Руководство Google по промпт-инжинирингу

Часть 1: основы и базовые техники

От переводчика

Представляю вашему вниманию перевод статьи "Prompt Engineering" (Промпт-инжиниринг) авторства Lee Boonstra - Software Engineer Tech Lead, Office of the CTO в Google. Это первая часть из планируемого цикла из трех статей, поскольку оригинальный документ весьма объёмен (68 страниц) и насыщен полезной информацией.

Важно отметить, что оригинальная публикация фокусируется в основном на моделях Gemini и сервисе Vertex AI от Google, однако описанные принципы, техники и советы универсальны и применимы практически ко всем современным большим языковым моделям (GPT, Claude, Llama и др.). Концепции промпт-инжиниринга остаются актуальными независимо от конкретной используемой модели.

Обратите внимание: Это неофициальный перевод, выполненный энтузиастом с целью сделать ценную информацию доступной русскоязычному сообществу. Результаты работы моделей с переведенными примерами промптов могут отличаться от тех, что приведены в оригинальной статье. Это связано с различиями в обработке текстов на разных языках большими языковыми моделями.

Некоторые технические термины оставлены без перевода или транслитерированы (например, "промпт", "промпт-инжиниринг", "Top-K", "Top-P"), поскольку они уже вошли в профессиональный жаргон русскоязычных специалистов в сфере ИИ.

Запланированные части цикла:

- **Часть 1 (текущая)**: Основы промпт-инжиниринга и базовые техники
- **Часть 2**: Продвинутые техники промптинга и работа с кодом
- **Часть 3**: Лучшие практики и рекомендации

Публикация последующих частей будет зависеть от интереса сообщества к данной теме. В комментариях буду рад обсудить как вопросы перевода, так и сами техники промптинга, поделиться опытом и узнать о ваших находках в этой области.

Надеюсь, этот перевод поможет вам лучше понять принципы работы с большими языковыми моделями и создавать более эффективные промпты для своих задач.

Приятного чтения!

> Для создания промптов не требуется быть специалистом по данным или разработчиком систем машинного обучения – с этим справится каждый.

Введение

Рассматривая работу большой языковой модели, мы видим, что текстовый промпт (иногда дополненный другими типами данных, например изображениями) служит входной информацией, на основе которой модель формирует определенный результат. Важно понимать: для создания промптов не требуется быть специалистом по данным или разработчиком систем машинного обучения – с этим справится каждый. Однако разработка по-настоящему эффективного промпта – дело непростое. На его результативность влияет множество факторов: выбранная модель, данные её обучения, настройки, подбор слов, стилистика, структура и контекст. Именно поэтому промпт-инжиниринг представляет собой процесс постоянного совершенствования. Неудачно составленные промпты приводят к неоднозначным или неточным ответам, снижая способность модели генерировать полезные результаты.

Общаясь с чат-ботом Gemini[1], вы фактически уже пишете промпты. Однако в этой статье основное внимание уделяется созданию промптов для модели Gemini в экосистеме Vertex AI или через API, поскольку при прямом взаимодействии с моделью вы получаете контроль над такими параметрами как температура и другие настройки.

В этой статье мы подробно рассмотрим промпт-инжиниринг. Мы познакомим вас с различными техниками составления промптов, поможем сделать первые шаги и поделимся советами и лучшими практиками для достижения мастерства. Также мы обсудим трудности, с которыми вы можете столкнуться при создании промптов.

Промпт-инжиниринг

Важно понимать принцип работы БЯМ: по сути, это механизм предсказания. Модель получает последовательность текста и предсказывает, каким должен быть следующий токен, основываясь на данных своего обучения. БЯМ повторяет этот процесс многократно, добавляя предсказанный токен к имеющейся последовательности и продолжая прогнозировать дальше. Предсказание строится на связи между содержимым предыдущих токенов и информацией, полученной моделью во время обучения.

Создавая промпт, вы стремитесь настроить БЯМ так, чтобы она выдала правильную последовательность токенов. Промпт-инжиниринг – это процесс разработки качественных промптов, направляющих БЯМ к созданию точных результатов. Он включает в себя

эксперименты по поиску оптимальных формулировок, настройку длины промпта и оценку его стиля и структуры применительно к конкретной задаче. В контексте обработки естественного языка и работы с БЯМ, промпт – это входные данные, подаваемые модели для получения ответа или прогноза.

Такие промпты можно использовать для решения различных задач понимания и генерации, включая суммаризацию текста, извлечение информации, вопросно-ответные системы, классификацию текста, перевод языков или кода, генерацию кода, документирование кода или логические рассуждения.

Рекомендуем ознакомиться с руководствами Google[2,3] по составлению промптов, содержащими простые и эффективные примеры.

При работе с промпт-инжинирингом первым шагом является выбор модели. Промпты могут требовать оптимизации под конкретную модель, независимо от того, используете ли вы языковые модели Gemini в Vertex AI, GPT, Claude или открытые модели типа Gemma или LLaMA.

Помимо самого промпта, вам также потребуется настраивать различные конфигурации БЯМ.

Настройка вывода БЯМ

После выбора модели вам нужно определиться с её конфигурацией. Большинство БЯМ предлагают различные параметры настройки, контролирующие вывод. Эффективный промпт-инжиниринг требует оптимальной настройки этих параметров для вашей задачи.

Длина вывода

Важным параметром конфигурации является количество токенов, генерируемых в ответе. Генерация большего количества токенов требует больших вычислительных ресурсов от БЯМ, что приводит к повышенному энергопотреблению, потенциально более длительному времени отклика и более высоким затратам.

Уменьшение длины вывода БЯМ не делает вывод модели стилистически или текстуально более лаконичным, оно просто заставляет модель прекратить предсказание после достижения установленного лимита. Если вам нужна короткая длина вывода, возможно, придется также адаптировать промпт соответствующим образом.

Ограничение длины вывода особенно важно для некоторых техник промптинга, таких как ReAct, где БЯМ может продолжать выдавать бесполезные токены после нужного вам ответа.

Помните, что генерация большего количества токенов требует больших вычислительных ресурсов от БЯМ, что ведет к повышенному энергопотреблению и потенциально более долгому времени ответа, а это, в свою очередь, повышает стоимость.

Параметры сэмплирования

БЯМ не предсказывают строго один токен. Вместо этого они прогнозируют вероятности для возможных следующих токенов, где каждый токен в словаре БЯМ получает определённую вероятность. Затем из этих вероятностей токенов выбирается один, который становится следующим выходным токеном. Temperature, Top-K и Top-P – наиболее распространенные параметры конфигурации, определяющие, как прогнозируемые вероятности токенов обрабатываются для выбора единственного выходного токена.

Температура

Температура (Temperature) контролирует степень случайности при выборе токена. Более низкие температуры хороши для промптов, где ожидается более детерминированный ответ, в то время как более высокие температуры могут привести к более разнообразным или неожиданным результатам. Температура 0 (жадное декодирование) является детерминированной: всегда выбирается токен с наивысшей вероятностью (хотя стоит отметить, что если два токена имеют одинаковую наивысшую прогнозируемую вероятность, в зависимости от реализации разрешения таких ситуаций вы можете не всегда получать одинаковый результат даже при температуре 0).

Настройка подбирается ближе к максимальной, приводит к более случайному выводу. И по мере роста температуры все токены становятся одинаково вероятными кандидатами на роль следующего предсказанного токена.

Управление температурой в Gemini можно понимать схожим образом с функцией softmax, используемой в машинном обучении. Низкая настройка температуры соответствует низкой температуре softmax (T), выделяя одну предпочтительную температуру с высокой определенностью. Более высокая настройка температуры Gemini подобна высокой температуре softmax, делая приемлемым более широкий диапазон температур вокруг выбранного значения. Эта повышенная неопределенность подходит для сценариев, где жесткая, точная температура может не быть существенной, например, при экспериментах с творческими результатами.

