

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пензенский государственный университет»

Факультет

вычислительной техники

Кафедра

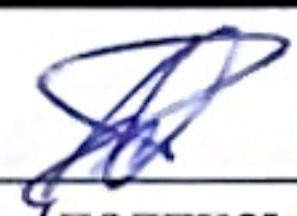
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
на тему
**КОРРЕКТИРОВЩИК ТЕКСТА В
ПОИСКОВЫХ ЗАПРОСАХ В БРАУЗЕРЕ С
ИСПОЛЬЗОВАНИЕМ ИИ**

Студент



30.05.2025 Отставнов Александр Михайлович
(подпись, дата) (ФИО полностью)

Руководитель



30.05.2025
(подпись, дата)

Синев М.П.
(фамилия, инициалы)

Нормоконтролёр



30.05.2025
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

зав. кафедрой Программирования ИИ, 30.05.2025
(должность, место работы) (подпись, дата)

Мартышкин А.И.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 № 16)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой

отлично

(протокол заседания ГЭК от 5.06.2025 № 4)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 29 » сентября 2023 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Отставнов Александр Михайлович гр.23ВВИМ1 факультета ВТ
2. Тема работы «Корректировщик текста в поисковых запросах в браузере с использованием ИИ»
Тема утверждена приказом ПГУ № 0986 от "29" сентября 2023г.
3. Руководитель магистерской диссертации к.т.н., доцент Синев М.П.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: разработка системы корректировки текста поискового запроса с применением ИИ.

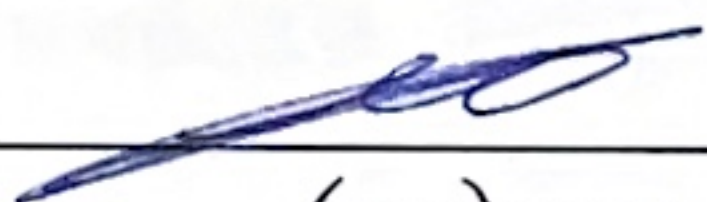
Разрабатываемая система должна выполнять автоматическую корректировку введенного текста, а также генерацию альтернативных формулировок запроса на основе семантической близости.

5. Перечень подлежащих разработке вопросов
 1. Анализ предметной области.
 2. Выявление основных типов ошибок в пользовательских поисковых запросах.
 3. Разработка алгоритма символьной коррекции пользовательского ввода и модуля генерации альтернативных формулировок поисковых запросов.
 4. Тестирование разработанной системы.
 5. Анализ полученных результатов.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	11%	29.11.2023- 29.01.2024	
2.	Анализ требований	11%	29.01.2024- 30.03.2024	
3.	Выбор технологий, применяемых при проектировании и разработки	22%	30.03.2024- 30.05.2024	
4.	Проектирование и реализация системы	27%	01.06.2024- 01.09.2024	
5.	Тестирование	17%	02.09.2024- 02.10.2024	
6.	Анализ полученных результатов исследования	6%	01.04.2025- 29.04.2025	
7.	Оформление пояснительной записки	6%	30.04.2025- 25.05.2025	

Дата выдачи задания " 29 " сентября 20 23 г.

Руководитель магистерской диссертации 
(подпись, дата)

Синев М.П.

Задание к исполнению принял студент 
(подпись, дата)

Отставнов А.М.

Работу к защите допустить
Декан факультета  04.06.2025
(подпись, дата)

Фионова Л.Р.

РЕФЕРАТ

Пояснительная записка содержит 68 листов, 8 рисунков, 8 таблиц, 16 источников, 1 приложение.

КОРРЕКТИРОВЩИК ТЕКСТА, ПОИСКОВОЙ ЗАПРОС, РЕЛЕВАНТНОСТЬ, RAG-СИСТЕМА, ОПЕЧАТКИ, PYTHON, SENTENCE-BERT, SYMSPELL, SCIKIT-LEARN, GOOGLE COLAB

Целью работы является разработка интеллектуального модуля предварительной обработки пользовательских поисковых запросов в браузере. Основная задача системы — автоматическая коррекция орфографических и семантических искажений, а также генерация логически релевантных альтернативных формулировок для повышения точности поиска. Модуль должен функционировать в режиме реального времени и быть пригодным для интеграции в существующие поисковые интерфейсы.

В результате выполнения проекта реализована и протестирована система, включающая в себя несколько логически связанных модулей: символьной коррекции на основе алгоритма SymSpell, генерации эмбеддингов с использованием модели Sentence-BERT, семантического сопоставления запросов с тегами, поиска тематически близких тегов и формирования альтернативных вариантов ввода.

Проект выполнен на языке программирования Python в среде Google Colab. Проведены функциональные испытания всех компонентов, подтверждающие корректную работу системы в условиях частично искажённого текста. Полученные результаты демонстрируют применимость разработанного решения в задачах интеллектуального поиска, автодополнения и анализа пользовательского ввода.

					ПГУ 1.090401.11.001 ПЗ				
Изм.	Лист	№ докум	Подпись	Дата	Корректировщик текста в поисковых запросах в браузере с использованием ИИ Пояснительная записка	Лит.		Лист	Листов
Разраб.		Отставнов А.М.		30.05.25				2	68
Провер.		Синев М.П.		30.05.25					
Н. Контр.		Бычков А.С.		30.05.25					
Утв.		Митрохин М.А.		30.05.25					
						ФВТ, каф. ВТ гр. 23ВВИм1			

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	6
ВВЕДЕНИЕ.....	7
1 Анализ предметной области	9
1.1 Типы ошибок в поисковых запросах	9
1.2 Методы исправления ошибок	10
1.2.1 Словарные и символьные методы	10
1.2.2 Статистические и вероятностные модели	12
1.2.3 Контекстные и нейросетевые модели	13
1.2.4 Гибридные подходы	15
1.3 Выбор инструментов разработки	16
1.3.1 Выбор языка программирования.....	16
1.3.1.1 Python	16
1.3.1.2 Java.....	16
1.3.1.3 Julia	18
1.3.1.4 Сравнительный анализ	18
1.3.2 Выбор среды разработки	20
1.3.2.1 Google Colab	20
1.3.2.2 Jupyter Notebook	21
1.3.2.3 PyCharm.....	23
1.3.2.4 Сравнительный анализ	24
1.4 Выводы.....	26
2 Реализация	27
2.1 стек технологий, используемый для разработки	27
2.2 Требования к разработке	29

2.3	Алгоритм работы сервиса	31
2.4	Описание структуры и подготовки датасета.....	33
2.5	Взаимодействие модулей системы.....	35
2.6	Реализация модулей системы	37
2.6.1	Модуль символьной коррекции.....	37
2.6.2	Модуль генерации эмбедингов.....	39
2.6.3	Модуль семантического сопоставления	40
2.6.4	Модуль поиска похожих тегов	41
2.6.5	Модуль генерации альтернативных запросов.....	43
2.7	Выводы.....	45
3	Методы тестирования.....	47
3.1	Проверка модуля символьной коррекции	47
3.1.1	Сценарий: одна ошибка в одном слове.....	48
3.1.2	Сценарий: две ошибки в одном слове	50
3.1.3	Сценарий: три ошибки в одном слове	51
3.1.4	Вывод по разделу	52
3.2	Проверка модуля генерации эмбедингов	52
3.3	Проверка модуля семантического сопоставления.....	53
3.4	Проверка модуля поиска похожих тегов	55
3.5	Проверка модуля генерации альтернативных тегов	56
3.6	Выводы.....	57
	ЗАКЛЮЧЕНИЕ	59
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	60
	ПРИЛОЖЕНИЕ А Листинг программы	62

___Приложение A.1 Файл <code>symspell_corrector.py</code>	63
___Приложение A.2 Файл <code>transformer.py</code>	64
___Приложение A.3 Файл <code>best_tags.py</code>	65
___Приложение A.4 Файл <code>find_tags.py</code>	66
___Приложение A.5 Файл <code>generator_query.py</code>	67
___Приложение A.6 Файл <code>main.py</code>	68

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

BERT (от англ. Bidirectional Encoder Representations from Transformers) – двунаправленная модель представления текста на основе трансформеров

CSV (от англ. Comma-Separated Values) – текстовый формат хранения табличных данных

NLP (от англ. Natural Language Processing) – обработка естественного языка

S-BERT – модифицированная версия модели BERT для работы с предложениями

SymSpell – алгоритм символьной коррекции на основе расстояния редактирования

UML – (от англ. Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения.

ВВЕДЕНИЕ

Ежедневно миллионы пользователей по всему миру формулируют поисковые запросы в интернете, ожидая от систем быстрой и точной выдачи информации. Однако в действительности запросы нередко содержат опечатки, грамматические ошибки, неполные формулировки, что приводит к снижению эффективности поиска.

Одним из перспективных направлений в области информационного поиска является создание систем интеллектуальной корректировки текстов запросов на основе методов ИИ. Такие системы способны автоматически выявлять и устранять ошибки в пользовательском вводе, подбирать релевантные альтернативы и улучшать общую точность поиска. Использование языковых моделей, эмбедингов и машинного обучения позволяет значительно повысить качество взаимодействия пользователя с поисковым интерфейсом.

Объект исследования: система корректировки поисковых запросов.

Предмет исследования: алгоритмы и методы корректировки текстов поисковых запросов с использованием технологий искусственного интеллекта.

Цель исследования: разработка и исследование интеллектуального корректировщика поисковых запросов, обеспечивающего автоматическое исправление ошибок и подбор релевантных альтернативных формулировок с применением моделей глубокого обучения и механизмов семантической близости.

Задачи исследования:

- анализ предметной области,
- подбор моделей для построения системы,
- разработка системы корректировки поискового запроса.

Актуальность (новизна) исследования: создание системы для обработки поисковых запросов, которая включает в себя автоматическую коррекцию текста и генерацию более точных и актуальных формулировок на основе RAG-архитектуры и семантической близости.

Практическая и научная значимость: интеграция разработанного модуля в поисковые системы для повышения устойчивости к различным видам ошибок.

Границы исследования: основной акцент ставится на реализации модулей коррекции, семантического сопоставления и генерации альтернативных запросов. В границы исследования не входят вопросы построения поискового индекса, архитектура хранения, а также обработка не текстовых запросов.

1 Анализ предметной области

В современном информационном пространстве поисковые системы (Google, Bing, Yandex и др.) выступают в роли основных посредников между пользователем и массивами данных. Качество поиска зависит напрямую от качества вводимого запроса. Однако запросы, которые формулируют пользователи, зачастую далеки от идеальных: они могут быть короткими, неструктурированными, содержать опечатки, жаргон, сленг, ошибки перевода или неполные формулировки.

Согласно исследованиям компаний Google и Microsoft до 10–15% всех пользовательских запросов содержат ошибки — орфографические, синтаксические или смысловые. Это приводит к снижению релевантности результатов или полному их отсутствию. Так, в отчётах Google Search Quality и в исследовании Microsoft Bing "Context-sensitive Spelling Correction Using Web Search" подчёркивается, что исправление ошибок является ключевым этапом обработки запроса для повышения качества поиска.

Задача автоматической корректировки текста запроса заключается в его преобразовании в максимально приближённый к «правильному» и интерпретируемому системой вид, что позволяет существенно улучшить точность и полноту выдачи результатов.

1.1 Типы ошибок в поисковых запросах

Ошибки в поисковых запросах можно условно разделить на следующие категории:

- орфографические ошибки. Возникают при неверном написании слов, чаще всего вследствие недостаточного знания языка или фонетической путаницы,

- опечатки. Представляют собой случайные ошибки, допускаемые при наборе текста, в основном из-за промахов по клавишам или неправильного порядка ввода символов. Часто имеют механическую природу и не связаны с незнанием языка,

– грамматические и лексические ошибки. Включают в себя нарушение правил синтаксиса, морфологии или согласования частей речи, а также использование слов в некорректных значениях или формах,

– семантические ошибки. Они возникают, когда формально корректный текст неверно передаёт замысел автора из-за неточной формулировки. В отличие от грамматических или синтаксических ошибок, они не нарушают норм языка, но искажают смысл на уровне интерпретации. Подобные неточности особенно критичны в технической и юридической коммуникации, где двусмысленность приводит к практическим ошибкам.

Подобная классификация способствует точной идентификации типа ошибки, что является ключевым условием для разработки оптимального алгоритма её автоматического исправления. Однако эффективная коррекция требует комплексного подхода, выходящего за рамки простого анализа символов. Важно учитывать контекст, статистическую значимость лексики, а также прагматический аспект – коммуникативное намерение автора.

1.2 Методы исправления ошибок

Системы интеллектуальной обработки поисковых запросов используют разнообразные подходы для автоматического устранения ошибок в пользовательском вводе. Эти подходы различаются по уровню обобщения, сложности, устойчивости к контексту и вычислительным затратам. Ниже рассмотрены основные классы методов, применяемых в современных системах коррекции.

1.2.1 Словарные и символьные методы

Словарные и символьные методы коррекции текста относятся к фундаментальным подходам автоматической обработки текстовых данных. Их теоретической основой выступает гипотеза о том, что большинство механических ошибок и опечаток могут быть устранены посредством сопоставления с эталонным словарём корректных лексических единиц. Характерной особенностью данных алгоритмов является их ориентация

исключительно на формальное сходство лексем на символьном уровне, без привлечения контекстного анализа или учёта синтаксических функций слов в предложении.

