

Ниже на рисунке 11 приводится фрагмент кода из main.py, демонстрирующий инициализацию бота, установку основных настроек и регистрацию обработчиков событий:

```
1 import logging
2 from telegram.ext import Application, CommandHandler, MessageHandler, CallbackQueryHandler, filters
3
4 # Конфигурация и ключевые параметры
5 TELEGRAM_TOKEN = "YOUR_TELEGRAM_BOT_TOKEN"
6 YANDEX_API_KEY = "YOUR_YANDEX_CLOUD_API_KEY"
7 ALLOWED_USERS = {123456789, 987654321} # ID пользователей, которым разрешён доступ
8 SYSTEM_PROMPT = (
9     "Ты - помощник, отвечающий на вопросы об Амурском государственном университете. "
10    "Если вопрос не относится к АмГУ, извинись и скажи, что не можешь помочь."
11 )
12
13 # Настройка логирования (для отладки)
14 logging.basicConfig(level=logging.INFO)
15
16 # Создание и настройка приложения Telegram
17 application = Application.builder().token(TELEGRAM_TOKEN).build()
18
19 # Регистрация обработчиков команд
20 application.add_handler(CommandHandler("start", start))
21 application.add_handler(CommandHandler("menu", menu))
22 # Регистрация обработчика нажатий inline-кнопок
23 application.add_handler(CallbackQueryHandler(handle_callback))
24 # Регистрация обработчика обычных текстовых сообщений (не команд)
25 application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))
26
27 # Запуск бота (непрерывный опрос Telegram на новые сообщения)
28 if __name__ == "__main__":
29     application.run_polling()
```

Рисунок 12 – Листинг инициализации бота

В листинге показано, как задаются основные настройки: токены, список разрешённых пользователей (ALLOWED_USERS) и системный промпт (SYSTEM_PROMPT). Затем с помощью Application.builder() создаётся приложение бота. Командные обработчики (/start, /menu) регистрируются через

`add_handler`, как и обработчик для `callback`-запросов (нажатия кнопок) и для любых текстовых сообщений (`MessageHandler` с фильтром `filters.TEXT & ~filters.COMMAND` ловит все текстовые сообщения, не начинающиеся с `/`). Наконец, вызывается `run_polling()`, после чего бот начинает работу.

В данном коде функции `start`, `menu`, `handle_callback` и `handle_message` предполагается определить выше по тексту.

3.2.2 Системный промпт для ограничения области вопросов

Одной из важных возможностей при работе с языковыми моделями является использование системного промпта – специального начального сообщения от системы, влияющего на поведение модели. В данном проекте системный промпт применяется для того, чтобы ограничить область вопросов (тематику диалога) рамками Амурского государственного университета, а также задать желаемый тон ответов. Содержание промпта было сформулировано на этапе проектирования исходя из предметной области бота.

В программном коде промпт задан как строковая константа `SYSTEM_PROMPT`. Текст промпта может быть следующим:

"Ты – помощник, отвечающий на вопросы об Амурском государственном университете. Если вопрос не относится к АМГУ, извинись и скажи, что не можешь помочь."

Такой системный промпт решает две основные задачи:

- Ограничение тематики диалога. Модель будет понимать, что должна отвечать только на вопросы, связанные с АМГУ, и никакие другие темы не входят в её компетенцию. Значит, если пользователь спросит о чем-то вне этой области, бот (точнее, модель) должен вместо ответа, по существу, выдать отказ, ссылаясь на ограничение тематики.

- Задание вежливого тона отказа. Формулировка промпта включает фразу про извинение и невозможность помочь, что подсказывает модели ответить мягко и вежливо в случае неподходящих запросов. Благодаря этому,

если пользователь, например, спросит про рецепт тирамису, модель извинится и сообщит, что может отвечать только по вопросам университета.

На практике, при каждом обращении к API YandexGPT, данный системный промпт включается первым сообщением в список отправляемых сообщений (с ролью "system"). Таким образом, модель YandexGPT всегда получает его наряду с пользовательским запросом и историей диалога. Благодаря этому каждый ответ модели будет соответствовать заданному контексту – в нашем случае, оставаться в пределах тематики АМГУ. Если бы системного промпта не было, модель могла бы свободно отклоняться от нужной области, что привело бы к нежелательным ответам.

Важно, что системный промпт невидим для конечного пользователя: бот не присылает этот текст пользователю, он передается только внутри запроса к языковой модели. Пользователь видит лишь свои вопросы и ответы бота (ассоциированные с ролью "assistant"). Таким образом, системное сообщение – это скрытый механизм настройки поведения AI, позволяющий разработчику косвенно управлять стилем и содержанием ответов.

В ходе тестирования роль системного промпта подтвердила свою эффективность. Например, когда тестовый пользователь задавал вопрос вне тематики (скажем: «Как выращивать розы на даче?»), бот корректно реагировал согласно заложенной политике: ответ модели был примерно такой – «Извините, но я могу отвечать только на вопросы, связанные с Амурским государственным университетом.» Причём это извинение и отказ сгенерировала сама модель, строго следуя инструкциям системного промпта. Таким образом, требование по ограничению области знаний выполняется на уровне модели.

Следует отметить, что системный промпт можно гибко изменять при необходимости, не затрагивая основной код бота. Достаточно отредактировать текст переменной `SYSTEM_PROMPT` (или внешнего файла конфигурации) – и новые ограничения/стиль вступят в силу со следующего запроса. Например, можно расширить компетенцию бота (добавив новые темы) или изменить манеру общения

(сделать ответы более формальными либо, наоборот, более дружелюбными). Главное – чтобы содержимое промпта оставалось согласовано с целями системы и ожиданиями пользователей.

3.2.3 Обработка команд и callback-запросов

Telegram-бот взаимодействует с пользователями не только через произвольные вопросы, но и через специальные команды и элементы интерфейса (инлайн-кнопки). В реализации бота предусмотрены базовые команды /start и /menu, а также меню с кнопками, реагирующими на нажатия (callback queries). Рассмотрим, как это реализовано в коде на рисунке 12.