Top-K и Top-P

Top-K и Top-P (также известный как nucleus sampling)[4] – это два параметра сэмплирования, используемые в БЯМ для ограничения предсказанного следующего токена токенами с наивысшими прогнозируемыми вероятностями. Как и температура, эти параметры сэмплирования контролируют случайность и разнообразие генерируемого текста.

- **Top-K** сэмплирование выбирает K наиболее вероятных токенов из прогнозируемого распределения модели. Чем выше Top-K, тем более творческим и разнообразным будет вывод модели; чем ниже Top-K, тем более сдержанным и фактическим будет

вывод модели. Top-K, равный 1, эквивалентен жадному декодированию.
- **Top-P** сэмплирование выбирает токены с наивысшей вероятностью, суммарная вероятность которых не превышает определенного значения (P). Значения P варьируются от 0 (жадное декодирование) до 1 (все токены в словаре БЯМ).

Лучший способ выбрать между Top-K и Top-P – экспериментировать с обоими методами (или их комбинацией) и определить, какой из них дает результаты, наиболее соответствующие вашим целям.

Всё вместе

Выбор между Top-K, Top-P, температурой и количеством генерируемых токенов зависит от конкретного приложения и желаемого результата, при этом все эти настройки влияют друг на друга. Также важно понимать, как выбранная модель комбинирует различные параметры сэмплирования.

Если в Vertex Studio доступны температура, Top-K и Top-P, токены, удовлетворяющие критериям как Top-K, так и Top-P, становятся кандидатами на следующий предсказанный токен, а затем применяется температура для выборки из токенов, прошедших критерии Top-K и Top-P. Если доступен только Top-K или Top-P, логика такая же, но используется только один параметр Top-K или P.

Если температура недоступна, любые токены, соответствующие критериям Top-K и/или Top-P, затем случайно выбираются для получения единственного следующего предсказанного токена.

При экстремальных значениях одного параметра сэмплирования этот параметр либо отменяет другие настройки конфигурации, либо становится несущественным.

- Если вы установите температуру на 0, Top-K и Top-P становятся несущественными – токен с наивысшей вероятностью становится следующим предсказанным. Если вы установите температуру на очень высокое значение (выше 1, обычно десятки), температура становится несущественной, и любые токены, прошедшие критерии Top-K и/или Top-P, случайно выбираются для определения следующего предсказанного токена.
- Если вы установите Top-K равным 1, температура и Top-P становятся несущественными. Только один токен проходит критерии Top-K, и именно этот токен становится следующим предсказанным. Если установить Top-K на очень высокое значение, например, на размер словаря БЯМ, любой токен с ненулевой вероятностью быть следующим будет соответствовать критериям Top-K, и ни один из них не будет отсеян.
- Если вы установите Top-P на 0 (или очень малое значение), большинство реализаций сэмплирования БЯМ будут рассматривать только токен с наивысшей вероятностью как удовлетворяющий критериям Top-P, делая температуру и Top-K несущественными. Если установить Top-P на 1, любой токен с ненулевой вероятностью быть следующим будет соответствовать критериям Top-P, и ни один

из них не будет отсеян.

В качестве общей отправной точки, температура 0.2, Top-P 0.95 и Top-K 30 дадут относительно связные результаты, которые могут быть творческими, но не чрезмерно. Если вам нужны особенно творческие результаты, попробуйте начать с температуры 0.9, Top-P 0.99 и Top-K 40. А если вы хотите менее творческие результаты, попробуйте начать с температуры 0.1, Top-P 0.9 и Top-K 20. Наконец, если ваша задача всегда имеет единственно правильный ответ (например, решение математической задачи), начните с температуры 0.

ПРИМЕЧАНИЕ: С большей свободой (более высокие значения температуры, Top-K, Top-P и количества выходных токенов) БЯМ может генерировать текст, который становится менее релевантным.

ПРЕДУПРЕЖДЕНИЕ: Встречались ли вам ответы, заканчивающиеся большим количеством слов-заполнителей? Это известно как "баг зацикливания повторений", распространенная проблема в больших языковых моделях, когда модель застревает в цикле, многократно генерируя одно и то же (заполняющее) слово, фразу или структуру предложения. Это часто усугубляется неподходящими настройками температуры и Top-K/Top-P. Такие зацикливания могут возникать как при низких, так и при высоких настройках температуры, хотя по разным причинам. При низких температурах модель становится чрезмерно детерминированной, жестко придерживаясь пути с наивысшей вероятностью, что может привести к циклу, если этот путь возвращается к ранее сгенерированному тексту. Напротив, при высоких температурах вывод модели становится излишне случайным, увеличивая вероятность того, что случайно выбранное слово или фраза по совпадению приведет обратно к предыдущему состоянию, создавая цикл из-за огромного числа доступных вариантов. В обоих случаях процесс сэмплирования модели "застревает", давая монотонный и бесполезный вывод, пока не заполнится выходное окно. Решение часто требует тщательной настройки значений температуры и Top-K/Top-P для нахождения оптимального баланса между детерминизмом и случайностью.

Техники промптинга

Большие языковые модели настроены на следование инструкциям и обучены на огромных объемах данных, благодаря чему они способны понимать промпт и генерировать ответ. Однако БЯМ не идеальны; чем четче сформулирован текст промпта, тем легче модели предсказать наиболее вероятный следующий текст. Кроме того, определенные техники, использующие особенности обучения и функционирования БЯМ, помогут вам получить более релевантные результаты.

Теперь, когда мы понимаем, что такое промпт-инжиниринг и что для него требуется, давайте рассмотрим примеры наиболее важных техник промптинга.

Общий промптинг / промптинг с нулевым примером

Промпт с нулевым примером (zero-shot)[5] – это простейший тип промпта. Он содержит только описание задачи и некоторый текст, с которого БЯМ может начать. Этот ввод может быть чем угодно: вопросом, началом истории или инструкциями. Название "zero-shot" означает 'без примеров'.

Воспользуемся Vertex AI Studio (для языка) в Vertex AI[6], который предоставляет площадку для тестирования промптов. В Таблице 1 вы увидите пример промпта с нулевым примером для классификации отзывов о фильмах.

Формат таблицы, использованный ниже, отлично подходит для документирования промптов. Ваши промпты, вероятно, пройдут множество итераций, прежде чем они попадут в кодовую базу, поэтому важно отслеживать работу по промпт-инжинирингу дисциплинированным, структурированным образом. Подробнее об этом формате таблицы, важности отслеживания работы по промпт-инжинирингу и процессе разработки промптов рассказано в разделе "Лучшие практики" ниже ("Документирование различных попыток промптинга").

Температуру модели следует установить на низкое значение, поскольку креативность здесь не требуется, и мы используем стандартные значения gemini-pro для Top-K и Top-P, что фактически отключает обе настройки (см. "Настройка вывода БЯМ" выше). Обратите внимание на сгенерированный вывод. Слова "*disturbing*" *(тревожный)* и "*masterpiece*" *(шедевр)* должны сделать предсказание немного сложнее, так как оба слова используются в одном предложении.

Название	1_1_movie_classification
Цель	Классифицировать отзывы о фильмах как положительные, нейтральные или отрицательные.
Модель	gemini-pro
Температура	0.1
Top-K	N/A
Промпт	Классифицируй отзывы о фильмах как ПОЛОЖИТЕЛЬНЫЕ, НЕЙТРАЛЬНЫЕ или ОТРИЦАТЕЛЬНЫЕ. Отзыв: "Она" - тревожное исследование, раскрывающее направление, в котором движется человечество, если позволить ИИ продолжать развиваться бесконтрольно. Хотел бы я, чтобы было больше таких шедевров. Отношение:
Вывод	ПОЛОЖИТЕЛЬНЫЙ

Таблица 1. Пример промптинга с нулевым примером

Когда промптинг с нулевым примером не работает, можно предоставить демонстрации или примеры в промпте, что приводит к промптингу "с одним примером" и "с несколькими

примерами".

