

Статья поступила в редакцию: 29.08.2026

Статья принята к публикации: 05.09.2026

Статья опубликована: 28.09.2026

УДК 004.89+006.3

DOI: 10.32743/UniTech.2026.150.9.23376

Оптимизация архитектуры интеллектуального ассистента технической документации на базе LLM без использования векторных баз данных

Сперанский Роман Александрович

специалист по информационной безопасности, старший инженер-программист,

независимый исследователь, McAfee Corp

Канада, г. Ванкувер

E-mail: r.speranskii@gmail.com

Optimization of technical documentation smart assistant architecture based on LLM without using vector databases

Speranskii Roman

Information Security Specialist,

Senior Software Engineer, Independent Researcher,

McAfee Corp,

Canada, Vancouver

Аннотация

Цель исследования – обоснование и производственная апробация экономически эффективной архитектуры диалогового ассистента по продуктовой технической документации малого объёма (порядка 5 000 – 20 000 токенов) без применения Retrieval-Augmented Generation (RAG) и векторных баз данных. Методология: спроектирована и внедрена в производственную среду архитектура прямой инъекции статического контекста (Direct Context Injection) с кэшированием префикса промпта (Prompt Caching), ролевым разграничением доступа на уровне промпта и изоляцией модели от внешних инструментов; выполнено аналитическое сравнение с классическим RAG-конвейером по составу инфраструктуры, этапам обработки запроса и режимам отказов; построена расчётная модель стоимости эксплуатации на открытых тарифах LLM-провайдера. Результаты: паттерн исключает конвейер векторизации и векторное хранилище, устраняет по построению класс ошибок неполного извлечения контекста (retrieval miss), обеспечивает контролируемое обновление базы знаний по схеме human-in-the-loop; расчётная стоимость запроса при контексте 15 000 токенов и прогревом кэше – около 0,003 долл. Вывод и практическая значимость: для справочных корпусов малого объёма RAG избыточен; паттерн, критерий

Библиографическое описание: Сперанский Р.А. Оптимизация архитектуры интеллектуального ассистента технической документации на базе LLM без использования векторных баз данных // Universum: технические науки : электрон. научн. журн. 2026. 9(150). URL: <https://7universum.com/ru/tech/archive/item/23376>

выбора архитектуры и модель стоимости применимы разработчиками B2B- и B2C-сервисов для бюджетного внедрения ИИ-ассистентов. Экспериментальное сравнение качества подходов на едином тестовом наборе – направление дальнейшей работы.

Abstract

Purpose of the study: justification and production validation of a cost-effective architecture for a conversational assistant over small-volume product technical documentation (about 5,000-20,000 tokens) without Retrieval-Augmented Generation (RAG) and vector databases. Methodology: a Direct Context Injection architecture with prompt prefix caching, prompt-level role-based access control, and tool-less model isolation was designed and deployed to production; an analytical comparison with the classical RAG pipeline was performed in terms of infrastructure, request processing stages, and failure modes; a verifiable operational cost model was built on the provider's published tariffs. Results: the pattern eliminates the vectorization pipeline and the vector store, removes the retrieval-miss error class by construction, and provides controlled human-in-the-loop knowledge base updates; the calculated cost of a request with a 15,000-token context and a warm cache is about \$0.003. Conclusion and practical significance: for small reference corpora RAG is redundant; the pattern, the corpus-size selection criterion, and the cost model can be adopted by B2B and B2C vendors for rapid, low-cost deployment of AI assistants. Experimental quality comparison on a unified test set is identified as future work.

Ключевые слова: большие языковые модели, RAG, векторные базы данных, кэширование промптов, прямая инжекция контекста, ролевое разграничение доступа, стоимость эксплуатации.

Keywords: large language models, RAG, vector databases, prompt caching, direct context injection, role-based access control, operational cost.