Ключевым элементом символьных методов является расстояние редактирования — метрика, определяющая минимальное количество операций (вставка, удаление, замена, перестановка), необходимых для трансформации одного слова в другое.

Ниже представлены одни из самых популярных алгоритмов коррекции:

- Levenshtein Distance — классический подход, учитывающий три основные операции: вставку, удаление и замену символа. Наиболее часто применяется в простых задачах автокоррекции, а также в предварительном сопоставлении текстов,

- Damerau-Levenshtein — расширяет Levenshtein за счёт добавления перестановки соседних символов, что особенно эффективно для обработки типичных опечаток,

- SymSpell — оптимизированный алгоритм, использующий префиксное дерево (trie) и предварительно построенные словарные индексы для быстрого поиска наиболее близких по написанию слов. Отличается высокой скоростью за счёт предобработки словаря и минимизации вычислений во время запроса.

Символьные методы широко применяются в поисковых движках (Elasticsearch, Lucene), мобильных клавиатурах, корректорах ввода и чат-ботах, где важна скорость реакции и минимальная задержка.

Преимущества:

- высокая скорость работы, особенно при использовании структур типа trie и hash-map,
- устойчивость к типичным опечаткам и искажениям,
- отсутствие необходимости в обучающих данных.

Недостатки:

- не учитывается контекст слова в запросе,
- нет поддержки многословных ошибок или семантической неоднозначности,
- невозможно различить слова с близким написанием, но разным смыслом.

Словарные и символьные методы остаются надёжным и экономичным инструментом первичной коррекции пользовательского ввода. Несмотря на их ограниченность в работе с контекстом и семантикой, они эффективно устраняют наиболее частотные искажения, особенно в условиях низких вычислительных ресурсов. В современных интеллектуальных системах такие алгоритмы часто используются в качестве первого этапа перед более глубокой семантической обработкой текста.

1.2.2 Статистические и вероятностные модели

Статистические методы опираются на анализ частотности слов и последовательностей слов (n-грамм), извлечённых из больших текстовых корпусов или логов поисковых запросов. Они пытаются предсказать, какое корректное слово или фраза, скорее всего, стояло за ошибочным вводом пользователя.

Классическим примером является модель зашумлённого канала (Noisy Channel Model), где предполагается, что пользователь имел в виду некий исходный (правильный) запрос, но в процессе его ввода «искажил» его (в результате опечатки, автозамены и т.п.). Задача системы — восстановить наиболее вероятный исходный вариант по формуле:

Для оценки этих вероятностей используются частоты слов и фраз в больших корпусах, биграммы, триграммы, а также байесовские оценки. Более современные подходы включают нейросетевые языковые модели, обученные на задачах восстановления повреждённого текста.

Преимущества:

- хорошо работают на частотных данных,

- адаптируются к статистике запросов в конкретной системе,
- применимы без сложных моделей обучения,
- позволяют учитывать вероятности нескольких кандидатов.

Недостатки:

- требуют больших объёмов обучающих данных,
- сложны в интерпретации и настройке,
- не решают проблему контекста на уровне предложения.

Статистические и вероятностные методы актуальны при обработке больших массивов пользовательских запросов и логов. Они особенно полезны в системах, где необходимо быстро адаптироваться к текущим трендам, типичным ошибкам и частотным фразам. Однако их ограниченность в работе с контекстом и глубоким смыслом ограничивает применение в задачах, где требуется точная семантическая интерпретация. В современных решениях они часто служат дополнением к нейросетевым моделям.

1.2.3 Контекстные и нейросетевые модели

Контекстные и нейросетевые модели представляют собой современное направление в области обработки естественного языка, опирающееся на архитектуры трансформеров и методы обучения на больших текстовых корпусах. Эти подходы обеспечивают глубокое понимание смысла текста за счёт моделирования взаимосвязей между словами в контексте всего предложения.

Ключевая идея заключается в преобразовании текста в векторы фиксированной размерности — семантические эмбединги, которые отражают смысл всего предложения, а не только отдельных слов. Благодаря этому становится возможным выполнять поиск и сопоставление по смыслу, а не только по форме слов.

Основные технологии и модели:

- Word2Vec, GloVe — методы построения статических векторов слов,

- BERT, RoBERTa, DistilBERT — контекстные трансформерные модели, учитывающие положение и окружение слова,
- T5, BART — модели генерации, способные переписывать, дополнять и исправлять предложения,
- Sentence-BERT (SBERT) — адаптация BERT, специально обученная для сравнения предложений по смыслу.

С помощью таких моделей запрос и предполагаемые теги кодируются в векторное пространство, и далее сравниваются с использованием метрики косинусной близости. Это позволяет системе не только понять, о чём «примерно» спрашивает пользователь, но и рекомендовать логически близкие теги или уточнения.

Преимущества:

- учитывают контекст всего предложения,
- демонстрируют высокую точность при обработке естественного языка,
- поддерживают многословные выражения и сложные семантические связи,
- способны не только исправлять, но и переформулировать запросы,
- хорошо масштабируются на задачи классификации, сопоставления и генерации.

Недостатки:

- высокая вычислительная сложность,
- зависимость от предобученных моделей,
- необходимость дополнительного обучения для специфических доменов.

Контекстные и нейросетевые модели являются современным стандартом в задачах интеллектуальной обработки текстов. Благодаря своей способности понимать и интерпретировать смысл высказываний, они позволяют добиться высокой точности при исправлении и переформулировке

запросов. В системах поиска и автокоррекции они применяются как основа для модуля семантического сопоставления и генерации альтернативных формулировок.

1.2.4 Гибридные подходы

Гибридные подходы представляют собой совмещение преимуществ различных стратегий коррекции: символьных, статистических и нейросетевых. Их основная идея заключается в том, чтобы компенсировать слабые стороны одного метода сильными сторонами другого. Такой многослойный подход особенно актуален в условиях, когда необходимо одновременно достигать высокой точности, устойчивости к шуму и разумной производительности.

На первом этапе происходит символьная коррекция. Применяется алгоритм для быстрой очистки запроса от орфографических и механических ошибок.

На втором этапе — статистическая фильтрация. Используются частотные модели, n-граммы и вероятностные оценки для определения наиболее типичных форм фраз и словосочетаний.

Последний этап — семантическая интерпретация. Преобразование запроса в эмбединг, поиск семантически наиболее близкого тега и генерация альтернативных формулировок с помощью моделей.

Преимуществами гибридного подхода являются:

- высокая точность на всех типах ошибок — от простых опечаток до семантических искажений,
- гибкость архитектуры. Каждый модуль может быть заменён, адаптирован или масштабирован независимо,
- возможность постепенной оптимизации. Внедрение кэширования, адаптация модели под домен,
- поддержка как точной коррекции, так и генерации новых запросов на основе смысла.

Типовые сценарии применения:

- интеллектуальные поисковые системы и ассистенты,
- контекстные подсказки в чат-ботах.

Гибридные методы обработки поисковых запросов сочетают в себе все подходы, описанные выше. Такой подход учитывает как орфографические, так и семантические искажения, что делает его особенно эффективным в условиях реального использования.

1.3 Выбор инструментов разработки

Выбор инструментов разработки играет ключевую роль при проектировании и реализации программного обеспечения. От правильного выбора инструментов будет зависеть производительность, скорость разработки, лёгкость сопровождения.

1.3.1 Выбор языка программирования

Основными языками для подобного круга задач на текущий момент являются Python, Java и Julia.

1.3.1.1 Python

Python занимает доминирующее положение в области обработки естественного языка (NLP) и разработки систем машинного обучения. Широкое применение обусловлено лаконичным синтаксисом, обширной экосистемой библиотек и высокой скоростью прототипирования. Среди наиболее полезных инструментов, активно применяемых в рамках проекта, можно отметить такие библиотеки, как `scikit-learn`, `spaCy`, `transformers`, `sentence-transformers`, а также встроенные средства работы с данными (`pandas`, `numpy`).

Дополнительным преимуществом Python является активное сообщество разработчиков, что упрощает поиск решений и интеграцию новых подходов. Также язык легко сочетается с другими технологиями, что

позволяет использовать Python в качестве связующего компонента в более сложных программных архитектурах.

Несмотря на указанные достоинства, язык имеет и свои ограничения. В частности, Python может проигрывать по производительности низкоуровневым языкам, таким как C++, особенно в вычислительно интенсивных задачах. Кроме того, особенности его реализации (например, наличие глобальной блокировки интерпретатора — GIL) иногда затрудняют реализацию эффективной многопоточности. Тем не менее, для целей данной работы, не требующей масштабных параллельных вычислений, эти ограничения не являются критичными.

1.3.1.2 Java

Java традиционно используется для создания надёжных, масштабируемых и производительных программных систем. Этот язык доминирует в сфере корпоративного программного обеспечения, облачных платформ, банковских и телекоммуникационных решений. В контексте разработки распределённых систем и веб-приложений Java остаётся одним из наиболее устойчивых и поддерживаемых языков.

Среди основных достоинств Java можно выделить его высокую производительность, обусловленную компиляцией в байт-код и последующим исполнением на виртуальной машине, а также хорошо развитую поддержку многозадачности. Это делает язык удобным для построения систем, обрабатывающих большое количество параллельных пользовательских запросов.

Однако в задачах, связанных с обработкой естественного языка и машинным обучением, Java уступает Python. Хотя существуют библиотеки вроде DeepLearning4j или Stanford NLP, их функциональность и гибкость, как правило, ниже по сравнению с Python-аналогами. Это связано как с меньшей активностью сообщества в данном направлении, так и с ограниченным

выбором предобученных моделей и инструментов быстрого прототипирования.

Таким образом, несмотря на архитектурную зрелость и высокую производительность Java, её использование в проектах, связанных с NLP, чаще всего требует дополнительных усилий по интеграции и настройке, что делает её менее предпочтительной для быстрого исследования и внедрения современных языковых моделей.

1.3.1.3 Julia

Julia — сравнительно молодой язык программирования, ориентированный на высокопроизводительные научные и численные вычисления. Он сочетает в себе лаконичность синтаксиса, характерную для высокоуровневых языков, с производительностью, сопоставимой с C и Fortran. Благодаря этим качествам Julia активно применяется в инженерных задачах, статистике и численном моделировании.

Среди ключевых преимуществ Julia следует отметить её высокую скорость выполнения программ и удобство при работе с большими объёмами данных. Встроенная поддержка параллелизма, возможность прямого обращения к C-библиотекам и современная система типов делают язык удобным для научных расчётов.

Тем не менее, в области обработки естественного языка и машинного обучения Julia пока значительно уступает более зрелым решениям, таким как Python. Количество специализированных библиотек и обучающих материалов ограничено, а сообщества пользователей и разработчиков ещё находятся в стадии становления. Это усложняет реализацию и интеграцию современных NLP-моделей, особенно в прикладных проектах. Все эти ограничения делают Julia непрактичным выбором для промышленного NLP.

1.3.1.4. Сравнительный анализ

Результаты анализа представлены в таблице 1.

Таблица 1. Сравнение языков программирования

Язык программирования	Достоинства	Недостатки	Примечания
Python	Популярен, простой синтаксис, широкая экосистема NLP и ИИ	Производительность ниже, чем у C++ или Java	Подходит для прототипирования, большая поддержка библиотек
Java	Высокая производительность, масштабируемость	Меньше библиотек для NLP и ИИ, сложный синтаксис	Менее гибок для быстрого прототипирования
Julia	Высокая производительность, хорош для научных вычислений	Меньше популярна, ограниченная экосистема для NLP	Не так популярна в области NLP

Учитывая направленность проекта на обработку текстов, семантический анализ, генерацию альтернативных формулировок и использование современных языковых моделей, Python был выбран в качестве основного языка разработки. Это объясняется:

- широкой поддержкой необходимых библиотек и моделей,
- простотой интеграции компонентов,
- богатым выбором инструментов анализа и визуализации,
- высокой скоростью тестирования.

Python обеспечивает необходимый баланс между удобством, функциональностью и скоростью разработки, что делает его оптимальным выбором для задач, связанных с построением интеллектуальных систем в области обработки естественного языка.

1.3.2 Выбор среды разработки

После определения языка программирования ключевым этапом становится выбор подходящей среды разработки. От этого зависит удобство написания и отладки кода, скорость выполнения и интеграция с необходимыми библиотеками. Проект ориентирован на работу с большими массивами текстовых данных и использует ресурсоёмкие модели (например, Sentence-BERT и T5), поэтому особое внимание нужно уделить поддержке вычислительных ускорителей (GPU/TPU), гибкости среды и удобству взаимодействия с внешними источниками данных.

1.3.2.1. Google Colab

Google Colab — это облачная среда выполнения Python-кода, предоставляемая компанией Google. Она основана на интерфейсе Jupyter Notebook и не требует локальной установки. Colab предоставляет бесплатный доступ к графическим процессорам (GPU) и тензорным ускорителям (TPU), что делает её особенно привлекательной для проектов в области машинного обучения.

Интерфейс Google Colab представлен на рисунке 1.

