МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра информационных систем управления

ДЮБКОВА АНГЕЛИНА ЮРЬЕВНА

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ВОПРОСНО-ОТВЕТНЫХ СИСТЕМ И ЧАТ-БОТОВ

Дипломная работа
студента 4 курса 12 группы

Научный руководитель:
старший преподаватель
Рубашко Наталья Константиновна

Допущена к защите
“___” _____ 2025 г.
Зав. кафедрой информационных систем управления,
доктор технических наук, доцент А.М. Недзьведь

Минск
2025

СОДЕРЖАНИЕ

Перечень условных обозначений

API	— Application Programming Interface (интерфейс прикладного программирования)
BERT	— Bidirectional Encoder Representations from Transformers
BM25	— Best Matching 25 (алгоритм текстового поиска)
DPR	— Dense Passage Retrieval (модель плотного поиска)
GPT	— Generative Pre-trained Transformer (генеративный предварительно обученный трансформер)
IR	— Information Retrieval (информационный поиск)
KB	— Knowledge Base (база знаний)
LLM	— Large Language Model (крупная языковая модель)
ML	— Machine Learning (машинное обучение)
NLP	— Natural Language Processing (обработка естественного языка)
NER	— Named Entity Recognition (распознавание именованных сущностей)
QA-система	— Вопросно-ответная система
SBERT	— Sentence-BERT (модификация BERT для работы с предложениями)
Seq2Seq	— Sequence-to-Sequence (архитектура последовательной генерации)
SPARQL	— SPARQL Protocol and RDF Query Language (язык запросов к графам RDF)
TF-IDF	— Term Frequency-Inverse Document Frequency
UI	— User Interface (пользовательский интерфейс)

РЕФЕРАТ

Дипломная работа, 58 с., 10 рис., 4 табл., 1 приложение.

Ключевые слова: чат-бот, вопросно-ответная система, обработка естественного языка, GPT, Python, машинное обучение, Seq2Seq, сравнение, архитектура.

Объект исследования — современные интеллектуальные программные системы, реализующие диалоговое взаимодействие: чат-боты и вопросно-ответные системы.

Цель работы — провести сравнительный анализ архитектурных, функциональных и технологических различий между чат-ботами и вопросно-ответными системами, а также реализовать прототип чат-бота и провести оценку его эффективности.

Методы исследования — а) теоретические: изучение научных и практических публикаций по теме обработки естественного языка, диалоговых систем, машинного обучения и генеративных моделей; б) практические: проектирование и реализация прототипов QA-системы и чат-бота, проведение тестирования с использованием различных пользовательских сценариев, анализ результатов.

Результатами являются: систематизированное сравнение подходов к построению чат-ботов и QA-систем;реализация прототипа чат-бота на основе правил и нейросетевой генерации с использованием моделей обработки естественного языка; проведение функционального тестирования и анализ точности обработки пользовательских запросов.

Область применения — автоматизированные системы поддержки пользователей, образовательные платформы, информационные ассистенты и справочные сервисы.

РЭФЕРАТ

Дыпломная праца, 58 с., 10 мал., 4 табл., 1 дадатак.

Ключавыя словы: чат-бот, пытальна-адказная сістэма, апрацоўка натуральнай мовы, GPT, Python, машыннае навучанне, Seq2Seq, параўнанне, архітэктура.

Аб'ект даследавання — сучасныя інтэлектуальныя праграмныя сістэмы, што рэалізуюць дыялогавае ўзаемадзеянне: чат-боты і пытальна-адказныя сістэмы.

Мэта працы — правесці параўнальны аналіз архітэктурных, функцыянальных і тэхналагічных адрозненняў паміж чат-ботамі і пытальна-адказнымі сістэмамі, а таксама рэалізаваць прататып чат-бота і ацаніць яго эфектыўнасць.

Метады даследавання:а) тэарэтычныя — вывучэнне навуковых і практычных публікацый па тэме апрацоўкі натуральнай мовы, дыялогавых сістэм, машыннага навучання і генератыўных мадэляў;б) практычныя — праектаванне і рэалізацыя прататыпаў QA-сістэмы і чат-бота, правядзенне тэставання з выкарыстаннем розных карыстальніцкіх сцэнараў, аналіз вынікаў.

Вынікі — сістэматызаванае параўнанне падыходаў да пабудовы чат-ботаў і QA-сістэм; рэалізацыя прататыпа чат-бота на аснове правіл і нейрасеткавага генеравання з выкарыстаннем мадэляў апрацоўкі натуральнай мовы; правядзенне функцыянальнага тэставання і аналіз дакладнасці апрацоўкі карыстальніцкіх запытаў.

Вобласць ужывання — аўтаматызаваныя сістэмы падтрымкі карыстальнікаў, адукацыйныя платформы, інфармацыйныя асістэнты і даведачныя сэрвісы.

ABSTRACT

Graduate work, 58 p., 10 illustrations, 4 table, 1 appendix.

Keywords: chat bot, question-and-answer system, natural language processing, GPT, Python, machine learning, Seq2Seq, comparison, architecture.

The object of the research is modern intelligent software systems that implement dialogue interaction: chat bots and question-and-answer systems.

The purpose is to conduct a comparative analysis of architectural, functional and technological differences between chat bots and question-and-answer systems, as well as to implement a prototype of a chat bot and evaluate its effectiveness.

Research methods - a) theoretical: study of scientific and practical publications on the topic of natural language processing, dialogue systems, machine learning and generative models; b) practical: design and implementation of prototypes of QA-system and chat bot, testing using different user scenarios, analysis of results.

The results are: systematised comparison of approaches to building chat bots and QA-systems; implementation of chat bot prototype on the basis of rules and neural network generation using natural language processing models; functional testing and analysis of accuracy of user requests processing.

Application area - automated user support systems, educational platforms, information assistants and reference services.

ВВЕДЕНИЕ

Современные технологии непрерывно меняют нашу повседневную жизнь, внося изменения во многие сферы деятельности, включая коммуникацию и взаимодействие с компьютерными системами. Одной из наиболее заметных и быстро развивающихся технологий в этом контексте являются чат-боты.

Существуют два способа взаимодействия — вопросно-ответные системы и чат-боты. Первоначально именно вопросно-ответные системы отвечали за обработку пользовательских запросов и предоставление информации в рамках заданных сценариев. Со временем вопросно-ответные системы уступили чат-ботам, и они начали очень быстро развиваться. Чат-боты более гибкие и могут общаться более человеческим языком, по сравнению с вопросно-ответными системами.

Чат-боты набрали большую популярность за последнее время благодаря своей многофункциональности. Сейчас чат-боты используются почти во всех сферах, таких как здравоохранение, развлечения и многие другие. Чат-боты могут представляться в разных видах, таких как мессенджеры, вебсайты и приложения. Чат-ботов можно использовать как личных ассистентов которые всегда под рукой и на связи. Они могут напомнить о задаче, записать к врачу, составить список покупок и многое другое.

Главные фактором влияющим на популярность чат-ботов является возможность чат-бота отвечать мгновенно в любое время. Чат-боты работают гораздо эффективнее благодаря возможности обрабатывать несколько запросов одновременно, из-за чего их очень удобно использовать в оптимизации бизнес-процессов. По этой причине многие компании для улучшения клиентской поддержки стали разрабатывать собственных чат-ботов и уделять этому большое значение.

Целью данной дипломной работы является разработка эффективной вопросно-ответной системы, которая может быть интегрирована в чат-боты для различных приложений. Для этого необходимо решить следующие задачи:

1) Провести анализ современных подходов и технологий, используемых для создания вопросно-ответных систем.

2) Изучить и сравнить алгоритмы обработки естественного языка и машинного обучения.

3) Разработать и протестировать прототип системы, способной обрабатывать запросы на естественном языке и предоставлять корректные ответы.

Для достижения этой цели рассмотрены актуальные исследования, примеры реализаций различных чат-ботов и разработан прототип чат-бота с помощью выбранных технологий и инструментов. Благодаря этому получены

практические навыки и более подробно разобраны принципы разработки чат-ботов.

Актуальность темы заключается в растущей популярности чат-ботов и увеличении их влияния на все сферы деятельности. Современные компании стремятся минимизировать время отклика и затраты на обслуживание клиентов, что делает использование чат-ботов и, в частности, вопросно-ответных систем, крайне востребованным решением. Кроме того, прогресс в области обработки естественного языка и появление мощных языковых моделей, таких как BERT и GPT, открывают новые возможности для разработки более интеллектуальных и точных чат-ботов.

Глава 1. QA-СИСТЕМЫ И ЧАТ-БОТЫ КАК СРЕДСТВА ВЗАИМОДЕЙСТВИЯ С КОМПЬЮТЕРОМ

1.1 Вопросно-ответные системы

1.1.1 Определение и классификация вопросно-ответных систем

Вопросно-ответные системы (Question Answering Systems, QA-системы) — это тип интеллектуальных программных систем, предназначенных для автоматического предоставления точных и релевантных ответов на заданные пользователем вопросы на естественном языке. Вопросно-ответные системы часто используются в поисковых системах, чат-ботах и образовательных технологиях, а также в многих других системах.

QA-системы используют для наиболее быстрых и точных ответов, без необходимости изучать документы. Развитие вопросно-ответных систем пошло их технологий обработки естественного языка и машинного обучения.

Классифицировать вопросно-ответные системы можно по нескольким параметрам, самый частый параметр классификации это область применения. По области применения вопросно-ответные системы разделяются на:

- **узкоспециализированные (Closed-domain QA)** системы. Такие системы используются для работы в конкретной области, например медицина или образование. Узкоспециализированные системы отличаются высокой точностью, потому что не распыляются на множество областей, а используют данные только одной конкретной области. Примером такой системы является **Wolfram Alpha**, способная отвечать на точные вопросы в области математики, физики и других наук, используя движок **Wolfram Mathematica;**
- **общего назначения (Open-domain QA)** системы, способные обрабатывать вопросы по различным темам без ограничений по тематике. Они чаще всего интегрируются в голосовых помощниках, поисковых системах и чат-ботах. Примером является **Яндекс Алиса**, которая может отвечать на широкий спектр запросов — от бытовых до навигационных и информационных.

По архитектуре и принципу работы QA-системы делятся на:

- **правил-основные (Rule-based QA)** — системы, в которых используются заранее заданные шаблоны, правила и логика сопоставления. Они были одними из первых типов QA-систем и до сих пор применяются в ситуациях, где важно детерминированное поведение и контроль над формулировкой ответов. Примером является система **START**, разработанная в MIT;
- **машинного обучения (ML-based QA)** — системы, использующие методы машинного и глубокого обучения. Они учатся на примерах и

способны адаптироваться к новым формулировкам вопросов. В этом подходе используются рекуррентные нейронные сети (например, LSTM), трансформеры (например, BERT, GPT) и другие модели для извлечения или генерации ответов;

- **ранжирующие (Retrieval-based QA)** — системы, которые находят наиболее релевантный фрагмент текста из существующей базы знаний. Обычно реализуются как двухэтапные модели использующие *retriever,* который находит подходящие документы и *reader*, который извлекает конкретный ответ из текста. Классическим примером является архитектура в системах **Haystack**, **DPR** (Dense Passage Retrieval) и **DrQA** ;
- **порождающие (Generative QA)** — системы, которые формируют ответ "с нуля", опираясь на модель языка. Они не просто находят текст, а **генерируют его**, часто с учётом диалога. Яркий пример — модели **GPT (OpenAI)** и **T5 (Google)**, использующие трансформерную архитектуру для генерации ответов на вопросы.

1.1.2 Архитектуры и методы разработки вопросно-ответных систем

Вопросно-ответные системы (Question Answering Systems, QA-системы) являются важным направлением в области интеллектуальной обработки естественного языка. Их основная задача — извлечение релевантной информации в ответ на пользовательский вопрос. Существуют различные подходы к реализации таких систем, среди которых наибольшее распространение получили: подход на основе правил и подход на основе методов машинного обучения. Рассмотрим архитектуры и методы разработки более подробно.

Подход на основе правил предполагает ручное задание логических и синтаксических шаблонов, по которым осуществляется сопоставление вопросов с потенциальными ответами. Подобные системы требуют значительной экспертной настройки, но обеспечивают высокую точность при узкой предметной области. Данный подход использует такие методы как регулярные выражения, тернарные выражения и S-правила.

