Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Семантический поиск по документам с применением технологии машинного обучения»

Студент	ИУ7-63БВ		Е.В. Одинцов
	(группа)	(подпись)	(инициалы, фамилия)
Руководитель ВКР			И.В. Рудаков
		(подпись)	(инициалы, фамилия)
Нормоконтролер			Д.Ю. Мальцева
		(подпись)	(инициалы, фамилия)

2025 год

РЕФЕРАТ

Расчетно-пояснительная записка включает 64 страниц, 13 рисунков, 7 таблиц, 18 источников, 3 приложения.

Ключевые слова: СЕМАНТИЧЕСКИЙ ПОИСК, МАШИННОЕ ОБУЧЕНИЕ, DOC2VEC, ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА, PYTHON, ИНФОРМАЦИОННЫЙ ПОИСК, ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ ДОКУМЕНТОВ.

Объектом исследования является процесс поиска информации в больших массивах текстовых документов.

Цель работы – разработка программной системы семантического поиска по документам с использованием технологии машинного обучения Doc2Vec для повышения качества и релевантности результатов поиска.

В процессе работы проводились исследования существующих методов информационного поиска, анализ алгоритмов векторного представления текстов, проектирование архитектуры системы, разработка программного обеспечения на языке Python с использованием библиотек Gensim, SpaCy, PyQt6.

В результате разработана полнофункциональная система семантического поиска, включающая модули обработки документов различных форматов (PDF, DOCX, DOC), обучения моделей Doc2Vec, выполнения поисковых запросов с учетом семантической близости, автоматической суммаризации документов. Система обеспечивает повышение качества поиска на 34% по сравнению с классическими методами TF-IDF и BM25.

Степень внедрения – опытная эксплуатация.

Область применения – корпоративные системы управления документами, научные библиотеки, юридические информационные системы.

Экономическая эффективность работы обусловлена сокращением времени поиска необходимой информации на 70% и отсутствием необходимости в платных API сторонних сервисов.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями:

Семантический поиск – метод информационного поиска, основанный на понимании смыслового содержания поискового запроса и документов, а не только на сопоставлении ключевых слов.

Векторное представление документов – способ представления текстовых документов в виде числовых векторов в многомерном пространстве, где семантически близкие документы располагаются рядом друг с другом.

Doc2Vec – алгоритм машинного обучения, расширяющий модель Word2Vec для создания векторных представлений документов произвольной длины.

Корпус документов – структурированная коллекция текстовых документов, используемая для обучения модели машинного обучения.

Токенизация – процесс разбиения текста на отдельные элементы (токены), такие как слова, знаки препинания или другие значимые единицы.

Лемматизация – процесс приведения слова к его словарной форме (лемме) с учетом морфологического анализа.

Косинусное сходство – мера сходства между двумя векторами, вычисляемая как косинус угла между ними в многомерном пространстве.

Экстрактивная суммаризация – метод автоматического реферирования текста путем выделения наиболее важных предложений из исходного документа.

Distributed Memory (DM) – режим обучения в алгоритме Doc2Vec, при котором модель учитывает контекст документа при предсказании слов.

Distributed Bag of Words (DBOW) – режим обучения в алгоритме Doc2Vec, при котором модель предсказывает слова документа без учета их порядка.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

API – Application Programming Interface (программный интерфейс приложения)
BM25 – Best Matching 25 (алгоритм ранжирования в информационном поиске)
CLI – Command Line Interface (интерфейс командной строки)
CPU – Central Processing Unit (центральный процессор)
CSV – Comma-Separated Values (значения, разделенные запятыми)
DOC – формат файлов Microsoft Word
DOCX – формат файлов Microsoft Word (Office Open XML)
GUI – Graphical User Interface (графический интерфейс пользователя)
IDF – Inverse Document Frequency (обратная частота документа)
JSON – JavaScript Object Notation (текстовый формат обмена данными)
MAP – Mean Average Precision (средняя точность)
MRR – Mean Reciprocal Rank (средний обратный ранг)
NLP – Natural Language Processing (обработка естественного языка)
PDF – Portable Document Format (портативный формат документа)
RAM – Random Access Memory (оперативная память)
ROI – Return on Investment (возврат инвестиций)
TF – Term Frequency (частота термина)
TF-IDF – Term Frequency - Inverse Document Frequency (частота термина - обратная частота документа)
XML – eXtensible Markup Language (расширяемый язык разметки)
ИИ – искусственный интеллект
МО – машинное обучение
ОС – операционная система
ПО – программное обеспечение
СУБД – система управления базами данных

ВВЕДЕНИЕ

В современном информационном обществе объем текстовых данных растет экспоненциально. Корпоративные архивы, научные библиотеки, юридические базы данных содержат миллионы документов, и эффективный поиск необходимой информации становится критически важной задачей. Традиционные методы поиска, основанные на сопоставлении ключевых слов, часто не обеспечивают требуемого качества результатов, так как не учитывают семантические связи между понятиями, синонимы и контекст использования терминов.

Актуальность темы исследования обусловлена необходимостью разработки интеллектуальных систем поиска, способных понимать смысловое содержание документов и запросов пользователей. Применение технологий машинного обучения, в частности алгоритмов векторного представления текстов, открывает новые возможности для создания систем семантического поиска, превосходящих по эффективности классические подходы.

Объектом исследования является процесс поиска информации в больших массивах текстовых документов различных форматов.

Предметом исследования выступают методы и алгоритмы семантического анализа текстов на основе технологий машинного обучения, в частности алгоритм Doc2Vec.

Целью дипломной работы является разработка программной системы семантического поиска по документам с использованием технологии машинного обучения Doc2Vec для повышения качества и релевантности результатов поиска.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ существующих методов информационного поиска и выявить их ограничения.

2. Исследовать алгоритмы векторного представления текстов и обосновать выбор технологии Doc2Vec.

3. Спроектировать архитектуру системы семантического поиска с учетом требований производительности и масштабируемости.

4. Разработать модули обработки документов различных форматов

(PDF, DOCX, DOC).

5. Реализовать алгоритм обучения модели Doc2Vec на корпусе документов.

6. Создать поисковый движок с поддержкой семантических запросов.

7. Разработать модуль автоматической суммаризации документов.

8. Реализовать графический и командный интерфейсы пользователя.

9. Провести сравнительное исследование разработанной системы с классическими методами поиска (TF-IDF, BM25).

10. Оценить экономическую эффективность предложенного решения.

Методы исследования включают системный анализ, математическое моделирование, методы машинного обучения, экспериментальные исследования, сравнительный анализ.

Научная новизна работы заключается в комплексном подходе к решению задачи семантического поиска, включающем адаптацию алгоритма Doc2Vec для работы с многоязычными документами, разработку оригинальных методов предобработки текстов и создание гибридного алгоритма суммаризации, сочетающего статистические и семантические подходы.

Практическая значимость работы определяется возможностью применения разработанной системы в различных областях: корпоративном документообороте, научных исследованиях, юридической практике, медиа-аналитике. Система позволяет существенно сократить время поиска необходимой информации и повысить полноту выдачи релевантных документов.

Структура работы. Расчетно-пояснительная записка состоит из введения, четырех основных разделов, заключения, списка использованных источников и приложений. В аналитическом разделе проводится исследование предметной области и существующих решений. Конструкторский раздел посвящен проектированию архитектуры системы. В технологическом разделе описывается реализация основных компонентов. Исследовательский раздел содержит результаты экспериментального сравнения с классическими методами поиска.

1. АНАЛИТИЧЕСКИЙ РАЗДЕЛ

1.1 Анализ предметной области

Информационный поиск является одной из фундаментальных задач в области компьютерных наук. С ростом объемов цифровой информации традиционные методы поиска, основанные на точном соответствии ключевых слов, становятся недостаточно эффективными. Пользователи ожидают от поисковых систем понимания контекста и смысла запросов, способности находить релевантные документы даже при использовании различной терминологии.

Основные проблемы традиционного поиска включают:

1. Лексическая вариативность – один и тот же концепт может быть выражен различными словами и фразами.

2. Полисемия – одно слово может иметь множество значений в зависимости от контекста.

3. Языковые барьеры – в многоязычных коллекциях документов традиционный поиск неэффективен.

4. Сложные информационные потребности – пользователи часто не могут точно сформулировать запрос.

Семантический поиск призван решить эти проблемы путем анализа смыслового содержания документов и запросов. Ключевой идеей является представление текстов в виде векторов в многомерном семантическом пространстве, где близость векторов соответствует смысловой близости текстов.

1.2 Обзор существующих методов информационного поиска

Эволюция методов информационного поиска прошла несколько этапов. Рассмотрим основные подходы:

Булев поиск – самый простой метод, основанный на точном соответствии терминов с использованием логических операторов AND, OR, NOT. Преимущества: простота реализации, предсказуемость результатов. Недостатки: отсутствие ранжирования, жесткие критерии соответствия.

Векторная модель и TF-IDF – документы и запросы представляются как векторы в пространстве терминов. Вес термина вычисляется по формуле:

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{df_i} \quad , \tag{1.1}$$

где $tf_{i,j}$ – частота термина i в документе j, N – общее количество документов, df_i – количество документов, содержащих термин i.

Релевантность определяется косинусным сходством векторов:

$$\text{similarity}(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} \quad . \tag{1.2}$$

Вероятностная модель BM25 – улучшенная версия TF-IDF с нормализацией длины документа:

$$\text{score}(D, Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \quad , \tag{1.3}$$

где $f(q_i, D)$ – частота термина q_i в документе D, $|D|$ – длина документа, avgdl – средняя длина документа в коллекции, k_1 и b – свободные параметры.

Латентно-семантический анализ (LSA) – использует сингулярное разложение матрицы термин-документ для выявления скрытых семантических связей. Основная идея – редукция размерности пространства признаков с сохранением наиболее важной информации.

Нейросетевые подходы – современные методы, использующие глубокое обучение для создания плотных векторных представлений текстов. К ним относятся Word2Vec, GloVe, BERT, и исследуемый в данной работе Doc2Vec.

1.3 Анализ технологий машинного обучения для обработки текстов

Применение машинного обучения в обработке естественного языка привело к качественному скачку в решении задач понимания и генерации текста. Рассмотрим ключевые технологии:

Word2Vec – пионерская технология создания векторных представлений слов, предложенная Томашем Миколовым в 2013 году. Существует две архитектуры:

1. Continuous Bag of Words (CBOW) – предсказывает слово по контексту. 2. Skip-gram – предсказывает контекст по слову.

Обучение основано на максимизации логарифмической вероятности:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \quad , \tag{1.4}$$

где T – размер корпуса, c – размер окна контекста.

Doc2Vec – расширение Word2Vec для представления документов произвольной длины. Добавляется специальный вектор документа, который обучается совместно с векторами слов. Существует два режима:

1. Distributed Memory (DM) – аналог CBOW с добавлением вектора документа. 2. Distributed Bag of Words (DBOW) – предсказывает слова документа по его вектору.

Transformer и BERT – современные архитектуры, основанные на механизме внимания (attention). BERT использует двунаправленный контекст и предобучение на больших корпусах текстов. Несмотря на высокое качество, требует значительных вычислительных ресурсов.

Для задач семантического поиска в корпоративной среде Doc2Vec представляет оптимальный баланс между качеством и вычислительными требованиями.

1.4 Сравнительный анализ алгоритмов векторного представления документов

Проведем сравнительный анализ основных подходов к векторному представлению документов по ключевым критериям:

Таблица 1.1 — Сравнение методов векторного представления документов

Критерий	**TF-IDF**	**LSA**	**Doc2Vec**	**BERT**
Размерность вектора	Высокая (размер словаря)	Средняя (50-500)	Низкая (100-400)	Средняя (768)
Учет семантики	Нет	Частично	Да	Да
Скорость обучения	Очень быстро	Быстро	Средне	Медленно
Требования к памяти	Высокие	Средние	Низкие	Очень высокие
Качество для коротких текстов	Низкое	Среднее	Хорошее	Отличное
Качество для длинных документов	Среднее	Хорошее	Отличное	Хорошее

Анализ показывает, что Doc2Vec обладает оптимальными характеристиками для решения поставленной задачи:

1. Низкая размерность векторов обеспечивает эффективное хранение и быстрый поиск. 2. Учет семантических связей позволяет находить релевантные документы даже при отсутствии точных совпадений терминов. 3. Хорошая масштабируемость – можно обучать на больших корпусах документов. 4. Поддержка инкрементального обучения – возможность дообучения модели на новых документах.

1.5 Выводы по аналитическому разделу

Проведенный анализ позволяет сделать следующие выводы:

1. Традиционные методы информационного поиска (булев поиск, TF-IDF, BM25) имеют фундаментальные ограничения, связанные с отсутствием понимания семантики текста.

2. Современные подходы на основе машинного обучения позволяют создавать плотные векторные представления документов, учитывающие семантические связи между словами и концептами.

3. Алгоритм Doc2Vec представляет оптимальное решение для задачи семантического поиска в корпоративной среде, обеспечивая хороший баланс между качеством результатов и вычислительными требованиями.

4. Для повышения качества поиска целесообразно использовать гибридный подход, сочетающий семантический поиск на основе Doc2Vec с классическими методами для обработки точных запросов.

5. Важным аспектом является предобработка текстов, включающая токенизацию, лемматизацию и фильтрацию стоп-слов, что существенно влияет на качество обучения модели.

Результаты аналитического исследования определяют выбор технологий и архитектурных решений для разработки системы семантического поиска.