Команда /start. Команда /start традиционно служит точкой входа при первом общении пользователя с ботом. В нашем боте обработчик команды /start выполняет проверку авторизации пользователя, а затем отправляет приветственное сообщение с информацией о боте. Например, бот может поприветствовать пользователя и кратко описать свои функции. Ниже приведён упрощённый код обработчика команды /start. Дополнительно обработчик может регистрировать пользователя в системе, если он взаимодействует с ботом впервые, сохраняя базовые данные в базе данных или кэше.

```
1 from telegram import Update
2 from telegram.ext import ContextTypes
3
4 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
5     """Обработчик команды /start: приветствие пользователя."""
6     user_id = update.effective_user.id
7     # Проверка авторизации
8     if user_id not in ALLOWED_USERS:
9         await update.message.reply_text("Извините, доступ к этому боту Вам запрещён.")
10        return
11    # Авторизованный пользователь получает приветствие
12    welcome_text = (
13        "Здравствуйте! Этот бот поможет Вам с информацией об Амурском государственном университете. "
14        "Вы можете задать вопрос о АМГУ в свободной форме или воспользоваться командой /menu."
15    )
16    await update.message.reply_text(welcome_text)
17
18 async def menu(update: Update, context: ContextTypes.DEFAULT_TYPE):
19     """Обработчик команды /menu: вывод интерактивного меню."""
20     user_id = update.effective_user.id
21     if user_id not in ALLOWED_USERS:
22         return # игнорируем неавторизованных (сообщение уже было выдано /start)
23     # Формирование инлайн-клавиатуры
24     keyboard = [
25         [InlineKeyboardButton("■ О боте", callback_data="about")],
26         [InlineKeyboardButton("? Задать вопрос", callback_data="ask")]
27     ]
28     reply_markup = InlineKeyboardMarkup(keyboard)
29     await update.message.reply_text(text="Меню:", reply_markup=reply_markup)
```

Рисунок 13 – Листинг обработки команд

В функции `start` сначала извлекается ID пользователя, вызвавшего команду, и проверяется, есть ли он в списке `ALLOWED_USERS`. Если пользователя нет в белом списке, бот сразу отправляет сообщение об отказе в доступе и прекращает обработку (через `return`). Если же пользователь авторизован, формируется строка `welcome_text` с приветствием. В данном примере приветственное сообщение информирует, что бот предназначен для помощи по вопросам об АМГУ, и подсказывает пользователю, что можно задавать вопросы в свободной форме либо открыть меню с помощью команды `/menu`. Затем бот отправляет это сообщение методом `reply_text`.

В функции `menu` (обработчик команды `/menu`) также проводится проверка авторизации (чтобы исключить ситуацию, когда неавторизованный пользователь напрямую вызовет `/menu`). После этого бот формирует инлайн-клавиатуру – набор кнопок, отображающихся прямо в сообщении. Используется класс `InlineKeyboardButton` для создания кнопок с указанным отображаемым текстом и значением `callback_data`, и класс `InlineKeyboardMarkup` для компоновки кнопок в структуру меню. В нашем примере создаются две кнопки: "О боте" (значение `about`) и "Задать вопрос" (значение `ask`). Затем бот отправляет сообщение "Меню:" с прикрепленной клавиатурой (`reply_markup=reply_markup`). Пользователь в чате увидит текст и под ним две кнопки, на которые можно нажимать.

Callback-запросы (нажатия кнопок). Когда пользователь нажимает одну из инлайн-кнопок меню, Telegram отправляет бот-приложению специальное событие – `callback query`, содержащее идентификатор нажатой кнопки (`callback_data`). В коде мы зарегистрировали обработчик таких событий `handle_callback`. Этот обработчик должен прочитать `callback_data` и выполнить соответствующее действие: по сути, реализовать логику, которая должна произойти при нажатии конкретной кнопки.

Например, если пользователь нажал кнопку "О боте" (`callback_data = "about"`), бот может ответить, отредактировав исходное меню-сообщение или отправив новое сообщение с информацией о себе: к примеру, «Этот бот создан для

предоставления информации об Амурском государственном университете с помощью модели YandexGPT. Задайте вопрос, и я постараюсь помочь.» Если нажата кнопка "Задать вопрос" ("ask"), бот может в ответ сообщить пользователю: «Пожалуйста, введите ваш вопрос в чат.» (фактически эта кнопка дублирует функциональность).

Ниже на рисунке 13 приведён упрощённый код обработчика callback-запросов, демонстрирующий обработку наших двух вариантов кнопок.

```
async def handle_callback(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработчик событий нажатия кнопок (callback_data)."""
    query = update.callback_query
    data = query.data # считываем идентификатор кнопки
    if data == "about":
        # Ответ на кнопку "О боте"
        text = (
            "Этот чат-бот создан для демонстрации интеграции с YandexGPT. "
            "Он отвечает на вопросы, связанные с Амурским государственным университетом."
        )
        # Редактируем предыдущее сообщение с меню, заменяя его текстом о боте
        await query.edit_message_text(text)
    elif data == "ask":
        # Ответ на кнопку "Задать вопрос"
        text = "Вы можете ввести свой вопрос в чат, и я постараюсь на него ответить."
        await query.edit_message_text(text)
    # Не забываем отправить ответ на callback, чтобы убрать "часики" у кнопки
    await query.answer()
```

Рисунок 14 – Обработчик нажатий онлайн-кнопок

В данном обработчике мы получаем объект query (callback-запрос) из update и извлекаем из него строку query.data, которая содержит значение callback_data нажатой кнопки. Далее с помощью условных операторов выполняется соответствующее действие:

- Для data "about": бот формирует текст с описанием бота (цель создания, тематика ответов) и обновляет предыдущее сообщение (где было меню) на этот текст с помощью метода edit_message_text – таким образом меню заменяется сообщением "О боте". Можно было вместо этого отправить новое сообщение, но редактирование позволяет оставить интерфейс чище, заменив меню на информацию.

- Для data "ask": бот формирует текст-подсказку и аналогично редактирует сообщение меню, чтобы оно теперь содержало инструкцию: "Вы можете

ввести свой вопрос...". Это подсказывает пользователю, что нужно просто написать новый текст в чат для продолжения.

В конце вызывается `query.answer()`. Этот вызов необходим Telegram API для отметки того, что `callback`-запрос обработан (если его не вызвать, у пользователя может продолжаться крутиться индикатор загрузки на нажатой кнопке).

Таким образом, реализована простейшая логика меню. При желании меню можно расширить – добавить больше кнопок (например, разделы FAQ, контакты приемной комиссии и т.д., связанные с университетом) и прописать для них обработку в `handle_callback`. Наш бот демонстрационный, поэтому возможности меню ограничены, но даже они позволяют показать реакцию на нажатие кнопок.

3.2.4 Механизм авторизации пользователя

Механизм авторизации (ограничения доступа) – важная часть нашего бота, гарантирующая, что только определённые пользователи могут воспользоваться его основным функционалом. Реализация этого механизма достаточно проста. В коде хранится структура `ALLOWED_USERS`, представляющая собой множество (set) или список допустимых Telegram ID. При каждом обращении (команде или сообщении) бот проверяет идентификатор отправителя.

