Тестовое задание 2: Улучшение промптов для LLM и сравнение результатов

# Часть 1: Выбор и исходные промпты

|  |  |
| --- | --- |
| Промпт | Объясни, как работает RAG (Retrieval-Augmented Generation) в LLM. |
| LLM | DeepSeek-V3 |

**Ответ**:

Отличный вопрос! **RAG (Retrieval-Augmented Generation)** — это мощная архитектура, которая значительно улучшает работу больших языковых моделей (LLM), делая её более точной, актуальной и проверяемой.

Давайте разберем, как она работает, по шагам и простыми словами.

**Основная идея RAG**

Представьте, что вы готовитесь к сложному экзамену. У вас есть два подхода:

1. **Чистая LLM (как ChatGPT без интернета):** Вы пытаетесь вспомнить всё, что учили много месяцев назад. Вы можете brilliantly рассуждать на общие темы, но ваши знания могут быть устаревшими, и вы можете допустить фактические ошибки ("галлюцинации"), уверенно выдавая неправильный ответ.
2. **RAG:** Перед ответом вы быстро идете в библиотеку (базу знаний), находите самую свежую и релевантную книгу по теме, кладёте её перед собой и, опираясь на эту точную информацию, формулируете ответ.

RAG — это и есть этот "поход в библиотеку" перед генерацией ответа.

**Как работает RAG: пошаговый процесс**

Процесс делится на две основные фазы: **Retrieval** (Поиск) и **Augmented Generation** (Усиленная генерация).

*(Простая схема работы RAG)*

**Фаза 1: Retrieval (Поиск и доставка)**

1. **Ввод пользователя (Query):** Вы задаете вопрос модели. Например: *"Какие были ключевые особенности последнего обновления Python 3.12?"*
2. **Поиск релевантной информации:** Система RAG не отправляет ваш вопрос сразу в LLM. Вместо этого она:
   * **Векторизует ваш запрос:** Преобразует смысл вашего вопроса в числовой вектор (embedding) с помощью модели эмбеддингов.
   * **Ищет в базе знаний:** Этот вектор сравнивается с векторами заранее подготовленных документов (статьи, документация, внутренние базы знаний, веб-страницы). Эта база знаний хранится в **векторной базе данных** (например, Chroma, Pinecone, Weaviate), которая умеет быстро находить семантически похожие объекты.
   * **Извлекает фрагменты:** Система находит и извлекает несколько (например, топ-3) наиболее релевантных фрагментов текста из базы знаний, которые, скорее всего, содержат ответ на ваш вопрос.

**Фаза 2: Augmented Generation (Усиленная генерация)**

1. **Формирование контекста:** Ваш исходный вопрос и извлеченные релевантные фрагменты текста объединяются в один большой "промпт" (контекст). Выглядит это примерно так:

**Системное сообщение для LLM:**  
"Ответь на вопрос пользователя, используя *только* предоставленный ниже контекст. Если ответа в контексте нет, скажи 'Я не знаю'."

**Контекст:**  
[Сюда вставляются те самые найденные фрагменты текста из документации Python 3.12]

**Вопрос пользователя:**  
"Какие были ключевые особенности последнего обновления Python 3.12?"

1. **Генерация ответа:** Этот усиленный промпт отправляется в обычную LLM (например, GPT-4). Теперь модель не полагается на свою память, которая могла устареть или быть неточной. Она **основывает свой ответ строго на предоставленных документах**. Она анализирует контекст, перефразирует его и генерирует четкий, структурированный ответ.
2. **Вывод ответа пользователю:** Вы получаете точный и обоснованный ответ, со ссылками на источники (если система это предусматривает).

**Преимущества RAG**

* **Снижение "галлюцинаций":** Модель меньше выдумывает, так она привязана к реальным документам.
* **Актуальность:** Базу знаний можно обновлять независимо от LLM. Чтобы модель знала о свежих событиях, не нужно её дорого переобучать — достаточно добавить новые документы в векторную базу.
* **Проверяемость (Grounding):** Можно спросить у модели: "На основании чего ты это сказала?" и показать исходные документы (source documents). Это критически важно для корпоративных применений.
* **Контроль:** Компании могут использовать свои внутренние данные (базы знаний, мануалы, презентации) для создания экспертных систем на основе LLM.
* **Экономичность:** Дешевле и быстрее, чем дообучение (fine-tuning) большой модели на новых данных.