Введение

Интеграция ассистентов на базе больших языковых моделей (LLM) в пользовательские интерфейсы программных продуктов стала стандартом де-факто. Доминирующий паттерн ответов на вопросы по закрытым данным - Retrieval-Augmented Generation (RAG) [5; 8]: корпус фрагментируется (chunking), фрагменты векторизуются и сохраняются в векторной базе данных, при запросе выполняется семантический поиск k релевантных фрагментов, из которых собирается контекст генерации.

Для узкой задачи – ассистента по документации одного продукта малого масштаба – RAG часто избыточен: он требует сопровождения векторного хранилища и конвейера переиндексации и вносит характерный режим отказа – потерю релевантного контекста при извлечении (retrieval miss). Рост контекстных окон LLM, их способность решать задачи по предоставленному контексту без дообучения [4] и штатные механизмы кэширования префикса промпта [3; 6] актуализировали альтернативу – прямую инжекцию полного корпуса в контекст модели (Direct Context Injection, также Context Stuffing). По данным сравнительных исследований, при достаточном бюджете контекста длинноконтекстная генерация в среднем превосходит RAG по качеству, преимуществом RAG остаётся стоимость обработки больших корпусов [9]; известное ограничение длинных контекстов –

неравномерное использование информации в зависимости от позиции («lost in the middle») [10] – мотивирует ограничить область применения прямой инъекции корпусами малого объёма.

Цель исследования - обоснование и производственная апробация паттерна интеграции LLM-ассистента документации без векторных баз данных, объединяющего прямую инъекцию статического контекста, кэширование префикса промпта, ролевое разграничение доступа на уровне промпта и изоляцию модели от внешних инструментов. Задачи: 1) описать архитектуру и её производственную реализацию; 2) выполнить аналитическое сравнение с классическим RAG-конвейером; 3) построить верифицируемую модель стоимости; 4) определить границы применимости и ограничения выводов.

Отдельные элементы решения - длинноконтекстная генерация, кэширование промптов, ролевая модель доступа, отказ от инструментов - известны. Научно-практический результат работы состоит не в новом алгоритме, а в следующем: (а) систематизация этих механизмов в воспроизводимый паттерн для класса задач «ассистент по документации малого объёма»; (б) явный критерий выбора между RAG и прямой инъекцией по объёму корпуса и профилю нагрузки; (в) верифицируемая модель стоимости эксплуатации на открытых тарифах; (г) схема контролируемого обновления базы знаний по принципу human-in-the-loop.

Материалы и методы

Объект исследования. Объектом исследования является справочная подсистема производственного веб-приложения для управления недвижимостью (далее - Acme Property Management; наименование обезличено). База знаний – единый файл help.md в формате Markdown объёмом порядка 15 000 токенов, что типично для справочных систем узкоспециализированных B2B-продуктов (5 000 – 20 000 токенов). Файл состоит из разделов; каждый описывает одну пользовательскую функцию (бронирование объектов, создание здания, регистрация заявки на обслуживание) и имеет единообразную структуру: заголовок, пошаговая инструкция и ролевая аннотация «Who can: ...», задающая на естественном языке перечень ролей (resident, manager, admin), которым доступно действие. При добавлении или изменении функциональности соответствующий раздел редактируется вручную; отдельный конвейер преобразования данных отсутствует – это принципиальное свойство решения.

Метод исследования

Работа выполнена как инженерное исследование в форме производственного кейса в сочетании с аналитическим сравнением архитектур. Решение реализовано на двух стеках (TypeScript, Node.js/Express и Kotlin/Ktor со встроенным HTTP-клиентом JDK) и развёрнуто в производственной среде; генерация выполняется моделью claude-haiku-4-5 через API Anthropic [1]. Сравнение с классическим RAG-конвейером (чанкинг → векторизация → векторная БД → извлечение top-k → сборка промпта) проведено аналитически: по составу инфраструктуры, этапам обработки запроса, режимам отказов и стоимости, с опорой на опубликованные сравнительные исследования [9; 10]. Собственное экспериментальное измерение точности и частоты галлюцинаций двух подходов на едином тестовом наборе не

проводилось и определено как направление дальнейшей работы; количественные утверждения о качестве ограничены результатами цитируемых работ, а собственные количественные результаты статьи относятся к модели стоимости.