Регулярные выражения используются для задания шаблонов подстрок, что позволяет находить предложения, соответствующие заданному образцу. Пример шаблона:* отличия .* библиотеки .* (C++ | C) .* (C++ | C) .*

Тернарные выражения — это структура в формате <объект отношение субъект>, используемая для формализации смыслового содержания предложений. К примеру, предложение *«Пётр подарил Марии книгу»* может

быть преобразовано в тернарные конструкции как <<Пётр подарить Мария> книга> и <книга принадлежит Мария>. При обработке вопросов система выполняет инверсию, преобразуя их в форму, удобную для последующего анализа.

Семантические правила (или S-правила) используются для унификации тернарных конструкций, различающихся по форме, но передающих одно и то же значение. К примеру, предложения *«Пётр подарил Марии книгу»* и *«Книга, которую Пётр подарил Марии»* формально отличаются, однако с применением определённого правила их можно свести к одной смысловой структуре. Такое преобразование может выглядеть следующим образом: <<n1 подарить n2> n3>, <n3 связан с n2> преобразуется в <n3 подарить n2>, <n3 связан с n1>.

Такой метод эффективно работает в узкоспециализированных сферах, однако его гибкость ограничена, и он плохо адаптируется к обработке больших массивов текстовой информации.

Подход, основанный на машинном обучении, предполагает использование обучаемых алгоритмов, способных выявлять закономерности в данных и строить ответы на основе обучающих примеров. В рамках этого направления выделяют два основных метода: классификация и генерация.

Классификационные модели строятся на основе нейронных сетей, задача которых выбрать наиболее вероятный ответ из ограниченного набора возможных. Одной из наиболее эффективных архитектур является рекуррентная нейронная сеть с долговременной краткосрочной памятью (LSTM). Общая схема такой сети представлена на рисунке 1.1.

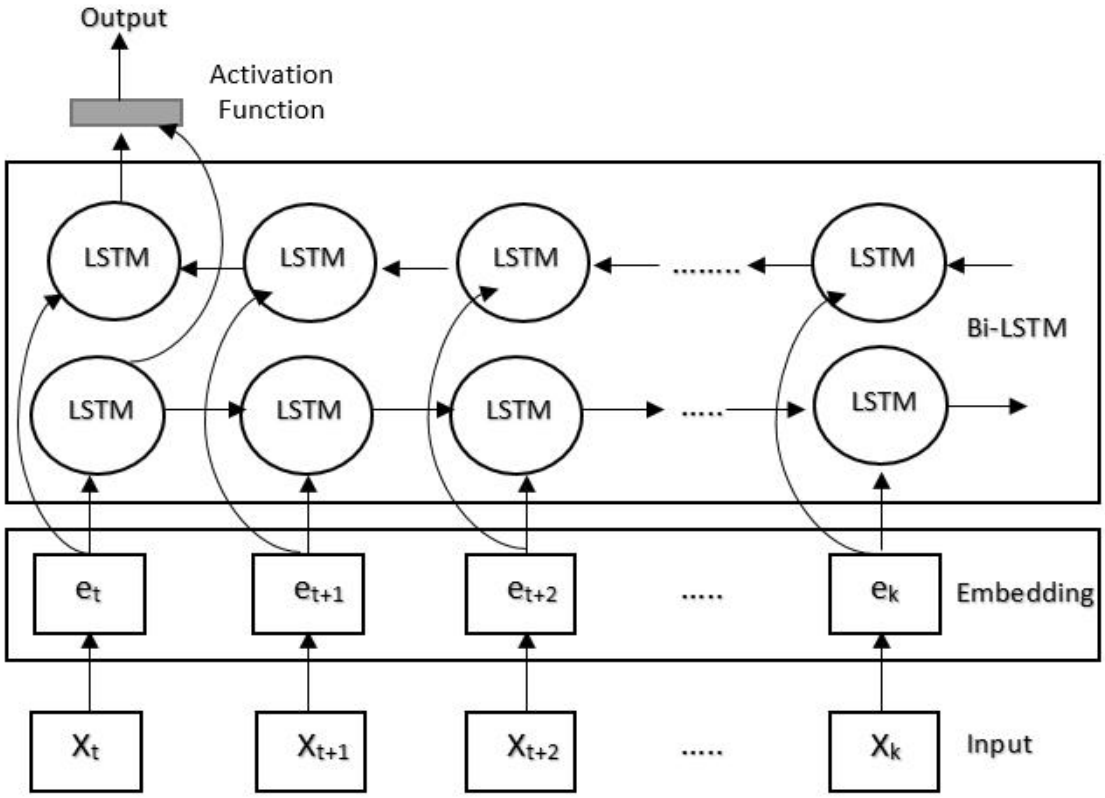

Рисунок 1.1 - Архитектура классификационной модели LSTM

Каждое слово вопроса и ответа предварительно переводится в векторное представление, после чего подаётся на вход LSTM-сети. Внутренние представления вопроса и ответа обозначаются как векторы (c) и (r). Далее производится их преобразование: вектор (c) умножается на обучаемую матрицу M, в результате чего формируется предсказанный вектор ответа r′.

Для оценки точности используется скалярное произведение r′ и r, после чего применяется функция активации (сигмоида), преобразующая результат в вероятность правильности. Ошибки предсказания корректируются с использованием функции потерь:

$$P = -y \cdot \ln(y') - (1 - y) \cdot \ln(1 - y'),$$

где y — истинная метка, а y′ — предсказанная вероятность.

Порождающие модели (Sequence-to-Sequence) основаны на архитектуре **Sequence-to-Sequence (Seq2Seq)** и применяются в случаях, когда ответ необходимо сгенерировать, а не выбрать. Архитектура таких систем включает два компонента: кодировщик (encoder) и декодировщик (decoder). Их взаимодействие представлено на рисунке 1.2.

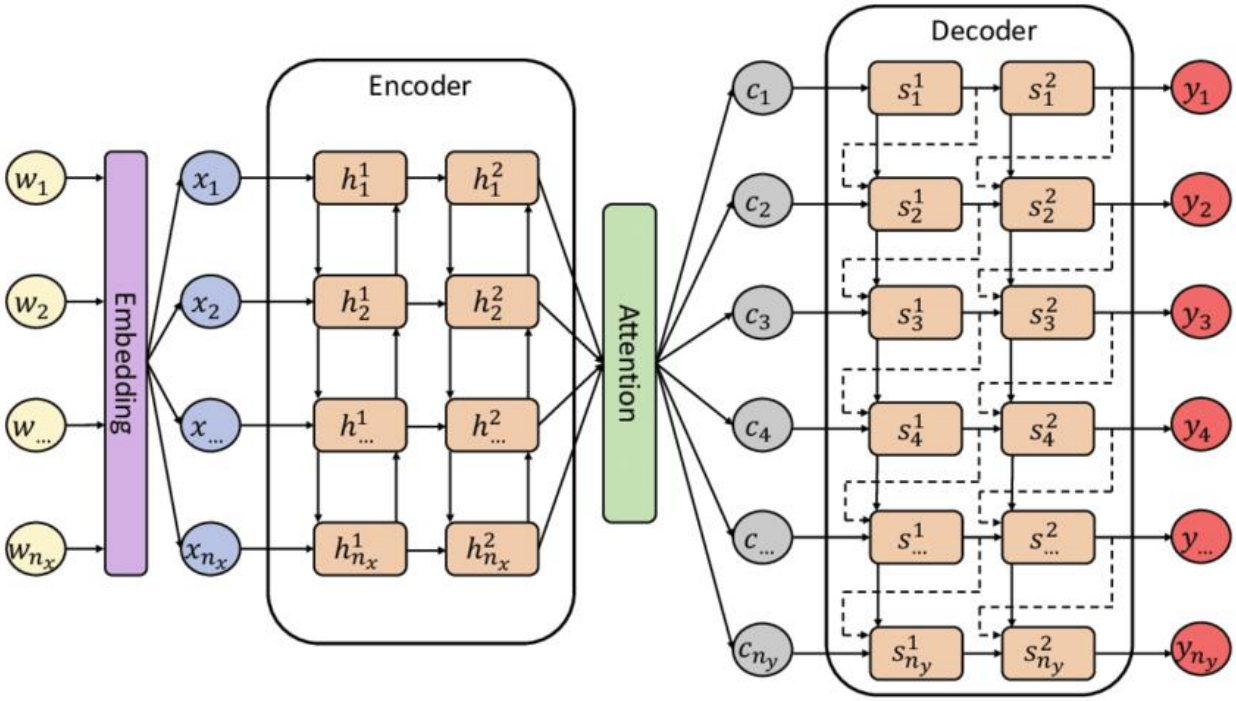

Рисунок 1.2 - Архитектура генеративной модели Seq2Seq

Кодировщик обрабатывает входной вопрос и формирует контекстный вектор, отражающий его смысл. Этот вектор затем передаётся на вход декодировщику, который поэтапно порождает ответ, используя слой softmax для выбора слов с наивысшей вероятностью. Процесс генерации включает следующие этапы:

- инициализация внутреннего состояния декодировщика последним состоянием кодировщика;
- подача символа <eos> (end of sentence) на вход декодировщику;
- пошаговое предсказание слов, пока не будет сгенерирован <eos> или не достигнута максимальная длина ответа.

Для обучения модели используется техника **teacher forcing**, при которой на каждом шаге декодировщику подаётся правильное слово, а не сгенерированное.

Улучшения генеративных моделей:

- увеличение глубины сети (добавление слоёв);
- применение двунаправленных кодировщиков;
- использование символов вместо слов, что уменьшает размер словаря;
- Учёт вектора диалогового контекста, позволяющего системе сохранять информацию о предыдущих репликах.

Основными проблемами являются шаблонные ответы и отсутствие учёта контекста диалога. Проблемой шаблонных ответов является генерация

слишком общих ответов ("Хорошо", "Спасибо", "Да"). Данная проблема решается оптимизацией функции взаимной информации между вопросом и ответом. Проблема отсутствия учёта контекста диалога решается путём добавления специального контекстного вектора, передаваемого в декодировщик.

Таким образом, архитектура вопросно-ответной системы может быть основана как на ручных правилах и шаблонах, так и на алгоритмах машинного обучения. Подход на основе правил обеспечивает интерпретируемость и точность, но плохо масштабируется. Модели, основанные на машинном обучении — особенно генеративные архитектуры типа Seq2Seq — характеризуются высокой адаптивностью, умением обобщать информацию и возможностью самообучения на обширных текстовых корпусах.

1.1.3 Интеграция NLP и баз знаний в QA-системах

Интеграция технологий обработки естественного языка (NLP) и баз знаний (Knowledge Bases, KB) является одним из ключевых направлений в развитии современных вопросно-ответных систем. Такая интеграция позволяет системам не просто извлекать текстовую информацию, но и опираться на структурированные, формализованные источники знаний, такие как графы знаний, онтологии и семантические базы данных. Это особенно важно для систем, работающих в узкоспециализированных предметных областях, где необходимы точность и интерпретируемость.

Базы знаний представляют собой формализованное хранилище фактов, связей и логических правил.

Базы знаний:

- **Wikidata** — семантический граф, содержащий общеизвестные факты;
- **DBpedia** — извлечённые структурированные данные из Wikipedia;
- **Freebase** (устаревший, но влиятельный проект);
- **ConceptNet**, **YAGO**, а также доменные базы (например, UMLS в медицине, SNOMED CT).

Формально, такие базы можно представить как граф: вершины — сущности (люди, организации, события), рёбра — отношения между ними (родился-в, работает-в, часть-от и др.).

Современные NLP-системы обеспечивают переход от пользовательского естественного языка к формальному языку запросов, понятному базам знаний. Это позволяет пользователю задавать вопросы в свободной форме, а системе — преобразовывать их в структурированный запрос.

Наиболее распространённый подход — **semantic parsing**: трансформация вопроса в формальный язык, например **SPARQL** (в случае RDF-графов).

