

Научная статья

УДК 004.032.26

DOI 10.25205/1818-7900-2025-23-1-46-66

Об одном подходе к разработке информационно-справочных систем на основе больших языковых моделей

Семен Евгеньевич Попов¹, Вадим Петрович Потапов²
Роман Юрьевич Замараев³

Федеральный исследовательский центр информационных и вычислительных технологий
Новосибирск, Россия

¹popov@ict.nsc.ru, <https://orcid.org/0000-0002-4763-2334>

²vadimptpv@gmail.com, <https://orcid.org/0000-0001-6959-3324>

³zamaraev@ict.nsc.ru, <https://orcid.org/0000-0003-4822-4794>

Аннотация

В статье рассматривается современный подход к разработке контекстно-вопросно-ответной системы, направленной на обеспечение информационной поддержки процессов территориального управления. Основой системы являются большие языковые модели, использующие подход Retrieval-Augmented Generation (RAG). Источником данных для базы знаний послужили официальные доклады «О состоянии окружающей среды», включающие текстовую и табличную информацию. Для эффективного хранения и быстрого доступа к данным использовалась векторная база данных PostgreSQL с расширением pgvector. Основной компонент системы – RAG-модель, включающая Bi- и Cross-энкодеры, которые обеспечивают качественную обработку контекста и точный подбор релевантных ответов. Серверная часть системы разработана с использованием платформы Text Generation Interface, а клиентская часть реализована на базе ReactJS и Telegram-бота, что обеспечило удобство взаимодействия с пользователями. Результаты работы демонстрируют, что предложенный подход позволяет повысить точность и эффективность ответов за счет использования фильтров классификации запросов, адаптации временных интервалов и быстрого поиска релевантной информации. Разработанный графический интерфейс делает систему доступной для широкого круга пользователей. Полученные результаты подтверждают эффективность внедрения современных технологий искусственного интеллекта в задачи территориального управления. В перспективе предложенная система может быть адаптирована для других прикладных областей.

Ключевые слова

большая языковая модель, контекстно-вопросно-ответная система, чат-бот, векторная база данных, RAG, Cross-Encoder, Bi-Encoder

Финансирование

Исследование выполнено в рамках государственного задания Минобрнауки России для Федерального исследовательского центра информационных и вычислительных технологий.

Для цитирования

Попов С. Е., Потапов В. П., Замараев Р. Ю. Об одном подходе к разработке информационно-справочных систем на основе больших языковых моделей // Вестник НГУ. Серия: Информационные технологии. 2025. Т. 23, № 1. С. 46–66. DOI 10.25205/1818-7900-2025-23-1-46-66

© Попов С. Е., Потапов В. П., Замараев Р. Ю., 2025

On an Approach to Developing Information and Reference Systems Based on Large Language Models

Semyon E. Popov¹, Vadim P. Potapov²
Roman Y. Zamaraev³

Federal Research Center for Information and Computational Technologies
Novosibirsk, Russian Federation

¹popov@ict.nsc.ru, <https://orcid.org/0000-0002-4763-2334>

²vadimptpv@gmail.com, <https://orcid.org/0000-0001-6959-3324>

³zamaraev@ict.nsc.ru, <https://orcid.org/0000-0003-4822-4794>

Abstract

The aim of this study is to develop a corporate context-aware question-answering system in the form of a chatbot to support territorial management by providing fast and accurate access to relevant information. The system is built upon large language models leveraging the Retrieval-Augmented Generation (RAG) approach, combined with modern data processing and retrieval techniques. The knowledge base of the system incorporates textual and tabular data extracted from official reports on environmental conditions. A PostgreSQL database with the pgvector extension was employed to store and retrieve large volumes of vectorized data efficiently. Advanced Bi- and Cross-encoder architectures were integrated to enhance the accuracy of context interpretation and the relevance of retrieved answers. The system's back-end was implemented using the Text Generation Interface platform, while the frontend relies on ReactJS and a Telegram bot, ensuring an intuitive user interface for diverse audiences. Additional features, such as query classification filters and temporal interval normalization, were introduced to refine the system's response precision and user experience. The results demonstrate the effectiveness of combining state-of-the-art artificial intelligence technologies with robust data management solutions. The developed system not only improves response accuracy and retrieval efficiency but also offers scalability and adaptability for application in various domains. This research highlights the potential of AI-driven solutions in addressing complex challenges in territorial management and beyond.

Keywords

Large Language Model (LLM), Contextual Question-Answering System, Chatbot, Vector Database, Retrieval-Augmented Generation (RAG), Cross-Encoder, Bi-Encoder

Funding

The research was carried out within the state assignment of Ministry of Science and Higher Education of the Russian Federation for Federal Research Center for Information and Computational Technologies.

For citation

Popov S. E., Potapov V. P., Zamaraev R. Y. On an Approach to Developing Information and Reference Systems Based on Large Language Models. *Vestnik NSU. Series: Information Technologies*, 2025, vol. 23, no. 1, pp. 46–66 (in Russ.) DOI 10.25205/1818-7900-2025-23-1-46-66

Введение

Компьютерные информационно-справочные системы (ИСС), реализующие сбор, хранение, обработку и поиск текстовой, мультимедийной информации и данных составляют подавляющую часть программных решений в сети Интернет и в корпоративных сетях, и значительную часть на компьютерах и гаджетах конечных пользователей.

Основной функционал ИСС со стороны пользователя – это поисковая (под)система, состоящая из алгоритмов и программного обеспечения, предоставляющих возможность доступа к обширной коллекции как структурированной, так и неструктурированной информации (текст, изображение, звук) с помощью запросов [1].

Ключевым критерием качества работы поисковой системы являются релевантность – это степень соответствия документа запросу пользователя. С со стороны пользователя это означает, что в возвращенном поисковой системой списке потенциально полезных документов наиболее подходящий запросу документ должен находиться как можно выше, оптимально – первым.

В настоящее время классический, «документальный» подход к поиску известен, прежде всего, в публичных реализациях от IT-гигантов (Google, Microsoft, Yandex и др.) для работы с глобальной сетью Интернет. Однако создание и использование специализированных (корпоративных), частично или полностью изолированных (конфиденциальных) ИСС имеет ряд особенностей, в том числе необходимость получения обобщений и заключений для поддержки принятия решений.

Прежде всего для эффективного поиска документов необходимо, чтобы пользователь обладал достаточной компетенцией по теме запроса. Это позволяет сформулировать запрос лексически и синтаксически аналогично текстам в известных поисковой системе документах, что, в свою очередь, обеспечивает попадание релевантных документов в верхнюю часть списка в поисковом ответе. Также компетенция необходима, чтобы из нескольких документов со схожим содержанием выбрать наиболее адекватные задаче поиска. Кроме того, потребуются усилия, чтобы из многостраничного документа извлечь, в свою очередь, наиболее адекватные разделы и параграфы, и привести их к виду обобщения или заключения.

Эффективность поиска и обобщения в такой ситуации может быть повышена за счет использования:

- контекстов – уточняющих условий (даты, названия и т. п.), ключевых слов и словосочетания, вплоть до небольших фрагментов текста, сужающих область поиска;
- ожидаемых вопросов – подготовленных заранее наиболее популярных вопросов (Frequently Answered Question, FAQ) и готовых решений (How To) по важным темам, к которым подобраны наиболее вероятные вопросы, самые релевантные документы и сделаны обобщения.