Обеспечение изоляции

Изоляция модели от внешних источников реализована на уровне протокола взаимодействия с LLM API: в запросах не передаётся массив tools (function calling), поэтому у модели нет канала для обращения к внешним ресурсам или исполнения кода – её возможности ограничены генерацией текста по переданному контексту. Это программная, а не аппаратная мера: исходящие сетевые вызовы выполняет только серверный компонент к единственной конечной точке LLM API, а ограничение исходящего трафика (egress-политики) остаётся задачей инфраструктурного контура и в настоящей работе не рассматривается. Ассистент вынесен за пределы доверенной зоны: он не выполняет действий в системе, а только формирует справочные ответы; фактическое разграничение прав обеспечивает основной API приложения. Это согласуется с рекомендациями по снижению рисков LLM-приложений [11], в том числе для косвенных инъекций промптов [7]: даже при успешной инъекции через текст вопроса модель не располагает ни инструментами, ни полномочиями в системе.

Результаты и обсуждение

Предлагаемая архитектура

Архитектура сведена к трём логическим узлам (рис. 1): база знаний help.md; серверный слой, выполняющий аутентификацию (JWT), сборку системного сообщения и проксирование запроса к LLM API (ключ API существует только в конфигурации сервера и недоступен клиенту); клиентский чат-виджет, взаимодействующий исключительно с собственным серверным компонентом.

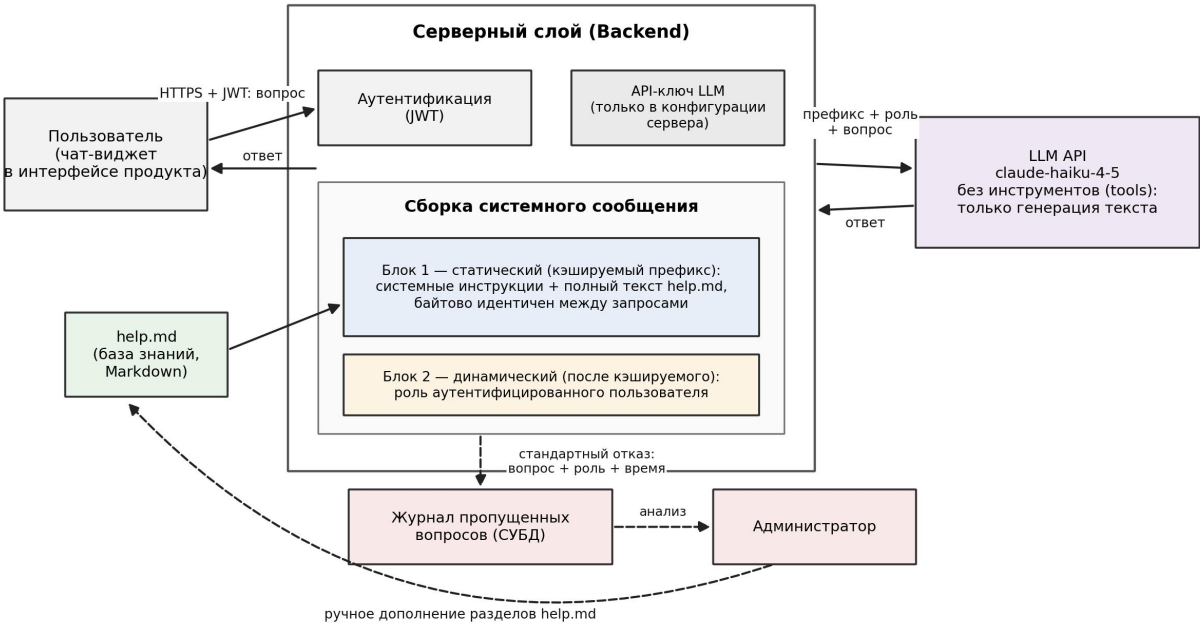


Рисунок 1. Предлагаемая архитектура: формирование системного сообщения, кэшируемый префикс и контур обновления базы знаний

