**УП Применение LLM**

***Выполнили Тищенко Е. В. и Арзамазов Н. А***

**Цели и бэйзлайн**

Целью проекта стало создание интеллектуального Telegram-бота, который способен отвечать на вопросы пользователей по различным текстам и новостным материалам. При этом в настройках бота предусмотрена возможность выбора между разными языковыми моделями, чтобы пользователь мог подобрать наиболее подходящую для своих задач или сравнить качество работы различных вариантов. В качестве источников информации используются заранее загруженные тексты — это могут быть как новостные статьи, так и тематические подборки по интересующим направлениям. Таким образом, бот совмещает удобный пользовательский интерфейс, гибкость настройки и современные возможности генеративных моделей для поиска и предоставления релевантных ответов.

В основе архитектуры нашего QA-бота лежит связка больших языковых моделей (LLM) и подхода Retrieval-Augmented Generation (RAG). Начнем с того, что LLM по своей природе обучаются на огромных корпусах текстов, собранных до определенного момента времени, после чего их знания становятся статичными. Таким образом, если попытаться получить от такой модели ответ на вопрос о свежих новостях или о событиях, произошедших недавно, она окажется бессильна, поскольку у нее нет доступа к интернету и актуальным данным.

**Как работает наш RAG**

Чтобы решить эту проблему, в проекте была реализована архитектура RAG, в основе которой лежит идея дополнения языковой модели свежей информацией, взятой из внешних источников. Мы организовали автоматическую обработку и загрузку новостных статей из разных СМИ: тексты сначала разбиваются на небольшие фрагменты, после чего каждый такой фрагмент преобразуется в векторное представление (эмбеддинг) с помощью специализированной модели. Для разбиения длинных текстов на чанки используется класс RecursiveCharacterTextSplitter, который реализован, например, в библиотеке LangChain. Этот инструмент предназначен для максимально эффективного разбиения текста на куски фиксированной длины с учетом структуры самого текста. В отличие от простого нарезания текста по количеству символов, RecursiveCharacterTextSplitter сначала старается разделить текст по более логичным границам — например, по абзацам или предложениям, а если это невозможно из-за слишком большого размера

чанка, то делит дальше уже по символам. Такой подход помогает избежать разрыва смысловых единиц, повышает качество эмбеддингов и, как следствие, улучшает релевантность поиска.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

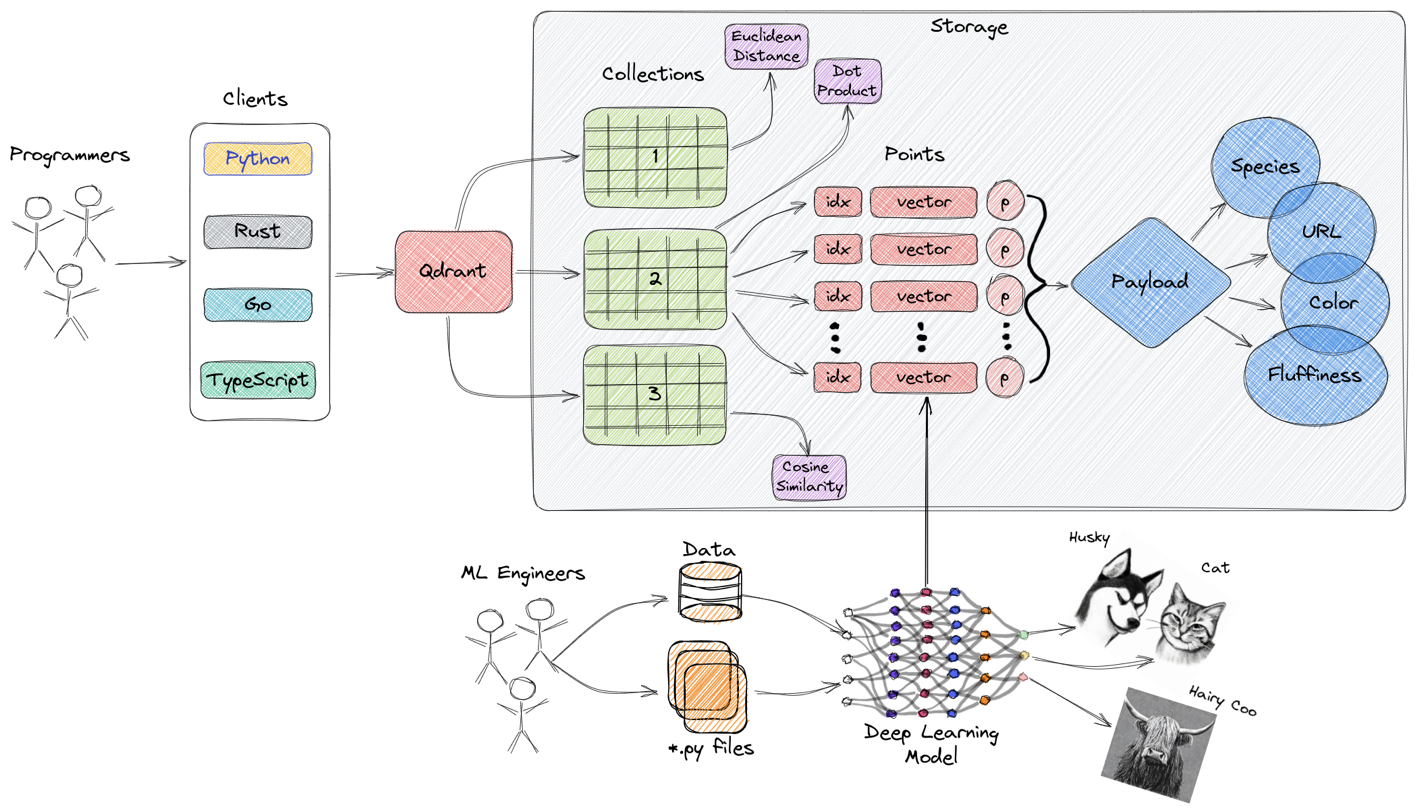
Содержимое, созданное искусственным интеллектом, может быть неверным.

