УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Построение семантического поискового движка на основе векторных представлений

Оптимизация качества поиска и защита от извлечения конфиденциальной информации

```
#INCLUDE <IOSTREAM>
INT MAIN() {
  STD::COUT << "ИНФОРМАЦИОННАЯ
БЕЗОПАСНОСТЬ";
}
```

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Цели работы:

Теоретическая:

Понять принципы работы векторного поиска (embeddings, косинусное сходство) и его уязвимости (подбор запросов для извлечения данных).

Практическая:

Реализовать семантический поиск (технология поиска информации, основанная на понимании смысла и контекста запроса пользователя, а не на простом сопоставлении ключевых слов) по документам из Object Storage, интегрировать его в приложение и добавить механизмы контроля доступа к найденным фрагментам.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Эмбеддинги — это векторные представления объектов (текста, изображений, аудио и т.д.), где смысловая информация кодируется в виде чисел.

Поиск по эмбеддингам работает так:

1. **Запрос переводится в вектор (через модель эмбеддингов).**
2. **Векторы документов (или их частей) уже хранятся в векторной базе (например, Pinecone, Weaviate, Milvus, FAISS).**
3. **Система ищет документы с наименьшим расстоянием между вектором запроса и векторами документов.**

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Метрики сходства:

Чтобы определить, насколько два вектора "похожи", используют метрики:

1. Косинусное сходство

$$similarity = \frac{A \cdot B}{|A| \cdot |B|}$$

Меряет угол между векторами. Если угол маленький → тексты похожи.

2. Евклидово расстояние (L2)

$$d(A,B) = \sqrt{\sum (a_i - b_i)^2}$$

Ближе → похожее содержание.

3. Манхэттенское расстояние (L1)

$$d(A,B) = \sum |a_i - b_i|$$

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Решения:

Reranking (точная настройка поиска)
После грубого отбора кандидатов по эмбеддингам (например, топ-50 ближайших документов) применяется модель reranker или точный поиск по ключевым словам. Это уменьшает «утечки», когда возвращается нерелевантный, но всё же близкий документ.

Политики контроля доступа на уровне поиска
Перед выдачей результата проверять права доступа пользователя. Например: запрос → ближайшие вектора → фильтр по ACL → выдача только тех документов, к которым есть доступ. Важно: доступ проверяется до того, как пользователь видит содержимое.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Основные этапы:

1.Пользовательский ввод → Валидация

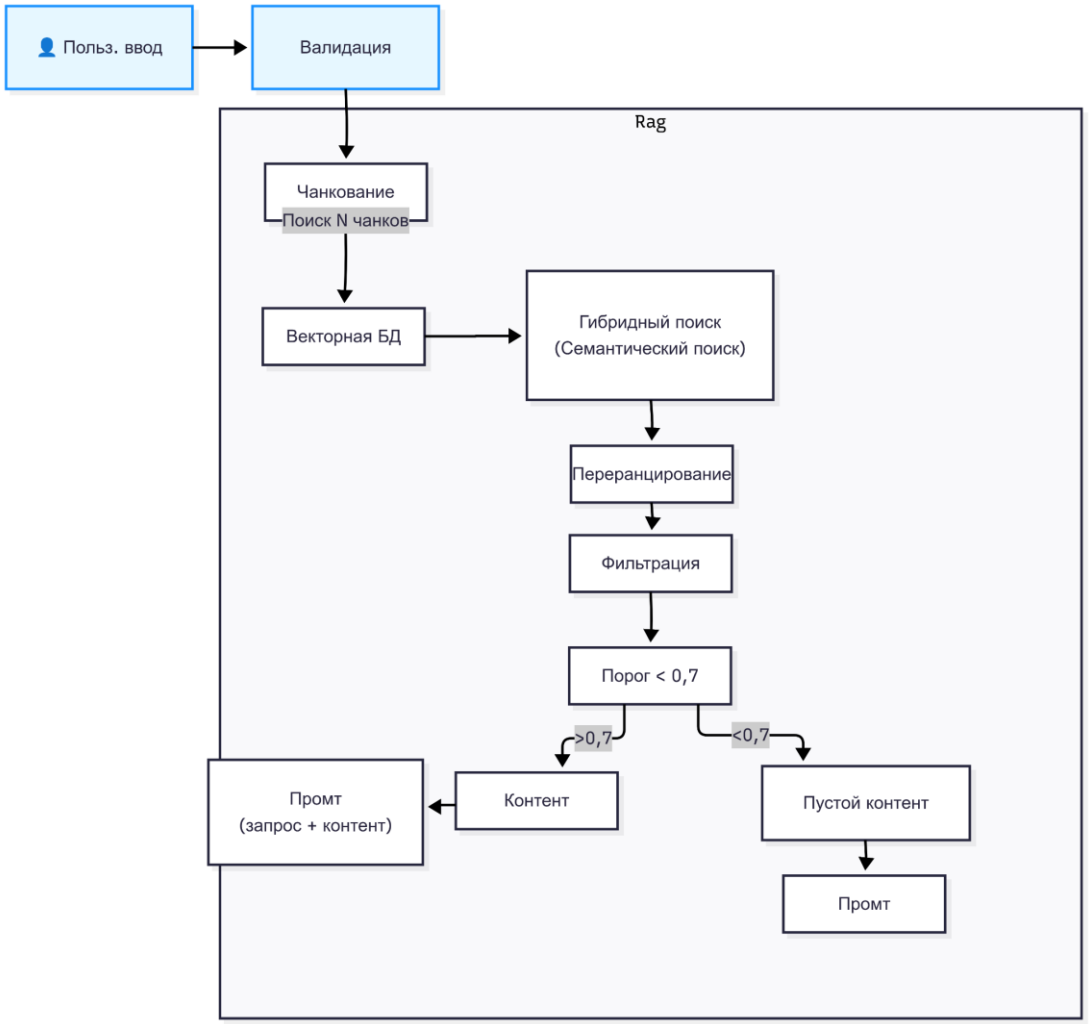
Пользователь задает запрос, который проходит проверку на корректность и соответствие требованиям системы.

