УДК 004.041+004.056

Реализация RAG-архитектуры для автоматизированной проверки документов в корпоративных системах хранения с использованием больших языковых моделей

Костин Максим Андреевич — студент бакалавриата 4-го курса направления 09.03.01 «Информатика и вычислительная техника». Научные интересы: информационные системы, искусственный интеллект, веб-разработка. E-mail: m.kkostin@yandex.ru

Давыдова Даяна Михайловна — старший преподаватель кафедры «Информационные и вычислительные системы». Научные интересы: информационные системы, обработка больших данных. E-mail: dayana-0820@bk.ru

Петров Владимир Евгеньевич — кандидат воен. наук, доцент кафедры «Информационные и вычислительные системы». Научные интересы: информационные системы, обработка больших данных, моделирование надежности. E-mail: petroffve@mail.ru

Петербургский государственный университет путей сообщения Императора Александра I, Россия, 190031, Санкт-Петербург, Московский пр., 9

Для цитирования: Костин М. А., Давыдова Д. М., Петров В. Е. Реализация RAG-архитектуры для автоматизированной проверки документов в корпоративных системах хранения с использованием больших языковых моделей // Интеллектуальные технологии на транспорте. 2026. № 2 (46). С. 5–16. DOI: 10.20295/2413-2527-2026-246-5-16

Аннотация. *Предлагается специализированное решение для анализа документов, использующихся в корпоративных системах документооборота.* ***Цель:*** *разработка и практическая реализация RAG-архитектуры для автоматизированной проверки корпоративных документов с использованием локально развернутых больших языковых моделей, обеспечивающей выявление пропущенных обязательных полей, ошибок форматов данных и содержательных противоречий в документах транспортных и логистических систем.* ***Методы:*** *проведены проектирование и программная реализация Java-приложения, интегрирующего модуль извлечения текста из PDF- и DOCX-документов на основе библиотек Apache, векторное хранилище с упрощенными эмбеддингами на основе частотного анализа слов, алгоритм семантического поиска через вычисление косинусного сходства и клиент взаимодействия с LLM через API сервера Ollama.* ***Результаты:*** *разработанная система продемонстрировала способность к контекстному анализу содержания документов и адаптивность к вариативным форматам представления информации, что позволяет преодолеть ограничения традиционных систем. Экспериментальная проверка выполнена на тестовом наборе корпоративных документов с преднамеренно внесенными ошибками различных типов; оценка эффективности проводилась по метрикам полноты, точности и их среднего, а также по времени отклика системы. Наилучшие результаты показала модель llama3.2:latest при полном исключении передачи конфиденциальных данных за пределы инфраструктуры организации.* ***Практическая значимость:*** *предложенное решение применимо для автоматизации контроля качества документации в корпоративных системах документооборота транспортных предприятий, государственных учреждений и промышленных организаций. Модульная архитектура обеспечивает масштабируемость на другие типы документов и возможность интеграции с существующими информационными системами при минимальных затратах на адаптацию. Использование открытых моделей и локального сервера Ollama снижает зависимость от сторонних облачных сервисов и обеспечивает соответствие требованиям информационной безопасности.*

Ключевые слова: *большие языковые модели, автоматизированная проверка документов, векторный поиск, обработка естественного языка, корпоративные системы, Ollama, извлечение текста, семантический анализ*

2.3.6 — *методы и системы защиты информации, информационная безопасность (технические науки);*
1.2.1 — *искусственный интеллект и машинное обучение (технические науки)*

Введение

Современные корпоративные системы документооборота сталкиваются с беспрецедентным ростом объемов обрабатываемой информации, и традиционные методы проверки документов, основанные на правилах и регулярных выражениях, демонстрируют свою ограниченность при работе с неструктурированными данными и вариативными форматами представления информации. Особенность описанных систем, заключающаяся в неспособности к контекстному анализу содержания, приводит к пропуску содержательных ошибок, некорректному распознаванию сущностей и необходимости постоянной ручной доработки.

Для решения представленных проблем необходимо использовать большие языковые модели (Large Language Models, LLM), которые открыли новые возможности для интеллектуальной обработки документов. Однако не стоит забывать, что прямое применение LLM для анализа корпоративной документации сталкивается с рядом существенных ограничений, в качестве примера которых можно выделить проблемы актуальности информации, тенденцию к «галлюцинациям», отсутствие доступа к специализированным знаниям предметной области и высокие вычислительные затраты. Эти вызовы обусловливают необходимость разработки гибридных подходов, сочетающих достоинства генеративных моделей с точностью традиционных методов информационного поиска.

Для решения подобных проблем необходимо использовать технологию, которая позволит связать механизмы семантического поиска с возможностями генерации текстовых ответов. В качестве такой технологии активно применяются алгоритмы генерации с дополненной выборкой (Retrieval-Augmented Generation, RAG) [1], позволяющие преодолеть ограничения стандартных языковых моделей. Преимущества RAG заключены в дополнении запроса релевантным контекстом, позволяющим повысить точность обработки корпоративных документов.

Данное исследование посвящено реализации архитектуры приложения на основе RAG-алгоритмов. Таким образом, архитектура системы будет изначально спроектирована для автоматизированной проверки транспортных накладных, заявлений работников и многих других документов, которые ежедневно участвуют в документообороте различных предприятий и организаций. Разработанное решение включает в себя несколько модулей по обработке документов разных форматов (от PDF до DOCX), а также реализации алгоритмов векторного поиска и анализа поступаемой в систему информации. Локально спроектированные и связанные модули программы с учетом платформы для использования языковых моделей Ollama позволят обеспечить конфиденциальность информации и минимизируют риски ее утечки во внешние системы.