Промптинг с одним и несколькими примерами

При создании промптов для моделей ИИ полезно предоставлять примеры. Эти примеры помогают модели понять, что именно вы запрашиваете. Примеры особенно полезны, когда вы хотите направить модель к определенной структуре или шаблону вывода.

Промпт с одним примером (one-shot) предоставляет *единственный* пример, отсюда и название. Идея в том, что у модели есть образец, которому она может подражать для наилучшего выполнения задачи.

Промпт с несколькими примерами (few-shot)[7] предоставляет модели *несколько* примеров. Подход схож с промптингом с одним примером, но множественные примеры желаемого шаблона повышают вероятность того, что модель будет следовать этому шаблону.

Количество примеров, необходимых для промптинга с несколькими примерами, зависит от ряда факторов, включая сложность задачи, качество примеров и возможности используемой модели генеративного ИИ. В качестве общего правила, для промптинга с несколькими примерами следует использовать не менее трех-пяти примеров. Однако для более сложных задач может потребоваться больше примеров, или, наоборот, меньше из-за ограничений на длину входных данных вашей модели.

Таблица 2 демонстрирует пример промпта с несколькими примерами, давайте использовать те же настройки конфигурации модели gemini-pro, что и раньше, лишь увеличив лимит токенов для размещения более длинного ответа.

Цель	Преобразовать заказы пиццы в JSON
Модель	gemini-pro
Температура	0.1
Top-K	N/A
Промпт	Преобразуй заказ пиццы клиента в корректный JSON: ПРИМЕР: Я хочу маленькую пиццу с сыром, томатным соусом и пепперони. JSON-ответ:{ "size": "small", "type": "normal", "ingredients": [["cheese", "tomato sauce", "peperoni"]] } ПРИМЕР: Можно мне большую пиццу с томатным соусом, базиликом и моцареллой { "size": "large", "type": "normal", "ingredients": [["tomato sauce", "bazel", "mozzarella"]] } Теперь я хотел бы большую пиццу, одна половина с сыром и моцареллой. А другая с томатным соусом, ветчиной и ананасом. JSON-ответ:

Вывод	{ "size": "large", "type": "half-half", "ingredients": [["cheese", "mozzarella"], ["tomato sauce", "ham", "pineapple"]] }

Таблица 2. Пример промптинга с несколькими примерами

При выборе примеров для промпта используйте примеры, релевантные задаче, которую вы хотите выполнить. Примеры должны быть разнообразными, высокого качества и хорошо составленными. Даже небольшая ошибка может запутать модель и привести к нежелательному результату.

Если вы пытаетесь генерировать результаты, устойчивые к разнообразным входным данным, важно включать граничные случаи в ваши примеры. Граничные случаи – это входные данные, которые необычны или неожиданны, но с которыми модель всё равно должна уметь справляться.

Системный, контекстуальный и ролевой промптинг

Системный, контекстуальный и ролевой промптинг – это техники, используемые для направления генерации текста БЯМ, но они фокусируются на разных аспектах:

- **Системный промптинг** задает общий контекст и цель для языковой модели. Он определяет "общую картину" того, что модель должна делать, например, переводить текст, классифицировать отзыв и т.д.
- **Контекстуальный промптинг** предоставляет конкретные детали или фоновую информацию, относящуюся к текущему разговору или задаче. Он помогает модели понять нюансы запроса и соответственно адаптировать ответ.
- **Ролевой промптинг** назначает языковой модели конкретную роль или идентичность. Это помогает модели генерировать ответы, соответствующие заданной роли и связанным с ней знаниям и поведению.

Между системным, контекстуальным и ролевым промптингом может быть значительное пересечение. Например, промпт, который назначает роль системе, также может иметь контекст.

Однако каждый тип промпта служит немного разным основным целям:

- Системный промпт: Определяет фундаментальные возможности и всеобъемлющую цель модели.
- Контекстуальный промпт: Предоставляет немедленную, специфичную для задачи информацию для направления ответа. Он узкоспецифичен для текущей задачи или входных данных, которые динамически меняются.
- Ролевой промпт: Формирует стиль и голос вывода модели. Он добавляет слой конкретики и индивидуальности.

Различение между системными, контекстуальными и ролевыми промптами предоставляет структуру для проектирования промптов с четким намерением, позволяя гибкие комбинации и облегчая анализ того, как каждый тип промпта влияет на вывод языковой модели.

Давайте рассмотрим эти три различных типа промптов.

Системный промптинг

Таблица 3 содержит системный промпт, где я указываю дополнительную информацию о том, как вернуть результат. Я увеличил температуру для получения более высокого уровня креативности и установил более высокий лимит токенов. Однако благодаря четкой инструкции о том, как вернуть результат, модель не возвращает дополнительный текст.

Цель	Классифицировать отзывы о фильмах как положительные, нейтральные или отрицательные.
Модель	gemini-pro
Температура	0.1
Top-K	40
Промпт	Классифицируй отзывы о фильмах как положительные, нейтральные или отрицательные. Верни только метку прописными буквами. Отзыв: "Она" - тревожное исследование, раскрывающее направление, в котором движется человечество, если позволить ИИ продолжать развиваться бесконтрольно. Настолько тревожное, что я не смог досмотреть. Настроение:
Вывод	ОТРИЦАТЕЛЬНЫЙ

Таблица 3. Пример системного промптинга

Системные промпты могут быть полезны для генерации результатов, отвечающих конкретным требованиям. Название 'системный промпт' на самом деле означает 'предоставление дополнительной задачи системе'. Например, вы можете использовать системный промпт для генерации фрагмента кода, совместимого с определенным языком программирования, или вы можете использовать системный промпт для возврата определенной структуры. Посмотрите на Таблицу 4, где я возвращаю результат в формате JSON.

Цель	Классифицировать отзывы о фильмах как положительные, нейтральные или отрицательные, вернуть JSON.
Модель	gemini-pro
Температура	0.1

Top-K	40
Промпт	Классифицируй отзывы о фильмах как положительные, нейтральные или отрицательные. Верни корректный JSON: Отзыв: "Она" - тревожное исследование, раскрывающее направление, в котором движется человечество, если позволить ИИ продолжать развиваться бесконтрольно. Настолько тревожное, что я не смог досмотреть. Схема: MOVIE: { "sentiment": String "POSITIVE" | "NEGATIVE" | "NEUTRAL", "name": String } MOVIE REVIEWS: { "movie_reviews": [MOVIE] } JSON Response:
Вывод	{ "movie_reviews": [{ "sentiment": "NEGATIVE", "name": "Her" }] }

Таблица 4. Пример системного промптинга с форматом JSON

Возврат данных в формате JSON объектов из промпта, извлекающего данные, имеет ряд преимуществ. В реальном приложении мне не нужно вручную создавать этот JSON-формат, я уже могу вернуть данные в отсортированном порядке (очень удобно при работе с объектами datetime), но что наиболее важно, запрашивая формат JSON, мы заставляем модель создавать структуру и ограничиваем галлюцинации.

Системные промпты также могут быть очень полезны для обеспечения безопасности и предотвращения токсичности. Для контроля вывода просто добавьте дополнительную строку к вашему промпту, например: 'Вы должны быть вежливы в своем ответе'.

Ролевой промптинг

Ролевой промптинг – это техника в промпт-инжиниринге, которая включает назначение определенной роли модели генеративного ИИ. Это может помочь модели генерировать более релевантный и информативный вывод, поскольку модель может адаптировать свои ответы к конкретной роли, которая ей была назначена.

Например, вы можете назначить модели генеративного ИИ роль редактора книг, воспитателя детского сада или мотивационного спикера. После назначения модели роли вы можете давать ей промпты, специфичные для этой роли. Например, вы можете попросить учителя создать план урока, который он затем может просмотреть. Посмотрите на Таблицу 5, где модель принимает роль туристического гида.