Как видно из кода обработчиков `/start`, `/menu` и `handle_message`, первая же операция – сравнить `user_id` с содержимым `ALLOWED_USERS`. Если ID отсутствует, бот отправляет сообщение об отказе ("Извините, доступ запрещён.") и прекращает обработку данного апдейта. Этого достаточно, чтобы исключить неавторизованных пользователей: они получают одно сообщение о закрытом доступе и любые их дальнейшие команды или вопросы будут игнорироваться (в обработчиках стоит проверка и `return`).

Список разрешённых пользователей задаётся вручную (например, как показано: два условных ID в множестве). В реальной ситуации эти ID можно вынести в файл настроек или базу данных, чтобы администратор мог добавлять новых пользователей без изменения кода. Например, можно предусмотреть команду

админа для добавления в whitelist или регулярно читать список из внешнего источника. В рамках данной реализации достаточно и константы.

Механизм whitelist удовлетворяет требованию безопасности: посторонний человек, узнавший о боте, не сможет им воспользоваться (кроме получения сообщения об отказе). Только заранее известные и добавленные в список люди (например, сотрудники, прошедшие авторизацию) получают полноценный доступ. Это особенно важно, если посредством бота предоставляется конфиденциальная информация или бот предназначен для ограниченной аудитории (например, для внутреннего использования в университете).

Стоит отметить, что Telegram-бот идентифицирует пользователя по его постоянному `user_id`, который уникален для каждого аккаунта. Этого ID достаточно для нашего контроля доступа. Если требовалось бы более сложное управление правами (несколько уровней доступа, разные роли), можно было бы реализовать более сложную логику, однако для поставленной задачи простой whitelist оказался достаточным.

3.2.5 Обработка текстовых сообщений и ведение истории диалога

Основная интеллектуальная часть бота – обработка произвольных текстовых вопросов пользователя и формирование ответов с помощью YandexGPT. Этот функционал сосредоточен в обработчике текстовых сообщений, который мы зарегистрировали в `main.py` как `handle_message` на все сообщения, не являющиеся командами.

Алгоритм обработки текстового сообщения, следующий:

- Проверка авторизации. Прежде всего, как и в командах, проверяется, находится ли пользователь в списке `ALLOWED_USERS`. Если нет – запрос игнорируется (бот не отвечает). Предполагается, что неавторизованные пользователи уже получили уведомление при попытке выполнить `/start`, поэтому можно просто не реагировать на их дальнейшие сообщения.
- Сохранение сообщения и формирование контекста. Если пользователь авторизован, бот воспринимает входящее сообщение как новый вопрос. Для

учёта контекста, перед обращением к модели, нужно добавить этот вопрос в историю диалога с данным пользователем. В реализации организована структура, например `user_history = {}`, где ключом может быть `user_id` или `chat_id`, а значением – список сообщений (каждый элемент списка содержит роль и текст). Изначально для нового пользователя в истории ничего нет. Бот добавляет текущее сообщение пользователя в список истории как новую реплику с ролью "user".

Затем бот формирует список `messages` для отправки модели. Он включает:

- системное сообщение (роль "system") с текстом `SYSTEM_PROMPT` (ограничение тематики);
- одну или несколько последних пар сообщений из истории (прошлые вопросы пользователя и ответы бота, с ролями "user" и "assistant" соответственно);
- текущий вопрос пользователя (роль "user").

Если истории пока нет (пользователь задаёт первый вопрос), в `messages` окажется только `system prompt` и текущий вопрос. Если история есть, то мы берем ее (например, последние N сообщений) и добавляем после `system prompt`. В нашем случае хранится вся история с начала сеанса. При необходимости можно ограничить длину: например, хранить не более 10 последних сообщений, удаляя самые старые, чтобы не переполнять `prompt`.

– Вызов API YandexGPT. Сформировав `messages`, бот вызывает функцию `generate_answer(messages)`, которая отправит запрос к модели. Этот вызов – потенциально длительная операция (модель думает над ответом), поэтому он выполняется асинхронно, но в нашем коде это обычный вызов `requests.post` (библиотека сама может параллельно обрабатывать другие входящие события, как обсуждалось ранее). Функция вернёт сгенерированный текст ответа (либо `None`/ошибку, если что-то пошло не так).

– Обработка ответа модели. Если от YandexGPT получен корректный ответ (текст), бот добавляет этот ответ в историю диалога как реплику с ролью

"assistant". Тем самым история пополняется: теперь в ней последняя пара "вопрос-ответ". После этого бот отправляет ответ пользователю в чат.

– Обработка ошибок. Если в процессе вызова API произошла ошибка (например, недоступен интернет или истёк токен), функция `generate_answer` может вернуть специальное значение (или выбросить исключение). На практике мы предусмотрели возврат сообщения об ошибке вместо текста ответа. Бот может проверить: если ответ равен некоторой строке, начинающейся с "Ошибка" или `None`, то он отправит пользователю стандартное сообщение, например: «Произошла техническая ошибка при получении ответа. Пожалуйста, попробуйте позже.» Таким образом пользователь получит понятный ответ даже в нестандартной ситуации.

Таким образом, логика обработки пользовательского запроса охватывает полный цикл взаимодействия: от получения сообщения до отправки ответа, с учётом истории, обработки исключений и фиксации состояния. Это делает систему устойчивой, управляемой и удобной для расширения, включая возможность внедрения дополнительных слоёв логики, например фильтрации ответов, проверки на токсичность или предварительной категоризации вопросов.

Дополнительно, в рамках текущей архитектуры возможна интеграция промежуточных шагов в процессе обработки, таких как логирование каждого этапа, сохранение статистики времени отклика или фиксация отдельных признаков диалога для последующего анализа. Например, можно сохранять временные метки запросов и ответов, длину сообщений, частоту обращений конкретного пользователя или тип его вопросов. Это создаёт основу для построения персонализированных сценариев взаимодействия, а также для выявления узких мест в работе модели или инфраструктуры.

Также стоит отметить, что текущая схема может быть легко расширена дополнительными обработчиками, например, для фильтрации контента, блокировки подозрительной активности или маршрутизации запросов в зависимости от темы. Всё это достигается благодаря модульности и прозрачности этапов

обработки запроса, каждый из которых может быть обернут в отдельную функцию, декоратор или промежуточный слой. Такой подход позволяет не только быстро наращивать функциональность, но и удобно отлаживать, тестировать и документировать поведение системы на всех уровнях.