Повсеместное внедрение таких решений ограничивалось чрезвычайной трудоемкостью предварительной обработки документов, особенно в части комплектования ожидаемых вопросов, ответов на них и формулировок готовых решений. В то же время как ответ на такую потребность в настоящее время наблюдается сверхбыстрое развитие технологий для работы с большими массивами документов. Эти технологии стали результатом впечатляющих достижений в области анализа данных и текстов на основе больших языковых моделей (БЯМ).

Принципиальным для темы статьи достижением в области БЯМ являются генеративные алгоритмы и сети на их основе [2], которые умеют «приходить» к ответу через цепочку «рассуждений» и «понимают» контекст, в том числе в виде медиаинформации (звук, изображение, видео). Здесь глаголы, используемые для описания рассудочной деятельности, сознательно взяты в кавычки, так как никакого интеллекта в этих алгоритмах, конечно же, нет.

Тем не менее современные генеративные сети (ChatGPT¹, ее клоны и аналоги) [3] с определенного уровня сложности (размера) действительно способны создавать рефераты и производить обобщения текстов не хуже специалиста на основе общих принципов построения предложений и оформления логического заключения, заложенных в письменные языки.

Для вывода этих принципов требуется огромный объем и высокое качество обучающих текстов и сопровождающих их данных. Это неизбежно требует огромных трудозатрат на подготовку текстов и вычислительных мощностей для их анализа – обучения БЯМ.

Выигрыш для разработчика корпоративной ИСС при использовании БЯМ состоит в том, что ее можно использовать (скачать², арендовать) в готовом, обученном виде, и с небольшими затратами доучить на корпоративных документах. При этом автоматически обеспечивается конфиденциальность и учет особенностей корпоративных данных.

Таким образом, актуальной видится задача разработки технологий создания корпоративных ИСС в виде чат-бота на основе генеративных БЯМ с обеспечением конфиденциальности и вычислительной доступности для широкого круга предприятий и организаций, и их сотрудников со средним уровнем компетенций в области анализа данных и БЯМ.

¹ <https://openai.com/index/chatgpt/>

² <https://huggingface.co/models>

Постановки задачи

Разработка корпоративной контекстно-вопросно-ответной системы в виде чат-бота в рамках обеспечения информационной поддержки территориального управления.

Исходные данные

Источником текстовой и табличной информации для базы знаний разработанной системы послужили официальные доклады «О состоянии окружающей среды» Министерства природных ресурсов и экологии Кемеровской и Новосибирской областей. Данные источники являются открытой информацией и расположены в сети Интернет³.

Описание системы

Информационно-справочная система для поддержки принятия решений в задачах экологического мониторинга представляет собой контекстно-вопросно-ответную (КВО) систему, которая использует методы предобработки естественного языка (NLP) и интеграцию больших языковых моделей (LLM) с внешними источниками данных. Основной задачей КВО-системы является предоставление точной и релевантной информации пользователю в ответ на его запросы, учитывая контекст взаимодействия. КВО-система состоит из следующих ключевых элементов:

1. **Модуль обработки запросов.** Распознает и анализирует запрос пользователя. Проводит разбор естественного языка (NLU) для выделения сущностей, намерений и контекста.

2. **Модуль формирования ответов.** Генерирует релевантный и персонализированный ответ на основе запроса пользователя. Обеспечивает учет истории взаимодействий для создания контекстуальных ответов.

3. **Интеграция внешних источников данных.** Обеспечивает доступ к специализированным базам данных, отчетам, научным публикациям и другим источникам экологической информации. Использует поиск информации и систему извлечения знаний (IR и IE).

4. **Интерфейс взаимодействия с пользователем.** Обеспечивает удобное и интуитивно понятное взаимодействие пользователя с системой через текстовый или голосовой интерфейс.

Контекст в рамках КВО-системы – это текстовая информация, содержащая сведения, на основе которых модель должна сформулировать ответ. Вопрос – задание, сформулированное пользователем, на которое модель должна дать ответ. Вопросы могут быть прямыми или сложными, требующими анализа и логического вывода. Ответ – результат, который генерирует модель, интерпретируя контекст и используя информацию из него для ответа на обозначенный в вопрос.

В разработанной системе контексты представляют собой отдельные фрагменты текста, в основном один или несколько связанных по смыслу абзацев текста источника данных (см. раздел «Исходные данные»). Контексты формируют базу знаний КВО. Каждый контекст, по возможности, дополняется метаинформацией, включающей название раздела, подраздела и год отчета (рис. 1)

Для каждого годового доклада контексты сохраняются в соответствии с разделом в отдельный файл, например, для раздела «Климат» из доклада 2014 г. имя файла будет eco-report-2014-weather.pdf.

³ <https://kuzbasseco.ru/doklady/o-sostoyanii-okruzhayushhej-sredy-kemerovskoj-oblasti/>; <https://mpr.nso.ru/page/3736>

```

{
  "section": "климат",
  "subsection": "погода, температура, осадки по месяцам",
  "year": "2014",
  "text": "В мае 2014 года наблюдалась неустойчивая погода с резкими колебаниями температуры и частыми обильными дождями во второй половине месяца. Наиболее теплой была первая декада мая, когда максимальная температура воздуха повышалась до +25, +27 °С. Минимальная температура воздуха понижалась до -1, -6 °С. Среднемесячная температура воздуха составила +7, +9 °С, что на 2 °С ниже нормы. По северным районам области осадков выпало 192-320 % от нормы, в остальных районах месячная сумма осадков составила 123-203 % от нормы."
}

```

Рис. 1. Фрагмент подготовленных контекстов базы знаний в json-формате
 Fig. 1. A fragment of prepared knowledge base contexts in json format

Одним из ключевых элементов контекстно-вопросно-ответной системы является понятие «Промпт» (анг. prompt), а также механизмы его корректного формирования. Промпт используется для управления поведением модели, помогая ей понять задачу, ожидания пользователя и контекст, в рамках которого она должна работать. Контекст является основным строительным блоком промпт-инструкций больших языковых моделей. В рамках разработанной КВО используются промты-инструкции, которые позволяют отвечать на вопрос пользователя только на основе тех текстов базы знаний, к которым у КВО есть доступ.

База знаний

Как указано в разделе «Исходные данные», база знаний формируется на основе докладов, представленных текстовыми файлами в формате docx и pdf.

Для более эффективного хранения контекстной информации, ее метаданных и эмбедингов была предложена концепция использования векторной базы. Векторные базы данных (Vector Databases) – это специализированные системы для хранения, поиска и работы с высокоразмерными эмбедингами, которые часто используются в системах Retrieval-Augmented Generation (RAG). Векторная база данных хранит информацию в виде числовых векторов, представляющих семантическое содержание текстов. Это позволяет эффективно искать документы или фрагменты текста, близкие к заданному запросу по смыслу, даже если ключевые слова между ними различаются. Для создания таких векторов обычно используют модели машинного обучения, такие как BERT, GPT и др. Эти модели обучаются преобразовывать тексты в векторы таким образом, чтобы похожие по смыслу тексты имели схожие векторы. Дополнительно могут храниться метаданные (например, текстовые или числовые атрибуты), которые связаны с каждым вектором для гибридного поиска.

Текстовая и векторная информация в векторной базе данных индексируется. **Индексирование** – это процесс подготовки данных для быстрого поиска ближайших соседей (Approximate Nearest Neighbor Search, ANNs). Выделяют следующие методы индексации: Flat Brute-Force, HNSW (Hierarchical Navigable Small World Graph), IVF (Inverted File Index), LSH (Locality-Sensitive Hashing).