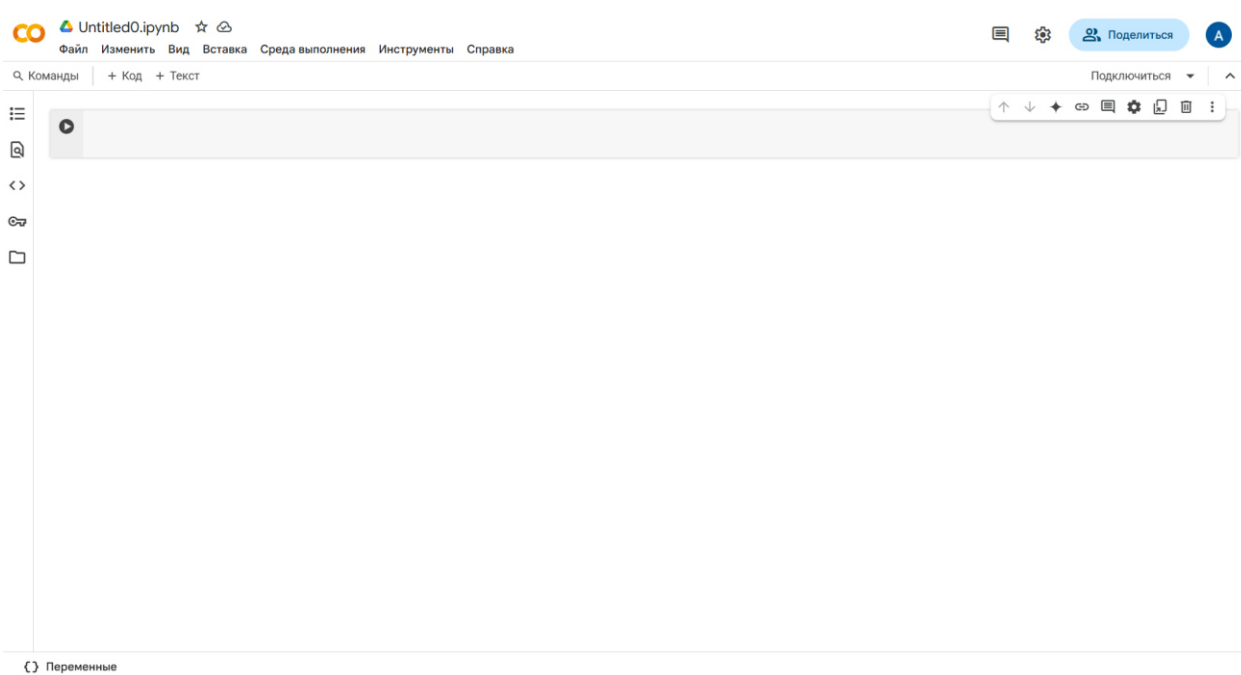


Рисунок 1 – Интерфейс Google Colab

Преимущества:

- Бесплатный доступ к GPU/TPU: возможность выполнять ресурсоёмкие вычисления без необходимости локального оборудования.
- Интеграция с Google Drive: упрощает хранение, доступ и обмен файлами, особенно при командной разработке.
- Мгновенная работа из браузера: не требует установки среды, что ускоряет старт и снижает зависимость от конфигурации локального компьютера.
- Поддержка популярных библиотек: большинство нужных инструментов для NLP и машинного обучения уже предустановлены (transformers, torch, pandas и др.).
- Возможность делиться ноутбуками: удобно при совместной работе и демонстрации результатов.

Недостатки:

- Ограничения по времени работы: бесплатные сессии ограничены 12 часами, после чего окружение сбрасывается.
- Ограниченный контроль над окружением: нельзя установить специфические версии библиотек на уровне системы, как в локальной среде.
- Зависимость от интернет-соединения: все операции выполняются в облаке, что делает среду чувствительной к нестабильной сети.

1.3.2.2. Jupyter Notebook

Jupyter Notebook представляет собой удобную интерактивную среду для написания и выполнения кода, которая особенно хорошо подходит для задач анализа данных, машинного обучения и быстрого прототипирования. Одной из отличительных особенностей среды является возможность сочетать исполняемый код, текстовые пояснения и графики в одном документе, что делает её популярной как среди исследователей, так и среди прикладных разработчиков.

Рабочее пространство организовано в виде ячеек, каждая из которых может содержать код, текст на языке Markdown или результат выполнения. Такой подход позволяет пошагово отлаживать алгоритмы, документировать эксперименты и оперативно вносить изменения. Пример интерфейса Jupyter Notebook приведён на рисунке 2.

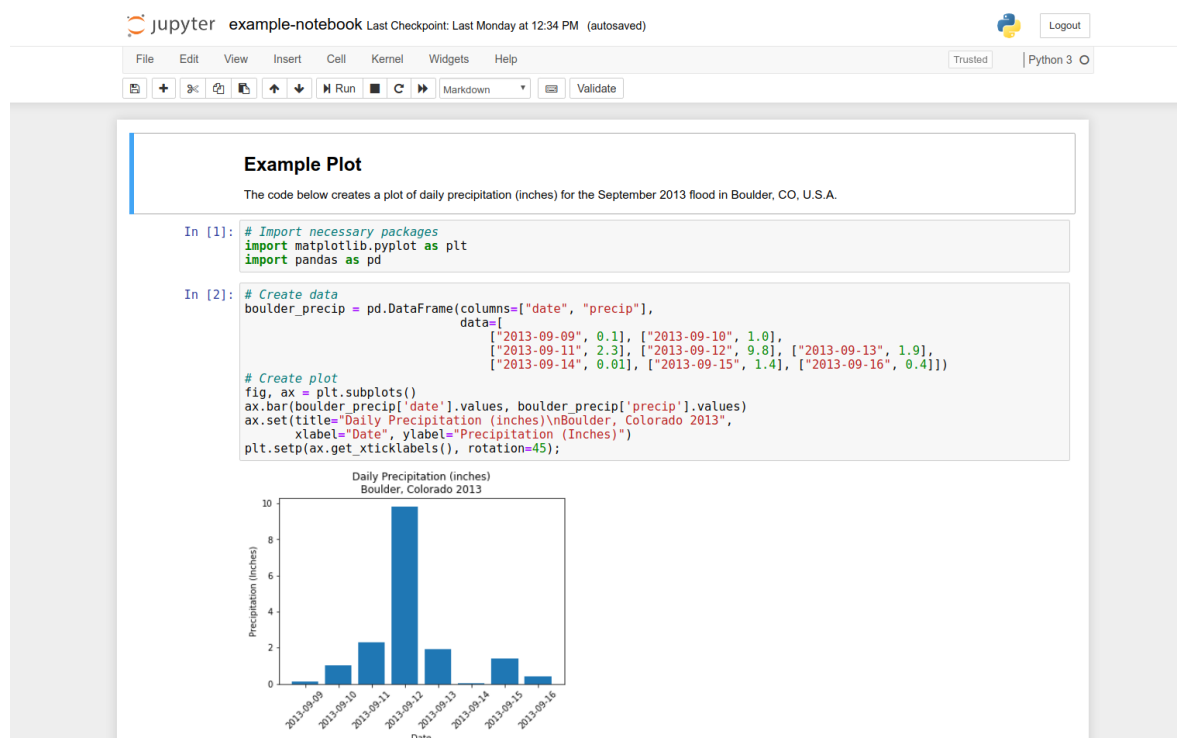


Рисунок 2 – Интерфейс Jupyter Notebook

К числу основных преимуществ Jupyter Notebook можно отнести:

- интерактивность: среда позволяет выполнять отдельные фрагменты кода, что удобно при исследовательской работе и пошаговом тестировании логики,
- гибкость настройки: поддержка большого числа библиотек и расширений делает среду универсальной для большинства задач в области анализа данных и машинного обучения,
- локальная работа: полная автономия в настройке окружения и зависимостей (с использованием `venv` или `conda`) позволяет адаптировать среду под конкретные потребности проекта.

Однако у Jupyter есть и определённые ограничения:

– отсутствие встроенной поддержки GPU: при запуске на локальной машине без дополнительных настроек невозможно использовать аппаратное ускорение, что может замедлять обработку больших моделей,

– ограниченные возможности облачного хранения: в отличие от облачных решений, таких как Google Colab, Jupyter не предлагает из коробки синхронизацию с облаком и требует ручной настройки для резервного копирования данных.

1.3.2.3. PyCharm

PyCharm

PyCharm — это интегрированная среда разработки, созданная компанией JetBrains, и ориентированная на профессиональную разработку на языке Python. Среда предоставляет широкий набор инструментов: статический анализ кода, встроенные средства отладки, профилирования, тестирования, а также поддержку систем контроля версий. Благодаря такому функционалу PyCharm особенно хорошо подходит для создания полноценных программных продуктов, в том числе с применением технологий искусственного интеллекта и анализа данных. Интерфейс среды представлен на рисунке 3.

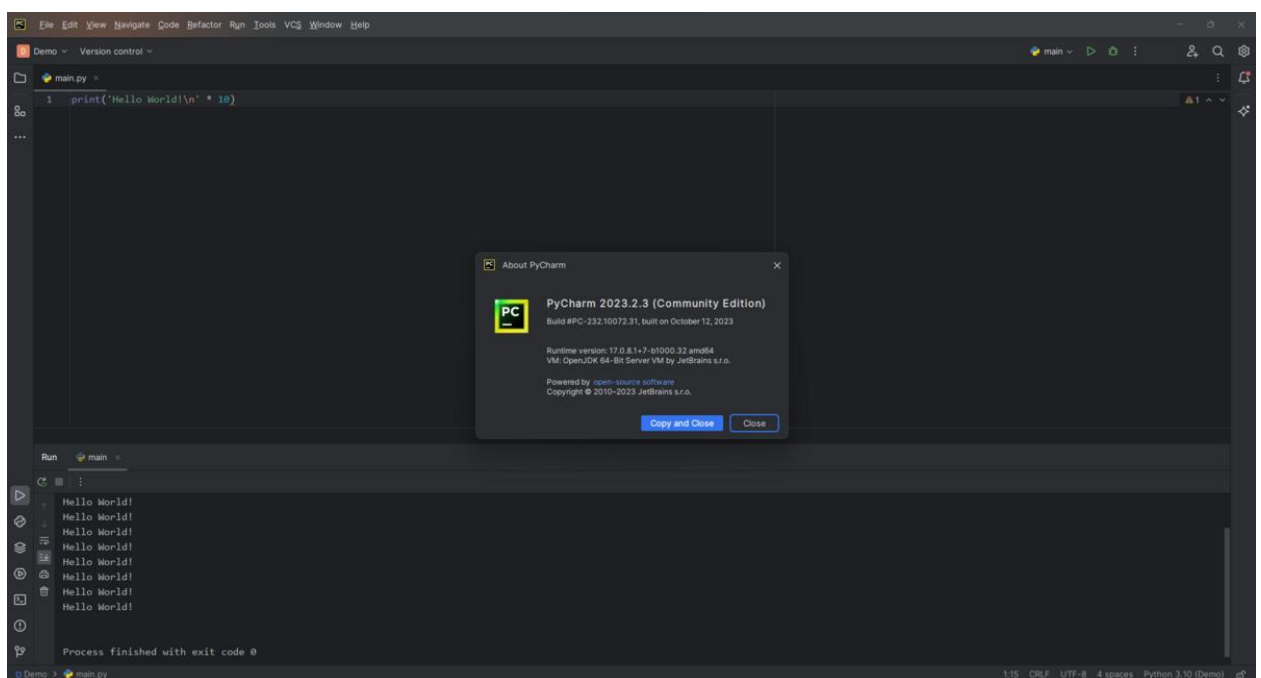


Рисунок 3 – Интерфейс PyCharm

К ключевым преимуществам среды можно отнести:

- удобная отладка и диагностика: встроенные средства позволяют отслеживать значения переменных, пошагово выполнять код, анализировать поведение программы и выявлять узкие места,
- гибкость в настройке проекта: IDE позволяет легко настраивать виртуальные окружения, указывать версии Python, подключать внешние библиотеки и управлять зависимостями,
- интеграция с инструментами промышленной разработки: поддержка Git, Docker, систем сборки и CI/CD позволяет использовать PyCharm в командной работе и в крупных проектах с полным жизненным циклом программного обеспечения.

Тем не менее, у среды есть и определённые ограничения:

- отсутствие встроенной поддержки GPU и TPU: при работе с задачами, требующими аппаратного ускорения (например, обучение нейросетей), потребуется ручная настройка окружения,
- платная версия: наиболее полный функционал доступен только в коммерческом издании, в то время как базовые возможности предоставляются в бесплатной Community-версии,
- меньшая гибкость для быстрого прототипирования: по сравнению с такими средами, как Google Colab или Jupyter Notebook, PyCharm не так удобен для интерактивной работы и тестирования отдельных блоков кода.

В целом, PyCharm — это мощная и надёжная среда, подходящая для реализации сложных проектов, требующих высокого уровня контроля, структурирования и сопровождения кода. В рамках данной разработки PyCharm может использоваться как вспомогательная среда для локальной отладки и настройки компонентов системы.

1.3.2.4. Сравнительный анализ

Результаты анализа представлены в таблице 2.