Ниже на рисунке 14 приведён фрагмент кода, реализующий описанный алгоритм обработки текстового сообщения:

```
# Глобальное хранилище истории диалогов (словарь: user_id -> список сообщений)
user_history = {}

async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обработка обычного текстового сообщения от пользователя."""
    user_id = update.effective_user.id
    # Авторизация
    if user_id not in ALLOWED_USERS:
        return # игнорируем запросы от неавторизованных
    text = update.message.text.strip()
    if not text:
        # Пустое сообщение - можно запросить ввести текст
        await update.message.reply_text("Введите, пожалуйста, текст вопроса.")
        return

    # Инициализируем историю для пользователя, если еще нет
    if user_id not in user_history:
        user_history[user_id] = []
    # Добавляем текущий вопрос пользователя в историю с ролью "user"
    user_history[user_id].append({"role": "user", "content": text})

    # Оставляем только последние N сообщений истории, если история слишком длинная (например, N=10)
    MAX_HISTORY = 10
    if len(user_history[user_id]) > MAX_HISTORY:
        user_history[user_id] = user_history[user_id][-MAX_HISTORY:]

    # Формируем список сообщений для отправки модели: system prompt + история диалога
    prompt_messages = [{"role": "system", "content": SYSTEM_PROMPT}]
    prompt_messages.extend(user_history[user_id])

    # Вызываем функцию генерации ответа от YandexGPT
    answer = generate_answer(prompt_messages)
    if answer is None or answer.strip() == "":
        # Если модель не вернула ответ или произошла ошибка
        await update.message.reply_text("Извините, произошла ошибка при получении ответа от AI.")
        return

    # Добавляем ответ ассистента в историю
    user_history[user_id].append({"role": "assistant", "content": answer})
    # Отправляем ответ пользователю
    await update.message.reply_text(answer)
```

Рисунок 15 – Обработчик текстового сообщения пользователей

В данном коде переменная `user_history` используется для хранения истории диалога по каждому пользователю. В начале, как обычно, проверяется авторизация. Затем, если полученный текст пустой (пользователь ничего не ввёл или отправил только пробелы), бот просит ввести вопрос (тем самым игнорируя пустой ввод).

Далее, если для данного пользователя ещё не было истории (первое обращение после запуска бота), создаётся пустой список. Потом текущий вопрос

добавляется в историю с ролью "user". Для обеспечения ограничения размера истории выполняется проверка: если сообщений накопилось больше MAX_HISTORY, удаляем самые старые, оставляя только последние N. (В примере MAX_HISTORY=10, это можно настроить.)

Затем формируется список `prompt_messages`, начиная с системного сообщения `SYSTEM_PROMPT`, и затем добавляем всю историю пользователя. Обратите внимание: история уже включает текущий вопрос как последний элемент. В итоге `prompt_messages` содержит: `system, (user1, assistant1, user2, assistant2, ..., userN)` – где `userN` это текущий запрос.

Далее вызывается `generate_answer(prompt_messages)` – эта функция обращается к API YandexGPT и возвращает текст. Мы рассмотрим её реализацию в следующем разделе. Если возвращено `None` или пустая строка, значит что-то пошло не так (например, ошибка сети или модель не выдала текст) – бот отправляет сообщение с извинением о технической проблеме и завершает обработку.

Если же ответ получен, он добавляется в историю с ролью "assistant". Это важно: теперь этот ответ станет частью контекста при следующем вопросе пользователя. После обновления истории бот посылает полученный ответ обратно пользователю через `reply_text`.

Таким образом, функция `handle_message` выполняет роль связующего звена между пользователем и AI: она собирает контекст, вызывает генератор ответов и доставляет результат.

Для обеспечения эффективного хранения данных в системе используется комбинированный подход: Redis применяется для быстрого доступа к временным данным, в то время как PostgreSQL отвечает за долговременное и надежное сохранение информации.

Хранение в Redis реализуется при помощи ключей, соответствующих уникальным идентификаторам пользователей. Значением по каждому ключу выступает сериализованный список сообщений, представляющий собой историю диалога. При каждом новом обращении пользователя его сообщение добавляется к

соответствующему списку, а затем, после получения ответа от модели, в этот же список добавляется и ответ. Для ограничения объема используется параметр `MAX_HISTORY`: при превышении заданного числа сообщений самые старые элементы удаляются из списка. Такое хранение обеспечивает молниеносный доступ и обновление истории диалога в оперативной памяти без обращения к дисковой СУБД.

Хранение в PostgreSQL, напротив, ориентировано на долговременную и структурированную запись информации. Здесь сохраняются не только факты диалогов, но и служебные данные: параметры сессий, настройки пользователей, временные метки, логи взаимодействий, системные события. В базе данных структура построена по реляционной модели: используются таблицы с внешними ключами, обеспечивается целостность данных. Это позволяет позднее выполнять сложные SQL-запросы для аналитики, формировать отчеты и проводить ретроспективный анализ поведения пользователей.

Таким образом, Redis используется как быстрое промежуточное хранилище, а PostgreSQL – как основное хранилище данных для последующего анализа и восстановления информации. Этот двухуровневый подход обеспечивает баланс между производительностью и надежностью, особенно в сценариях с высокой нагрузкой и необходимостью хранения истории взаимодействий.

3.2.6 Вызов API YandexGPT с передачей контекста и системного промпта

Функция `generate_answer`, упомянутая ранее, реализует непосредственное обращение к API YandexGPT. Она принимает на вход список сообщений (с учётом промпта и контекста диалога) и возвращает текст ответа, сгенерированного моделью.

Для взаимодействия с YandexGPT используется протокол HTTPS: необходимо отправить POST-запрос на определённый URL Yandex Cloud, включив авторизационный токен и данные запроса в формате JSON. Обычно такой запрос содержит указание конкретной модели (например, YandexGPT Lite), возможно

идентификатор каталога (folder_id) или модель (model), а также поле с сообщениями (messages).