На текущий момент существует большое количество специализированных систем управления базами данных, поддерживающих хранение и обработку векторных представлений для текстовой информации. Ниже приведена таблица с некоторыми из них, с их достоинствами и недостатками (табл. 1.)

Таблица 1

Достоинства и недостатки некоторых векторных баз данных

Table 1

Advantages and disadvantages of some vector databases

Название	Достоинства	Недостатки	Области применения
Milvus⁴	Поддержка больших объемов данных; высокая производительность; интеграция с ML-фреймворками	Требуется настройка инфраструктуры; высокие требования к ресурсам при работе с большими данными	Поиск по изображениям, видео, текстам; рекомендательные системы
Pinecone⁵	Облачное решение; простота в использовании; автоматическое масштабирование	Зависимость от облака; платное решение	Быстрое прототипирование; поиск похожих данных в веб-приложениях
FAISS⁶	Высокая производительность; оптимизация под GPU; поддержка пользовательских индексов	Не поддерживает гибридный поиск; отсутствие встроенных механизмов масштабирования	Локальные проекты; эксперименты с большими наборами данных
ElasticSearch⁷ / OpenSearch⁸	Гибридный поиск (векторы + текст); хорошо известный стек технологий	Ограниченная оптимизация для векторов; трудности с настройкой масштабирования	Универсальные поисковые системы
pgvector⁹	Интеграция с PostgreSQL; простота настройки; подходит для гибридного поиска	Ограниченная производительность на больших объемах; меньшая оптимизация ANNs по сравнению с HNSW	Небольшие RAG-системы; гибридный поиск

Выбор базы данных зависит от конкретных требований объема данных, доступных ресурсов и ожидаемой производительности. Для разработанной системы, принимая во внимание нижеперечисленные преимущества, была выбрана СУБД PostgreSQL с расширением pgvector.

1. Унифицированное хранилище данных позволило использовать единую базу данных PostgreSQL как для регулярных данных системы, так и для векторных данных.

2. Расширенные SQL-возможности СУБД имплементируют комбинированный поиск по векторным данным с традиционными SQL-запросами, включая фильтры, агрегации, соединения, пространственные данные и прочее. Возможность хранить векторы вместе с метаданной (текстовые данные, числовые атрибуты), что упрощает выполнение гибридных запросов.

3. Удалось снизить до минимума зависимости в проекте, так как расширение pgvector не требует установки сложных инструментов, как в случае с отдельными векторными базами. Также позволило использовать настроенными системами мониторинга и резервного копирования корпоративного кластера СУБД.

⁴ <https://milvus.io/>

⁵ <https://www.pinecone.io/>

⁶ <https://github.com/facebookresearch/faiss>

⁷ <https://www.elastic.co/elasticsearch>

⁸ <https://opensearch.org/>

⁹ <https://github.com/pgvector/pgvector>

4. СУБД PostgreSQL, развернутая в кластерном исполнении, практически ни в чем не уступает специализированным решениям в задачах по хранению, извлечению и обслуживанию векторных представлений текстовых данных, например таких, как Elasticsearch или OpenSearch.

5. Расширение pgvector имеет богатые возможности для работы с векторными данными: методы поиска на основе алгоритмов ближайших соседей, таких как L2 (евклидово расстояние), косинусное расстояние и расстояние Минковского; создание индексов для ускорения поиска (например, HNSW); поддержка высокоразмерных векторов, что удобно для работы с эмбедингами создаваемыми современными большими языковыми моделями.

6. Легкость интеграции. Поддержка языка программирования Python через библиотеки, такие как SQLAlchemy, Psycopg и другие ORM.

Таким образом, выбор программного решения pgvector для работы с базой знаний контекстно-вопросно-ответной системы видится вполне актуальным благодаря гибкости, масштабируемости и интеграции в СУБД PostgreSQL, а также расширенным возможностям оперирования векторными данными.

Для хранения эмбедингов была создана таблица «contexts» со следующими столбцами: id – номер записи, тип: index; text – исходный текст, тип: text; year – год доклада, тип: varchar(4); section – название раздела в докладе, тип: varchar(128); subsection – название подраздела в разделах доклада, тип: varchar(256); embeddings – столбец, содержащий массивы эмбедингов, тип: vector(1024).

Для создания эмбедингов использовалась модель d0rj/e5-large-en-ru¹⁰. При этом предварительно подготовленные контексты в виде файлов json-формата (рис. 1) загружались в векторную базу данных в два этапа. На первом этапе данные кодируются в эмбединги, используя энкодер модели d0rj/e5-large-en-ru и составляется SQL-инструкция с дополнительной метainформацией для каждого из них. На втором этапе подготовленные инструкции выполняются в виде транзакции вставки построчно в базу данных (рис. 2). Таким образом получаем заполненную базу данных (знаний), где каждая запись несет полную информацию для успешного извлечения релевантных текстов и формирования промпт-инструкции согласно запросу пользователя (табл. 2).

Таблица 2

Пример записи в базе данных для представления контекстов в виде эмбедингов

Table 2

Example of a database entry for representing contexts as embeddings

id	text	years	section	subsection	embeddings
2054	В январе 2012 г. наблюдалась неустойчивая, со значительным недобором осадков погода. Максимальная температура воздуха повышалась до –10...–15 °С. Минимальная температура воздуха понижалась до –36...–40 °С. Среднемесячная температура воздуха в январе составила –20...–23 °С, что на 3–6 °С ниже нормы. Осадки наблюдались преимущественно слабые и составили 22–65 % от нормы.	2012	климат	погода, температура, осадки по месяцам	[1,1712, –0,0270, –1,6838, ..., 0,6163, –0,7662, –0.0960]

¹⁰ <https://huggingface.co/d0rj/e5-large-en-ru>

```

...
model = SentenceTransformer("d0rj/e5-large-en-ru")
model.to("cuda")
...
input_files = [eco-report-2014-weather.pdf.json,...]

# Инициализация соединения с БД
connection = psycopg2.connect(database="...", user="...", password="...",
                             host="...", port="...")
# Создание курсора соединения
cursor = connection.cursor()

# Каждый файл контекстов открываем в json-формате
for input_file in input_files:
    file = open(input_file, encoding="utf-8")
    data = json.load(file)
    ....

    # Читаем и обрабатываем каждую json-запись
    for context in data:
        # Читаем поля текст, год, раздел, подраздел
        text = context["text"]
        year = context["year"]
        section = context["section"]
        subsection = context["subsection"]

        # Создаем эмбединги
        text_embeddings = model.encode(text, show_progress_bar=False,
                                       convert_to_numpy=True,
                                       normalize_embeddings=True)
        # Ковертируем массив Python в формат SQL-массива.
        embeddings_pg_vector = f"['{','.join(map(str, text_embeddings))}']"

        # Формируем полную SQL-инструкцию.
        sql = (f"INSERT INTO rag.contexts (text, years, months, section,
                                         subsection, keywords, embeddings) "
              f"VALUES ('{text}', '{years}', '{months}', '{section}', "
              f"'{subsection}', '{keywords}', {embeddings_pg_vector})")
        # Добавляем инструкцию в массив на исполнение
        sqls.append(sql)

# Вставляем данные в таблицу contexts, используя механизм транзакций
for idx, sql in sqls:
    cursor.execute(sql)
connection.commit()
....