Таблица 2. Сравнение сред разработки

Среда разработки	Достоинства	Недостатки	Примечания
Google Colab	Бесплатный доступ к GPU, облачная среда, простота использования	Ограничения по времени сессий, ограничения по ресурсам	Подходит для быстрого прототипирования и работы с большими данными
Jupyter Notebook	Интерактивная среда для разработки, поддержка множества библиотек	Нет поддержки GPU/TPU	Подходит для локальных исследований и разработки
PyCharm	Мощная IDE с инструментами для отладки и тестирования	Нет поддержки GPU/TPU	Подходит для разработки сложных приложений и крупных проектов

Для реализации данного проекта, предполагающего работу с трансформерными моделями и большими объёмами текстовых данных, было принято решение использовать Google Colab в качестве основной среды разработки. Это обусловлено следующими факторами:

- Доступ к GPU: существенно ускоряет обучение современных нейросетевых моделей (Sentence-BERT, T5 и др.).
- Минимальные затраты на развёртывание: не требует настройки локального окружения, легко масштабируется.
- Интеграция с Google Drive: обеспечивает удобную структуру хранения данных и результатов экспериментов.
- Удобство совместной работы: позволяет делиться ноутбуками с коллегами и сохранять ход разработки в облаке.

Таким образом, Google Colab обеспечивает идеальный баланс между доступностью, производительностью и удобством при разработке и тестировании интеллектуальных компонентов системы обработки пользовательских запросов.

1.4 Выводы

В результате проведённого анализа предметной области были выделены основные проблемы, возникающие при формировании поисковых запросов пользователями, а также рассмотрены подходы к их автоматической корректировке. Анализ также показал, что ошибки в поисковых запросах могут быть разнородными по своей природе: от простых опечаток до семантических и контекстных искажений. Это требует гибкого и многоуровневого подхода к их обработке.

Наиболее подходящим решением для поставленной задачи является гибридный подход, сочетающий скорость и точность символьной коррекции с глубокой семантической интерпретацией, обеспечиваемой контекстными трансформерными моделями. Такой подход позволяет не только устранить поверхностные ошибки, но и интерпретировать намерение пользователя, адаптируя поведение системы под реальные сценарии взаимодействия с поисковым интерфейсом.

Также были определены ключевые инструменты разработки. На основании сравнительного анализа было принято решение использовать язык программирования Python, обладающий широким набором библиотек для обработки естественного языка и семантического анализа. В качестве среды разработки выбран Google Colabo, предоставляющая доступ к GPU и облачному хранилищу, что существенно ускоряет работу с трансформерными моделями и обеспечивает удобство тестирования и хранения данных.

2 Реализация

Перед началом разработки интеллектуального коррективщика поисковых запросов необходимо определить используемые технологии, функциональные требования, а также общую архитектуру системы. На данном этапе формируется структура приложения, логика обработки запроса, выбираются инструменты для семантического анализа и коррекции текста.

Система реализуется по модульному принципу и включает следующие компоненты:

- корректор символьных ошибок,
- преобразование запросов в эмбединги,
- семантическое сопоставление с тегами,
- генерация альтернативных формулировок.

Хотя классическая база данных не применяется, определяется структура входного датасета, его обработка и взаимосвязь между сущностями запроса и тегов.

2.1 Стек технологий, используемый для разработки

Реализация интеллектуальной системы коррекции поисковых запросов с элементами RAG-подхода потребовала выбора современных и проверенных технологий, способных обеспечить как корректную работу с текстовыми данными, так и высокую производительность при семантическом анализе. Основу проектирования составляет язык программирования Python — благодаря своей универсальности, широкому набору библиотек для обработки естественного языка и машинного обучения, а также простоте интеграции с различными инструментами.

В рамках данного проекта были использованы следующие ключевые технологии:

Обработка текста и генерация эмбедингов:

- Sentence-BERT (paraphrase-MiniLM-L6-v2) — трансформерная модель, основанная на BERT, адаптированная для вычисления

семантического сходства между текстами. Позволяет кодировать текстовые запросы и теги в многомерные векторные представления (эмбединги), которые сохраняют смысловое содержание. Используется как для семантического сопоставления, так и для поиска аналогий. Модель обладает высокой скоростью и точностью при сравнении коротких предложений.

Коррекция текста:

– SymSpell (symspellpy) — алгоритм символьной коррекции, обеспечивающий высокую скорость и масштабируемость при проверке правописания. Использует префиксное дерево и заранее рассчитанные расстояния редактирования для исправления ошибок. Позволяет производить корректировку многословных запросов с учётом контекста. Метод `lookup_compound` обеспечивает восстановление полной фразы по частям, включая слова с опечатками.

Семантическое сопоставление:

– scikit-learn — библиотека, предоставляющая инструменты для машинного обучения и анализа данных. В проекте используется для вычисления метрики косинусного сходства между эмбедингами. Позволяет определить степень семантической близости между вектором запроса и векторами тегов, необходимую для ранжирования и выбора наиболее релевантного результата.

Генерация альтернативных запросов:

– Flan-T5 (google/flan-t5-base) — языковая трансформерная модель семейства T5, дообученная на широком спектре задач. Применяется для генерации новых формулировок пользовательского запроса с опорой на найденные семантически близкие теги. Позволяет разнообразить и уточнить исходный запрос, что особенно важно в рамках подхода RAG (генерация с привлечением внешнего знания).

Работа с данными:

– pandas — библиотека для обработки структурированных данных. Используется для работы с обучающим датасетом, фильтрации и предобработки запросов и тегов, расчёта частот слов и построения словаря.

– numpy — библиотека для числовых вычислений, обеспечивающая векторизацию операций, что важно при работе с эмбедингами и матрицами сходства.

Таким образом, выбранный стек технологий обеспечивает полный цикл обработки запроса: от символьной коррекции до генерации новых смысловых формулировок, опираясь на современные методы NLP и архитектуры трансформеров. Это делает систему масштабируемой, точной и пригодной для применения в реальных поисковых системах.

2.2 Требования к разработке

Разрабатываемая система предназначена для автоматического улучшения пользовательских поисковых запросов за счёт исправления опечаток, уточнения формулировок и генерации смысловых альтернатив. Предполагается, что модуль будет использоваться в составе поисковых интерфейсов, браузеров или рекомендательных сервисов. Система должна реагировать на ввод пользователя в реальном времени и формировать исправленный и дополненные варианты запроса до передачи его на этап поиска.

Одним из ключевых требований является независимость системы от конкретного интерфейса — она должна работать как отдельный модуль, получающий на вход текст запроса и возвращающий список исправленных и альтернативных вариантов. Это обеспечит гибкость и простоту интеграции в различные платформы.

Также важным является низкое время отклика: система должна обрабатывать запросы достаточно быстро, чтобы результаты успевали обновляться динамически по мере набора текста.

Основные функциональные требования:

- исправление орфографических, символьных и опечаток в пользовательском запросе,
- формирование эмбединга на основе предобученной модели Sentence-BERT,
- подбор наиболее релевантного тега по смыслу,
- поиск расширенного набора схожих тегов,
- генерация альтернативных логически связанных формулировок запроса,
- сериализация векторов и повторное использование результатов без повторной обработки.

Нефункциональные требования:

- устойчивость к ошибочному, частично написанному или фрагментарному вводу,
- расширяемость: возможность замены модели без изменения всей архитектуры,
- независимость от постоянного подключения к интернету (после загрузки модели),
- возможность масштабирования на датасеты и словари больших объемов.

Функции программного модуля:

- ввод пользовательского запроса,
- коррекция запроса на основе словаря SymSpell,
- преобразование исправленного текста в вектор,
- вычисление косинусного сходства с предопределёнными тегами,
- ранжирование и отбор наиболее релевантных результатов,
- генерация нескольких смысловых вариантов на основе найденных тегов.

2.3 Алгоритм работы сервиса

Последовательность работы интеллектуального коррективщика поисковых запросов представляет собой строго структурированный процесс, в котором каждый модуль системы выполняет определённую задачу. В совокупности эти действия обеспечивают высокую точность и гибкость работы системы.

На первом этапе пользователь вводит текстовый запрос в поисковую строку интерфейса (например, в браузере, мобильном приложении или веб-платформе). При каждом изменении текста (например, после ввода символа или слова) на клиентской стороне формируется и отправляется HTTP-запрос на сервер, реализованный с использованием архитектурного стиля REST. Такой подход позволяет производить обработку запросов в реальном времени и поддерживает масштабируемость системы.

После получения запроса на сервере начинается его последовательная обработка. На первом этапе запрос передаётся в модуль символьной коррекции, задача которого — устранить орфографические, типографические и механические ошибки. Благодаря своей архитектуре, алгоритм SymSpell способен быстро обрабатывать как отдельные слова, так и целые фразы, что положительно сказывается на последующих этапах обработки.

Очищенный и исправленный текст поступает в модуль генерации эмбедингов. Здесь запрос преобразуется в вектор фиксированной размерности при помощи предобученной трансформерной модели Sentence-BERT. Такой эмбединг отражает смысловое содержание фразы и позволяет проводить семантическое сравнение текстов.

Полученный вектор далее передаётся в модуль семантического сопоставления. На этом этапе производится сравнение эмбединга запроса с эмбедингами всех доступных тегов на основе метрики косинусного сходства. В результате выбирается наиболее релевантный тег — то есть тот, который по смыслу ближе всего соответствует исходному запросу. Этот тег

затем используется в качестве отправной точки для дальнейшего анализа и генерации альтернативных формулировок.

Чтобы расширить охват и контекст поиска, основной тег используется в модуле поиска похожих тегов. Здесь формируется список тегов, семантически близких к основному, с дополнительной фильтрацией по релевантности. Эти данные позволяют построить более широкую семантическую рамку для анализа и генерации новых формулировок.

На заключительном этапе используется модуль генерации альтернативных запросов. Используя исходный запрос и набор семантически близких тегов, языковая модель (на основе трансформеров) — в частности, Flan-T5 от Google — формирует несколько альтернативных формулировок, отражающих возможные уточнения или переформулировки исходного запроса. Это обеспечивает гибкость в интерпретации пользовательского ввода и повышает шансы получения релевантных результатов в поисковой системе.

Итогом работы системы являются следующие элементы:

- исправленный и очищенный вариант исходного запроса,
- определённый основной тег, отражающий смысл запроса,
- список дополнительных тегов, близких по смыслу,
- сгенерированные альтернативные формулировки, подходящие для уточнённого поиска.

Алгоритм реализует полноценный RAG-подход, сочетающий извлечение релевантной информации и генерацию новых форм на основе семантического анализа. Это позволяет строить интеллектуальные интерфейсы взаимодействия с пользователем, устойчивые к ошибкам и способные к контекстному уточнению ввода.

Диаграмма последовательности представлена на рисунке 4.

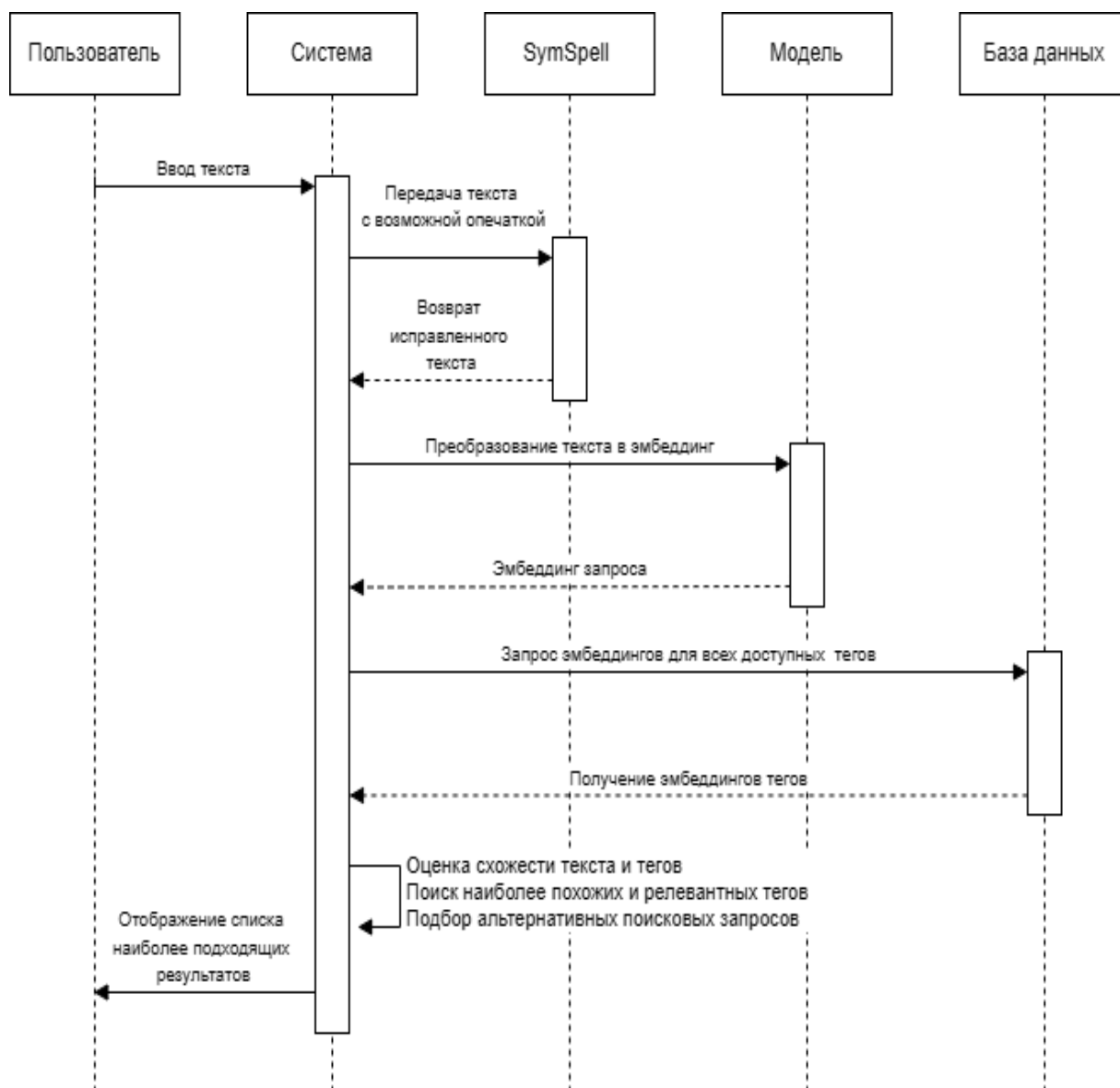


Рисунок 4 – UML-диаграмма последовательности

2.4 Описание структуры и подготовки датасета

Для построения системы интеллектуальной обработки поисковых запросов использовался датасет, содержащий текстовые пары «поисковый запрос — тег», а также числовую оценку релевантности (Relevance).