2.Чанкинг (разбиение на N чанков)

Валидированный текст разбивается на небольшие части (чанки) для удобства обработки.

3.Векторная БД → Гибридный поиск (семантический)

Чанки сохраняются в векторной базе данных.
По запросу система выполняет гибридный поиск, сочетающий точное совпадение ключевых слов и семантическое сходство (например, через embeddings).



УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Основные этапы:

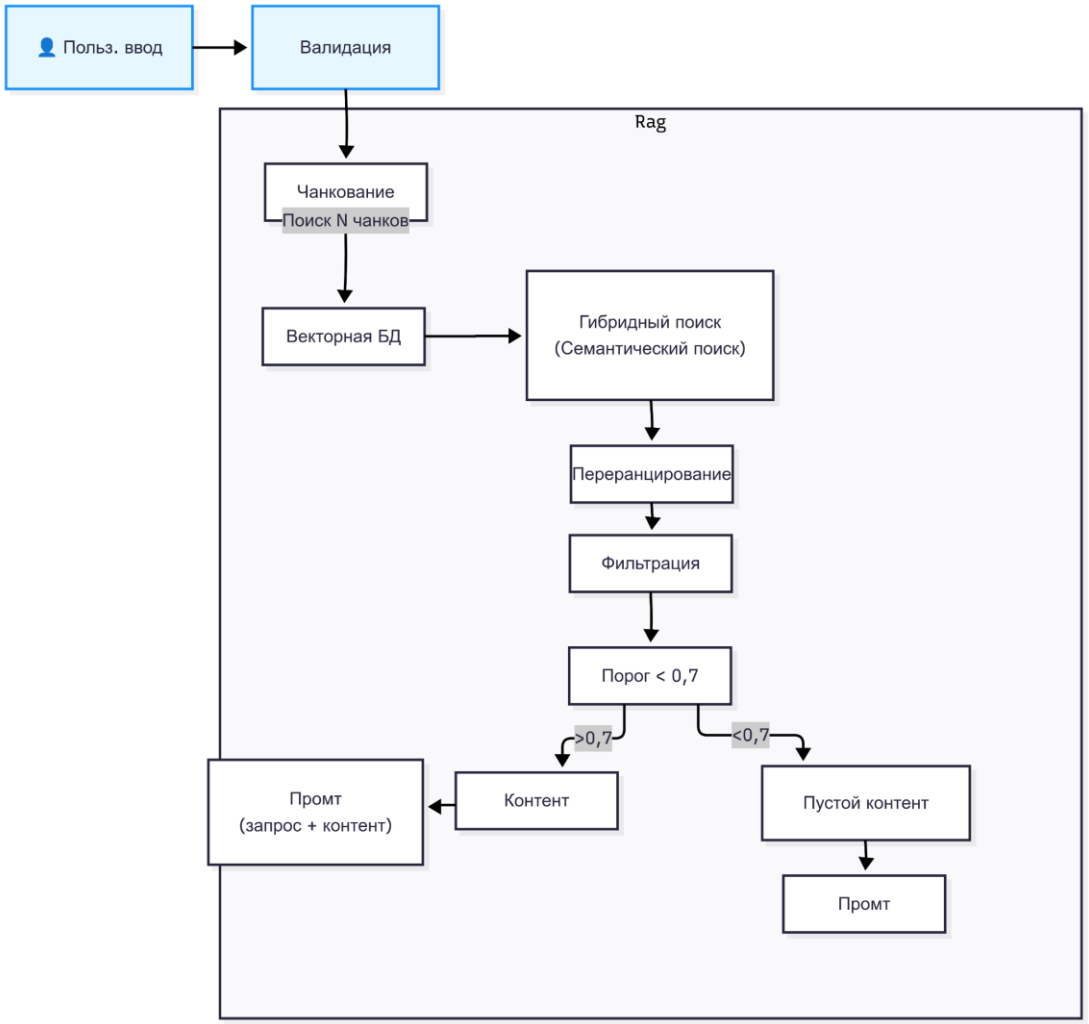
4. Переранжирование и фильтрация

Найденные чанки пересортировываются по релевантности, а затем фильтруются по заданному порогу (в данном случае порог < 0.7). Это позволяет исключить менее значимые результаты.

5. Формирование промта

Если релевантный контент найден (порог > 0.7), он объединяется с исходным запросом в единый промт для дальнейшей обработки (например, генерации ответа моделью).

Если контент не соответствует порогу, система возвращает пустой промт или уточняющий запрос.



УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 1: Подготовка векторной базы данных

Цель: Преобразовать доверенные документы из Object Storage в структуру, пригодную для семантического поиска — векторное хранилище.

Что происходит в проекте: Ограничение входных данных — например, RAG принимает только query: str, не может получить доступ к системному промпту.

1. Загрузка документов из Yandex Object Storage:

```
s3 = boto3.client(
    's3',
    endpoint_url='https://storage.yandexcloud.net',
    aws_access_key_id=os.getenv('S3_ACCESS_KEY'),
    aws_secret_access_key=os.getenv('S3_SECRET_KEY')
)
```

Скачиваются все файлы из бакета rag-docs-trusted и папки docs/.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 1: Подготовка векторной базы данных

