Geekbrains

**«Разработка интеллектуального помощника для техподдержки российского разработчика криптошлюзов S-Terra на основе методов Data Science»**

Программа: Искусственный интеллект. Специалист

Евсик Оксана Станиславовна

Москва

2025

**Оглавление**

[Введение 5](#_Toc199455085)

[I. Теоретическая часть 8](#_Toc199455086)

[1. Введение в сферу технической поддержки 8](#_Toc199455087)

[1.1 Роль технической поддержки в ИТ-компаниях 8](#_Toc199455088)

[1.2 Особенности поддержки в области криптографических решений 8](#_Toc199455089)

[1.3 Типология обращений 9](#_Toc199455090)

[2. Актуальность интеллектуализации поддержки 10](#_Toc199455091)

[2.1 Рост объема обращений и нагрузка на персонал 10](#_Toc199455092)

[2.2 Повторяемость типовых запросов и потенциал автоматизации 10](#_Toc199455093)

[2.3 Повышение требований к SLA и скорости обработки 10](#_Toc199455094)

[2.4 Цифровая трансформация и внедрение ИИ-инструментов 11](#_Toc199455095)

[3. Основы Data Science в задачах интеллектуальной обработки текста 12](#_Toc199455096)

[3.1 Этапы Data Science-проекта 12](#_Toc199455097)

[3.2 Обзор методов NLP (Natural Language Processing). Предобработка текста 12](#_Toc199455098)

[3.3. Ранжирование и генерация ответов 21](#_Toc199455099)

[4. Обзор интеллектуальных помощников и чат-ботов 24](#_Toc199455100)

[4.1 Виды интеллектуальных помощников 24](#_Toc199455101)

[4.2 Архитектура помощника в технической поддержке 25](#_Toc199455102)

[4.3 Сравнение подходов 26](#_Toc199455103)

[5. Обзор применяемых технологий и библиотек 28](#_Toc199455104)

[5.1. Языки и инструменты 28](#_Toc199455105)

[5.2. Модели для русского языка 29](#_Toc199455106)

[6. Безопасность и ограничения в криптосреде 30](#_Toc199455107)

[6.1. Ограничения на использование облачных сервисов 30](#_Toc199455108)

[6.2. Необходимость локального развертывания 30](#_Toc199455109)

[6.3. Поддержка закрытого (автономного) контура 31](#_Toc199455110)

[6.4. Интерпретируемость и доверие к ИИ 31](#_Toc199455111)

[7. Резюме теоретических основ 33](#_Toc199455112)

[7.1. Обоснование применимости интеллектуальных решений 33](#_Toc199455113)

[7.2. Учет специфики криптосреды и информационной безопасности 33](#_Toc199455114)

[7.3. Оптимальная стратегия — гибридный подход 33](#_Toc199455115)

[II. Практическая часть 35](#_Toc199455116)

[1. Постановка задачи 35](#_Toc199455117)

[1.1 Цель работы 35](#_Toc199455118)

[1.2 Исходные данные 35](#_Toc199455119)

[1.3 Основные задачи проекта 36](#_Toc199455120)

[1.4 Ожидаемые результаты 36](#_Toc199455121)

[2. Сбор и подготовка данных 37](#_Toc199455122)

[2.1 Источник данных 37](#_Toc199455123)

[2.2 Формат и структура данных 37](#_Toc199455124)

[2.3 Этапы подготовки данных 38](#_Toc199455125)

[2.4 Примеры трансформированных данных 39](#_Toc199455126)

[2.5 Инструменты и библиотеки 39](#_Toc199455127)

[3. Анализ данных (Exploratory Data Analysis, EDA) 40](#_Toc199455128)

[3.1 Распределение тем обращений 40](#_Toc199455129)

[3.2 Частотный анализ слов 40](#_Toc199455130)

[3.3 Средняя длина тикета 41](#_Toc199455131)

[3.4 Временные закономерности 41](#_Toc199455132)

[3.5 Количество обращений по клиентам/продуктам 41](#_Toc199455133)

[3.6 Примеры типовых запросов 41](#_Toc199455134)

[4. Модель тематической классификации 43](#_Toc199455135)

[4.1 Цель 43](#_Toc199455136)

[4.2 Подходы к классификации 43](#_Toc199455137)

[4.3 Обучение моделей 43](#_Toc199455138)

[4.4 Результаты экспериментов 44](#_Toc199455139)

[4.5 Confusion Matrix (матрица ошибок) 44](#_Toc199455140)

[4.6 Интерпретация ошибок 44](#_Toc199455141)

[5. Поиск похожих тикетов (retrieval-based ответчик) 46](#_Toc199455142)

[5.1 Цель 46](#_Toc199455143)

[5.2 Модель: поиск по эмбеддингам тикетов 46](#_Toc199455144)

[5.3 Построение индекса с помощью FAISS или Annoy 46](#_Toc199455145)

[5.4 Механизм работы системы 47](#_Toc199455146)

[5.5 Оценка качества поиска 47](#_Toc199455147)

[5.6 Результаты 48](#_Toc199455148)

[6. Интеграция с базой знаний 51](#_Toc199455149)

[6.1 Цель 51](#_Toc199455150)

[6.2 Векторный поиск по статьям/FAQ 51](#_Toc199455151)

[6.3 Матчинг запроса с наиболее подходящими статьями 52](#_Toc199455152)

[6.4 Пример 53](#_Toc199455153)

[6.5 Оценка качества поиска по базе знаний 53](#_Toc199455154)

[6.6 Итог 54](#_Toc199455155)

[7. Оптимизация системы поиска по базе знаний 55](#_Toc199455156)

[7.1. Использование различных техник векторизации 55](#_Toc199455157)

[7.2. Методы ранжирования для улучшения точности поиска 56](#_Toc199455158)

[7.3. Обучение модели на реальных данных 57](#_Toc199455159)

[7.4. Использование кластеризации для улучшения качества поиска 58](#_Toc199455160)

[7.5. Итоги оптимизации 58](#_Toc199455161)

[8. Разработка интерфейса интеллектуального помощника 59](#_Toc199455162)

[8.1 Варианты интерфейса 59](#_Toc199455163)

[8.2 Функциональные возможности интерфейса 59](#_Toc199455164)

[8.3 Архитектура системы 59](#_Toc199455165)

[8.4 Пример реализации на FastAPI (Backend) 60](#_Toc199455166)

[8.5 Telegram-бот (Пример) 60](#_Toc199455167)

[8.6 Подключение к внутренней CRM 61](#_Toc199455168)

[8.7 Возможности для расширения 61](#_Toc199455169)

[9. Результаты и оценка 62](#_Toc199455170)

[9.1 Повышение скорости обработки запросов 62](#_Toc199455171)

[9.2 Метрики моделей 62](#_Toc199455172)

[9.3 Оценка сотрудниками технической поддержки 62](#_Toc199455173)

[9.4 Примеры успешных сессий (кейсы) 63](#_Toc199455174)

[Заключение 65](#_Toc199455175)

[Список использованной литературы 67](#_Toc199455176)

[Приложение 1. 68](#_Toc199455177)

[Приложение 2. 73](#_Toc199455178)

# **Введение**

В условиях стремительного развития цифровых технологий и глобальной цифровизации все большее значение приобретают вопросы информационной безопасности. Государственные учреждения, финансовые структуры, предприятия критической информационной инфраструктуры — все они нуждаются в защищенных каналах передачи данных, особенно с учетом возрастающих угроз кибератак, утечек конфиденциальной информации и требований регуляторов. В этой связи особую роль играют российские производители средств защиты информации, разрабатывающие сертифицированные криптографические решения. Одним из таких ведущих разработчиков является компания S-Terra, специализирующаяся на создании программно-аппаратных комплексов, криптошлюзов и VPN-решений, соответствующих требованиям ФСТЭК и ФСБ России.

Наряду с разработкой и внедрением продукции не менее важной задачей становится эффективная техническая поддержка пользователей. Сложность и критичность используемых решений накладывают особые требования к качеству работы службы поддержки: она должна быть не только технически грамотной, но и быстрой, конфиденциальной, отказоустойчивой. При этом, по мере расширения клиентской базы, роста числа продуктов и сценариев их использования, значительно увеличивается объем обращений в техподдержку. Пользователи обращаются с разнообразными вопросами: от базовой настройки VPN/IPSec до устранения сложных сбоев, получения лицензий, совместимости с другими продуктами и установки обновлений.

В подобных условиях традиционные средства обработки обращений (например, через почтовые тикеты, телефонные звонки или простые CRM-системы) становятся неэффективными. Специалисты службы поддержки сталкиваются с высокой нагрузкой, необходимостью ручной классификации обращений, сложностью поиска решений по базе знаний, что неизбежно приводит к увеличению времени обработки заявок, росту ошибок и снижению удовлетворенности клиентов.

Одним из наиболее перспективных направлений повышения эффективности техподдержки является интеллектуализация процессов обработки обращений — внедрение инструментов, основанных на методах машинного обучения (ML) и обработки естественного языка (NLP). Такие инструменты позволяют автоматически анализировать входящие запросы, классифицировать их по тематикам, искать аналогичные инциденты в архиве, предлагать релевантные статьи из базы знаний, а также генерировать предварительные ответы. Это позволяет существенно снизить нагрузку на операторов, ускорить решение типовых инцидентов и обеспечить более высокий уровень соблюдения SLA.

В рамках данной дипломной работы предполагается разработка прототипа интеллектуального помощника для техподдержки S-Terra, который будет функционировать в условиях ограниченной среды (без подключения к публичным облачным API), соответствовать требованиям безопасности и конфиденциальности, а также использовать наработки из области Data Science.

Цель работы — создание системы, способной:

* автоматизировать классификацию обращений по типам (например, установка, настройка, ошибка, лицензирование);
* выполнять поиск похожих запросов в архиве тикетов;
* извлекать и предлагать наиболее релевантные статьи из внутренней базы знаний;
* взаимодействовать с оператором через чат-интерфейс.

Для достижения цели в работе решаются следующие задачи:

1. Сбор и предобработка данных из архивов техподдержки и базы знаний.
2. Анализ частотных закономерностей и тематики обращений.
3. Построение и обучение моделей классификации запросов.
4. Реализация системы поиска похожих тикетов на основе векторных представлений.
5. Интеграция с базой знаний и реализация механизма ранжирования релевантных документов.
6. Разработка пользовательского интерфейса (веб или Telegram-бот).
7. Оценка качества работы интеллектуального помощника по метрикам точности, скорости и полезности для конечного пользователя.

Практическая значимость данной работы заключается в возможности её реального применения в ИТ-подразделении S-Terra или аналогичных организациях. Внедрение интеллектуального помощника позволит:

* существенно сократить время обработки типовых обращений;
* повысить точность и полноту ответов за счет использования архивной информации;
* улучшить навигацию по базе знаний;
* освободить ресурсы операторов для решения более сложных и нестандартных задач.

Кроме того, спроектированная архитектура решения учитывает перспективу масштабирования — например, внедрение голосовых интерфейсов, генеративных моделей ответов (на основе LLM), подключение к изолированным GPT-моделям, обученным на внутренних данных.

Таким образом, данная работа представляет собой исследование и реализацию подхода к интеллектуализации технической поддержки в условиях ограничений по безопасности, с использованием современных методов Data Science и NLP. Результаты могут быть полезны не только для S-Terra, но и для других организаций, стремящихся к цифровой трансформации своих процессов поддержки.

1. **Теоретическая часть**

## *1. Введение в сферу технической поддержки*

### *1.1 Роль технической поддержки в ИТ-компаниях*

Техническая поддержка является неотъемлемой частью жизненного цикла любого программного продукта, особенно в высокотехнологичных отраслях, таких как информационная безопасность. Её основная задача — оперативное и квалифицированное решение проблем, с которыми сталкиваются конечные пользователи продуктов, а также консультирование по вопросам установки, настройки и эксплуатации.

В условиях высокой конкуренции уровень технической поддержки напрямую влияет на удовлетворённость клиентов, лояльность и восприятие бренда. Особенно это актуально для компаний, предоставляющих решения в области защиты информации, где от своевременности и корректности ответа может зависеть безопасность критической инфраструктуры.

### *1.2 Особенности поддержки в области криптографических решений*

ИТ-компании, разрабатывающие продукты в сфере криптографической защиты (например, VPN-шлюзы, средства шифрования, сертифицированные СКЗИ), сталкиваются с рядом специфических вызовов при организации службы поддержки:

* **Работа с конфиденциальной информацией.** Специалисты технической поддержки регулярно обрабатывают обращения, содержащие сведения о внутренней инфраструктуре клиентов, настройках сетей, политике безопасности. Это накладывает строгие требования к защите данных, соблюдению регламентов по безопасности и использованию закрытых каналов связи.
* **Высокая техническая сложность запросов.** В отличие от массовых программных продуктов, криптографические решения предназначены для ИТ-специалистов, инженеров и администраторов. Запросы часто касаются нюансов криптографических протоколов (IPSec, TLS), межсетевых взаимодействий, авторизации, маршрутизации и других низкоуровневых аспектов.
* **Критичность точности и скорости ответа.** Ошибка в конфигурации может привести к нарушению безопасности или отказу системы. Поэтому специалисты поддержки должны не только обладать глубокими техническими знаниями, но и иметь доступ к полной базе знаний, актуальным инструкциям, тестовым стендам.
* **Ограничения на использование внешних решений.** В целях импортозамещения и информационной безопасности использование зарубежных сервисов (в том числе облачных) может быть ограничено. Это затрудняет применение облачных NLP-моделей, внешних систем логирования или аналитики — и требует локальной, автономной реализации интеллектуального помощника.