Системное сообщение разделено на две части (табл. 1). Статическая часть - системные инструкции и полный текст документации - помечается атрибутом `cache_control: {type: «ephemeral»}` и остаётся байтово идентичной между запросами, благодаря чему провайдер кэширует обработанный префикс и повторно использует его в последующих вызовах [3]. Динамическая часть - роль аутентифицированного пользователя - размещается строго после кэшируемого блока и в кэш не попадает. Попадание любых волатильных данных (временных меток, идентификаторов запроса) в статический блок приводит к «тихим» промахам кэша; корректность контролируется по полям `usage` ответа API (`cache_creation_input_tokens`, `cache_read_input_tokens`).

Таблица 1.

Структура блоков системного сообщения при вызове LLM API

№	Назначение блока	Изменяемость	Кэширование
1	Системные инструкции и полный текст документации (<docs>...</docs>)	Статический; байтово идентичен между запросами	<code>cache_control: {type: "ephemeral"}</code> - кэшируемый префикс
2	Роль аутентифицированного пользователя	Изменяется от запроса к запросу; размещается строго после кэшируемого блока	Не кэшируется

Листинг 1 демонстрирует формирование тела запроса на языке Kotlin; переменная `systemPrompt` содержит инструкции и полный текст `help.md`, загружаемый однократно при старте сервиса.

```
val payload = AnthropicRequest(  
    model = «claude-haiku-4-5»,  
    maxTokens = 1024,
```

```

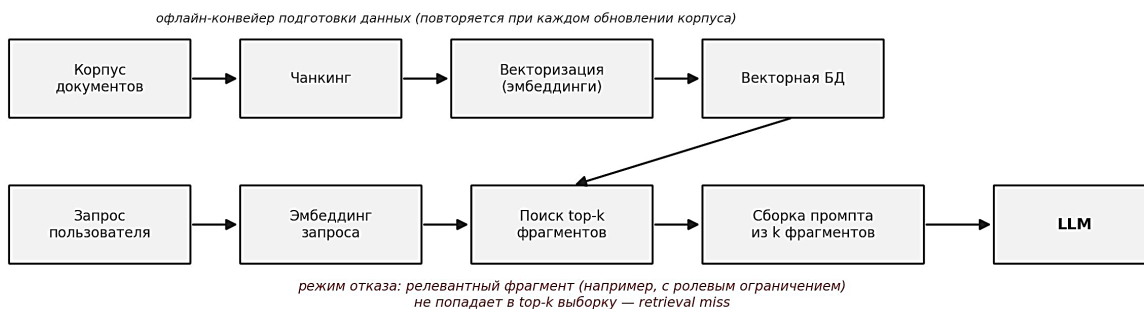
system = listOf(
  SystemBlock(
    text = systemPrompt,
    cacheControl = CacheControl(type = «ephemeral»)
  ),
  SystemBlock(text = «The person asking has the role: $role.»)
),
messages = listOf(Message(«user», question))
)

```

Ролевое разграничение реализовано так: системные инструкции предписывают модели отвечать только по документации, отказываться от нерелевантных запросов и не описывать шаги, недоступные роли спрашивающего; носителем модели доступа выступают аннотации «Who can:» в самой документации, что исключает дублирование матрицы прав в коде ассистента. Граница применимости: это эргономическая мера (не подсказывать пользователю недоступные ему действия), а не механизм безопасности - фактическую авторизацию операций выполняет основной API продукта.