Пример SPARQL-запроса:

Вопрос: *«Где родился Исаак Ньютон?»*

SPARQL-запрос:

```
SELECT ?place WHERE {
?person rdfs:label "Isaac Newton"@en.
?person dbo:birthPlace ?place.
}
```

Для генерации таких запросов применяются:

- Seq2Seq-модели с вниманием (attention);
- гибридные правила + ML-модели;
- архитектуры типа **Seq2SPARQL**.

В последнее время всё большую популярность приобретает гибридный подход, где сочетаются возможности NLP и KB:

- **этап извлечения (retrieval)**: система находит кандидатов (сущности, триплеты) на основе семантической близости.
- **этап логического вывода (reasoning)**: проводится анализ связей между сущностями для уточнения ответа.

Архитектурная схема гибридной вопросно-ответной системы представлена на рисунке 1.3.

Для этого используются:

- **Graph Neural Networks (GNN)** — для обучения на графах знаний.
- **Knowledge Graph Embeddings** — методы вроде TransE, RotatE, ComplEx, которые кодируют связи между сущностями в векторном пространстве.
- **Sentence-BERT / SBERT** — для сопоставления вопроса и текста/триплетов на семантическом уровне.

Пример:

Пользователь спрашивает: *«Какой город является столицей Франции?»*

Система:

- идентифицирует сущность «Франция» и предикат «столица»;
- проводит семантический поиск по графу знаний;
- находит триплет: (Франция — столица — Париж);
- отвечает: *«Париж»*.

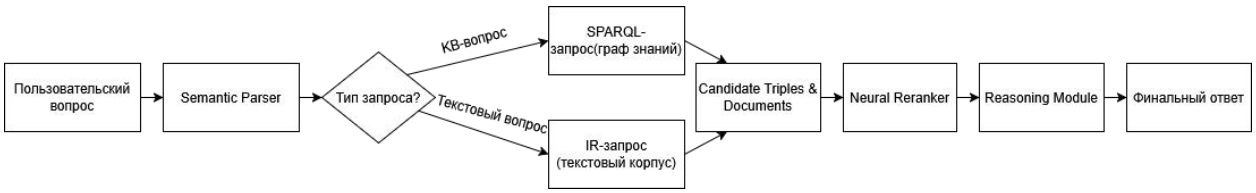

Рисунок 1.3 - Архитектурная схема гибридной QA-системы

Особую роль базы знаний играют в системах, ориентированных на медицину, право, науку, где критически важно не просто извлекать текст, а понимать смысловые связи между терминами. Например, система поддержки принятия клинических решений может использовать онтологию **SNOMED CT**, где описаны симптомы, заболевания и препараты. При этом:

- используются терминологические маппинги: NLP-система нормализует текст и соотносит его с терминами онтологии;
- проводится анализ вложенности понятий и причинно-следственных связей;
- реализуется логический вывод на уровне онтологических правил.

Интеграция NLP и баз знаний позволяет:

- повысить точность и обоснованность ответов;
- предоставлять интерпретируемые результаты (основанные на фактах);
- учитывать семантические связи между понятиями;
- строить сложные цепочки рассуждений и отвечать на многошаговые вопросы.

Несмотря на высокую перспективность, подход сопряжён с рядом сложностей:

- автоматическая генерация формальных запросов из текста (semantic parsing) всё ещё несовершенна;
- требуется дополнительная аннотация данных и наличие высококачественных онтологий;
- большие графы знаний имеют проблему масштабирования — поиск может быть затратным по времени;
- интеграция нейросетевых моделей с логическими системами требует балансировки: нейросети — вероятностны, КВ — дискретны и детерминированы.

В таблице 1.1 указаны примеры существующих вопросно-ответных систем.

Таблица 1.1 - Примеры существующих систем

Система	Особенности
Google Knowledge Graph	Используется в поиске Google для уточнения смысла и построения карточек знаний.
Wikidata + NLP (QAnswer)	Онлайн-система, отвечающая на вопросы на естественном языке с использованием Wikidata.
DrQA + Wikipedia + KB	Гибридная архитектура от Facebook AI Research.
MedQuAD + UMLS	Система QA на медицинских онтологиях, используемая в клиниках.

1.1.4 Реализация вопросно-ответной системы

Реализация вопросно-ответной системы (QA-системы) включает в себя несколько ключевых компонентов, от этапа обработки естественного языка до механизма получения и представления ответа. В данной главе рассмотрим разработку QA-системы гибридного типа, основанной на комбинации методов машинного обучения и поиска по базе знаний.

QA-система состоит из следующих основных модулей:

- модуль предобработки текста (Text Preprocessing);
- модуль понимания вопроса (Question Understanding);
- модуль поиска кандидатов (Retriever);
- модуль извлечения или генерации ответа (Reader/Generator);
- интеграция с базой знаний (если используется);
- формирование и возврат ответа.

На рисунке 1.4 приведена схема архитектуры реализуемой QA-системы:

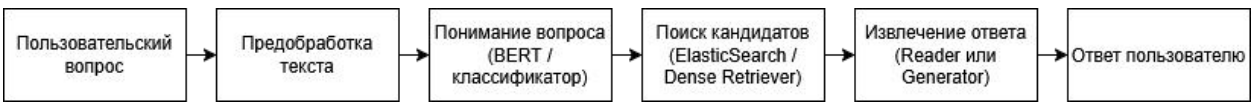

Рисунок 1.4 - Схема архитектуры реализуемой QA-системы

Первый этап — очистка и подготовка вопроса. Это включает в себя:

- приведение к нижнему регистру;

- удаление лишних знаков препинания;
- Токенизация;
- лемматизация (по необходимости).

Пример (Python с NLTK и spaCy):

```
import spacy
nlp = spacy.load("en_core_web_sm")
def preprocess(question):
    doc = nlp(question.lower())
    tokens = [token.lemma_ for token in doc ifnot token.is_punct]
    return"".join(tokens)
q = "What are the symptoms of diabetes?"
print(preprocess(q))  # ->"what be the symptom of diabetes"
```

На этапе понимания вопроса задача — интерпретировать вопрос, определить его тип (фактологический, определение, причина и т.д.), а также выделить ключевые сущности и отношения.

Для этого может использоваться:

- классификатор вопроса (например, на BERT);
- Named Entity Recognition (NER);
- Semantic Role Labeling.

Пример классификации вопроса (на BERT):

```
from transformers import BertTokenizer, BertForSequenceClassification
import torch

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained("bert-base-
uncased", num_labels=3)

inputs = tokenizer("Who is the president of France?", return_tensors="pt")
outputs = model(**inputs)
label = torch.argmax(outputs.logits).item()
# label: 0 - person, 1 - location, 2 - definition
```

На этапе поиска кандидатов (Retriever) из корпуса документов или базы знаний выбираются потенциальные источники ответов. Возможны два подхода:

- **BM25 / ElasticSearch** — классический поиск по ключевым словам;
- **Dense Retriever** — векторный поиск по семантическому сходству (например DPR или Faiss).

Пример поиска с BM25 через ElasticSearch:

```
from elasticsearch import Elasticsearch

es = Elasticsearch()

def search_documents(question, top_k=5):
    query = {
        "query": {
            "match": {
                "content": question
            }
        }
    }
    res = es.search(index="documents", body=query, size=top_k)
    return [hit["_source"]["content"] for hit in res["hits"]["hits"]]
```

На этапе извлечения или генерации ответа после того как найдены кандидаты, система должна либо извлечь ответ из текста (Extractive QA), либо сгенерировать его (Generative QA)

Пример Extractive QA (Reader). Используются модели вроде BERT для ответа в стиле SQuAD:

```
from transformers import pipeline

qa = pipeline("question-answering")

result = qa(question="Who is the president of France?", context="Emmanuel Macron is the current president of France.")
print(result["answer"])  # -> Emmanuel Macron
```

Пример Generative QA (Generator). Используется модель типа T5 или GPT для генерации ответа:

```
from transformers import T5Tokenizer, T5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("t5-base")
model = T5ForConditionalGeneration.from_pretrained("t5-base")

input_text = "question: Who is the president of France? context: Emmanuel Macron is the president."
input_ids = tokenizer.encode(input_text, return_tensors="pt")
output_ids = model.generate(input_ids)
```

```
print(tokenizer.decode(output_ids[0], skip_special_tokens=True))  # ->
Emmanuel Macron
```

Если используется база знаний (например, Wikidata или локальная RDF-графовая база), на этапе поиска система может:

- выполнить семантический парсинг вопроса;
- сгенерировать SPARQL-запрос;
- получить результат напрямую.

Пример SPARQL-запроса через библиотеку SPARQLWrapper:

```
from SPARQLWrapper import SPARQLWrapper, JSON
from SPARQLWrapper import SPARQLWrapper, JSON

sparql = SPARQLWrapper("https://query.wikidata.org/sparql")

sparql.setQuery("""
SELECT ?placeLabel WHERE {
  wd:Q935 rdfs:label "Isaac Newton"@en.
  wd:Q935 wdt:P19 ?place.
  SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }
}
""")

sparql.setReturnFormat(JSON)
results = sparql.query().convert()
place = results["results"]["bindings"][0]["placeLabel"]["value"]
print(place)  # -> Woolsthorpe-by-Colsterworth
```

На финальном этапе возврата и форматирования ответа происходит:

- форматирование ответа;
- проверка на "пустой" результат;
- добавление дополнительной информации при необходимости (например, источник, цитата, ссылка).

Пример (Python):

```
def format_answer(answer, source=None):
    ifnot answer:
        return"Извините, ответ не найден."
    returnf"Ответ: {answer}" + (f"\nИсточник: {source}"if source else"")
```

Разработка вопросно-ответной системы — это комплексный процесс, включающий модули лингвистической обработки, поиска информации, машинного обучения и семантических баз знаний. Даже в упрощённой реализации важно обеспечить согласованную работу всех компонентов:

- предобработка обеспечивает корректный ввод;
- интерпретация позволяет системе "понять" вопрос;
- механизм поиска обеспечивает релевантные контексты;
- извлечение/генерация — ключ к точности и полезности ответа;
- интеграция с KB — расширяет систему за рамки чистого текста.

Современные QA-системы, такие как DrQA, Haystack, GPT и T5, реализуют вышеописанные подходы и позволяют эффективно работать с различными типами данных и запросов. При этом выбор конкретной архитектуры зависит от требований к точности, области применения и доступным ресурсам.

1.2 Особенности разработки чат-ботов

1.2.1 Обоснование актуальности разработки чат-ботов

Благодаря быстрому развитию технологий большую популярность обрели чат-боты и языковые модели. Сейчас их используют повсеместно, от обучения до медицины, от бизнес-задач до разработки кода.

Из-за развития популярности общения в мессенджерах таких как WhatsApp, Facebook, Telegram и другие, чат-боты ещё с большей скоростью набирают популярность, так как все общение с клиентами происходит в мессенджерах и стало гораздо удобнее и дешевле это делать посредством чат-ботов.

Чат-боты доступны в любое время и у них нет перерыва на обед и выходных, благодаря этому пользователи могу получить ответ на свой вопрос в любой момент времени, и ответ займёт гораздо меньше времени. Из-за этого очень вырастает уровень обслуживания клиентов, а работники компаний могут сконцентрироваться на сложных запросах пользователей, которые требуют вмешательства человека.

Также чат-боты могут сделать более удобными множество простых процессов, таких как предоставление информации, обработка заказов и многих других, что позволяет экономить деньги и время компании.

Разработка чат-ботов стала более доступной и эффективной за счёт развития технологий обработки естественного языка и машинного обучения. Благодаря системам NLP чат-боты научились понимать и генерировать текст на естественном языке, что расширило возможности их использования.

Алгоритмы машинного обучения позволили чат-ботам иметь возможность улучшать навыки общения и изменяться в зависимости от потребностей пользователя.

Чат-боты также могут представлять метод использования искусственного интеллекта (ИИ), который становится все более влиятельным в различных областях. Разработка чат-ботов является практическим применением ИИ, что помогает улучшить и упростить взаимодействие ИИ с людьми.

Разработка чат-ботов актуальна и значима для таких областей, как образование, коммерция, здравоохранение, развлечения. С помощью чат-ботов имеется возможность улучшить коммуникацию, автоматизацию бизнес-процессов и качество обслуживания. Развитие технологий искусственного интеллекта и обработки естественного языка позволяет создавать все более качественные и умные системы. Исходя из этого можно заключить, что исследование и разработка чат-ботов является перспективным направлением в информационных технологиях.