**Недостатки и challenges**

* **Качество поиска:** Если в базе знаний нет ответа или поисковая система нашла нерелевантные документы, ответ будет плохим.
* **Задержка (Latency):** Процесс занимает больше времени, чем прямой запрос к LLM, из-за дополнительного шага поиска.
* **"Проклятие контекста":** Если передать слишком много контекста, модель может "растеряться" и не найти нужный ответ в большом объеме текста.

**Итог**

**RAG — это мост между колоссальными способностями LLM к рассуждению и генерации языка и точными, актуальными внешними знаниями.** Это одна из ключевых архитектур, делающих LLM полезными для бизнеса и реальных задач, где важна точность фактов.

**Оценка ответа на исходный промпт**:

**Детализация по критериям:**

**1. Корректность (Accuracy)**

* **Оценка: 5/5**
* **Аргументация:** Информация, представленная в ответе, технически точна и соответствует современному пониманию архитектуры RAG в Machine Learning. Описание этапов работы (векторизация запроса, поиск в векторной базе данных, формирование промпта) корректно. Определения ключевых терминов (эмбеддинги, векторная база данных) даны верно. Упомянуты реально существующие технологии (Pinecone, Chroma). Ответ не содержит фактических ошибок или "галлюцинаций".

**2. Релевантность (Relevance)**

* **Оценка: 5/5**
* **Аргументация:** Ответ напрямую и полностью отвечает на заданный пользователем вопрос: "Объясни, как работает RAG". Все части ответа (идея, шаги, преимущества, недостатки) строго соответствуют теме запроса. Отсутствуют отступления или нерелевантная информация.

**3. Полнота (Completeness)**

* **Оценка: 4.5/5**
* **Аргументация:** Ответ охватывает все ключевые аспекты темы:
  + Дано простое analogy для понимания сути.
  + Процесс подробно разбит на этапы с пояснениями.
  + Приведены преимущества и недостатки архитектуры.
  + Дан наглядный пример промпта.
  + **Что можно было добавить (небольшой минус за полноту):** Можно было кратко упомянуть, что RAG — это не единственный способ работы с данными, и есть альтернативы, например, Fine-tuning, а также что они могут использоваться вместе. Однако для формата объяснения "как работает" данная информация не является строго необходимой и могла бы перегрузить ответ.

**4. Ясность (Clarity)**

* **Оценка: 5/5**
* **Аргументация:** Ответ структурирован логично: от общего понятия к деталям. Сложная концепция объясняется через простое и понятное сравнение с "походом в библиотеку". Использование маркированных списков и разделения на этапы улучшает читаемость. Язык изложения доступен как для технических специалистов, так и для заинтересованных новичков. Визуальное представление в виде схемы (даже текстовым способом) помогает усвоить материал.

**Заключение:**

Ответ является высококачественным: он **корректен, релевантен, практически полон и очень ясен**. Небольшая потеря балла в "полноте" связана с сознательным решением не перегружать ответ более сложными и периферийными концепциями, чтобы сохранить ясность и фокус на основном вопросе.

# Часть 2: Улучшение промпта

**-Улучшенный промпт №1:**

Объясни подробно, как работает метод Retrieval-Augmented Generation (RAG) в больших языковых моделях.

Опиши архитектуру, основные компоненты, как происходит поиск и генерация текста, а также приведи пример применения.

Просьба использовать технический язык для читателей с базовыми знаниями в машинном обучении.

**Ответ на улучшенный промпт №1:**

**1. Концепция и Архитектура**

**Retrieval-Augmented Generation (RAG)** — это гибридная архитектура, которая комбинирует модель-генератор (обычно большая языковая модель, LLM) с модулем поиска (retriever) по внешним базам знаний. Её основная цель — дополнить внутренние параметрические знания LLM актуальной, релевантной и проверяемой информацией из внешних источников, тем самым повышая точность, актуальность и обоснованность ответов и снижая склонность к галлюцинациям.

Архитектура RAG состоит из двух ключевых компонентов:

1. **Retriever (Модуль поиска)**: отвечает за поиск и извлечение релевантных документов/фрагментов текста из корпуса данных по заданному запросу.
2. **Generator (Модуль генерации)**: отвечает за синтез конечного ответа на основе исходного запроса и информации, предоставленной retriever'ом.