Схема работы RAG в целом

Полученные эмбеддинги сохраняются во внешней векторной базе данных, благодаря чему становятся доступны для быстрого поиска. Когда пользователь задает вопрос, этот вопрос также преобразуется в эмбеддинг. После этого система с помощью семантического поиска находит в базе наиболее релевантные фрагменты новостей и «подкладывает» их в контекст языковой модели вместе с исходным вопросом. Благодаря такой схеме языковая модель может сгенерировать осмысленный ответ, опираясь не только на свои «замороженные» знания, но и на реально актуальную информацию из внешних источников. Это позволяет боту отвечать даже на вопросы о событиях, произошедших буквально несколько часов назад.

Технически весь пайплайн реализован следующим образом. В качестве интерфейса с пользователем выступает Telegram-бот, созданный на базе библиотеки python-telegram-bot. Данный выбор обусловлен тем, что библиотека поддерживает полностью асинхронный код, что обеспечивает высокую скорость реакции, а также обладает интуитивно понятным API и уже была хорошо знакома команде. Далее новости, поступающие из парсера, проходят этап обработки: тексты разбиваются на отдельные смысловые чанки с помощью упомянутого RecursiveCharacterTextSplitter, после чего каждый чанк отправляется в отдельный сервис на FastAPI, реализующий Embedder API. Этот сервис отвечает за получение эмбеддингов — в нем развернута модель intfloat/multilingual-e5-small, которая выделяется хорошей скоростью работы, поддержкой множества языков, включая русский, и компактным размером, что позволяет запускать ее на обычных серверах без мощного оборудования. Инференс реализован с использованием базовых библиотек transformers и torch, без специализированных фреймворков для инференса, которые часто требуют наличия GPU.