```

Рис. 2. Фрагмент кода на языке Python для загрузки текстов в векторную базу

Fig. 2. A fragment of Python code for uploading texts to a vector database

RAG-система

Одной из ключевых проблем организации функционирования КВО является эффективность, точность и автоматизация построения промптов, релевантных запросу пользователя. Для решения данной проблемы был использован подход Retrieval-Augmented Generation (RAG). RAG – это технология, сочетающая в себе поиск релевантной информации в существующих хранилищах данных и генерацию текста с помощью языковых моделей, для создания более точных и информативных ответов [4]. Использование RAG в разработанной КВО-сис-

теме позволило обойти ряд ограничений, присущих таким специализированным системам, работающих на малых графических кластерах (не более 8 GPU):

- **Ограничение на количество токенов.** Современные большие языковые модели имеют ограничение на максимальное количество токенов, которое они могут обрабатывать за раз. Это делает невозможным хранение и обработку большого объема информации непосредственно внутри модели.
- **Актуальность данных.** Языковые модели обучаются на статических данных, которые не совсем актуальны для, например, региональной экологии. RAG позволяет получать актуальную информацию из внешних источников (векторных баз данных) в режиме реального времени.
- **Экономия ресурсов.** Использование RAG снижает нагрузку на языковую модель, так как большая часть знаний находится во внешних источниках и загружается только при необходимости. RAG состоит из двух основных компонентов: модуль извлечения (Retriever) и модуль генерации (Generator). Модуль извлечения ищет релевантную информацию в подключенной векторной базе данных (например, PostgreSQL/pgvector). Модуль генерации (Generator) создает текстовые ответы на основе данных, предоставленных модулем извлечения.

В разработанной системе для поиска и извлечения релевантной информации используются подходы, основанные на применении менее сложных и малых по параметрам нейронных сетей, а именно Bi-Encoder и Cross-Encoder. Bi-Encoder и Cross-Encoder являются подходами, используемыми в контексте плотного поиска (dense retrieval).

Bi-Encoder – это подход, который строит отдельные векторные представления для запроса и документа (контекста). Преимущества Bi-Encoder включают высокую скорость обработки, эффективность использования памяти, масштабируемость и простоту интеграции. Так как запрос и документ кодируются отдельно двумя экземплярами одной и той же модели, то векторы запросов и документов (текстов) проецируются в общее пространство, где сходство между ними вычисляется с помощью метрик, таких как косинусное расстояние или скалярное произведение. Такой подход позволяет формировать эмбединги только для запроса пользователя, а вычислять метрику на основе запроса средствами векторной СУБД. Основываясь на этих преимуществах, Bi-Encoder позволяет достаточно быстро сформировать пул из, например, 30–80 корпусов наиболее релевантных текстов для последующей передачи их в более медленный, но наиболее точный Cross-Encoder. На рис. 3 представлен код извлечения текстов из векторной базы данных в модуле Bi-Encoder.

Bi-Encoder имеет некоторые ограничения в плане точности и релевантности между запросом и извлекаемыми текстами – это отсутствие прямого взаимодействия на этапе кодирования, что может приводить к потере информации о контексте. Другими словами, релевантные тексты могут получить меньшее значение, например, евклидова расстояния. Следовательно, для гарантированного «охвата» нужных, необходимо оставлять большее их количество в результате запроса. Для нивелирования данного ограничения был использован подход на основе Cross-Encoder.

Cross-Encoder использует модель для совместного кодирования запроса и документа, что позволяет учитывать их взаимодействие на этапе кодирования. Запрос и документ конкатенируются и подаются в один общий трансформер. Модель генерирует представление для их комбинации и вычисляет оценку релевантности (например, вероятность, что документ соответствует запросу). Поэтому точность ранжирования и, следовательно, оценка релевантности у подхода с использованием Cross-Encoder заметно выше, чем у Bi-Encoder.

В текущей версии КВО для подхода Cross-Encoder была использована дообученная модель all-MiniLM-L6-v2¹¹. Дообучение проводилось на специальной выборке MMarco¹² для русского языка. Выборка состояла из обучающих и валидационных записей (рис. 4, 5).

¹¹ <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

¹² <https://huggingface.co/datasets/unicamp-dl/mmarco/tree/main>

```

...
model = SentenceTransformer('d0rj/e5-large-en-ru', device="cuda")
...
# Получаем массив эмбедингов для пользовательского
# запроса (переменная question)
embeddings = model.encode(question, convert_to_numpy=True,
                           normalize_embeddings=True)
# Ковертируем массив Python в формат SQL-массива.
embeddings_pg_vector = f"[{' '.join(map(str, embeddings))}]"
....
# Выполняем SQL-инструкцию по поиску релевантных
# тестов на основе евклидовых расстояний
# для более точного ответа в SQL-инструкцию
# добавляется метаданная (год, раздел, подраздел)
query_result_rows = cursor.execute("SELECT text,
    inner_product(embeddings, %s) as inner_product
    FROM rag.contexts
    WHERE year=%s AND subsection = %s
    ORDER BY inner_product DESC LIMIT %s",
    (embeddings_pg_vector, years, subsection,
    context_fetch_limit)
    )

# Создаем массив текстов для последующей передачи
# в Cross-Encoder
texts = []
for row in query_result_rows:
    texts.append(row[0])

```

Рис. 3. Фрагмент кода извлечения релевантных текстов на основе подхода Bi-encoder и векторной базы данных PostgreSQL/pgvector

Fig. 3. A fragment of code for retrieving relevant texts based on the Bi-encoder approach and PostgreSQL/pgvector vector database

```

{
  "text": "В 2021 году в Новокузнецком городском округе максимальная из разовых  
концентрация оксида углерода 2,7 ПДК и наибольшая повторяемость проб выше ПДК - 2,0 %  
отмечены в Центральном районе.",
  "question": "В Центральном районе была определена максимальная концентрация оксида  
углерода за один раз - какая она была?",
  "rank": 4.4
}

```

Рис. 4. Пример элемента обучающей выборки для модели all-MiniLM-L6-v2

Fig. 4. Example of a training sample for the all-MiniLM-L6-v2 model

```

{
  "query": "Что такое валовая заработная плата, включая надбавку?",
  "positive": [
    "Компоненты валовой заработной платы включают базовую заработную плату, надбавку за дороговизну, надбавку за аренду дома, городскую компенсационную надбавку и другие вознаграждения."
  ],
  "negative": [
    "Выигрыши в азартных играх. Денежные выигрыши включаются в федеральный валовой доход и, следовательно, должны быть включены в валовой доход штата Массачусетс. Помимо денежных выигрышей, выигрыши в азартных играх также включают справедливую рыночную стоимость безналичных призов, таких как автомобили, дома и поездки.",
    "Для новоприбывших около 15 процентов ежемесячной брутто-зарплаты работника вычитаются работодателем из части налога на заработную плату правительства Бермудских островов.",
    ....
  ]
}

```

Рис. 5. Пример элемента валидационной выборки для модели all-MiniLM-L6-v2

Fig. 5. Example of a validation sample for the all-MiniLM-L6-v2 model

Данная большая выборка является аккумулярованием различных датасетов, составленных различными группами исследователей. Ранги проставлялись вручную, равно как и формировались наборы «positive» и «negative» в валидационных датасетах. Общий размер выборки порядка 9 Гб текстовой информации. Общее время дообучения модели составило около 5 дней на шести видеокартах Nvidia A100 (80 Гб VRAM).

Отличие обучающих и валидационных выборок обусловлено требуемой спецификой работы подхода Cross-Encoder. А именно, было важным не только назначить ранги соответствия запрос – текст, но и получить правильный порядок вывода текстов из базы данных. Применение простой сортировки в данном случае было не совсем уместно. Модель должна во время обучения сопоставлять вычисленный ранг с местом (порядком) следования текстов. Данный результат был достигнут применением комбинации двух метрик: MRR (Mean Reciprocal Rank) и NDCG (Normalized Discounted Cumulative Gain)¹³.