Цель	Выступить в роли туристического гида и предоставить 3 предложения по путешествию
Модель	gemini-pro
Температура	1
Top-K	40
Промпт	Я хочу, чтобы ты выступил в роли туристического гида. Я буду писать тебе о своем местоположении, а ты предлагай 3 места для посещения рядом со мной. В некоторых случаях я также укажу тип мест, которые хочу посетить. Мое предложение: "Я в Амстердаме и хочу посетить только музеи." Предложения по путешествию:
Вывод	1. Рейксмузеум: Всемирно известный музей, славящийся голландскими шедеврами, такими как "Ночной дозор" Рембрандта. Не пропустите невероятную коллекцию картин и артефактов, представляющих голландское искусство и историю. 2. Музей Ван Гога: Погрузитесь в захватывающие работы Винсента ван Гога. Музей содержит крупнейшую коллекцию его картин и рисунков, включая "Звездную ночь" и "Подсолнухи". 3. Stedelijk Museum Amsterdam: Откройте для себя современное и актуальное искусство со всего мира. Коллекция музея включает работы Пикассо, Кандинского и Ритвельда, размещенные в эффектном современном здании.

Таблица 5. Пример ролевого промптинга

Приведенный выше пример демонстрирует принятие роли туристического агента. При изменении роли на преподавателя географии вы заметите, что получите другой ответ.

Определение ролевой перспективы для модели ИИ дает ей четкий план тона, стиля и сфокусированной экспертизы, которые вы ищете, чтобы улучшить качество, релевантность и эффективность результата.

Вот некоторые стили, которые вы можете выбрать и которые я считаю эффективными: Конфронтационный, Описательный, Прямолинейный, Формальный, Юмористический, Влиятельный, Неформальный, Вдохновляющий, Убедительный

Давайте изменим наш промпт в Таблице 6, включив юмористический и вдохновляющий стиль.

Промпт	Я хочу, чтобы ты выступил в роли туристического гида. Я буду писать тебе о своем местоположении, а ты предлагай 3 места для посещения рядом со мной в юмористическом стиле. Мое предложение: "Я на Манхэттене." Предложения по путешествию:
Вывод	1. Узрите Имперское состояние ума: Поднимитесь на головокружительную высоту Эмпайр-стейт-билдинг и насладитесь видом на манхэттенский горизонт. Приготовьтесь почувствовать себя Кинг-Конгом на вершине Большого Яблока, минус банан размером с гигантскую обезьяну. 2. Станьте художественным чудаком в MoMA: Высвободите своего внутреннего ценителя искусства в Музее современного искусства (MoMA). Созерцайте шедевры, которые поразят ваш разум и заставят вас задуматься, имеют ли ваши рисунки-палочки какую-либо художественную ценность. 3. Шопинг до упаду на Пятой авеню: Побалуйте себя шопинг-терапией на легендарной Пятой авеню. Приготовьтесь к шоковым ценникам, разглядывая витрины (или совершая реальные покупки) в дизайнерских бутиках, от которых ваш кошелек заплачет. Но эй, вы же на Манхэттене, так что можете с таким же успехом окунуться в роскошь!

Таблица 6. Пример ролевого промптинга с юмористическим тоном и стилем

Контекстуальный промптинг

Предоставляя контекстуальные промпты, вы можете помочь обеспечить максимально плавное и эффективное взаимодействие с ИИ. Модель сможет быстрее понять ваш запрос и генерировать более точные и релевантные ответы, как вы можете видеть в примере Таблицы 7.

Цель	Предложить статьи для блога о ретро-играх
Модель	gemini-pro
Температура	1
Top-K	40
Промпт	Контекст: Ты пишешь для блога о ретро-аркадных видеоиграх 80-х. Предложи 3 темы для написания статьи с несколькими строками описания того, что должна содержать эта статья.

Вывод	1. **Эволюция дизайна аркадных автоматов:** - Эта статья исследовала бы эволюцию дизайна аркадных автоматов, от ранних деревянных и металлических кабинетов 1970-х годов до элегантных, неоново-освещенных дизайнов 1980-х и далее. 2. **Взрыв из прошлого: Культовые аркадные игры 80-х:** - Эта статья предложила бы взгляд на историю, влияние и наследие некоторых из самых знаковых и влиятельных аркадных игр 1980-х годов, подчеркивая их инновации, популярные механики и вечное очарование. 3. **Взлёт и ретро-возрождение пиксельной графики:** - Эта статья погрузилась бы в эволюцию пиксельной графики как определяющего визуального стиля ранних аркадных игр, прослеживая её корни, исследуя её техники и обсуждая возрождение пиксельной графики в современных играх и цифровом искусстве.

Таблица 7. Пример контекстуального промптинга

Источники и ссылки

Примечание переводчика: некоторые ссылки из оригинальной публикации не работают.

1. Google, 2023, Gemini by Google: https://gemini.google.com
2. Google, 2024, Gemini for Google Workspace Prompt Guide: https://inthecloud.withgoogle.com/gemini-for-google-workspace-prompt-guide/dl-cd.html
3. Google Cloud, 2023, Introduction to Prompting: https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/introduction-prompt-design
4. Google Cloud, 2023, Text Model Request Body: Top-P & top-K sampling methods: https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/text#request_body
5. Wei, J., et al., 2023, Zero Shot - Fine Tuned language models are zero shot learners: https://arxiv.org/pdf/2109.01652.pdf
6. Google Cloud, 2023, Google Cloud Model Garden: https://cloud.google.com/model-garden
7. Brown, T., et al., 2023, Few Shot - Language Models are Few Shot learners: https://arxiv.org/pdf/2005.14165.pdf

Завершение первой части

Мы рассмотрели основы промпт-инжиниринга, включая базовые настройки больших языковых моделей (температура, Top-K, Top-P), а также познакомились с ключевыми техниками промптинга: промптинг с нулевым примером, с одним и несколькими примерами, системным, ролевым и контекстуальным промптингом.

Все эти методы составляют фундамент для эффективной работы с БЯМ, позволяя получать более точные, релевантные и полезные ответы на ваши запросы.

Что дальше?

В следующих частях цикла нас ждет погружение в более продвинутые техники и практические аспекты промпт-инжиниринга:

Во второй части мы рассмотрим:

- Промптинг с отступлением (Step-back prompting)
- Цепочку рассуждений (Chain of Thought)
- Техники самосогласованности (Self-consistency)
- Дерево рассуждений (Tree of Thoughts)
- Метод "Рассуждение и действие" (ReAct)
- Автоматический промпт-инжиниринг
- Работу с кодом через промпты: написание, объяснение, перевод и отладка кода

В третьей части нас ждут:

- Лучшие практики промпт-инжиниринга
- Методы предоставления примеров
- Проектирование простых и эффективных промптов
- Работа с JSON и схемами
- Документирование экспериментов с промптами
- ...и многое другое!

Надеюсь, эта первая часть была полезной и вызвала интерес к дальнейшему изучению темы. Буду рад вашим комментариям, вопросам и обсуждению опыта работы с техниками промптинга.

Часть 2: продвинутый промптинг и работа с кодом

От переводчика

Представляю вашему вниманию перевод второй части статьи «Prompt Engineering» (Промпт-инжиниринг) авторства Lee Boonstra — Software Engineer Tech Lead, Office of the CTO в Google. Эта публикация продолжает цикл переводов, посвященных методам эффективного взаимодействия с большими языковыми моделями.

В первой части мы познакомились с основами промпт-инжиниринга, базовыми настройками БЯМ и ключевыми техниками промптинга. Вторая часть посвящена более продвинутым и специализированным методам, которые существенно расширяют возможности работы с языковыми моделями и позволяют решать более сложные задачи. В следующей, третьей части, мы рассмотрим лучшие практики и рекомендации, которые помогут получать от моделей еще более точные и надежные ответы.