Целью работы является апробация RAG-архитектуры для автоматизированной проверки документов, обеспечивающей выявление пропущенных значений, идентификацию ошибок формата и содержательных противоречий.

Практическая значимость исследования заключается в создании адаптируемого решения для автоматизации контроля качества корпоративных документов, способного снизить трудозатраты на ручную проверку и минимизировать ошибки в документации.

Проектирование

Для создания системы обработки документов необходимо использовать современные подходы

и методы разработки программного обеспечения. В основе программы, реализующей RAG-архитектуру для работы большой языковой модели, требуется использовать объектно-ориентированный способ организации кода на одном из высокоуровневых языков программирования. Так как в основе большинства приложений банковского сектора, государственных систем документооборота, а также промышленных решений применяется язык программирования Java, то и реализованное приложение было создано на основе его пакетов и системных библиотек. Алгоритм обработки документов начинается с парсинга текста и формирования фрагментов фиксированного размера — чанков. Для считывания текста из документов необходимо создать утилитный класс Document Processor, в котором будут использоваться библиотеки Apache POI. Последовательное исполнение алгоритма обработки документа через LLM с векторным поиском приведено на рис. 1.

Для функционала работы клиента Ollama и подключения к LLM необходимо реализовать класс, создающий запрос для большой языковой модели

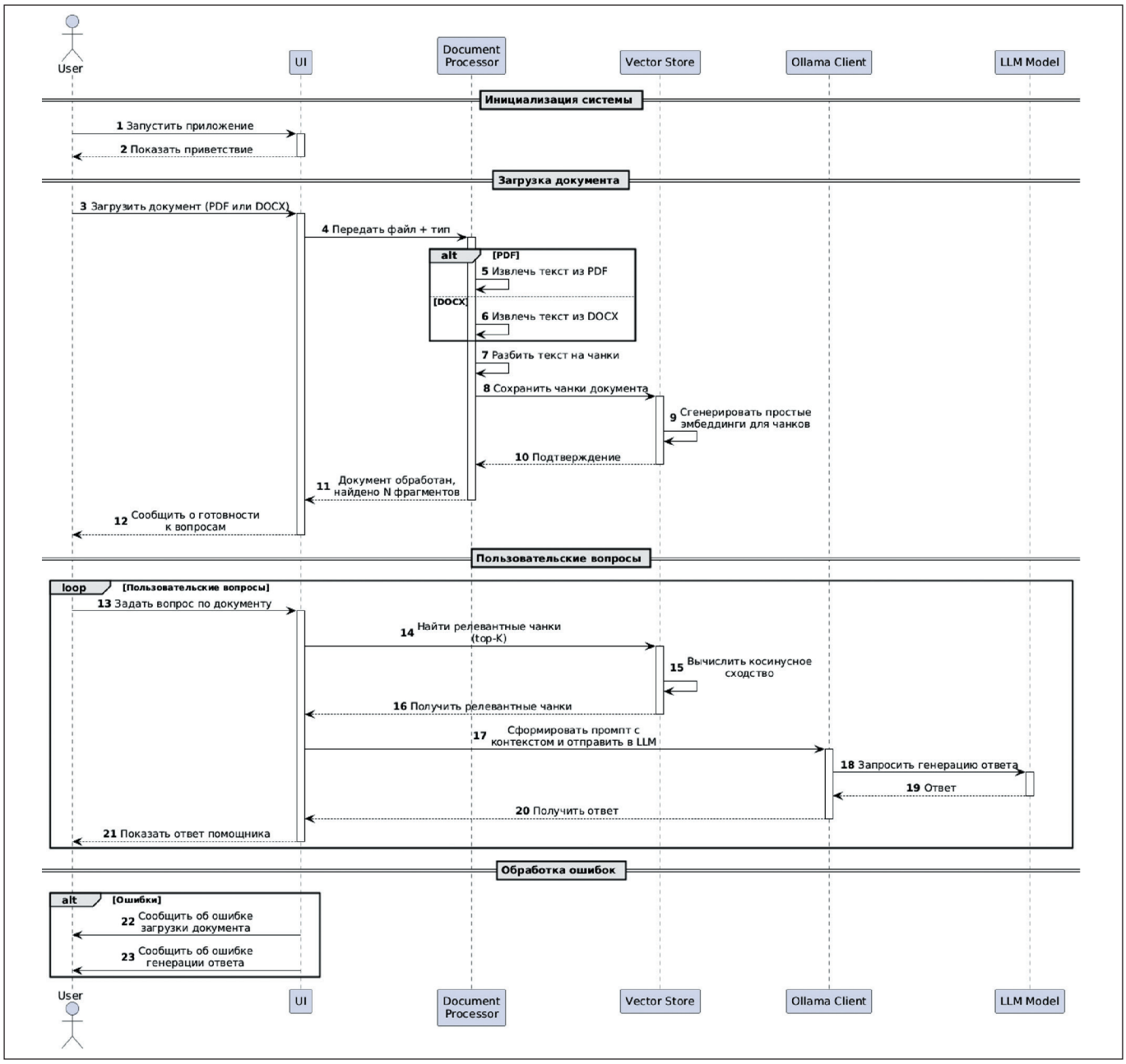

Рис. 1. Диаграмма последовательности системы анализа документации на основе RAG-архитектуры

и десериализирующий ее ответ из JSON-формата в строковый тип [2]. В основе векторного хранилища предполагается использовать класс, генерирующий числовое представление фрагментов текста на основе частоты слов, один из методов которого представлен в листинге 1.