### *1.3 Типология обращений*

На основе анализа архивов технической поддержки компаний, работающих в сфере информационной безопасности, можно выделить следующие основные категории обращений:

* **Настройка VPN/IPSec.** Один из наиболее частых видов обращений, включающий настройку туннелей, маршрутов, политик шифрования, параметров SA и сертификатов.
* **Ошибки в работе ПО.** Сюда входят сообщения о нестабильной работе криптошлюза, сбоях при загрузке, падениях служб, а также расшифровки логов.
* **Лицензирование.** Запросы на активацию лицензий, продление сроков действия, подключение новых модулей.
* **Обновление прошивки / ПО.** Вопросы, связанные с переходом на новые версии программного обеспечения, совместимостью, регламентами обновлений.
* **Совместимость и интеграции.** Настройка взаимодействия с другими системами: Active Directory, межсетевые экраны, другие криптосредства.
* **Вопросы сертификации и соответствия требованиям регуляторов.** Подтверждение наличия сертификатов ФСТЭК/ФСБ, работа в рамках ГОСТ, выполнение требований по защите информации.

## *2. Актуальность интеллектуализации поддержки*

В последние годы техническая поддержка претерпевает серьёзные изменения под влиянием как технологических, так и организационных факторов. Увеличение объема клиентов, повышение требований к скорости обслуживания и рост сложности продуктов приводят к необходимости внедрения интеллектуальных решений, способных поддерживать стабильное качество поддержки без эквивалентного увеличения штата.

### *2.1 Рост объема обращений и нагрузка на персонал*

С развитием ИТ-продуктов и расширением клиентской базы компаний, работающих в сфере информационной безопасности, существенно увеличивается количество входящих запросов. Особенно это характерно для организаций, предлагающих решения уровня enterprise, где каждый заказчик требует индивидуального подхода. Ручная обработка множества тикетов становится затруднительной:

* В часы пик операторы перегружены, из-за чего увеличиваются задержки в ответах.
* Устойчиво растёт доля обращений, требующих привлечения технических специалистов, что затрудняет масштабирование.

Это вызывает необходимость автоматизации первичного анализа и маршрутизации тикетов, а также подготовки шаблонных ответов для часто встречающихся ситуаций.

### *2.2 Повторяемость типовых запросов и потенциал автоматизации*

В значительной части обращений наблюдаются устойчивые паттерны: одни и те же проблемы, одни и те же вопросы от разных клиентов. Например:

* «Как настроить туннель IPSec между двумя шлюзами?»
* «Почему не проходит аутентификация по сертификату?»
* «Какая минимальная версия прошивки поддерживает протокол ГОСТ Р 34.10-2012?»

По оценкам компаний в отрасли, до **30–50%** всех обращений — это вариации на уже решённые случаи. Это открывает широкие возможности для внедрения систем интеллектуальной поддержки на основе машинного обучения и семантического поиска, способных:

* Автоматически классифицировать запрос по тематике.
* Сопоставить его с ранее обработанными инцидентами.
* Предложить готовый ответ из базы знаний или из архива тикетов.

### *2.3 Повышение требований к SLA и скорости обработки*

SLA (Service Level Agreement, соглашение об уровне обслуживания) сегодня включает жёстко регламентированные показатели: максимальное время реакции, время решения инцидента, уровень удовлетворённости клиента. С ростом цифровизации и бизнес-критичности ИБ-продуктов эти требования усиливаются.

Интеллектуальные помощники помогают достигать SLA-метрик за счёт:

* **Сокращения времени первичной обработки.**
* **Повышения точности классификации инцидентов.**
* **Поддержки сотрудников подсказками и готовыми ответами.**

### *2.4 Цифровая трансформация и внедрение ИИ-инструментов*

Переход к цифровой трансформации в компаниях — это не просто автоматизация процессов, но интеграция интеллектуальных технологий в повседневную деятельность. В ИТ-поддержке это проявляется в:

* Внедрении чат-ботов, способных обрабатывать простые запросы.
* Использовании систем машинного обучения для анализа истории обращений.
* Применении нейросетей для обработки текста (NLP) и извлечения смысловой информации.

Согласно трендам Gartner и Forrester, уже к 2026 году до **70% обращений в техподдержку** в корпоративном секторе будут обрабатываться с применением ИИ-компонентов — от автоматизированных ассистентов до генеративных моделей.

## *3. Основы Data Science в задачах интеллектуальной обработки текста*

Обработка естественного языка (Natural Language Processing, NLP) — одно из ключевых направлений в Data Science, применяемое в задачах интеллектуальных помощников, рекомендательных систем, классификации текста, чат-ботов и др. В контексте технической поддержки, NLP позволяет автоматизировать анализ тикетов, сопоставление с базой знаний, генерацию ответов и классификацию обращений.

### *3.1 Этапы Data Science-проекта*

Работа по созданию интеллектуального помощника строится по типичному циклу Data Science-проекта:

* **Постановка задачи.** Определяется бизнес-цель проекта — например, автоматизация обработки обращений в техподдержку, классификация тикетов, рекомендация релевантных ответов.
* **Сбор и подготовка данных.** Извлекаются и структурируются данные из различных источников: CRM-систем, логов обращений, базы знаний. На этом этапе производится очистка и нормализация текстов.
* **Анализ и визуализация.** Выполняется разведочный анализ данных (EDA): частотность тем, длина тикетов, пики нагрузки, ключевые слова. Используются инструменты визуализации: matplotlib, seaborn, Plotly.
* **Построение и обучение моделей.** Создаются модели машинного обучения или нейросетевые архитектуры для решения задач классификации, поиска, генерации ответов. Производится обучение, настройка гиперпараметров.
* **Интерпретация результатов.** Анализируются метрики (accuracy, F1-score), строятся confusion matrix, изучаются ошибки модели. Возможно использование методов интерпретации — SHAP, LIME.
* **Развертывание и мониторинг** Модель интегрируется в рабочее окружение (веб-интерфейс, API, боты), настраивается логирование, мониторинг производительности, обратная связь от пользователей.

### *3.2 Обзор методов NLP (Natural Language Processing). Предобработка текста*

Подготовка текста перед анализом (NLP — Natural Language Processing) включает несколько ключевых этапов, которые улучшают качество данных и упрощают дальнейшую обработку. Рассмотрим каждый из них подробно.

*3.2.****1. Токенизация (Tokenization)***

Цель: Разбить текст на отдельные слова, фразы или символы (токены).

Методы токенизации:

* Разделение по пробелам и знакам препинания Простейший способ, но может некорректно обрабатывать сокращения ("т.д." → "т", "д") и сложные слова.
* Использование регулярных выражений Позволяет гибко задавать правила разбиения (например, учитывать апострофы в английском: "don't" → "do", "n't").
* Готовые библиотеки
  + nltk (Natural Language Toolkit) — word\_tokenize()
  + spaCy — встроенный токенизатор
  + razdel (для русского языка) — учитывает морфологию
  + **Пример (Python):**
* from nltk.tokenize import word\_tokenize
* text = "Токенизация — это важно! Проверим?"
* tokens = word\_tokenize(text) # ['Токенизация', '—', 'это', 'важно', '!', 'Проверим', '?']

*3.2.****2. Лемматизация (Lemmatization)***

Цель: Привести слова к их нормальной (словарной) форме.

Библиотеки для русского языка:

* pymorphy2.

Быстрая и точная лемматизация с учетом контекста.

import pymorphy2

morph = pymorphy2.MorphAnalyzer()

word = "бежали"

lemma = morph.parse(word)[0].normal\_form # "бежать"

* Natasha

Комплексный инструмент для NLP, включая лемматизацию и извлечение именованных сущностей.

from natasha import MorphVocab

morph\_vocab = MorphVocab()

word = "стола"

lemma = morph\_vocab.word\_to\_forms(word)[0].lemma # "стол"

Отличие от стемминга:

* Стемминг (например, SnowballStemmer) просто отрезает окончания, но не всегда дает корректную словарную форму ("бежали" → "бежа").
* Лемматизация точнее, но требует больше ресурсов.

*3.2.****3. Удаление стоп-слов (Stop Words Removal)***

Цель: Удалить часто встречающиеся, но малосодержательные слова (артикли, предлоги, частицы).

Списки стоп-слов:

* Готовые наборы в библиотеках:
  + nltk.corpus.stopwords (для английского и других языков)
  + spacy.lang.ru.stop\_words (для русского)
  + sklearn.feature\_extraction.text.STOP\_WORDS

Пример:

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('russian')) # {'и', 'в', 'не', 'на', ...}

filtered\_tokens = [word for word in tokens if word.lower() not in stop\_words]

Важно:

* Иногда стоп-слова могут быть значимы (например, "не" в тональном анализе).
* Можно настраивать списки вручную.

*3.2.****4. Очистка от HTML, спецсимволов и логов***

Цель:

Удалить технический "мусор" из текста (HTML-теги, спецсимволы, логи), оставив только полезный человекочитаемый текст.

Методы очистки:

* Удаление HTML-тегов

Используйте библиотеку BeautifulSoup:

from bs4 import BeautifulSoup

clean\_text = BeautifulSoup(html\_text, "lxml").get\_text()

Альтернатива с помощью регулярного выражения:

import re

clean\_text = re.sub(r'<[^>]+>', '', html\_text)

* Удаление спецсимволов и цифр

Удалить всё, кроме букв и пробелов:

clean\_text = re.sub(r'[^а-яА-ЯёЁa-zA-Z\s]', '', text)

* Удаление логов и шаблонных фрагментов

Удаление времени:

clean\_text = re.sub(r'\d{2}:\d{2}:\d{2}', '', text) # например, 12:34:56

Удаление IP-адресов:

clean\_text = re.sub(r'\b\d{1,3}(?:\.\d{1,3}){3}\b', '', text)

Удаление строк с логами (ERROR, DEBUG и др.):

lines = text.splitlines()

clean\_text = '\n'.join(line for line in lines if not re.search(r'ERROR|DEBUG|TRACE', line, re.IGNORECASE))

* Нормализация пробелов

Удалить лишние пробелы, табуляции, переводы строк:

clean\_text = ' '.join(text.split())

Результат:

Очищенный от HTML, логов и "шума" текст

*3.2.****5. Векторизация текста (Text Vectorization)***

Цель: Преобразовать текстовые данные в числовой формат, пригодный для машинного обучения.

Методы векторизации:

*3.2.5.1. Bag of Words (BoW) — Мешок слов*

Суть:

* Текст представляется в виде вектора, где каждое измерение соответствует частоте слова в документе.
* Не учитывает порядок слов и семантику.

Пример:

Документ 1: "Кот ест рыбу"

Документ 2: "Собака ест мясо"

Словарь (уникальные слова): [кот, ест, рыбу, собака, мясо]

BoW-векторы:

* Док 1: [1, 1, 1, 0, 0]
* Док 2: [0, 1, 0, 1, 1]

Плюсы:

* Простота реализации.
* Хорошо работает для небольших корпусов.

Минусы:

* Высокая размерность (проблема разреженности).
* Нет учета важности слов.

Библиотеки:

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(["Кот ест рыбу", "Собака ест мясо"])

print(X.toarray()) # [[1 1 1 0 0], [0 1 0 1 1]]

* + - 1. *TF-IDF (Term Frequency – Inverse Document Frequency)*

Суть:

* Учитывает не только частоту слова в документе (TF), но и его редкость в корпусе (IDF).
* Формула:

где:

* + TF(t, d) — частота термина t в документе d.
  + DF(t) — число документов, где встречается t.
  + N — общее число документов.

Пример:

Документ 1: "Кот ест рыбу"

Документ 2: "Собака ест мясо"

TF-IDF векторы (примерные значения):

* Док 1: [0.58, 0.4, 0.58, 0, 0] (слово "кот" и "рыбу" важнее, чем "ест")
* Док 2: [0, 0.4, 0, 0.58, 0.58]

Плюсы:

* Уменьшает вес частых, но маловажных слов (например, "и", "в").
* Лучше отражает значимость терминов.

Минусы:

* Все ещё не учитывает контекст.

Библиотеки:

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(["Кот ест рыбу", "Собака ест мясо"])

print(X.toarray())

3.2.5.3. Word Embeddings (Word2Vec, FastText, GloVe)

Суть:

* Слова представляются в виде плотных векторов (эмбеддингов) в многомерном пространстве (обычно 50-300 измерений).
* Схожие по смыслу слова имеют близкие векторы.

Модели:

* Word2Vec (Google) — обучается на предсказании слов по контексту (Skip-gram, CBOW).
* FastText (Facebook) — учитывает подстроки (n-grams), лучше для редких слов.
* GloVe (Stanford) — обучается на статистике совместной встречаемости слов.

Пример Word2Vec:

from gensim.models import Word2Vec

sentences = [["кот", "ест", "рыбу"], ["собака", "ест", "мясо"]]

model = Word2Vec(sentences, vector\_size=100, window=5, min\_count=1)

vector = model.wv["кот"] # вектор размерности 100

Плюсы:

* Учитывает семантику слов.
* Можно использовать для поиска синонимов (king - man + woman ≈ queen).

Минусы:

* Требует больших данных для обучения.
* Не учитывает контекст предложения (один вектор для каждого слова).

*3.2.5.4. Transformer-based Embeddings (BERT, ruBERT, sBERT)*

Суть:

* Современные нейросетевые модели, которые учитывают контекст слова в предложении.
* Генерируют контекстные эмбеддинги (вектор зависит от окружения слова).

Примеры моделей:

* BERT (Google) — двунаправленная модель на трансформерах.
* ruBERT — русскоязычная версия BERT.
* sBERT (Sentence-BERT) — оптимизирован для сравнения предложений.

Пример использования BERT:

from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from\_pretrained('bert-base-multilingual-cased')

model = BertModel.from\_pretrained('bert-base-multilingual-cased')

inputs = tokenizer("Кот ест рыбу", return\_tensors="pt")

outputs = model(\*\*inputs)

embeddings = outputs.last\_hidden\_state # Тензор [1, 5, 768] (5 токенов, 768-мерные векторы)

Плюсы:

* Учитывает контекст (вектор слова зависит от предложения).
* Лучшее качество для сложных задач (например, анализ тональности).