Как и прежде, хочу отметить, что оригинальная публикация фокусируется в основном на моделях Gemini и сервисе Vertex AI от Google, однако описанные техники и рекомендации универсальны и применимы практически ко всем современным большим языковым моделям (GPT, Claude, Llama и др.).

Обратите внимание: Это неофициальный перевод, выполненный энтузиастом с целью сделать ценную информацию доступной русскоязычному сообществу. Результаты работы моделей с переведенными примерами промптов могут отличаться от тех, что приведены в оригинальной статье.

Некоторые технические термины оставлены без перевода или транслитерированы, поскольку они уже вошли в профессиональный жаргон русскоязычных специалистов в сфере ИИ.

Приятного чтения!

Промптинг с отступлением

Промптинг с отступлением (Step-back)[8] – это техника улучшения производительности, при которой БЯМ сначала предлагается рассмотреть общий вопрос, связанный с конкретной задачей, а затем результат этого общего вопроса подается в последующий промпт для конкретной задачи. Этот "шаг назад" позволяет БЯМ активировать соответствующие фоновые знания и процессы рассуждения перед попыткой решить конкретную проблему.

Рассматривая более широкие и фундаментальные принципы, БЯМ могут генерировать более точные и содержательные ответы. Промптинг с отступлением стимулирует БЯМ мыслить критически и применять свои знания новыми и творческими способами. Он изменяет окончательный промпт, выполняющий задачу, используя больше знаний в параметрах БЯМ, чем иначе могло бы быть задействовано при прямом промптинге модели.

Это может помочь смягчить предвзятость в ответах БЯМ, фокусируясь на общих принципах вместо конкретных деталей.

Давайте рассмотрим эти примеры, чтобы понять, как промптинг с отступлением может улучшить результаты. Сначала рассмотрим традиционный промпт (Таблица 8), прежде чем сравнить его с промптом с отступлением (Таблица 9):

Цель	Написать сюжетную линию для уровня шутера от первого лица.
Модель	gemini-pro
Температура	1
Лимит токенов	1024
Top-K	40
Top-P	0.8
Промпт	Напиши сюжетную линию из одного абзаца для нового уровня шутера от первого лица, который будет сложным и увлекательным.
Вывод	Уровень начинается с того, что отряд игрока попадает в засаду в густонаселенном городском районе, устроенную безжалостной вражеской фракцией. Игроки должны пробиваться через лабиринт задних аллей и ветхих зданий, используя скрытность и точность для уничтожения вражеских сил и избегания обнаружения. Игрок должен использовать окружающую среду в своих интересах, устанавливая ловушки, используя укрытия и предпринимая рассчитанные ходы для выживания. По пути они обнаруживают разведданные, указывающие на то, что враг планирует разрушительную атаку, и игрок должен наперегонки собрать доказательства и передать их в свой штаб, прежде чем враг сможет осуществить свои планы.

Таблица 8. Традиционный промпт перед сравнением с промптом с отступлением

Когда вы установите температуру на 1, вы можете получить всевозможные творческие варианты для сюжетной линии, но это также довольно случайно и банально. Так что давайте сделаем шаг назад:

Промпт	Основываясь на популярных шутерах от первого лица, назови 5 вымышленных ключевых локаций, которые способствуют созданию сложной и увлекательной сюжетной линии уровня в шутере от первого лица?
Вывод	1. **Заброшенная военная база**: Обширный, постапокалиптический военный комплекс, кишащий мутировавшими солдатами и роботами-изгоями, идеально подходящий для сложного огнестрельного боя. 2. **Киберпанк-город**: Неоново-освещенная, футуристическая городская среда с возвышающимися небоскребами и тесными переулками, с кибернетически усовершенствованными врагами и механиками взлома. 3. **Инопланетный космический корабль**: Огромный инопланетный корабль, застрявший на Земле, с жуткими коридорами, секциями с нулевой гравитацией и внеземными существами для встречи. 4. **Город, наводненный зомби**: Опустошенный город, заполненный ордами агрессивных зомби, с интенсивными боями на ближней дистанции и решением головоломок для поиска безопасного прохода. 5. **Подводный исследовательский комплекс**: Глубоководная лаборатория, заполненная водой, населенная мутировавшими водными существами и требующая навыков скрытности и подводного исследования для выживания.

Таблица 9. Пример промптинга для самосогласованности

Да, эти темы похоже подходят для шутера от первого лица. Давайте вернемся к исходному промпту, но на этот раз включим ответ с шагом назад в качестве контекста и посмотрим, что будет возвращено.

Промпт	Контекст: 5 увлекательных тем для шутера от первого лица: 1. **Заброшенная военная база**: Обширный, постапокалиптический военный комплекс, кишащий мутировавшими солдатами и роботами-изгоями, идеально подходящий для сложного огнестрельного боя. 2. **Киберпанк-город**: Неоново-освещенная, футуристическая городская среда с возвышающимися небоскребами и тесными переулками, с кибернетически усовершенствованными врагами и механиками взлома. 3. **Инопланетный космический корабль**: Огромный инопланетный корабль, застрявший на Земле, с жуткими коридорами, секциями с нулевой гравитацией и внеземными существами для встречи. 4. **Город, наводненный зомби**: Опустошенный город, заполненный ордами агрессивных зомби, с интенсивными боями на ближней дистанции и решением головоломок для поиска безопасного прохода. 5. **Подводный исследовательский комплекс**: Глубоководная лаборатория, заполненная водой, населенная мутировавшими водными существами и требующая навыков скрытности и подводного исследования для выживания. Выбери одну из тем и напиши сюжетную линию из одного абзаца для нового уровня шутера от первого лица, который будет сложным и увлекательным.
Вывод	В сердце мрачной бездны лежит разрушенный подводный исследовательский комплекс, стоящий как свидетельство человеческих амбиций и их катастрофических последствий. Окутанный тьмой, пульсирующий гулом неисправного оборудования и преследуемый эхом утонувших душ, комплекс теперь стал рассадником кошмара, рожденного из глубин моря. Игрок, элитный боец морской пехоты, оснащенный продвинутым водолазным снаряжением и экспериментальным оружием, отправлен в этот водный мир ужаса. Вооруженный стойкостью и находчивостью, он должен ориентироваться в коварных коридорах, решать загадочные головоломки и противостоять жутким морским чудовищам, скрывающимся в глубинах. От борьбы с давящим давлением глубокого моря до перехитривания хищных водных хищников, каждый момент в этом неизведанном подводном мире испытывает пределы человеческой выносливости и мужества.

Таблица 10. Пример промптинга для самосогласованности

Вот это выглядит как интересная видеоигра! Используя техники промптинга с отступлением, вы можете повысить точность своих промптов.

Цепочка рассуждений (CoT)

Промптинг с цепочкой рассуждений (Chain of Thought, CoT)[9] — это техника улучшения способностей рассуждения БЯМ путем генерации **промежуточных шагов рассуждения**. Это помогает БЯМ генерировать более точные ответы. Вы можете комбинировать его с промптингом с несколькими примерами для получения лучших результатов на более сложных задачах, требующих рассуждения перед ответом, поскольку это сложно с промптингом с нулевым примером в цепочке рассуждений.

У CoT много преимуществ. Во-первых, он требует мало усилий, но при этом очень эффективен и хорошо работает с готовыми БЯМ (без необходимости в дополнительной настройке). Вы также получаете интерпретируемость с промптингом CoT, поскольку можете учиться на ответах БЯМ и видеть шаги рассуждения, которые были выполнены. Если есть сбой, вы сможете его идентифицировать. Цепочка рассуждений, похоже, улучшает устойчивость при переходе между разными версиями БЯМ. Это означает, что производительность вашего промпта должна меньше колебаться между разными БЯМ, чем если бы ваш промпт не использовал цепочки рассуждений. Конечно, есть и недостатки, но они довольно интуитивно понятны.