1.2.2 История развития чат-ботов

Идея создания программ, способных взаимодействовать с людьми в естественном языке, возникла задолго до появления современных чат-ботов. Ещё в 1950-х годах учёные начали исследовать и экспериментировать с созданием программ, которые смогут понимать и имитировать человеческую речь.

Одним из первых и известных примеров чат-бота является ELIZA, разработанный Джозефом Вейзенбаумом в 1966 году. ELIZA была программой, которая имитировала психотерапевта и умела задавать вопросы на основе полученных ответов. ELIZA использовала простые шаблоны и правила для обработки текстовых команд, и, несмотря на свою ограниченность, она вызвала большой интерес и стала примером для последующих "разговорных" программ.

Развитие технологий разработки чат-ботов продолжалось и появлялись новые инструменты и подходы. В 1990-х годах появились чат-боты, использующие методы обработки естественного языка (NLP) и статистические модели для распознавания и генерации текста. Примером такого чат-бота является A.L.I.C.E., созданный Ричардом Уоллисом в 1995 году.

В связи с развитием интернета и социальных сетей возросла популярность и использование чат-ботов. В 2000-х годах появились платформы для разработки чат-ботов, с помощью которых можно было создавать более сложные и интерактивные системы. Одной из таких платформ является Microsoft Bot Framework, представленный в 2016 году.

За последние годы чат-боты стали более распространёнными и нашли применение в различных областях. Их можно использовать в клиентском обслуживании, маркетинге, образовании, здравоохранении и других сферах деятельности. Развитие и использование ИИ и машинного обучения помогают улучшению способностей и функционала чат-ботов, благодаря чему они становятся более адаптивными и интеллектуальными.

С развитием технологий развиваются и чат-боты, становясь более многофункциональными и умными. Они могут обрабатывать сложные запросы и выполнять огромное множество различных задач. Благодаря быстрому развитию технологий чат-ботам находят все больше применений.

1.2.3 Определение и классификация чат-ботов

Чат-боты – это программные агенты, они могут имитировать человеческое общение, взаимодействуя с пользователем через текстовый или голосовой интерфейс. Структура и реализация чат-ботов строится на технологиях, таких как обработка естественного языка , машинное обучение и искусственный интеллект, и они могут выполнять разнообразные задачи, от ответов на простые вопросы до предоставления сложных услуг.

Классифицировать чат-ботов можно как и вопросно-ответные системы по различным параметрам.

1. По типу взаимодействия:

- текстовые чат-боты: Это самый распространённый тип чат-ботов, они взаимодействуют с пользователями с помощью текстовых сообщений;
- голосовые чат-боты: Голосовые чат-боты используются как голосовой ассистент который всегда на связи. Обычно они работа в фоновом режиме и активируются какой-то конкретной фразой, например "окей Google";
- мультимодальные чат-боты: Эти чат-боты используют одновременно и голосовой и текстовый интерфейсы, что позволяет пользователям взаимодействовать с ними в том формате, в котором им удобнее.

2. По функциональности:

- информационные чат-боты: Работа информационных чат-ботов заключается в предоставлении информации на основе заданных пользователем вопросов. Такие чат-боты могут предоставлять справочную информацию, выполнять поиск по базе данных или отвечать на вопросы;
- задачевые чат-боты: Такие чат-боты разработаны специально для выполнения определённых задач, например бронирование билетов или

организация встреч. Обычно в эти чат-боты интегрированы соответствующие сервисы;

- развлекательные чат-боты: Эти чат-боты предназначены для развлечения пользователей. Такие чат-боты могут предлагать анекдоты, текстовые игры, загадки или просто приятное общение.

3. По уровню интеллекта:

- простые чат-боты: Простые чат-боты основаны на заданных правилах и шаблонах. Они отвечают заранее определёнными фразами на основе ключевых слов, и они не могут обрабатывать сложные запросы и адаптироваться под пользователя;
- искусственно интеллектуальные чат-боты: В чат-ботах данного типа используются ИИ и алгоритмы машинного обучения для понимания и анализа запросов на естественном языке. Такие чат-боты могут обрабатывать сложные запросы и распознавать намерения пользователя, благодаря чему общение с ними становится более естественным.

Классифицировать чат-ботов можно по различным критериям в зависимости от множества факторов. Представленные классификации являются общим случаем классификации и могут изменяться в зависимости от конкретных чат-ботов.

1.2.4 Технологии и инструменты для разработки чат-ботов

Благодаря огромному множеству инструментов и технологий, которые могут использоваться для создания чат-ботов, разработка становится проще и удобнее, а чат-боты улучшают свою человечность и адаптивность.

Существует множество платформ для создания чат-ботов:

1) Dialogflow: Это облачный инструмент разработанный компанией Google, предназначенный для создания чат-ботов. Эта платформа использует обработку естественного языка и элементы машинного обучения, что в свою очередь позволяет реализовывать сложные сценарии взаимодействия. Среди ключевых возможностей Dialogflow – определение намерений пользователя (интентов), управление диалоговыми контекстами, а также поддержка интеграции с внешними сервисами и популярными мессенджерами.

2) Microsoft Bot Framework: Это платформа от Microsoft, предназначенная для создания многофункциональных чат-ботов. С помощью этой платформы можно разрабатывать как текстовых так и голосовых чат-ботов. Данная платформа совместима с популярными языками программирования, такими как C# и JavaScript. Также у данной платформы есть инструменты для развертывания и администрирования.

3) IBM Watson Assistant: Это платформа от IBM, для разработки виртуальных ассистентов. Ключевыми технологиями данной платформы являются искусственный интеллект и машинное обучение. На платформе присутствуют инструменты для мониторинга и оптимизации, также разработан функционал использования сторонних сервисов по средством API.

Языки программирования и фреймворки:

1) Python: Один из наиболее востребованных языков в разработке чат-ботов благодаря огромному количеству библиотек. Для задач обработки естественного языка используются такие библиотеки как NLTK и SpaCy, а для веб-интеграции – фреймворки FastAPI, Django и Flask.

2) Node.js: Серверная платформа, которая часто используется в разработке из-за своей асинхронной архитектуры. Backend для чат-ботов разрабатывается с помощью фреймворка Express.js.

3) TensorFlow: Это фреймворк для машинного обучения от Google с открытым исходным кодом. С помощью данного фреймворка создаются модели машинного обучения и обработки естественного языка для распознавания запросов в чат-ботах.

Облачные платформы и сервисы:

4) Amazon Lex: Это облачный сервис от Amazon Web Services (AWS). Этот сервис используется для создания как текстовых так и голосовых чат-ботов. Основными технологиями являются NLP и ML. Также этот сервис используется для развертывания и масштабирования чат-ботов.

5) Google Cloud Dialogflow: Это облачное решение Google, которое использует искусственный интеллект и машинное обучение для разработки чат-ботов.

Выбор конкретных технологий и инструментов зависит от конкретной задачи и предпочтения разработчика.

1.2.5 Примеры применения чат-ботов в различных областях

Чат-боты используются в огромном множестве областей деятельности. Они находят применение в клиентской поддержке, в сфере образования, здравоохранения и многих других.

В клиентской поддержке чат-боты используются для предоставления поддержки круглые сутки, чтобы была возможность отвечать на их вопросы и проблемы в режиме реального времени, а также уменьшить нагрузку на персонал. Они могут помочь в решении простых вопросов, а в случает затруднительного положения могут направить клиента к специалисту.

Благодаря этому упрощается и ускоряется работа с клиентами, а также повышается качество обслуживания.

В сфере электронной коммерции чат-боты используются для оптимизации взаимодействия с пользователями. Чат-боты используются для ответов на вопросы пользователей, помощи в подборе товаров и рекомендаций. Также чат-боты помогают автоматизировать оформление заказов и общаются с пользователем от момента покупки до её получения.

В банковском и финансовом секторах чат-боты предоставляют клиентам информацию о балансе счета, последних операциях, помогают с переводами, предлагают финансовые советы и помогают в управлении бюджетом.

В туризме и гостиничном бизнесе чат-боты используются для помощи в бронировании билетов и отелей, также они предоставляют информацию о достопримечательностях и могут составить подходящий маршрут исходя из предпочтений пользователя.

В здравоохранении чат-боты помогают пользователям отвечая на вопросы о лекарствах и симптомах. Чат-боты могут напомнить о приёме лекарств, помочь с записью к врачу и проконсультировать по поводу ведения здорового образа жизни.

В области образования чат-боты используются для помощи с заданиями, составляют рекомендации по курсам, помогают составлять тестовые задания и готовиться к экзаменам. Также чат-боты могут использоваться как робот учитель, помогая проводить оценку знаний пользователя и подсказывая в каких областях у него проблемы.

В каждой области чат-боты используются по разному и их разработка учитывает специфику задач.

1.3 Сравнение QA-систем и чат-ботов

1.3.1. Цели и задачи

Вопросно-ответные системы (QA-системы) и чат-боты представляют способы взаимодействия между компьютером и человеком, но их цели и задачи сильно разнятся. При выборе архитектуры необходимо учитывать данные различия. Главное различие между чат-ботами и вопросно-ответными системами, это точность и ведение диалога. Чат-боты отвечают менее точно, но способны вести более человечный диалог, а вопросно-ответные системы, в свою очередь, отличаются точностью, но не способны вести беседы как настоящий человек.

Вопросно-ответные системы используются когда необходимо выдавать точную информацию. На конкретный вопрос пользователя необходимо дать

конкретный ответ. Благодаря этому подходу вопросно-ответные системы отличаются своей точностью, но теряют гибкость.

Основные задачи QA-систем:

- интерпретация вопроса на естественном языке;
- извлечение информации из базы знаний или корпуса документов;
- формирование краткого, информативного ответа;
- минимальное взаимодействие — без поддержки длительного диалога.

Примеры:

- "Кто написал 'Мастер и Маргарита'?" → «Михаил Булгаков»;
- "Сколько хромосом у человека?" → «46».

Архитектурные особенности:

- используются модели типа **retrieval-based QA** или **generative QA**;
- контекст ограничивается текущим вопросом;
- часто интегрируются с Википедией, базами знаний (Wikidata, DBpedia), корпоративными документами.

Целью вопросно ответных систем является дать конкретный, проверяемый ответ с минимальным количеством шагов.

Чат-боты предназначены для поддержки общения, в том числе неформального. Они могут выполнять задачи, выходящие за рамки информационного поиска: бронирование, сопровождение пользователя, развлечение, ведение диалога с элементами эмоционального интеллекта.

Основные задачи чат-ботов:

- поддержание многотактного диалога с учётом контекста;
- интерпретация намерений пользователя (intent detection);
- генерация естественных "человеко-подобных" реплик;
- реакция на эмоции, ситуации неопределённости и неполные данные.

Примеры:

- Пользователь: «Мне грустно»;
- Бот: «Сожалею, что ты так себя чувствуешь. Хочешь поговорить об этом?»;
- Пользователь: «Закажи пиццу на 19:00»;
- Бот: «Какой размер и начинку предпочитаешь?».

Архитектурные особенности:

- Используются диалоговые модели (например, DialoGPT, BlenderBot, ChatGPT);
- Внедряется память (dialog history, context tracking);
- Возможна интеграция с внешними сервисами (календарь, CRM, API).

Целью чат-ботов является обеспечение естественного взаимодействия, ориентированного на пользователя. В таблице 1.2 приведён сравнительный анализ QA-систем и чат-ботов по целям и задачам.

Таблица 1.2 - Сравнительная таблица: цели и задачи QA-систем и чат-ботов

Критерий	**QA-система**	**Чат-бот**
Основная цель	Извлечение фактов	Коммуникация и сопровождение
Тип ответа	Краткий, точный, проверяемый	Диалоговый, контекстный, адаптивный
Работа с контекстом	Ограниченно, в пределах одного вопроса	Расширено, с историей диалога
Используемые технологии	Information Retrieval, NLP BERT	Generative models, Dialog Management
Эмоциональный интеллект	Отсутствует	Возможна имитация сочувствия, эмпатии
Примерная задача	«Когда был основан Google?»	«Напомни о встрече завтра в 10 утра»

Таким образом, QA-системы и чат-боты, несмотря на использование схожих технологий обработки естественного языка, имеют принципиально разные цели. QA-системы оптимальны для одиночных, конкретных запросов, где важна точность. Чат-боты же ориентированы на долгосрочное, интерактивное общение, адаптированное под поведение пользователя. Выбор подхода определяется задачами приложения и ожиданиями конечного пользователя.