Также необходимо отметить и использование метода для вычисления сходства между векторами через измерение угла между двумя ненулевыми векторами, более известного как алгоритм косинусного сходства:

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \times B}{\|A\| \times \|B\|},$$

где $\|A\|$ и $\|B\|$ — нормы векторов;

$A \times B$ — их скалярное произведение.

Реализация метода для нахождения метрики косинусного сходства должна включать вычисление скалярного произведения векторов и их евклидовых норм с последующим нормированием результата. Значение метрики лежит в диапазоне [0, 1], где 1 соответствует полной идентичности направлений векторов. Подробно метод представлен в листинге 2.

Реализация

В ходе исследования было реализовано приложение на языке программирования Java с использованием сборщика Maven и библиотек Apache для работы с документами различных форматов. Для полноценной работы с сервером Ollama через команду ollama serve был запущен сервис [3], к которому созданное Java-приложение обращается через HTTP-запросы [4]. Сервер Ollama работает как фоновая служба и предоставляет API по сетевому адресу, с его помощью вопросы пользователя переадресуются выбранной LLM. На рис. 2 показана диаграмма запуска Ollama-сервера.

```
// Метод для векторного представления текста private List<Double>
generateSimpleEmbedding(String text) {
    // Генерация эмбеддингов
    String[] words = text.toLowerCase().split(«\\s+»);
    Map<String, Integer> wordFreq = new HashMap<>();
    // Вычисление частоты слов в строковом массиве
    for (String word : words) {
        wordFreq.put(word, wordFreq.getOrDefault(word, 0) + 1);
    }
    List<Double> embedding = new ArrayList<>();
    // Фиксированный размер эмбеддинга - используется ограничитель (100)
    for (int i = 0; i < 100; i++) {
        embedding.add(0.0);
    }
    int index = 0;
    for (String word : wordFreq.keySet()) {
        if (index >= 100) break;
        embedding.set(index, (double) wordFreq.get(word));
        index++;
    }
    //[2.0, 2.0, 1.0, 1.0, 0.0, 0.0, 0.0, ..., 0.0]
    return embedding;
}
```

Листинг 1. Метод создания векторного представления текста

```
private double cosineSimilarity(List<Double> vec1, List<Double> vec2) {
    double dotProduct = 0.0;
    double norm1 = 0.0;
    double norm2 = 0.0;
    for (int i = 0; i < vec1.size(); i++) {
        dotProduct += vec1.get(i) * vec2.get(i);
        norm1 += Math.pow(vec1.get(i), 2); //норма вектора 1 и 2
        norm2 += Math.pow(vec2.get(i), 2);
    }
    return dotProduct / (Math.sqrt(norm1) * Math.sqrt(norm2));
}
```