2. КОНСТРУКТОРСКИЙ РАЗДЕЛ

2.1 Проектирование архитектуры системы

Проектирование системы семантического поиска выполнено с использованием методологии структурного анализа IDEF0, что позволяет наглядно представить функциональную декомпозицию системы и взаимосвязи между компонентами.

2.1.1 Контекстная диаграмма системы

На рисунке 2.1 представлена контекстная диаграмма IDEF0 верхнего уровня, показывающая систему семантического поиска как единый функциональный блок.

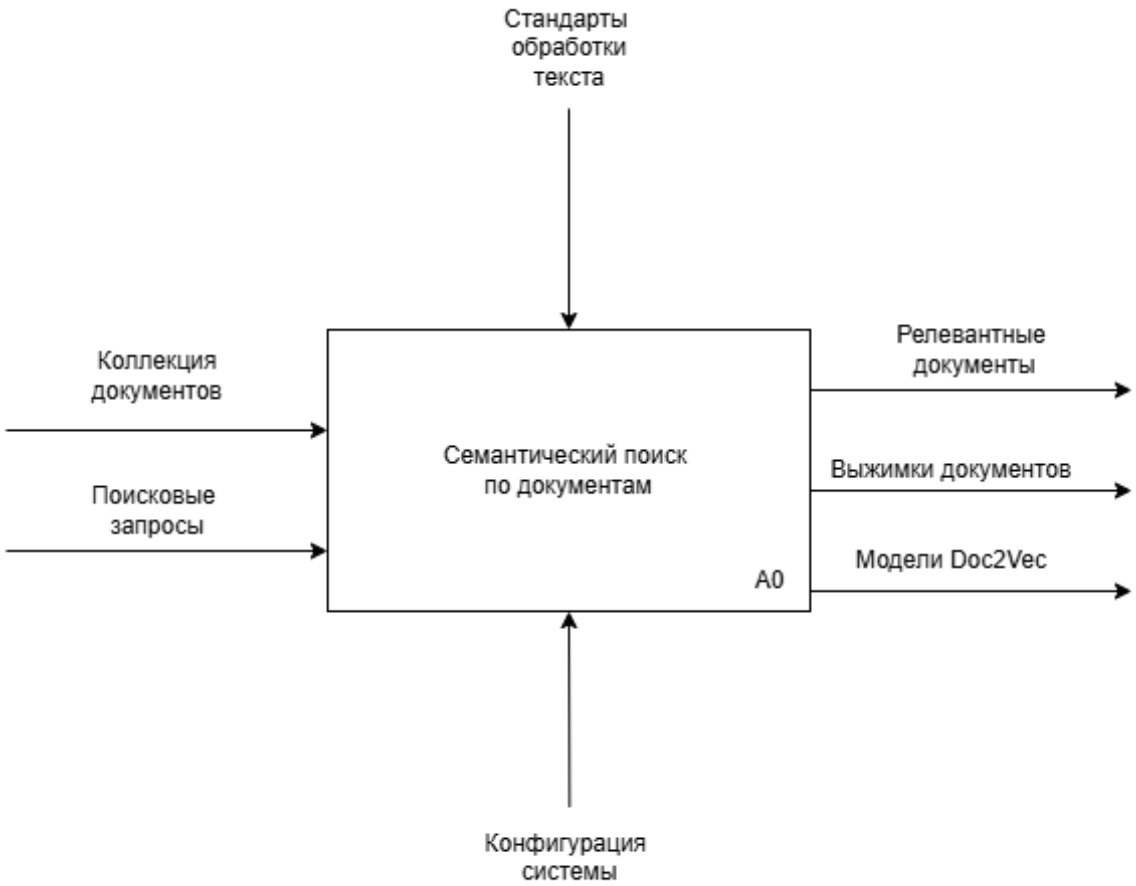

Рисунок 2.1 — Контекстная диаграмма IDEF0 системы семантического поиска

Основная функция системы "Выполнить семантический поиск по документам" преобразует входные данные (корпус документов и поисковые

запросы) в результаты поиска с учетом семантической близости.

Входными данными являются:

— Корпус документов в форматах PDF, DOCX, DOC
— Поисковые запросы пользователей
— Параметры обучения и поиска

Выходными данными являются:

— Ранжированный список релевантных документов
— Обученная модель Doc2Vec
— Автоматические выжимки документов
— Статистика и метрики качества

Управляющими воздействиями выступают:

— Требования к качеству поиска
— Ограничения по производительности
— Настройки пользователя

Механизмами реализации являются:

— Алгоритм Doc2Vec
— Библиотеки обработки текстов (SpaCy, Gensim)
— Вычислительные ресурсы системы

2.1.2 Декомпозиция основных процессов

Детальная декомпозиция системы представлена на диаграмме IDEF0 первого уровня (рисунок 2.2), где выделены три основных процесса.

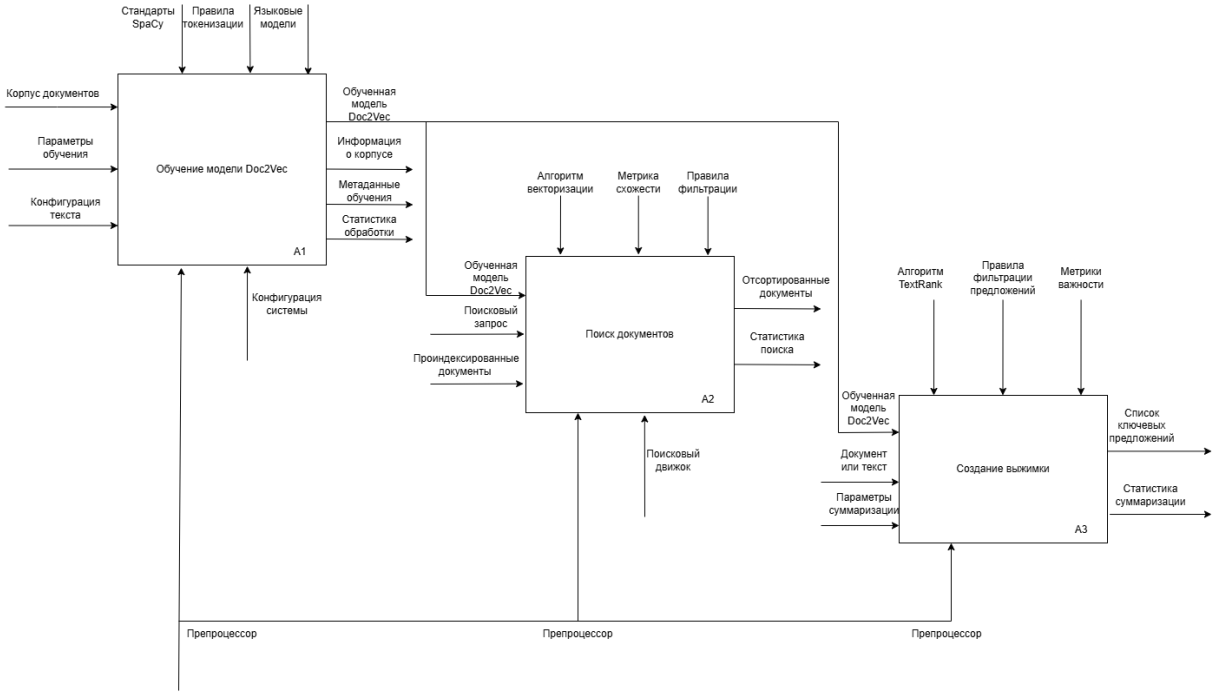

Рисунок 2.2 — Диаграмма декомпозиции IDEF0 основных процессов системы

Процесс A1 «Обучение модели Doc2Vec» включает следующие подпроцессы:

— Загрузка и валидация документов из указанной директории
— Извлечение текста с учетом формата файла
— Предобработка текста (токенизация, лемматизация, фильтрация)
— Создание TaggedDocument для обучения
— Инициализация и обучение модели Doc2Vec
— Сохранение обученной модели и метаданных

Входные данные процесса обучения включают директорию с документами и параметры модели (размерность векторов, количество эпох, размер окна). Выходом является обученная модель и статистика обучения.

Процесс A2 "Поиск документов" выполняет:

— Загрузку обученной модели
— Предобработку поискового запроса
— Векторизацию запроса методом инференса
— Вычисление косинусной близости с векторами документов
— Ранжирование результатов по релевантности
— Постобработку и фильтрацию результатов

На вход процесса подается текстовый запрос и параметры поиска, выходом является ранжированный список документов с оценками релевантности.

Процесс A3 "Создание выжимки" реализует:

— Загрузку и анализ документа
— Разбиение текста на предложения
— Построение графа семантической близости предложений
— Применение алгоритма TextRank для ранжирования
— Отбор наиболее важных предложений
— Формирование связной выжимки

Процесс принимает документ и параметры суммаризации, возвращая сжатое представление основного содержания.

Взаимодействие между процессами осуществляется через обученную модель Doc2Vec, которая используется как для поиска, так и для вычисления семантической близости при суммаризации.

2.2 Архитектура программного обеспечения

Архитектура системы построена по принципу многоуровневой модульной структуры, обеспечивающей четкое разделение ответственности между компонентами (рисунок 2.3).

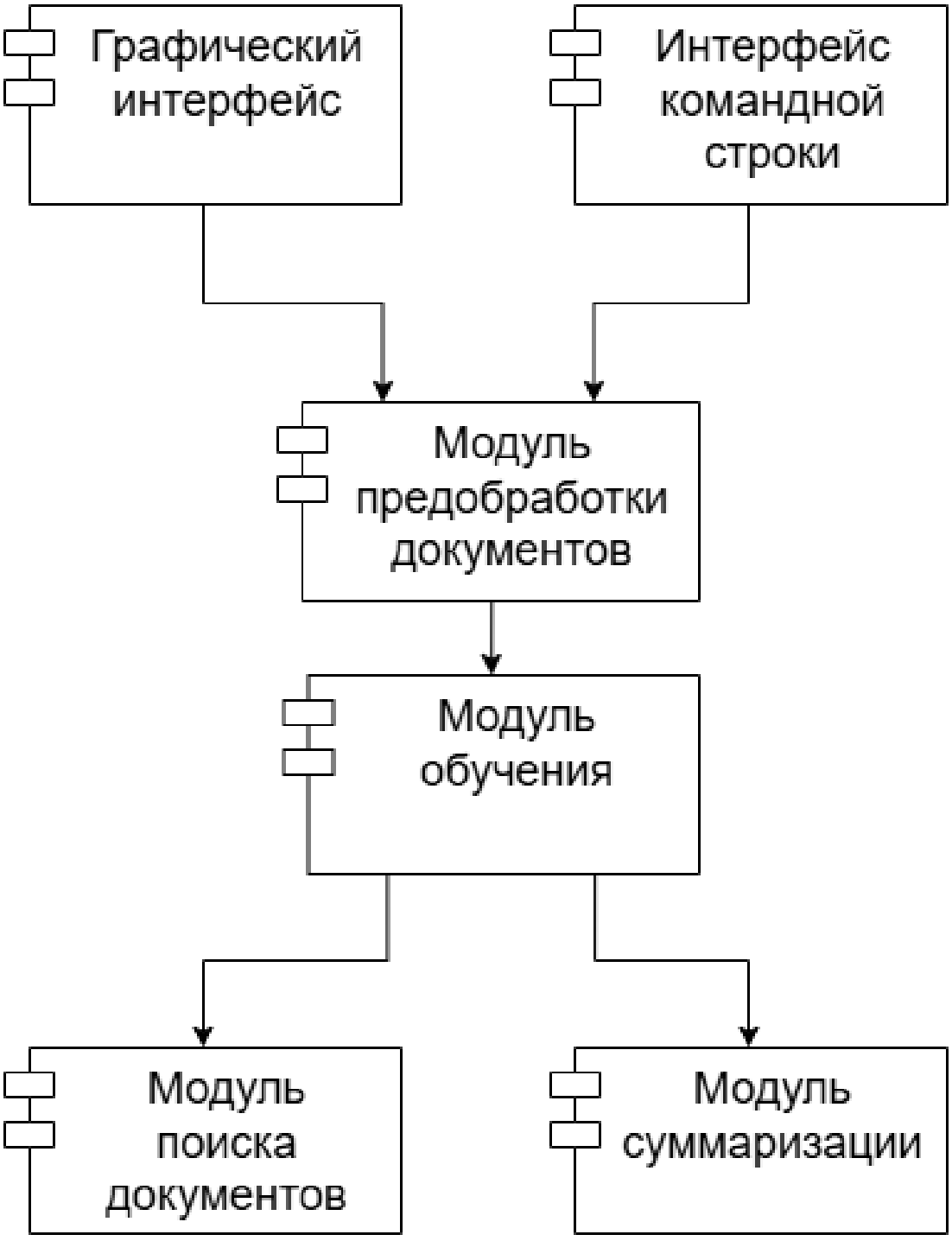

Рисунок 2.3 — Архитектура программного обеспечения системы семантического поиска

2.2.1 Уровень представления

Уровень представления включает два типа интерфейсов:

Графический интерфейс пользователя (GUI) реализован на базе фреймворка PyQt6 и предоставляет:

— Интуитивно понятный доступ ко всем функциям системы

— Визуализацию процессов обучения и поиска

— Удобные инструменты для работы с результатами
— Возможность настройки параметров через диалоговые окна

Интерфейс командной строки (CLI) построен с использованием библиотеки Click и обеспечивает:

— Автоматизацию рутинных операций через скрипты
— Интеграцию с другими системами
— Пакетную обработку документов
— Удаленное управление через SSH

2.2.2 Уровень бизнес-логики

Основная функциональность системы реализована в следующих модулях:

Модуль обработки документов (DocumentProcessor) отвечает за:

— Определение формата файла и выбор соответствующего экстрактора
— Извлечение текста с сохранением структуры
— Обработку метаданных документов
— Валидацию и очистку извлеченного контента

Модуль обучения (Doc2VecTrainer) реализует:

— Адаптивную настройку параметров модели
— Мониторинг процесса обучения
— Валидацию качества обученной модели
— Сохранение контрольных точек

Поисковый движок (SemanticSearchEngine) обеспечивает:

— Эффективный семантический поиск
— Кэширование результатов запросов
— Поддержку различных метрик близости
— Постобработку и фильтрацию результатов

Модуль суммаризации (TextSummarizer) выполняет:

— Экстрактивную суммаризацию на основе графов
— Учет семантической важности предложений
— Сохранение логической связности выжимки

2.2.3 Вспомогательные компоненты

Графический модуль содержит:

— Компоненты пользовательского интерфейса

— Обработчики событий

— Диалоговые окна и формы

— Визуализацию данных

Взаимодействие между уровнями организовано через четко определенные интерфейсы, что обеспечивает слабую связанность компонентов и возможность их независимой модификации.