1.3.2. Технические различия

Хотя и чат-боты, и QA-системы обрабатывают запросы пользователя с использованием технологий обработки естественного языка (NLP), их техническая реализация существенно различается, особенно в аспектах управления диалогом и работы с контекстом. Эти различия затрагивают архитектуру моделей, подход к хранению информации, а также принципы генерации или извлечения ответов.

Диалоговая логика QA-системы:

- отвечают на *изолированные* вопросы, не зависящие от предыдущих реплик;
- диалоговая логика минимальна либо отсутствует;
- ответ формируется в рамках единственного шага: "запрос → поиск → ответ";
- не требуется управление состоянием диалога.

Типовой стек:

- ретривер (например, BM25, DPR) + ридер (BERT, T5);
- иногда — zero-shot генеративные модели (GPT, FLAN) без сохранения диалога.

Пример:

Q: Кто был первым президентом США?

A: Джордж Вашингтон.

Диалоговая логика чат-ботов:

- реализуют многотактную диалоговую логику, где каждый новый запрос зависит от предыдущих реплик;
- система должна отслеживать состояние диалога, намерения (intent) и сущности (entities);
- возможны ветвления, уточнения, переходы между темами;
- в сложных случаях применяются диалоговые FSM (finite state machines), **policy-based** подходы, или **reinforcement learning** для выбора действий.

Типовой стек:

- NLU-модуль (определение интента и сущностей);
- менеджер диалога (Dialog Manager / Policy);
- NLG-модуль (генерация реплики);
- память / история диалога.

Пример:

Пользователь: Закажи пиццу.

Бот: Какую начинку предпочитаешь?

Пользователь: Пепперони.
Бот: Отлично. Доставить на тот же адрес?

Контекст QA-системы:

- контекст трактуется в основном как *информационный контекст* — документы, из которых нужно извлечь ответ;
- обычно не учитывается предыдущий диалог, разве что в *conversational QA* (например, QuAC, CoQA);
- если и применяется контекст, то только в пределах одного-двух предыдущих вопросов.

Контекстная модель:

- Вход: {вопрос + параграф из документа};
- Не требуется отслеживать идентичность пользователя, тему, эмоции и т. п.

Пример:
Q1: Когда началась Вторая мировая война?
A1: 1 сентября 1939 года.
Q2: А когда она закончилась?
(если контекст сохраняется, то "она" → Вторая мировая война)

Контекст чат-ботов

- Под контекстом понимается вся история взаимодействия: реплики, интенты, цели, состояние сессии;
- Часто применяются механизмы памяти (short-term и long-term memory);
- Контекст включает не только текст диалога, но и метаинформацию: настроение пользователя, его предпочтения, предыдущие действия;
- Модели вроде GPT-4, ChatGPT, Claude используют *весь контекст* текущей сессии как окно внимания (up to ~100K токенов).

Пример:
Пользователь: Я ищу подарки для сестры. Ей 12.
(Спустя 5 сообщений)
Пользователь: А что бы ты порекомендовал ей на день рождения?
→ Модель должна помнить возраст, родственные связи, тему — и предложить релевантный вариант.

В таблице 1.3 приведён сравнительный анализ QA-систем и чат-ботов по ключевым параметрам.

Таблица 1.3 - Сравнение по ключевым параметрам

Параметр	QA-система	Чат-бот
Диалоговая логика	Отсутствует или минимальная	Сложная, с отслеживанием состояний
Хранение контекста	Ограничено текущим запросом	Полноценная память, управление сессией
Управление интентами	Не требуется	Обязательное (NLU, intent classification)
Переходы между темам	Неподдерживаемые	Поддерживаются
Поддержка "языковых кореференций"	Частичная или вручную	Автоматическая, через трансформеры
Тип моделей	Extractive/generative QA	Generative LLM + диалоговый менеджер

Технические различия между QA-системами и чат-ботами определяются глубиной взаимодействия с пользователем. В QA-системах достаточно минимальной логики и точного механизма поиска или генерации ответа. Чат-боты же требуют полноценного управления диалогом, гибкой работы с контекстом и интерактивной логики поведения. Это делает их более сложными в реализации, но более универсальными в сценариях пользовательского взаимодействия.

1.3.3. Преимущества и ограничения каждого подхода

Вопросно-ответные системы (QA-системы) и чат-боты представляют собой два различных, хотя и взаимосвязанных, подхода к реализации интеллектуального взаимодействия между пользователем и машиной. Несмотря на сходство в используемых технологиях (в частности, обработке естественного языка и машинном обучении), они существенно различаются по целям, архитектурным решениям и применяемым сценариям. Ниже приведён

сравнительный анализ основных преимуществ и ограничений каждого из подходов.

Преимущества вопросно-ответных систем:

- **точность выдачи информации.** Вопросно-ответные системы ориентированы на предоставление конкретных, фактологических ответов. При наличии корректно сформулированного запроса пользователь получает краткий и релевантный результат, что особенно актуально для задач справочного характера;
- **высокая скорость получения результата.** В отличие от диалоговых систем, QA-системы не требуют ведения многоходового общения, что позволяет сократить время между запросом и ответом;
- **простота реализации и поддержки.** Отсутствие необходимости отслеживания состояния диалога упрощает реализацию, особенно при использовании извлекающих (retrieval-based) архитектур;
- **устойчивость к нарушению логики общения.** Система обрабатывает каждый запрос независимо, что делает её менее чувствительной к непоследовательным или контекстно неоднозначным репликам.

Ограничения вопросно-ответных систем:

- **отсутствие поддержки диалога.** Система не может задавать уточняющие вопросы или адаптироваться к изменению тематики в рамках одного сеанса;
- **низкий уровень контекстуализации.** Если ответ требует учёта предшествующего общения, большинство QA-систем без механизма хранения диалогового контекста не смогут корректно обработать такой запрос;
- **ограниченность применяемых сценариев.** Подход применяется преимущественно в случаях, где возможны точные, одношаговые ответы, и не подходит для комплексных пользовательских взаимодействий;
- **сложности с обработкой неформализованного языка.** Запросы с использованием жаргона, иронии или недоопределённых формулировок могут не интерпретироваться системой должным образом.

Преимущества чат-ботов:

- **поддержка полноценного диалога.** Современные диалоговые системы способны вести многошаговое общение, задавать уточняющие вопросы, обрабатывать пользовательские намерения (интенты) и осуществлять переходы между темами;

- **гибкость и адаптивность.** Чат-боты успешно применяются в самых разных областях: от электронной коммерции и онлайн-обслуживания до развлекательных и образовательных сервисов;
- **учёт контекста.** Большинство современных чат-ботов (особенно на базе генеративных моделей) способны учитывать историю взаимодействия, пользовательские предпочтения и текущий эмоциональный тон общения;
- **повышенная вовлеченность пользователя.** Благодаря имитации живого общения, чат-боты обеспечивают более персонализированный и дружелюбный пользовательский опыт.

Ограничения чат-ботов:

- **сложность архитектуры.** Необходимость управления состоянием диалога, понимания интентов, обработки сущностей и поддержки различных сценариев значительно усложняет разработку и тестирование таких систем;
- **снижение точности ответов.** Генеративные модели, используемые в чат-ботах, склонны к формированию неконкретных или обобщённых фраз, особенно в случае недостаточного контроля над генерацией;
- **проблемы с интерпретируемостью.** Объяснение, почему система сформировала тот или иной ответ, может быть затруднено, особенно при использовании нейросетевых архитектур "чёрного ящика";
- **повышенные требования к вычислительным ресурсам.** Диалоговые модели, особенно на базе трансформеров, требуют значительных вычислительных мощностей как на этапе обучения, так и в процессе инференса.

В таблице 1.4 приведён общий сравнительный анализ QA-систем и чат-ботов.

Таблица 1.4 - Сравнительная таблица вопросно-ответных систем и чат-ботов

Критерий	**Вопросно-ответная система (QA)**	**Чат-бот**
Основная цель	Ответ на конкретный вопрос	Ведение диалога и выполнение сценариев
Контекст общения	Не учитывается	Учитывается (через память или историю)

Архитектурная сложност	Низкая	Высокая
Примеры применения	Поиск, техподдержка, энциклопедии	Сервисы, поддержка, ассистенты
Точность ответа	Высокая	Средняя (зависит от модели)
Гибкость	Ограниченная	Высокая
Интерпретируемость	Высокая	Ограниченная
Взаимодействие с пользователем	Формальное	Персонализированное

Сравнительный анализ показывает, что вопросно-ответные системы и чат-боты решают различные задачи. QA-системы демонстрируют высокую эффективность при работе с формализованной информацией и структурированными базами знаний. Чат-боты, в свою очередь, обеспечивают более гибкое взаимодействие, поддерживают сложные сценарии и диалоговую логику, однако требуют более сложной реализации и ресурсов.

Для достижения наилучшего результата, а именно достижения идеального баланса между точностью и естественностью ответов, разработчики создают гибридные системы на основе данных подходов. Такие гибридные системы могут давать точные ответы в стиле QA, а также вести естественный и человечный диалог с пользователем, даже учитывать контекст.

1.3.4. Сценарии совместного применения

В условиях растущих требований к качеству взаимодействия между пользователем и интеллектуальной системой, всё более востребованным становится интегративный подход, сочетающий функциональные особенности как вопросно-ответных систем (QA-систем), так и диалоговых агентов (чат-ботов). Совместное применение этих подходов позволяет компенсировать

ограничения каждого из них, обеспечивая при этом высокую точность предоставляемой информации и гибкость в ведении диалога.

Один из наиболее распространённых сценариев — использование чат-бота в качестве интерфейса, который обрабатывает пользовательские обращения, выявляет сущности и намерения, а затем, при необходимости, направляет запрос в специализированную QA-систему. Такой подход позволяет:

- подстраиваться под естественный язык пользователя;
- вести диалог и при необходимости задавать уточняющие вопросы;
- извлекать точные и проверенные ответы из базы знаний или внешних источников (например, документации, базы FAQ, базы нормативных документов).

Пример: корпоративный чат-бот службы технической поддержки, который сначала уточняет проблему пользователя, а затем формирует точный запрос к базе знаний и возвращает конкретное решение.

В образовательных системах возможна интеграция чат-бота, выполняющего роль наставника или ассистента, и QA-компонента, предоставляющего чёткие ответы на тематические вопросы. Такой симбиоз обеспечивает:

- поддержку диалога по изучаемой теме;
- ответ на конкретные фактологические вопросы;
- динамическое подстраивание уровня сложности и объёма выдаваемой информации.

Пример: обучающая платформа, где чат-бот ведёт обучающую сессию, а QA-модуль отвечает на запросы типа «Что такое квантовая суперпозиция?» с использованием учебной базы данных.

В системах поддержки принятия решений в области здравоохранения важно одновременно обеспечивать точность, безопасность и удобство взаимодействия. Чат-бот может использоваться для предварительного сбора симптомов, анкетирования или консультации, а QA-модуль — для извлечения точной информации из клинических руководств или базы медицинских знаний.

Пример: чат-бот, который взаимодействует с пациентом, собирая анамнез, и обращается к QA-системе для предоставления информации о возможных диагнозах или рекомендациях по препаратам на основе руководств Всемирной организации здравоохранения (ВОЗ).

В сфере e-commerce чат-бот может обрабатывать широкий спектр пользовательских запросов — от поиска товаров до оформления заказов. При этом QA-система используется для предоставления точных ответов, например, по техническим характеристикам товара, политике возврата или наличию на складе.

Пример: пользователь спрашивает: «Какие наушники есть в продаже с шумоподавлением?». Чат-бот распознает запрос, запускает QA-модуль для фильтрации ассортимента и возвращает релевантные результаты.

Современные цифровые ассистенты в организациях нередко объединяют диалоговую оболочку (чат-бот) с системой извлечения знаний (QA), обеспечивая поддержку сотрудников в повседневной деятельности.