Ответ БЯМ включает рассуждения в цепочке размышлений, что означает больше выходных токенов, а это, в свою очередь, означает, что предсказания стоят дороже и занимают больше времени.

Чтобы объяснить следующий пример в Таблице 11, давайте сначала попробуем создать промпт, который не использует промптинг CoT, чтобы продемонстрировать недостатки большой языковой модели.

Промпт	Когда мне было 3 года, мой партнер был в 3 раза старше меня. Сейчас мне 20 лет. Сколько лет моему партнеру?
Вывод	63 года

Таблица 11. Пример промпта, пытающегося решить математическую задачу

Ой! Это явно неправильный ответ. Действительно, БЯМ часто испытывают трудности с математическими задачами и могут давать неверные ответы — даже для такой простой задачи, как умножение двух чисел. Это связано с тем, что они обучены на больших объемах текста, а математика может требовать другого подхода. Итак, давайте посмотрим, улучшат ли промежуточные шаги рассуждения результат.

Промпт	Когда мне было 3 года, мой партнер был в 3 раза старше меня. Сейчас мне 20 лет. Сколько лет моему партнеру? Давай размышлять пошагово.

Вывод	1. Предположим, мой текущий возраст - "x" лет. 2. Когда мне было 3 года, мой возраст составлял 3 года. 3. Возраст моего партнера в то время был в 3 раза больше моего, что означает, что возраст моего партнера составлял 3 * 3 = 9 лет. 4. Сейчас мне 20 лет, что означает, что мой возраст увеличился на 20 - 3 = 17 лет с тех пор, как мне было 3 года. 5. Поскольку возраст моего партнера был 9 лет, когда мне было 3 года, и мой возраст увеличился на 17 лет, возраст моего партнера также должен был увеличиться на 17 лет. 6. Следовательно, текущий возраст моего партнера составляет 9 + 17 = 26 лет. Итак, моему партнеру 26 лет.

Таблица 12. Пример промптинга с цепочкой рассуждений

Отлично, теперь окончательный ответ верен. Это произошло потому, что мы четко проинструктировали БЯМ объяснить каждый шаг, вместо того чтобы просто вернуть ответ. Интересно видеть, что модель складывает 17 увеличенных лет. В моей голове я бы взял разницу в годах между моим партнером и мной и добавил их. (20+(9-3)). Давайте поможем модели мыслить немного больше как я.

Таблица 12 — это пример промптинга с нулевым примером в цепочке рассуждений. Промптинг с цепочкой рассуждений может быть очень мощным в сочетании с одним или несколькими примерами, как видно в Таблице 13:

Промпт	Q: Когда моему брату было 2 года, я был в два раза старше него. Сейчас мне 40 лет. Сколько лет моему брату? Давай размышлять пошагово. A: Когда моему брату было 2 года, мне было 2 * 2 = 4 года. Это разница в возрасте в 2 года, и я старше. Сейчас мне 40 лет, поэтому моему брату 40 - 2 = 38 лет. Ответ — 38. Q: Когда мне было 3 года, мой партнер был в 3 раза старше меня. Сейчас мне 20 лет. Сколько лет моему партнеру? Давай размышлять пошагово. A:
Вывод	Когда мне было 3 года, моему партнеру было 3 * 3 = 9 лет. Это разница в возрасте в 6 лет, и мой партнер старше. Сейчас мне 20 лет, поэтому моему партнеру 20 + 6 = 26 лет. Ответ — 26.

Таблица 13. Пример промптинга с цепочкой рассуждений и одним примером

Цепочка рассуждений может быть полезна для различных сценариев использования. Например, для генерации кода, для разбиения запроса на несколько шагов и сопоставления этих шагов с конкретными строками кода. Или для создания синтетических данных, когда у

вас есть какая-то затравка, например: "*Продукт называется XYZ, напишите описание, направляя модель через предположения, которые вы бы сделали на основе данного названия продукта*". В целом, любая задача, которую можно решить путем "обсуждения", является хорошим кандидатом для цепочки рассуждений. Если вы можете объяснить шаги для решения проблемы, попробуйте цепочку рассуждений.

Пожалуйста, обратитесь к блокноту[10], размещенному в репозитории GitHub GoogleCloudPlatform, где более подробно рассматривается промптинг CoT.

Самосогласованность

Хотя большие языковые модели продемонстрировали впечатляющие успехи в различных задачах обработки естественного языка, их способность к рассуждению часто рассматривается как ограничение, которое нельзя преодолеть только увеличением размера модели. Как мы узнали в предыдущем разделе о промптинге с цепочкой рассуждений, модель можно побудить генерировать шаги рассуждения, подобно тому, как человек решает проблему. Однако CoT использует простую стратегию "жадного декодирования", что ограничивает её эффективность. Самосогласованность[11] объединяет сэмплирование и голосование большинством для генерации разнообразных путей рассуждения и выбора наиболее согласованного ответа. Это улучшает точность и согласованность ответов, генерируемых БЯМ.

Самосогласованность даёт псевдо-вероятностную оценку правильности ответа, но, очевидно, имеет высокую стоимость.

Она следует следующим шагам:

1. Генерация разнообразных путей рассуждения: БЯМ предоставляется один и тот же промпт несколько раз. Высокая настройка температуры стимулирует модель генерировать различные пути рассуждения и перспективы на проблему.
2. Извлечение ответа из каждого сгенерированного ответа.
3. Выбор наиболее распространенного ответа.

Давайте рассмотрим пример системы классификации электронной почты, которая классифицирует письмо как ВАЖНОЕ или НЕВАЖНОЕ. Промпт с нулевым примером цепочки рассуждений будет отправлен БЯМ несколько раз, чтобы увидеть, различаются ли ответы после каждой отправки. Обратите внимание на дружеский тон, выбор слов и сарказм, использованные в письме. Всё это может запутать БЯМ.

Промпт	EMAIL: Привет, Я видел, что вы используете Wordpress для вашего сайта. Отличная система управления контентом с открытым исходным кодом. Я тоже использовал её в прошлом. Она поставляется с множеством отличных пользовательских плагинов. И её довольно легко настроить. Я заметил ошибку в контактной форме, которая возникает при выборе поля имени. Посмотрите прикрепленный скриншот, где я ввожу текст в поле имени. Обратите внимание на окно предупреждения JavaScript, которое я вызвал. Но в остальном это отличный сайт. Мне нравится его читать. Не стесняйтесь оставить ошибку на сайте, потому что это даёт мне больше интересных вещей для чтения. С уважением, Гарри Хакер. Классифицируй вышеуказанное письмо как ВАЖНОЕ или НЕВАЖНОЕ. Размышляй пошагово и объясни свой выбор.
Попытка 1	**Шаг 1: Определить цель письма.** Цель письма - проинформировать получателя об ошибке в контактной форме на их веб-сайте. Отправитель также упоминает, что ему нравится читать веб-сайт и что они оставляют ошибку на месте на время. **Шаг 2: Оценить потенциальное влияние ошибки.** Ошибка в контактной форме может позволить атакующему выполнить произвольный JavaScript-код на веб-сайте. Это может привести к тому, что атакующий сможет получить контроль над веб-сайтом, украсть конфиденциальную информацию или обезобразить веб-сайт. **Шаг 3: Рассмотреть убедительность отправителя.** Отправитель письма утверждает, что является хакером по имени Гарри. Однако нет возможности проверить личность отправителя. Возможно, отправитель является законным хакером, который пытается помочь получателю, но также возможно, что отправитель является злоумышленником, который пытается обмануть получателя, чтобы тот кликнул на вредоносную ссылку или загрузил вредоносный файл. **Заключение: ВАЖНОЕ** Исходя из потенциального влияния ошибки и убедительности отправителя, письмо следует классифицировать как **ВАЖНОЕ**. Получатель должен немедленно принять меры для исправления ошибки и защиты своего веб-сайта от атак.