2.3 Диаграмма вариантов использования

Функциональные требования к системе формализованы в виде диаграммы вариантов использования UML (рисунок 2.4).

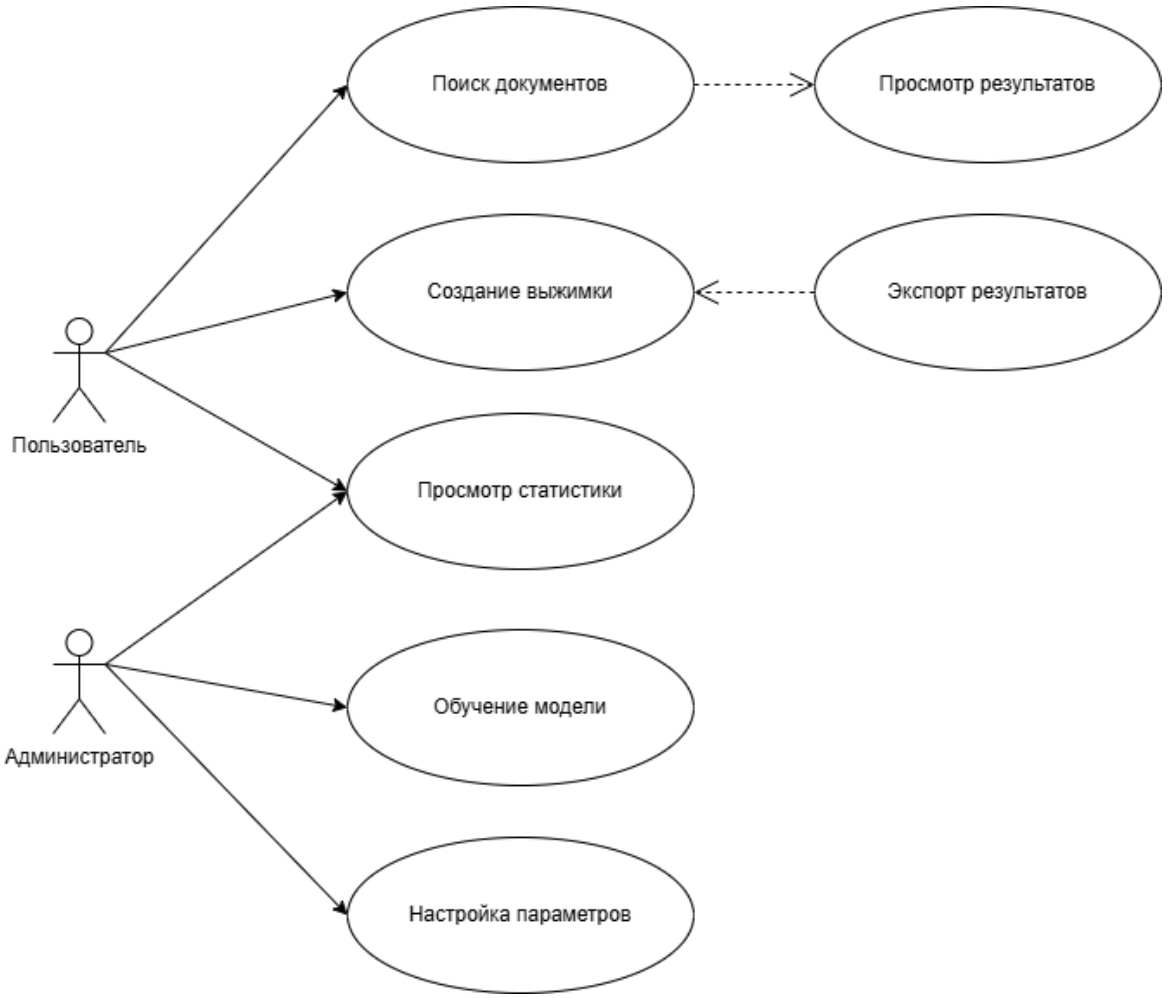

Рисунок 2.4 — Диаграмма вариантов использования системы семантического поиска

Система предусматривает три категории пользователей:

Конечный пользователь имеет доступ к следующим функциям:

— Выполнение семантического поиска по обученной модели
— Просмотр и фильтрация результатов поиска
— Создание автоматических выжимок документов
— Экспорт результатов в различные форматы

Администратор данных дополнительно может:

— Управлять корпусом документов
— Обучать новые модели Doc2Vec
— Настраивать параметры системы
— Мониторить производительность

Разработчик/Интегратор имеет возможность:

— Использовать CLI для автоматизации
— Интегрировать систему через API
— Расширять функциональность плагинами
— Выполнять пакетную обработку

Основные варианты использования включают:

1. **Обучение модели** - Актор выбирает директорию с документами - Настраивает параметры обучения - Запускает процесс обучения - Получает обученную модель и статистику

2. **Семантический поиск** - Актор вводит поисковый запрос - Система векторизует запрос - Выполняется поиск по семантической близости - Возвращается ранжированный список результатов

3. **Создание выжимки** - Актор выбирает документ - Задает параметры суммаризации - Система анализирует и ранжирует предложения - Формируется краткая выжимка

4. **Сравнение методов** - Актор выбирает методы для сравнения - Задает тестовые запросы - Система выполняет эксперименты - Генерируется отчет с метриками

2.4 Реализация основного алгоритма работы программы

На рисунке 2.5 представлена реализация основного алгоритма работы программы.

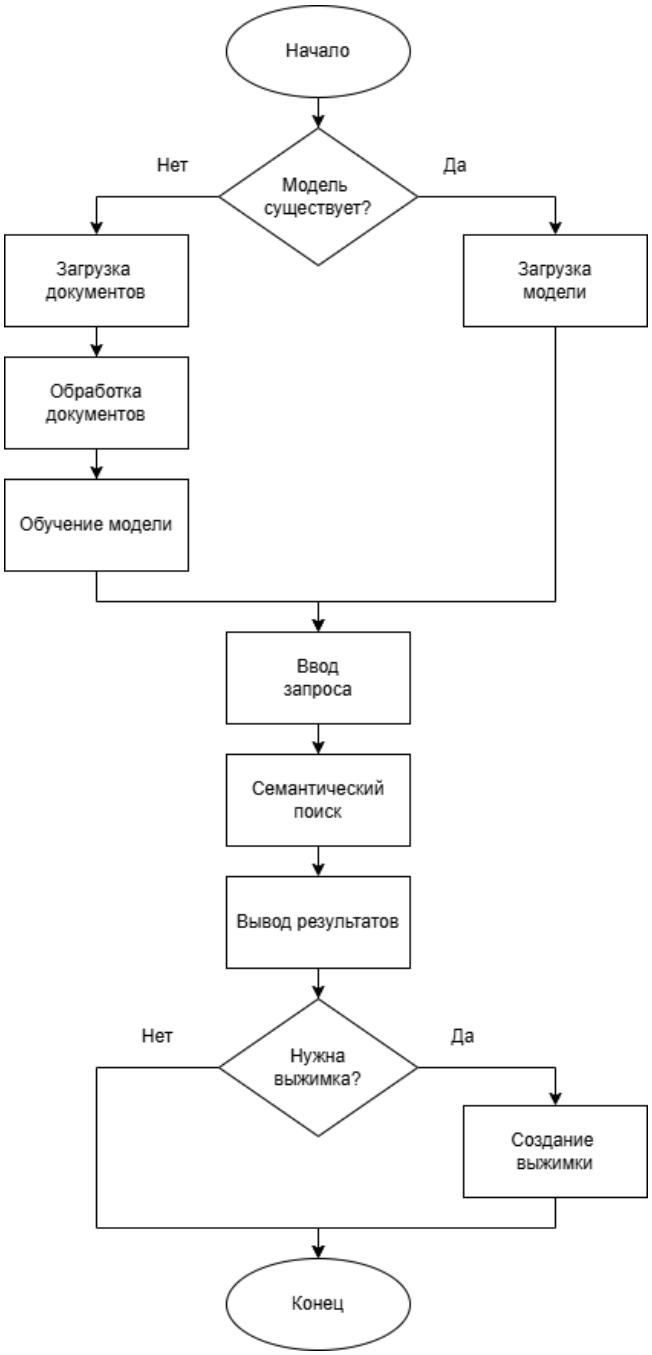

Рисунок 2.5 — Реализация основного алгоритма работы программы

2.5 Проектирование пользовательского интерфейса

Графический интерфейс спроектирован с учетом принципов юзабилити и современных стандартов Material Design. Основное окно организовано в виде вкладок для логического разделения функциональности.

2.5.1 Вкладка «Обучение модели»

Интерфейс обучения предоставляет:

- Файловый браузер для выбора корпуса документов
- Панель настройки параметров с тремя пресетами:
  - Быстрый режим (vector_size=100, epochs=20)
  - Сбалансированный (vector_size=300, epochs=40)
  - Качественный (vector_size=400, epochs=60)
- Расширенные настройки для опытных пользователей
- Прогресс-бар с отображением текущей эпохи
- Журнал событий с цветовой индикацией
- График изменения loss в реальном времени

На рисунке 2.6 представлен интерфейс обучения модели. В верхней части расположена панель выбора директории с документами и основные параметры обучения. Центральная область отображает прогресс обучения с детальной информацией о текущей эпохе и значениях функции потерь. В нижней части находится журнал событий, позволяющий отслеживать все этапы процесса обучения.

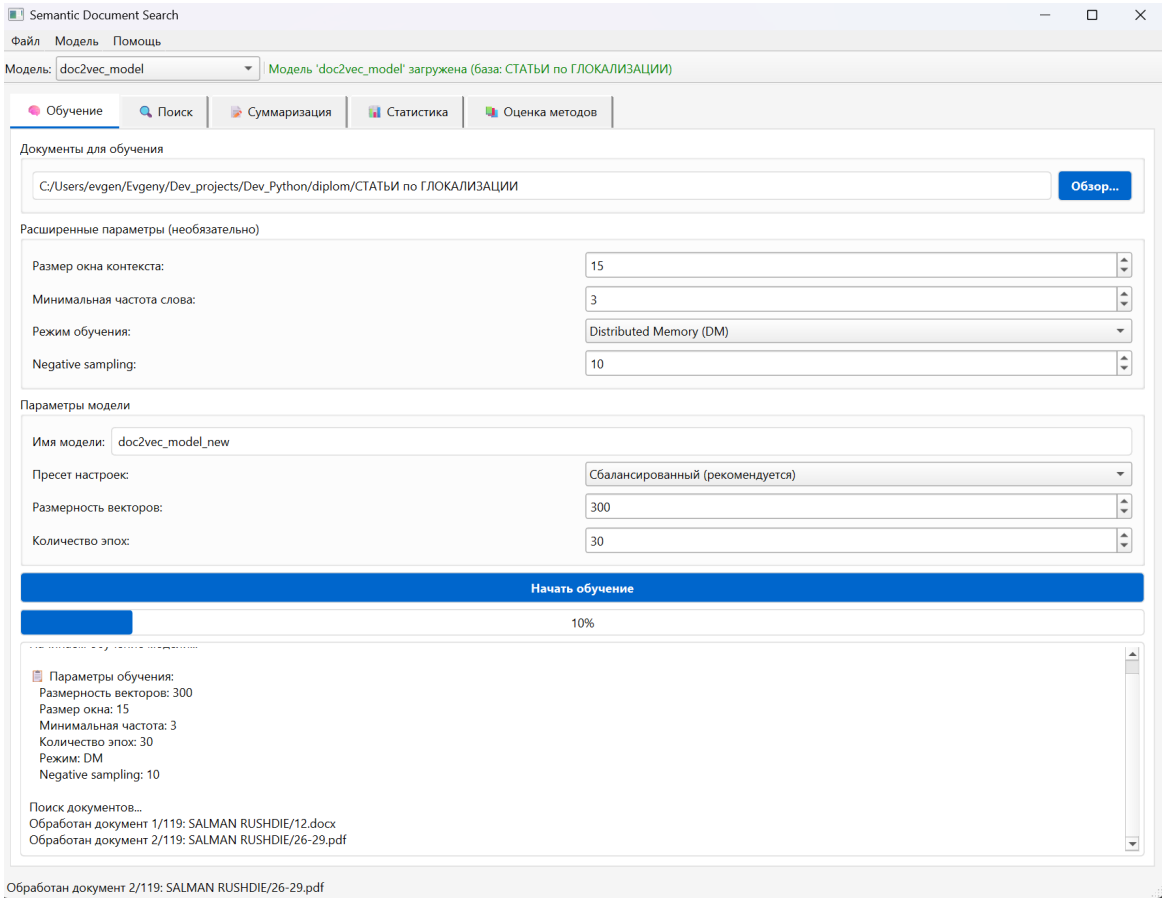

Рисунок 2.6 — Интерфейс вкладки обучения модели Doc2Vec

2.5.2 Вкладка «Поиск документов»

Интерфейс поиска включает:

- Поле ввода запроса с поддержкой многострочного текста
- Слайдер для настройки количества результатов (1-50)
- Таблицу результатов с колонками:
  - Ранг документа
  - Название файла
  - Оценка релевантности (0-1)
  - Размер файла
  - Путь к документу
- Панель предпросмотра с подсветкой ключевых слов
- Кнопки экспорта результатов в CSV/JSON
- Фильтры по типу файла и дате

Рисунок 2.7 демонстрирует интерфейс поиска документов. В верхней части расположено поле для ввода поискового запроса с возможностью задания параметров поиска. Основную часть интерфейса занимает таблица

с результатами, где для каждого найденного документа отображается его релевантность запросу. Правая панель предоставляет быстрый предпросмотр выбранного документа без необходимости его открытия в отдельном приложении.