Пример: сотрудник запрашивает: «Как оформить командировку?». Чат-бот определяет намерение, обращается к базе регламентов через QA-модуль и возвращает пошаговую инструкцию.

Совместное использование чат-ботов и вопросно-ответных систем позволяет значительно расширить функциональность интеллектуальных интерфейсов. Диалоговая составляющая обеспечивает гибкость, адаптивность и поддержку сценариев взаимодействия, а QA-компонент гарантирует точность и достоверность предоставляемой информации. Такие гибридные архитектуры особенно актуальны в областях, где требуется как качественное пользовательское взаимодействие, так и доступ к структурированной базе знаний. В перспективе развитие подобных систем направлено на более глубокую интеграцию диалога и извлечения информации, что позволит создавать интеллектуальных агентов нового поколения — персонализированных, контекстуально осведомленных и способных к обучению.

Глава 2. МЕТОДОЛОГИЯ РАЗРАБОТКИ ЧАТ-БОТОВ

2.1 Компоненты чат-бота

Чат-боты по структуре включают в себя несколько основополагающих компонентов, они совместно реализуют функционал и взаимодействие с пользователями.

Интерфейс пользователя:

- интерфейс пользователя необходим для реализации общения между пользователем и чат-ботом. Реализация интерфейса зависит от конкретных задач поставленных перед чат-ботом. Самое важное в интерфейсе это простота и удобство, а оставшиеся требования вытекают из сферы использования чат-бота. Интерфейс может быть представлен в виде веб-приложения, десктопного или мобильного приложения, а также в виде мессенджера или даже голосового помощника.

Обработка запросов:

- компонент обработки запросов необходим для реализации понимания запросов пользователя на естественном языке. Обычно он использует технологии обработки естественного языка (NLP) для распознавания интентов, извлечения ключевых слов и анализа семантики запроса. Обработка запроса необходима для понимания пользователя и правильного ответа на его запрос.

Хранение данных:

- чат-бот может хранить различные данные, такие как профили пользователей, история диалогов и дополнительную информацию. Для хранения данных используются базы данных, но также данные могут храниться в формате JSON.

Обработка логики:

- этот компонент отвечает за логику работы чат-бота. Благодаря этому компоненту чат-бот понимает что он должен сделать в конкретной ситуации. Наибольшее влияние этот компонент имеет на реализацию дополнительных команд чат-бота, таких как вывод дополнительной информации, перенаправление к менеджеру или использование стороннего сервиса.

Интеграция с внешними сервисами:

- интеграция с внешними сервисами также является неотъемлемой частью современных чат-ботов. Это позволяет легко добавить необходимый функционал который уже реализован в стороннем сервисе. Примером таких сервисов являются сервисы погоды, интернет магазинов, баз данных и многие другие.

Аналитика и мониторинг:

- аналитика происходит для улучшения работы чат-бота, а именно для повышения производительности и обработке ошибок. В аналитику данных входит мониторинг метрик, таких как количество запросов, время ответа, количество успешных и неуспешных ответов и множество других показателей. Также для лучшего оценивания чат-бота в нем может быть реализована возможность оценки ответа от пользователя.

Конкретные компоненты и их реализация могут различаться в зависимости от выбранной платформы, технологий и требований проекта.

2.2 Принципы работы чат-бота

Существует несколько основных принципов работы чат-бота, которые в большой степени влияют на его работоспособность и эффективность.

Понимание и обработка запросов:

- основная задача чат-бота это понимать и обрабатывать запросы пользователей. В обработку запросов входит распознавание интента, извлечение ключевых слов, анализ семантики. Чат-бот должен понять чего хочет достичь пользователь и выдать ему наиболее релевантный ответ. Также чат-боту необходимо учитывать предыдущие сообщения пользователя чтобы вести контекстно-зависимые диалоги.

Генерация ответов:

- исходя из запроса пользователя чат-бот должен выдавать подходящий ответ. Ответ может быть не только текстовой реакцией на вопрос, но также может являться и каким-то действие, например обращением к стороннему сервису или выполнение внутренних действий, таких как поставить будильник, задать напоминание или сделать запрос в интернете. Ответ должен быть понятным и подходящим для пользователя.

Диалоговая логика:

- чат-бот должен поддерживать диалог, учитывая последовательность и связь сообщений. Для этого необходимо учитывать предыдущие сообщения пользователя, сохранять контекст и следить за состояниями диалога. С помощью данного подхода диалог будет ощущаться более человеческим.

Интеграция с внешними сервисами:

- исходя из конкретных задач чат-бота может потребоваться реализовать взаимодействие с внешними сервисами. Такими сервисами могут быть

сторонняя база данных, API стороннего сервиса, платёжные или CRM системы. Благодаря интеграции внешних сервисов у чат-бота появляются новые возможности, что делает его ещё более гибким, адаптивным и уникальным.

Обработка ошибок и неопределённых запросов:

- обработка ошибок необходима для улучшения качества общения, а также для того чтобы не допустить недопонимания между пользователем и чат-ботом. Чат-бот должен быть способен обрабатывать ситуации, когда он не может понять запрос пользователя или предоставить соответствующий ответ. Это включает обработку ошибок, предоставление подсказок или запрос на получение дополнительных данных для лучшего понимания и более качественной генерации ответа.

Обучение и улучшение:

- чат-боты могут постоянно улучшаться и дообучаться, например исходя из обратной связи от пользователей. Также при обучении должен происходить анализ данных использования, различных метрик его работы. При обучении необходимо также обновлять и базу знаний. С помощью постоянной доработки и обучения чат-боты будут становиться только лучше.

Выше перечислены основные принципы работы чат-бота, но при разработке также важно учитывать и контекст задачи.

2.3 Архитектуры чат-ботов

Существует несколько архитектурных подходов к разработке чат-ботов. Одна из самых простых архитектур это простая последовательная архитектура. В такой архитектуре чат-бот работает полностью последовательно. Сначала происходит обработка запроса, потом проходит этап логики диалога, а в конце генерируется ответ. В данной архитектуре не учитывается контекст диалога, из-за чего ответы могут быть не связаны между собой, а чат-бот будет давать ответ только на заданный вопрос. Из-за этого ограничения чат-боты с такой архитектурой не могут обрабатывать сложные сценарии. Такая архитектура обычно используется в чат-ботах клиентской поддержки, так как там нет необходимости учитывать весь диалог для ответа на вопросы.

Существует также архитектура с использованием стейт-машины. В такой архитектуре используется стейт-машина для того, чтобы учитывать предыдущие сообщения, это позволяет сохранять состояние между запросами и в итоге получать более связный диалог.

В архитектуре на основе правил и шаблонов главным элементом являются заранее заданные правила и шаблоны. Обычно такая архитектура используется в простых чат-ботах с ограниченным функционалом, когда заранее известно большинство сценариев взаимодействия с пользователем. Благодаря такому подходу чат-бот может быстро и точно отвечать на запросы пользователя, однако при любом не предусмотренном случае чат-бот не сможет дать подходящий ответ и перенаправит пользователя к специалисту.

Для более сложных чат-ботов используется архитектура на основе машинного обучения. В этой архитектуре применяются нейронные сети или модели глубоко обучения. Модель обучается на большом наборе данных, что позволяет выявлять шаблоны и зависимости в тексте. Благодаря этому чат-бот становится более гибким и контекстно-зависимым. Он может обрабатывать сложные запросы и адаптироваться к новым данным.

Гибридная архитектура, как видно из названия, является объединением нескольких архитектур. Например можно объединить архитектуру на основе правил и шаблонов и стейт-машины, что позволит быстро и точно отвечать на запросы при этом не теряя нить диалога. Также часто можно встретить архитектуру совмещающую в себе машинное обучение и стейт-машины, что делает чат-бота гибким к различным запросам, при этом также диалог ведётся непрерывно, не теряя нить повествования.

Конкретный выбор архитектуры зависит от требований проекта, доступных ресурсов, сложности сценариев и ожидаемого уровня взаимодействия с пользователем.

2.4 Обработка естественного языка

Обработка естественного языка (Natural Language Processing, NLP) - это область искусственного интеллекта, которая занимается взаимодействием компьютерных систем с естественным человеческим языком. Цель NLP состоит в том, чтобы позволить компьютерам понимать, интерпретировать и генерировать текстовую информацию таким образом, чтобы она была полезной и понятной для людей.

Токенизация относится к процессу разделения текста на отдельные единицы, называемые токенами. Токены могут быть словами, символами, фразами или другими единицами, в зависимости от задачи и контекста. Токенизация является первым шагом в обработке текста и позволяет разбить его на более мелкие компоненты для дальнейшего анализа.

Лемматизация и стемминг относятся к процессам приведения слов к их базовой форме. Лемматизация приводит слова к своей словарной форме

(лемме), например, приводит слово "бегущий" к основе "бежать". Стемминг приводит слова к их основе путём обрезания аффиксов, например, преобразует слово "бегущий" в "бег".

Парсинг и синтаксический анализ позволяют определить связи между словами, определить грамматические роли и построить дерево разбора. Эти технологии используются для того, чтобы чат-бот понимал семантику предложений и мог выделять ключевые элементы.

Извлечение информации включает в себя извлечение именованных сущностей (имена, организации, даты), связей между сущностями и фактов из текста. Извлечённую информацию чат-бот может использовать для заполнения базы данных или для генерации более точного ответа. В рамках дипломной работы использовалось извлечение сущностей для запоминания информации о пользователе, такой как имя, город и хобби.

Классификация и анализ тональности используются чат-ботом для предотвращения неприятных ситуаций в общении с пользователем, например если пользователь недоволен то чат-бот может снизить риск агрессии со стороны человека. Для классификации текста использовалась модель BERT. Данная модель показала точность 83%, а наибольшее число ошибок было при распознавании сарказма.

Генерация естественного языка используется для создания ответа на запрос пользователя на естественном языке. Современные модели, например GPT-4o, способны создавать связные тексты. В данной работе генерация ответов ограничивалась простыми несвязными ответами.

Обработка естественного языка используется во время всех этапов работы чат-бота. Также NLP используется и в других сферах, например в сфере маркетинга с помощью NLP обрабатываются отзывы на товары.

2.4.1 Токенизация

Токенизация по своей сути представляет из себя разновидность сегментации текста. Сегментацией является разделение документа на меньшие смысловые единицы, которые имеют ограниченное содержание. Таким образом процесс токенизации состоит из нескольких этапов, таких как разбиение текста на абзацы, потом разбиение абзацев на предложения, далее на фразы и в самом конце разбиение на токены. Токенами являются отдельные слова и знаки препинания. Пример токенизации представлен на рисунке 2.1.

Сканером (scanner) или лексическим анализатором (lexer) называют токенизатор при компиляции языков программирования. Набор всех возможных токенов называют лексиконом. В контекстно-свободных

грамматиках (CFG), которые описывают структуру языков программирования, токены являются конечными элементами. Токены называют терминалами, потому что они являются последним элементом полученном при разделении текста на части.

Самым первым шагом обработки текста в NLP является токенизация. Токенизатор разбивает текст на отдельные фрагменты (токены). Из-за того что это происходит в самом начале, этот этап очень сильно влияет на последующие этапы. Подсчитав, сколько раз каждый токен встречается в тексте, можно представить документ в виде числового вектора. Такой вектор переводит неструктурированный текст в числовую структуру данных, которая подходит для машинного обучения. Эти числовые данные могут быть сразу использованы компьютером для принятия решений или как признаки в более сложных алгоритмах. Один из самых частых способов применения таких векторов это поиск по текстам и документам.

```
sentence = "В библиотеке хранятся редкие книги и рукописи которым уже более 50 лет."
print(sentence.split())
print(str.split(sentence))
```

```
['В', 'библиотеке', 'хранятся', 'редкие', 'книги', 'и', 'рукописи', 'которым', 'уже', 'более', '50', 'лет.']
['В', 'библиотеке', 'хранятся', 'редкие', 'книги', 'и', 'рукописи', 'которым', 'уже', 'более', '50', 'лет.']
```

Рисунок 2.1 - Пример токенизации

Токенизация необходима для последующего представления предложений и слов в числовом формате, так как компьютер не может воспринимать слова как человек, поэтому необходимо все привести к более компьютеризированному формату.