Наряду с преимуществами подход Cross-Encoder также имеет свои ограничения. Это медленное выполнение ввиду невозможности предобработать тексты, т. е. каждая пара запрос – текст должна быть вычислена в реальном времени. Следовательно, для увеличения точности выбора релевантных текстов относительно пользовательского запроса необходимо использовать комбинацию Bi-Encoder и Cross-Encoder. То есть сформировать пул из, например, 30–50 корпусов текстов при помощи подхода Bi-Encoder и передать их на вход в более медленный, но наиболее точный Cross-Encoder, и уже на основе его классификатора проводить переранжирование по степени схожести, в результате оставляя 2–3 их них, из которых потом формировать окончательный промпт для большой языковой модели.

Для еще более точного извлечения релевантных текстов из базы данных применялись дополнительные фильтры перед отправкой пользовательского запроса в модуль Bi-Encoder. Фильтры включали проверку орфографии, классификацию вопроса по подразделам корпусов

¹³ <https://blog.stackademic.com/ndcg-vs-mrr-ranking-metrics-for-information-retrieval-in-rags-2061b04298a6>

текстов, а также фильтрацию запроса (по необходимости) по годам и месяцам для сужения области поиска и уточнения контекстной информации для LLM.

Для проверки орфографии использовались сервис проверки орфографии от Яндекс¹⁴, а также в случае его недоступности модель ai-forever/RuM2M100-1.2B¹⁵. Классификация вопроса по подразделам корпусов текстов проводилась с помощью специально дообученной модели google/mt5-large¹⁶. Дообучение проводилось на подготовленном датасете, включающем различные варианты запросов пользователя и их соответствия подразделам базы знаний (рис. 6). Всего было подготовлено более 17 000 вопросов по 20 разделам.

```
{
  "question": "Какой процентный рост массы выбросов загрязняющих веществ
    произошел в Калтанском городском округе в 2020 году по сравнению с
    предыдущим годом?",
  "class_id": 12,
  "class_name": "выбросы загрязняющих веществ по административным террито"
```

Рис. 6. Пример элемента обучающей выборки для классификации запроса пользователя в название подраздела

Fig. 6. Example of a training sample for classifying a user query into a subsection title

Для определения года, месяца, замены номеров месяцев, кварталов, различных выражений, обозначающих временные промежутки и т. п., использовались нативные регулярные выражения языка Python. Целью их использования была замена нерелевантных фрагментов временных выражений в запросе пользователя на использование только названий месяцев и цифровых обозначений годов. Например, для грамматически неверного вопроса «Какая *срдня температура* была во втором квартал 2014 года в Кемеровской области?», последовательное применение дополнительных фильтров вернет преобразованный вопрос «Какая средняя температура была в апреле, мае, июне 2014 года в Кемеровской области?», определит, что он относится к докладам 2014 года и подразделу «Погода, температура, осадки по месяцам», при этом работает шаблон регулярных выражений следующего вида «(\b[авдкнопс]*\b)\s*(\b[авгдziejклмнопрстцщшщюя]*\b)\s*(1|2|3|4)\s*квартал\w{0,3}».

Тестирование вышеуказанных подходов показало, что точность извлечения релевантных корпусов текстов повысилась с 50 до 93,7 %. А именно, частота попадания релевантного корпуса в набор повысилась до 99 % против 73 % (только Bi-Encoder с косинусным расстоянием), при этом релевантный контекст всегда попадал в 5 лучших текстов, отсортированных по убыванию ранга. Тестирование проводилось на 567 различных тематических пользовательских вопросах, подсчет и проверка осуществлялась вручную с поиском правильного ответа непосредственно в тексте файлов официальных докладов «О состоянии окружающей среды».

Ниже представлен полный вид промпт-инструкции для большой языковой модели КВО (рис. 7).

Представленная версия КВО поддерживает ответы на запросы пользователя в табличном виде. Таблицы в тексте докладов были переведены в таблицы базы данных PostgreSQL. Текущая версия системы может работать с таблицами, представленными в первой форме нормализации и не имеющими соединений по вторичному ключу. Для корректного составления промп-

¹⁴ <https://speller.yandex.net/>

¹⁵ <https://huggingface.co/ai-forever/RuM2M100-1.2B>

¹⁶ <https://huggingface.co/google/mt5-large>

та для ответа в табличном виде использовался дополнительный модуль RAG-системы. Данные модуля позволяют классифицировать вопрос пользователя в название таблицы. Для классификации применялась дообученная модель google/mt5-large. Дообучение проводилось на подготовленном датасете, включающем различные варианты запросов пользователя и их соответствия названиям таблиц в базе данных PostgreSQL (рис. 8). Всего было подготовлено более 6 245 вопросов для 6 таблиц.

```
{
  "messages": [
    {
      "role": "system",
      "content": "Ты - ИИ-помощник. Веди себя так, как будто у тебя нет других источников информации кроме Текста. Внимательно следуй тексту. Ты должен дать подробный ответ на Вопрос, используя данный Текст."
    }, {
      "role": "user",
      "content": "Текст:\nВ июле 2019 года умеренно прохладная, с недобором осадков, дымкой, утренними туманами погода. Максимальная температура воздуха была от +25 до +32 °С. Минимальная температура воздуха понижалась до +7, +10 °С. Среднемесячная температура воздуха составила +17, +19 °С, что на 1 °С ниже нормы. Ливневые дожди с грозами наблюдались по большинству районов. В сумме за месяц осадков выпало 40-83 % нормы."
    }, {
      "role": "bot",
      "content": "В июле 2019 года в области была умеренно прохладная погода с недобором осадков, дымкой, утренними туманами. Максимальная температура воздуха варьировалась от +25 до +32 °С, а минимальная температура понижалась до +7, +10 °С. Среднемесячная температура составила +17, +19 °С, что на 1 °С ниже нормы. Ливневые дожди с грозами были характерны по большинству районов, а суммарное количество осадков за месяц составило 40-83 % от нормы."
    }
  ]
}
```

Рис. 7. Промпт-инструкция для вопроса/ответа БЯМ в текстовом виде. Элемент массива «messages» с ключом «role», равное «bot», добавляется при ответе БЯМ на запрос пользователя

Fig. 7. Prompt instruction for question/answer LLM in textual format. The “messages” array element with the key “role” equal to “bot” is added when the LLM responds to a user query

```
{
  "question": "В каком году в городе Новокузнецке было наибольшее количество повторений северного направления ветра?",
  "class_id": 5,
  "class_name": "repeat_wind_direction_calms"
}
```

Рис. 8. Пример элемента обучающей выборки для классификации запроса пользователя в название таблицы