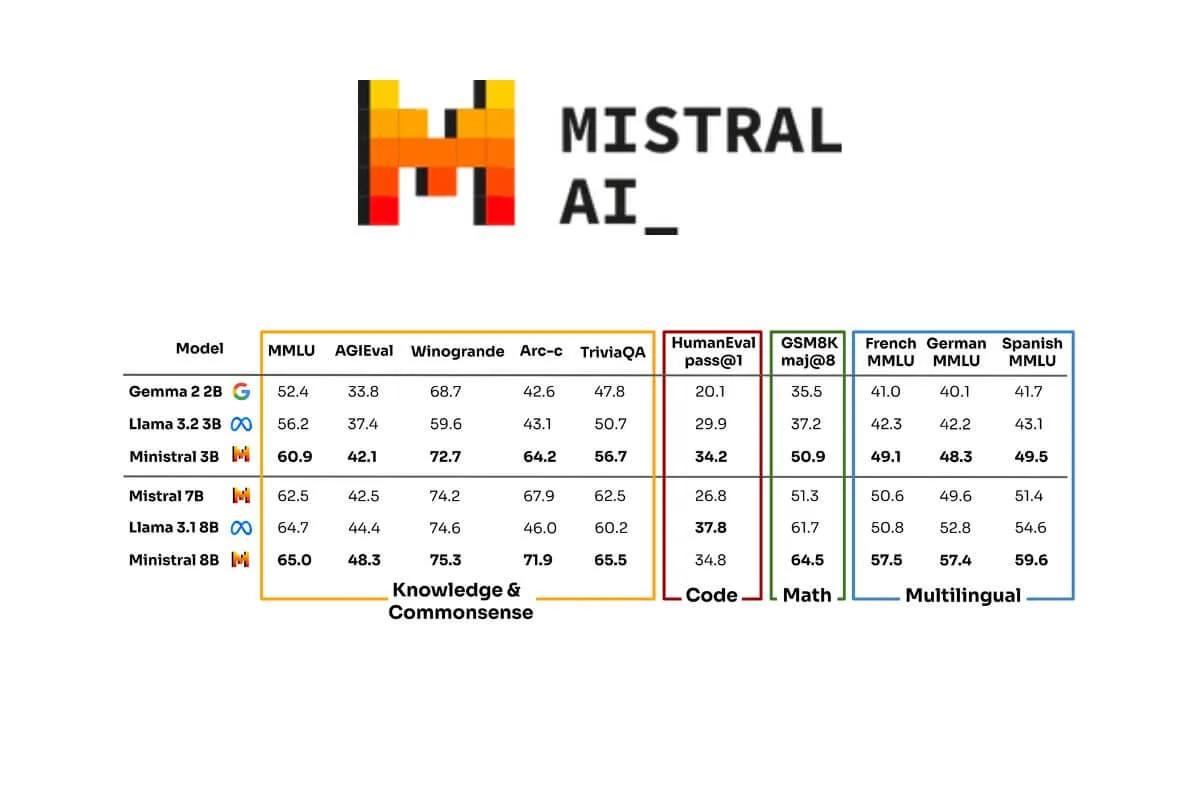
Для хранения и поиска эмбеддингов мы выбрали векторную базу данных Qdrant. Такое решение было принято благодаря высокой производительности Qdrant, написанного на языке Rust, и чрезвычайно простой процедуре развертывания — для запуска требуется всего один контейнер. Взаимодействие сервисов с базой происходит через Qdrant SDK для Python, что позволяет легко интегрировать функционал семантического поиска прямо в приложение. Также Qdrant обладает современным web-интерфейсом для администрирования, удобными инструментами отладки и возможностью фильтрации по метаданным. Для поиска наиболее подходящих новостей используется стандартная метрика косинусного расстояния между эмбеддингами.



Возможности Qdrant

На этапе генерации ответа, изначально для LLM мы использовали модель Vikhr-Qwen-2.5-1.5B-Instruct-GGUF — это относительно компактная языковая модель, которую можно запускать даже на серверах без мощных GPU, и она дообучена на русскоязычных данных. Однако в ходе дальнейшей работы мы перешли к использованию Ollama для поднятия LLM. Это решение позволило значительно упростить развертывание языковой модели и повысить стабильность работы. С появлением новых облегченных моделей от Meta мы стали тестировать Llama 3-1B — благодаря поддержке Ollama ее удалось быстро внедрить в пайплайн, и сейчас основная LLM-часть системы работает именно на Llama 3-1B. Данная модель отличается отличным качеством на небольшом объеме ресурсов, быстро запускается и хорошо справляется с задачами генерации, особенно в связке с качественным retrieve. Это модель и находится в нашем продакшине.

Но, для тестирования наших пайплайнов и простоты разработки мы остановили выбор на Mistral API. Одним из ключевых преимуществ Mistral является то, что их API практически полностью совместимо с OpenAI API, который де-факто стал стандартом индустрии для взаимодействия с языковыми моделями и интеграции инструментов. Благодаря этому процесс внедрения и интеграции максимально прост — можно легко использовать существующие библиотеки и пайплайны, разработанные для OpenAI. Более того, Mistral активно развивает open source модели.