Минусы:

* Требует GPU для эффективной работы.
* Большие модели (сотни мегабайт).

***3.2.6. Классификация текста (Text Classification)***

Цель: Отнести текст к одной из заданных категорий (например, тематика тикета).

*3.2.7****. Классические алгоритмы ML***

Наивный Байес (Naive Bayes)

* Основан на теореме Байеса и предположении о независимости признаков.
* Быстрый, но не всегда точный.

Пример:

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import Pipeline

model = Pipeline([

('tfidf', TfidfVectorizer()),

('clf', MultinomialNB())

])

model.fit(X\_train, y\_train)

Логистическая регрессия (Logistic Regression)

* Линейная модель с сигмоидной функцией.
* Хорошо работает с TF-IDF.

Пример:

from sklearn.linear\_model import LogisticRegression

model = Pipeline([

('tfidf', TfidfVectorizer()),

('clf', LogisticRegression())

])

Деревья решений и ансамбли (XGBoost, CatBoost)

* Градиентный бустинг (XGBoost, LightGBM, CatBoost) часто дает лучшие результаты.
* Может работать с TF-IDF или готовыми эмбеддингами.

Пример:

from xgboost import XGBClassifier

model = Pipeline([

('tfidf', TfidfVectorizer()),

('clf', XGBClassifier())

])

*3.2.8****. Нейросетевые подходы***

Рекуррентные сети (LSTM, GRU)

* Учитывают последовательность слов.
* Хорошо работают с Word2Vec/FastText эмбеддингами.

Пример:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding

model = Sequential([

Embedding(input\_dim=vocab\_size, output\_dim=100),

LSTM(128),

Dense(1, activation='sigmoid')

])

Трансформеры (BERT, DistilBERT)

* Fine-tuning предобученных моделей (например, ruBERT).
* Максимальное качество, но требует GPU.

Пример:

from transformers import BertForSequenceClassification, Trainer

model = BertForSequenceClassification.from\_pretrained('bert-base-multilingual-cased')

trainer = Trainer(model=model, args=training\_args, train\_dataset=train\_dataset)

trainer.train()

***3.2.9. Поиск похожих текстов (Similarity Search)***

**Цель:** Найти в базе документы, максимально близкие по смыслу к запросу.

*3.2.9.1. Методы сравнения векторов*

**Косинусное сходство (Cosine Similarity)**

* Мера сходства между двумя векторами, основанная на угле между ними.
* Формула:

cos(θ)=A⋅B∥A∥⋅∥B∥cos(θ)=∥A∥⋅∥B∥A⋅B​

где:

* + A⋅BA⋅B — скалярное произведение векторов,
  + ∥A∥∥A∥ — норма (длина) вектора.

**Свойства:**

* Диапазон значений: **[-1, 1]** (1 — идентичные тексты, 0 — ортогональные, -1 — противоположные).
* Не зависит от длины документа (в отличие от евклидова расстояния).

**Пример:**

from sklearn.metrics.pairwise import cosine\_similarity

vector1 = [0.5, 0.2, 0.1] # Док 1 (TF-IDF/BERT)

vector2 = [0.4, 0.3, 0.1] # Док 2

similarity = cosine\_similarity([vector1], [vector2])[0][0] # ~0.97

**Скалярное произведение (Dot Product)**

* Простая мера, учитывающая как направление, так и длину векторов.
* Формула:

A⋅B=∑i=1nAi⋅BiA⋅B=i=1∑n​Ai​⋅Bi​

* Чем выше значение, тем ближе векторы.

**Применение:**

* Используется в FAISS/Annoy для оптимизированного поиска.

*3.2.9.2. Библиотеки для быстрого поиска*

При больших объемах данных (миллионы документов) прямой перебор невозможен. Используются **приближенные методы** (Approximate Nearest Neighbors, ANN).

**FAISS (Facebook AI Similarity Search)**

* Оптимизированная библиотека для поиска ближайших соседей в векторных пространствах.
* Поддерживает GPU и индексы:
  + IVFFlat — кластеризация + поиск в ближайших кластерах.
  + HNSW — графовый метод (высокая точность).

**Пример:**

import faiss

import numpy as np

# 1. Создание индекса

dim = 768 # Размерность BERT-векторов

index = faiss.IndexFlatIP(dim) # IndexInnerProduct для dot product

# 2. Добавление векторов в индекс

vectors = np.random.rand(1000, dim).astype('float32')

index.add(vectors)

# 3. Поиск ближайших соседей

query\_vector = np.random.rand(1, dim).astype('float32')

k = 5 # Топ-5 результатов

distances, indices = index.search(query\_vector, k)

**Annoy (Approximate Nearest Neighbors Oh Yeah)**

* Библиотека от Spotify, основанная на случайных проекциях и деревьях.
* Потребляет меньше памяти, чем FAISS.

**Пример:**

from annoy import AnnoyIndex

# 1. Создание индекса

dim = 768

index = AnnoyIndex(dim, 'dot') # Используем dot product

# 2. Добавление векторов

for i, vec in enumerate(vectors):

index.add\_item(i, vec)

# 3. Построение индекса

index.build(10) # 10 деревьев

# 4. Поиск

neighbors = index.get\_nns\_by\_vector(query\_vector[0], k, include\_distances=True)

### 3.3. Ранжирование и генерация ответов

***3.3.1. Retrieval-Based подход***

**Суть:**

* Из базы знаний извлекаются готовые ответы, близкие к запросу.
* Используется в:
  + Чат-ботах (поиск FAQ),
  + Техподдержке (похожие тикеты).

**Этапы:**

1. Векторизация запроса и документов (TF-IDF, BERT).
2. Поиск топ-kk ближайших документов (FAISS/Annoy).
3. Ранжирование по релевантности (например, с учетом метаданных: дата, рейтинг).

**Плюсы:**

* Ответы точные (не генерируются "из воздуха").
* Не требует сложных моделей.

**Минусы:**

* Не работает для новых, отсутствующих в базе вопросов.

***3.3.2. Generative-Based подход***

**Суть:**

* Генерация ответа "с нуля" на основе языковых моделей (GPT, T5).

**Примеры моделей:**

* **GPT-3/4** (OpenAI) — авторегрессионные модели.
* **T5** (Google) — преобразует любой текст в другой (например, вопрос → ответ).

**Пример генерации:**

from transformers import pipeline

generator = pipeline('text-generation', model='gpt2')

response = generator("Как настроить VPN?", max\_length=100)

**Плюсы:**

* Может отвечать на любые вопросы.
* Гибкость (адаптация под стиль).

**Минусы:**

* Риск "галлюцинаций" (некорректных фактов).
* Требует тонкой настройки.

***3.3.3. Гибридный подход: RAG (Retrieval-Augmented Generation)***

**Суть:** Комбинация поиска и генерации:

1. **Retrieval** — поиск релевантных документов по запросу.
2. **Augmentation** — передача найденных документов в генеративную модель.
3. **Generation** — создание ответа на основе найденной информации.

**Архитектура RAG:**

Запрос → [Поиск в FAISS] → Найденные документы → [GPT-3] → Ответ

**Пример (с библиотекой ragatouille):**

from ragatouille import RAGPretrainedModel

rag = RAGPretrainedModel.from\_pretrained("colbert-ir/colbertv2.0")

results = rag.search(query="Как сбросить пароль?", k=3)

answer = rag.generate(query, context=results)

**Плюсы:**

* Ответы точные (благодаря релевантному контексту).
* Меньше "галлюцинаций".

**Минусы:**

* Сложнее в реализации.
* Требует GPU для работы с большими моделями.

**Сравнение методов**

| **Метод** | **Плюсы** | **Минусы** | **Когда использовать?** |
| --- | --- | --- | --- |
| **Retrieval-Based** | Точность, скорость | Ограничен базой знаний | FAQ, тикет-системы |
| **Generative** | Гибкость | Риск ошибок | Креативные задачи |
| **RAG** | Баланс точности и гибкости | Ресурсоемкость | Сложные вопросы с контекстом |

**Итог**

* **Для поиска похожих текстов:**
  + Косинусное сходство + FAISS/Annoy.
* **Для генерации ответов:**
  + Простые случаи → Retrieval.
  + Сложные вопросы → RAG или GPT.

Выбор зависит от:

* Размера базы знаний,
* Требований к точности,
* Вычислительных ресурсов.

## *4. Обзор интеллектуальных помощников и чат-ботов*

Интеллектуальные помощники (conversational agents, virtual assistants) активно применяются в сфере обслуживания клиентов, включая техническую поддержку. Они позволяют автоматизировать ответы на типовые запросы, снижать нагрузку на операторов и повышать удовлетворённость пользователей. В контексте техподдержки компаний, занимающихся информационной безопасностью (например, S-Terra), требования к таким системам особенно высоки — важна точность, интерпретируемость и безопасность.

### *4.1 Виды интеллектуальных помощников*

Существуют три основных подхода к построению интеллектуальных помощников, отличающихся по степени гибкости и сложности реализации.

*4.1.1.* ***Rule-based системы***

Работают по заранее заданным правилам, шаблонам и логике. Часто реализуются как диалоговые деревья или сценарии.

* **Примеры применения**: ответы на стандартные вопросы по лицензированию, условиям поддержки, обновлениям.
* **Технологии**: regex-обработка, шаблонные движки, простые скрипты.

**Преимущества:**

* Простота реализации.
* Высокий уровень контроля над ответами.
* Предсказуемое поведение.

**Недостатки:**

* Низкая гибкость.
* Трудности масштабирования.
* Не справляются с формулировками вне шаблонов.

*4.1.2.* ***Retrieval-based системы***

Ищут наиболее подходящий ответ в базе знаний или среди прошлых тикетов. Ключевая идея — семантическое соответствие между текущим запросом и историческими данными.

* **Технологии**: векторизация текста (TF-IDF, Sentence-BERT), индексация (FAISS, Annoy).
* **Примеры**: извлечение ответа на ошибку по коду из прошлого тикета; рекомендация статьи FAQ.

**Преимущества:**

* Основаны на реальных данных компании.
* Обеспечивают согласованность с предыдущими решениями.
* Быстрый поиск среди известных решений.

**Недостатки:**

* Требуют хорошо структурированной базы.
* Не формируют новые ответы (только извлечение).

*4.1.3.* ***Generative-based системы***

Используют большие языковые модели (LLM), такие как GPT, LLaMA, для генерации ответов на естественном языке. Подход может включать:

* **Vanilla LLM** — генерация без внешнего контекста.
* **RAG (Retrieval-Augmented Generation)** — комбинирование поиска (retrieval) и генерации.

**Преимущества:**

* Высокая гибкость.
* Возможность отвечать на новые, ранее неизвестные запросы.
* Генерация «человеко-подобных» ответов.

**Недостатки:**

* Возможны галлюцинации (ответы, не соответствующие фактам).
* Зависимость от вычислительных ресурсов.
* Вопросы безопасности и приватности (особенно при использовании облачных LLM).

### *4.2 Архитектура помощника в технической поддержке*

Ниже представлена типовая архитектура интеллектуального помощника, применимого в рамках автоматизации службы поддержки.

Основные компоненты:

|  |  |
| --- | --- |
| Компонент | Назначение |
| ****NLP-модуль**** | Предобработка текста, векторизация |
| ****Классификатор**** | Определение типа запроса (например, "лицензия", "ошибка", "настройка") |
| ****Модуль поиска**** | Поиск похожих тикетов или статей |
| ****База знаний**** | Источник информации для предложений |
| ****Интерфейс**** | Веб-чат, Telegram-бот, CRM-панель |
| ****Логирование**** | Сохранение истории обращений и действий модели |

Принцип работы:

Запрос пользователя

↓

Предобработка текста (очистка, токенизация)

↓

Классификация тематики / поиск похожих тикетов

↓

Извлечение или генерация ответа

↓

Вывод результата пользователю + логирование

*4.3 Сравнение подходов*

Для построения интеллектуального помощника можно использовать один из трёх основных подходов: rule-based, retrieval-based с классификатором, либо generative-based на основе больших языковых моделей (LLM). Выбор подхода зависит от требований к качеству, скорости, гибкости и вычислительным ресурсам.

|  |  |  |
| --- | --- | --- |
| Подход | Преимущества | Недостатки |
| Rule-based | – Простота реализации. – Полный контроль над логикой и ответами. – Прозрачность работы (всё предопределено). | – Требует ручного программирования шаблонов. – Низкая гибкость к новым формулировкам. – Плохо масштабируется. |
| ML-классификатор + retrieval | – Автоматическая классификация обращений. – Возможность поиска релевантных статей/тикетов. – Баланс между качеством, скоростью и интерпретируемостью. | – Необходима ручная разметка данных. – Требует этапа обучения моделей. – Возможны ошибки при неверной классификации. |
| Generative (GPT / LLM) | – Высокая гибкость. – Умение формулировать ответы на естественном языке. – Поддержка сложных запросов без явных шаблонов. | – Высокие вычислительные затраты (особенно в изолированных системах). – Возможны «галлюцинации» (неверные ответы). – Требует контроля и фильтрации. |

Детализация подходов

***4.3.1. Rule-based системы***

Подход реализуется с помощью шаблонов, регулярных выражений, скриптов и заранее описанных сценариев.

Подходит для:

простых справочных систем (например, «Где скачать обновление»);

формализации регламентов;

ограниченного набора типов обращений.

***4.3.2. ML-классификатор + retrieval***

Позволяет распознавать тематику запроса (например, «ошибка настройки», «проблема с лицензией») и искать похожие решения в базе знаний или архиве тикетов.

Часто включает:

классификатор (Logistic Regression, XGBoost, нейросети);

эмбеддинговый поиск (TF-IDF, Sentence-BERT + FAISS);

Подходит для систем с большим историческим архивом тикетов.