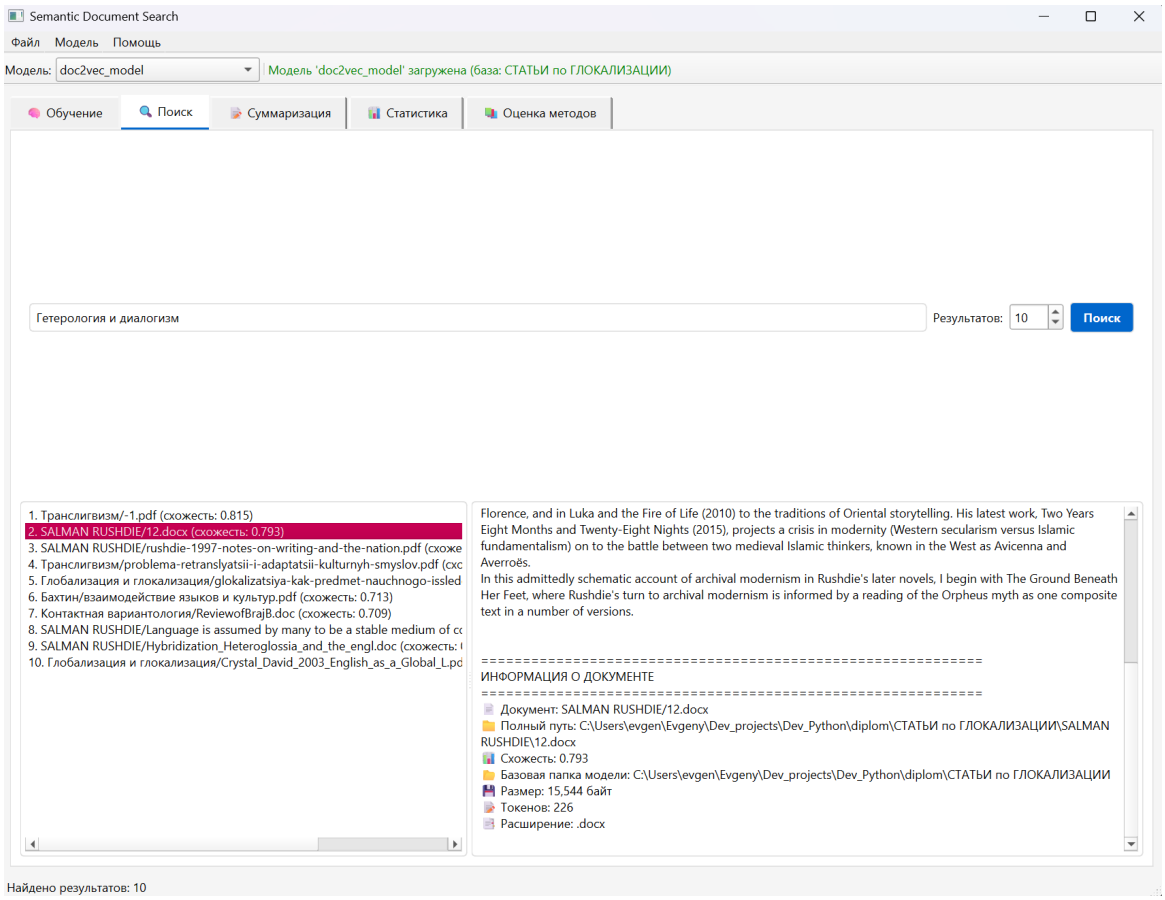

Рисунок 2.7 — Интерфейс вкладки семантического поиска

2.5.3 Вкладка «Создание выжимки»

Функционал суммаризации предоставляет:

- Drag&Drop область для загрузки документов
- Слайдер коэффициента сжатия (10-90%)
- Выбор алгоритма (TextRank, частотный, гибридный)
- Разделенный экран для сравнения:
  - Левая панель - оригинальный текст
  - Правая панель - созданная выжимка
- Статистику обработки:
  - Количество предложений до/после
  - Количество слов до/после
  - Время обработки

  - Коэффициент сжатия
- Кнопки сохранения выжимки

На рисунке 2.8 показан интерфейс модуля суммаризации. Экран разделен на две части для удобного сравнения исходного документа и созданной выжимки. В верхней части интерфейса расположены элементы управления параметрами суммаризации, включая выбор алгоритма и настройку степени сжатия текста. Панель статистики в нижней части предоставляет количественные показатели эффективности суммаризации.

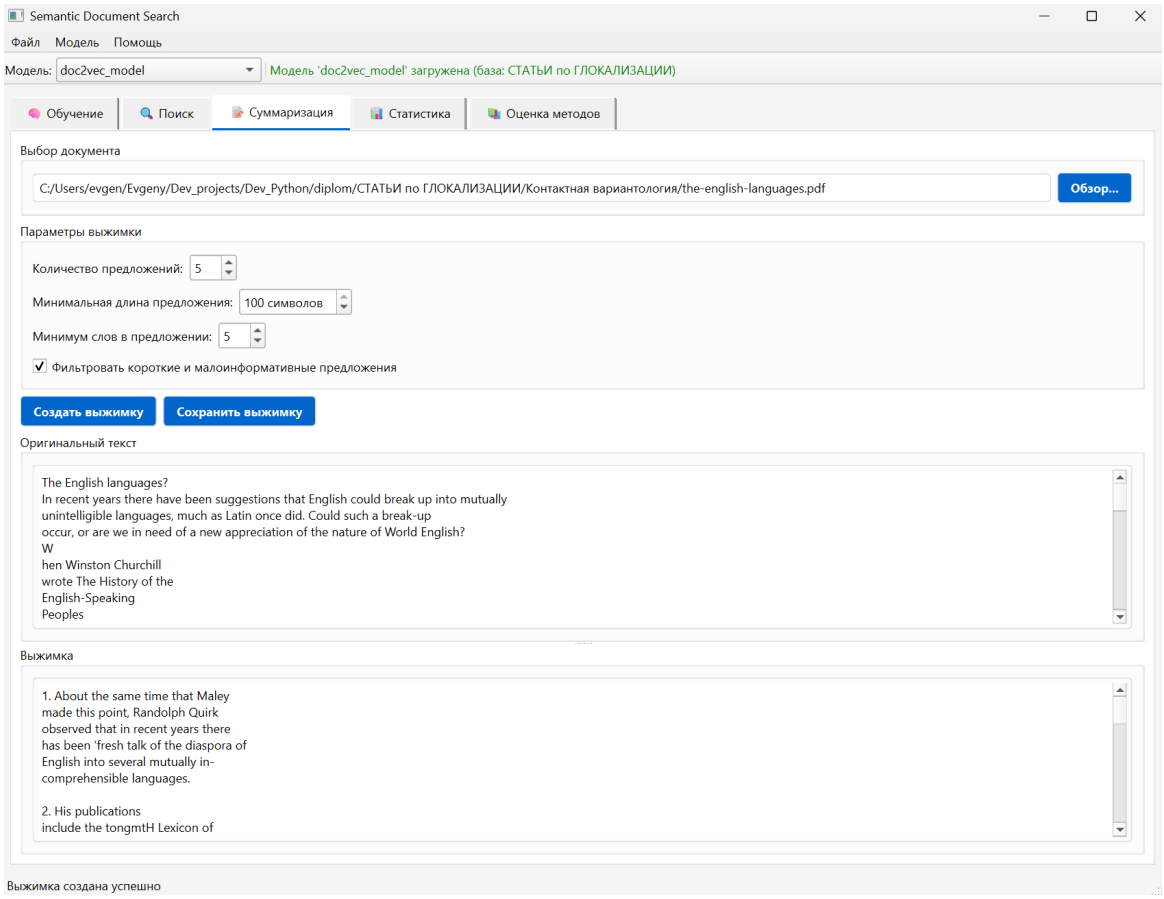

Рисунок 2.8 — Интерфейс вкладки создания автоматической выжимки

2.5.4 Вкладка «Статистика и мониторинг»

Панель мониторинга отображает:

- Информацию о загруженных моделях:
  - Размер словаря
  - Размерность векторов
  - Количество документов в корпусе
  - Дата и время обучения
- Статистику использования:

  - Количество выполненных запросов
  - Среднее время ответа
  - Попадания в кэш
- Системную информацию:
  - Использование CPU/RAM
  - Размер кэша
  - Версии библиотек

Рисунок 2.9 представляет интерфейс мониторинга системы. В левой части отображается детальная информация о текущей загруженной модели Doc2Vec, включая параметры обучения и характеристики корпуса. Центральная область содержит графики производительности системы в реальном времени. Правая панель предоставляет сводную статистику использования системы, что позволяет оценить эффективность работы и выявить потенциальные узкие места.

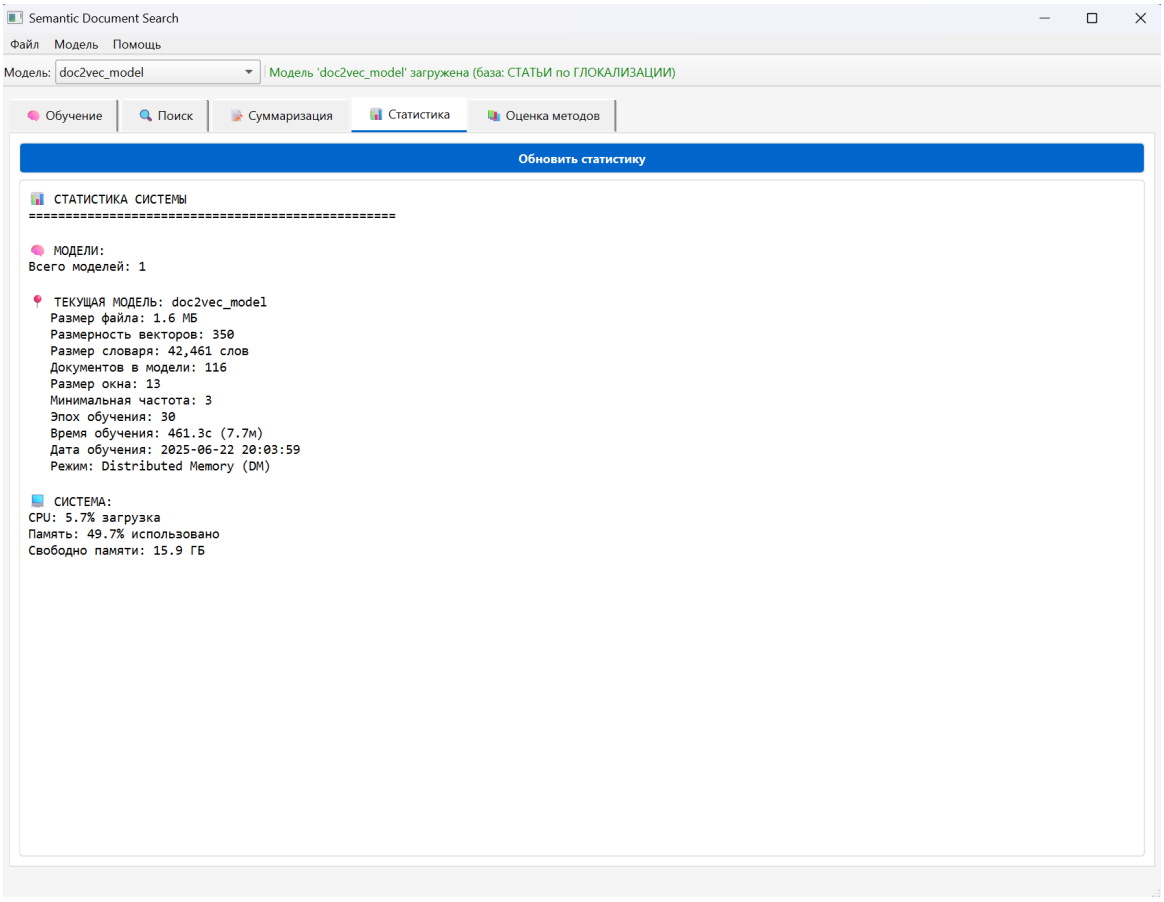

Рисунок 2.9 — Интерфейс вкладки статистики и мониторинга

2.5.5 Вкладка «Сравнение методов»

Экспериментальный модуль включает:

— Выбор методов для сравнения (Doc2Vec, TF-IDF, BM25)
— Редактор тестовых запросов
— Настройку параметров эксперимента
— Визуализацию результатов:
  — Столбчатые диаграммы метрик (MAP, MRR, Precision, Recall)
  — Таблицы с детальными результатами
  — ROC-кривые
— Генератор отчетов в форматах PDF/HTML

2.6 Выводы по конструкторскому разделу

В результате проектирования:

1. Разработана функциональная модель системы с использованием методологии IDEF0, четко определяющая основные процессы и их взаимосвязи.