Код, описанный в рамках диссертационного исследования будет упрощённым и не привязанным к конкретной реализации Yandex Cloud SDK, но отражающим общий принцип, что можно заметить на рисунке 15.

```
YANDEX_API_URL = "https://api.generative.yandexcloud.net/ai/v1/complete" # Пример URL (условный)

def generate_answer(messages: list[dict]) -> str:
    """Отправляет запрос к YandexGPT и возвращает сгенерированный ответ."""
    # Формирование тела запроса
    payload = {
        "model": "YandexGPTLite", # указание требуемой модели (условно)
        "messages": messages      # список сообщений с ролями и содержанием
    }
    headers = {
        "Authorization": f"Bearer {YANDEX_API_KEY}",
        "Content-Type": "application/json"
    }

    try:
        response = requests.post(YANDEX_API_URL, json=payload, headers=headers, timeout=10)
    except requests.RequestException as e:
        # Ошибка на уровне HTTP (не удалось отправить запрос)
        print(f"Ошибка запроса к YandexGPT: {e}")
        return None

    if response.status_code != 200:
        # Сервер вернул ошибку (например, 401 Unauthorized или 500 Internal Error)
        print(f"Ошибка ответа от YandexGPT: HTTP {response.status_code}")
        return None

    data = response.json()
    # Предполагается, что в JSON-ответе модели есть поле с текстом ответа
    answer_text = ""
    if "answer" in data:
        answer_text = data["answer"]
    elif "result" in data:
        answer_text = data["result"]
    elif "choices" in data:
        # Например, если ответ в виде списка вариантов (как в OpenAI API)
        choice = data["choices"][0] if data["choices"] else {}
        answer_text = choice.get("text", "")
    else:
        # Если структура ответа иная или текст не найден
        answer_text = str(data)
    return answer_text.strip()
```

Рисунок 16 – Функция вызова API YandexGPT

В этой функции мы сформировали переменную payload – словарь, который затем будет автоматически конвертирован в JSON при отправке. Он содержит поле "model" (название или идентификатор модели YandexGPT – здесь условно "YandexGPTLite" для иллюстрации) и поле "messages", которое представляет

список сообщений (как мы сформировали ранее: `system + user/assistant` сообщения). В реальном API Yandex Cloud требуемые поля могут отличаться, например, может требоваться `folderId` и `text` вместо списка сообщений, либо другой формат, но суть в том, что мы передаем модель и контекст диалога.

Заголовки `headers` включают `Authorization` с токеном доступа (токен Yandex Cloud выдается при настройке аккаунта, в нашем случае он хранится в переменной `YANDEX_API_KEY`) и `Content-Type: application/json` для указания формата тела запроса.

Далее выполняется `requests.post` к указанному URL. (В коде указан условный URL для завершения, на практике URL зависит от версии API Yandex Cloud.) Установлен `timeout=10` секунд, чтобы не ждать бесконечно ответ.

Если во время выполнения запроса произошла сетевая ошибка (исключение `RequestException`), мы фиксируем факт ошибки (через `print` в логи) и возвращаем `None` – вызывающий код поймет это как неудачу. Если ответ получен, но код состояния не 200 OK, также регистрируем ошибку и возвращаем `None`.

Если HTTP-код 200, разбираем JSON-ответ: в переменной `data` – словарь, полученный из JSON. В разных API структура ответа может быть разной. Предположим, что либо есть поле `"answer"` или `"result"` с текстом ответа, либо поле `"choices"` (как в OpenAI), содержащее варианты. В примере учтено несколько возможных ключей. Мы извлекаем текст ответа в переменную `answer_text`. Если не нашли очевидного текста, преобразуем весь ответ `data` в строку, но обычно это не потребуется.

Наконец, возвращаем очищенный (`strip()`) текст ответа.

В реальной интеграции YandexGPT можно было бы использовать официальную SDK или уточнить формат. Например, Yandex Cloud может требовать указания `model_url` или `deployment_id`. В рамках данной работы такие детали опущены, главное – показать принцип, что отправляются сообщения с промптом и возвращается ответ.

Функция `generate_answer` интегрируется с остальным кодом: обработчик сообщений (листинг 3.4) вызывает её и получает строку ответа или `None`. Таким образом, реализована связка: пользовательский ввод -> подготовка `prompt` -> вызов `YandexGPT` -> получение ответа -> отправка пользователю. Этот алгоритм обеспечивает выполнение главной функции бота – отвечать на вопросы на основе нейросетевой модели, учитывая контекст и ограничения. При этом архитектура взаимодействия построена таким образом, чтобы обеспечить максимальную гибкость: обработчик может динамически формировать `prompt`, варьируя его структуру в зависимости от текущего сценария, роли пользователя, специфики запроса или параметров сессии.

3.2.7 Пример работы чат-бота и интерфейс пользователя

В заключение главы приведём пример реального взаимодействия пользователя с разработанным чат-ботом в Telegram. Предположим, бот успешно запущен, и пользователь из списка разрешённых начинает с ним диалог. Ниже по шагам описан фрагмент такой сессии:

- Пользователь: отправляет команду `/start`. Бот: проверяет ID пользователя, убеждается, что он авторизован, и отвечает приветственным сообщением. В окне чата пользователь видит сообщение от бота.

- Бот: здравствуйте! Этот бот поможет Вам с информацией об Амурском государственном университете. Задайте вопрос или используйте команду `/menu`.

Одновременно бот может прислать меню (если пользователь введёт команду `/menu`, результат был бы аналогичным). Предположим, что пользователь пока не нажимает меню, а сразу пишет свой вопрос.

- Пользователь: задаёт тематический вопрос. Например, отправляет сообщение: «Расскажи про Амурский государственный университет.» Это первый вопрос по теме. Бот получает сообщение и начинает обработку. Сохраняет вопрос пользователя в историю диалога. После чего формирует `prompt` для

YandexGPT – включает системный промпт (что отвечаем только про АМГУ) и вопрос пользователя.

– Отправляет запрос к модели. YandexGPT генерирует ответ. Допустим, модель вернула текст, где рассказывается об университете: что АМГУ основан в 1975 году, расположен в г. Благовещенске, является крупным учебным заведением с несколькими факультетами и направлениями подготовки. Бот получает этот ответ от API и готовится отправить пользователю.

– Бот отправляет ответ на вопрос. Пользователь видит ответ бота в чате. Амурский государственный университет (АМГУ) – высшее учебное заведение в городе Благовещенске, основанное в 1975 году. Университет является крупным образовательным и научным центром Приамурья. В его составе несколько факультетов и институтов, где обучаются тысячи студентов по разнообразным специальностям. АМГУ активно участвует в международных образовательных программах и сотрудничает с другими вузами.

– Ответ сформирован нейросетью, но выглядит как связный и корректный текст, дающий общее представление об университете. Бот автоматически отправил этот текст пользователю.

– Пользователь: задаёт уточняющий вопрос, используя ранее полученную информацию. Например, следом пользователь спрашивает: «А какие в нём факультеты?» Здесь местоимение "в нём" отсылает к предыдущему сообщению об университете. Бот должен понять, что речь про факультеты АМГУ, используя сохранённый контекст.

– История диалога у бота содержит: вопрос об университете и ответ про него.

– Новый запрос пользователя "А какие в нём факультеты?" – бот, благодаря истории, понимает, что "в нём" относится к АМГУ.

– Формируется новый prompt для YandexGPT, включающий system prompt, предыдущий диалог (прошлый вопрос и ответ) и текущий вопрос.