Fig. 8. Example of a training sample for classifying a user query into a table title

```

SELECT 'CREATE TABLE ' || pc.relname || E'(\n' ||
    string_agg(pa.attname || ' ' || pg_catalog.format_type(pa.atttypid,
        pa.atttypmod) || coalesce(' DEFAULT ' || (
        SELECT pg_catalog.pg_get_expr(d.adbin, d.adrelid)
        FROM pg_catalog.pg_attrdef d
        WHERE d.adrelid = pa.attrelid
        AND d.adnum = pa.attnum
        AND pa.atthasdef
    ), '')) || ' ' ||
    CASE pa.attnotnull
        WHEN TRUE THEN 'NOT NULL'
        ELSE 'NULL'
    END, E',\n') ||
    coalesce((SELECT E',\n' || string_agg('CONSTRAINT ' ||
        pc1.conname || ' ' ||
        pg_get_constraintdef(pc1.oid), E',
        \n' ORDER BY pc1.conindid)
        FROM pg_constraint pc1
        WHERE pc1.conrelid = pa.attrelid), '') ||
    E');'
FROM pg_catalog.pg_attribute pa
    JOIN pg_catalog.pg_class pc
        ON pc.oid = pa.attrelid
        AND pc.relname = 'repeat_wind_direction_calms'
    JOIN pg_catalog.pg_namespace pn
        ON pn.oid = pc.relnamespace
        AND pn.nspname = 'rag_tables'
WHERE pa.attnum > 0
    AND NOT pa.attisdropped
GROUP BY pn.nspname, pc.relname, pa.attrelid;

```

Рис. 9. SQL-инструкция для генерации DDL-инструкции создания выбранной таблицы (таблица «repeat_wind_direction_calms», схема «rag_tables»)

Fig. 9. SQL instruction for generating the DDL statement to create the selected table (table “repeat_wind_direction_calms”, schema “rag_tables”)

После классификации вопроса, на основе имени таблицы, используя специальный SQL-запрос (рис. 9), формировалась ее DDL-инструкция вместе с описанием столбцов и таблицы. Данная DDL-инструкция включалась в промпт-инструкцию для вопроса/ответа в табличном виде (рис. 10).

Серверная часть системы

В текущей версии в ядре разработанной системы поддерживаются следующие большие языковые модели: casperhansen/llama-3-70b-fp16¹⁷ для ответов в текстовом виде и defog/sqlcoder-70b¹⁸ для ответов в табличном виде. Для ускорения ответов БЯМ была использована технология Text Generation Interface (TGI)¹⁹. Запуск моделей производился параллельно на не-

¹⁷ <https://huggingface.co/casperhansen/llama-3-70b-fp16>

¹⁸ defog/sqlcoder-70b-alpha

¹⁹ <https://github.com/huggingface/text-generation-inference?tab=readme-ov-file#run-locally>

```

{
  "messages": [
    {
      "role": "user",
      "content": "## Task\n\tGenerate a SQL query to answer \n\t[QUESTION]Покажи  

      все информацию из таблицы \"Средняя температура  

      воздуха\"?[/QUESTION]\n\n\t### Instructions\n\t- If you cannot answer the  

      question with the available database schema, return 'I do not know'\n\n\t###  

      Database Schema\n\tThe query will run on a database with the following  

      schema:\n\tCREATE TABLE average_monthly_air_temperature(id integer NOT NULL,  

      month character varying(1024) NULL,\n\tvalue numeric(5,1) NULL, year  

      integer NULL, locality character varying(1024) NULL,\n\tCONSTRAINT  

      average_monthly_air_temperature_pkey PRIMARY KEY (id));\n\n\tCOMMENT ON  

      TABLE average_monthly_air_temperature IS 'Средняя температура воздуха,  

      С';\n\tCOMMENT ON COLUMN id IS NULL;\n\tCOMMENT ON COLUMN month IS  

      'Месяц';\n\tCOMMENT ON COLUMN value IS 'Средняя температура';\n\tCOMMENT ON  

      COLUMN year IS 'Год';\n\tCOMMENT ON COLUMN locality IS 'Административная  

      территория';\n\n\t### The average_monthly_air_temperature table Data  

      Example:\n\tThe COLUMN 'id' can contain any numeric data in range from 1 to  

      72\n\tThe COLUMN 'month' can contain any text data such as: Ноябрь, Апрель,  

      Декабрь, Июнь ... \n\tThe COLUMN 'value' can contain any numeric data in  

      range from -20.7 to 19.6\n\tThe COLUMN 'year' can contain any numeric data  

      in range from 2020 to 2022\n\tThe COLUMN 'locality' can contain any text  

      data such as: Новокузнецк, Кемерово"
    },
    {
      "role": "bot",
      "content": "### Answer\n\tGiven the database schema, here is the SQL query  

      that answers \n\t[QUESTION]Покажи все информацию из таблицы \"Средняя  

      температура воздуха\"?[/QUESTION]\n\t[SQL]\n\t'SELECT * FROM  

      average_monthly_air_temperature;'"
    }
  ]
}

```

Рис. 10. Промпт-инструкция для вопроса/ответа БЯМ в табличном виде. Элемент массива «messages» с ключом «role», равное «bot», добавляется при ответе БЯМ на запрос пользователя

Fig. 10. Prompt instruction for question/answer LLM in tabular format. The “messages” array element with the key “role” equal to “bot” is added when the LLM responds to a user query

скольких GPU с использованием Text Generation Interface (TGI). Это позволило повысить производительность и скорость генерации текста. Основные механизмы параллельного запуска БЯМ включали:

- модульную декомпозицию (Model Sharding). Модель разбивается на части (шарды), и каждая часть размещается на отдельном GPU. Это полезно для размещения больших моделей, которые не помещаются в память одного устройства;

- параллельное выполнение (Tensor Parallelism). Вычисления для одной модели распределяются между несколькими GPU. Используется для ускорения обработки запросов;
- обработка запросов (Batching). Пакетная обработка запросов распределяется между GPU. Это позволяет эффективно использовать ресурсы;
- оптимизация передачи данных. Синхронизация между GPU минимизируется за счет использования оптимизированных библиотек, таких как NCCL (NVIDIA Collective Communications Library).

Для запуска моделей llama3 или sqlcoder использовалась команда «text-generation-launcher --sharded false --num-shard=2 --max-input-length 4096 --max-total-tokens 8192 --model-id defog/sqlcoder-70b-alpha --port 8098», где --num-shard указывает количество GPU для шардинга модели, --model-id – уникальный идентификатор модели на ресурсе Huggingface²⁰ либо путь к локальной папке с файлами модели, --port – сетевой порт для обращения к TGI через протокол http.

Для генерации отправки промпт-инструкции и получения ответа БЯМ можно использовать любой http-клиент либо программную библиотеку. Например, команда «curl localhost:8080/v1/chat/completions -X POST -d ‘{“messages”: <message>, “max_tokens”: 2048}’ -H ‘Content-Type: application/json’», где <message> – текст в json-формате (рис. 8).

В разработанной КВО все обращения к TGI реализованы программно на языке Python. Также разработанная система имеет возможность взаимодействия с открытыми API веб-порталов ChatGPT²¹ и GigaChat²².

Для разработки серверной части КВО был использован следующий технологический стек: Python 3.10, FastAPI, Transformers, Pytorch, CUDA 12.4. БЯМ размещены на графическом сервере с 6xGPU Nvidia A100, 80 Гб VRAM.

Клиентская часть системы

Клиент КВО выполнен в виде веб-сайта²³ (рис. 11) в сети Интернет, со следующей функциональностью: аутентификация и авторизация пользователя посредством Telegram, создание/удаление чатов, справочная информация о системе, ввод пользовательского запроса вручную или голосом, повторение ответов системы, вывод ответов в табличном виде, переключение между выводом в текстовом или табличном виде. Также клиент разработанной системы доступен в виде Telegram-бота²⁴.

Для разработки был использован следующий технологический стек: ReactJS, Redux Toolkit, Styled-components.

Таким образом, подробно рассмотренное в предыдущих разделах описание КВО можно представить в следующих диаграммах взаимодействия основных компонентов (рис. 12, 13).