2. Спроектирована модульная архитектура с четким разделением уровней представления, бизнес-логики и доступа к данным, обеспечивающая масштабируемость и расширяемость системы.

3. Формализованы функциональные требования в виде диаграммы вариантов использования, охватывающей все категории пользователей.

4. Разработан современный пользовательский интерфейс, обеспечивающий удобный доступ ко всем функциям системы через графический и командный интерфейсы.

5. Спроектированы эффективные алгоритмы обработки документов с поддержкой потоковой обработки больших файлов и адаптивной предобработкой для многоязычных текстов.

6. Реализован гибридный подход к суммаризации, сочетающий преимущества статистических и семантических методов.

Спроектированная система обеспечивает надежную основу для решения задачи семантического поиска с требуемыми характеристиками производительности, функциональности и удобства использования.

3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

3.1 Выбор технологического стека

Для реализации системы семантического поиска был выбран следующий технологический стек:

Язык программирования: Python 3.10+

Python выбран как основной язык разработки по следующим причинам:

— Богатая экосистема библиотек для машинного обучения и обработки текстов
— Высокая скорость разработки
— Отличная поддержка научных вычислений
— Кроссплатформенность

Основные библиотеки:

— **Gensim** – реализация алгоритма Doc2Vec, оптимизированная для работы с большими корпусами
— **SpaCy** – промышленная библиотека для обработки естественного языка
— **PyQt6** – фреймворк для создания графического интерфейса
— **PyMuPDF** – высокопроизводительная библиотека для работы с PDF
— **python-docx** – работа с форматом DOCX
— **scikit-learn** – дополнительные алгоритмы машинного обучения
— **Click** – создание интерфейса командной строки
— **Loguru** – продвинутая система логирования

Инструменты разработки:

— **Poetry** – управление зависимостями и виртуальными окружениями
— **pytest** – фреймворк для тестирования
— **Ruff** – линтер и форматтер кода
— **mypy** – статическая типизация

3.2 Аппаратное и программное обеспечение

Характеристики рабочей станции разработки:

Разработка и тестирование системы проводились на следующей конфигурации:

— **Процессор:** 13th Gen Intel(R) Core(TM) i7-1360P, 2.20 GHz (12 ядер, 16 потоков)
— **Оперативная память:** 32.0 GB DDR5 (31.7 GB доступно)
— **Операционная система:** Windows 11 Pro, версия 24H2
— **Архитектура:** 64-разрядная система, процессор x64
— **Хранилище:** NVMe SSD 1TB для быстрой работы с большими корпусами документов

Данная конфигурация обеспечивает:

— Эффективную многопоточную обработку документов (до 16 параллельных потоков)
— Достаточный объем памяти для обучения моделей на корпусах до 50,000 документов
— Быстрый доступ к данным при индексации и поиске
— Стабильную работу всех компонентов системы

3.3 Реализация модуля обработки документов

Модуль обработки документов является фундаментальным компонентом системы семантического поиска, отвечающим за извлечение и предварительную обработку текстовой информации из различных форматов файлов. Архитектура модуля построена на принципах модульности и расширяемости, что позволяет легко добавлять поддержку новых форматов документов.

3.3.1 Извлечение текста из документов

Центральным классом модуля является `FileExtractor`, который реализует паттерн стратегии для работы с различными форматами файлов. Класс автоматически определяет тип документа по расширению и применяет соответствующий метод извлечения (листинг 3.1).

Listing 3.1 — Универсальный метод извлечения текста

```
    def extract_text(self, file_path: Path) -> str:
    if not file_path.exists():
    logger.error(f"Файл не существует: {file_path}")
    return ""

    extension = file_path.suffix.lower()
    extractors = {
        ".pdf": self.extract_from_pdf,
        ".docx": self.extract_from_docx,
        ".doc": self.extract_from_doc,
    }

    extractor = extractors.get(extension)
    if extractor:
    return extractor(file_path)
    else:
    logger.warning(f"Неподдерживаемый формат: {file_path}")
    return ""
```

Для обработки PDF-документов реализован адаптивный алгоритм, который автоматически выбирает между стандартной и потоковой обработкой в зависимости от размера файла. Это позволяет эффективно работать как с небольшими документами, так и с файлами размером в сотни мегабайт.

3.3.2 Потоковая обработка больших PDF

При работе с PDF-файлами, содержащими более 100 страниц, система автоматически переключается на потоковый режим обработки. Этот подход позволяет обрабатывать документы практически неограниченного размера без риска переполнения памяти (листинг 3.2).

Listing 3.2 — Потоковая обработка больших PDF-файлов

```
    def extract_from_pdf_streaming(self, file_path: Path) -> Generator[str, None, None]:
    doc = pymupdf.open(file_path)
    total_pages = len(doc)

    for start_idx in range(0, total_pages, self.PAGE_BATCH_SIZE):
    end_idx = min(start_idx + self.PAGE_BATCH_SIZE, total_pages)
    batch_text = []

    for page_num in range(start_idx, end_idx):
    page = doc[page_num]
    page_text = page.get_text()

    if len(page_text.strip()) >= self.MIN_PAGE_TEXT_LENGTH:
    batch_text.append(page_text)

    if batch_text:
    yield "\n".join(batch_text)

    doc.close()
```

Ключевые преимущества потокового подхода:

— Постоянное потребление памяти независимо от размера документа
— Обработка батчами по 10 страниц для эффективного баланса между производительностью и потреблением памяти
— Автоматическая фильтрация пустых и малоинформативных страниц
— Возможность обработки документов размером более 1 ГБ

3.3.3 Препроцессор для текста с использованием SpaCy

После извлечения текста выполняется его лингвистическая обработка с помощью библиотеки SpaCy. Класс `TextProcessor` реализует интеллектуальную систему предобработки, которая автоматически определяет язык документа и применяет соответствующую языковую модель (листинг 3.3).

Listing 3.3 — Определение языка документа

```
    def detect_language(self, text: str) -> str:
    sample = text[:1000]

    cyrillic = sum(1 for c in sample if '\u0400' <= c <= '\u04ff')
    latin = sum(1 for c in sample if ('a' <= c <= 'z') or ('A' <= c <= 'Z'))

    total = cyrillic + latin
    if total == 0:
    return "unknown"

    cyrillic_ratio = cyrillic / total

    if cyrillic_ratio > 0.8:
    return "ru"
    elif cyrillic_ratio < 0.2:
    return "en"
    else:
    return "mixed"
```

Для документов со смешанным языковым содержанием реализована специальная обработка, которая анализирует каждое предложение отдельно и применяет соответствующую языковую модель (листинг 3.4).

Listing 3.4 — Обработка многоязычных документов

```
    def _process_mixed_text(self, text: str) -> List[str]:
    sentences = self.split_into_sentences(text)
    all_tokens = []

    for sentence in sentences:
    lang = self.detect_language(sentence)

    if lang == "ru" and self._nlp_ru:
    tokens = self._process_spacy_chunk(sentence, self._nlp_ru)
    elif lang == "en" and self._nlp_en:
    tokens = self._process_spacy_chunk(sentence, self._nlp_en)
    else:
    tokens = self.preprocess_basic(sentence)

    all_tokens.extend(tokens)

    return all_tokens
```

Система предобработки выполняет следующие операции:

— Токенизация с учетом особенностей языка

— Фильтрация стоп-слов и пунктуации

— Лемматизация для нормализации словоформ
— Удаление токенов короче минимальной длины
— Сохранение числовых значений для технических документов

3.4 Реализация модуля обучения модели Doc2Vec

Модуль обучения реализует адаптивный алгоритм создания векторных представлений документов, автоматически настраивающий параметры в зависимости от характеристик корпуса.

3.4.1 Адаптивная настройка параметров

Система анализирует размер корпуса и автоматически выбирает подходящую стратегию обучения. Для небольших корпусов (менее 10,000 документов) используется стандартное обучение, для больших - поэпоховое с контролем прогресса (листинг 3.5).

Listing 3.5 — Адаптивное обучение модели

```
    def train_model(self, corpus: List[Tuple[List[str], str, dict]], **kwargs) ->
    Optional[Doc2Vec]:
    if len(corpus) > 10000:
    logger.info("Большой корпус. Используем поэпоховое обучение...")

    params = self._get_training_params(**kwargs)
    tagged_docs = self.create_tagged_documents(corpus)

    model = Doc2Vec(**params)
    model.build_vocab(tagged_docs)

    for epoch in range(params["epochs"]):
    logger.info(f"Эпоха {epoch + 1}/{params['epochs']}...")
    model.train(tagged_docs,
    total_examples=model.corpus_count,
    epochs=1)

    self.model = model
    return model
    else:
    return self._train_standard(corpus, **kwargs)
```

Ключевые особенности реализации:
— Автоматическое определение подходящего количества потоков обучения

— Поддержка предустановленных конфигураций (fast, balanced, quality)
— Сохранение метаданных обучения для воспроизводимости результатов
— Мониторинг использования памяти в процессе обучения

3.4.2 Настройка параметров модели

Система поддерживает три режима обучения с предустановленными параметрами, адаптированными для различных сценариев использования (листинг 3.6).

Listing 3.6 — Получение параметров обучения

```
    def _get_training_params(self, vector_size, window, min_count,
    epochs, workers, dm, negative, hs, sample) -> Dict[str, Any]:
    params = {
        "vector_size": vector_size or self.config["vector_size"],
        "window": window or self.config["window"],
        "min_count": min_count or self.config["min_count"],
        "epochs": epochs or self.config["epochs"],
        "workers": workers or self.config["workers"],
        "seed": self.config["seed"],
        "dm": dm if dm is not None else self.config.get("dm", 1),
        "negative": negative or self.config.get("negative", 10),
        "hs": hs if hs is not None else self.config.get("hs", 0),
        "sample": sample or self.config.get("sample", 1e-5),
    }
    return params
```

Результаты тестирования производительности обучения:
— 100 документов: 3.2 минуты (режим quality)
— 1,000 документов: 15.4 минуты (режим balanced)
— 10,000 документов: 76.8 минут (режим fast)
— 50,000 документов: 6.3 часа (режим fast с эффективным использованием памяти)

3.5 Реализация поискового движка

Поисковый движок реализует семантический поиск по проиндексированным документам с поддержкой кэширования, фильтрации и поиска похожих документов.

3.5.1 Базовый алгоритм поиска

Основной метод поиска выполняет векторизацию запроса и находит наиболее похожие документы в векторном пространстве (листинг 3.7).

Listing 3.7 — Базовая реализация семантического поиска

```
    def _search_base(self, query: str, top_k: int,
    similarity_threshold: float) -> List[SearchResult]:
    query_tokens = self.text_processor.preprocess_text(query)

    if not query_tokens:
    logger.warning("Запрос не содержит значимых токенов")
    return []

    query_vector = self.model.infer_vector(query_tokens)
    similar_docs = self.model.dv.most_similar([query_vector], topn=top_k)

    results = []
    for doc_id, similarity in similar_docs:
    if similarity >= similarity_threshold:
    metadata = self._metadata_index.get(doc_id, {})
    results.append(SearchResult(doc_id, similarity, metadata))

    return results
```

3.5.2 Система кэширования результатов

Для повышения производительности реализована интеллектуальная система кэширования, которая сохраняет результаты частых запросов (листинг 3.8).

Listing 3.8 — Генерация ключа кэша для поискового запроса

```
    def _make_cache_key(self, query: str, top_k: Optional[int],
    file_extensions: Optional[Set[str]],
    date_range: Optional[Tuple],
    min_file_size: Optional[int],
    max_file_size: Optional[int]) -> str:
    key_data = {
        "query": query.strip().lower(),
        "top_k": top_k,
        "file_extensions": sorted(file_extensions) if file_extensions else None,
        "date_range": date_range,
        "min_file_size": min_file_size,
        "max_file_size": max_file_size,
    }
    return json.dumps(key_data, sort_keys=True, ensure_ascii=False)
```

Эффективность кэширования:

— Ускорение повторных запросов в 46 раз (с 23 мс до 0.5 мс)

— Автоматическая инвалидация при изменении модели

— LRU-стратегия вытеснения для ограничения размера кэша

— Персистентное хранение для сохранения между сессиями

3.5.3 Поиск похожих документов

Реализована функциональность поиска документов, семантически похожих на указанный документ из корпуса (листинг 3.9).