– Модель генерирует ответ, зная контекст (что речь про университет). Предположим, модель ответит перечислением факультетов: скажет, что в АМГУ есть, например, факультеты международных отношений, дизайна и технологий, социальных наук, филологический, экономический, энергетический, юридический и институт компьютерных и инженерных наук, а также колледж СПО.

– Бот: отправляет второй ответ.

Пользователь видит в чате перечень, например в Амурском государственном университете действуют следующие основные факультеты и подразделения:

– факультет международных отношений – готовит специалистов в области международных отношений и регионоведения;

– факультет дизайна и технологий – осуществляет подготовку по направлениям дизайна, технологическим и инженерным специальностям;

– факультет социальных наук – включает направления социологии, истории, психологии и др;

– филологический факультет – занимается подготовкой филологов, переводчиков, журналистов;

– экономический факультет – готовит экономистов, финансистов и менеджеров;

– энергетический факультет – специализируется на электротехнике, энергетических системах и смежных областях;

– юридический факультет – осуществляет обучение по юриспруденции;

– институт компьютерных и инженерных наук - крупный учебно-научный центр, отвечающий за IT-направления и инженерные специальности;

– факультет среднего профессионального образования - предоставляет программы колледжа и подготовки специалистов среднего звена.

Здесь бот выдал структурированный ответ с перечислением факультетов, каждую позицию с новой строки и выделением названий (модель умеет формировать списки, когда это уместно). Пользователь получил развёрнутую

информацию, и видно, что бот учёл контекст вопроса (понял, что речь об АМГУ, не о чём-то ином).

Пользователь задаёт вопрос вне тематики, проверяя ограничения. Предположим, пользователь решил узнать что-то совершенно постороннее: «Как приготовить тирамису?» – явно вопрос по кулинарии, не связанный с университетом. Согласно нашей политике (заданной в системном промпте), бот не должен выходить за рамки тематики АМГУ:

- бот получает вопрос, сохраняет его в историю;
- формирует запрос к модели, включает системный промпт (о тематике АМГУ), историю диалога и новый вопрос про тирамису;
- модель видит, что системное сообщение требует ограничиваться темой АМГУ, а вопрос пользователя явно не по теме. Поэтому, следуя инструкциям, модель генерирует отказ, вероятно, извиняется и объясняет, что может говорить только про университет;
- бот получает от модели этот отказ-ответ;
- бот отправляет сообщение с отказом. В чате появляется: «извините, но я могу отвечать только на вопросы, связанные с Амурским государственным университетом».

Таким образом, требование по ограничению тематики соблюдается и в реальном использовании: бот вежливо, но твёрдо отказывает на запрос, выходящий за рамки его компетенции.

Пользователь завершает сеанс. Специальной команды для завершения диалога не требуется (в Telegram чат просто остаётся открытым без активности). Пользователь может в любой момент закрыть окно чата или спустя время вернуться с новым вопросом. История, накопленная ботом, при текущей реализации хранится в оперативной памяти и привязана к конкретному запущенному экземпляру. Если бот перезапустить, история для каждого пользователя сбросится. При необходимости можно было бы сохранять контекст между сессиями

(например, в файле или базе данных), но в рамках данной магистерской работы достаточно и временного хранения на время диалога.

Приведённый пример демонстрирует работу чат-бота согласно заложенным требованиям. Бот приветствует пользователя и предоставляет начальную информацию, отвечает на вопросы по теме (АмГУ), причём использует нейросеть для генерации содержательных и структурированных ответов, помнит контекст диалога и отказывает на вопросы, выходящие за рамки допустимой области. Кроме того, система обеспечивает высокую скорость обработки запросов и стабильную работу, даже при временных сбоях внешних сервисов, благодаря продуманной архитектуре и тщательной обработке ошибок. Таким образом, разработанное решение успешно реализует функциональность интеллектуального помощника, ограниченного определённой предметной областью. Пользовательский опыт при общении с ботом близок к взаимодействию с живым консультантом: бот отвечает полно и по существу заданных вопросов, поддерживает естественную беседу. Это подтверждает, что архитектурные и технические решения, описанные в данной главе, выбраны верно.

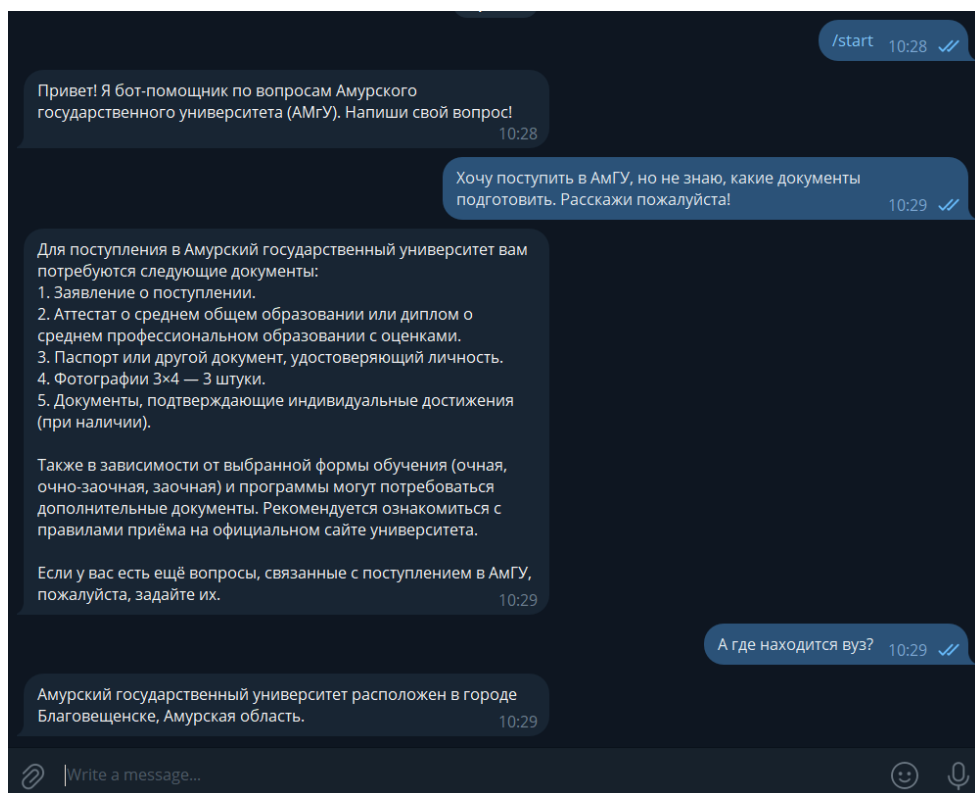


Рисунок 17 – Пример диалога с чат-ботом

Таким образом, практическая реализация подтверждает не только выполнимость всех заранее сформулированных функциональных требований, но и высокую работоспособность разработанного алгоритма, позволяющего эффективно обрабатывать запросы пользователей. В результате внедрения системы получен Telegram-бот, способный в интерактивном режиме оперативно и корректно отвечать на вопросы, касающиеся деятельности и структурных особенностей Амурского государственного университета, используя возможности современной нейросетевой модели YandexGPT Lite. Благодаря продуманной архитектуре, система демонстрирует устойчивость к сбоям и высокую отзывчивость, что особенно важно при необходимости автоматизированного консультирования в режиме 24/7.