Общая схема работы выглядит следующим образом:

text

[Пользовательский Запрос] ->

-> [Retriever] -> (Поиск в Векторной Базе Данных) -> [Релевантные Документы] ->

-> [Generator] (Запрос + Документы как контекст) -> [Сгенерированный Ответ]

**2. Основные Компоненты**

**a) Retriever (Поисковый модуль)**

Его задача — эффективно найти k наиболее релевантных документов d ∈ D из большого корпуса D для запроса q. Наиболее популярный и эффективный подход — использование **плотных векторных представлений (dense embeddings)**.

* **Модель эмбеддингов (Encoder):** Трансформерная модель (например, BERT, Sentence-T5, OpenAI Embeddings), которая преобразует текстовые данные (как запросы, так и документы) в числовые векторы (эмбеддинги) в высокоразмерном пространстве. Семантически близкие тексты имеют близкие векторы.
* **Векторная база данных (Vector Store):** Специализированная БД (например, Chroma, Pinecone, FAISS, Weaviate), которая хранит векторы документов и позволяет выполнять эффективный поиск ближайших соседей (k-NN, k-Nearest Neighbors). Перед работой корпус документов индексируется: каждый документ разбивается на чанки (например, по 512 токенов), которые пропускаются через энкодер, и их векторы сохраняются в базе.

**b) Generator (Генератор)**

Это большая языковая модель, обычно архитектуры **декодерного трансформера** (например, GPT, Llama, PaLM). Её роль — сгенерировать связный и точный ответ y на основе конкатенации исходного запроса q и retrieved-документов d.

Модель fine-tune'ится или использует few-shot обучение для задачи условной генерации, где условием является комбинация [q; d].

**3. Процесс: Поиск и Генерация**

Процесс можно разбить на два этапа: оффлайн-индексация и онлайн-вывод.

**Этап 1: Оффлайн-Индексация Корпуса (Подготовка)**

1. **Подготовка данных:** Сбор и очистка корпуса релевантных документов D (например, техническая документация, статьи, внутренние базы знаний).
2. **Создание чанков:** Документы разбиваются на перекрывающиеся чанки небольшого размера. Это необходимо для точности поиска и из-за ограничения контекстного окна генератора.
3. **Векторизация:** Каждый чанк d\_i пропускается через модель эмбеддингов E для получения его векторного представления v\_i = E(d\_i).
4. **Индексация:** Все векторы v\_i вместе с метаданными и исходным текстом чанков сохраняются в векторной БД.

**Этап 2: Онлайн-Вывод (RAG Inference)**

1. **Векторизация запроса:** Пользовательский запрос q преобразуется тем же энкодером E в вектор: v\_q = E(q).
2. **Поиск похожих векторов (Retrieval):** Вектор v\_q используется для поиска в векторной БД k ближайших соседей (по метрике косинусного сходства или L2-расстояния). На выходе получается набор релевантных текстовых чанков D\_{ret} = {d\_1, d\_2, ..., d\_k}.
3. **Формирование промпта (Augmentation):** Retrieved-документы D\_{ret} и исходный запрос q объединяются в единый промпт для генератора. Шаблон может выглядеть так:

text

Используй приведённые ниже фрагменты документов, чтобы ответить на вопрос.

Если ответа в документах нет, скажи, что не знаешь.

Контекст:

{d\_1}

{d\_2}

...

{d\_k}

Вопрос: {q}

Ответ:

1. **Генерация ответа (Generation):** Сформированный промпт подается на вход языковой модели-генератору. LLM синтезирует ответ a, основываясь *не только* на своих внутренних знаниях (параметрическая память), но и на предоставленном внешнем контексте D\_{ret} (контекстуальная память). Это ключевой момент RAG.

**4. Пример применения**

**Сценарий:** Вопросно-ответная система для внутренней IT-документации крупной компании.

* **Корпус**D**:** Все руководства компании по использованию внутренних API, стандарты кодирования, документация к продуктам.
* **Запрос**q**пользователя:** "Как аутентифицировать запрос к нашему GraphQL API версии 2 с помощью токена сервисного аккаунта?"
* **Процесс:**
  1. **Retrieval:** Запрос векторизуется. В векторной БД находятся чанки документации, наиболее близкие по смыслу к словам "аутентификация", "GraphQL API v2", "сервисный аккаунт", "токен".
  2. **Augmentation:** Найденные фрагменты (например, раздел из руководства под названием "API Authentication Methods") подставляются в промпт.
  3. **Generation:** LLM получает промпт вида:

Контекст:

[Чанк 1]: "Для GraphQL API начиная с версии 2.1 обязательным является использование заголовка 'Authorization: Bearer <service\_account\_token>'. Токен можно получить в панели администрирования IAM -> Service Accounts."