Сравнение Mistral 8b с другими моделями

**Выводы по бэйзлайну и его ограничения**

Таким образом, вся архитектура QA-бота построена на принципе дополнения языковой модели актуальной информацией из внешних источников. Однако, несмотря на высокую гибкость и расширяемость такого решения, у данного подхода есть ограничения.

Во-первых, качество поиска релевантных фрагментов напрямую зависит от выбранной модели для создания эмбеддингов. Если модель не справляется с задачей выделения смысла или плохо работает на каком-либо языке, результаты поиска могут оказаться нерелевантными, что ухудшает качество финального ответа.

Во-вторых, чтобы бот всегда был в курсе последних событий, требуется регулярно обновлять коллекцию новостей в базе: для этого реализован парсер, который по расписанию опрашивает выбранные новостные источники. Оптимально запускать парсер каждые 10–30 минут, чтобы новостная база всегда оставалась свежей, однако при таком подходе база быстро наполняется огромным количеством документов. Это требует внедрения механизма автоматической очистки базы — например, с помощью TTL для каждой новости или регулярных скриптов, удаляющих устаревшие данные. В противном случае в базе будет скапливаться слишком много неактуальных или дублирующихся записей, что ухудшит скорость и точность поиска. Кроме того, этот подход не избавлен от так называемой «мертвой зоны»: если пользователь задает вопрос о событии, которое только что произошло, а парсер еще не успел загрузить новую статью, то бот не сможет дать корректный ответ — база знаний всегда немного отстает от реального времени. И, наконец, вся система в целом является довольно ресурсоемкой: для нормального функционирования требуется поддерживать работу как минимум пяти-шести отдельных сервисов и контейнеров — бота, самой языковой модели, модели для создания эмбеддингов, векторной базы данных Qdrant, парсера и вспомогательных API. Все это увеличивает требования к инфраструктуре и усложняет развертывание и сопровождение решения.

**Что еще попробовали**

Перед тем как пойти дальше, хочется поделиться некоторыми экспериментами, которые мы проводили в ходе разработки данного решения. В первую очередь мы тестировали различные варианты retrieval. Например, пробовали использовать гибридный поиск, то есть совмещать семантический поиск по эмбеддингам с классическими алгоритмами по типу BM-25. На практике это не дало прироста качества — классические ключевые слова, наоборот, зашумляли retrieve и мешали модели давать релевантные ответы, так как в контекст попадало слишком много нерелевантной информации.

Мы также экспериментировали с различными моделями эмбеддингов: тестировали intfloat/multilingual-e5-large, bge-m3 и даже варианты с SentenceTransformers. В ряде случаев большие модели действительно давали лучшее качество поиска, но были слишком тяжелыми для наших серверов без GPU. Мы пробовали менять параметры чанкинга текстов — варьировали размер фрагментов, чтобы находить оптимальный баланс между полнотой retrieve и объемом информации, который получает LLM на входе. Отдельный эксперимент касался обработки длинных вопросов: мы пробовали перефразировать запросы, чтобы повысить точность поиска в базе.

**Tool Calling**

В процессе проектирования архитектуры мы изначально рассматривали вариант с регулярным обновлением базы новостей с помощью парсера, запускаемого по расписанию (например, через cron). Однако довольно быстро стало понятно, что этот подход не решает ключевых проблем: между появлением новости и ее попаданием в базу все равно образуется временное окно, в течение которого бот оказывается не в курсе самых свежих событий. К тому же, такой способ связан с избыточным расходом ресурсов и затрудняет масштабирование — особенно если количество источников новостей возрастает.

В результате мы отказались от идеи обновления данных по расписанию и начали искать более современное и гибкое решение, позволяющее получать свежую информацию по запросу, а не с заданной периодичностью. Проанализировав современные подходы, мы обратили внимание на так называемый *tool calling* — механизм, позволяющий языковой модели напрямую инициировать вызовы внешних инструментов или API по мере необходимости. Такой подход активно используется в современных LLM-агентах, например, в сервисе Perplexity.ai.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Схема работы tool calling в целом

Суть механизма *tool calling* заключается в том, что языковая модель, получив вопрос пользователя, может автоматически сформировать структурированный запрос (обычно в формате JSON) к внешней функции — например, к поисковому API. При этом LLM лишь формирует запрос, а его исполнение берет на себя внешний оркестратор, который затем возвращает результат обратно в модель для дальнейшей генерации ответа.

В качестве поискового инструмента для генерации свежей информации мы интегрировали Tavily Search API — это специализированный сервис, оптимизированный для LLM, с готовой фильтрацией и ранжированием результатов, что критично для генерации корректных и современных ответов.