Датасет представлен в формате CSV и включает как текстовые, так и числовые характеристики, используемые на всех этапах: от коррекции и преобразования запросов до ранжирования и генерации альтернативных формулировок.

Структура датасета и описание полей представлены в таблице 3.

Таблица 3. Структура датасета

Поле	Тип данных	Описание	Пример данных
id	Целое число	Уникальный идентификатор записи	17
queries	Строка	Поисковый запрос	how to bake soft cookies
tags	Строка	Один или несколько тегов, связанных с данным запросом	baking soft cookies
TFS	Число	Оценка важности тега по частоте его использования	42.0
QLS	Число	Оценка длины запроса	1.0
SSS	Число	Семантическая близость к тематически значимым шаблонам	0.86
Relevance	Число	Итоговая оценка релевантности (в диапазоне от 0 до 1)	0.91

Для более точной оценки качества запроса и релевантности тегов в датасете дополнительно рассчитываются следующие числовые показатели:

- Tag Frequency Score (TFS) — отражает значимость тега на основе его частоты в датасете. Чем чаще тег используется, тем выше его оценка.

- Query Length Score (QLS) — оценивает длину поискового запроса. Оптимальной длиной запроса принято считать от 5 до 10 слов. Запросы вне этого диапазона получают пониженную оценку.

- Semantic Similarity Score (SSS) — характеризует смысловую близость запроса к набору часто встречающихся и тематически значимых поисковых фраз. Эта метрика позволяет учитывать направленность запроса без использования ручной аннотации.

Поле Relevance рассчитывается на основе взвешенного объединения трёх вышеуказанных показателей.

Итоговая формула включает индивидуальные веса и нормализацию результата в диапазоне от 0 до 1 с использованием метода MinMax-масштабирования:

$$Relevance = w_1 * TFS + w_2 * QLS + w_3 * SSS,$$

где w_1 , w_2 , w_3 — веса, определённые эмпирическим путём.

Подготовленный датасет используется на всех ключевых этапах реализации системы:

- построение словаря в модуле символьной коррекции,
- генерация эмбеддингов запросов и тегов,
- оценка степени релевантности и поиска ближайших тегов,
- анализ точности и полноты обработки пользовательского ввода.

2.5 Взаимодействие модулей системы

Разрабатываемая система представляет собой последовательную цепочку взаимосвязанных программных модулей, каждый из которых выполняет строго определённую функцию в процессе обработки пользовательского поискового запроса.

Взаимодействие между модулями построено по линейной схеме: каждый модуль принимает данные от предыдущего и передаёт результат следующему, что позволяет реализовать чёткий и предсказуемый порядок выполнения операций.

Диаграмма развертывания представлена на рисунке 5.

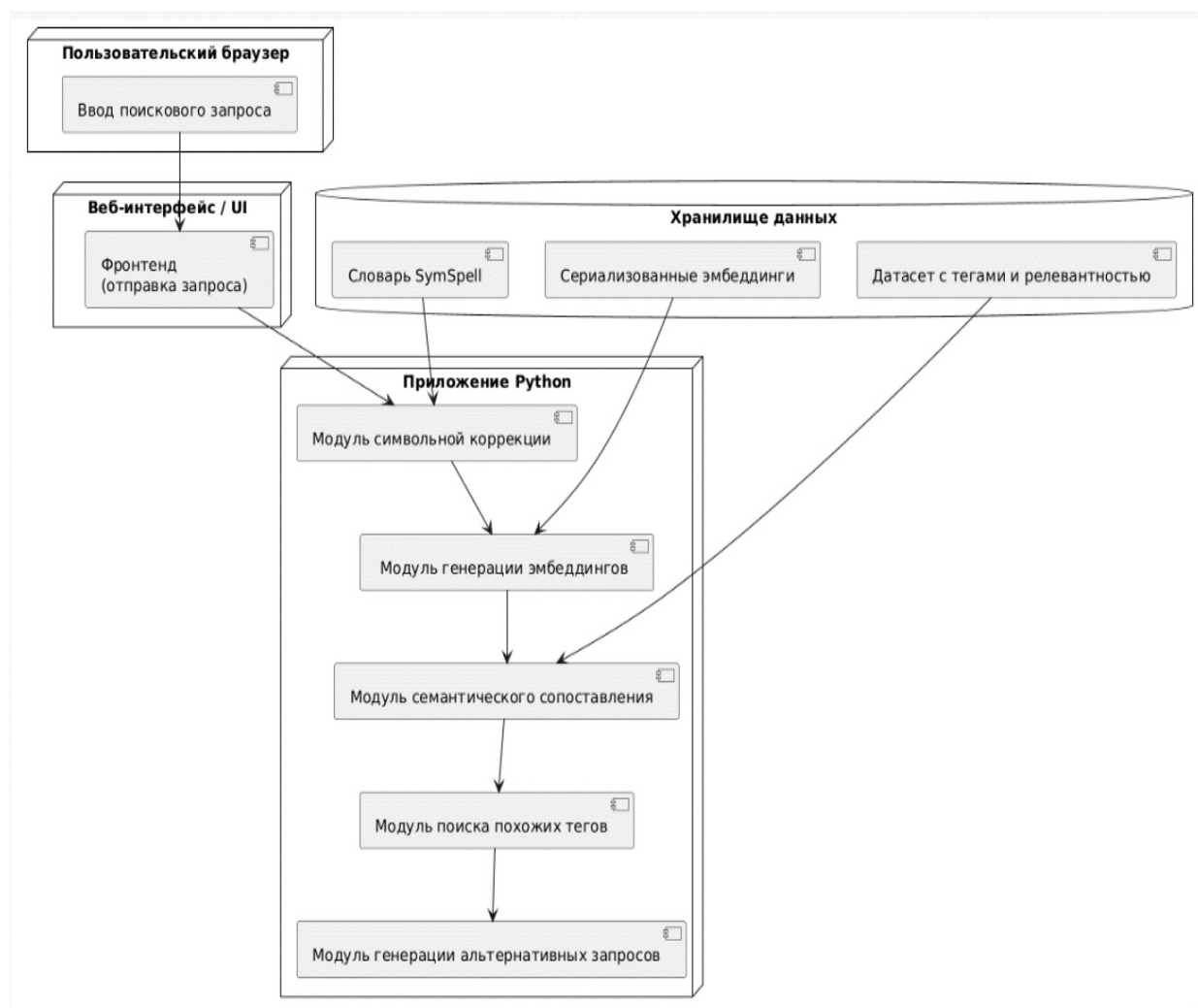


Рисунок 5 – Диаграмма развертывания

Обработка начинается с ввода текста запроса пользователем. Полученный текст передаётся в модуль символьной коррекции, где устраняются орфографические и типографические ошибки на основе словаря и алгоритма SymSpell. После этого откорректированный текст поступает в модуль эмбедингов, где преобразуется в вектор фиксированной размерности. Данный вектор отражает смысловую составляющую запроса и используется далее для сопоставления с ранее закодированными тегами.

Модуль семантического сопоставления получает эмбединг запроса и сравнивает его с эмбедингами всех доступных тегов. На основании метрики косинусного сходства определяется наиболее релевантный тег, отражающий тематику запроса. Затем формируется список наиболее близких по смыслу

тегов — это расширяет спектр возможных интерпретаций пользовательского ввода и подготавливает данные для следующего этапа.

Завершающая стадия обработки осуществляется модулем генерации альтернативных формулировок. На основе исправленного запроса и выявленных семантических тегов данный компонент системы синтезирует набор вариативных формулировок, которые не только дублируют исходный запрос в исправленном виде, но и предлагают его содержательные уточнения и расширения, потенциально повышающие релевантность конечных результатов поиска.

Ключевой особенностью данной архитектуры является принцип слабой связанности компонентов, позволяющий модифицировать и заменять отдельные модули без необходимости перепроектирования всей системы. Такое решение обеспечивает гибкость системы и возможность её адаптации к различным предметным областям и специфическим требованиям обработки текстовых данных.

2.6 Реализация модулей системы

Разработка системы выполнена по модульной архитектуре, где каждый компонент отвечает за отдельную задачу в цепочке обработки поискового запроса. Данный подход обладает рядом значимых преимуществ, включая:

- высокую степень гибкости при адаптации к различным типам текстовых данных,
- возможность масштабирования и расширения функциональных возможностей,
- удобство верификации и тестирования алгоритмов.

2.6.1 Модуль символьной коррекции

Модуль символьной коррекции является первой стадией обработки пользовательского поискового запроса и отвечает за устранение орфографических, механических и случайных ошибок, часто возникающих при вводе с клавиатуры. Его основная задача — преобразовать искажённый

текст в максимально близкую к исходной форму, сохраняя при этом общий смысл запроса.

Для реализации данного функционала используется библиотека SymSpell, основанная на алгоритме символьной коррекции с префиксным деревом и частотным словарём. В отличие от традиционного подхода на основе расстояния Левенштейна, SymSpell применяет предварительную индексацию возможных корректных слов, что позволяет достигать значительного ускорения при поиске исправлений. Это делает модуль пригодным для работы в условиях реального времени и с большим объёмом запросов.

Перед использованием модуля необходимо сформировать словарь из корпуса данных, включающего все уникальные слова из запросов и тегов, с указанием их частотности. Этот словарь записывается в текстовый файл и используется SymSpell в качестве справочника для поиска наиболее вероятных вариантов исправления.

Коррекция выполняется методом `lookup_compound`, который обрабатывает всю строку целиком и способен учитывать не только отдельные слова, но и их контекст. Это особенно важно при исправлении многословных фраз, где корректность одной части зависит от другой. В случае, если подходящий вариант найден, возвращается исправленный запрос. Если же подходящего исправления не найдено, запрос возвращается в исходном виде.

Преимуществом реализации на базе SymSpell является высокая точность и скорость обработки, а также устойчивость к наиболее распространённым видам ошибок: удалению, вставке, перестановке и замене символов. Кроме того, метод корректно работает с фразами, в которых встречаются сразу несколько искажённых слов.

Листинг:

```
from symspellpy.symspellpy import SymSpell
def correct_query_with_symspell(query):
```

```

        sym_spell = SymSpell(max_dictionary_edit_distance=2,
                               prefix_length=7)
        sym_spell.load_dictionary("symspell_dictionary.txt",
                                  term_index=0, count_index=1)
        suggestions = sym_spell.lookup_compound(query,
                                                  max_edit_distance=2)
        return suggestions[0].term if suggestions else query

```

2.6.2 Модуль генерации эмбедингов

Модуль символьной коррекции является первой стадией обработки пользовательского поискового запроса и отвечает за устранение орфографических, механических и случайных ошибок, часто возникающих при вводе с клавиатуры. Его основная задача — преобразовать искажённый текст в максимально близкую к исходной форму, сохраняя при этом общий смысл запроса.

Для реализации данного функционала используется библиотека SymSpell, основанная на алгоритме символьной коррекции с префиксным деревом и частотным словарём. В отличие от традиционного подхода на основе расстояния Левенштейна, SymSpell применяет предварительную индексацию возможных корректных слов, что позволяет достигать значительного ускорения при поиске исправлений. Это делает модуль пригодным для работы в условиях реального времени и с большим объёмом запросов.

Перед использованием модуля необходимо сформировать словарь из корпуса данных, включающего все уникальные слова из запросов и тегов, с указанием их частотности. Этот словарь записывается в текстовый файл и используется SymSpell в качестве справочника для поиска наиболее вероятных вариантов исправления.

Коррекция выполняется методом `lookup_compound`, который обрабатывает всю строку целиком и способен учитывать не только отдельные слова, но и их контекст. Это особенно важно при исправлении

многословных фраз, где корректность одной части зависит от другой. В случае, если подходящий вариант найден, возвращается исправленный запрос. Если же подходящего исправления не найдено, запрос возвращается в исходном виде.

Преимуществом реализации на базе SymSpell является высокая точность и скорость обработки, а также устойчивость к наиболее распространённым видам ошибок: удалению, вставке, перестановке и замене символов. Кроме того, метод корректно работает с фразами, в которых встречаются сразу несколько искажённых слов.