Листинг 2. Реализация алгоритма косинусного сходства двух векторов

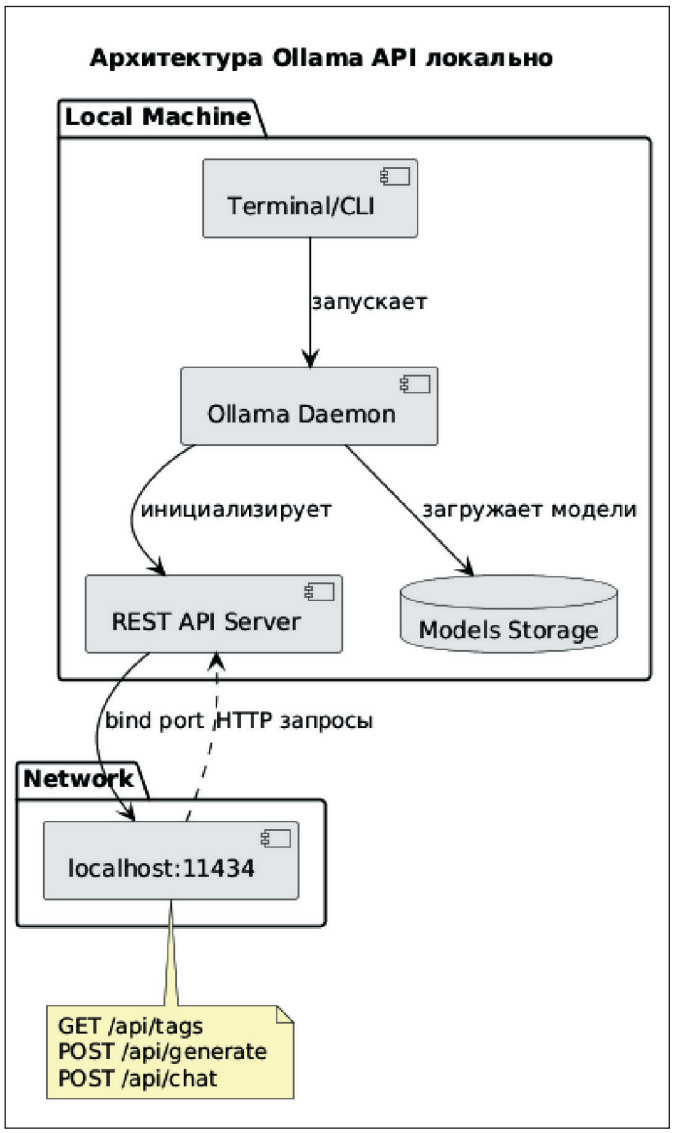

Рис. 2. Запуск сервиса Ollama для инициализации фоновой службы

В разработанном приложении предоставляется выбор LLM по желанию пользователя. Для демонстрации RAG-алгоритма и работы векторного поиска использовались несколько моделей: phi3:mini, qwen3, llama3.2. К преимуществам моделей, развернутых локально, можно отнести повышенную безопасность и контроль за передаваемой информацией. Так как доступ к модели и данным, которые она обрабатывает, есть только у владельцев инфраструктуры и ПО, использующих ее, утечка данных минимизируется [3]. Однако в ходе исследования был проведен анализ выбранных моделей, описанный в следующем разделе. Клиентская часть системы взаимодействует с сервером Ollama через HTTP/HTTPS-протокол, используя Java HttpClient [5], введенный в Java 11. Для каждого запроса формируется JSON-объект с параметром temperature, установленным на уровне 0,1, что обеспечивает детерминированность ответов модели. Это критически важно для задач проверки документов, где требуются высокая точность и воспроизводимость результатов. Параметр top_*p* = 0,9 в JSON ограничивает выбор токенов наиболее вероятными кандидатами, снижая вероятность генерации нерелевантного контента. Полная реализация метода для обращения к LLM отражена в листинге 3.

В ходе реализации был создан и GUI пользователя с возможностью отправления в систему нескольких видов документов [6]. Производительность обработки текста и скорость ответа зависят

```
public String generateResponse(String prompt) throws Exception {
    String jsonPayload = String.format("""
            {
              "model": "%s",
              "prompt": "%s",
              "stream": false,
              "options": {
                "temperature": 0.1,
                "top_p": 0.9
              }
            }
            """, modelName, escapeJson(prompt));
    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(GENERATE_URL))
            .header("Content-Type", "application/json")
            .POST(HttpRequest.BodyPublishers.ofString(jsonPayload))
            .build();
    HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
    if (response.statusCode() != 200) {
        throw new RuntimeException("HTTP Error: " + response.statusCode() + " - " + response.body());
    }
    JsonObject jsonResponse = JsonParser.parseString(response.body()).getAsJsonObject();
    return jsonResponse.get("response").getAsString();
}
```

Листинг 3. Клиентский метод для обращения к языковой модели и возвращения ее ответа

от типа используемой модели: чем больше языковая модель, тем больший объем знаний и алгоритмов в ней заложен, что существенно влияет на результат. В разработанном решении использовались небольшие языковые модели, при всех своих недостатках гарантирующие разбор отправленного на анализ текста. В качестве примера при тестировании системы использовался классический документ любой организации — заявление о предоставлении отпуска, представленный на рис. 3, в котором заранее была предусмотрена ошибка.

Языковая модель успешно справилась с ошибкой и вернула системе результат анализа (рис. 4), в котором подробно описываются недочеты документа. Помимо единичного ответа, пользователю предоставляется и возможность продолжить беседу с LLM, так как в системе предусмотрено хранение истории текущего чата, основанное на компонентах библиотеки Java Swing [6].

Программная реализация векторного представления текста, основанная на разбиении на слова, создании словаря и формировании вектора фиксированной длины, показана в алгоритмической схеме на рис. 5.

Алгоритм векторизации текста включает четыре основных этапа:

1. Токенизация текста. На данной стадии алгоритма исходная текстовая строка приводится к нижнему регистру для обеспечения регистронезависимости, после чего разбивается на отдельные слова по пробельным символам. Результатом этапа является упорядоченный массив токенов (слов).

2. Построение частотного словаря. Следующим этапом осуществляется последовательный

Руководителю ООО «ОМЕГА»

Печкину Ивану Алексеевичу

От менеджера по продажам

Малининой О.И.

Заявление о предоставлении отпуска

Прошу предоставить ежегодный основной оплачиваемый отпуск с июля 2024 года продолжительностью 14 календарных дней.