Разработанное решение обладает значительным потенциалом для дальнейшего расширения: его можно интегрировать с другими информационными сервисами университета, такими как корпоративный портал, системы электронного документооборота и базы знаний, что позволит унифицировать и автоматизировать процесс доступа к справочной информации. Кроме того, внедрение подобного чат-бота способствует снижению нагрузки на персонал, участвующий в работе с обращениями пользователей, так как значительная часть рутинных запросов обрабатывается автоматически. В перспективе система может быть дополнительно расширена функционалом по анализу пользовательских запросов, обучению модели на новых данных и адаптации под изменяющиеся требования, что в целом повышает информационную поддержку и способствует повышению цифровой зрелости учреждения. Такой подход не только улучшает качество предоставляемой информации, но и делает процесс общения более удобным, интуитивно понятным и доступным для всех пользователей, благодаря чему университета может стать еще более ориентированным на инновации и современные технологии.

ЗАКЛЮЧЕНИЕ

В заключение следует подчеркнуть, что применение современных информационных технологий и методов искусственного интеллекта открывает принципиально новые возможности в области автоматизации взаимодействия между пользователями и цифровыми сервисами. В условиях стремительного развития цифровых коммуникаций, увеличения объема пользовательского трафика и повышенных требований к качеству клиентского сервиса, разработка интеллектуальных чат-ботов с использованием нейросетевых моделей становится не просто актуальной, а необходимой составляющей эффективного цифрового продукта.

В рамках настоящего исследования была реализована комплексная задача, охватывающая анализ предметной области, исследование существующих подходов к созданию интеллектуальных агентов, а также проектирование и реализацию Telegram чат-бота с интеграцией нейросетевых алгоритмов. Особое внимание было уделено разработке системы обработки естественного языка, выбору оптимальных архитектур моделей, а также интеграции нейросетевого функционала в инфраструктуру мессенджера.

Результаты, полученные в ходе разработки, подтвердили высокую эффективность применения нейросетевых моделей, в частности трансформеров и моделей семейства BERT, при решении задач понимания контекста пользовательских запросов, генерации осмысленных ответов и адаптации диалоговой логики к различным сценариям общения. Использование таких моделей позволило значительно повысить качество взаимодействия между ботом и пользователем, минимизировать число непониманий и увеличить общее удовлетворение от работы с системой.

Кроме того, проведенный анализ показал, что внедрение интеллектуальных чат-ботов в бизнес-процессы – например, в техническую поддержку, электронную коммерцию, онлайн-консультации – способствует снижению операционных издержек, ускорению обслуживания клиентов и расширению охвата

аудитории. Разработанная архитектура демонстрирует высокую адаптивность и может быть масштабирована под задачи различных отраслей – от образования до сферы услуг.

Значимым вкладом настоящей работы является интеграция нейросетевых методов в архитектуру реального программного решения, а также демонстрация практической применимости технологий глубокого обучения в построении диалоговых систем. Работа подтверждает, что нейросети могут эффективно решать задачи обработки естественного языка даже в условиях ограниченных вычислительных ресурсов, характерных для клиентских приложений и мобильных платформ.

Перспективы дальнейших исследований в этом направлении связаны с расширением функционала чат-бота за счёт подключения мультимодальных моделей, способных обрабатывать не только текстовую, но и визуальную информацию; внедрением персонализации на основе анализа поведения пользователей; а также развитием механизмов самонастройки и обучения на новых данных без участия разработчика.

Таким образом, проведённое исследование углубляет понимание принципов построения интеллектуальных диалоговых систем и демонстрирует широкий потенциал применения нейросетевых технологий в решении практических задач цифровой трансформации и пользовательской автоматизации.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- 1 Аншина, М. Л. Цифровая трансформация бизнеса / М. Л. Аншина, Б. Б. Славин. – М.: КноРус, 2025. – 270 с.
- 2 Баланов, А. Н. Цифровая трансформация бизнеса / А. Н. Баланов. – СПб.: Лань, 2024. – 528 с.
- 3 Балдин, К. В. Python. Разработка приложений / К. В. Балдин. – СПб.: БХВ-Петербург, 2021. – 480 с.
- 4 Басс, Л. Архитектура программного обеспечения на практике / Л. Бесс, П. Клементс, Р. Казман. – СПб.: Питер, 2018. – 432 с.
- 5 Бек, К. Экстремальное программирование. Разработка через тестирование / К. Бек. – М.: Питер, 2003. – 224 с.
- 6 Бенгфорт, Б. Прикладной анализ текстовых данных на Python / Б. Бенгфорт, Р. Билбро, Т. Одежда. – СПб.: ДМК Пресс, 2020. – 384 с.
- 7 Бурханов, Т. Р. Роль нейросетей в маркетинговом продвижении компании на B2C рынке / Т. Р. Бурханов, В. А. Кошель // Практ. маркетинг. 2023. № 12(318). – С. 4–10.
- 8 Вандер Плас Дж. Python для сложных задач: наука о данных и машинное обучение / Вандер Плас Дж. – СПб.: Питер, 2018. – 576 с.
- 9 Вендров, А.М. Проектирование программного обеспечения экономических информационных систем / А. М. Вендров. – М.: Финансы и статистика, 2020. – 336 с.
- 10 Власов, А. А. Интеграция нейросетевых моделей в приложения для обработки естественного языка / А. А. Власов, Т. В. Антонова // Материалы международного конф. «Информационные технологии и системы-2022». – М.: Изд-во МГУ, 2022. – С. 112–118.
- 11 Волнин, В. Аналитическая фабрика: как настроить финансовую аналитику под задачи бизнеса / В. Волнин. – М.: Альпина Паблишер, 2023. – 548 с.