Листинг:

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
query_embeddings = model.encode(df['queries'].tolist(),
show_progress_bar=True)
tag_embeddings = model.encode(df['tags'].tolist(),
show_progress_bar=True)
```

2.6.3 Модуль семантического сопоставления

После получения эмбединга запроса следующим шагом является определение наиболее подходящего тега — то есть выбор такой текстовой метки, которая по смыслу наиболее близка к запросу пользователя. Для этого реализован модуль семантического сопоставления, который сравнивает эмбединг запроса с эмбедингами всех тегов, предварительно сгенерированных и сохранённых в векторном пространстве.

В основе сравнения лежит метрика косинусного сходства, широко используемая в задачах анализа текстов. Она позволяет оценить угол между двумя векторами: чем ближе они друг к другу, тем выше значение метрики и тем выше смысловое соответствие между запросом и тегом.

Сравнение выполняется векторно, после чего выбирается тег с наибольшим значением сходства. Этот тег считается наиболее релевантным для текущего запроса и используется как якорный в дальнейших этапах обработки: поиске похожих тегов, генерации альтернативных запросов и расширении смыслового поля исходной фразы.

Листинг:

```
from sklearn.metrics.pairwise import cosine_similarity

def find_most_relevant_tag(query_embedding, tag_embeddings,
tags):
    similarities = cosine_similarity(query_embedding,
tag_embeddings)
    best_index = similarities[0].argmax()
    return tags[best_index]
```

Модуль обеспечивает основу для построения интеллектуального интерфейса, способного подбирать тематические ключевые слова к произвольному пользовательскому вводу. Его точность напрямую влияет на релевантность результатов и успех последующих этапов генерации смысловых расширений.

2.6.4 Модуль поиска похожих тегов

После определения основного тега, наиболее соответствующего смыслу пользовательского запроса, система расширяет контекст с помощью модуля поиска похожих тегов. Этот компонент необходим для формирования тематических расширений — ключевой части RAG-подхода. Похожие теги впоследствии используются при генерации уточнённых альтернативных запросов, позволяя системе предлагать пользователю дополнительные формулировки, близкие по тематике.

Поиск осуществляется в эмбединговом пространстве. Эмбединг основного тега сравнивается с эмбедингами всех остальных тегов с использованием метрики косинусного сходства. Таким образом формируется шкала семантической близости между тегами. Однако, чтобы исключить

нерелевантные или слабо информативные совпадения, дополнительно вводится фильтрация по значению релевантности. В итоговый список включаются только те теги, которые имеют как высокую схожесть с основным, так и превосходят его по значению поля Relevance — это позволяет повысить значимость результатов.

Если среди подходящих тегов встречаются дубликаты, модуль сохраняет только наиболее релевантные из них. Результатом работы является отсортированный список тегов, упорядоченных по степени семантической близости и значимости.

Листинг:

```
def find_similar_tags(best_tag, best_tag_embedding,
best_relevance, top_n=10):
    tag_similarities = cosine_similarity([best_tag_embedding], tag_embeddings)[0]
    unique_tags = {}
    for idx, (tag, relevance) in enumerate(zip(df['tags'], df['Relevance'])):
        if tag == best_tag or relevance <= best_relevance:
            continue
        if tag not in unique_tags:
            unique_tags[tag] = (tag_similarities[idx], relevance)
        else:
            existing_similarity, existing_relevance = unique_tags[tag]
            if existing_relevance < relevance:
                unique_tags[tag] = (tag_similarities[idx], relevance)
    similar_tags = [(tag, similarity, relevance) for tag, (similarity, relevance) in unique_tags.items()]
    similar_tags.sort(key=lambda x: (-x[1], -x[2]))
    return similar_tags[:top_n]
```

Данный модуль расширяет функциональность системы, создавая основу для генерации разнообразных, но тематически согласованных формулировок поискового запроса, что делает его неотъемлемым компонентом RAG-подхода в данной реализации.

2.6.5 Модуль генерации альтернативных запросов

Модуль генерации альтернативных запросов отвечает за расширение исходного пользовательского ввода за счёт формирования уточнённых и переформулированных вариантов запросов. Его основная цель — помочь пользователю в формулировке более точных или информативных запросов на основе семантически близких тем (тегов), извлечённых из набора релевантных данных.

Работа модуля построена на архитектурном RAG -подходе, сочетающем два этапа: семантическое извлечение информации и генерацию текста на основе полученного контекста. На первом этапе система определяет наиболее релевантный тег к запросу, а затем извлекает список близких по смыслу тегов с помощью косинусного сходства эмбедингов. Эти теги служат контекстом для генеративной языковой модели.

Для генерации формулировок используется языковая модель семейства T5 — в частности, предобученная модель "google/flan-t5-base", отличающаяся способностью следовать инструкциям в текстовом виде. Генерация осуществляется с помощью текстового prompt'a, который включает исходный запрос и перечень наиболее релевантных тегов.

Пример prompt'a

Original query: "how to bake soft cookies"

Related keywords: baking flour cookies, baking cookies texture, baking sugar cookies, baking butter cookies, baking chewy cookies

Instruction: Rewrite the query to include at least two of the listed keywords, while keeping the result short, relevant, and different from the original.

Таким образом, модель получает как цель генерации, так и контекст, который помогает ей строить логически и тематически связанные запросы. В ходе генерации применяются следующие параметры:

- `do_sample=True` — используется стохастическая генерация,
- `top_k=50, top_p=0.95` — вероятностное сужение выборки,
- `temperature=0.8` — баланс между креативностью и точностью,
- `repetition_penalty=1.2` и `no_repeat_ngram_size=3` — предотвращение заикливания и повторов.

Генерация выполняется в несколько итераций (по умолчанию 5), после чего результаты фильтруются: исключаются дублирующиеся формулировки и слишком близкие к оригиналу по тексту. Конечной целью является формирование набора коротких, логичных и лексически разнообразных альтернатив, которые расширяют смысловое пространство запроса и повышают вероятность получения релевантных результатов при последующем поиске.

Листинг:

```
from transformers import T5Tokenizer,
T5ForConditionalGeneration

tokenizer = T5Tokenizer.from_pretrained("google/flan-t5-
base")
model_t5 = T5ForConditionalGeneration.from_pretrained("google/flan-t5-
base")

def generate_rag_query(original_query, similar_tags,
num_return_sequences=5):
    prompt = (
        f"Original query: \"{corrected_query}\"\\n"
        f"Related keywords: {' '.join(similar_tags[:5])}\\n"
        f"Instruction: Rewrite the query to include at least two
of the listed keywords, "
```

```

        f"while keeping the result short, relevant, and
different from the original."
    )
    inputs = tokenizer(prompt, return_tensors="pt",
max_length=128, truncation=True)

    outputs = model_t5.generate(
        **inputs,
        max_length=64,
        num_return_sequences=num_return_sequences,
        do_sample=True,
        top_k=50,
        top_p=0.95,
        temperature=0.8,
        repetition_penalty=1.2,
        no_repeat_ngram_size=3
    )