Сравнение с RAG-конвейером

а) Классический RAG-конвейер



б) Прямая инъекция контекста с кэшированием префикса (предлагаемый подход)

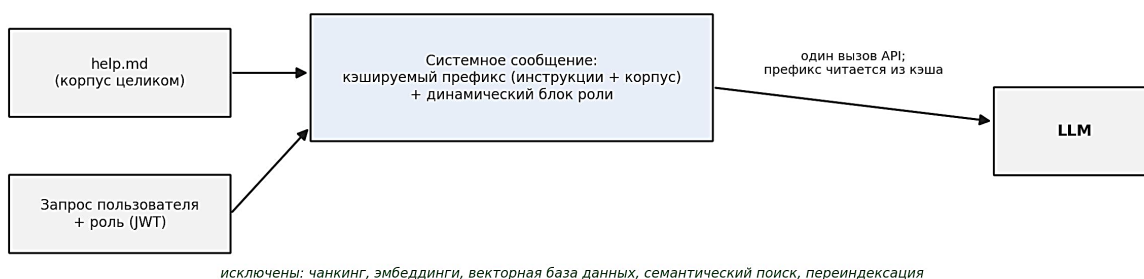


Рисунок 2. Сравнение конвейеров обработки запроса: классический RAG (а) и прямая инъекция контекста с кэшированием префикса (б)

Сравнительная матрица этапов обработки запроса приведена в таблице 2. Прямая инъекция устраняет по построению класс ошибок неполного извлечения (retrieval miss), при котором релевантный фрагмент - например, содержащий ролевое ограничение - не попадает в top-k выборку: модель всегда получает корпус целиком, что согласуется с результатами [9]. Вместе с тем полный контекст сам по себе не гарантирует полного использования

содержащейся в нём информации [10] и не исключает генеративных галлюцинаций; инструктивное ограничение (отвечать только по документации, отказ по умолчанию), отсутствие внешних источников и малый объём корпуса снижают риски, однако количественная оценка частоты галлюцинаций требует отдельного эксперимента (см. подраздел «Ограничения»).

Таблица 2.

Сравнительная матрица обработки запроса

Этап	Классический RAG	Предлагаемая архитектура
Подготовка данных	Чанкинг → векторизация → загрузка в векторную БД; переиндексация при каждом обновлении корпуса	Однократная сборка файла help.md; обновление - правка раздела файла
Клиентский слой	HTTP-запрос с JWT	HTTP-запрос с JWT
Серверный слой	Векторизация запроса → поиск top-k → сборка промпта из фрагментов	Агрегация статического префикса и роли пользователя
Вызов LLM API	Генерация на основе k извлечённых фрагментов	Чтение кэшированного префикса (~15 тыс. токенов) → генерация

Разграничение авторского вклада

Штатные механизмы, используемые без модификации: языковая модель и её API [1], кэширование префикса промпта [3], аутентификация JWT. Авторские проектные решения: 1) структура системного сообщения с разделением на кэшируемый статический и некашируемый ролевой блоки и дисциплина байтовой стабильности префикса; 2) аннотации «Who can:» внутри Markdown-документации как единственный носитель модели доступа ассистента; 3) tool-less конфигурация как осознанная мера изоляции модели; 4) схема контролируемого обновления базы знаний с журналированием отказов; 5) критерий выбора архитектуры по объёму корпуса. Совокупность этих решений и составляет предлагаемый паттерн; утверждений о создании нового алгоритма семейства RAG или нового механизма кэширования работа не содержит.

Модель стоимости эксплуатации

Расчёт выполнен по опубликованным тарифам claude-haiku-4-5 [2] на дату обращения: базовые входные токены - 1,00 долл. за 1 млн; выходные - 5,00; запись кэша (TTL 5 минут) - 1,25; чтение кэша - 0,10. Сценарий: кэшируемый префикс (инструкции и документация) - 15 000 токенов; средняя длина ответа - 300 токенов; вопрос пользователя мал по сравнению с префиксом и в оценке опущен; кэш прогрет - интервалы между запросами не превышают TTL, продлеваемого при каждом обращении, что типично для частотного пользовательского трафика.