Малинина О.И. 10.03.2024

Рис. 3. Шаблон документа с ошибкой, который был отправлен в систему

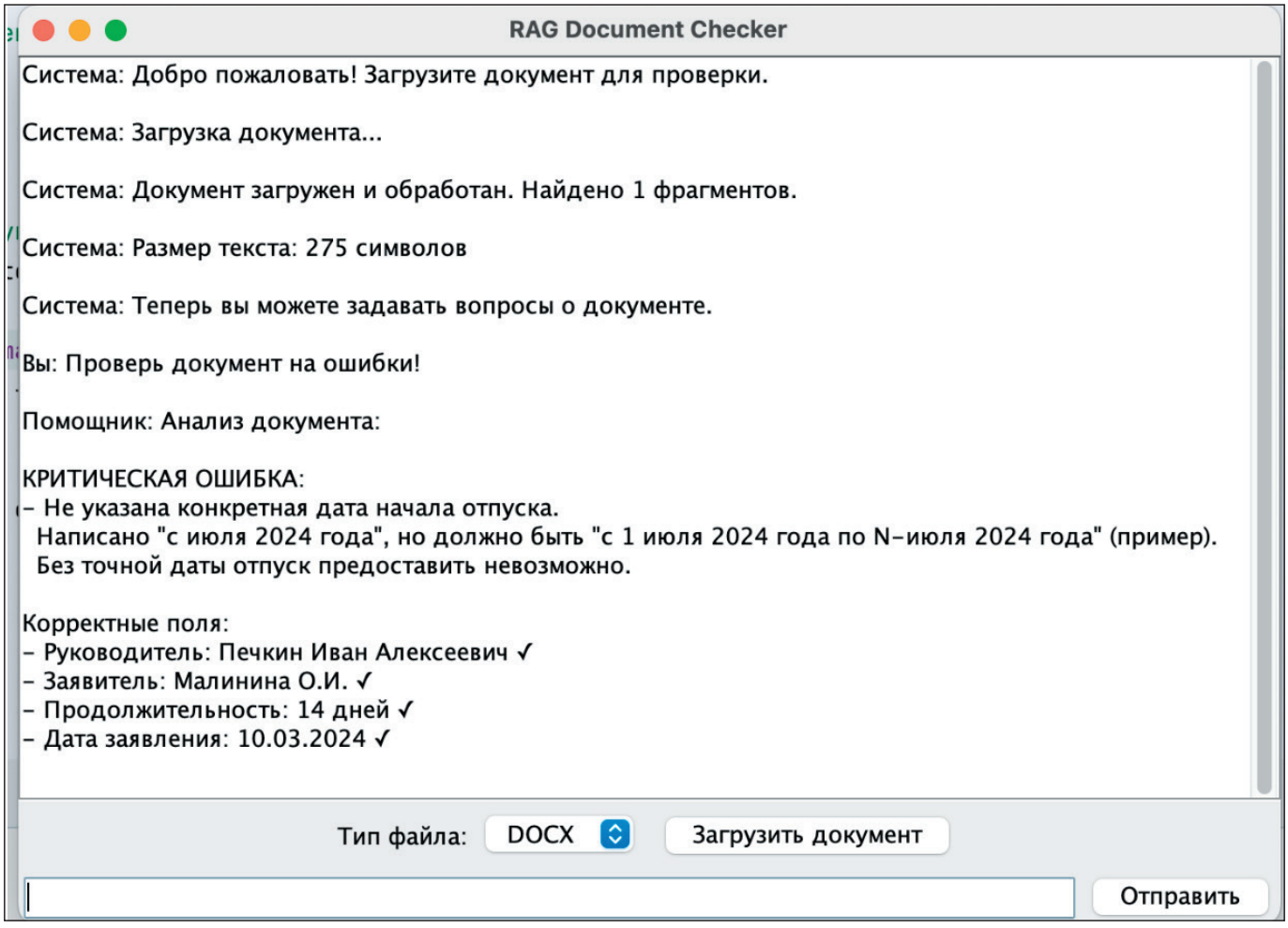

Рис. 4. Результат работы системы на LLM llama:3.2:latest после обработки документа с ошибкой

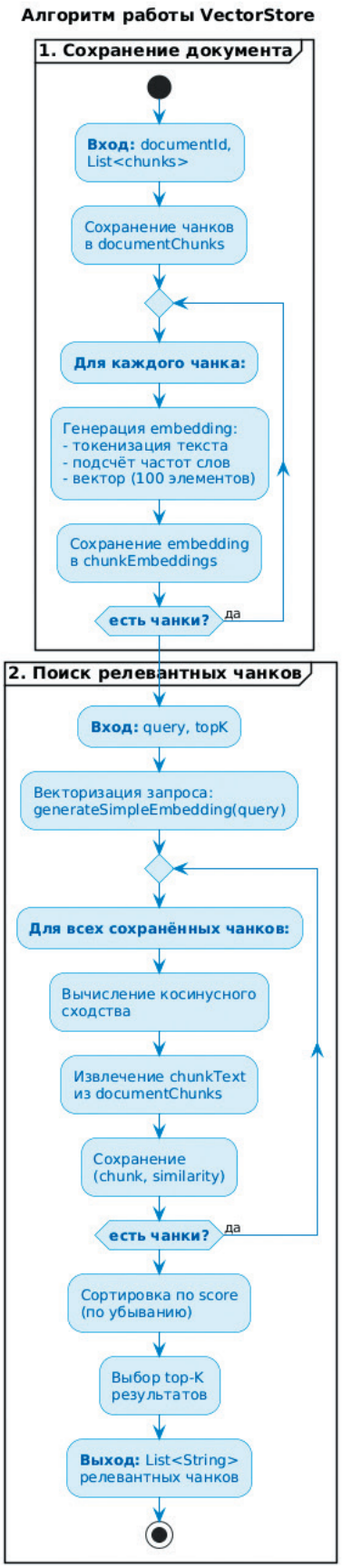

Рис. 5. Реализация метода векторного представления текста с ограничением длины вектора 100 символов

обход массива слов с подсчетом частоты вхождения каждого уникального токена [7]. Для хранения частотного распределения используется ассоциативный массив (словарь), где ключом выступает слово, а значением — количество его вхождений в текст.

3. Инициализация вектора фиксированной размерности. На этой стадии алгоритма создается числовой вектор заранее определенной длины (100 измерений), все элементы которого инициализируются нулевыми значениями. Длина вектора фиксируется, чтобы обеспечить совместимость с векторной базой данных.