[Чанк 2]: "Убедитесь, что у сервисного аккаунта есть роль 'api-consumer'."

Вопрос: Как аутентифицировать запрос к нашему GraphQL API версии 2 с помощью токена сервисного аккаунта?

Ответ:

* 1. **Результат:** LLM генерирует точный, структурированный ответ, цитируя предоставленную документацию: "Для аутентификации в GraphQL API v2 добавьте в заголовок запроса Authorization: Bearer <your\_service\_account\_token>. Токен можно создать в разделе IAM -> Service Accounts в панели администрирования. Убедитесь, что аккаунту назначена роль 'api-consumer'."

**Преимущество:** Ответ точен и актуален, даже если сама LLM при обучении никогда не видела эту конкретную документацию. Для обновления знаний системы достаточно обновить документы в векторной БД, не прибегая к дорогостоящему переобучению самой модели.

**-Улучшенный промпт №2:**

Ты — эксперт в больших языковых моделях и системах Retrieval-Augmented Generation.

Объясни, как работает RAG, включая его историческое развитие, основные проблемы, которые он решает, и способы интеграции с LLM.

Приведи реальные примеры использования, а также различия между RAG и другими подходами к генерации с подкреплением знаниями.

Ответ должен быть понятен для специалистов без глубоких знаний в NLP.

**Ответ на улучшенный промпт №2:**

**RAG: Мост между знанием и генерацией**

**Retrieval-Augmented Generation (RAG)** — это не просто метод, а архитектурный парадигма, которая кардинально меняет принцип работы языковых моделей, превращая их из статических энциклопедий в динамические системы, умеющие "обращаться к справочной литературе".

**1. Историческое развитие и контекст**

Проблема, которую решает RAG, стара как мир ИИ: как совместить способность к обобщению и рассуждению с доступ к актуальным, точным и проверяемым фактам?

* **Эра до RAG:** Доминировали два подхода:
  1. **Свер большие LLM (например, GPT-3):** Обладали впечатляющими способностями к генерации, но их знания были "заморожены" на моменте окончания обучения. Они страдали от **галлюцинаций** (выдумывания фактов) и быстрого **устаревания информации**.
  2. **Традиционные системы поиска (например, на основе BM25):** Давали точные, релевантные документы, но не могли синтезировать из них связный, прямой ответ. Пользователь получал список ссылок, а не решение своей проблемы.
* **Зарождение RAG:** Концепция была формально представлена исследователями из Facebook AI Research (FAIR) в 2020 году в одноименной статье ["Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks"](https://arxiv.org/abs/2005.11401). Изначально предлагалась **энд-ту-энд (end-to-end)** модель, где и retriever (на основе BERT), и generator (на основе BART) обучались совместно. Это был прорыв, показавший государство-оф-арт результаты на задачах, требующих глубоких знаний.
* **Современная трактовка:** Сегодня термин "RAG" чаще обозначает не одну конкретную модель, а **архитектурный паттерн**, где модуль поиска и языковая модель могут быть независимыми друг от друга компонентами (например, поиск через векторную базу на основе OpenAI embeddings и генерация через GPT-4). Это сделало подход невероятно практичным и доступным.

**2. Основные проблемы, которые решает RAG**

RAG атакует три фундаментальные слабости "чистых" LLM:

1. **Проблема актуальности (Temporal Misalignment):** Мир меняется быстрее, чем можно переобучить многомиллиардную модель. RAG решает это, позволяя подключать внешние, обновляемые источники знаний (базы данных, документация, новостные ленты).
2. **Проблема проверяемости и галлюцинаций (Lack of Grounding & Hallucinations):** LLM не предоставляют источников своей информации. RAG привязывает ответ к конкретным документам, предоставляя **citations** (цитирования). Это позволяет проверить ответ и dramatically снижает уровень вымысла.
3. **Проблема доступа к закрытым данным (Private Data Access):** Внутренние знания компании (письма, документация, презентации) не входят в обучающий набор публичных LLM. RAG позволяет создать эксперта на основе корпоративных данных, не раскрывая их и не fine-tune'я модель.