²⁰ <https://huggingface.co/>

²¹ <https://chatgpt.com/>

²² <https://giga.chat/>

²³ <http://eco-expert.ict.nsc.ru/>

²⁴ https://t.me/eco_expert_bot

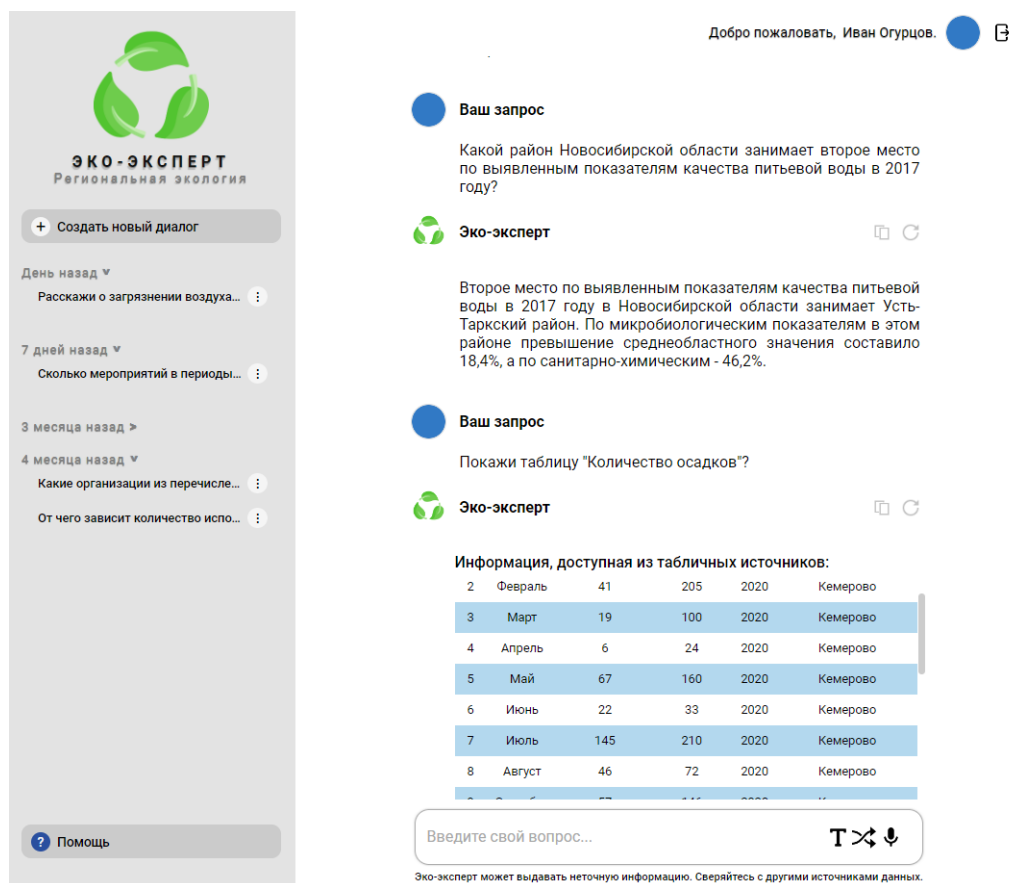


Рис. 11. Фрагмент графического интерфейса разработанной КВО
Fig. 11. Fragment of the graphical interface of the developed KVO

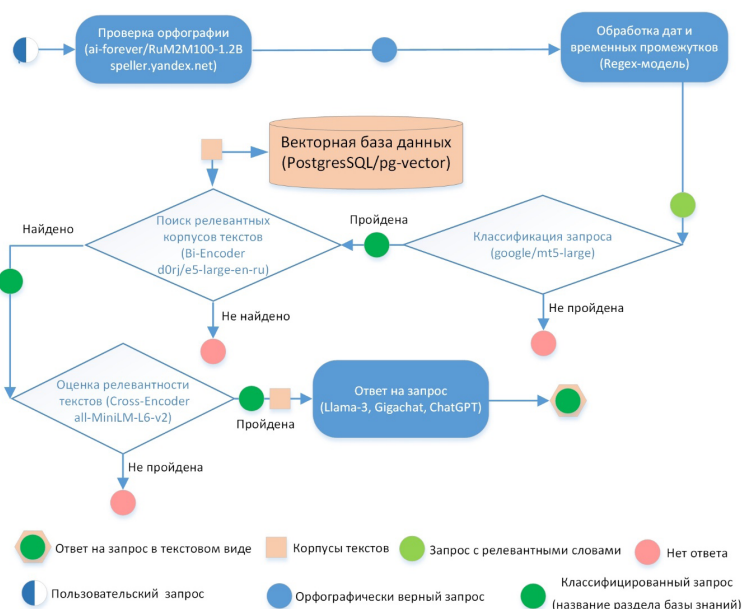


Рис. 12. Схема информационных потоков в RAG-системе для ответа LLM в текстовом виде
передать названия БЯМ-моделей

Fig. 12. Diagram of information flows in a RAG system for LLM response in textual format with modified names of LLM models

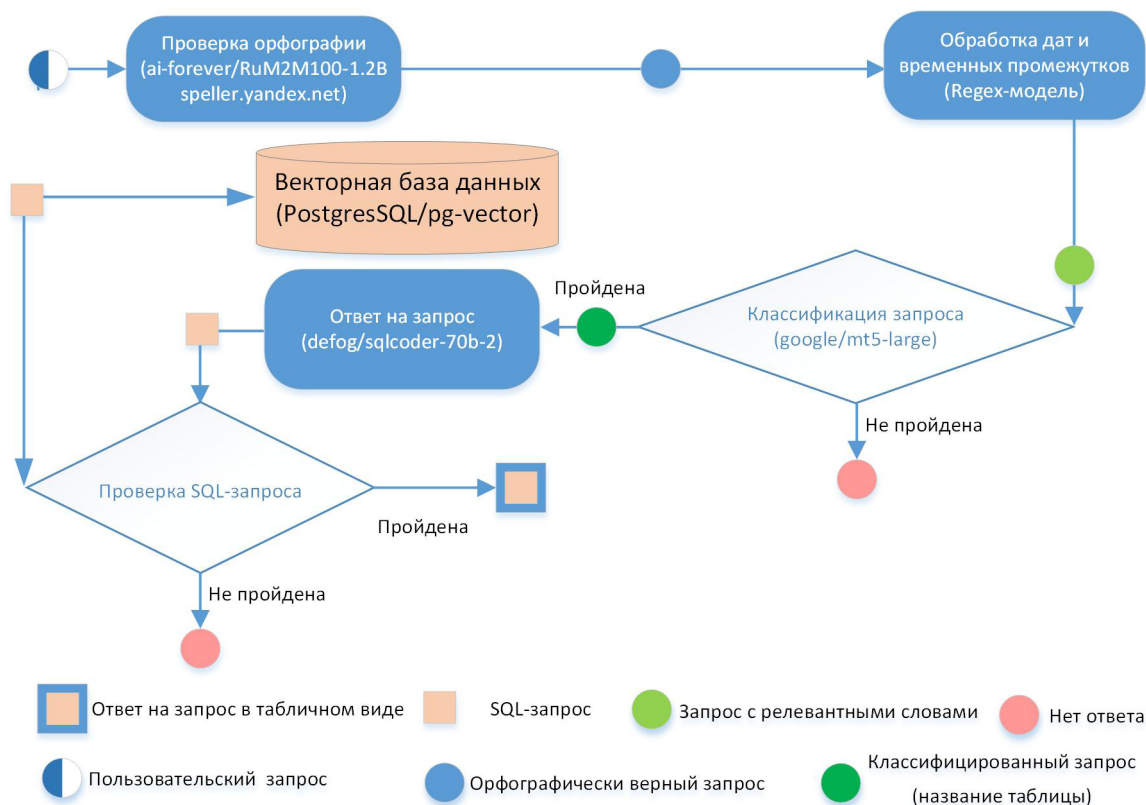


Рис. 13. Схема информационных потоков в RAG-системе для ответа LLM в табличном виде

