**Исследование методов повышения эффективности системы RAG и генерации ответов с использованием LLM**

**1. Введение**

В рамках проекта проводилось прикладное исследование, направленное на оптимизацию компонентов Retrieval-Augmented Generation (RAG) и генерации ответов на основе больших языковых моделей (LLM). Основное внимание уделялось адаптации системы к особенностям русскоязычных текстов и оценке различных архитектур, способных повысить точность и релевантность выдаваемых ответов.

**2. Алгоритмы сегментации текста**

Одним из ключевых этапов в работе RAG-системы является разбиение исходных документов на логически обоснованные чанки. В рамках экспериментов были протестированы различные стратегии разбиения в зависимости от типа исходного текста. Были эмпирически подобраны следующие параметры сегментации:

Технические документы: размер чанка – 400 токенов, перекрытие – 80 токенов.

Юридические тексты: 350 / 70.

Новостные статьи: 450 / 50.

Научные статьи: 300 / 100.

Общие тексты: 500 / 50.

Эти параметры позволили достичь баланса между полнотой контекста и эффективностью поиска, демонстрируя важность выбора стратегии разбиения в зависимости от специфики домена.

**3. Векторизация и подбор моделей эмбедингов**

Для этапа семантического поиска были протестированы различные модели эмбедингов, ориентированные на русскоязычные и многоязычные тексты:

intfloat/multilingual-e5-small — универсальная и умеренно быстрая модель, показавшая средние результаты по качеству эмбедингов.

cointegrated/rubert-tiny2 — компактная модель с высокой скоростью и приемлемым качеством.

ai-forever/sbert\_large\_mt\_nlu\_ru — ресурсозатратная, но качественная модель.

ai-forever/ru-en-RoSBERTa — сбалансированная по скорости и качеству модель.

Оптимизация генерации эмбедингов достигалась за счёт батчевой обработки и распараллеливания на CPU. Потенциальный прирост скорости при использовании GPU не был реализован из-за ограничений вычислительных ресурсов.

**4. Параметр n\_results и влияние на метрики поиска**

Проводился анализ зависимости метрик качества (precision@k и recall@k) от параметра n\_results, определяющего количество документов, извлекаемых из векторной базы данных для последующей генерации ответа.

При низких значениях n\_results наблюдалась высокая точность, но низкая полнота, из-за чего релевантные документы могли быть упущены.

При высоких значениях полнота возрастала, но точность снижалась за счёт появления нерелевантных результатов.

Таким образом, оптимальное значение n\_results должно подбираться с учётом ограничений модели LLM по количеству токенов, а также предпочтений пользователя в сторону либо точности, либо полноты.

**5. Тестирование моделей LLM**

Для генерации финального ответа после поиска были протестированы следующие языковые модели:

meta-llama/Llama-3.1-8B-Instruct — средняя скорость работы, но слабая адаптация под русский язык.

meta-llama/Llama-3.2-3B-Instruct — высокая скорость, но часто генерировала нерелевантный текст.

IlyaGusev/saiga\_mistral\_7b\_lora — неплохая генерация, средняя скорость.

yandex/YandexGPT-5-Lite-8B-instruct — хорошее соотношение скорости и качества при генерации на русском языке.

**6. Оптимизация хранения и поиска в векторной базе данных**

Для хранения эмбедингов и поиска использовалась векторная база данных ChromaDB. В целях ускорения работы было реализовано создание индексов в коллекциях, но существуют еще следующие меры:

Использование кэширования результатов запросов.

Применение структур данных, оптимизированных для поиска (например, Struct of Arrays).

Адаптация выбора субд под конкретные задачи.

**7. Архитектура сервера LLM и генерация**

Генерация ответов происходила через API-интерфейс, предоставляемый локальной LLM, запущенной в LM Studio. В частности, использовалась квантованная версия yandexgpt-5-lite-8b-instruct. Также приводится пример кода для запуска модели через transformers:

from transformers import AutoModelForCausalLM, AutoTokenizer

MODEL\_NAME = "yandex/YandexGPT-5-Lite-8B-instruct"

tokenizer = AutoTokenizer.from\_pretrained(MODEL\_NAME)

model = AutoModelForCausalLM.from\_pretrained(

MODEL\_NAME,

device\_map="cuda",

torch\_dtype="auto",

)

**8. Логирование и мониторинг**

Для мониторинга системных метрик и отслеживания результатов генерации использовалась платформа Weights & Biases (wandb). Пример отслеживаемого эксперимента:

<https://wandb.ai/yugik-tomsk-state-university/rag-eval/runs/k1jtj4fz?nw=nwuseryugik>

**9. Метрики качества**

Планировалась реализация автоматического логирования метрик качества генерации, включая:

ROUGE-L

BLEU

Однако на момент подготовки отчёта интеграция с этими метриками ещё не завершена, но создана библиотека ждя работы с этими метриками.

**Выводы:**  
  
Эффективность RAG-системы существенно зависит от грамотной настройки на всех этапах — от разбиения текста и выбора моделей эмбедингов до конфигурации параметров поиска и подбора LLM. При правильной комбинации этих элементов возможно существенно повысить как точность, так и полноту выдачи, особенно в условиях работы с русскоязычными данными.