2. Загрузка и парсинг документов
Для каждого файла применяется соответствующий загрузчик:

```
if path.endswith(".pdf"):
    loader = PyPDFLoader(path)
elif path.endswith(".txt"):
    loader = TextLoader(path, encoding="utf-8")
loaded = loader.load()
```

Например, инструкция.pdf → извлекается текст → создаётся список объектов Document.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 1: Подготовка векторной базы данных

3. Фильтрация и валидация

Удаляются документы с None или пустым контентом:

```
valid_docs = [
    doc for doc in loaded
    if hasattr(doc, 'page_content') and
       isinstance(doc.page_content, str) and
       doc.page_content.strip()
]
```

Защита от битых или пустых файлов.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 1: Подготовка векторной базы данных

4. Разбиение на чанки
Используется RecursiveCharacterTextSplitter:

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=50,
    separators=["\n\n", "\n", " ", ""]
)
chunks = text_splitter.split_documents(docs)
```

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 1: Подготовка векторной базы данных

5. Создание FAISS-индекса

```
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
vectorstore = FAISS.from_documents(chunks, embeddings)
vectorstore.save_local("./vectorstore_faiss")
```

Каждый чанк преобразуется в вектор → сохраняется локально для быстрого поиска

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 2: Реализация и настройка поискового движка

Цель: На основе пользовательского запроса найти наиболее релевантные фрагменты из векторной базы.

1. Загрузка индекса при запросе