Fig. 13. Diagram of information flows in a RAG system for LLM response in tabular format

Метрики и тестирование системы

Для тестирования разработанной КВО-системы использовались следующие метрики:

Bi-Encoder

Precision@k – доля релевантных документов среди первых k-предсказанных.

$$\text{Precision@k} = \frac{\text{Количество релевантных документов в top} - k}{k}$$

Recall@k – доля релевантных документов, найденных среди первых k-предсказаний.

$$\text{Precision@k} = \frac{\text{Количество релевантных документов в top} - k}{\text{Общее количество релевантных}}$$

Cross-Encoder

MRR измеряет, насколько высоко в списке ранжированных результатов находится первый релевантный элемент. Метрика используется для задач ранжирования, где важен порядок результатов, например, в информационном поиске (1):

$$MRR = \frac{1}{N} \sum_{i=1}^N \frac{1}{rank_i}, \quad (1)$$

где N – общее количество запросов, $rank_i$ – позиция первого релевантного документа для i -го запроса.

Метрика NDCG измеряет качество ранжирования, учитывая как порядок, так и степень релевантности документов. Нормализованная версия метрики позволяет сравнивать результаты между запросами с разным числом релевантных документов (2):

$$NDCG = \frac{DCG}{IDCG}, IDCG = \sum_{i=1}^P \frac{rel_i^{sorted}}{\log_2(i+1)}, DCG = \sum_{i=1}^P \frac{rel_i}{\log_2(i+1)}, \quad (2)$$

где P – число результатов, rel_i – релевантность документа на позиции, rel^{sorted} – релевантности документов, отсортированные по убыванию.

MAP (Mean Average Precision) – средняя точность всех запросов, учитывая все релевантные документы:

$$MAP = \frac{1}{N} \sum_{j=1}^N AP_i,$$

где AP_i – качество ранжирования релевантных документов для конкретного запроса.

$$AP_i = \frac{\sum_{k=1}^n Precision@k \cdot Relevant_k}{\text{Количество релевантных документов для запроса}_i},$$

где n – общее количество документов в ранжированном списке, $Precision@k$ – точность на k -й позиции (доля релевантных документов среди первых k документов). $Relevant@k$ – индикаторная функция, равная 1, если k -й документ релевантен, и 0 в противном случае, количество релевантных документов для запроса i – общее число релевантных документов для данного запроса.

Лlama3 (оценка ответов)

BERTScore – метрика, основанная на представлениях из модели BERT, которая оценивает семантическое сходство между сгенерированным текстом и эталонным ответом.

ROUGE-L – метрика, которая оценивает перекрытие длинных последовательностей (longest common subsequence) между сгенерированным текстом и эталонным ответом.

METEOR – метрика, которая учитывает стемминг, синонимию и словоформы для более точного сравнения текстов.

Таблица 3

Сводная таблица метрик для КВО-системы

Table 3

Summary table of metrics for the CQA-system

Метрика	Вi-энкодер	Cross-энкодер	Llama3
Precision@5	0,85	0,92	–
Recall@10	0,78	0,93	–
MRR	0,89	0,95	–
nDCG@10	0,87	0,93	–
MAP	–	0,92	–
F1-score	–	0,91	–
BERTScore	–	–	0,85
ROUGE-L	–	–	0,72
METEOR	–	–	0,43

Примечание. Для модели Llama3 сравнивалось 345 ответов, составленных человеком и БЯМ.

Выводы

В рамках настоящего исследования разработана контекстно-вопросно-ответная система, основанная на современном подходе, использующем большие языковые модели и Retrieval-Augmented Generation (RAG). Реализованная система продемонстрировала значительное улучшение точности и эффективности ответов на вопросы, благодаря интеграции передовых алгоритмов обработки текста и механизмов поиска релевантной информации.

Использование Bi- и Cross-энкодеров обеспечило качественную обработку контекстуальной информации и более точную интерпретацию сложных запросов, что позволило системе адаптироваться к широкому спектру сценариев взаимодействия. Эти технологии оптимизировали процесс выбора наиболее релевантного ответа, повысив общую производительность системы.

Применение дополнительных фильтров предварительной классификации вопросов по разделам базы знаний значительно улучшило точность поиска. Этот подход минимизировал вероятность ошибок, возникающих при подборе ответа, и сократил вычислительные затраты.

Конвертация временных промежутков в явные названия месяцев и годов позволила устранить неоднозначность при обработке временных запросов. Данный подход улучшил пользовательский опыт и повысил точность ответов на вопросы, связанные с хронологией.

Использование векторной базы данных обеспечило хранение больших объемов информации и быстрый доступ к ним. Векторизация данных позволила ускорить поиск релевантных документов, что критически важно для систем, работающих с масштабируемыми наборами данных.

Применение RAG-подхода, дополненного методами классификации, векторизацией данных и визуализацией, доказало свою эффективность для решения широкого круга задач.

В перспективе система может быть расширена за счет интеграции новых алгоритмов машинного обучения, оптимизации графического интерфейса и увеличения объемов обрабатываемых данных. Эти направления могут открыть возможности для использования системы в новых прикладных областях, таких как медицина, юриспруденция, образование и пр.

Список литературы / References

1. **Jansen B. J., Rieh S.** The Seventeen Theoretical Constructs of Information Searching and Information Retrieval. *Journal of the American Society for Information Sciences and Technology*, 2010, vol. 61(8), pp. 1517–1534. DOI: 10.1002/asi
2. **Wang H., Qin Z., Wan T.** Text generation based on generative adversarial nets with latent variables. *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018*, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II 22. – Springer International Publishing, 2018, pp. 92–103. <https://doi.org/10.48550/arXiv.1712.00170>
3. **Roumeliotis K. I., Tselikas N. D.** ChatGPT and Open-AI Models: A Preliminary Review. *Future Internet*, 2023, vol. 15(6), p. 192. <https://doi.org/10.3390/fi15060192>
4. **Vidivelli S., Ramachandran M., Dharunbalaji A.** Efficiency-Driven Custom Chatbot Development: Unleashing LangChain, RAG, and Performance-Optimized LLM Fusion. *Computers, Materials and Continua*, 2024, vol. 80, iss. 2, pp. 2423–2442.

Информация об авторах

Попов Семен Евгеньевич, кандидат технических наук, старший научный сотрудник

Потапов Вадим Петрович, доктор технических наук, профессор, главный научный сотрудник

Замараев Роман Юрьевич, кандидат технических наук, старший научный сотрудник

Information about the Authors

Semyon E. Popov, Candidate of Technical Sciences, Senior Researcher

Vadim P. Potapov, Doctor of Technical Sciences, Professor, Chief Researcher

Roman Yu. Zamaraev, Candidate of Technical Sciences, Senior Researcher

*Статья поступила в редакцию 10.01.2025;
одобрена после рецензирования 27.02.2025; принята к публикации 27.02.2025*

*The article was submitted 10.01.2025;
approved after reviewing 27.02.2025; accepted for publication 27.02.2025*