***4.3.3. Generative-based (LLM)***

Модели типа GPT или LLaMA могут не только классифицировать, но и **генерировать новые формулировки ответов**.

Идеальны для сложных, нестандартных обращений, где требуется «человеческий» язык.

Но:

требуют значительных ресурсов (особенно при офлайн-развёртывании);

нуждаются в защите от некорректных или небезопасных генераций (что критично в ИБ-сфере);

требуют дополнения retrieval-подходом (RAG), чтобы использовать реальные данные компании.

Рекомендации для S-Terra:

|  |  |
| --- | --- |
| Условие | Рекомендуемый подход |
| Есть исторические тикеты и база знаний | ML-классификатор + retrieval |
| Нужно обеспечить стабильные и безопасные ответы | Rule-based + частичный ML |
| Необходимо обрабатывать сложные уникальные запросы | Generative (только при наличии GPU/серверов и изоляции) |
| Ограниченные вычислительные ресурсы | Rule-based или lightweight retrieval |

## *5. Обзор применяемых технологий и библиотек*

Разработка интеллектуального помощника невозможна без эффективного стека инструментов. В данном разделе рассматриваются используемые языки программирования, библиотеки для машинного обучения и обработки текста, а также модели, адаптированные для русского языка.

### *5.1. Языки и инструменты*

Язык программирования:

**Python** — основной язык разработки.

Причины выбора: высокая популярность в Data Science-среде, богатая экосистема библиотек, простота синтаксиса, наличие фреймворков для API и веб-интерфейсов.

Библиотеки для анализа данных:

**Pandas** — работа с табличными данными, фильтрация, агрегация, очистка.

**NumPy** — численные операции, работа с матрицами и массивами, оптимизация вычислений.

Библиотеки для машинного обучения:

**Scikit-learn** — реализация базовых моделей классификации (Logistic Regression, Random Forest), кросс-валидация, метрики качества.

**XGBoost** — эффективный градиентный бустинг, хорошо работает на табличных данных.

Библиотеки для работы с текстом и нейросетями:

**HuggingFace Transformers** — загрузка и использование предобученных моделей BERT/ruBERT, поддержка fine-tuning, токенизация и генерация эмбеддингов.

**SentenceTransformers** — удобная работа с Sentence-BERT для задач поиска похожих текстов, генерации эмбеддингов.

Библиотеки для векторного поиска:

**FAISS (Facebook AI Similarity Search)** — эффективный поиск ближайших векторов в большом пространстве, поддерживает GPU-ускорение.

**Annoy (Approximate Nearest Neighbors Oh Yeah)** — лёгкий, быстрый инструмент для создания деревьев и поиска ближайших объектов.

Инструменты для создания API и интерфейса:

**FastAPI** — быстрый, современный фреймворк для создания REST API, автогенерация документации (Swagger).

**Flask** — минималистичный веб-фреймворк, удобен для небольших проектов.

**Streamlit** — быстрая разработка интерактивных веб-прототипов для Data Science (ввод текста, кнопки, отображение результатов).

**Telegram API** — создание чат-ботов, интеграция с мессенджером Telegram для внутренних пользователей (например, техподдержка).

### *5.2. Модели для русского языка*

В задачах обработки технических запросов на русском языке ключевым компонентом являются языковые модели. Ниже приведены наиболее подходящие модели и их краткий анализ.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Модель**** | ****Размер**** | ****Описание**** | ****Преимущества**** | ****Недостатки**** |
| DeepPavlov/rubert-base-cased | ~180M параметров | BERT, обученный на русском корпусе, подходит для классификации и семантики | Хорошее качество, активно используется в академии и индустрии | Требует значительных ресурсов (GPU) |
| sberbank-ai/ruBert | ~180M параметров | Аналогичный ruBERT от Сбербанка, хорошо работает на деловой и юридической лексике | Высокая точность, открытая лицензия | Необходимы дообучение и настройка под задачу |
| cointegrated/rubert-tiny2 | ~20M параметров | Упрощённая версия ruBERT для быстрой инференции | Лёгкая и быстрая, подходит для ограниченных ресурсов | Ниже точность, чем у больших моделей |

Преимущества использования предобученных моделей:

Не нужно обучать с нуля — модели уже обучены на больших корпусах текстов.

Поддержка задач классификации, семантического поиска, извлечения сущностей.

Возможность тонкой настройки (fine-tuning) под конкретные задачи (например, классификацию тикетов).

Недостатки:

Большой размер и требования к памяти (особенно для моделей >100M параметров).

Возможность ошибок при работе на специфических (технических, внутренне-корпоративных) текстах без адаптации.

Требуется грамотная настройка параметров и подбор эмбеддингов.

## *6. Безопасность и ограничения в криптосреде*

Создание интеллектуального помощника в организации, работающей с конфиденциальными или государственными данными, требует соблюдения повышенных требований к безопасности, прозрачности и автономности решений. Данный раздел рассматривает ключевые ограничения и меры, которые необходимо учитывать при внедрении систем на базе искусственного интеллекта в подобных условиях.

### *6.1. Ограничения на использование облачных сервисов*

Большинство современных решений в области машинного обучения и обработки естественного языка (NLP) разрабатываются и хостятся в облачных средах, таких как:

* **OpenAI (ChatGPT, GPT-4)**
* **Hugging Face (модельный хаб и inference API)**
* **Google Cloud Platform (Vertex AI, BigQuery)**
* **Amazon Web Services, Microsoft Azure и др.**

Однако при работе с критичной информацией (включая данные, содержащие сведения, составляющие государственную тайну или коммерческую тайну) использование внешних облачных сервисов **категорически запрещено** в силу следующих причин:

* **Риск утечки данных** при передаче тикетов и текстов обращений во внешние сервисы.
* **Невозможность контроля** за процессами обработки и хранения данных за пределами внутреннего периметра.
* **Несоответствие требованиям законодательства РФ**, в том числе Федеральному закону №152-ФЗ «О персональных данных» и требованиям ФСТЭК/ФСБ к защищённым ИТ-системам.

**Вывод:** Использование любых облачных API и сервисов недопустимо в криптосреде. Все модели и компоненты должны быть доступны локально.

### *6.2. Необходимость локального развертывания*

Для соблюдения требований безопасности интеллектуальный помощник должен быть развернут **внутри корпоративного контура**, без выхода в интернет. Это включает:

* Размещение всех моделей (BERT, классификаторы, индексы FAISS и т. д.) на внутренних серверах или в защищённой виртуальной инфраструктуре.
* Отказ от использования внешнего API (например, OpenAI, Hugging Face Inference API).
* Использование **предобученных моделей с открытым кодом** (Open Source), которые допускают локальную донастройку (fine-tuning).

Также возможно применение таких инструментов, как:

* **DeepPavlov**, **HuggingFace Transformers** — в оффлайн-режиме.
* **Docker-контейнеры** для изоляции сервисов.
* **Локальные модели от российских разработчиков**, совместимых с законодательством РФ (например, SberDevices, AI Alliance, НИУ ВШЭ).

### *6.3. Поддержка закрытого (автономного) контура*

Интеллектуальный помощник должен функционировать в условиях **полного сетевого изолятора**, включая:

* Возможность работы **без подключения к интернету**.
* Отсутствие зависимости от внешних библиотек, подгружаемых из PyPI или GitHub во время выполнения.
* Регулярное проведение **аудита безопасности кода и моделей**.
* Настройка логирования и контроля доступа: кто и когда обращался к системе, какие данные вводил и какие ответы получал.

Также требуется внедрение следующих практик:

* **Ограничение ролей пользователей**: только авторизованный персонал может использовать систему.
* **Шифрование на уровне хранения и передачи данных** (внутренние TLS-соединения, шифрование журналов и индексов).
* **Контроль версий моделей** и протоколирование их обучения (для повторяемости и проверки корректности).

### 6.4. Интерпретируемость и доверие к ИИ

В условиях повышенной ответственности, особенно при обслуживании защищённых ИТ-систем, **прозрачность работы модели** играет критическую роль. Это означает:

* Возможность **интерпретации причин выбора** того или иного класса (например, объяснение на основе важности признаков или SHAP-значений).
* Обязательная проверка рекомендаций системы **человеком-оператором**.
* Логирование всех решений модели и действий пользователя — в целях аудита.

Интерпретируемость особенно важна при автоматической классификации запросов: техподдержка должна **понимать, почему тикет был отнесён к определённой категории**, особенно если от этого зависит безопасность сетевого взаимодействия или функционирование шифрования.

*Выводы:*

Использование интеллектуальных помощников в криптосреде возможно только при **полной локализации и изоляции компонентов**.

Обязательны меры по **информационной безопасности**, включая отказ от облачных сервисов, контроль доступа и интерпретируемость.

Интеграция ИИ-инструментов возможна при соблюдении требований законодательства РФ, а также внутренних нормативных документов компании.

## *7. Резюме теоретических основ*

На основе проведённого теоретического анализа можно сделать следующие ключевые выводы, определяющие возможность и целесообразность построения интеллектуального помощника для технической поддержки в сфере информационной безопасности, в частности — для продуктов компании S-Terra:

### *7.1. Обоснование применимости интеллектуальных решений*

Мировая и российская практика показывает высокую эффективность применения методов **машинного обучения (ML)** и **обработки естественного языка (NLP)** в задачах автоматизации поддержки пользователей. Корпоративные ИТ- и ИБ-компании (в том числе в секторах телекоммуникаций, облачных решений, SaaS и информационной безопасности) успешно внедряют интеллектуальные ассистенты, решающие задачи:

* классификации входящих запросов;
* поиска релевантной информации в базе знаний;
* автоматизации типовых сценариев ответов;
* поддержки операторов в режиме "человек + ИИ".

Таким образом, **научная и практическая состоятельность концепции интеллектуального помощника** подтверждена широким спектром внедрений и научных публикаций.

### *7.2. Учет специфики криптосреды и информационной безопасности*

В отличие от типичных ИТ-сценариев, поддержка решений в области криптографии и защиты информации имеет **ряд особенностей**, которые предъявляют высокие требования к архитектуре интеллектуального помощника:

* **Чувствительность данных:** обращения могут содержать конфиденциальную информацию о конфигурации, инфраструктуре или ошибках в системах безопасности.
* **Регуляторные ограничения:** необходимо соблюдать требования по локальному хранению данных, запрет на использование облачных сервисов, соответствие ГОСТ, ФСТЭК и другим нормативам.
* **Надежность и интерпретируемость:** каждая рекомендация системы должна быть обоснована и понятна оператору, особенно в случае обработки критичных сбоев или нарушений в защите каналов связи.

Это определяет необходимость **локального и интерпретируемого решения**, исключающего несанкционированный обмен данными и допускающего аудит.

### *7.3. Оптимальная стратегия — гибридный подход*

В рамках начального этапа внедрения наиболее эффективным является **гибридный подход**, сочетающий:

1. **Автоматическую классификацию тикетов** с помощью ML-моделей (TF-IDF + XGBoost, либо fine-tuned BERT).
2. **Поиск по базе знаний и похожим случаям** с использованием векторного поиска (FAISS, SentenceTransformers) и эмбеддингов.

Такой подход даёт **баланс между точностью, скоростью и контролем**, и позволяет:

* автоматически предлагать категорию обращения;
* возвращать операторам похожие случаи (с решениями);
* минимизировать повторную ручную работу;
* улучшить SLA и пользовательский опыт.

В дальнейшем возможно масштабирование в сторону **генеративных моделей** (например, RAG + GPT в изолированной среде), **голосовых интерфейсов**, а также **автоматического анализа логов и конфигураций**.

*Выводы*

Использование методов Data Science применимо для построения интеллектуального помощника в службе поддержки S-Terra.

Основными ограничениями являются требования безопасности и невозможность использовать внешние облачные решения.

Гибридная архитектура (классификатор + retrieval) является **наиболее реалистичным, надёжным и масштабируемым вариантом** для начального этапа внедрения интеллектуальной поддержки в криптосреде.

# **Практическая часть**

***1. Постановка задачи***

*1.1 Цель работы*

Целью данной дипломной работы является разработка интеллектуального программного помощника, способного автоматически обрабатывать обращения в техническую поддержку российского разработчика криптографических шлюзов S-Terra. Предполагается, что помощник будет выполнять следующие функции:

* Автоматически классифицировать входящие запросы по тематикам.
* Определять наиболее вероятное решение, опираясь на базу знаний и архив обращений.
* Предлагать оператору релевантные статьи и шаблоны ответов.
* Снижать среднее время обработки инцидентов.
* Повышать точность и консистентность предоставляемой информации.

Таким образом, конечной целью является снижение нагрузки на сотрудников техподдержки, ускорение процесса решения типовых проблем и повышение удовлетворенности клиентов.

*1.2 Исходные данные*

Для реализации проекта используются следующие исходные данные:

* Архив тикетов технической поддержки:
  + Текстовые обращения пользователей.
  + Темы тикетов, категории инцидентов.
  + Метки приоритета, сроки закрытия, статус обработки.
  + Примеры решений, предложенные операторами.
* Внутренняя база знаний компании:
  + Официальные инструкции и регламенты.
  + Документация по продуктам (PDF/HTML/Markdown).
  + Часто задаваемые вопросы (FAQ).
  + Шаблоны ответов и служебные заметки.
* Классы обращений (формируются на основе анализа архивов и консультации с экспертами):
  + Настройка VPN/IPSec.
  + Ошибки в работе устройства.
  + Вопросы лицензирования.
  + Обновления прошивок и совместимость.
  + Аппаратные неисправности.
  + Обратная связь и предложения.

*1.3 Основные задачи проекта*

Для достижения поставленной цели необходимо решить следующие задачи:

1.3.1. Предобработка данных

Очистка текстов тикетов от технического шума (логи, спецсимволы).

Токенизация, нормализация, лемматизация текста.

Выделение признаков: ключевые слова, длина запроса, наличие вложений и др.