4. Проставление значений вектора. Заключительный этап алгоритма включает заполнение исходного массива специальными значениями, которые показывают частоту встречаемых слов. Данный процесс происходит вследствие размещения строковых переменных в векторе последовательно, начиная с нулевой позиции. Так происходит до тех пор, пока либо не закончатся все уникальные слова из общего перечня, либо не будет достигнут допустимый размер вектора.

Результат алгоритма (он же описанный вектор) представляет собой числовую характеристику выбранного фрагмента текста. В таком массиве первые позиции содержат частотные характеристики, а оставшиеся хранят нулевые значения.

Так как обработка данных, которые будут заполнять вектор, не может быть безграничной, важной частью программной системы является эффективное управление памятью. Для минимизации потребления оперативной памяти, которая активно задействуется для локальных языковых моделей, при работе с крупными файлами реализован механизм потоковой обработки данных. Суть механизма состоит в разделении поступающей информации по блокам и их последующей обработке. Предлагаемый подход позволит обрабатывать крупные массивы входной информации, так как с ним программное решение не будет загружать сразу весь массив в память сервера или персонального устройства: вместо этого данные будут обрабатываться последовательно по блокам [8].

Дополнительным решением, принятым при разработке системы для минимизации потребляемых ресурсов, на которое необходимо обратить внимание, стала возможность управления количеством чанков для одного обрабатываемого фрагмента текста. Описанное количество регулируется через параметр, который может редактироваться при запуске ПО.

Отдельно стоит упомянуть встроенный в Java механизм своевременного освобождения ресурсов, который активно задействуется во всех классах программы. Этот механизм реализован с использованием конструкции try-with-resources [9], позволяющей автоматически закрывать сетевые соединения и потоки ввода-вывода при возникновении нестандартных ситуаций.

Можно заключить, что архитектура разработанного решения спроектирована с учетом принципов модульности и открытости/закрытости, что обеспечивает возможность расширения функционала без модификации существующих стабильных компонентов. Добавление новых типов проверок документов реализуется посредством модификации системного промпта, направляемого большой языковой модели. Это позволяет гибко адаптировать логику анализа под специфические требования предметной области без изменения программного кода приложения [10].

Интеграция дополнительных источников знаний осуществляется через расширение интерфейса векторного хранилища, что открывает возможность подключения внешних баз знаний, онтологий или специализированных словарей для повышения точности семантического поиска.

Поддержка мультимодальных документов, содержащих изображения, таблицы и структурированные данные, предусмотрена на уровне модульной архитектуры процессора документов: добавление нового парсера для конкретного формата требует реализации соответствующего обработчика, совместимого с общим интерфейсом извлечения текста, при этом остальные компоненты системы, включая векторное хранилище и клиент взаимодействия с языковой моделью, остаются неизменными.

Описанный подход обеспечивает масштабируемость решения и возможность его адаптации к разнообразным сценариям использования в корпоративных системах документооборота различных отраслей.

Экспериментальная проверка

Экспериментальная проверка разработанной системы проводилась на наборе тестовых документов, включающем заявления на отпуск, служебные записки и транспортные накладные. В документы преднамеренно вносились различные типы ошибок: пропущенные обязательные поля (дата, подпись, Ф. И. О.), орфографические и грамматические ошибки, несоответствия форматов дат и числовых значений, логические противоречия в содержании.

В ходе тестирования были оценены базовые языковые модели, доступные Ollama и не требующие высоких вычислительных ресурсов. К ним относятся Phi-3 Mini, Qwen3b и Llama 3.2. Тесты показали, что Phi3 обладает высокой скоростью обработки, но имеет ограничения в выявлении сложных смысловых противоречий по сравнению с более сбалансированными моделями Qwen3 и Llama 3.2. Наилучшей моделью, обеспечивающей оптимальную производительность при низком потреблении оперативной памяти, стала Llama 3.2 с 3 млрд параметров.

Для оценки эффективности системы использовались метрики полноты ответов, для выявления ошибок от общего числа внесенных ошибок, и точности, характеризующей долю корректно выявленных ошибок от общего числа обнаруженных недочетов. Результаты показали, что среднее значение характеристик точности и полноты находится на уровне 0,85 при выявлении пропущенных фрагментов текста и около 0,78 при обнаружении противоречий. Данные характеристики неотъемлемы от времени отклика системы, которое вычислялось при тестировании текстового документа в 2000 символов. Время отклика системы с языковой моделью Llama 3.2 при проверке различного функционала представлено на рис. 6.