    return [tokenizer.decode(output, skip_special_tokens=True)
for output in outputs]

```

Модуль генерации альтернативных запросов особенно важен в контексте пользовательских ошибок, неполных формулировок или неточного выбора слов. Он позволяет не только исправить исходную форму, но и предложить более релевантные с точки зрения поиска варианты, потенциально раскрывающие дополнительные аспекты информационного запроса.

2.7 Выводы

В рамках данного раздела была реализована программная система интеллектуальной обработки пользовательских поисковых запросов, включающая модули коррекции, семантического анализа, сопоставления с тегами и генерации уточнённых формулировок. Архитектура построена на модульном принципе, что обеспечило гибкость, изоляцию компонентов и возможность их масштабируемой замены или повторного использования.

Функциональная часть проекта охватывает основные этапы обработки: от устранения орфографических ошибок с помощью словарного алгоритма SymSpell до преобразования текста в эмбединги с использованием модели Sentence-BERT и дальнейшего сопоставления запроса с релевантными тегами. Особое внимание уделено генерации альтернативных запросов, позволяющих адаптировать исходный ввод к различным пользовательским сценариям.

Для тестирования модулей и настройки логики использовался специализированный датасет, содержащий пары «запрос — тег» и оценку их релевантности. Эта оценка учитывает частотность тегов, длину запроса и семантическую направленность текста. Данные были нормализованы и подготовлены для использования в ключевых модулях системы.

Реализация выполнена на языке Python с использованием специализированных библиотек (symSpellpy, sentence-transformers, scikit-learn и др.). Также обеспечена сериализация промежуточных данных (эмбедингов, модели, словаря), что позволяет ускорить работу системы при повторных запусках.

В результате выполненной работы разработан программный модуль, способный функционировать как автономно, так и в составе более сложных интеллектуальных систем. Он устойчив к наиболее типичным ошибкам пользовательского ввода в рамках заявленного уровня обработки, ориентированного на повышение точности формулировки запроса и его семантической интерпретации. Возвращаемые системой результаты пригодны для последующего анализа, визуализации или интеграции в приложения, связанные с поиском и рекомендациями.

3 Методы тестирования

Система интеллектуальной обработки поисковых запросов проверяется с целью оценки корректности работы её основных функциональных компонентов. В отличие от классических бизнес-приложений, в данной системе ключевое значение имеют качество смысловой интерпретации текста и осмысленность генерируемых формулировок, что делает ручную проверку более объективной и репрезентативной.

Основное внимание в тестировании уделяется прикладному и содержательному анализу результатов, получаемых на различных этапах обработки пользовательского запроса. Автоматическое модульное тестирование функций допустимо и потенциально полезно, однако для данной категории задач оно дополняет, но не заменяет экспертную оценку релевантности и лексической приемлемости текста.

Методика испытаний включает:

- проверку поведения системы при неполном, искажённом и бессмысленном вводе,
- оценку устойчивости и повторяемости выходных данных,
- анализ соответствия тегов и логической связности альтернативных запросов.

Каждый модуль рассматривается как отдельный логический блок, функционирование которого анализируется на основании заданных сценариев. Результаты интерпретируются не только с точки зрения технической корректности, но и с учётом смысловой целостности и пригодности для дальнейшего использования в системах поиска и рекомендаций.

3.1 Проверка модуля символьной коррекции

Для проверки устойчивости модуля символьной коррекции к различным уровням искажений используется серия тестов, построенная по принципу контроля числа ошибок в запросе. В каждом тесте фиксируется

количество слов с ошибками в пределах одного запроса, а также количество ошибок, допущенных в каждом из этих слов. Далее измеряется доля успешно исправленных запросов.

Каждая группа тестов формируется отдельно и содержит одинаковое число запросов (по 100 в каждой категории), что обеспечивает объективность при сравнении результатов. Все тестовые запросы составляются на основе корректных формулировок, в которые вручную вносятся опечатки заданного типа и количества.

При этом:

- структура фраз сохраняется максимально близкой к исходной,
- слова подбираются с учётом их реального использования в запросах,
- ошибки моделируются типовыми действиями пользователя: удаление, замена, вставка или перестановка символа.

Модуль оценивается как успешно сработавший, если он возвращает откорректированный запрос, совпадающий с ожидаемой формой. Частично восстановленные запросы (например, с исправлением не всех слов) считаются неуспешными для целей статистики. Каждый тест проводится в изолированной среде без повторного обучения или вмешательства в словарь.

3.1.1 Сценарий: одна ошибка в одном слове

В данной серии тестов анализируется влияние количества слов с ошибками на успешность восстановления исходного запроса. Во всех тестовых примерах искажённые слова содержат ровно одну ошибку (удаление, замена, вставка или перестановка символа). При этом меняется лишь количество таких слов в одном запросе.

Целью является оценка устойчивости модуля к увеличению масштаба искажений в пределах запроса при сохранении лёгкой формы ошибок. Это позволяет определить, насколько чувствителен алгоритм к «распределённому шуму» и при каком уровне искажений точность коррекции начинает резко снижаться.

Проведено четыре подсерии по 100 тестов каждая:

- в первой группе искажено только одно слово,
- во второй — два слова,
- в третьей — три слова,
- в четвёртой — четыре и более слов с ошибкой.

Таблица 4 – Результаты тестирования

Количество слов с ошибками в запросе	Число тестов	Успешно исправлено	Не исправлено
1 слово	100	97	3
2 слова	100	93	7
3 слова	100	85	15
4 слова и более	100	72	28

Зависимость качества исправления от числа слов с одной ошибкой представлена на рисунке 6.

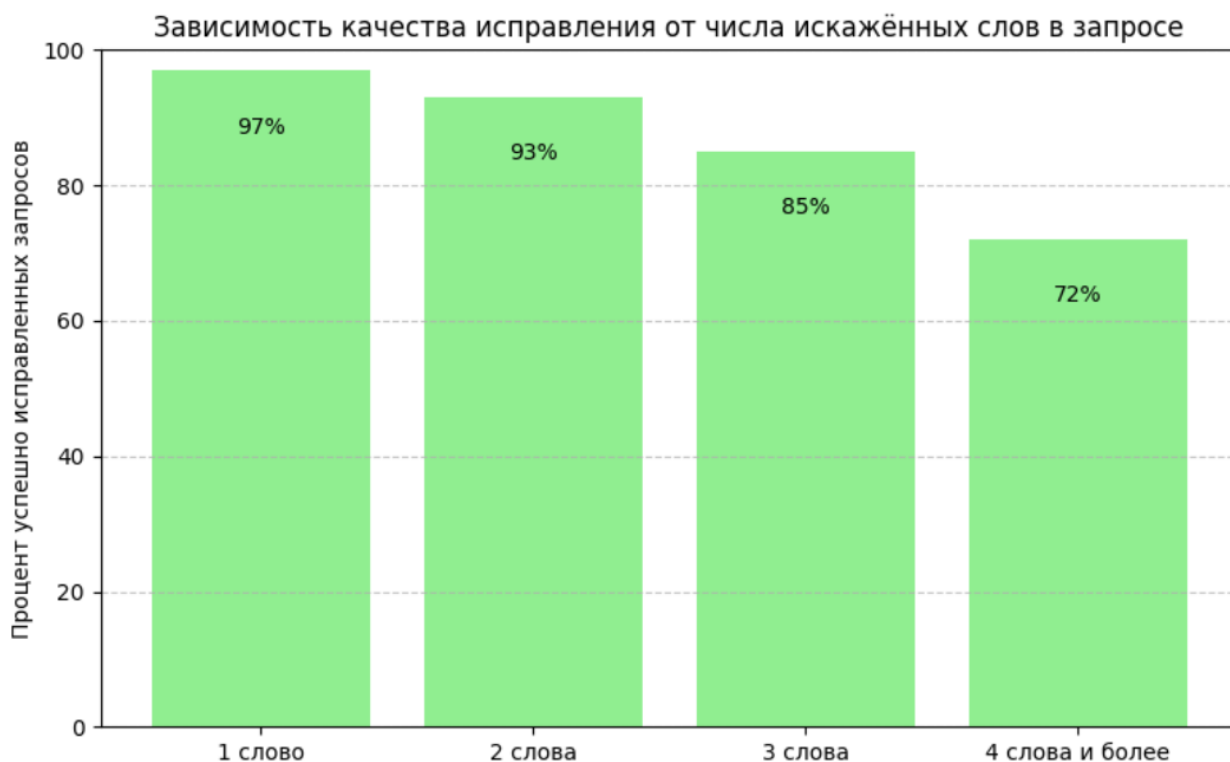


Рисунок 6 – Зависимость качества исправления от числа слов с одной ошибкой

3.1.2 Сценарий: две ошибки в одном слове

В данной серии тестов рассматривается поведение модуля символьной коррекции в условиях увеличения сложности искажений: в каждом испорченном слове допускается ровно две ошибки. При этом, как и в предыдущем сценарии, варьируется количество слов с такими искажениями в пределах одного запроса.

Таблица 5 – Результаты тестирования

Количество слов с ошибками в запросе	Число тестов	Успешно исправлено	Не исправлено
1 слово	100	94	6
2 слова	100	90	10
3 слова	100	80	20
4 слова и более	100	65	35

Зависимость качества исправления от числа слов с одной ошибкой представлена на рисунке 7.

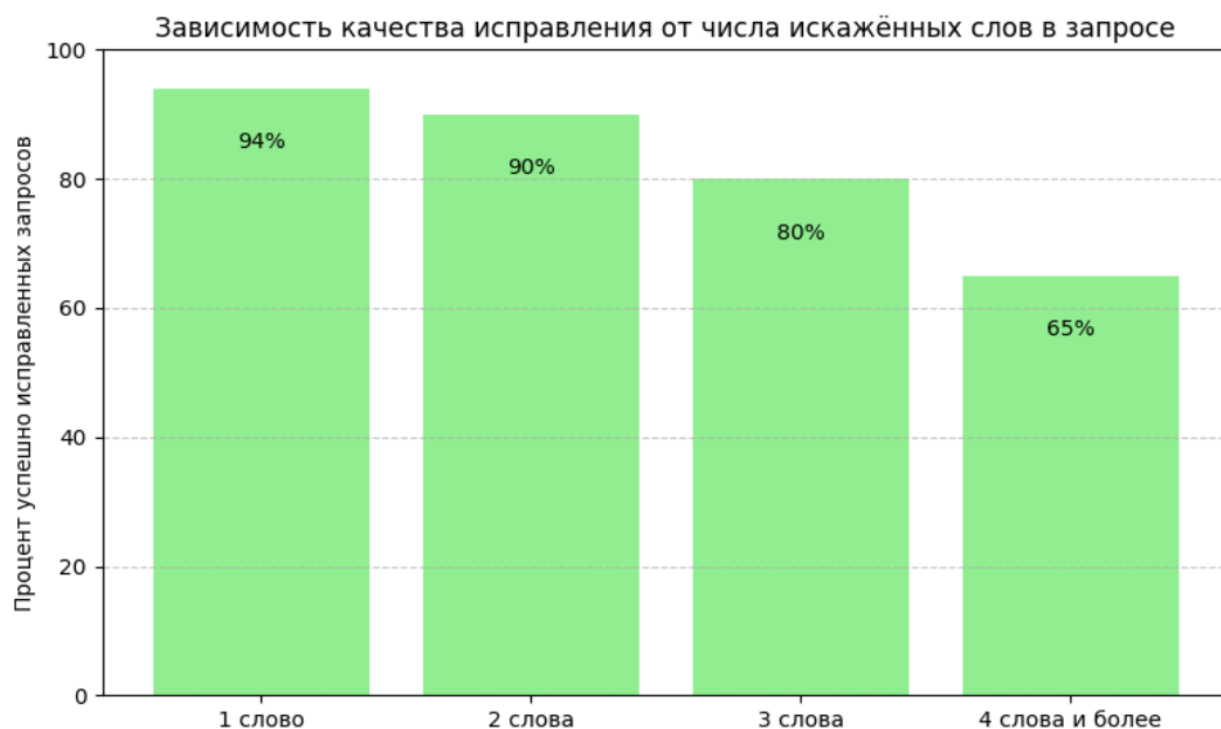


Рисунок 7 – Зависимость качества исправления от числа слов с одной ошибкой

3.1.3 Сценарий: три ошибки в одном слове

В данной серии тестов оценивается устойчивость модуля символьной коррекции к максимально усложнённым искажениям в пределах одного слова. В каждом искажённом слове намеренно вносятся три ошибки — различных типов и в разных позициях, что приближает слово к состоянию, максимально далёкому от исходной формы. Такие искажения могут возникать при повторных опечатках, сильных ошибках автозамены или наборе на чужой раскладке клавиатуры.

Таблица 6 – Результаты тестирования

Количество слов с ошибками в запросе	Число тестов	Успешно исправлено	Не исправлено
1 слово	100	87	13
2 слова	100	83	17
3 слова	100	71	29
4 слова и более	100	53	47

Зависимость качества исправления от числа слов с одной ошибкой представлена на рисунке 8.

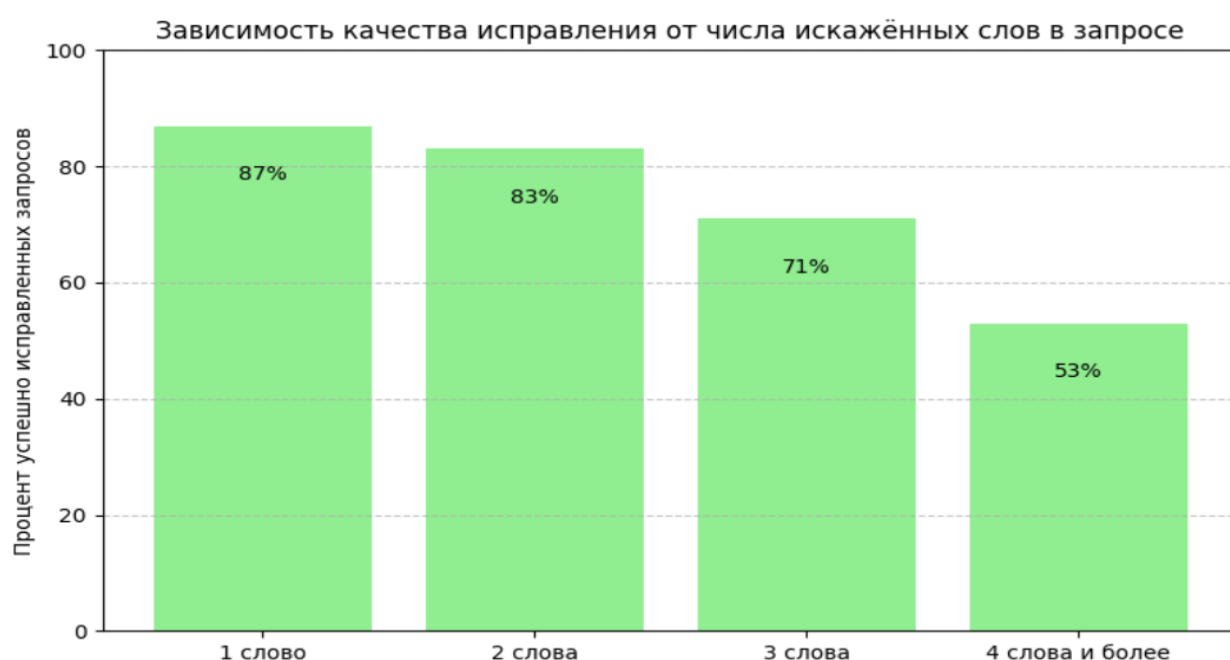


Рисунок 8 – Зависимость качества исправления от числа слов с одной ошибкой

3.1.4 Вывод по разделу

Результаты серии тестов, проведённых в рамках оценки работы модуля символьной коррекции, подтверждают его высокую эффективность при работе с пользовательскими запросами, содержащими ограниченное количество несложных ошибок. Наиболее устойчивыми оказались случаи с одной ошибкой в одном слове — при условии, что таких слов в запросе не более двух, точность восстановления приближалась к 95–97%.

По мере увеличения количества слов с ошибками наблюдается закономерное снижение процента успешно восстановленных запросов. Особенно критичным фактором оказывается увеличение количества ошибок внутри одного слова: при трёх ошибках даже в одном слове точность снижается до 87%, а при четырёх и более испорченных лексемах — до уровня менее 55%.

Таким образом, модуль показывает надёжную работу при типичных пользовательских опечатках и незначительном уровне искажений, но испытывает затруднения при более глубоких или массовых искажениях.

3.2 Проверка модуля генерации эмбеддингов

Модуль генерации эмбеддингов выполняет преобразование текстовых запросов и тегов в векторы фиксированной размерности, отражающие их смысловое содержание. В данной системе используется предобученная модель Sentence-BERT, выдающая на выходе эмбеддинг длиной 384 чисел. Эти векторы применяются для дальнейшего семантического сопоставления, поэтому важно убедиться в корректности и стабильности работы данной части системы.

В ходе тестирования проверяется, что модель корректно загружается и возвращает вектор ожидаемой размерности. Отдельное внимание уделяется повторяемости: один и тот же запрос должен приводить к идентичному вектору, если модель применяется без изменений. Также проводится

качественная проверка смысловой близости между парой запросов. Например, фразы «how to bake cookies» и «how do I make cookies» должны иметь высокое сходство, тогда как «salad» и «cake» — заметно меньше. Это подтверждает, что модель действительно различает смысловую нагрузку текстов.

В качестве дополнительного параметра оценивается производительность — среднее время генерации эмбединга одного запроса. Все эмбединги проходят контроль на стабильность диапазона значений и отсутствие сбоев при обработке коротких или редких фраз.

Ниже в таблице 7 приведены примеры запросов с различной степенью смысловой близости и рассчитанным между ними косинусным сходством.

Таблица 7 – Результаты тестирования

Запрос 1	Запрос 2	Косинусное сходство
how to bake cookies	how do I make cookies	0.91
coffee brewing techniques	methods for brewing coffee	0.87
grilling meat	cooking chicken	0.62
healthy salad	chocolate cake	0.23

Результаты показывают, что модуль работает стабильно и воспроизводимо, формируя правильные векторные представления смысловой структуры запроса. Полученные эмбединги пригодны для дальнейшего использования в задачах сопоставления, генерации тегов и анализа релевантности.

3.3 Проверка модуля семантического сопоставления

Модуль семантического сопоставления отвечает за выбор наиболее релевантного тега к пользовательскому запросу на основе сравнения эмбедингов. Для измерения степени смысловой близости используется метрика косинусного сходства между эмбедингом запроса и эмбедингами

всех доступных тегов. Тег с наибольшим значением сходства считается наилучшим кандидатом.

Тестирование проводится вручную на заранее подготовленном наборе запросов. Каждый запрос преобразуется в эмбединг с помощью Sentence-BERT и сравнивается по косинусному сходству со всеми тегами. На выходе фиксируется наиболее подходящий тег и значение сходства. Далее оценивается, насколько полученный результат логически соответствует смыслу исходного запроса.

Для оценки результата используется условная шкала: при значении сходства от 0.90 и выше считается, что найденный тег полностью совпадает по смыслу с запросом. При значениях от 0.75 до 0.89 (включительно) — тег тематически близок. Если сходство находится в пределах от 0.60 и до 0.74, это оценивается как частичное соответствие, а при значении ниже 0.60 — связь между тегом и запросом считается слабой или ее вовсе нет.

Результаты тестирования представлены в таблице 8.

Таблица 8 – Результаты тестирования

Запрос	Предсказанный тег	Косинусное сходство	Оценка соответствия
how to bake soft cookies	baking soft cookies	0.93	Полное совпадение
chocolate cake recipe	chocolate cake	0.89	Тематически близко
healthy lunch salad	vegetarian meals	0.78	Тематически близко
pasta recepie with sauce	pasta recipe	0.86	Тематически близко
grilled meat and veggies	grilling	0.65	Частично релевантно
salad and fruits combination	protein meals	0.52	Слабая связь

Модуль демонстрирует высокую точность сопоставления при корректных и даже частично искажённых запросах. При сильной

тематической расплывчатости или двузначных конструкциях качество предсказаний может снижаться, но общее поведение остаётся логичным и интерпретируемым. Полученные результаты подтверждают применимость модуля в задачах автоматического тегирования и поиска по смыслу.

3.4 Проверка модуля поиска похожих тегов

Тестирование модуля проводится на запросах, по которым был заранее определён корректный основной тег. После его определения запускается поиск похожих тегов, которые сравниваются как по тематике, так и по качеству ранжирования. Дополнительно анализируются случаи, когда основной тег имеет низкую релевантность — ожидается, что в таких ситуациях модуль сможет поднять более релевантные теги выше в списке.

Тестирование включает:

- выбор запросов с устойчивым смыслом,
- определение основного тега и его эмбединга,
- вызов метода и получение списка ближайших тегов,
- оценку логической близости возвращённых тегов к исходной теме.

Пример:

Для запроса «how to bake soft cookies» основным тегом определяется «baking soft cookies» с релевантностью 0.82. В списке похожих тегов также присутствуют следующие записи:

- «chewy cookies»,
- «cookie dough»,
- «cookie texture»,
- «baking tips» и др.

Модуль также демонстрирует правильное поведение при равных по сходству, но разных по релевантности тегах: он предпочитает более значимые. Это означает, что фильтрация по релевантности действительно работает, и список содержит семантически и при этом содержательно ценные результаты.

Результаты подтверждают, что модуль устойчив к вторичным шумам, избегает тематически далёких тегов и хорошо масштабируется при увеличении общего числа возможных кандидатов. Он может эффективно применяться для генерации смысловых расширений и интеллектуальных подсказок, особенно в составе диалоговых систем или поисковых интерфейсов.

3.5 Проверка модуля генерации альтернативных тегов

Тестирование проводится вручную по следующим шагам:

- задаётся исходный запрос (корректный или слегка искажённый),
- формируется список похожих тегов (на основе предыдущего модуля),
- выполняется генерация альтернатив,
- оценивается разнообразие, лексическая корректность и логическая применимость каждого варианта.

В среднем на 100 запросов генерируется примерно 500 уникальных альтернатив (в зависимости от количества подходящих тегов). Однако только 70-80% из них можно считать лексически корректными и логически осмысленными, так как остальные случаи либо допускали тавтологию (например, из-за составных тегов), либо получались стилистически громоздкими — такие отклонения минимальны и не влияют на общую применимость.

Примеры успешных альтернатив представлены ниже.

Исходный запрос: how to bake cookies

Альтернативы:

- how to bake cookies with chocolate,
- how to bake cookies chewy,
- how to bake cookies about cookie dough.

В результате тестирования подтверждено, что модуль способен генерировать релевантные, разнообразные и грамматически допустимые

варианты запроса. Однако в будущем стоит сделать акцент на улучшение данного модуля.

3.6 Выводы

Проведённое тестирование подтвердило функциональную состоятельность всех основных компонентов системы интеллектуальной обработки поисковых запросов. Модули показали высокую устойчивость к различным типам пользовательского ввода, в том числе к опечаткам, искажениям, вариативности формулировок и неполному контексту.

Наиболее надёжным оказался модуль символьной коррекции. При одной ошибке в каждом слове и не более двух искажённых слов в запросе его точность приближалась к 95–97%.

Модуль генерации эмбедингов с использованием Sentence-BERT продемонстрировал стабильную работу: векторы формировались воспроизводимо, соответствовали ожидаемой размерности и отражали смысловую близость между логически связанными запросами. Это обеспечило надёжную основу для дальнейшего семантического сопоставления.

Модуль подбора релевантного тега показал высокую точность. Большинство соответствий попадали в диапазон от 0.75 до 0.95 по косинусному сходству, что позволяет считать их логически релевантными. Используемая шкала оценки позволила формализовать результаты, а ручная проверка подтвердила интерпретируемость выдачи.

Поиск похожих тегов эффективно дополняет основной тег, формируя расширенный смысловой контекст. Применение дополнительной сортировки по релевантности помогает удерживать в списке наиболее значимые предложения. Выборка ближайших тегов оказалась логически согласованной, без существенных шумов.

Наименее устойчивым оказался модуль генерации альтернативных формулировок. Несмотря на общее соответствие и шаблонную структуру,

часть результатов содержала тавтологии. Только 70-80% сгенерированных запросов можно признать допустимыми, поэтому при дальнейшем развитии стоит обратить внимание на улучшение этого модуля, например подбором другой модели.

В целом система показала высокий уровень готовности к применению в задачах поиска, автоуточнения, тегирования и интеллектуального автодополнения. Выявленные слабые места позволяют обоснованно наметить направления для последующей оптимизации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения магистерской диссертации была проанализирована предметная область, связанная с обработкой пользовательских поисковых запросов и автоматической корректировкой ошибок, возникающих при их вводе поисковых запросов. Было проведено исследование основных видов ошибок, изучены современные подходы к их обработке, а также обоснован выбор стека технологий и архитектурных решений для разработки корректировщика.

В результате проведенной работы был разработан модуль, выполняющий исправление и генерацию поисковых запросов на основании семантического анализа данных, позволяя учитывать как орфографические, так и смысловые ошибки. Таким образом, можно заключить, что проведенная исследовательская работа полностью соответствует поставленным целям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Федоров, Д. Ю. Программирование на python: учебное пособие для вузов. Юрайт, 2023. — 187 с.
2. Context-sensitive Spelling Correction Using Google Web 1T 5-Gram Information [Электронный ресурс] – URL: https://www.researchgate.net/publication/224850694_Context_sensitive_Spelling_Correction_Using_Google_Web_1T_5-Gram_Information (дата обращения 15.01.2024).
3. Python Documentation [Электронный ресурс] – URL: <https://pydocs.ru/home/> (дата обращения 24.03.2024).
4. Java Documentation [Электронный ресурс] – URL: <https://docs.oracle.com/en/java/> (дата обращения 24.03.2024).
5. Julia Documentation [Электронный ресурс] – URL: <https://docs.julialang.org/en/v1/> (дата обращения 24.03.2024).
6. Google Colaboratory [Электронный ресурс] – URL: <https://colab.research.google.com/> (дата обращения 30.04.2024).
7. PyCharm Documentation [Электронный ресурс] – URL: <https://www.jetbrains.com/help/pycharm/quick-start-guide.html> (дата обращения 30.04.2024).
8. Jupyter Notebook Documentation [Электронный ресурс] – URL: <https://docs.jupyter.org/en/latest/index.html> (дата обращения 30.04.2024).
9. Basic NLP in Python [Электронный ресурс] – URL: https://seantrott.github.io/nlp_basics/ (дата обращения 10.09.2024).
10. Основы Natural Language Processing для текста [Электронный ресурс] – URL: <https://habr.com/ru/companies/Voximplant/articles/446738/> (дата обращения 14.09.2024).
11. Sentence-Transformers Documentation [Электронный ресурс] – URL: <https://www.sbert.net/> (дата обращения 27.10.2024).
12. SymSpell Algorithm Description [Электронный ресурс] – URL: <https://github.com/wolfgarbe/SymSpell/> (дата обращения 27.10.2024)..

13. Scikit-learn: Machine Learning in Python [Электронный ресурс] – URL: <https://scikit-learn.org/> (дата обращения 27.10.2024).
14. Pandas Documentation [Электронный ресурс] – URL: <https://pandas.pydata.org/> (дата обращения 17.11.2024).
15. Flan-T5 [Электронный ресурс] – URL: https://huggingface.co/docs/transformers/model_doc/flan-t5 (дата обращения 18.11.2024).
16. NumPy Documentation [Электронный ресурс] – URL: <https://numpy.org/doc/> (дата обращения 18.11.2024).

Приложение А

(Обязательное)

Корректировщик текста в поисковых запросах в браузере

с использованием ИИ

Листинг программы

Приложение А.1

Файл `symspell_corrector.py`