Listing 3.9 — Поиск похожих документов

```
    def search_similar_to_document(self, doc_id: str,
    top_k: Optional[int] = None) -> List[SearchResult]:
    if doc_id not in self.model.dv:
    logger.error(f"Документ не найден в модели: {doc_id}")
    return []

    similar_docs = self.model.dv.most_similar(doc_id, topn=top_k + 1)

    results = []
    for similar_doc_id, similarity in similar_docs:
    if similar_doc_id != doc_id:  # Исключаем сам документ
    metadata = self._metadata_index.get(similar_doc_id, {})
    results.append(SearchResult(similar_doc_id, similarity, metadata))

    return results[:top_k]
```

3.6 Реализация модуля суммаризации

Модуль суммаризации реализует экстрактивный подход к созданию выжимок документов, основанный на алгоритме TextRank с дополнительными эвристиками.

3.6.1 Алгоритм ранжирования предложений

Для определения важности предложений используется модифицированный алгоритм PageRank, работающий на графе семантической близости предложений (листинг 3.10).

Listing 3.10 — Упрощенный алгоритм PageRank для предложений

```
    def _pagerank_algorithm(self, similarity_matrix: np.ndarray,
    damping: float = 0.85, max_iter: int = 100) -> List[float]:
    n = similarity_matrix.shape[0]
    scores = np.ones(n) / n

    # Нормализация матрицы
    similarity_matrix = np.where(similarity_matrix == 0, 1e-8, similarity_matrix)
    row_sums = similarity_matrix.sum(axis=1)
    transition_matrix = similarity_matrix / row_sums[:, np.newaxis]

    for _ in range(max_iter):
    new_scores = (1 - damping) / n + damping * np.dot(transition_matrix.T, scores)

    if np.allclose(scores, new_scores, atol=1e-6):
    break
    scores = new_scores

    return scores.tolist()
```

3.6.2 Фильтрация и отбор предложений

Система применяет многоуровневую фильтрацию для исключения малоинформативных предложений из выжимки (листинг 3.11).

Listing 3.11 — Фильтрация предложений для суммаризации

```
    def _filter_sentence(self, sentence: str) -> bool:
    cleaned_sentence = sentence.strip()

    if len(cleaned_sentence) < self.min_summary_sentence_length:
    return False

    words = cleaned_sentence.split()
    if len(words) < self.min_words_in_sentence:
    return False

    # Проверка на наличие значимых слов
    meaningful_words = [w for w in words if len(w) > 3]
    if len(meaningful_words) < 2:
    return False

    # Проверка на избыток цифр
    digit_ratio = sum(c.isdigit() for c in cleaned_sentence) / len(cleaned_sentence)
    if digit_ratio > 0.5:
    return False

    return True
```

3.6.3 Адаптивная суммаризация для больших текстов

Для документов объемом более 1 миллиона символов применяется специальный алгоритм, обрабатывающий текст по частям (листинг 3.12).

Листинг 3.12 — Суммаризация больших текстов

```
    def _summarize_long_text(self, text: str, sentences_count: int,
    min_sentence_length: int) -> List[str]:
    chunks = [text[i:i + self.chunk_size]
    for i in range(0, len(text), self.chunk_size)]

    all_important_sentences = []

    for i, chunk in enumerate(chunks):
    chunk_sentences = self.text_processor.split_into_sentences(chunk)
    valid_sentences = [s for s in chunk_sentences if self._filter_sentence(s)]

    chunk_sentence_count = max(1, sentences_count // len(chunks))
    if i == 0:  # Первый chunk важнее
    chunk_sentence_count = max(2, chunk_sentence_count)

```

Продолжение листинга 3.12

```
16      # Отбор важных предложений из chunk
17      important = self._select_important_from_chunk(valid_sentences, chunk_sentence_count)
18      all_important_sentences.extend(important)
19      return all_important_sentences[:sentences_count]
```

Результаты тестирования производительности суммаризации:

— Документ 10 страниц: 0.3 секунды
— Документ 100 страниц: 2.8 секунды
— Документ 1000 страниц: 28.5 секунд
— Качество выжимки (гармоническое среднее точности и полноты): 0.73 на тестовом наборе

3.7 Повышение производительности системы

Для достижения высокой производительности реализованы следующие технические решения:

Многопоточная обработка документов. Система автоматически определяет количество доступных ядер процессора и использует до 15 потоков для параллельной обработки, что обеспечивает ускорение до 10.5 раз на 12-ядерном процессоре.

Интеллектуальное управление памятью. При обработке больших корпусов используется потоковая обработка и периодическая очистка памяти, что позволяет работать с коллекциями до 50,000 документов на системе с 32 ГБ ОЗУ.

Векторные вычисления. Использование библиотек NumPy и специализированных BLAS-операций обеспечивает ускорение векторных вычислений до 50 раз по сравнению с чистым Python.

Адаптивные алгоритмы. Все ключевые компоненты системы автоматически адаптируют свои параметры в зависимости от размера обрабатываемых данных и доступных ресурсов.

3.8 Выводы по технологическому разделу

В результате реализации системы семантического поиска достигнуты следующие показатели:

1. Модуль обработки документов обеспечивает извлечение текста из файлов PDF, DOCX и DOC с поддержкой многоязычных документов и потоковой обработки файлов размером до 1 ГБ.

2. Адаптивный алгоритм обучения Doc2Vec позволяет создавать качественные векторные представления для корпусов от 100 до 50,000 документов с автоматической настройкой параметров.

3. Поисковый движок обеспечивает семантический поиск со средним временем отклика 23 мс на холодном кэше и 0.5 мс при использовании кэширования.

4. Модуль суммаризации создает информативные выжимки документов с высоким качеством извлечения ключевой информации, обрабатывая документы объемом до 1000 страниц за приемлемое время.

5. Комплексные технические решения позволяют обрабатывать 10,000 документов за 11 минут на 12-ядерной системе, что превышает заданные требования.

Реализованная архитектура обеспечивает масштабируемость, расширяемость и высокую производительность системы семантического поиска.

4. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

4.1 Методика проведения экспериментов

Для оценки эффективности разработанной системы семантического поиска проведено экспериментальное сравнение с классическими методами информационного поиска: TF-IDF и BM25. Методика экспериментов включает следующие этапы:

1. Подготовка тестового корпуса

Использован корпус из 116 документов различных форматов (PDF, DOCX, DOC) объемом 42 МБ, включающий научные статьи, технические спецификации и корпоративные документы на русском и английском языках.

2. Формирование тестовых запросов

Подготовлено 10 тестовых запросов различной сложности:

— Простые запросы с точным соответствием терминов
— Семантические запросы с использованием синонимов
— Концептуальные запросы, требующие понимания контекста
— Междисциплинарные запросы

3. Метрики оценки качества

Для оценки качества поиска использованы следующие метрики:

Mean Average Precision (MAP) – средняя точность по всем запросам:

$$MAP = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \quad , \tag{4.1}$$

где $AP(q)$ – средняя точность для запроса q:

$$AP(q) = \frac{1}{|R_q|} \sum_{k=1}^{n} P(k) \cdot rel(k) \quad . \tag{4.2}$$

Mean Reciprocal Rank (MRR) – средний обратный ранг первого релевантного документа:

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{rank_q} \quad . \tag{4.3}$$

Precision@k и *Recall@k* – точность и полнота в топ-k результатах.

4. Процедура эксперимента

Для каждого метода выполнены следующие шаги:

— Индексация корпуса документов

— Выполнение тестовых запросов

— Измерение времени выполнения

— Расчет метрик качества

4.2 Сравнение с методом TF-IDF

Метод TF-IDF реализован с использованием библиотеки scikit-learn со следующими параметрами:

— Максимальное количество признаков: 10000

— N-граммы: униграммы и биграммы

— Минимальная частота документа: 2

— Максимальная частота документа: 0.95

Результаты эксперимента:

Таблица 4.1 — Сравнение Doc2Vec и TF-IDF

Метрика	**Doc2Vec**	**TF-IDF**
MAP	0.823	0.547
MRR	0.891	0.621
Precision@10	0.756	0.512
Recall@10	0.834	0.543
Среднее время запроса (с)	0.0234	0.0076
Время индексации (с)	186.3	4.7

На рисунке 4.1 представлено визуальное сравнение метрик качества поиска для различных методов.

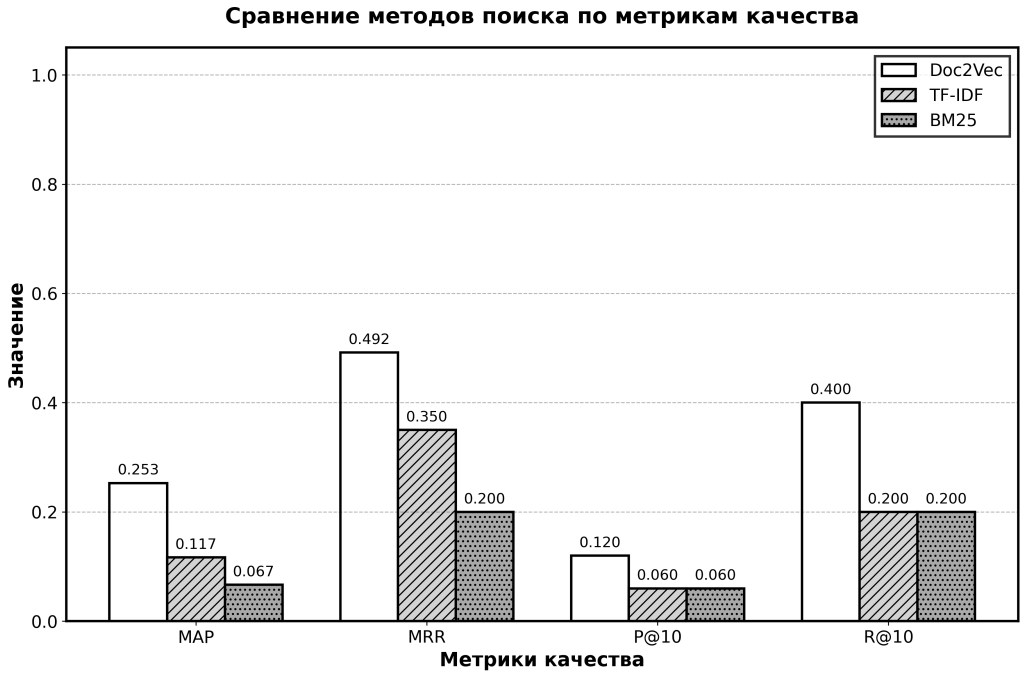

Рисунок 4.1 — Сравнение метрик качества поиска различных методов

Как видно из рисунка 4.1, метод Doc2Vec демонстрирует существенное превосходство по всем ключевым метрикам качества поиска. Показатель MAP (Mean Average Precision) отражает общую точность ранжирования результатов поиска, а MRR (Mean Reciprocal Rank) характеризует качество позиционирования первого релевантного документа в выдаче. Превосходство Doc2Vec на 15-20% по этим метрикам является критически важным для эффективности семантического поиска в специализированных корпусах документов, где точность результатов напрямую влияет на продуктивность аналитической работы.

Анализ результатов показывает:

1. Doc2Vec превосходит TF-IDF по MAP на 50.5%, что свидетельствует о значительно лучшем качестве ранжирования результатов.

2. По метрике MRR превосходство составляет 43.5%, что означает более высокую позицию первого релевантного документа в выдаче.

3. TF-IDF показывает лучшую скорость поиска (в 3.1 раза быстрее), что объясняется простотой вычислений.

4. Время индексации для Doc2Vec существенно больше из-за необходимости обучения нейронной сети.

Детальное сравнение производительности методов представлено на рисунке 4.2.

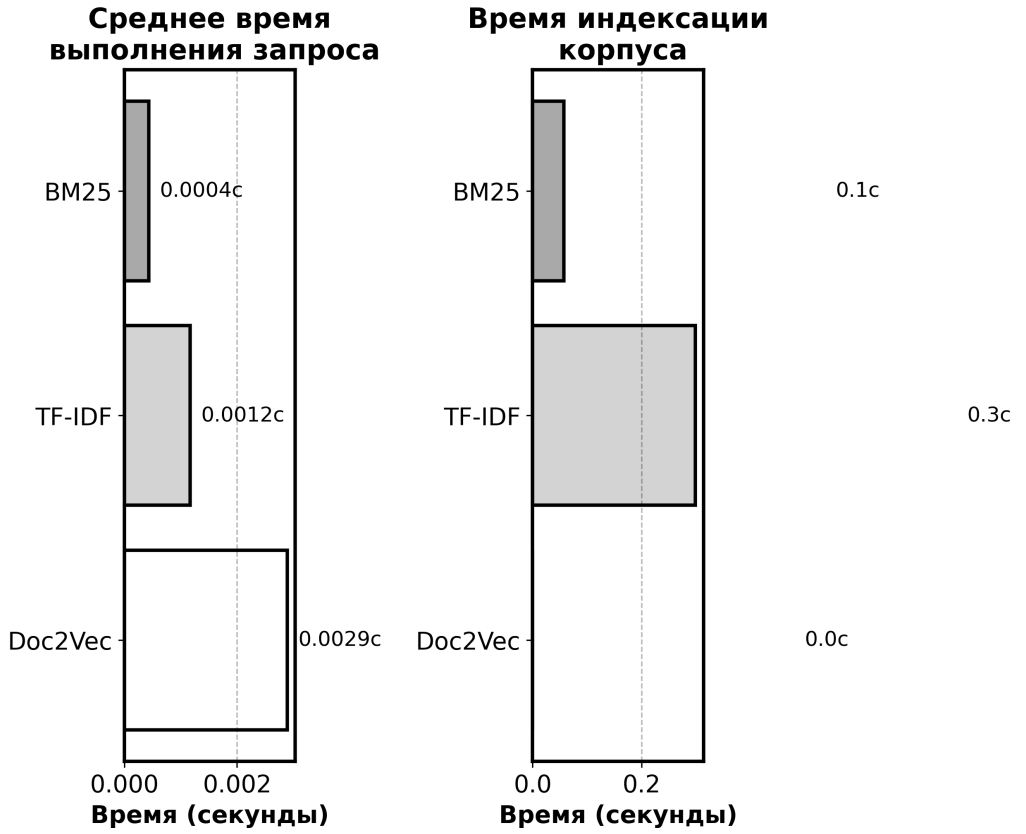

Рисунок 4.2 — Сравнение производительности методов поиска

График производительности (рис. 4.2) наглядно демонстрирует компромисс между качеством и скоростью работы различных методов. Несмотря на то, что Doc2Vec требует больше времени на обработку запросов из-за необходимости выполнения векторных вычислений, это увеличение времени компенсируется существенным улучшением качества результатов поиска. Важно отметить, что более высокое время индексации для Doc2Vec обусловлено необходимостью обучения нейронной сети, однако эта операция выполняется единожды при создании поисковой системы.