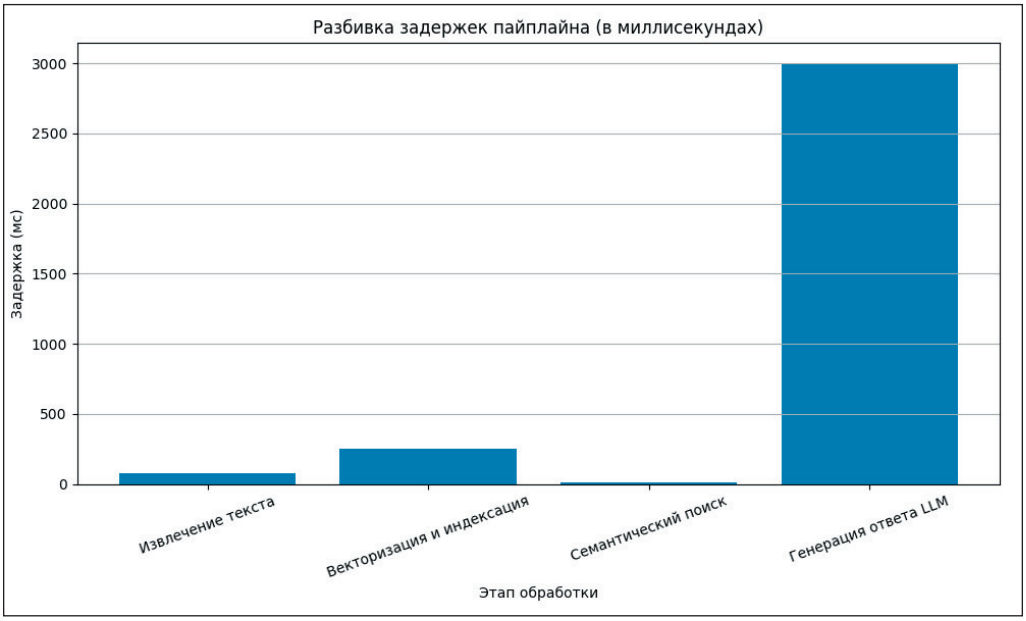

Рис. 6. Результаты тестирования отклика программной системы при использовании модели Llama 3.2

В результате проведенного анализа можно заметить, что общее время работы созданного программного решения не выше 5 сек. при любом режиме работы (от извлечения текста до генерации ответа), следовательно, нормы интерактивного режима работы успешно выполняются.

Заключение

В ходе настоящего исследования было реализовано программное многомодульное решение на языке программирования Java. Основа архитектуры этого приложения включает в себя RAG-алгоритмы, методы для взаимодействия с локальными языковыми моделями, а также специализированные функции для парсинга текста из документов и их последующей обработки. Главное достижение заключается в реализации локального контекстного анализа текстовой информации, поступающей на вход в систему. Разработанное приложение может не только искать ошибки, но и разбирать недочеты в содержании поступающей информации. Эта характеристика позволяет как преодолеть ограничения стандартных языковых моделей, так и сформировать предпосылки для использования данной разработки в уже существующих системах документооборота с повышенными требованиями к конфиденциальности данных.

Система отвечает в среднем за 5 сек., что позволяет использовать ее в интерактивном режиме. При этом не приходится переписывать ядро системы, так как модульная архитектура упрощает добавление новых типов документов. Вместе с тем текущая реализация имеет ряд ограничений: упрощенные эмбеддинги на основе частотного анализа уступают современным трансформерным моделям в точности семантического поиска, а поддержка многоязычных текстов и документов со сложной верстkой требует дополнительной доработки модуля извлечения данных.

Дальнейшее повышение точности выявления смысловых противоречий целесообразно связать с дообучением модели на специализированном корпусе документов транспортной отрасли.

Научная и практическая ценность работы заключается в доказательстве технической реализуемости эффективных систем верификации на базе открытых технологий, что полностью соответствует современным тенденциям суверенизации ИТ-инфраструктуры и снижает зависимость корпоративного сектора от коммерческих облачных API.

СПИСОК ИСТОЧНИКОВ

1. Ротман Д. RAG и генеративный ИИ. Создаем собственные RAG-пайплайны с помощью LlamaIndex, Deep Lake и Pinecon = RAG-Driven Generative AI: Build custom retrieval augmented generation pipelines with LlamaIndex, Deep Lake, and Pinecone. СПб.: Питер, 2025. 320 с.
2. Java SE/JDK Version 26 API Specification: Module java.base. URL: http://docs.oracle.com/en/java/javase/26/docs/api/java.base/module-summary.html (дата обращения: 30.01.2026).
3. Ollama's Documentation. URL: http://docs.ollama.com (дата обращения: 02.02.2026).
4. Шилдт Г. Java. Полное руководство. 12-е изд. = Java: The Complete Reference. Twelfth Edition / пер. с англ. и ред. Ю. Н. Артеменко. СПб.: Диалектика, 2023. 1344 с.
5. Хорстманн К. С. Java. Библиотека профессионала. Т. 1. Основы. 10-е изд. = Core Java. Volume I — Fundamentals. Tenth Edition / пер. с англ. и ред. И. В. Берштейна. М.: Вильямс, 2016. 864 с.
6. Портянкин И. А. Swing. Эффектные пользовательские интерфейсы. 2-е изд. М.: Лори, 2011. 607 с.
7. Huang D., Wang Z. LLMs at the Edge: Performance and Efficiency Evaluation with Ollama on Diverse Hardware // Proceedings of the International Joint Conference on Neural Networks (IJCNN 2025) (Rome, Italy, 30 June — 5 July 2025). Institute of Electrical and Electronics Engineers, 2025. 8 p. DOI: 10.1109/IJCNN64981.2025.11228317
8. Vahaj M., Raza S. M., Nehra V. Retrieval Augmented Generation (RAG) using LLMs // Proceedings of the Annual International Conference on Data Science, Machine Learning and Blockchain Technology (AICDMB 2025) (Mysuru, India, 27–28 June 2025). Institute of Electrical and Electronics Engineers, 2025. 5 p. DOI: 10.1109/AICDMB64359.2025.11277692
9. Блох Д. Java. Эффективное программирование. 3-е изд. = Effective Java. Third Edition / пер. с англ. и ред. И. В. Красикова. СПб.: Диалектика, 2019. 464 с.
10. Claim Verification in the Age of Large Language Models: a Survey / A. Dmonte [et al.] // ArXiv. 2024. Vol. 2408.14317. 9 p. DOI: 10.48550/arXiv.2408.14317