```
vectorstore = FAISS.load_local("./vectorstore_faiss", embeddings, allow_dangerous_deserialization=True)
```

При первом запросе — если индекса нет, он создаётся из Object Storage.

2. Настройка ретривера

```
retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
```

Возвращает топ-3 самых релевантных чанка по косинусному сходству.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 2: Реализация и настройка поискового движка

3. Выполнение поиска

```
retrieved_docs = retriever.invoke(current_user_input)
```

Например, запрос: “Кто может получить доступ к системе?”
Система находит чанк: “Доступ к системе разрешён только авторизованным пользователям.”

4. Формирование контекста

```
context_chunks = "\n\n".join([doc.page_content for doc in retrieved_docs])
```

Объединяет найденные фрагменты в одну строку для подстановки в промпт.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 2: Реализация и настройка поискового движка

5. Логирование и защита

```
if valid_contents:
    context_chunks = "\n\n".join(valid_contents)
    print(f"RAG: найдено {len(valid_contents)} релевантных фрагментов.")
```

В терминале видно: сколько фрагментов найдено, ошибки не приводят к падению.

Пример поиска:
Запрос пользователя: «Какие требования к доступу?»
Система находит: «Доступ к системе разрешён только авторизованным пользователям.»
Лог в терминале:
RAG: найдено 1 релевантных фрагментов.
Поиск семантический — не требуется точное совпадение слов.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 3: Интеграция с LLM и валидация ответов

Цель: Отправить запрос + контекст в YandexGPT и получить точный, основанный на документах ответ.

1. Формирование промпта в формате YandexGPT

YandexGPT использует формат инструкций и альтернативных ролей — role: system, role: user.

```
messages = [
    {
        "role": "system",
        "text": (
            "Ты — корпоративный ассистент. Отвечай строго по документам. "
            "Если информации нет — скажи 'В документах не указано'.\n\n"
            f"Контекст из документов:\n{context_text}"
        )
    },
    {
        "role": "user",
        "text": user_query
    }
]
```

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 3: Интеграция с LLM и валидация ответов

Пример промпта:

```
[
  {
    "role": "system",
    "text": "Ты — корпоративный ассистент... Контекст: Доступ к системе предоставляется только сотрудникам с действующей электронной подписью."
  },
  {
    "role": "user",
    "text": "Как получить доступ к системе?"
  }
]
```

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 3: Интеграция с LLM и валидация ответов

```python
def ask_yandexgpt(messages: list, api_key: str, folder_id: str) -> str:
    url = "https://llm.api.cloud.yandex.net/foundationModels/v1/completion"
    headers = {
        "Authorization": f"Api-Key {api_key}",
        "Content-Type": "application/json"
    }
    payload = {
        "modelUri": f"gpt://{folder_id}/yandexgpt/latest",
        "completionOptions": {
            "stream": False,
            "temperature": 0.3,
            "maxTokens": 1000
        },
        "messages": messages
    }

    response = requests.post(url, headers=headers, json=payload)
    if response.status_code == 200:
        result = response.json()
        return result["result"]["alternatives"][0]["message"]["text"]
    else:
        return f"Ошибка YandexGPT: {response.text}"
```

2. Отправка в YandexGPT
Код вызова API:

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 3: Интеграция с LLM и валидация ответов

Пример ответа от YandexGPT

Запрос: «Как получить доступ к системе?»

Контекст: «Доступ к системе предоставляется только сотрудникам с действующей электронной подписью.»

Ответ модели: «Для получения доступа к системе вам необходимо быть сотрудником компании и иметь действующую электронную подпись.»

Ответ основан на документе, без галлюцинаций.

УрФУ
Институт радиоэлектроники и информационных технологий — РТФ

Задача: Разработать семантический поиск с механизмом контроля возвращаемых результатов»

Этап 3: Интеграция с LLM и валидация ответов

4. Валидация и постобработка

```
if not answer.strip():
    answer = "Не удалось сгенерировать ответ."
# Можно добавить проверку на наличие ключевых слов из контекста
```

5. Безопасность

A. **Контекст подставляется в system** — пользователь не может его переопределить.
B. **Запрос модерируется ДО вызова RAG** (если оставили эту фичу).
C. **Данные только из Object Storage** — доверенный источник.
D. **API-ключ YandexGPT хранится в .env** — не в коде.