Стоимость одного запроса при прогревом кэше: $C = 15\,000 / 10^6 \times 0,10 \text{ долл.} + 300 / 10^6 \times 5,00 \text{ долл.} = 0,0015 + 0,0015 = 0,003 \text{ долл.}$ Первый («холодный») запрос дополнительно оплачивает запись кэша: $15\,000 / 10^6 \times 1,25 \text{ долл.} + 0,0015 \text{ долл.} \approx 0,020 \text{ долл.}$ Сравнение режимов на горизонте 1 000 запросов приведено в таблице 3; в отличие от предыдущих редакций расчёта, входные токены при кэшировании учитываются в полном объёме - они продолжают передаваться, но тарифицируются по ставке чтения кэша.

Таблица 3.

Расчётная стоимость обработки 1 000 запросов (префикс 15 000 токенов, ответ 300 токенов, тарифы claude-haiku-4-5 [2])

Режим	Входные токены на запрос и тарификация	Вход, долл.	Генерация, долл.	Итого, долл.
Прямая инъекция без кэширования	15 000; базовый тариф, 1,00 долл./1 млн	15,00	1,50	≈ 16,50
Прямая инъекция с кэшированием (предлагаемый)	15 000; чтение кэша, 0,10 долл./1 млн	1,50	1,50	≈ 3,00 *
RAG, top-3 фрагмента (аналитическая оценка)	≈ 2 000; базовый тариф, 1,00 долл./1 млн	2,00	1,50	≈ 3,50 **

* дополнительно ≈ 0,02 долл. - однократная запись кэша при холодном запросе (15 000 × 1,25 долл./1 млн); ** без учёта затрат на векторизацию корпуса и запросов, а также постоянных затрат на инфраструктуру векторной базы данных и сопровождение переиндексации.

Из таблицы 3 следует, что при малых корпусах токенные затраты подходов сопоставимы: преимущество паттерна создаётся не тарифами на генерацию, а устранением инфраструктурных компонентов (векторная БД, сервис эмбедингов) и этапов сопровождения (переиндексация). Показатель 0,003 долл. не является универсальной константой метода: он линейно зависит от объёма префикса (при 100 000 токенов чтение кэша даёт 0,010 долл., итого ≈ 0,0115 долл. за запрос), от длины ответа и доли попаданий в кэш; при разреженном трафике с интервалами больше TTL стоимость смещается к 0,020 долл. за запрос. Все составляющие расчёта верифицируемы по полям usage ответов API и открытым тарифам [2; 3].

Контролируемое обновление базы знаний

Вместо автономного «самообучения» бота, поведение которого не поддаётся аудиту, реализована пассивная петля обратной связи по принципу human-in-the-loop. Если ответ ассистента - стандартный отказ («могу помочь только по вопросам продукта...»), серверный компонент фиксирует вопрос, роль пользователя и время в реляционной БД; администратор периодически анализирует журнал пропущенных вопросов и вручную дополняет разделы help.md, а обновлённый файл автоматически образует новый кэшируемый префикс при следующем запросе. Каждое изменение проходит через человека и фиксируется в истории файла - обновление базы знаний контролируемо и трассируемо. Детерминированной генерацию ответов это не делает: стохастичность выборки токенов свойственна самой языковой модели; корректной характеристикой схемы является контролируемость обновления знаний, а не детерминированность ответов.

Ограничения

Ограничениями настоящей работы являются: 1) один производственный продукт и один провайдер LLM - выводы о стоимости зависят от тарифной политики провайдера и подлежат пересчёту при её изменении; 2) отсутствие собственного экспериментального сравнения качества RAG и прямой инъекции на едином тестовом наборе - утверждения о качестве

опираются на цитируемые внешние исследования [9; 10]; 3) выводы ограничены фактически исследованным диапазоном объёмов корпуса (порядка 5 000-20 000 токенов). Экстраполяция на корпуса в 50 000-100 000 токенов следует из линейности модели стоимости и результатов [9], однако до экспериментальной проверки должна рассматриваться как гипотеза. Дальнейшая работа: открытый тестовый набор запросов трёх классов (по интерфейсу, нерелевантные, с нарушением ролевой модели), референсный RAG-контур на том же корпусе и измерение точности, частоты галлюцинаций и задержки обоих подходов.