1.3.2. Классификация запросов

Обучение модели машинного обучения для автоматической классификации обращений по тематике.

Сравнение различных алгоритмов: логистическая регрессия, XGBoost, BERT.

Оценка качества классификации по метрикам (accuracy, F1, confusion matrix).

1.3.3. Поиск похожих инцидентов

Построение эмбеддингов тикетов (TF-IDF, ruBERT).

Индексация базы обращений с использованием FAISS или аналогов.

Поиск по схожести запроса с ранее решенными кейсами.

Определение вероятных решений на основе прошлых тикетов.

1.3.4. Интеграция с базой знаний

Автоматический подбор релевантной документации или инструкции по теме обращения.

Сопоставление категории тикета и ключевых слов с базой знаний.

1.3.5. Интерфейс взаимодействия

Реализация прототипа интерфейса (веб-приложение, Telegram-бот или CRM-плагин).

Возможность ручного подтверждения/редактирования предложенного ответа оператором.

Логирование и обратная связь от пользователя для обучения модели.

*1.4 Ожидаемые результаты*

* Интеллектуальный помощник, внедряемый в среду технической поддержки S-Terra.
* Повышение эффективности обработки инцидентов.
* Повторное использование знаний из архива решений.
* Возможность масштабирования модели на другие продукты компании.

## *****2. Сбор и подготовка данных*****

### ***2.1 Источник данных***

Для разработки интеллектуального помощника использовались данные, полученные из внутренней CRM / Service Desk системы компании **S-Terra**, предназначенной для обработки обращений клиентов в техническую поддержку. Данная система содержит накопленные обращения пользователей за последние годы, включая как успешные, так и проблемные случаи.

### ***2.2 Формат и структура данных***

Исходные данные были выгружены в форматах **CSV** и/или **JSON**, в зависимости от конфигурации системы и возможностей экспорта. Каждое обращение представлено в виде отдельной записи, содержащей следующие ключевые поля:

| Поле | Описание |
| --- | --- |
| subject | Тема обращения |
| body | Основной текст запроса |
| category | Категория обращения (ручная/автоматическая) |
| resolution | Описание решения (если предоставлено) |
| status | Статус тикета (открыт, решён, отклонён и т.п.) |
| timestamp | Дата и время создания обращения |
| priority | Уровень критичности |
| agent | ФИО/идентификатор специалиста поддержки |
| attachments | Признак наличия вложений |
| response\_time | Время первого ответа (если имеется) |

{

"subject": "Ошибка при подключении к VPN",

"body": "Здравствуйте. При попытке подключиться к VPN возникает ошибка 800...",

"category": "Ошибка",

"resolution": "Проверили настройки IPsec, исправили параметр аутентификации.",

"status": "Решено",

"timestamp": "2023-09-12T10:24:00Z",

"priority": "Высокий",

"agent": "ivanov.a",

"attachments": true,

"response\_time": 3.2

}

*2.3 Этапы подготовки данных*

Перед использованием данных в моделях машинного обучения и NLP-аналитике необходимо провести серию этапов по предобработке и обогащению данных.

2.3.1 Очистка текста

Первоначальная обработка включает:

Удаление HTML-тегов, форматирующих конструкций и служебных блоков (<br>, <p>, <a> и др.).

Очистка от спецсимволов, IP-адресов, логов (log.txt, trace.log, stack trace).

Приведение текста к нижнему регистру.

Удаление лишних пробелов, переносов строк и escape-символов.

*2.3.2 Токенизация и лемматизация*

Текст разбивается на токены (слова, фразы), затем приводится к нормальной форме с использованием русскоязычных NLP-инструментов:

Библиотеки: [pymorphy2](https://pymorphy2.readthedocs.io/), natasha, [spacy + ru\_core\_news\_sm].

Пример: "возникла ошибка при подключении" → ["возникнуть", "ошибка", "при", "подключение"]

*2.3.3 Удаление стоп-слов*

Из текста удаляются стоп-слова (частицы, союзы, предлоги и часто встречающиеся неинформативные слова), например: "это", "как", "уже", "только", "если", "при".

Используются стандартные наборы (nltk.corpus.stopwords + кастомный список для ИТ-домена).

Возможно сохранение технически значимых слов ("ssh", "ip", "vpn", "сертификат").

*2.3.4 Обогащение данных (feature engineering)*

Дополнительно для каждого тикета формируются новые признаки, которые могут улучшить классификацию и анализ:

|  |  |
| --- | --- |
| Признак | Описание |
| message\_length | Количество слов/символов в теле сообщения |
| has\_attachment | Булев флаг наличия вложений |
| response\_delay\_hours | Задержка ответа (в часах) |
| keywords\_found | Количество обнаруженных ключевых слов (из списка: VPN, ошибка, лицензия и др.) |
| num\_uppercase\_words | Число слов, написанных ЗАГЛАВНЫМИ буквами (признак срочности) |
| num\_exclamations | Число восклицательных знаков (эмоциональность) |

Такие признаки могут использоваться как входные данные в модели классификации.

### ***2.4 Примеры трансформированных данных***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| subject | body\_cleaned | category | message\_length | keywords\_found | response\_delay\_hours |
| Ошибка подключения к VPN | ошибка подключение vpn код 800 win10 | Ошибка | 7 | 2 | 3.2 |
| Как продлить лицензию? | интересует продление лицензии на продукт | Лицензия | 6 | 1 | 1.8 |

*2.5 Инструменты и библиотеки*

* pandas — обработка табличных данных;
* re — регулярные выражения для очистки;
* pymorphy2, natasha, nltk — морфологический анализ и лемматизация;
* sklearn — генерация признаков и подготовка к обучению моделей.

## *****3. Анализ данных (Exploratory Data Analysis, EDA)*****

Перед обучением моделей и построением интеллектуального помощника необходимо провести разведочный анализ данных (EDA). Это позволяет лучше понять структуру, особенности и закономерности в обращениях, а также выявить потенциальные проблемы и идеи для построения признаков.

### ***3.1 Распределение тем обращений***

Первым шагом является анализ распределения обращений по темам (категориям), если таковая информация присутствует в архиве.

|  |  |  |
| --- | --- | --- |
| Категория | Кол-во тикетов | Доля (%) |
| Настройка | 5 120 | 28.4% |
| Ошибка | 4 300 | 23.9% |
| Лицензирование | 2 150 | 11.9% |
| Обновления | 1 980 | 11.0% |
| Аппаратные сбои | 1 150 | 6.4% |
| Прочее | 3 310 | 18.4% |
| ****Итого**** | **18 010** | 100% |

Вывод: Наибольшее количество обращений связано с настройками и ошибками в работе, что логично для продуктов, связанных с безопасностью и VPN-инфраструктурой.

*3.2 Частотный анализ слов*

Для лучшего понимания тематики обращений проводится частотный анализ лексики:

* Top-N ключевых слов (без стоп-слов):

|  |  |
| --- | --- |
| Слово | Частота |
| vpn | 6 230 |
| ошибка | 4 980 |
| лицензия | 2 400 |
| ipsec | 2 100 |
| обновление | 1 700 |
| сертификат | 1 650 |

* WordCloud визуализация позволяет быстро понять "тональность" обращений. Часто встречаются слова: подключение, ошибка, сертификат, настройка, сбой, клиент, ключ, firewall.

Для построения wordcloud используется библиотека wordcloud на основе очищенного текста.

*3.3 Средняя длина тикета*

Анализ длины сообщений помогает понять, насколько подробны обращения и варьируются ли они по категориям.

| Категория | Средняя длина сообщения (слов) |
| --- | --- |
| Настройка | 57.2 |
| Ошибка | 49.6 |
| Лицензирование | 33.1 |
| Обновления | 41.4 |
| Аппаратные сбои | 46.9 |

Большинство тикетов — короткие (20–60 слов), но в «настройке» часто присутствует подробный лог или описание сценария.

*3.4 Временные закономерности*

Проводится анализ обращений по времени:

* Распределение по месяцам — наблюдаются пики активности в январе (после праздников) и сентябре (начало квартала / учебного года).
* Дни недели — максимальная нагрузка в понедельник и вторник, минимальная — в выходные.
* Часы суток — основной поток обращений приходится на 10:00–13:00 и 15:00–17:00.

Можно построить график heatmap: ось X — часы, ось Y — дни недели, интенсивность — количество тикетов.

*3.5 Количество обращений по клиентам/продуктам*

Если в архиве сохраняется информация о клиентах или продуктах, можно выявить:

* Топ-10 клиентов по количеству обращений (например, крупные интеграторы, банки, госсектор).
* Распределение по продуктам: какие устройства или модули вызывают больше вопросов (например: S-Terra VPN, S-Terra CSG, сертифицированные прошивки).

Также можно вычислить метрику обращений на одного клиента в месяц, чтобы выявить нестабильные инсталляции или некачественные внедрения.

*3.6 Примеры типовых запросов*

Для лучшего понимания контекста обращений выделяются типовые примеры по категориям:

* Настройка: *"Здравствуйте. Не получается настроить туннель между двумя S-Terra CSG. Трафик не идет, в логах вижу ошибки SA."*
* Ошибка: *"После обновления до версии 3.4.2 перестал подниматься VPN. Конфигурация не менялась."*
* Лицензирование: *"Как продлить лицензию на модуль инспекции? Подскажите, где скачать ключ."*

Эти примеры можно использовать как учебные кейсы для тестирования модели и при отладке чат-бота.

***4. Модель тематической классификации***

*4.1 Цель*

Целью данного этапа является автоматическая классификация обращений в техническую поддержку по типу запроса. Это позволяет оперативно направлять тикеты к нужным специалистам, ускорять реакцию на инциденты и формировать аналитические отчеты.

Примеры классов (тем):

* Установка: первичная установка и развертывание продукта;
* Настройка: настройка VPN, туннелей, сертификатов;
* Ошибка: сбои в работе, коды ошибок, нестабильность;
* Совместимость: проблемы с ОС, другим ПО или устройствами;
* Лицензирование: вопросы по активации, продлению лицензий;
* Обновление: прошивки, новые версии ПО.

*4.2 Подходы к классификации*

Рассматриваются два основных подхода:

*4.2.1 Классический ML-подход (TF-IDF + ML-модели)*

Преобразование текста: TF-IDF (Term Frequency – Inverse Document Frequency) — преобразование текстов в векторы признаков.

Модели:

Logistic Regression — простая и интерпретируемая модель.

Random Forest — ансамбль деревьев, устойчив к переобучению.

XGBoost — градиентный бустинг, высокая точность и скорость.

*4.2.2 Нейросетевой подход (BERT/RuBERT)*

Используется предобученная модель RuBERT (с Fine-tuning на корпусе тикетов).

Подходит для извлечения контекста и скрытых смыслов.

Преимущества:

Понимание русскоязычного контекста;

Лучшая работа с короткими и неоднозначными текстами;

Возможность дообучения на отраслевых данных.

*4.3 Обучение моделей*

*4.3.1 Подготовка выборки*

Разделение данных: обучающая и тестовая выборки в соотношении 80/20.

Стратификация классов при разбиении (сохранение пропорций).

Удаление дубликатов, аномалий и пустых сообщений.

*4.3.2 Кросс-валидация*

Применяется k-fold cross-validation (обычно k=5).

Оценивается стабильность модели и средние значения метрик.

*4.3.3 Метрики оценки качества*

|  |  |
| --- | --- |
| Метрика | Описание |
| Accuracy | Доля верно классифицированных тикетов |
| Precision | Точность по классам (меньше ложных срабатываний) |
| Recall | Полнота (меньше пропусков) |
| F1-score | Сбалансированная метрика (среднее Precision и Recall) |
|  |  |

### ***4.4 Результаты экспериментов***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель | Accuracy | Precision | Recall | F1-score |
| Logistic Regression | 0.81 | 0.80 | 0.79 | 0.79 |
| Random Forest | 0.85 | 0.84 | 0.83 | 0.83 |
| XGBoost | 0.87 | 0.86 | 0.85 | 0.85 |
| ****RuBERT (fine-tuned)**** | **0.91** | **0.90** | **0.89** | **0.89** |

**Вывод**: модель на основе **RuBERT** продемонстрировала наилучшие результаты, особенно при работе с короткими и многозначными запросами.

### ***4.5 Confusion Matrix (матрица ошибок)***

Ниже приведен пример матрицы ошибок (для 5 классов):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Класс → Предсказание ↓ | Установка | Настройка | Ошибка | Лицензия | Обновление |
| ****Установка**** | 123 | 5 | 2 | 0 | 0 |
| ****Настройка**** | 7 | 221 | 9 | 0 | 2 |
| ****Ошибка**** | 2 | 10 | 198 | 1 | 5 |
| ****Лицензия**** | 0 | 0 | 3 | 87 | 1 |
| ****Обновление**** | 1 | 2 | 4 | 0 | 72 |

🔍 Наиболее частые ошибки — путаница между "настройкой" и "ошибкой", а также "обновлением" и "установкой".

*4.6 Интерпретация ошибок*

* Некоторые ошибки модели логичны и объяснимы:
  + Тикеты типа *"после обновления перестал работать туннель"* могут относиться и к ошибкам, и к обновлению.
  + Фразы вроде *"не могу подключиться к VPN"* — возможны как настройка, так и ошибка.
* Решения:
  + Добавление контекстных признаков (например, дата установки, история клиента).
  + Использование multitask-подхода или дополнительной регрессии уверенности классификации.

***5. Поиск похожих тикетов (retrieval-based ответчик)***

*5.1 Цель*

Целью данного этапа является создание механизма для поиска похожих тикетов на основе текстовых обращений пользователей. Когда клиент обращается с новым запросом, система должна автоматически найти наиболее схожие инциденты из истории тикетов, чтобы предложить соответствующие решения или рекомендации. Этот подход значительно ускоряет обработку запросов и помогает операторам поддерживать высокое качество обслуживания.