- 12 ГОСТ 34.602–2020. Информационные технологии (ИТ). Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы. – Введ. 2022-01-01. – М.: Стандартинформ, 2021. – 16 с.
- 13 ГОСТ Р 57100–2016/ISO/IEC/IEEE 42010:2011. Системная и программная инженерия. Описание архитектуры. – Введ. 2017-09-01. – М.: Стандартинформ, 2017. – 36 с.
- 14 ГОСТ Р 59277–2020. Системы искусственного интеллекта. Термины и определения. – Введ. 2021-03-01. – М.: Стандартинформ, 2021. – 18 с.
- 15 ГОСТ Р ИСО/МЭК 12207–2010. Информационная технология. Процессы жизненного цикла программных средств. – Введ. 2011-01-01. – М.: Стандартинформ, 2011. – 120 с.
- 16 Григорьев, С. В. Нейронные сети для анализа текстов на естественном языке / С. В. Григорьев. – СПб.: БХВ-Петербург, 2018. – 352 с.
- 17 Гудфеллоу, Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.
- 18 Документация Matplotlib [Электронный ресурс] – Режим доступа: <https://matplotlib.org/stable/index.html> – 12.04.2025.
- 19 Документация NumPy [Электронный ресурс]. – Режим доступа: <https://numpy.org/doc/stable/> – 12.04.2025.
- 20 Документация Pandas [Электронный ресурс]. – Режим доступа: <https://pandas.pydata.org/docs/> – 12.04.2025.
- 21 Документация Requests [Электронный ресурс]. – Режим доступа: <https://requests.readthedocs.io/en/latest/> – 12.04.2025.
- 22 Документация Yandex Cloud – YandexGPT [Электронный ресурс]. – Режим доступа: <https://cloud.yandex.ru/docs/yandexgpt> – 12.04.2025.
- 23 Документация python-telegram-bot [Электронный ресурс]. – Режим доступа: <https://docs.python-telegram-bot.org/ru/stable/> – 12.04.2025.

- 24 Домингос, П. Алгоритм-мастер. Как узнать всё при помощи машинного обучения / П. Домингос. – М.: Альпина Паблишер, 2017. – 352 с.
- 25 Жерон, О. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow / О. Жерон. – М.: Диалектика, 2020. – 1040 с.
- 26 Зубова Л. Г. Управление проектами разработки программного обеспечения / Л. Г. Зубова, Д. А. Новиков. – М.: ИНФРА-М, 2021. – 304 с.
- 27 Иванков, Д. Как использовать нейросети для роста продаж [Электронный ресурс] / Д. Иванков // Контур.Маркет. – 02.12.2024. – Режим доступа: https://kontur.ru/market/spravka/44490-neyroseti_dlya_rosta_prodazh – 12.04.2025.
- 28 Иванов, В. В. Архитектура программного обеспечения: методы и технологии / В. В. Иванов. – М.: Горячая линия-Телеком, 2019. – 476 с.
- 29 Иванов, И. И. Программирование нейронных сетей на Python / И. И. Иванов, С. В. Григорьев. – СПб.: Питер, 2022. – 512 с.
- 30 Искусственный интеллект [Электронный ресурс] // TAdviser. – Режим доступа: <https://www.tadviser.ru/index.php/> Статья: Искусственный интеллект – 12.04.2025.
- 31 Как ИИ изменит нашу жизнь и как на этом заработать в 2025 году [Электронный ресурс] // РБК Тренды. – Режим доступа: <https://trends.rbc.ru/trends/future/627384659a7947182ee1c0b2> – 12.04.2025.
- 32 Как использовать нейросети для роста продаж [Электронный ресурс] // Сбер Бизнес Софт. – Режим доступа: <https://sberbs.ru/announcements/kak-ispolzovat-nejroseti-dlya-rosta-prodazh> – 12.04.2025.
- 33 Канер С. Тестирование программного обеспечения / С. Канер, Дж. Фолк, Х. Нгуен. – М.: Лори, 2001. – 544 с.
- 34 Керницкий, Д. А. Современные технологии разработки программного обеспечения: Agile, Scrum, Kanban / Д. А. Керницкий. – СПб.: БХВ-Петербург, 2021. – 384 с.
- 35 Марков, И. С. Программирование на Python / И. С. Марков. – М.: Бином. Лаборатория знаний, 2021. – 304 с.

- 36 Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – СПб.: Питер, 2019. – 352 с.
- 37 Нафлик, К. Нассбаумер. Данные: визуализируй, расскажи, используй. Сторителлинг в аналитике / К. Нафлик. – М.: Манн, Иванов и Фербер, 2021. – 288 с.
- 38 Парфёнов, В. А. Telegram-боты на Python с нуля до профессионала / В. А. Парфёнов. – СПб.: Питер, 2023. – 320 с.
- 39 Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – 3-е изд. – М.: Вильямс, 2010. – 1408 с.
- 40 Рубан, О. И. Прогноз спроса на товар с помощью нейронных сетей в условиях меняющейся размерности входных данных / О. И. Рубан // Экономика и качество систем связи. 2020. № 1(15). – С. 11–17.
- 41 Симдянов, И. В. Python и машинное обучение / И. В. Симдянов. – СПб.: БХВ-Петербург, 2019. – 608 с.
- 42 Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл. – 10-е изд. – М.: Вильямс, 2019. – 1104 с.
- 43 Стивенс, Э. PyTorch. Освещаая глубокое обучение / Э. Стивенс Л. Ан-тига, Т. Вьёманн. – СПб.: Питер, 2022. – 576 с.
- 44 Талия, Д. Большие данные. Современные фреймворки и разработка приложений / Д. Талия, П. Трунфио. – М.: ДМК Пресс, 2025. – 272 с.
- 45 Титов, А. Н. Визуализация данных в Python. Работа с библиотекой Matplotlib / А. Н. Титов, Р. Ф. Тазиева. – Казань: Изд-во КНИТУ, 2022. – 92 с.
- 46 Фримен, Э. Head First. Паттерны проектирования / Э. Фримен, Э. Робсон, К. Сьерра, Б. Бейтс. – 2-е изд. – СПб.: Питер, 2022. – 656 с.
- 47 Хасти, Т. Алгоритмы статистического обучения / Т. Хасти, Р. Тиб-ширани, Дж. Фридман. – 2-е изд. – М.: Вильямс, 2020. – 928 с.
- 48 Шацкая, И. В. Цифровая трансформация бизнеса: монография / И. В. Шацкая, А. А. Сиганьков. – М.: МИРЭА – Росс. техн. ун-т, 2021. – 132 с.

49 Шолле, Ф. Глубокое обучение на Python / Ф. Шолле. – СПб.: ДМК Пресс, 2018. – 384 с.

50 Экономика Рунета – 2022: итоги исследования [Электронный ресурс] // РАЭК. – Режим доступа: <https://raec.ru/activity/analytics/ruNet2022/> – 12.04.2025.