Также, на этапе выбора архитектурных решений мы протестировали ряд популярных агентских фреймворков (например, smolagents, PydanticAI и аналогичные инструменты), которые позволяют строить сложные цепочки вызовов и динамически управлять множеством тулзов. Эти фреймворки отлично подходят для задач, где нужно комбинировать сразу несколько внешних функций, памяти и различных API в одном пайплайне. Однако в нашем случае требовался только один внешний инструмент — поисковый API, поэтому использование полноценного агентного фреймворка выглядело избыточным и излишне усложняло логику проекта. Мы убедились, что для конкретной задачи проще и надежнее реализовать обработку *tool calling* “вручную” через стандартный API LLM, без дополнительного слоя абстракций. Такой подход сделал код проще, легче для поддержки и повысил стабильность всей системы.

Итоговый пайплайн выглядит следующим образом: когда пользователь задает вопрос, языковая модель анализирует его и, если считает необходимым, формирует tool call для поиска через Tavily. Оркестратор обрабатывает этот запрос, получает свежие релевантные данные и передает их обратно языковой модели, которая уже на их основе генерирует финальный ответ пользователю. Такой подход обеспечивает максимальную актуальность информации, гибкость архитектуры и возможность дальнейшего масштабирования или переноса на собственные решения без значительных трудозатрат.

**Бенчмарк**

Для оценки нашего подхода мы составили бенчмарк из 21-го вопроса по ключевым темам — от денежно-кредитной политики ЦБ РФ и макроэкономических показателей до новостной повестки и экономической теории. Оценивались ответы двух моделей: mistral-large-latest и mistral-8b-latest (первая побольше и вторая поменьше).

Предложенные вопросы включали как фундаментальные знания и официальные формулировки, так и вопросы, требующие актуальных данных (новости последнего дня, свежие показатели, стоимость валюты на конкретные даты и др.). Под верными ответами мы здесь подразумеваем в первую очередь фактически верные и полные ответы, всякого рода полу-верные ответы здесь считаем за неверные.

Результаты следующие:

Большая модель *mistral-large-latest* с *tool calling*– 14 верных из 21 вопроса

Маленькая модель *ministral-8b-latest* с *tool calling* – 12 верных из 21 вопроса

Бэйзлайн с RAG *mistral-large-latest* – 8 верных из 21

| **№** | **Вопрос** | **Tool Calling mistral-large-latest** | **Tool Calling ministral-8b-latest** | **RAG** |
| --- | --- | --- | --- | --- |
| 1 | Какова текущая ключевая ставка Банка России и как она изменилась за последние 12 месяцев? | Да | Да | Да |
| 2 | Какой сейчас в России уровень среднегодовой инфляции? | Нет | Нет | Нет |
| 3 | В чем заключается основная цель ЦБ РФ? | Да | Да | Нет |
| 4 | Каков официальный прогноз роста ВВП на 2025 год от Минэкономразвития? | Да | Да | Да |
| 5 | Как Банк России использует операционные ориентиры для достижения инфляционного таргета? | Да | Да | Нет |
| 6 | Как изменилась структура валютных резервов России за последние два года? | Нет | Нет | Нет |
| 7 | Какие ограничения действуют на трансграничные валютные операции в 2025 году? | Да | Нет | Нет |
| 8 | Какие страны являются крупнейшими торговыми партнёрами России в 2025 году? | Да | Нет | Нет |
| 9 | Каков был объём экспорта России в 2024 году? | Нет | Нет | Нет |
| 10 | Какие страны ввели персональные санкции против ЦБ РФ в 2022 году? | Да | Да | Да |
| 11 | Какие ключевые риски для финансовой стабильности ЦБ РФ выделяет в последнем отчете? | Нет | Нет | Да |
| 12 | Каков был курс рубля к доллару США на 10 июня 2025 года? | Да | Да | Нет |
| 13 | Каков был курс рубля к доллару США на 10 мая 2025 года? | Да | Да | Да |
| 14 | Какие банки входили в топ-3 по объёму активов в России в 2023 году? | Да | Да | Нет |
| 15 | Каковы тенденции в банковском кредитовании населения и бизнеса в 2024–2025 годах? | Нет | Нет | Нет |
| 16 | Какой сейчас индекс Мосбиржи и дай краткое пояснение такому положению | Нет | Нет | Да |
| 17 | Есть ли какие-то новости за последние два дня о снижении ставок по вкладам? | Да | Да | Да |
| 18 | Как отреагировала Мосбиржа на новости о переговорах в Стамбуле? | Да | Да | Да |
| 19 | Как ЦБ РФ интерпретирует разницу между фактической и ожидаемой инфляцией в коммуникациях с общественностью? | Да | Да | Нет |
| 20 | Где можно найти методологическое описание расчета индекса инфляционных ожиданий? | Нет | Нет | Нет |
| 21 | Зачем в принципе нужен цифровой рубль? | Да | Да | Нет |