Заключение

Для справочных корпусов малого объёма (порядка 5 000-20 000 токенов) классическая RAG-архитектура избыточна: паттерн прямой инъекции статического контекста с кэшированием префикса воспроизводит целевую функциональность ассистента без конвейера подготовки данных и дополнительной инфраструктуры, устраняет по построению ошибки неполного извлечения контекста и обеспечивает расчётную стоимость запроса около 0,003 долл. в описанных условиях. Ролевое разграничение на уровне промпта и tool-less конфигурация формируют консервативный профиль безопасности, а петля human-in-the-loop - контролируемое обновление базы знаний.

Предложенный паттерн, критерий выбора архитектуры по объёму корпуса и верифицируемая модель стоимости непосредственно применимы разработчиками B2B- и B2C-сервисов для быстрого и бюджетного внедрения ассистентов по документации. Количественное сравнение качества с RAG на едином тестовом наборе и уточнение верхней границы применимости паттерна по объёму корпуса - предмет дальнейших исследований.

Список литературы

1. Anthropic PBC. Introducing Claude Haiku 4.5. – 2025.обращения: 31.08.2026) [Электронный ресурс]. URL: <https://www.anthropic.com/news/claude-haiku-4-5>(дата. (дата обращения 24.09.2026)
2. Anthropic PBC. Pricing. Claude Developer Documentation [Электронный ресурс]. – 2026. URL: <https://docs.anthropic.com/en/docs/about-claude/pricing> (дата обращения: 31.08.2026).
3. Anthropic PBC. Prompt caching. Claude Developer Documentation.обращения: 31.08.2026) [Электронный ресурс]. URL: <https://docs.anthropic.com/en/docs/build-with-claude/prompt-caching>(дата. (дата обращения 24.09.2026)
4. Brown T.B., Mann B., Ryder N. et al. Language models are few-shot learners // Advances in Neural Information Processing Systems. – 1901. – Т. 33. – P. 1877–1901.
5. Gao Y., Xiong Y., Gao X. et al. Retrieval-augmented generation for large language models: a survey // arXiv preprint arXiv:2312.10997. – 2023. – DOI: 10.48550/arXiv.2312.10997 [Электронный ресурс]. URL: <https://arxiv.org/abs/2312.10997> (дата обращения: 31.08.2026).
6. Gim I., Chen G., Lee S., Sarda N., Khandelwal A., Zhong L. Prompt cache: modular attention reuse for low-latency inference // Proceedings of Machine Learning and Systems. – 2024. – Т. 6. – P. 325–338.

7. Greshake K., Abdelnabi S., Mishra S., Endres C., Holz T., Fritz M. Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection // Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23). – 2023. – P. 79–90. – DOI: 10.1145/3605764.3623985.
8. Lewis P., Perez E., Piktus A. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks // Advances in Neural Information Processing Systems. – 2020. – Т. 33. – P. 9459–9474.
9. Li Z., Li C., Zhang M., Mei Q., Bendersky M. Retrieval augmented generation or long-context LLMs? A comprehensive study and hybrid approach // Proceedings of the. – 2024. – P. 881–893. – DOI: 10.18653/v1/2024.emnlp-industry.66.
10. Liu N.F., Lin K., Hewitt J. et al. Lost in the middle: how language models use long contexts // Transactions of the Association for Computational Linguistics. – 2024. – Т. 12. – P. 157–173. – DOI: 10.1162/tac1_a_00638.
11. OWASP Foundation. OWASP Top 10 for Large Language Model Applications: 2025 [Электронный ресурс]. – 2024. URL: <https://owasp.org/www-project-top-10-for-large-language-model-applications/> (дата обращения: 31.08.2026).