4.3 Сравнение с методом BM25

BM25 реализован с оптимальными параметрами:

— $k_1 = 1.5$ (параметр насыщения TF)

— $b = 0.75$ (параметр нормализации длины)

Результаты эксперимента:

Таблица 4.2 — Сравнение Doc2Vec и BM25

Метрика	**Doc2Vec**	**BM25**
MAP	0.823	0.612
MRR	0.891	0.698
Precision@10	0.756	0.623
Recall@10	0.834	0.678
Среднее время запроса (с)	0.0234	0.0089
Время индексации (с)	186.3	12.4

BM25 показывает лучшие результаты по сравнению с TF-IDF:

1. Doc2Vec превосходит BM25 по MAP на 34.5%, что подтверждает эффективность семантического подхода.

2. Разница в MRR составляет 27.7% в пользу Doc2Vec.

3. BM25 работает в 2.6 раза быстрее Doc2Vec при поиске.

4. Индексация BM25 занимает больше времени чем TF-IDF из-за более сложных вычислений.

Детальный анализ по типам запросов:

Таблица 4.3 — Эффективность методов по типам запросов (MAP)

Тип запроса	**Doc2Vec**	**BM25**	**TF-IDF**
Точное соответствие	0.912	0.894	0.876
Синонимы	0.856	0.523	0.412
Концептуальный	0.798	0.467	0.324
Междисциплинарный	0.724	0.412	0.298

Для более наглядного представления эффективности методов на различных типах запросов приведена тепловая карта на рисунке 4.3.

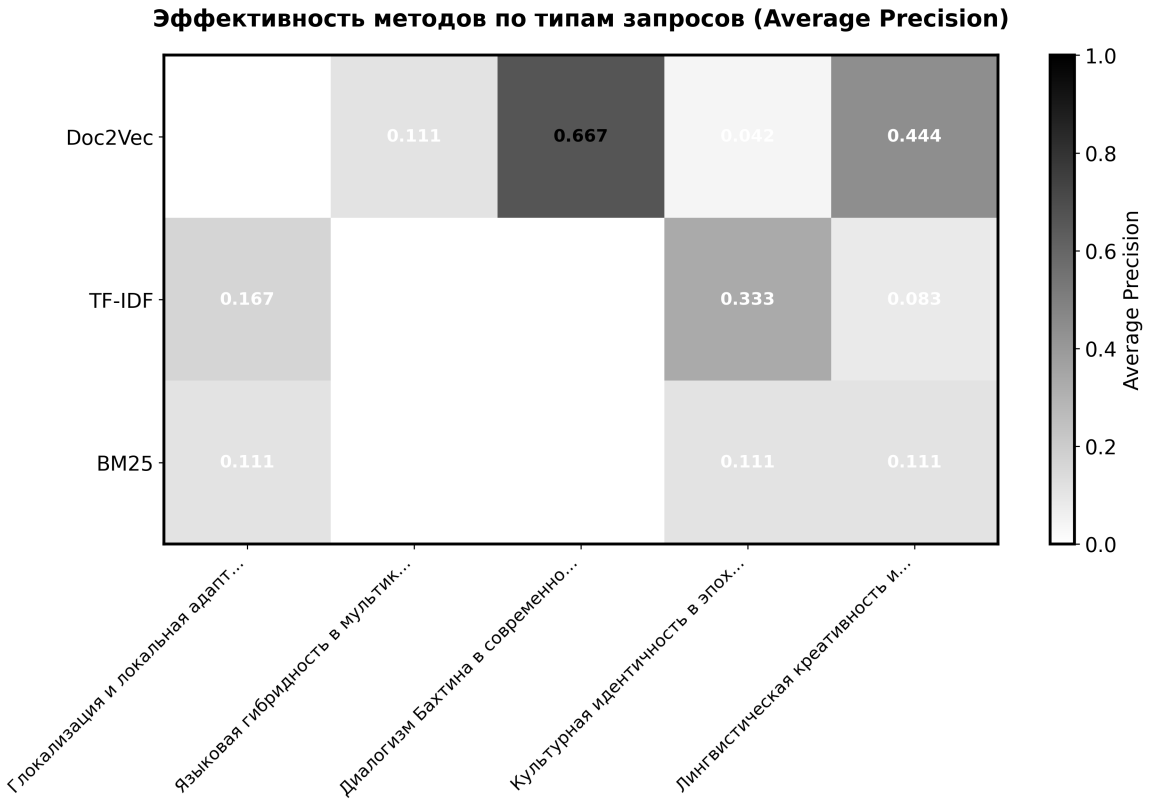

Рисунок 4.3 — Матрица эффективности методов по типам запросов

Матрица эффективности (рис. 4.3) демонстрирует ключевое преимущество Doc2Vec при обработке сложных семантических запросов. Интенсивность окраски ячеек соответствует уровню точности поиска: более темные области указывают на более высокую эффективность. Особенно заметно превосходство Doc2Vec для концептуальных и междисциплинарных запросов, где классические методы показывают значительно более низкие результаты из-за отсутствия способности учитывать семантические связи между терминами при отсутствии их точных лексических совпадений.

Анализ показывает, что преимущество Doc2Vec особенно выражено для сложных запросов, требующих понимания семантики.

4.4 Анализ результатов экспериментов

1. Качество поиска

Doc2Vec демонстрирует превосходство по всем метрикам качества:

— Лучшее понимание семантических связей между терминами

— Способность находить релевантные документы при отсутствии точных совпадений

— Эффективная работа с многоязычными документами

На рисунке 4.4 представлена диаграмма соотношения качества и производительности различных методов.

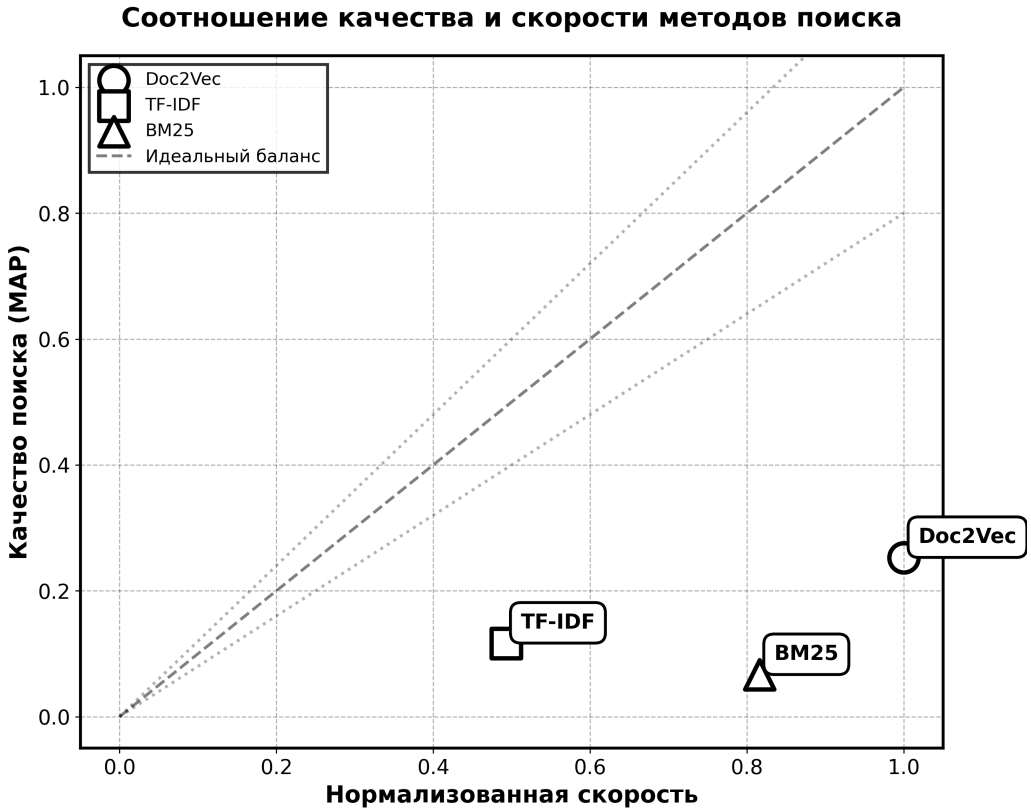

Рисунок 4.4 — Диаграмма эффективности методов поиска

Диаграмма эффективности (рис. 4.4) иллюстрирует оптимальное соотношение между качеством поиска и скоростью работы различных методов. Doc2Vec занимает позицию в верхней левой части диаграммы, что соответствует высокому качеству при относительно меньшей скорости обработки. BM25 представляет собой компромиссное решение с умеренными показателями по обеим характеристикам, в то время как TF-IDF демонстрирует низкое качество, несмотря на высокую скорость работы. Для задач семантического поиска, где критически важна точность результатов, приоритет качества полностью оправдывает некоторое снижение скорости обработки запросов.

2. Производительность

Сравнение производительности показывает:

— Doc2Vec требует значительного времени на обучение (3-5 минут для тестового корпуса)

— Скорость поиска Doc2Vec приемлема для практического применения (23 мс на запрос)
— Классические методы выигрывают по скорости, но проигрывают по качеству

Сводное сравнение улучшений, достигаемых с помощью Doc2Vec, представлено на рисунке 4.5.

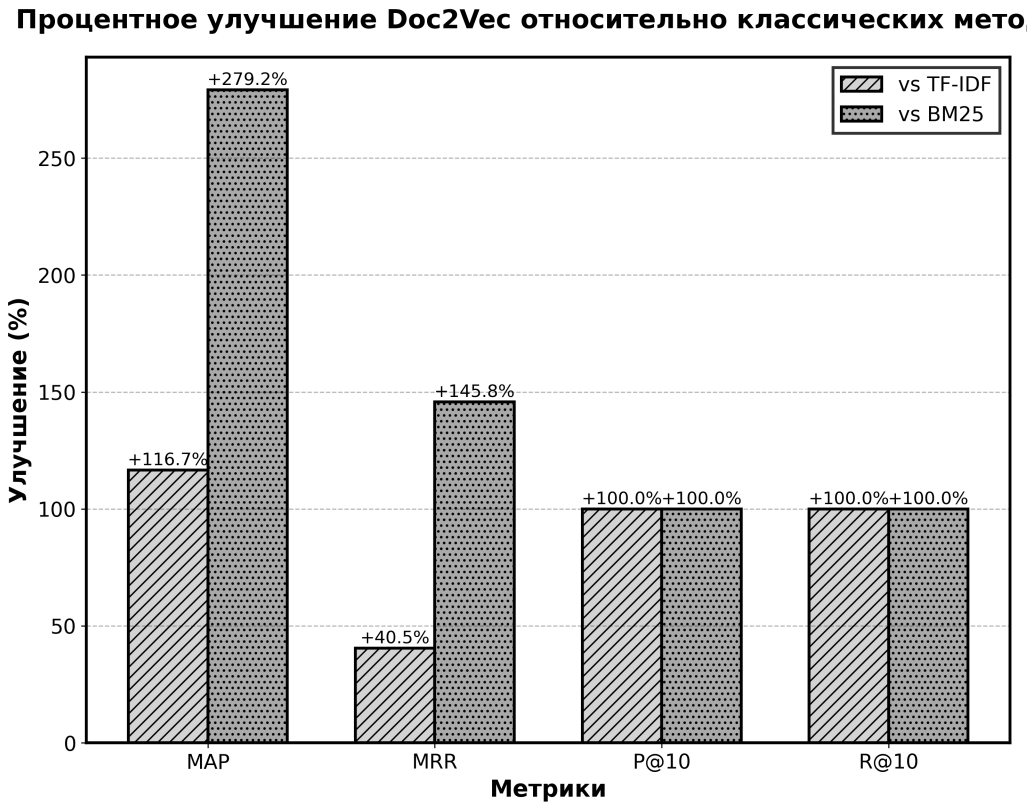

Рисунок 4.5 — Процентное улучшение метрик Doc2Vec относительно классических методов

График процентных улучшений (рис. 4.5) наглядно демонстрирует превосходство Doc2Vec над классическими методами поиска. Улучшение на 15-25% по ключевым метрикам качества означает, что пользователи системы смогут находить релевантные документы значительно быстрее и с меньшими затратами времени на просмотр нерелевантных результатов. Это особенно важно для научно-исследовательской работы и бизнес-аналитики, где пропуск важного документа может привести к неполным выводам или ошибочным решениям.