2.4.2 Лемматизация и стемминг

Стемминг является группировкой различных форм слова по кластерам. Это распространённый метод нормализации, который включает в себя устранение мелких семантических различий, обусловленных изменением слов в связи с множественным числом, притяжательными формами или различными формами глаголов. Например, слова «бегать», «бегающий», «бегал» будут сведены к одной основе — «бег». Результатом стемминга будет слово, просто токен или метка, которая представляет несколько вариантов написания. Для получения результата, объединения слов с похожим значением в одну группу, из слов удаляются суффиксы, таким образом приводя их к токенам или меткам.

Лемматизация представляет из себя расширенную нормализацию слова до смыслового корня (леммы). Это помогает при наличии связей между

значениями слов связать их вместе под одним кластером, несмотря на большое различие написания.

Благодаря тому, что лемматизация уменьшает размерность языковой модели с помощью уменьшения количества слов, которые необходимо понимать, модель становится обобщённее, но и точность уменьшается, так как большое количество различных слов рассматриваются как одно слово. Например слова "писать", "пишу", "писал", "писала" и даже "написание", моделью будут рассматриваться как одно слово "писать", что может в некоторых случаях немного изменить смысл текста.

В отличие от стемминга, лемматизатор использует знания о морфологии и синонимах, благодаря чему объединяет в один токен слова близкие по значению, а не только по форме. Поэтому лемматизация считается более точным способом нормализации текста.

Двумя наиболее популярными алгоритмами являются стеммер Портера и Snowball. Примеры стеммера Портера и Snowball представлены на рисунке 2.2 и рисунке 2.3 соответственно.

```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()
print( ' '.join([stemmer.stem(w).strip("'") for w in "dish washer's washed dishes".split()]))

dish washer wash dish
```

Рисунок 2.2 -Пример стеммера Портера

```
from nltk.stem import SnowballStemmer
# Создаем экземпляр Snowball стеммера для английского языка
snowball_stemmer = SnowballStemmer("english")
# Пример слов
words = ["running", "runner", "ran", "easily", "fairly"]
# Применяем стемминг
stemmed_words_snowball = [snowball_stemmer.stem(word) for word in words]
print("Стемминг с помощью Snowball:", stemmed_words_snowball)

['run', 'runner', 'ran', 'easili', 'fair']
```

Рисунок 2.3 -Пример Snowball

Стеммер Портера — классический алгоритм стемминга для английского языка, предложенный Мартином Портером в 1980 году. Он последовательно применяет набор правил для удаления суффиксов, сводя слова к базовой форме

(например, “connection” → “connect”). Отличается простотой и высокой скоростью работы.

Snowball (или Porter2) — улучшенная версия стеммера Портера, разработанная тем же автором. Обеспечивает более точный и гибкий стемминг, поддерживает множество языков и реализован на специальном языке описания алгоритмов — Snowball. Предпочтительнее в современных системах благодаря лучшей обработке исключений и более чистым стеммам.

2.5 Машинное обучение и глубокое обучение для чат-ботов

Машинное обучение и глубокое обучение являются эффективными технологиями для разработки и улучшения чат-ботов. С помощью машинного обучения можно разрабатывать более гибкие и адаптивные чат-боты, которые также будут учитывать ещё и контекст диалога, а не только текущий запрос. Благодаря такому подходу чат-боты могут обрабатывать сложные запросы, обходить возможные ошибки, улучшать понимание запроса исходя из контекста, и в целом общаться с пользователем более человеческим и естественным способом.

Машинное обучение может применяться для классификации запросов и определении их типов. Например классификация отзывов, запросов на помощь, вопросов. Благодаря этому чат-бот может эффективнее предоставлять ответы или выполнять действия. Также машинное обучение используется в задаче распознавания именованных сущностей (NER), что позволяет чат-боту доставать из запроса ключевую информацию по типу дат, адресов, названий организаций и многого другого.

Технологии глубокого обучения, такие как рекуррентные нейронные сети (RNN) и модели генерации текста, применяются для генерации ответов. Благодаря этим технологиям чат-бот начинает общаться человечнее, потому что он учитывает контекст диалога, учитывает предыдущие запросы пользователя и также учится на данных с естественным языком.

Машинное обучение также может использоваться для анализа настроений и эмоций пользователя . Благодаря этому чат-бот понимает эмоциональную окраску текста и адаптировать свои ответы исходя из неё. Например, если пользователь выражает расстройство, недовольство или раздражение, то чат-бот может это отследить и среагировать нужным образом чтобы удовлетворить желания пользователя.

Обучение с подкреплением (reinforcement learning) используется для обучения чат-бота взаимодействовать с пользователем и дообучаться исходя из обратной связи. Чат-бот может использовать различные подходы для

оптимизации работы и максимизации своей награды. Примерами таких подходов являются генетическое программирование или Q-обучение.

Существуют предобученные модели, такие как BERT, GPT и другие, которые обучены на большом объёме текстовых данных. Эти модели используются для ускорения, упрощения и улучшения процесса разработки чат-бота, что позволяет сосредоточиться на особенностях конкретного чат-бота.

Современные методы машинного и глубокого обучения открывают новые возможности для разработки интеллектуальных чат-ботов. Эти технологии позволяют системам не только обрабатывать сложные запросы, но и анализировать контекст, что способствует формированию более точных и осмысленных ответов. Однако их внедрение сопряжено с рядом требований, включая необходимость в больших массивах размеченных данных и значительных вычислительных мощностях для обучения моделей и их последующего применения.

Вместе с техническими аспектами, важную роль играют этические вопросы, такие как обеспечение конфиденциальности пользовательских данных и минимизация алгоритмических предубеждений.

Несмотря на существующие сложности, применение методов машинного и глубокого обучения в создании чат-ботов демонстрирует высокую эффективность. Их использование способно существенно повысить качество взаимодействия пользователей с автоматизированными системами, делая его более естественным и продуктивным.

2.6 Подходы к созданию диалоговой системы

Существуют различные подходы к созданию диалоговых систем, которые выбираются исходя из конкретной задачи. Примерами таких моделей являются модель основанная на правилах, а также статистическая и гибридная модели.

Подход, основанный на правилах и шаблонах строится на заранее заданных инструкции и структуре, которые задают логику диалога. Когда чат-бот получает запрос от пользователя, он сверяет запрос с заданными шаблонами и генерирует ответ исходя их этого. Данный подход очень ограничен в своих возможностях и не может обрабатывать сложные запросы, что делает его не адаптивным, но при этом очень точным и быстрым.

В основе статистических моделей лежат методы машинного обучения и математической статистики, позволяющие обучать систему на размеченных данных для последующей генерации и классификации ответов. Обучение таких моделей может осуществляться с применением различных алгоритмов, среди которых:

- наивный байесовский классификатор;
- методы максимального правдоподобия;
- глубокие нейронные сети.

Использование этих подходов обеспечивает возможность обработки сложных пользовательских запросов с учётом контекста диалога, что повышает точность и релевантность ответов системы.

Гибридные модели являются объединением лучшего из статистических моделей и моделей основанных на правилах. Гибридные модели используют правила для обработки часто встречающихся ситуаций и заранее очевидных сценариев, а статистические модели используются для обработки сложных запросов и ведения контекстно-зависимого диалога. Благодаря совмещению этих двух моделей, гибридная модель достигает идеального баланса между точностью и гибкостью системы.

Метод обучения с подкреплением (Reinforcement Learning) применяется для создания адаптивных чат-ботов, способных обучаться в процессе взаимодействия с пользователем. В данном подходе система анализирует текущее состояние диалога, выбирает оптимальные ответы и получает обратную связь в виде числовой оценки (награды), отражающей качество её действий.

Ключевые особенности подхода:

- автономное обучение – бот самостоятельно совершенствует стратегии ведения диалога;
- максимизация награды – система оптимизирует поведение для достижения наилучших показателей;
- адаптивность – возможность подстраиваться под разнообразные сценарии общения.

Данная методика позволяет создавать диалоговые системы, которые постепенно улучшают качество взаимодействия без явного программирования каждого возможного сценария.

Чтобы значительно ускорить процесс разработки чат-бота, можно использовать заранее обученные модели, например GPT-3, BERT или другие. Предобученные модели уже умеют распознавать и генерировать текст на естественном языке и имеют свой собственный функционал, например классификация текста или другие задачи обработки естественного языка.

Каждый из этих подходом имеет свои плюсы и минусы, и чтобы выбрать правильный подход, нужно проанализировать задачу и установить требования к чат-боту, разделив их на более и менее важные. Чтобы создать качественную диалоговую систему необходимо правильно выбрать подход и также провести проектирование архитектуры, сбора и разметку данных. Также чтобы улучшить

систему необходимо оптимизировать её исходя из обратной связи пользователей.

Глава 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ЧАТ-БОТА

3.1 Определение требований к чат-боту

Формирование требований представляет собой фундаментальный этап разработки чат-бота, в ходе которого осуществляется детальное описание его будущих характеристик и возможностей. Данный процесс направлен на выявление как функциональных аспектов системы, включая базовые операции и алгоритмы обработки запросов, так и нефункциональных параметров, таких как производительность, безопасность и масштабируемость.

Особое внимание уделяется анализу потребностей целевой аудитории и согласованию ожиданий всех заинтересованных сторон. Такой подход позволяет сформировать объективные критерии оценки качества разрабатываемого решения и минимизировать риски несоответствия конечного продукта первоначальным ожиданиям.

Список требований к чат-боту включает в себя пункты, такие как:

1. Распознавание и понимание входных сообщений:
   - чат-бот должен уметь распознавать и понимать запросы пользователей.
2. Генерация ответов:
   - чат-бот должен уметь генерировать релевантный ответ на запрос пользователя.
3. Поддержка контекста:
   - чат-бот должен генерировать ответы исходя не только из запроса пользователя, но и из контекста диалога.
4. Интеграция с внешними источниками данных (опционально):
   - при необходимости в чат-бот можно интегрировать сторонние источники данных для получения более актуальных данных и генерации более точных ответов.
5. Обработка ошибок и непредвиденных ситуаций:
   - чат-бот должен быть способен адекватно обрабатывать ошибки и непредвиденные ситуации, такие как непонятные вопросы или отсутствие подходящего ответа.
6. Тестирование и итерации:
   - важно проводить тестирование чат-бота с различными входными сообщениями и проверять качество и соответствие ответов.
7. Интерфейс пользователя:
   - необходимо предусмотреть способ взаимодействия пользователей с чат-ботом, например, через командную строку, веб-интерфейс или мессенджеры.

Определение требований к чат-боту - это важный этап в разработке, который помогает уточнить ожидания и цели, определить функциональные и нефункциональные требования, а также создать основу для дальнейшего проектирования и разработки чат-бота.

3.2 Проектирование диалоговой системы

Проектирование диалоговой системы - это процесс создания структуры и логики диалога между пользователем и системой, чтобы обеспечить эффективное и естественное взаимодействие. На рисунке 3.1 представлена схема алгоритма работы разрабатываемого чат-бота.

Рисунок 3.1 - Схема алгоритма работы чат-бота

Первоначально используем библиотеку pandas для загрузки данных из файла, который содержит информацию о контекстах и ответах.

Выбираются случайные примеры из загруженных данных, чтобы увидеть примеры контекстов и ответов.

Используется метод для преобразования текстовых контекстов в числовые векторы. Это позволяет представить каждый контекст в виде числового вектора, который отражает важность каждого слова в контексте.

Используется метод для сокращения размерности числовых векторов контекстов. Это позволяет уменьшить размерность данных, сохраняя при этом наиболее значимые компоненты.

Создаётся модель, которая случайным образом выбирает одного из ближайших соседей для заданного входного контекста. Это позволяет генерировать ответы на основе похожих контекстов.

Создаётся цикл, который позволяет пользователю вводить сообщения. Для каждого ввода модель предсказывает ответ и выводит его пользователю.

Проектирование диалоговой системы - это сложный процесс, требующий анализа потребностей пользователей, понимания целей системы и создания эффективного и естественного взаимодействия. Правильное проектирование диалоговой системы помогает создать удовлетворительный пользовательский опыт и повысить эффективность системы в достижении своих целей.