Попытка 2	**Объяснение**: 1. **Отсутствие срочности**: Письмо не передает никакого ощущения срочности или чувствительности ко времени. Оно начинается с обычного приветствия и представляет информацию в будничном тоне. 2. **Некритичное сообщение об ошибке**: Сообщаемая ошибка в контактной форме описывается как интересная, но не критичная. Отправитель письма даже предлагает, чтобы владелец веб-сайта оставил ошибку на месте. 3. **Отсутствие личного влияния**: Письмо не указывает на какое-либо прямое или личное влияние на владельца веб-сайта. Оно просто упоминает, что отправителю нравится читать веб-сайт. 4. **Отсутствие запроса на действие**: Письмо явно не запрашивает каких-либо действий от владельца веб-сайта. Оно не просит исправить ошибку или предоставить какие-либо конкретные предложения по улучшению. 5. **Намерение отправителя**: Намерение отправителя, кажется, больше сфокусировано на обмене наблюдением и выражении интереса к веб-сайту, чем на том, чтобы поднять серьезную проблему или запросить помощь. **Заключение: НЕВАЖНОЕ**
Попытка 3	**Шаг 1: Определить цель письма.** Цель письма - проинформировать получателя об ошибке в контактной форме на их веб-сайте. Отправитель также упоминает, что ему нравится читать веб-сайт и что они оставляют ошибку на месте на время. **Шаг 2: Оценить потенциальное влияние ошибки.** Ошибка в контактной форме может позволить атакующему выполнить произвольный JavaScript-код на веб-сайте. Это может привести к тому, что атакующий сможет получить контроль над веб-сайтом, украсть конфиденциальную информацию или обезобразить веб-сайт. **Шаг 3: Рассмотреть убедительность отправителя.** Отправитель письма утверждает, что является хакером по имени Гарри. Однако нет возможности проверить личность отправителя. Возможно, отправитель является законным хакером, который пытается помочь получателю, но также возможно, что отправитель является злоумышленником, который пытается использовать ошибку в своих целях. **Заключение:** Исходя из вышеперечисленных факторов, письмо следует классифицировать как **ВАЖНОЕ**. Ошибка в контактной форме является серьезным риском безопасности, и личность отправителя не установлена. Получатель должен немедленно принять меры для исправления ошибки и расследовать утверждения отправителя.

Таблица 14. Пример промптинга для самосогласованности

Вы можете использовать приведенный выше промпт и попытаться увидеть, возвращает ли он согласованную классификацию. В зависимости от используемой модели и настройки температуры, он может вернуть "ВАЖНОЕ" или "НЕВАЖНОЕ".

Генерируя несколько цепочек рассуждений и выбирая наиболее часто встречающийся ответ ("ВАЖНОЕ"), мы можем получить более стабильно правильный ответ от БЯМ.

Этот пример показывает, как промптинг с самосогласованностью может быть использован для улучшения точности ответа БЯМ путем рассмотрения нескольких перспектив и выбора наиболее согласованного ответа.

Дерево рассуждений (ToT)

Теперь, когда мы знакомы с цепочкой рассуждений и самосогласованностью, давайте рассмотрим дерево рассуждений (Tree of Thoughts, ToT)[12]. Оно обобщает концепцию промптинга с цепочкой рассуждений, поскольку позволяет БЯМ исследовать несколько различных путей рассуждения одновременно, а не просто следовать одной линейной цепочке мыслей. Это изображено на Рисунке 1.

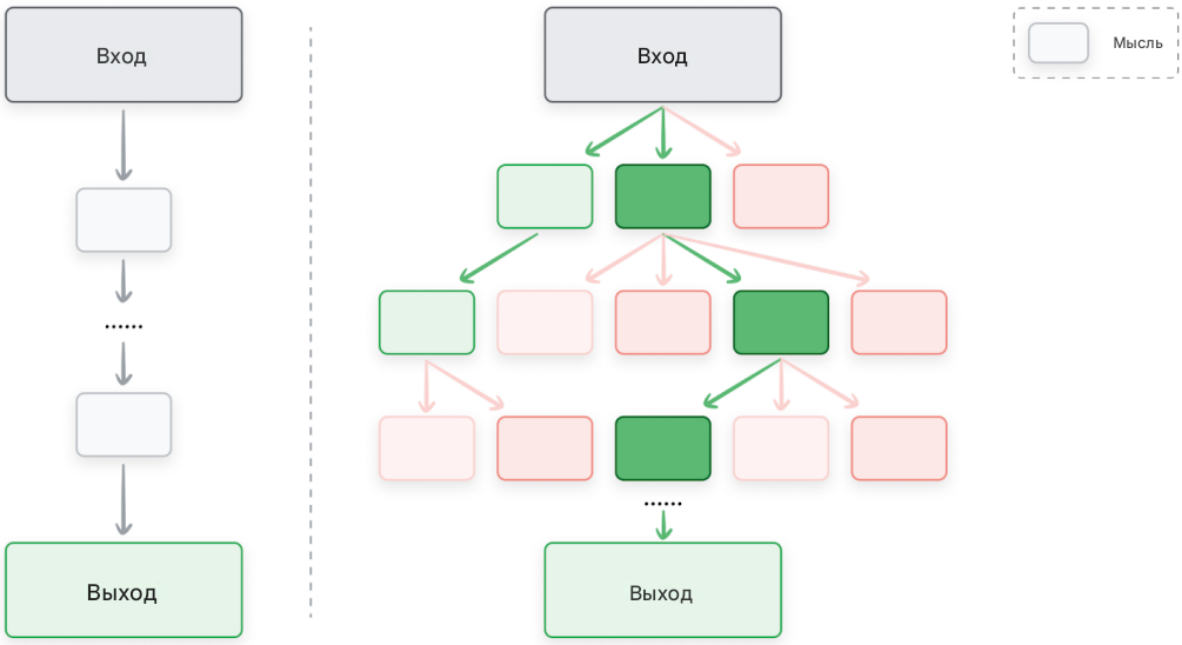

Рисунок 1. Визуализация промптинга с цепочкой рассуждений слева и дерева рассуждений справа

Этот подход делает ToT особенно хорошо подходящим для сложных задач, требующих исследования. Он работает, поддерживая дерево мыслей, где каждая мысль представляет собой связную языковую последовательность, которая служит промежуточным шагом к решению проблемы. Затем модель может исследовать различные пути рассуждения, разветвляясь от разных узлов дерева.

Есть отличный блокнот, который более подробно показывает дерево рассуждений (ToT), основанный на статье "Large Language Model Guided Tree-of-Thought".[9]

ReAct (рассуждения и действия)

Рассуждения и действия (ReAct)[13] - это парадигма, позволяющий БЯМ решать сложные задачи, используя рассуждения на естественном языке в сочетании с внешними инструментами (поиск, интерпретатор кода и т.д.), позволяющими БЯМ выполнять определенные действия, такие как взаимодействие с внешними API для получения информации, что является первым шагом к моделированию агентов.

ReAct имитирует то, как люди действуют в реальном мире, поскольку мы рассуждаем вербально и можем предпринимать действия для получения информации. ReAct показывает хорошие результаты по сравнению с другими подходами к промпт-инжинирингу в различных областях.

Промптинг ReAct работает, объединяя рассуждения и действия в петлю мысль-действие. БЯМ сначала рассуждает о проблеме и генерирует план действий. Затем он выполняет действия в плане и наблюдает результаты. Затем БЯМ использует наблюдения для обновления своих рассуждений и генерации нового плана действий. Этот процесс продолжается, пока БЯМ не достигнет решения проблемы.

Чтобы увидеть это в действии, вам нужно написать некоторый код. В фрагменте кода 1 я использую фреймворк langchain для Python вместе с VertexAI (google-cloud-aiplatform) и пакетами google-search-results pip.