Дата поступления: 26.03.2026
Решение о публикации: 25.05.2026

Implementation of RAG Architecture for Automated Document Verification in Corporate Storage Systems Using Large Language Models

Maksim A. Kostin — 4th year Bachelor's Degree Student in 09.03.01 Informatics and Computer Technology. Research interests: information systems, artificial intelligence, web development. E-mail: m.kkostin@yandex.ru

Dayana M. Davydova — Senior lecturer of the "Information and Computing Systems" Department. Research interests: information systems, big data processing. E-mail: dayana-0820@bk.ru

Vladimir E. Petrov — PhD in Military Sciences, Associate Professor of the "Information and Computing Systems" Department. Research interests: information systems, big data processing, reliability modeling. E-mail: petroffve@mail.ru

Emperor Alexander I St. Petersburg State Transport University, 9 Moskovsky ave., Saint Petersburg, 190031, Russia

For citation: Kostin M. A., Davydova D. M., Petrov V. E. Implementation of RAG Architecture for Automated Document Verification in Corporate Storage Systems Using Large Language Models, *Intellectual Technologies on Transport*, 2026, no. 2 (46), pp. 5–16. DOI: 10.20295/2413-2527-2026-246-5-16 (In Russian)

Abstract. *A specialized solution is offered for analyzing documents used in corporate document management systems.* ***Purpose:*** *to develop and implement a RAG architecture for automated verification of corporate documents*

using locally deployed large language models that identify missing required fields, data format errors, and substantive contradictions in documents of transport and logistics systems. ***Methods:*** *Java application was designed and implemented programmatically, integrating a text extraction module from PDF and DOCX documents based on Apache libraries, a vector storage with simplified embeddings based on frequency analysis of words, a semantic search algorithm through cosine similarity calculation and an LLM interaction client via the Ollama server API.* ***Results:*** *the developed system has demonstrated the ability to contextually analyze the content of documents and adaptability to variable information presentation formats, which makes it possible to overcome the limitations of traditional systems. The experimental verification was performed on a test set of corporate documents with intentionally introduced errors of various types; the effectiveness was assessed by the metrics of completeness, accuracy and their average, as well as by system response time. The llama3.2:latest model showed the best results, while completely excluding the transfer of confidential data outside the organization's infrastructure.* ***Practical significance:*** *the proposed solution is applicable for automation of documentation quality control in corporate document management systems of transport enterprises, government agencies and industrial organizations. The modular architecture provides scalability to other types of documents and the ability to integrate with existing information systems at minimal cost of adaptation. Using open models and a local Ollama server reduces dependence on third-party cloud services and ensures compliance with information security requirements.*

Keywords: *large language models, automated document verification, vector search, natural language processing, corporate systems, Ollama, text extraction, semantic analysis*

REFERENCES

1. Rothman D. RAG i generativnyy II. Sozdaem sobstvennye RAG-payplayny s pomoshchyu LlamaIndex, Deep Lake i Pinecon [RAG-Driven Generative AI: Build Custom Retrieval Augmented Generation Pipelines with Llamaindex, Deep Lake and Pinecone]. Saint Petersburg, Piter Publishing House, 2025, 320 p. (In Russian)

2. Java SE/JDK Version 26 API Specification: Module java.base. Available at: http://docs.oracle.com/en/java/javase/26/docs/api/java.base/module-summary.html (accessed: January 30, 2026).

3. Ollama's Documentation. Available at: http://docs.ollama.com (accessed: February 02, 2026).

4. Schildt H. Java. Polnoe rukovodstvo, 12-e izdanie [Java: The Complete Reference. Twelfth Edition]. Saint Petersburg, Dialektika Publishing House, 2023, 1344 p. (In Russian)

5. Horstmann C. S. Java. Biblioteka professionala. Tom 1. Osnovy. Desyatoe izdanie [Core Java. Volume I — Fundamentals. Tenth Edition]. Moscow, Williams Publishing House, 2016, 864 p. (In Russian)

6. Portyankin I. A. Swing. Effektnye polzovatelskie interfeysy [Spectacular User Interfaces]. Moscow, Lori Publishing House, 2011, 607 p. (In Russian)

7. Huang D., Wang Z. LLMs at the Edge: Performance and Efficiency Evaluation with Ollama on Diverse Hardware, *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2025)*, Rome, Italy, June 30 — July 5, 2025. Institute of Electrical and Electronics Engineers, 2025, 8 p. DOI: 10.1109/IJCNN64981.2025.11228317

8. Vahaj M., Raza S. M., Nehra V. Retrieval Augmented Generation (RAG) using LLMs, Proceedings of the Annual International Conference on Data Science, *Machine Learning and Blockchain Technology (AICDMB 2025)*, Mysuru, India, June 27–28, 2025. Institute of Electrical and Electronics Engineers, 2025, 5 p. DOI: 10.1109/AICDMB64359.2025.11277692

9. Bloch J. Java. Effektivnoe programmirovanie. Tretye izdanie [Effective Java. Third Edition]. Saint Petersburg, Dialektika Publishing House, 2019, 464 p. (In Russian)

10. Dmonte A., et al. Claim Verification in the Age of Large Language Models: A Survey, *ArXiv*, 2024, vol. 2408.14317, 9 p. DOI: 10.48550/arXiv.2408.14317

Received: March 26, 2026
Accepted: May 25, 2026