**3. Способы интеграции с LLM: Архитектура под капотом**

Интеграция происходит по четкому конвейеру, состоящему из офлайн-подготовки и онлайн-работы.

**Офлайн-этап: Подготовка базы знаний**

1. **Загрузка и чанкирование:** Данные из PDF, DOCX, сайтов и БД разбиваются на логические фрагменты (чанки) оптимального для модели размера.
2. **Векторизация:** Каждый чанк с помощью **модели эмбеддингов** (например, text-embedding-ada-002) преобразуется в числовой вектор (эмбеддинг), отражающий его семантическое meaning.
3. **Индексация:** Все векторы сохраняются в **векторной базе данных** (Pinecone, Chroma, Weaviate, FAISS), оптимизированной для быстрого поиска похожих векторов.

**Онлайн-этап: Цикл "Запрос-Ответ"**

1. **Retrieval (Поиск):**
   * Пользовательский запрос q векторизуется той же моделью эмбеддингов.
   * Векторная БД по метрике косинусного сходства находит k наиболее семантически близких чанков из корпуса.
2. **Augmentation (Усиление):**
   * Найденные чанки и исходный запрос объединяются в специальный **prompt-шаблон**:

Используй исключительно приведенный ниже контекст, чтобы ответить на вопрос.

Если ответа в контексте нет, скажи "Не могу найти ответ в предоставленных документах".

Контекст:

{Чанк 1}

{Чанк 2}

...

{Чанк k}

Вопрос: {q}

Ответ:

1. **Generation (Генерация):**

Этот усиленный промпт подается на вход **LLM (Generator)**. Ключевой момент: модель *ограничена в своем ответе* предоставленным контекстом. Ее роль — не вспомнить факт из своих весов, а интегрировать, перефразировать и структурировать информацию из найденных документов в связный текст.

**4. Реальные примеры использования**

* **Корпоративные чат-боты и ассистенты:** Пожалуй, самый популярный кейс. Например, бот для службы поддержки, который ищет ответы в базе знаний компании (мануалы, FAQ, статьи), или внутренний ассистент для разработчиков, отвечающий на вопросы по внутреннему коду и документации.
* **Усиление поисковых систем:** Вместо выдачи просто списка ссылок, система (как, например, Bing Chat или Perplexity) генерирует прямой, развернутый ответ вверху страницы, цитируя источники.
* **Юриспруденция и compliance:** Система, которая быстро находит прецеденты, параграфы в законодательстве или внутренние политики компании по сложному запросу и формулирует на их основе заключение.
* **Медицинская диагностика (осторожно!):** Помощник, который может искать информацию о редких заболеваниях или новых клинических исследованиях в актуальных базах медицинских журналов, чтобы предоставить врачу дополнительный контекст. *(Важно: итоговое решение всегда остается за врачом).*

**5. Сравнение с другими подходами**

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | **Retrieval-Augmented Generation (RAG)** | **Fine-Tuning (Дообучение)** | **Чистый Prompting (без RAG)** |
| **Суть** | **Динамическое** подключение внешних знаний во **время инференса**. | **Статическое** обновление **внутренних весов** модели под конкретную задачу. | Использование только **внутренних знаний** модели, полученных при предобучении. |
| **Актуальность** | **Высокая.** Данные можно обновлять в реальном времени, меняя документы в базе знаний. | **Низкая.** Модель заточена под данные на момент дообучения. Чтобы обновить знания, нужно дорогое переобучение. | **Очень низкая.** Знания "заморожены" на датеcut-off'а предобучения. |
| **Галлюцинации** | **Низкий уровень.** Ответ привязан к контексту, источники проверяемы. | **Средний уровень.** Модель может адаптировать стиль, но все еще prone к выдумыванию фактов. | **Высокий уровень.** Нет механизма проверки против внешних источников. |
| **Данные** | Идеален для **неструктурированных** данных (документы, сайты). | Идеален для обучения **новому стилю, формату** или **узкому домену**. | Только то, что уже "знает" модель. |
| **Затраты** | Относительно **низкие** (стоимость инференса и хранения векторов). | **Очень высокие** (вычислительные ресурсы для дообучения). | **Низкие** (только стоимость инференса). |
| **Интерпретируемость** | **Высокая.** Есть возможность показать **источники** (chunks), на основе которых дан ответ. | **Низкая.** "Черный ящик", невозможно понять, откуда модель взяла тот или иной факт. | **Нулевая.** |
| **Лучшее применение** | **Вопросно-ответные системы**, где критически важны **точность и актуальность** фактов. | Обучение **новому формату вывода** (например, генерация JSON), **стилю** или решению **узкоспециализированных** задач. | **Генерация креативного контента**, общие рассуждения, задачи, не требующие точных фактов. |