```
from symspellpy.symspellpy import SymSpell
from collections import Counter

def build_symspell_dictionary(df, output_path,
columns=('queries', 'tags')):
    word_counter = Counter()
    for column in columns:
        for sentence in df[column]:
            word_counter.update(sentence.split())
    with open(output_path, 'w', encoding='utf-8') as f:
        for word, count in word_counter.items():
            f.write(f"{word} {count}\n")

def correct_query_with_symspell(query, dictionary_path,
max_edit_distance=2):
    sym_spell = SymSpell(max_edit_distance, prefix_length=7)
    sym_spell.load_dictionary(dictionary_path, term_index=0,
count_index=1)

    suggestions = sym_spell.lookup_compound(query,
max_edit_distance=max_edit_distance)
    return suggestions[0].term if suggestions else query
```

Приложение A.2

Файл transformer.py

```
from sentence_transformers import SentenceTransformer
import joblib

def load_model(model_path):
    return SentenceTransformer(model_path)

def encode_texts(model, texts):
    return model.encode(texts, show_progress_bar=True)

def save_embeddings(embeddings, path):
    joblib.dump(embeddings, path)

def load_embeddings(path):
    return joblib.load(path)
```

Приложение А.3

Файл `best_tags.py`

```
from sklearn.metrics.pairwise import cosine_similarity

def recommend_best_tag(query_embedding, tag_embeddings, tags,
relevances):
    similarities = cosine_similarity(query_embedding,
tag_embeddings)
    best_index = similarities[0].argmax()
    return tags[best_index], tag_embeddings[best_index],
relevances[best_index]
```

Приложение A.4

Файл `find_tags.py`

```
from sklearn.metrics.pairwise import cosine_similarity

def find_similar_tags(best_tag, best_tag_embedding, df,
                      tag_embeddings, top_n=10):
    tag_similarities = cosine_similarity([best_tag_embedding],
                                         tag_embeddings)[0]
    unique_tags = {}
    for idx, (tag, relevance) in enumerate(zip(df['tags'],
                                              df['Relevance'])):
        if tag == best_tag or relevance <= df['Relevance'][idx]:
            continue
        if tag not in unique_tags or unique_tags[tag][1] <
relevance:
            unique_tags[tag] = (tag_similarities[idx],
relevance)
    sorted_tags = sorted(unique_tags.items(), key=lambda x: (-
x[1][0], -x[1][1]))
    return [(tag, sim, rel) for tag, (sim, rel) in
sorted_tags[:top_n]]
```

Приложение A.5

Файл `generator_query.py`

```
from transformers import T5Tokenizer, T5ForConditionalGeneration

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

def generate_rag_query(corrected_query, similar_tags,
num_return_sequences=5):
    prompt = (
        f"Original query: \"{corrected_query}\"\\n"
        f"Related keywords: {' '.join(similar_tags[:5])}\\n"
        f"Instruction: Rewrite the query to include at least two
of the listed keywords, "
        f"while keeping the result short, relevant, and
different from the original."
    )
    inputs = tokenizer(prompt, return_tensors="pt",
max_length=128, truncation=True)
    outputs = model.generate(
        **inputs,
        max_length=64,
        num_return_sequences=num_return_sequences,
        do_sample=True,
        top_k=50,
        top_p=0.95,
        temperature=0.8,
        repetition_penalty=1.2,
        no_repeat_ngram_size=3
    )
    return [tokenizer.decode(out, skip_special_tokens=True) for
out in outputs]
```

Приложение А.6

Файл main.py

```
from modules.symspell_module import correct_query_with_symspell
from modules.embedding_module import load_embeddings, load_model
from modules.tag_matching_module import
recommend_best_tag_by_query
from modules.similar_tags_module import find_similar_tags
from modules.rag_generator_module import generate_rag_query

def process_query(user_query):
    dictionary_path = '/content/drive/My
Drive/Диплом/symspell_dictionary.txt'
    tag_embeddings_path = '/content/drive/My
Drive/Диплом/tag_embeddings.pkl'
    query_embeddings_path = '/content/drive/My
Drive/Диплом/query_embeddings.pkl'
    model_path = '/content/drive/My
Drive/Диплом/sentence_transformer_model'
    dataset_path = '/content/drive/My
Drive/Диплом/dataSets/rel_dataset_v2.csv'
    df, model, tag_embeddings = load_embeddings(model_path,
tag_embeddings_path, dataset_path)
    corrected_query = correct_query_with_symspell(user_query,
dictionary_path)
    best_tag, best_tag_emb, best_relevance =
recommend_best_tag_by_query(corrected_query, model,
tag_embeddings, df)
    similar_tags = [tag for tag, _, _ in
find_similar_tags(best_tag, best_tag_emb, best_relevance,
tag_embeddings, df)]
    rag_queries = generate_rag_query(corrected_query,
similar_tags)
    return [corrected_query] + rag_queries
```