Очевидно, на вопросах на самые актуальные темы (ставка, курс, прогноз, новые лимиты, кто в новостях, свежие публикации) система справляется лучше классического RAG на внутренней базе, у которого на эти вопросы могут отсутствовать актуальные чанки, либо они могут не успеть подгрузиться из-за технической невозможности обновлять их слишком часто, либо возможно парсер, которым мы собираем эти чанки попросту не ходит в нужный для данного вопроса источник, в то время как наш пайплайн с походом в поисковую систему решает этот вопрос.

Ошибки происходят по вопросам, где требуется глубокий регламент/официальная методика (например: "где найти официальное описание расчета метода X", если в выдаче нет прямой ссылки или поисковый движок не находит специфическую страницу, а классический RAG с “официальным архивом” возможно бы просто выдал нужный документ).

Также ошибки бывают связаны с тем, что даже наша большая модель в сущности довольно слабая и если к нашей системе прикрутить, например, через API более мощную LLM, то и ответы будут более точные и содержательные, так как чанки полученные с поисковой выдачи будут обрабатываться более осмысленно. Это мы можем заметить, по вопросам 7 и 8, чтобы ответить на них, модели нужно получить на вход сразу много новостей и данных связанных с ограничениями на трасграничные переводы и структурой экспорта России, и суммаризировав их выдать ответ. Поэтому слабая модель справляется с этим хуже большой, а значит, возможно если взять еще более мощную модель, мы сможем намного улучшить метрики даже на нашем весьма ограниченном бенчмарке.

Основные примеры преимуществ подхода *tool calling* перед базовым RAG:

*Пример 1.* Курс рубля, данные на конкретную дату, последние прогнозы, рейтинги/банки — всё это поступающая оперативная информация.

В классической RAG базе это либо устарело (не обновлено), либо наш парсер может упускать важные новые источники.

В нашем случае даже вопрос “Курс рубля/доллара на 10 мая/июня” (вплоть до “нет информации”, если дата ещё не наступила) — система дает правильный, свежий ответ или корректно отказывает в выдаче (см. вопрос 12 и 13).

*Пример 2.* Новости о снижении ставок, свежая реакция рынка, санкции, новые ограничения.

Наш RAG на поиске находит сведения о снижении ставок за последние два дня (свежие новости), что труднее реализуемо для статически наполняемого локального архива документов.

*Пример 3.* Банковский рынок, топ-3 банков, прогнозы Минэкономразвития.

Оперативная интеграция нового прогноза/обновленных данных не требует ожидания обновления базы. Официальные публикации, новые релизы — подгружаются чаще, чем их успеют внести в классический knowledge base.

Также следует выделить слабые места таког подхода, которые видны на нашем бенчмарке:

* Релевантность поиска. Иногда поисковый движок может “промахнуться” по точной странице (особенно по узкоспециализированным вопросам, методологиям), тогда как строгий фиксированный датасет чанков может содержать нужные регламенты.
* Информационный шум. Не все snippets релевантны; непомешал бы фильтр качества выдачи/дополнительный ранкинг.
* Валидация на критичных темах. Некоторые ответы требуют строгой точности (например, по нормативным актам). Для таких разделов классический RAG может быть сильнее.

Таким образом, предложенный подход с *tool calling* и интеграцией внешнего поиска хорошо себя зарекомендовал при работе с новостями и быстро обновляемой информацией, а его эффективность напрямую зависит от качества внешних источников, возможностей поискового API и мощности используемой языковой модели. Тем не менее, для ряда задач (например, по строго формализованным регламентам и методологиям) классический RAG с тщательно отобранной внутренней базой может оставаться предпочтительным вариантом. Но так как наша задача была разработать пайплайн именно новостного Telegram-бота, то мы остановились на *tool calling* подходе.