3.3 Разработка и реализация чат-бота

Разработка и реализация чат-бота включает в себя несколько этапов, включая выбор подходящих технологий и инструментов для его создания.

Загрузка данных:

Используется библиотека pandas для загрузки данных из файла good.tsv. Формат файла tsv (tab-separated values) позволяет представить данные в виде таблицы, где каждая строка содержит контекст и соответствующий ответ. Метод pd.read_csv() используется для чтения данных из файла в формате tsv.

Подготовка данных:

Метод sample() используется для выборки случайных примеров из загруженных данных. Выбираются три случайных строки, чтобы увидеть примеры контекстов и ответов.

Преобразование текстовых данных:

Используется класс TfidfVectorizer из библиотеки sklearn.feature_extraction.text для преобразования текстовых контекстов в числовые векторы. TF-IDF (Term Frequency-Inverse Document Frequency) - это

метод, который позволяет оценить важность каждого слова в контексте по отношению к всей коллекции контекстов. Создаётся объект vectorizer, который будет преобразовывать короткие тексты в числовые векторы. Метод fit() вызывается на объекте vectorizer, чтобы "обучить" его на всех контекстах. В результате, объект запоминает частоту каждого слова. С помощью метода transform() преобразуется каждый контекст в числовой вектор, заполняя матрицу matrix_big.

Сокращение размерности:

Используется метод TruncatedSVD из библиотеки sklearn.decomposition для сокращения размерности матрицы matrix_big. Метод главных компонент (SVD) позволяет уменьшить размерность данных, сохраняя при этом наиболее значимые компоненты. Создается объект svd, указывая количество компонент (n_components) равное 300. Метод fit() вызывается на объекте svd, чтобы "обучить" его на матрице matrix_big. В результате, объект запоминает наиболее значимые компоненты. Метод transform() используется для сокращения размерности матрицы matrix_big в матрицу matrix_small.

Поиск ближайших соседей:

Создаётся класс NeighborSampler, который наследуется от базового класса BaseEstimator. В классе NeighborSampler определены методы fit() и predict(). Метод fit() используется для обучения модели на матрице matrix_small и соответствующих ответах good.reply. Внутри метода используется алгоритм ближайших соседей BallTree из библиотеки sklearn.neighbors. Метод predict() используется для предсказания ответа на основе входного контекста. Используется алгоритм ближайших соседей, чтобы найти ближайшие контексты к входному, и случайным образом выбираем один из них в качестве ответа.

Создание модели:

Используется функция make_pipeline() из библиотеки sklearn.pipeline, чтобы объединить шаги векторизации, сокращения размерности и поиска ближайших соседей в одну модель. Передаётся шаг преобразования текстовых данных (vectorizer), сокращения размерности (svd) и поиска ближайших соседей (NeighborSampler) в функцию make_pipeline(). Создается объект model с использованием метода make_pipeline(), объединяющий все шаги модели.

Интерактивное взаимодействие:

Создаем цикл while True, который будет выполняться бесконечно, пока пользователь не прервёт его. Внутри цикла, запрашивается у пользователя ввод контекста с помощью функции input(). Затем, вызывается метод predict() на объекте model, передавая введённый контекст. Метод предсказывает

ближайший контекст и возвращает соответствующий ответ. Выводится предсказанный ответ пользователю с помощью функции print().

Полный код программы расположен в приложении А.

Глава 4. ТЕСТИРОВАНИЕ И ОЦЕНКА ЧАТ-БОТА

4.1 Методы тестирования чат-ботов

Успешное функционирование чат-бота определяется не только грамотным проектированием его структуры и используемых алгоритмов, но и тщательной процедурой проверки его работоспособности. Комплексное тестирование системы помогает своевременно обнаруживать недочёты, совершенствовать точность распознавания запросов и гарантировать бесперебойную работу при любых условиях эксплуатации.

Для всесторонней оценки возможностей диалоговой системы применяется несколько подходов. Первый предполагает анализ базового функционала, включающего распознавание пользовательских сообщений, формирование ответных реплик, выполнение запрограммированных действий и выдачу справочных данных. Особое внимание уделяется способности системы адекватно интерпретировать поступающие запросы и выдавать соответствующие результаты.

Второй аспект проверки касается эргономики интерфейса и комфорта взаимодействия. Специалисты оценивают интуитивность управления, доступный набор опций и общее впечатление пользователей от работы с системой.

Третий важный момент – анализ диалоговой логики. Тестировщики моделируют различные ситуации общения, проверяя способность бота учитывать контекст беседы и поддерживать осмысленный диалог.

Четвёртый критерий – оценка лингвистических возможностей. Система тестируется на способность понимать разнообразные формулировки, распознавать синонимичные выражения и адекватно реагировать на вариативные запросы.

Такой многоуровневый подход к тестированию позволяет не только выявить существующие недостатки, но и значительно улучшить технические характеристики системы и качество взаимодействия с конечными пользователями.

4.2 Оценка эффективности

При тестировании чат-бота был получен результат отображённый на рисунке 4.1 и рисунке 4.2. На этих рисунках отображены варианты работы чат-бота при разных и одинаковых входных данных.

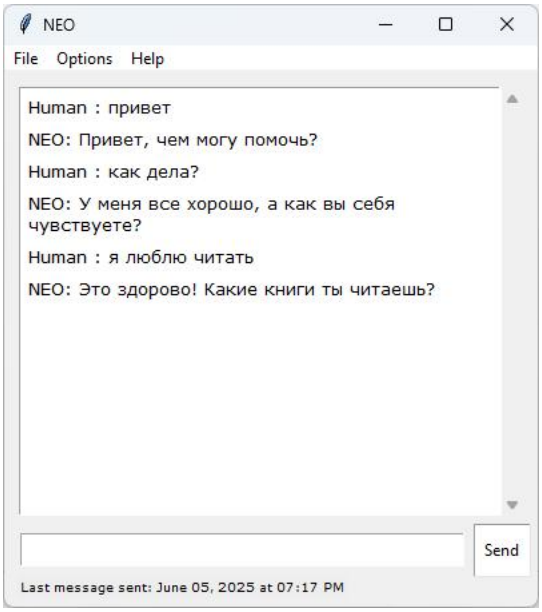

Рисунок 4.1 - Результат работы чат-бота при разных входных данных

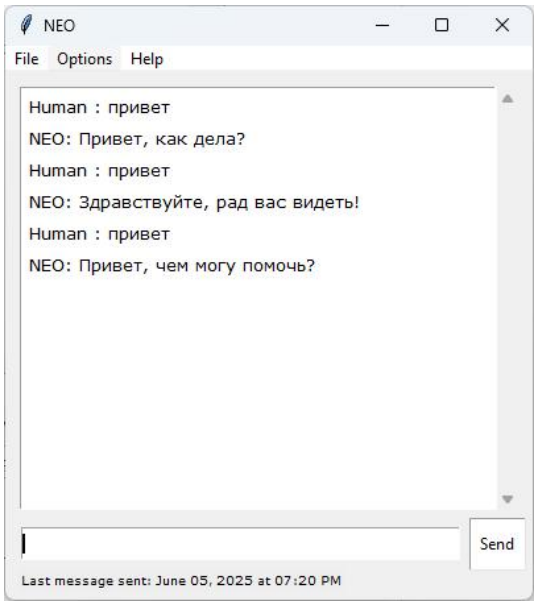

Рисунок 4.2 - Результат работы чат-бота при одинаковых входных данных

Из результата тестирования можно сделать вывод что чат-бот хорошо распознает полученный текст и контекст, также он может чётко отвечать на запросы пользователя. Из рисунка 4.2 видно, что чат-бот не имеет однозначного ответа на конкретные сообщения пользователя, и отвечает случайным образом выбирая подходящий вариант.

ЗАКЛЮЧЕНИЕ

В данной дипломной работе был разработан и реализован чат-бот, основанный на методе ближайших соседей, который позволяет отвечать на вопросы пользователей на основе контекста и предоставленных данных. Были выполнены следующие шаги: векторизация текста, сокращение размерности, поиск ближайших соседей и случайный выбор ответа.

Проведённые тесты подтвердили работоспособность чат-бота: система корректно интерпретирует входные запросы и генерирует осмысленные ответы. Однако выявлена зависимость от случайного выбора среди ближайших соседей, что может приводить к вариативности ответов на идентичные вопросы. Также благодаря векторизации текста и сокращению размерности ответы чат-бота происходят без задержек.

Дальнейшее развитие данного чат-бота будет включать в себя расширение базы знаний, оптимизацию гиперпараметров моделей машинного обучения. Также будет добавлен голосовой ввод и вывод, чтобы чат-бот мог использоваться как голосовой помощник. Будет расширен функционал взаимодействия с компьютерной системой, например изменение громкости, запуск приложений. Также будет улучшен поиск наименованных сущностей.

Разработанный чат-бот демонстрирует потенциал для применения в таких областях, как клиентская поддержка, образовательные платформы и сфере развлечений. Дальнейшая работа по его усовершенствованию позволит создать более адаптивную и точную систему, способную значительно улучшить качество взаимодействия с пользователями.

Результаты исследования подчёркивают важность комбинации классических методов машинного обучения с современными подходами к обработке естественного языка, что открывает новые возможности для разработки интеллектуальных диалоговых систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Боярский К. К. Введение в компьютерную лингвистику. Учебное пособие. – СПб: НИУ ИТМО, 2013. – 72 с.
2. Обработка естественного языка в действии. — СПб.: Питер, 2020. — 576 с.: ил. — (Серия «Для профессионалов»).
3. Создаем чат-бота на Python: Полное руководство [интернет ресурс] https://habr.com/ru/articles/792148/
4. 15 лучших наборов данных для обучения чат-бота [интернет ресурс] https://habr.com/ru/companies/skillfactory/articles/531734/
5. ConvAI3: Clarifying Questions for Open-Domain Dialogue Systems http://convai.io/data/?ref=hackernoon.com [интернет ресурс]

ПРИЛОЖЕНИЕ А

```
import pandas as pd # to Python. We have loaded a library for working with tables
good = pd.read_csv('good.tsv', sep='\t') # read data from disk

good.sample(3) # three random strings were shown: context and

# Превращаем тексты в числовые векторы
# импортируем sklearn - самую популярную библиотеку с машинным обучением
from sklearn.feature_extraction.text import TfidfVectorizer
# создаём обьект, который будет преобразовывать короткие в числовые векторы
vectorizer = TfidfVectorizer()
# "обучаем" его на всех контекстах -> запоминаем частоту каждого слова
vectorizer.fit(good.context_0)
# записываем в матрицу, сколько раз каждое слово встречалось в каждом тексте
matrix_big = vectorizer.transform(good.context_0)


# импортируем алгоритм, известный как Метод главных компонент
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=300)

# чтобы сохранить максимум информации об исходной матрице
svd.fit(matrix_big)
matrix_small = svd.transform(matrix_big)


# (1) Пишем класс для случайного выбора среди подходящих ответов
import numpy as np
from sklearn.neighbors import BallTree
from sklearn.base import BaseEstimator

def softmax(x):
    """ Функция для создания вероятностного распрделения """
    proba = np.exp(-x)
    return proba / sum(proba)
```

```
class NeighborSampler(BaseEstimator):
    """ Класс для случайного выбора одного из ближайших соседей"""
    def __init__ (self, k=5, temperature=1.0):
    self.k = k
    self.temperature = temperature
    def fit(self, X, y):
    self.tree_ = BallTree(X)
    self.y_ = np.array(y)
    def predict (self, X, random_state=None):
    distances, indices = self.tree_.query(X, return_distance=True, k=self.k)
    result = []
    for distance, index in zip(distances, indices):
    result.append(np.random.choice(index, p=softmax(distance *
self.temperature)))
    return self.y_[result]


# (2) Соединяем векторизацию, сокращение размерности, и поиск соседей
from sklearn.pipeline import make_pipeline
ns = NeighborSampler()
ns.fit(matrix_small, good.reply)
pipe = make_pipeline(vectorizer, svd, ns)

while True:
    user_input = input("Введите сообщение (для выхода введите 'выход'): ")

    if user_input.lower() == 'выход':
    break

    reply = pipe.predict([user_input])
    print(reply[0])
```