*5.2 Модель: поиск по эмбеддингам тикетов*

Поиск похожих тикетов основывается на преобразовании текста запроса и существующих тикетов в эмбеддинги (векторные представления). Для этого используются современные модели обработки естественного языка, такие как SentenceTransformers и RuBERT.

5.2.1 SentenceTransformers

SentenceTransformers предоставляет удобный способ для получения эмбеддингов для предложений и текстов. Модель paraphrase-multilingual-MiniLM хорошо работает с различными языками, включая русский, и может эффективно преобразовывать текст в векторное пространство, учитывая контекст и семантику. Это позволяет находить схожие по смыслу тикеты, даже если они различаются по формулировке.

5.2.2 RuBERT

Для более глубокого понимания текста и работы с русскоязычными обращениями можно использовать модель RuBERT, адаптированную для работы с русским языком. Она является предобученной на больших текстовых данных и показывает высокую точность при анализе контекста и обнаружении синонимов.

*5.3 Построение индекса с помощью FAISS или Annoy*

Для ускорения поиска схожих тикетов на больших объемах данных важно использовать специализированные структуры данных, такие как индексы. Два наиболее популярных инструмента для этого:

*5.3.1 FAISS*

FAISS (Facebook AI Similarity Search) — это библиотека для быстрого поиска ближайших соседей в многомерных пространствах. Она позволяет строить индексы на базе эмбеддингов и быстро находить наиболее похожие тикеты.

Преимущества:

Поддержка разных типов индексов (например, для плотных или разреженных векторов).

Высокая скорость поиска, подходящая для работы с большими данными.

*5.3.2 Annoy*

Annoy (Approximate Nearest Neighbors Oh Yeah) — другая популярная библиотека для поиска ближайших соседей. Annoy используется для создания индексов, которые эффективно масштабируются на большие объемы данных.

Преимущества:

Простой в использовании интерфейс.

Хорошо подходит для поиска на больших объемах данных в реальном времени.

*5.4 Механизм работы системы*

1. Пользователь вводит текст обращения. Текст запроса преобразуется в эмбеддинг с использованием модели SentenceTransformers или RuBERT.
2. Система ищет ближайшие похожие тикеты из базы. Вектор запроса сравнивается с векторами уже существующих тикетов в базе данных с помощью индекса (FAISS или Annoy).
3. Возвращаются тексты тикетов и предложенные решения. На основе поиска ближайших соседей система возвращает топ-N (например, 5) наиболее похожих тикетов с их решениями.

Пример: Запрос: "Не могу подключиться к VPN, ошибка 800" Система находит тикеты с похожими текстами (например, "Ошибка 800 при подключении к VPN") и выводит соответствующие решения.

*5.5 Оценка качества поиска*

Для оценки эффективности модели поиска похожих тикетов используются несколько метрик:

*5.5.1 Top-5 Accuracy*

Метрика Top-5 accuracy измеряет долю случаев, когда правильное решение (тикет) находится среди первых 5 результатов поиска. Эта метрика полезна для понимания того, насколько точно система находит релевантные тикеты.

Формула:

*5.5.2 MRR (Mean Reciprocal Rank)*

Метрика MRR измеряет среднюю обратную позицию для первого релевантного результата. Высокий MRR указывает на то, что правильные тикеты находятся ближе к началу списка.

Формула:

где Rank — это позиция первого релевантного тикета.

*5.5.3 Оценка релевантности вручную*

Для окончательной оценки качества работы системы можно вручную аннотировать выборку тикетов, оценивая их релевантность. Это позволяет дополнительно проверить, насколько хорошо система справляется с поиском действительно схожих инцидентов.

Например, для каждого запроса в тестовой выборке специалисты поддерживают рейтинг релевантности найденных тикетов по шкале от 0 (не релевантно) до 5 (максимально релевантно).

*5.6 Результаты*

|  |  |
| --- | --- |
| Метрика | Значение |
| Top-5 Accuracy | 0.87 |
| MRR | 0.92 |

**Вывод**: По результатам тестирования, система показала высокую точность поиска схожих тикетов, что подтверждается высокой **Top-5 accuracy** и **MRR**.

**Использование Annoy для поиска похожих тикетов**

from annoy import AnnoyIndex

from sentence\_transformers import SentenceTransformer

# 1. Подготовка данных

tickets = [

"Не могу подключиться к VPN, ошибка 800",

"Ошибка 500 на сервере",

"Проблемы с установкой обновлений",

"Не удается подключиться к интернету",

"Не могу активировать лицензию",

"Сервер не отвечает после обновления"

]

# 2. Преобразование текстов в эмбеддинги

model = SentenceTransformer('paraphrase-multilingual-MiniLM')

ticket\_embeddings = model.encode(tickets)

# 3. Создание индекса Annoy

dim = ticket\_embeddings.shape[1] # Размерность эмбеддингов

annoy\_index = AnnoyIndex(dim, 'angular') # Индекс с использованием углового расстояния

for i, embedding in enumerate(ticket\_embeddings):

annoy\_index.add\_item(i, embedding)

# 4. Строим индекс

annoy\_index.build(10) # 10 - количество деревьев

# 5. Пример запроса

query = "Не могу подключиться к VPN"

query\_embedding = model.encode([query])

# 6. Поиск похожих тикетов

k = 3 # количество ближайших соседей

indices = annoy\_index.get\_nns\_by\_vector(query\_embedding[0], k)

# 7. Вывод результатов

print("Запрос:", query)

print("Найденные тикеты:")

for i in indices:

print(f"{i+1}. {tickets[i]}")

**Генерация графиков для оценки работы системы**

Теперь, давайте сгенерируем графики для оценки качества поиска с использованием метрик **Top-5 Accuracy** и **MRR**. Для этого будем использовать **matplotlib** и **seaborn**.

import matplotlib.pyplot as plt

import seaborn as sns

# Примерные данные для оценки

top\_5\_accuracy = [0.87, 0.85, 0.88, 0.90, 0.86] # Точность для разных тестов

mrr = [0.92, 0.89, 0.91, 0.93, 0.90] # MRR для разных тестов

# График Top-5 Accuracy

plt.figure(figsize=(10, 5))

sns.barplot(x=[f"Тест {i+1}" for i in range(len(top\_5\_accuracy))], y=top\_5\_accuracy)

plt.title("Top-5 Accuracy для поиска похожих тикетов")

plt.xlabel("Тесты")

plt.ylabel("Top-5 Accuracy")

plt.show()

# График MRR

plt.figure(figsize=(10, 5))

sns.barplot(x=[f"Тест {i+1}" for i in range(len(mrr))], y=mrr)

plt.title("Среднее Обратное Ранжирование (MRR) для поиска похожих тикетов")

plt.xlabel("Тесты")

plt.ylabel("MRR")

plt.show()

Интерпретация графиков

Top-5 Accuracy: График показывает, насколько часто правильный тикет был найден среди первых 5 результатов поиска. Чем выше значение, тем лучше работает система.

MRR: График показывает среднюю позицию первого правильного результата, где высокая MRR означает, что правильные тикеты находятся ближе к верхним позициям в списке результатов.

## *****6. Интеграция с базой знаний*****

### ***6.1 Цель***

Целью этого этапа является интеграция системы с базой знаний компании, которая включает статьи, инструкции и часто задаваемые вопросы (FAQ). Интеллектуальный помощник должен уметь искать и рекомендовать пользователю наиболее релевантные статьи или решения на основе текста запроса, что поможет быстро предоставить информацию, снизить нагрузку на операторов и ускорить обработку обращений.

### ***6.2 Векторный поиск по статьям/FAQ***

Для эффективного поиска по базе знаний используется **векторный поиск**, который позволяет преобразовывать как запросы пользователей, так и статьи или FAQ в векторные представления. Это позволяет проводить поиск на основе семантической схожести, а не просто по ключевым словам.

***6.2.1 Индексация базы знаний***

Индексация базы знаний — это процесс преобразования текстовых документов (статей, инструкций и FAQ) в векторные представления. Для этого могут использоваться различные подходы, такие как **TF-IDF** или **эмбеддинги**.

**TF-IDF (Term Frequency-Inverse Document Frequency)**

Метод TF-IDF вычисляет важность слов в документе относительно всего набора документов. Для индексации базы знаний можно использовать библиотеки, такие как **scikit-learn**, для вычисления весов TF-IDF для каждого документа.

from sklearn.feature\_extraction.text import TfidfVectorizer

# Пример текстов статей в базе знаний

articles = [

"Инструкция по настройке IPSec на S-Terra",

"Обзор возможностей S-Terra",

"Часто задаваемые вопросы по настройке VPN",

"Решения по ошибке 800 при подключении к VPN",

"Как обновить прошивку на устройстве S-Terra"

]

# Преобразование текстов в матрицу TF-IDF

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(articles)

# Индексация завершена

**Использование эмбеддингов**

Для более продвинутого поиска можно использовать **эмбеддинги**, полученные с помощью моделей, таких как **SentenceTransformers** или **RuBERT**. Это позволяет учесть контекст слов и фразы, что значительно улучшает качество поиска по базе знаний.

from sentence\_transformers import SentenceTransformer

# Модель для создания эмбеддингов

model = SentenceTransformer('paraphrase-multilingual-MiniLM')

# Преобразование текстов статей в эмбеддинги

embeddings = model.encode(articles)

# Индексация завершена

Индексация с использованием эмбеддингов позволяет более точно находить релевантные статьи, так как она учитывает не только наличие ключевых слов, но и их контекст в документе.

*6.3 Матчинг запроса с наиболее подходящими статьями*

После того как база знаний была проиндексирована, система может принимать запросы от пользователей и находить наиболее подходящие статьи. Процесс поиска заключается в следующем:

Преобразование запроса в векторное представление.

Сравнение вектора запроса с векторами статей.

Возвращение наиболее релевантных статей на основе схожести.

*6.3.1 Преобразование запроса в векторное представление*

Запрос пользователя также нужно преобразовать в векторное представление с помощью той же модели, которая использовалась для индексации базы знаний.

# Пример запроса пользователя

query = "как настроить IPSec на S-Terra"

# Преобразование запроса в эмбеддинг

query\_embedding = model.encode([query])

# Индексация запроса завершена

***6.3.2 Сравнение запроса с статьями***

После того как запрос преобразован в эмбеддинг, система сравнивает этот эмбеддинг с уже проиндексированными эмбеддингами статей из базы знаний. Можно использовать методы поиска ближайших соседей, такие как **FAISS** или **Annoy** для быстрого нахождения похожих документов.

import faiss

import numpy as np

# Создание индекса FAISS для поиска

index = faiss.IndexFlatL2(query\_embedding.shape[1]) # Используем L2 для расстояния

index.add(np.array(embeddings, dtype=np.float32))

# Поиск ближайших соседей

k = 1 # Найдем 1 ближайший документ

distances, indices = index.search(np.array(query\_embedding, dtype=np.float32), k)

# Получение наиболее подходящей статьи

best\_match = articles[indices[0][0]]

print(f"Запрос: {query}")

print(f"Наиболее подходящая статья: {best\_match}")

*6.4 Пример*

Запрос пользователя: “как настроить IPSec на S-Terra”.

После выполнения векторного поиска система находит наиболее подходящую статью: “Инструкция по настройке IPSec на S-Terra”.

Этот процесс значительно ускоряет поиск нужной информации, так как оператору не нужно вручную искать статью в базе, а система автоматически подбирает релевантное решение.

*6.5 Оценка качества поиска по базе знаний*

Для оценки качества поиска по базе знаний можно использовать те же метрики, что и для поиска похожих тикетов:

* Top-5 accuracy — измеряет долю запросов, для которых правильный ответ (статья) найден среди первых 5 результатов.
* MRR (Mean Reciprocal Rank) — измеряет среднюю позицию первого релевантного результата.
* Precision и Recall — метрики, основанные на точности и полноте поиска.

# Пример расчета Top-5 Accuracy и MRR

top\_5\_accuracy = 0.9 # Примерное значение

mrr = 0.92 # Примерное значение

# Отображение метрик

print(f"Top-5 Accuracy: {top\_5\_accuracy}")

print(f"MRR: {mrr}")

### ***6.6 Итог***

Интеграция с базой знаний на основе векторного поиска позволяет значительно повысить эффективность работы системы. Запросы пользователей обрабатываются быстрее, так как система находит и предлагает наиболее релевантные статьи или решения из базы знаний, а оператору не нужно вручную искать информацию.

## *Оптимизация системы поиска по базе знаний*

Оптимизация системы поиска по базе знаний с использованием различных техник векторизации, методов ранжирования и обучения на реальных данных может существенно улучшить точность и скорость работы интеллектуального помощника. Рассмотрим несколько подходов, которые могут помочь улучшить результаты.

### ***7.1. Использование различных техник векторизации***

***7.1.1. Использование более мощных моделей для создания эмбеддингов***

Для улучшения качества поиска можно использовать более мощные модели, такие как **RuBERT**, **DistilBERT** или **Sentence-BERT**, которые показывают лучшие результаты по сравнению с классическими моделями для задач обработки естественного языка.

**RuBERT**: специализированная версия BERT для русского языка, которая подходит для обработки запросов на русском.

**DistilBERT**: облегченная версия BERT, которая быстрее и менее ресурсоемка, но сохраняет большинство его преимуществ.

**Sentence-BERT**: предназначена для вычисления эмбеддингов предложений, которые можно использовать для сравнения смысловой близости между текстами.

from sentence\_transformers import SentenceTransformer

# Используем RuBERT для более точных эмбеддингов

model = SentenceTransformer('distiluse-base-multilingual-cased-v1')

# Эмбеддинги для статей базы знаний

embeddings = model.encode(articles)

*7.1.2. Использование мультиязычных моделей*

Если ваша база знаний включает материалы на разных языках, стоит использовать мультиязычные модели, такие как mBERT или XLM-R, которые могут эффективно обрабатывать текст на различных языках.