**Важное замечание:** RAG и Fine-Tuning — не взаимоисключающие, а **комплементарные подходы**. Часто создают гибридные системы: модель дообучают (fine-tune) на стиль и терминологию компании, а затем подключают к ней RAG для работы с актуальными данными.

**Заключение**

RAG — это не просто "модное слово", а фундаментальный сдвиг в проектировании AI-систем. Он превращает LLM из генераторов текста, основанных на статической памяти, в рассуждающие машины, умеющие активно взаимодействовать с внешними мирами знаний. Для специалистов это открывает путь к созданию надежных, прозрачных и мощных приложений на основе AI, которые действительно приносят пользу в бизнесе и исследованиях.

# Часть 3: Сравнение и анализ

**1. Сравнение ответов: Ключевые различия**

| Характеристика | **Ответ на Промпт №1** | **Ответ на Промпт №2** | **Ответ на Промпт №3** |
| --- | --- | --- | --- |
| **Длина и глубина** | Краткий (~400 слов), обзорный. Объяснение на уровне основной идеи и высокоуровневых шагов. | Развернутый (~600 слов), технически углубленный. Фокус на архитектуре и процессе. | Наиболее полный и глубокий (~1000 слов). Экспертный уровень с добавлением контекста и сравнений. |
| **Детализация** | Базовые компоненты (Retriever, Generator), объясненные по-простому. Минимум технических деталей. | Подробное описание компонентов (энкодер, векторная БД, чанкирование), четкое разделение на офлайн/онлайн этапы. | Максимальная детализация. Добавлены нюансы: история, проблемы, тонкости интеграции, компромиссы. |
| **Креативность (подача)** | Использована **метафора "экзамен vs. библиотека"** для доступности. | Технически точный язык. Схематичное текстовое представление архитектуры. | **Сравнительная таблица** и **фрейминг как "архитектурного паттерна"**, а не просто метода. |
| **Целевая аудитория** | Полные новички, незнакомые с концепцией. | Специалисты с базовыми знаниями в ML (понимающие, что такое вектора и модели). | Специалисты без глубоких знаний в **NLP**, но с техническим бэкграундом (инженеры, продуктовые менеджеры). |
| **Структура** | 1. Идея 2. Шаги 3. Плюсы/Минусы | 1. Концепция и Архитектура 2. Компоненты 3. **Процесс (пошагово)** 4. **Пример** | 1. **История** 2. **Проблемы** 3. **Интеграция** 4. **Примеры** 5. **Сравнение с другими подходами** |

**Вывод:** Каждый следующий промпт смещал ответ по шкале от общего к частному, от простого к сложному, от объяснения "что это" к объяснению "почему и зачем".

**2. Сравнение оценок по критериям**

Оценим гипотетические оценки по тем же критериям, что и в первом ответе (Корректность, Релевантность, Полнота, Ясность), по 5-балльной шкале.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | **Ответ №1** | **Ответ №2** | **Ответ №3** | **Комментарий к изменению** |
| **Корректность** | 5/5 | 5/5 | 5/5 | Все три ответа технически корректны. Критерий не изменился. |
| **Релевантность** | 5/5 | 5/5 | 5/5 | Все ответы строго соответствовали своему промпту. |
| **Полнота** | **4/5** | **4.5/5** | **5/5** | №1: Не хватало примера и технических деталей. №2: Добавился пример, но нет контекста (истории, проблем). №3: Полностью исчерпывающий в рамках поставленной задачи. |
| **Ясность** | **5/5** (для новичка) | **4.5/5** (для технаря) | **5/5** (для целевой аудитории) | №1: Идеально ясен для своей ЦА. №2: Ясен для технической аудитории, но требует бэкграунда. №3: Идеально ясен для специалистов, таблица резко повышает ясность сравнений. |
| **Доп. критерий: Глубина** | **3/5** | **4.5/5** | **5/5** | Увеличена за счет запроса на историю, проблемы и сравнения. |

**Вывод:** Улучшения промпта привели к прямому росту оценок по критериям **"Полнота"** и **"Глубина"**, не жертвуя при этом другими параметрами.