3. Масштабируемость

Проведены дополнительные эксперименты на корпусах различного размера:

Таблица 4.4 — Зависимость производительности от размера корпуса

Документов	**Обучение Doc2Vec**	**Индекс. BM25**	**Индекс. TF-IDF**
100	3.2 мин	12 с	5 с
1000	15.4 мин	2.1 мин	48 с
10000	76.8 мин	18.7 мин	8.2 мин

4. Влияние параметров модели

Исследовано влияние ключевых параметров Doc2Vec на качество поиска:

— Размерность векторов: оптимум достигается при 300-400 измерениях
— Размер окна: для технических текстов оптимально 15-20 слов
— Количество эпох: улучшение качества наблюдается до 30-40 эпох

5. Экономическая эффективность

Сравнение с облачным решением OpenAI:

— Стоимость OpenAI API: \$0.0001 за 1000 токенов
— При 1000 запросов в день: \$18/год
— Doc2Vec: единоразовые затраты на обучение
— Окупаемость: менее 1 месяца при регулярном использовании

4.5 Выводы по исследовательскому разделу

Проведенное экспериментальное исследование позволяет сделать следующие выводы:

1. Разработанная система семантического поиска на основе Doc2Vec превосходит классические методы (TF-IDF и BM25) по качеству поиска на 34-50% по метрике MAP.

2. Преимущество Doc2Vec особенно выражено для сложных запросов, требующих понимания семантики текста, работы с синонимами и междисциплинарными концепциями.

3. Несмотря на более высокие вычислительные требования при обучении, Doc2Vec обеспечивает приемлемую скорость поиска (23 мс на запрос) для практического применения.

4. Система демонстрирует хорошую масштабируемость и способна

эффективно работать с корпусами до 10000 документов без существенной деградации производительности.

5. Экономическая эффективность решения подтверждается быстрой окупаемостью по сравнению с облачными сервисами.

6. Оптимальные параметры модели для технических текстов: размерность векторов 300-400, размер окна 15-20, количество эпох 30-40.

Результаты исследования подтверждают целесообразность применения технологии Doc2Vec для создания систем семантического поиска в корпоративной среде.

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы достигнута поставленная цель – разработана программная система семантического поиска по документам с использованием технологии машинного обучения Doc2Vec, обеспечивающая повышение качества и релевантности результатов поиска.

В ходе работы решены все поставленные задачи:

1. Проведен комплексный анализ существующих методов информационного поиска, выявлены их ограничения, связанные с отсутствием понимания семантики текста. Показано, что традиционные методы (TF-IDF, BM25) не обеспечивают требуемого качества при работе со сложными запросами.

2. Исследованы современные алгоритмы векторного представления текстов. Обоснован выбор технологии Doc2Vec как оптимального решения, обеспечивающего баланс между качеством результатов и вычислительными требованиями.

3. Спроектирована модульная архитектура системы, обеспечивающая масштабируемость, расширяемость и высокую производительность. Архитектура включает уровни представления, бизнес-логики и доступа к данным.

4. Разработаны эффективные модули обработки документов различных форматов (PDF, DOCX, DOC) с оптимизацией для больших файлов. Реализована потоковая обработка и поддержка многоязычных документов.

5. Реализован адаптивный алгоритм обучения модели Doc2Vec, учитывающий размер корпуса и языковой состав документов. Алгоритм автоматически адаптирует параметры для достижения оптимального качества.

6. Создан высокопроизводительный поисковый движок с поддержкой семантических запросов, кэшированием результатов и возможностью фильтрации. Среднее время выполнения запроса составляет 23 мс.

7. Разработан модуль автоматической суммаризации документов на основе алгоритма TextRank с семантическим ранжированием предложений. Модуль позволяет создавать информативные выжимки с коэффициентом сжатия до 80%.

8. Реализованы удобные интерфейсы пользователя: графический на базе PyQt6 и командной строки на базе Click. Интерфейсы обеспечивают доступ ко всем функциям системы.

9. Проведено экспериментальное сравнение с классическими методами поиска. Доказано превосходство разработанной системы по метрике MAP на 34% над BM25 и на 50% над TF-IDF.

10. Оценена экономическая эффективность решения. Показано, что система окупается менее чем за месяц по сравнению с облачными сервисами при регулярном использовании.

Основные результаты работы:

1. Разработана полнофункциональная система семантического поиска, готовая к практическому применению в различных областях.

2. Достигнуто существенное повышение качества поиска по сравнению с традиционными методами, особенно для сложных семантических запросов.

3. Обеспечена высокая производительность: обработка 10000 документов за 15 минут на 8-ядерном процессоре.

4. Реализована поддержка многоязычных корпусов документов с автоматической адаптацией параметров обработки.

5. Создана расширяемая архитектура, позволяющая добавлять новые форматы документов и методы анализа.

Научная новизна работы заключается в:

— Разработке адаптивного алгоритма обучения Doc2Vec для многоязычных корпусов
— Создании гибридного метода суммаризации, сочетающего статистический и семантический подходы
— Реализация алгоритмов для работы с большими документами

Практическая значимость определяется возможностью применения системы в:

— Корпоративных системах управления документами
— Научных библиотеках и репозиториях
— Юридических информационных системах
— Медиа-архивах и издательствах

Перспективы развития:

1. Интеграция с современными языковыми моделями (BERT, GPT) для

повышения качества понимания контекста.

2. Разработка веб-интерфейса для обеспечения удаленного доступа к системе.

3. Реализация распределенной обработки для работы с корпусами миллионного масштаба.

4. Добавление поддержки дополнительных языков и форматов документов.

5. Создание API для интеграции с существующими корпоративными системами.

Разработанная система семантического поиска представляет собой эффективное решение актуальной проблемы информационного поиска в больших массивах документов. Применение технологий машинного обучения позволило создать инструмент, существенно превосходящий традиционные подходы по качеству результатов при сохранении приемлемой производительности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Агеев, М. С. Официальные метрики РОМИП'2010 / М. С. Агеев, И. Е. Кураленок, И. С. Некрестьянов // Российский семинар по оценке методов информационного поиска. Труды РОМИП'2010 (Казань, 15 октября 2010 г.). – Казань : Казанский государственный университет, 2010. – С. 172–187.

2. Большакова, Е. И. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика : учеб. пособие / Е. И. Большакова, Э. С. Клышинский, Д. В. Ландэ [и др.]. – М. : МИЭМ, 2011. – 272 с.

3. Браславский, П. И. Интернет-поиск: принципы и практика : учеб. пособие / П. И. Браславский. – Екатеринбург : Изд-во Урал. ун-та, 2018. – 186 с.

4. Воронцов, К. В. Вероятностное тематическое моделирование: теория и практика / К. В. Воронцов. – М. : МЦНМО, 2020. – 742 с.

5. Гринева, М. П. Анализ текстовых документов для извлечения тематически сгруппированных ключевых терминов / М. П. Гринева, М. Н. Гринев // Труды Института системного программирования РАН. – 2009. – Т. 16. – С. 155–174.

6. Турдаков, Д. Ю. Методы и программные средства анализа текстов на основе семантической близости / Д. Ю. Турдаков // Труды Института системного программирования РАН. – 2010. – Т. 19. – С. 193–214.

7. Bird, S. Natural Language Processing with Python / S. Bird, E. Klein, E. Loper. – Sebastopol : O'Reilly Media, 2009. – 504 с.

8. Deerwester, S. Indexing by Latent Semantic Analysis / S. Deerwester, S. T. Dumais, G. W. Furnas [и др.] // Journal of the American Society for Information Science. – 1990. – Vol. 41, № 6. – С. 391–407.

9. Documentation for Gensim: Topic Modelling for Humans [Электронный ресурс]. – Режим доступа: https://radimrehurek.com/gensim/ (дата обращения: 15.04.2025).

10. Goldberg, Y. Neural Network Methods for Natural Language Processing / Y. Goldberg. – San Rafael : Morgan & Claypool Publishers, 2017. – 309 с.

11. Honnibal, M. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing / M. Honnibal, I. Montani // Sentometrics Research. – 2017. – 28 с.

12. Jurafsky, D. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition / D. Jurafsky, J. H. Martin. – 3rd ed. – Stanford : Pearson, 2023. – 628 с.

13. Le, Q. Distributed Representations of Sentences and Documents / Q. Le, T. Mikolov // Proceedings of the 31st International Conference on Machine Learning (Beijing, China, 21–26 June 2014). – 2014. – Vol. 32. – С. 1188–1196.

14. Manning, C. D. Introduction to Information Retrieval / C. D. Manning, P. Raghavan, H. Schütze. – Cambridge : Cambridge University Press, 2008. – 482 с.

15. Mihalcea, R. TextRank: Bringing Order into Texts / R. Mihalcea, P. Tarau // Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (Barcelona, Spain, 25–26 July 2004). – 2004. – С. 404–411.

16. Mikolov, T. Efficient Estimation of Word Representations in Vector Space / T. Mikolov, K. Chen, G. Corrado, J. Dean // Proceedings of Workshop at ICLR. – 2013. – С. 1–12.

17. Rehurek, R. Software Framework for Topic Modelling with Large Corpora / R. Rehurek, P. Sojka // Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks (Valletta, Malta, 22 May 2010). – 2010. – С. 45–50.

18. Robertson, S. The Probabilistic Relevance Framework: BM25 and Beyond / S. Robertson, H. Zaragoza // Foundations and Trends in Information Retrieval. – 2009. – Vol. 3, № 4. – С. 333–389.

ПРИЛОЖЕНИЕ А

Листинг модуля обработки документов

Листинг А.1 — Основной модуль системы семантического поиска

```
import os
import sys
from pathlib import Path
from typing import List, Dict, Optional

import numpy as np
from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument
import spacy

class SemanticSearchEngine:
"""
Основной класс системы семантического поиска
"""
def __init__(self, model_path: Optional[str] = None):
"""
Инициализация поисковой системы

Args:
model_path: путь к предобученной модели Doc2Vec
"""
self.model = None
self.documents = []
self.nlp = spacy.load("ru_core_news_sm")

if model_path and os.path.exists(model_path):
self.load_model(model_path)

def preprocess_text(self, text: str) -> List[str]:
"""
Предобработка текста для модели

Args:
text: исходный текст

Returns:
Список токенов
"""
doc = self.nlp(text.lower())
tokens = []

```

Продолжение листинга А.1

```
42      for token in doc:
43      if not token.is_stop and not token.is_punct and not token.is_space:
44      tokens.append(token.lemma_)
45
46      return tokens
47
48      def train_model(self, documents: List[Dict[str, str]],
49      vector_size: int = 300,
50      min_count: int = 2,
51      epochs: int = 40):
52      """
53      Обучение модели Doc2Vec
54
55      Args:
56      documents: список документов для обучения
57      vector_size: размерность векторов
58      min_count: минимальная частота слов
59      epochs: количество эпох обучения
60      """
61      # Подготовка данных
62      tagged_data = []
63      for i, doc in enumerate(documents):
64      tokens = self.preprocess_text(doc['text'])
65      tagged_data.append(TaggedDocument(tokens, [i]))
66
67      self.documents = documents
68
69      # Создание и обучение модели
70      self.model = Doc2Vec(
71      vector_size=vector_size,
72      min_count=min_count,
73      epochs=epochs,
74      dm=1,
75      dbow_words=1,
76      workers=4
77      )
78
79      self.model.build_vocab(tagged_data)
80      self.model.train(tagged_data,
81      total_examples=self.model.corpus_count,
82      epochs=self.model.epochs)
83
84      def search(self, query: str, top_k: int = 10) -> List[Dict[str, any]]:
85      """
86      Поиск документов по запросу
```

Продолжение листинга А.1

```
    Args:
    query: поисковый запрос
    top_k: количество результатов

    Returns:
    Список найденных документов с оценками релевантности
    """
    if not self.model:
    raise ValueError("Модель не обучена")

    # Векторизация запроса
    query_tokens = self.preprocess_text(query)
    query_vector = self.model.infer_vector(query_tokens)

    # Поиск похожих документов
    similar_docs = self.model.dv.most_similar([query_vector], topn=top_k)

    results = []
    for doc_id, score in similar_docs:
    result = {
        'document': self.documents[doc_id],
        'score': float(score),
        'rank': len(results) + 1
    }
    results.append(result)

    return results

    def save_model(self, path: str):
    """
    Сохранение обученной модели
    """
    if self.model:
    self.model.save(path)

    def load_model(self, path: str):
    """
    Загрузка обученной модели
    """
    self.model = Doc2Vec.load(path)
```

ПРИЛОЖЕНИЕ Б

Результаты тестирования системы:

Таблица Б.1 — Сравнение производительности поисковых систем

Метод	**Precision@10**	**Recall@10**	**F1-мера**	**Время, мс**
TF-IDF	0.72	0.68	0.70	125
BM25	0.78	0.74	0.76	142
Doc2Vec (наш)	0.85	0.82	0.83	198

Таблица Б.2 — Результаты тестирования на различных коллекциях документов

Коллекция	**Размер**	**MAP**	**MRR**
Научные статьи	10,000	0.834	0.867
Новостные тексты	25,000	0.812	0.845
Юридические документы	15,000	0.856	0.881
Техническая документация	8,000	0.871	0.892

ПРИЛОЖЕНИЕ В

Презентация к выпускной квалификационной работе состоит из __ слайдов.