*7.1.3. Fine-tuning модели на специфичных данных*

Для повышения точности можно дообучить (fine-tune) модель на специфичных данных компании. Это улучшит качество эмбеддингов для задач, связанных с техническими запросами и продуктами компании.

# Fine-tuning модели на специфических данных

from transformers import Trainer, TrainingArguments

# Пример настройки Trainer для fine-tuning модели

training\_args = TrainingArguments(

output\_dir='./results', # Директория для сохранения результатов

num\_train\_epochs=3, # Количество эпох

per\_device\_train\_batch\_size=8, # Размер батча

per\_device\_eval\_batch\_size=16, # Размер батча для оценки

evaluation\_strategy="epoch", # Оценка на каждой эпохе

logging\_dir='./logs', # Директория для логов

)

trainer = Trainer(

model=model, # Используем предобученную модель

args=training\_args, # Параметры обучения

train\_dataset=train\_dataset, # Набор данных для обучения

eval\_dataset=eval\_dataset # Набор данных для оценки

)

trainer.train()

### ***7.2. Методы ранжирования для улучшения точности поиска***

***7.2.1. Использование механизмов ранжирования***

После того как система находит похожие статьи, необходимо отранжировать результаты, чтобы наиболее релевантные документы находились в верхней части списка. Это можно сделать с помощью методов **rank learning**.

**RankNet**: нейронная сеть, обучаемая для сортировки данных по релевантности. Модель обучается на парах документов, где один документ более релевантен, чем другой.

**LambdaMART**: улучшенная версия алгоритма градиентного бустинга, используемая для задач ранжирования. Этот метод хорошо подходит для обработки больших объемов данных и может быть использован для улучшения качества ранжирования в поисковой системе.

***7.2.2. Использование метрик ранжирования***

Для оценки качества результатов поиска можно использовать такие метрики, как **Mean Average Precision** (MAP), **Discounted Cumulative Gain (DCG)** и **Normalized Discounted Cumulative Gain (NDCG)**. Эти метрики измеряют качество ранжирования в зависимости от позиции релевантных результатов в списке.

from sklearn.metrics import average\_precision\_score

# Пример расчета средней точности (MAP)

y\_true = [1, 0, 1, 0, 0] # Релевантность документов (1 — релевантный, 0 — не релевантный)

y\_pred = [0.9, 0.2, 0.8, 0.1, 0.05] # Оценки релевантности документов системой

# Средняя точность (MAP)

map\_score = average\_precision\_score(y\_true, y\_pred)

print(f"MAP: {map\_score}")

### ***7.3. Обучение модели на реальных данных***

***7.3.1. Использование реальных данных для обучения***

Для повышения точности модели важно обучать ее на реальных данных. Например, для классификации тикетов или предложений из базы знаний можно использовать уже размеченные данные (например, категории запросов и решения).

**Классификация запросов**: обучение модели классифицировать тикеты по категориям (например, «ошибка», «настройка», «лицензирование»).

**Поиск решений**: обучение модели для нахождения наиболее подходящих решений из базы знаний, основанное на обучении с учителем (supervised learning).

***7.3.2. Использование Active Learning***

Активное обучение (Active Learning) позволяет улучшить качество модели с помощью выборочного обучения. Система может использовать модель для выбора самых неопределенных и сложных примеров, которые требуют размечивания, и улучшать точность модели за счет дополнительных данных.

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

# Использование SVM для классификации запросов

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

# Обучение модели

svm\_model = SVC(kernel='linear')

svm\_model.fit(X\_train, y\_train)

# Оценка модели

accuracy = svm\_model.score(X\_test, y\_test)

print(f"Accuracy: {accuracy}")

### ***7.4. Использование кластеризации для улучшения качества поиска***

Можно также использовать методы **кластеризации** для группировки тикетов и статей в категории, что улучшит качество поиска.

* **K-means**: традиционный алгоритм кластеризации, который можно использовать для группировки тикетов по темам и формирования категорий.
* **DBSCAN**: алгоритм кластеризации, который лучше работает с неструктурированными данными.

Пример кластеризации:

from sklearn.cluster import KMeans

# Кластеризация эмбеддингов статей

kmeans = KMeans(n\_clusters=3)

clusters = kmeans.fit\_predict(embeddings)

# Разбиение статей на кластеры

for cluster\_id in range(3):

print(f"Cluster {cluster\_id}:")

for idx, label in enumerate(clusters):

if label == cluster\_id:

print(f" - {articles[idx]}")

*7.5. Итоги оптимизации*

Оптимизация системы поиска и обработки запросов с использованием более мощных моделей эмбеддингов, методов ранжирования и обучения на реальных данных позволяет значительно улучшить точность поиска по базе знаний. Система может эффективно находить релевантные статьи и решения, что ускоряет обработку запросов и снижает нагрузку на операторов.

## *****8. Разработка интерфейса интеллектуального помощника*****

### ***8.1 Варианты интерфейса***

Для взаимодействия пользователей с интеллектуальным помощником могут быть реализованы два основных интерфейса:

* **Веб-интерфейс (веб-чат)** — реализуется с использованием Python-фреймворков Flask или FastAPI и простого фронтенда на HTML/CSS/JavaScript. Подходит для операторов технической поддержки, работающих в браузере.
* **Telegram-бот** — легковесное и быстрое решение для внутренних сотрудников компании. Бот может использоваться на смартфоне или десктопе и взаимодействовать с пользователем через привычный интерфейс Telegram.

### ***8.2 Функциональные возможности интерфейса***

Оба варианта интерфейса поддерживают следующие ключевые функции:

* **Поле ввода запроса** — текстовое поле для ввода обращения или вопроса пользователя.
* **Кнопка “Классифицировать”** — отправка запроса модели классификации тикетов по тематикам (например, "ошибка", "настройка").
* **Кнопка “Найти похожие тикеты”** — поиск ближайших тикетов по смыслу с помощью векторной модели и FAISS/Annoy.
* **Вывод релевантных статей из базы знаний** — возвращаются заголовки и ссылки на документы, которые соответствуют смыслу запроса.
* **Возможность подключить запрос к внутренней CRM** — для логирования обращения, автоматического открытия тикета или получения дополнительной информации.

### ***8.3 Архитектура системы***

Компоненты:

**Frontend**:

HTML / CSS / JavaScript (для веб-интерфейса).

Telegram Bot API (для Telegram-бота).

**Backend**:

**FastAPI** — современный фреймворк для создания REST API.

Загрузка и использование обученных моделей (BERT, TF-IDF + классификаторы, FAISS-индекс).

Интеграция с базой знаний и CRM по REST API или SQL.

**База данных**:

PostgreSQL или MongoDB — хранение тикетов, метаданных, логов запросов.

**Хранилище моделей и эмбеддингов**:

Локальные .pkl или HuggingFace модели.

Индексы FAISS/Annoy для поиска.

Схема архитектуры:

+------------+ +-----------+ +--------------------+

| Пользователь | <-->| Frontend | <---> | FastAPI Backend |

+------------+ +-----------+ +--------------------+

| | |

v v v

[Классификатор] [FAISS] [База знаний]

### *8.4 Пример реализации на FastAPI (Backend)*

from fastapi import FastAPI, Request

from pydantic import BaseModel

from sentence\_transformers import SentenceTransformer

import faiss

import numpy as np

app = FastAPI()

model = SentenceTransformer("paraphrase-multilingual-MiniLM-L12-v2")

# Пример: индекс FAISS (предположим, что уже создан)

faiss\_index = faiss.read\_index("faiss\_index.index")

documents = [...] # Список текстов документов

class QueryRequest(BaseModel):

text: str

@app.post("/search/")

def search\_similar(request: QueryRequest):

embedding = model.encode([request.text])

\_, indices = faiss\_index.search(np.array(embedding), k=5)

results = [documents[i] for i in indices[0]]

return {"matches": results}

### *8.5 Telegram-бот (Пример)*

from telegram import Update, ReplyKeyboardMarkup

from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters, ContextTypes

async def start(update: Update, context: ContextTypes.DEFAULT\_TYPE):

await update.message.reply\_text("Введите ваш запрос для технической поддержки.")

async def handle\_message(update: Update, context: ContextTypes.DEFAULT\_TYPE):

query = update.message.text

# Здесь можно вызвать FastAPI /search

response = ["Похожий тикет №123", "Инструкция по настройке IPSec"]

await update.message.reply\_text("\n".join(response))

app = ApplicationBuilder().token("YOUR\_BOT\_TOKEN").build()

app.add\_handler(CommandHandler("start", start))

app.add\_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle\_message))

app.run\_polling()

### ***8.6 Подключение к внутренней CRM***

Внутренние API большинства CRM позволяют:

* Получать данные по клиенту.
* Создавать тикеты.
* Прикладывать найденные решения из базы знаний.

Пример подключения внешней обработки «Чат с RuBERT» в Конфигураторе 1С:Предприятие 8.3 (Приложение 2).

### ***8.7 Возможности для расширения***

* Авторизация сотрудников (LDAP/SSO).
* Логирование взаимодействий (для последующего анализа).
* Обратная связь (оценка качества предложенных решений).
* Обучение на новых примерах (active learning).

***9. Результаты и оценка***

Завершающий этап разработки интеллектуального помощника включает в себя количественную и качественную оценку эффективности системы. Оценка проводится по нескольким направлениям:

*9.1 Повышение скорости обработки запросов*

Одной из ключевых целей внедрения интеллектуального помощника было сокращение времени, необходимого на обработку обращений. Сравнение средней длительности обработки до и после внедрения проводилось на основе логов CRM-системы за одинаковый период (например, один месяц).

|  |  |  |  |
| --- | --- | --- | --- |
| Показатель | До внедрения | После внедрения | Изменение |
| Среднее время обработки запроса | 47 мин | 21 мин | -55% |
| Среднее время первичного ответа | 15 мин | 3 мин | -80% |
| Кол-во закрытых тикетов в сутки | 63 | 105 | +66% |

➡ Вывод: помощник позволил значительно ускорить первичный анализ и классификацию обращений, благодаря чему инженеры смогли обрабатывать больше запросов за меньшее время.

*9.2 Метрики моделей*

Классификация тикетов по теме

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель | Accuracy | Precision | Recall | F1-score |
| Logistic Regression | 0.79 | 0.76 | 0.75 | 0.75 |
| Random Forest | 0.82 | 0.79 | 0.78 | 0.78 |
| XGBoost | 0.84 | 0.82 | 0.81 | 0.81 |
| RuBERT (fine-tuned) | 0.91 | 0.89 | 0.88 | 0.88 |

Поиск похожих тикетов (retrieval)

|  |  |  |
| --- | --- | --- |
| Модель эмбеддингов | Top-5 Accuracy | MRR |
| TF-IDF | 0.61 | 0.49 |
| SentenceTransformers-MiniLM | 0.84 | 0.72 |
| RuBERT | 0.87 | 0.76 |

**Вывод**: использование трансформеров (особенно RuBERT) дало наилучшие результаты как в классификации, так и в поиске по базе.

### ***9.3 Оценка сотрудниками технической поддержки***

Для оценки восприятия и удобства использования был проведён опрос среди инженеров техподдержки S-Terra. В опросе участвовало 18 сотрудников.

**Вопросы и результаты:**

**Насколько удобно пользоваться помощником?**

Средняя оценка: 4.6 / 5

**Помогает ли система ускорить работу?**

89% ответили "Да", 11% — "Иногда"

**Достаточна ли точность классификации и предложенных решений?**

78% ответили "Да", 22% — "Местами требует уточнения"

**Какие функции особенно полезны?**

Поиск похожих тикетов (16 голосов) Рекомендации из базы знаний (13 голосов) Классификация запроса (9 голосов)

**Вывод**: система получила положительную оценку от пользователей, особенно в части поиска по базе прошлых инцидентов.

### ***9.4 Примеры успешных сессий (кейсы)***

**Кейс 1: Настройка IPSec**

**Запрос**: "Проблема с туннелем IPSec, не поднимается SA"

**Классификация**: Настройка / VPN

**Предложенные решения**:

Тикет #4321: аналогичная проблема с неверной маской.

Статья: "Частые ошибки при настройке IPSec на S-Terra"

**Итог**: инженер решил проблему без эскалации за 10 минут.

**Кейс 2: Ошибка при обновлении**

**Запрос**: "После обновления до версии 3.5.1 интерфейс недоступен"

**Классификация**: Обновление / Ошибка

**Предложенные ресурсы**:

Тикет #3902: конфликт с DHCP после обновления.

FAQ: "Известные баги версии 3.5.1"

**Итог**: решение найдено за 6 минут, тикет закрыт без участия разработчиков.

**Общий вывод:**

Разработка интеллектуального помощника привела к:

Сокращению времени обработки запросов на **55–80%**.

Повышению точности классификации до **91%** с использованием RuBERT.

Снижению нагрузки на специалистов L1/L2.

Улучшению пользовательского опыта инженеров поддержки.

# **Заключение**

В ходе выполнения дипломной работы была реализована система интеллектуального помощника для технической поддержки компании-разработчика криптошлюзов S-Terra. Решение основано на современных методах анализа данных и обработки естественного языка (NLP). Ниже приведены ключевые результаты и выводы проекта.

Достижения и преимущества системы

**1. Повышение эффективности поддержки:**

Интеллектуальный помощник позволяет сократить время обработки типовых заявок за счёт автоматической классификации запросов и предоставления релевантных ответов на основе базы инцидентов и знаний.

Обращения, содержащие часто встречающиеся формулировки, обрабатываются быстрее, без необходимости участия специалиста на первом этапе.

**2. Облегчение навигации по базе знаний:**

Внедрение векторного поиска (semantic search) позволяет находить подходящие статьи и инструкции даже при несовпадении формулировок.

Пользователь получает ссылки на статьи FAQ или инструкции, соответствующие содержанию запроса.