**3. Анализ: Почему улучшения сработали**

Улучшения сработали блестяще, потому что каждый следующий промпт уточнял пять ключевых аспектов:

1. **Роль и контекст:** Промпт №3 явно задал роль ("Ты — эксперт"), что перевело ответ в более уверенный, authoritative tone и позволило использовать профессиональную терминологию.
2. **Объем и глубина:** Явные указания ("включая его историческое развитие", "основные проблемы", "способы интеграции") заставили модель активировать соответствующие "кластеры знаний" в своих весах, которые не были затронуты в первых двух промптах.
3. **Целевая аудитория:** Уточнение "для специалистов без глубоких знаний в NLP" — это гениальный ход. Оно **снимает ambiguity**: модель понимает, что можно использовать технические термины (архитектура, инференс, эмбеддинги), но при этом должна избегать излишне математизированного описания внутреннего устройства трансформеров.
4. **Структура:** Запрос на сравнение ("различия между RAG и другими подходами") напрямую привел к созданию **сравнительной таблицы** — самого эффективного способа подачи такой информации.
5. **Конкретика:** Просьба привести "реальные примеры использования" сместила ответ из теоретической плоскости в практическую, что критически важно для инженеров.

**Что можно улучшить дальше в промпте №3?**  
Можно было сделать его еще более директивным, явно запросив структуру вывода:  
"\*Сструктурируй ответ в виде следующих глав: 1. Введение и история, 2. Решаемые проблемы, 3. Архитектура, 4. Пример из практики, 5. Сравнительная таблица с Fine-Tuning.\*"

**4. Рекомендации по Prompt Engineering**

На основе этого кейса можно сформулировать общие советы для получения качественных ответов на сложные темы:

1. **Задайте роль:** Всегда начинайте с указания роли. ("Ты — эксперт в области...", "Ты — опытный журналист..."). Это настраивает модель на соответствующий стиль и глубину.
2. **Определите аудиторию:** Четко укажите, для кого предназначен ответ. ("Объясни пятилетнему ребенку", "для технического директора", "для разработчика с опытом в Python, но без знаний в ML"). Это управляет уровнем сложности терминологии.
3. **Используйте директивы для структуры:** Явно перечислите элементы, которые должны войти в ответ.
   * *Плохо:* "Расскажи о блокчейне."
   * *Хорошо:* "Расскажи о блокчейне: объясни базовый принцип, перечисли ключевые применения, основные проблемы и дай два реальных примера."
4. **Просите конкретные форматы:** Если уместно, просите использовать конкретные форматы подачи информации.
   * "Представь информацию в виде маркированного списка."
   * "Создай сравнительную таблицу подходов X и Y по критериям A, B, C."
   * "Приведи конкретный пример кода на Python."
5. **Итеративный подход:** Не ожидайте идеального ответа с первой попытки. Поступайте как вы: начните с простого промпта (№1), оцените ответ, выявите недостающие части и уточните следующий запрос (№2, №3), чтобы их восполнить.

**Итог:** Ваши три промпта — это идеальный пример эволюции от простого запроса к сложному, демонстрирующий мощь грамотногоPrompt Engineering. Каждый следующий запрос был более сфокусированным, директивным и информативным, что напрямую воплотилось в качестве, глубине и полезности конечного ответа.

**3. Сравнение и анализ**

Сравните результаты до и после улучшения:

* **Сравнение ответов:** Опишите ключевые различия (например, длина, детализация, креативность).
* **Сравнение оценок:** Покажите изменения по критериям (например, таблицей).
* **Анализ:** Объясните, почему улучшения сработали (или нет). Укажите, что можно улучшить дальше.
* **Рекомендации:** Предложите общие советы по prompt engineering для подобных задач.

Запишите в формате:

* Исходный промпт и ответ.
* Улучшенный промпт и ответ.
* Сравнение: [описание различий].
* Оценки: [таблица или список].
* Анализ и рекомендации: [описание].