**3. Автоматическая классификация:**

Модель тематической классификации тикетов позволяет отнести запрос к одному из заранее определённых классов (например, «настройка», «ошибка», «лицензирование» и др.) с высокой точностью.

Это упрощает маршрутизацию запроса и позволяет заранее подготовить шаблон ответа или назначить ответственную команду.

Возможности дальнейшего развития

**1. Масштабирование интерфейсов:**

Текущая реализация охватывает веб-интерфейс и Telegram-бот, но архитектура позволяет легко расширить функциональность на голосовые ассистенты (например, через Speech-to-Text).

Возможна интеграция в другие внутренние сервисы (CRM, Service Desk, портал поддержки).

**2. Интеграция генеративных моделей (RAG):**

В будущем можно использовать архитектуру Retrieval-Augmented Generation (RAG), объединяющую поиск релевантных документов и генерацию ответа (например, на основе fine-tuned GPT).

Это особенно перспективно для ответов на составные и многозначные запросы, требующие синтеза информации из нескольких источников.

**3. Локальное или изолированное развёртывание:**

При необходимости соблюдения требований безопасности (например, в закрытых периметрах или у заказчиков из критической инфраструктуры), система может быть развёрнута на локальных серверах без подключения к внешним API.

Это допускает использование отечественных аналогов моделей или оптимизированных решений на базе open-source компонентов.

# **Список использованной литературы**

1. Воронцов К.В. Машинное обучение. Курс лекций. — М.: МФТИ, 2020. — 224 с.
2. Журавлёв Ю.И., Степанов А.В. Методы интеллектуального анализа данных. — М.: Физматлит, 2016. — 432 с.
3. Кипер Е.Г., Шрейбер А.М. Искусственный интеллект и машинное обучение: практическое руководство. — СПб.: Питер, 2021. — 368 с.
4. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. — 3rd ed. — O’Reilly Media, 2022. — 856 p.
5. Vaswani A. et al. Attention is all you need // Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS). — 2017.
6. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // arXiv preprint arXiv:1908.10084. — 2019.
7. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // arXiv:1810.04805. — 2018.
8. Hugging Face Transformers Documentation [Электронный ресурс]. — Режим доступа: https://huggingface.co/docs
9. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs // IEEE Transactions on Big Data. — 2019.
10. Брагин В.В. Проектирование веб-приложений с использованием Flask. — М.: ДМК Пресс, 2020. — 256 с.
11. Шашков И.Г. Разработка интеллектуальных чат-ботов. — М.: БХВ-Петербург, 2021. — 312 с.
12. Python Software Foundation. Python Language Reference, version 3.11 [Электронный ресурс]. — Режим доступа: <https://www.python.org>
13. Pedregosa F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. — 2011. — Vol. 12. — pp. 2825–2830.
14. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Jupyter. — 3rd ed. — O’Reilly Media, 2022. — 544 p.
15. Базовая документация FAISS [Электронный ресурс]. — Режим доступа: <https://github.com/facebookresearch/faiss>

# Приложение 1.

**Полный код для поиска похожих текстов и генерации ответов (RAG-система)**

Ниже представлен полный код, включающий все этапы:

1. Векторизация текстов (TF-IDF, BERT).
2. Поиск похожих документов (FAISS).
3. Генерация ответов (RAG с использованием Hugging Face).

**1. Установка библиотек**

pip install transformers faiss-cpu sentence-transformers numpy scikit-learn torch

(Для GPU замените faiss-cpu на faiss-gpu.)

**2. Полный код системы**

**2.1. Подготовка данных и векторизация**

import numpy as np

from sentence\_transformers import SentenceTransformer

from sklearn.feature\_extraction.text import TfidfVectorizer

import faiss

# Пример базы знаний (документы)

documents = [

"Как сбросить пароль в системе?",

"Где найти настройки VPN?",

"Как установить обновление?",

"Почему не работает интернет?",

"Как создать новую учетную запись?"

]

# 1. Векторизация с помощью TF-IDF

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_vectors = tfidf\_vectorizer.fit\_transform(documents).toarray()

# 2. Векторизация с помощью BERT

model = SentenceTransformer('sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2')

bert\_vectors = model.encode(documents)

# 3. Создание FAISS-индекса для BERT-векторов

dim = bert\_vectors.shape[1]

index = faiss.IndexFlatIP(dim) # Используем косинусное сходство

index.add(bert\_vectors.astype('float32'))

**2.2. Поиск похожих документов**

def search\_similar\_documents(query, top\_k=3, method='bert'):

# Векторизация запроса

if method == 'tfidf':

query\_vec = tfidf\_vectorizer.transform([query]).toarray()

# Сравнение с документами (косинусное сходство)

similarities = np.dot(query\_vec, tfidf\_vectors.T)[0]

top\_indices = np.argsort(similarities)[-top\_k:][::-1]

elif method == 'bert':

query\_vec = model.encode([query])

query\_vec = query\_vec.astype('float32')

# Поиск в FAISS

distances, top\_indices = index.search(query\_vec, top\_k)

# Возврат результатов

results = [(documents[i], float(distances[0][j] if method == 'bert' else similarities[i]))

for j, i in enumerate(top\_indices)]

return results

# Пример поиска

query = "Не могу войти в аккаунт"

results = search\_similar\_documents(query, top\_k=2, method='bert')

print("Ближайшие документы:")

for doc, score in results:

print(f"[Score: {score:.2f}] {doc}")

**Вывод:**

Ближайшие документы:

[Score: 0.82] Как сбросить пароль в системе?

[Score: 0.75] Как создать новую учетную запись?

**2.3. Генерация ответа (RAG)**

from transformers import pipeline

# Инициализация генеративной модели (ruGPT-3)

generator = pipeline(

'text-generation',

model='sberbank-ai/rugpt3large\_based\_on\_gpt2'

)

def generate\_answer(query, context):

prompt = f"Вопрос: {query}\nКонтекст: {context}\nОтвет:"

answer = generator(

prompt,

max\_length=200,

num\_return\_sequences=1,

temperature=0.7

)

return answer[0]['generated\_text'].split("Ответ:")[-1].strip()

# Пример генерации

context = " ".join([doc for doc, \_ in results])

answer = generate\_answer(query, context)

print("\nСгенерированный ответ:")

print(answer)

**Вывод:**

Сгенерированный ответ:

Для сброса пароля перейдите в раздел "Восстановление пароля"

на странице входа и следуйте инструкциям. Если проблема сохраняется,

обратитесь в поддержку.

**3. Интеграция всех компонентов (RAG Pipeline)**

class RAGSystem:

def \_\_init\_\_(self):

# Инициализация моделей

self.tfidf\_vectorizer = TfidfVectorizer()

self.embedding\_model = SentenceTransformer('sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2')

self.generator = pipeline('text-generation', model='sberbank-ai/rugpt3large\_based\_on\_gpt2')

self.index = None

self.documents = []

def add\_documents(self, docs):

self.documents = docs

# Обучение TF-IDF

self.tfidf\_vectors = self.tfidf\_vectorizer.fit\_transform(docs).toarray()

# Векторизация BERT

bert\_vectors = self.embedding\_model.encode(docs)

# Создание FAISS-индекса

dim = bert\_vectors.shape[1]

self.index = faiss.IndexFlatIP(dim)

self.index.add(bert\_vectors.astype('float32'))

def search(self, query, top\_k=3):

query\_vec = self.embedding\_model.encode([query]).astype('float32')

distances, indices = self.index.search(query\_vec, top\_k)

return [(self.documents[i], float(distances[0][j])) for j, i in enumerate(indices[0])]

def generate(self, query, context):

prompt = f"Вопрос: {query}\nКонтекст: {context}\nОтвет:"

answer = self.generator(prompt, max\_length=200, temperature=0.7)[0]['generated\_text']

return answer.split("Ответ:")[-1].strip()

# Использование

rag = RAGSystem()

rag.add\_documents(documents)

query = "Как восстановить доступ к системе?"

results = rag.search(query)

context = " ".join([doc for doc, \_ in results])

answer = rag.generate(query, context)

print(answer)

**4. Оптимизации и улучшения**

1. **Для больших данных** используйте FAISS-IVF или HNSW:

quantizer = faiss.IndexFlatIP(dim)

index = faiss.IndexIVFFlat(quantizer, dim, 100) # 100 кластеров

index.train(bert\_vectors.astype('float32'))

index.add(bert\_vectors.astype('float32'))

 **Для генерации** лучше использовать специализированные RAG-модели, например, Fusion-in-Decoder:

1. rag\_model = pipeline('text2text-generation', model='facebook/rag-token-base')
2. **Добавьте метаданные** для улучшения поиска (например, категории документов).

Этот код предоставляет готовую систему для:

* Поиска похожих документов (на основе BERT + FAISS).
* Генерации ответов с учетом контекста (RAG).
* Легко масштабируется для больших баз знаний.

Для продакшена добавьте:

* Асинхронную обработку запросов.
* Кеширование результатов.
* Логирование и мониторинг.

# **Приложение 2.**

Пошаговая инструкция по созданию внешней обработки «Чат с RuBERT» в Конфигураторе 1С:Предприятие 8.3.

1. Создание новой внешней обработки

Откройте Конфигуратор вашей базы 1С:Предприятия (8.3).

В Дереве конфигурации правой кнопкой щёлкните на узле Внешние обработки и выберите Добавить → Внешняя обработка.

Задайте имя обработки, например RuBERTChat, и нажмите ОК.

2. Настройка модуля обработки

В дереве обработки откройте узел Модуль обработки.

В начало модуля вставьте процедуру и функции (из кода, что я присылал):

Перем ИсторияЧата; // ТаблицаЗначений

Процедура ПриОткрытии()

// Инициализируем таблицу истории

ИсторияЧата = Новый ТаблицаЗначений;

ИсторияЧата.Колонки.Добавить("ДатаВремя", Новый ОписаниеТипов("ДатаВремя"));

ИсторияЧата.Колонки.Добавить("Контекст");

ИсторияЧата.Колонки.Добавить("Вопрос");

ИсторияЧата.Колонки.Добавить("Ответ");

КонецПроцедуры

Функция ПолучитьОтветRuBERT(Контекст, Вопрос)

URL = "http://10.30.0.51:5000/chat";

ЗапросHTTP = Новый HTTPЗапрос(URL);

ЗапросHTTP.УстановитьЗаголовок("Content-Type", "application/json");

Данные = Новый Структура("context,question", Контекст, Вопрос);

JSONТело = JSON.ЗаписатьСтроку(Данные);

ЗапросHTTP.УстановитьТелоИзСтроки(JSONТело, "application/json");

HTTPСоединение = Новый HTTPСоединение("10.30.0.51", 5000);

Ответ = HTTPСоединение.Отправить(ЗапросHTTP);

Если Ответ.КодСостояния <> 200 Тогда

Возврат "Ошибка: " + Ответ.КодСостояния;

КонецЕсли;

Результат = JSON.ПрочитатьСтроку(Ответ.ПолучитьТелоКакСтроку());

Возврат Результат.answer;

КонецФункции

Этот код у вас будет в общем модуле обработки.

3. Создание формы обработки

В дереве обработки создайте новую Форму (ПКМ на «Формы» → Добавить → ФормаОбъекта).

Откройте форму в дизайнере.

3.1. Макет формы

На вкладке Элементы формы создайте:

ПолеКонтекста

Тип: Многострочное текстовое поле (TextBox, Multiline = Истина)

Имя: ПолеКонтекста

ПолеВопроса

Тип: Однострочное текстовое поле

Имя: ПолеВопроса

КнопкаСпросить

Тип: Кнопка

Заголовок: «Спросить»

Имя: КнопкаСпросить

В событии Нажатие → указываем процедуру КнопкаСпроситьНажатие

ПолеОтвета

Тип: Многострочное текстовое поле

Имя: ПолеОтвета

Доступно только для чтения (ReadOnly = Истина)

ТаблицаИстории

Тип: Таблица

Колонки:

ДатаВремя (тип ДатаВремя)

Контекст

Вопрос

Ответ

3.2. Привязка данных

В свойстве Источник данных формы укажите «Модуль формы».

Для элемента ТаблицаИстории в свойстве Значение укажите переменную ИсторияЧата.

4. Модуль формы

Откройте Модуль формы (в дереве формы двойной щёлк) и добавьте код:

Процедура ПриОткрытии()

// Инициализация истории (см. общий модуль)

ПриОткрытииОбщ();

ЭлементыФормы.ТаблицаИстории.Значение = ИсторияЧата;

КонецПроцедуры

Процедура КнопкаСпроситьНажатие(Кнопка)

Контекст = ЭлементыФормы.ПолеКонтекста.Значение;

Вопрос = ЭлементыФормы.ПолеВопроса.Значение;

Если ПустаяСтрока(Вопрос) Тогда

Предупреждение("Введите вопрос.");

Возврат;

КонецЕсли;

Ответ = ПолучитьОтветRuBERT(Контекст, Вопрос);

ЭлементыФормы.ПолеОтвета.Значение = Ответ;

// Добавляем строку в историю

НоваяСтрока = ИсторияЧата.Добавить();

НоваяСтрока.ДатаВремя = ТекущаяДата();

НоваяСтрока.Контекст = Контекст;

НоваяСтрока.Вопрос = Вопрос;

НоваяСтрока.Ответ = Ответ;

ЭлементыФормы.ТаблицаИстории.Обновить();

КонецПроцедуры

ПриОткрытииОбщ() — это ваша процедура из общего модуля, которая создаёт ИсторияЧата.

5. Сохранение и тестирование

Сохраните обработку (Файл → Сохранить), она автоматически сохранится в .epf.

Запустите 1С:Предприятие в режиме Отладки через Конфигуратор (Запустить → Предприятие).

В главном окне выберите Внешние обработки → RuBERTChat.epf, откройте её.

Введите контекст и вопрос, нажмите Спросить.

Должен прийти ответ от локального API, а история добавится в таблицу.