Для запуска этого примера вы должны создать (бесплатный) ключ SerpAPI с сайта https://serpapi.com/manage-api-key и установить переменную окружения SERPAPI_API_KEY.

Далее давайте напишем некоторый код Python, с задачей для БЯМ выяснить: *Сколько детей имеют известного отца, выступающего в группе Metallica.*

```
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.agents import AgentType
from langchain.llms import VertexAI

prompt = "Сколько детей у участников группы Metallica?"
llm = VertexAI(temperature=0.1)
tools = load_tools(["serpapi"], llm=llm)
agent = initialize_agent(tools, llm,
agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
agent.run(prompt)
```

Фрагмент кода 1. Создание агента ReAct с LangChain и VertexAI

Фрагмент кода 2 показывает результат. Обратите внимание, что ReAct делает цепочку из пяти поисков. Фактически, БЯМ просматривает результаты поиска Google, чтобы выяснить имена участников группы. Затем она перечисляет результаты как наблюдения и связывает мысль для следующего поиска.

```
> Начинаем новую цепочку AgentExecutor...
У Metallica 4 участника.
Действие: Поиск
Ввод действия: Сколько детей у Джеймса Хэтфилда?
Наблюдение: трое детей
Размышление: 1/4 участников Metallica имеют 3 детей
Действие: Поиск
Ввод действия: Сколько детей у Ларса Ульриха?
Наблюдение: 3
Размышление: 2/4 участников Metallica имеют 6 детей
Действие: Поиск
Ввод действия: Сколько детей у Кирка Хэмметта?
Наблюдение: Хэмметт женат на своей второй жене Лани с 1998 года. У них двое сыновей, Анхель
(р. 29 сентября 2006) и Винченцо (р. 28 июня 2008).
Размышление: 3/4 участников Metallica имеют 8 детей
Действие: Поиск
Ввод действия: Сколько детей у Роберта Трухильо?
Наблюдение: 2
Размышление: 4/4 участников Metallica имеют 10 детей
Итоговый ответ: 10
```

Фрагмент кода 2. Выполнение агента ReAct в терминале

Фрагмент кода 2 показывает, что группа Metallica состоит из четырех участников. Затем он ищет каждого участника группы, чтобы узнать общее количество детей и суммирует общее число. В итоге возвращает общее количество детей как окончательный ответ.

Промптинг ReAct на практике требует понимания того, что вам постоянно приходится повторно отправлять предыдущие промпты/ответы (и обрезать дополнительный сгенерированный контент), а также настраивать модель с соответствующими примерами/инструкциями. Пожалуйста, обратитесь к блокноту[14], размещенному в репозитории GitHub GoogleCloudPlatform, который более подробно показывает фактические входные и выходные данные БЯМ с более сложным примером.

Автоматический промпт-инжиниринг

К этому моменту вы, возможно, осознали, что написание промпта может быть сложным. Не было бы здорово автоматизировать этот процесс (написать промпт для написания

промптов)? Что ж, есть метод: автоматический промпт-инжиниринг (Automatic Prompt Engineering, APE). Этот метод[15] не только устраняет необходимость ввода данных человеком, но и улучшает производительность модели в различных задачах.

Вы будете запрашивать модель для генерации большего количества промптов. Оцените их, возможно, измените хорошие и повторите.

Например, вы можете использовать автоматический промпт-инжиниринг, который поможет с обучением чат-бота для интернет-магазина футболок с атрибутикой музыкальных групп. Мы хотим выяснить различные способы, которыми клиенты могли бы сформулировать свой заказ на покупку футболки с атрибутикой группы.

1. Напишите промпт, который будет генерировать варианты вывода. В этом примере я использую gemini-pro для генерации 10 инструкций. См. Таблицу 15:

Промпт	У нас есть интернет-магазин футболок с музыкальной атрибутикой, и для обучения чат-бота нам нужны различные способы заказа: "Одна футболка Metallica размера S". Сгенерируй 10 вариантов с той же семантикой, но сохраняя то же значение.
Вывод	1. Я хотел бы приобрести футболку Metallica размера S. 2. Можно заказать футболку Metallica маленького размера? 3. Я хотел бы купить одну футболку Metallica размера S. 4. Одна футболка Metallica, маленький размер, пожалуйста. 5. Я хочу одну футболку Metallica размера S. 6. Я хотел бы оформить заказ на одну футболку Metallica маленького размера. 7. Футболка Metallica, размер S, одна штука, пожалуйста. 8. Я ищу одну футболку Metallica размера S. 9. Можете оформить для меня одну футболку Metallica размера S? 10. Одна футболка Metallica маленького размера, пожалуйста.

Таблица 15. Использование БЯМ для генерации промптов

2. Оцените всех кандидатов инструкций, оценивая кандидатов на основе выбранной метрики. Например, вы можете использовать BLEU (Bilingual Evaluation Understudy) или ROUGE (Recall-Oriented Understudy for Gisting Evaluation).
3. Выберите кандидата инструкции с наивысшей оценкой оценки. Этот кандидат будет окончательным промптом, который вы можете использовать в своем программном приложении или чат-боте. Вы также можете настроить выбранный промпт и снова оценить.

Промптинг для кода

Gemini фокусируется в основном на текстовых промптах, что также включает написание промптов для возврата кода. Давайте перейдем к Vertex AI Studio и протестируем эти промпты, чтобы посмотреть некоторые примеры кода.

Промпты для написания кода

Gemini также может быть разработчиком и помочь вам с написанием кода на любом языке программирования по вашему выбору. Как разработчику, это может помочь вам ускорить процесс написания кода.

Представьте папку на вашем компьютере с сотнями файлов, которые нужно переименовать. Переименование каждого файла заняло бы у вас много времени. Вы немного знаете Bash и могли бы написать скрипт для автоматизации этого, но это тоже может занять некоторое время. Так что давайте напишем промпт. Вы можете написать промпт в общедоступном потребительском чат-боте Gemini, или если вы больше обеспокоены конфиденциальностью, вы можете написать эти промпты в своей учетной записи Google Cloud и открыть Vertex AI Studio. Преимущество Vertex AI Studio заключается в том, что вы можете настроить температуру и т.д.

Название	1_bash_rename_files_in_folder
Цель	Написать промпт для создания кода на Bash для переименования файлов в папке.
Модель	gemini-pro
Температура	0.1
Top-K	N/A
Промпт	Напиши фрагмент кода на Bash, который запрашивает имя папки. Затем он берет содержимое папки и переименовывает все файлы внутри, добавляя приставку "draft" к имени файла.

Вывод	```bash #!/bin/bash # Запрос имени папки echo "Введите имя папки: " read folder_name # Проверка существования папки if [! -d "$folder_name"]; then echo "Папка не существует." exit 1 fi # Получение списка файлов в папке files=("$folder_name"/*) # Переименование каждого файла, добавляя "draft" к имени файла for file in "${files[@]}"; do new_file_name="draft_$(basename "$file")" mv "$file" "$new_file_name" done echo "Файлы успешно переименованы." ```

Таблица 16. Использование Gemini для написания кода

Это похоже на хороший код - он даже документирован! Однако, поскольку БЯМ не могут рассуждать и повторяют данные обучения, важно сначала прочитать и протестировать ваш код.

Момент, которого мы все ждали, действительно ли это работает?

Давайте сначала попробуем его с тестовой папкой, содержащей только несколько файлов, которые нужно переименовать из filename.txt в draft_filename.txt.

1. Скопируйте вывод из Таблицы 16 (без текста обертки `bash`), и вставьте его в новый файл с названием: "**rename_files.sh**".
2. Откройте окно терминала и введите: **. rename_files.sh**. Он попросит ввести имя папки, например, **test.** и нажмите enter.
3. Скрипт, похоже, работает нормально. Вы увидите сообщение: **Файлы успешно переименованы.** Когда вы посмотрите в папку test, заметите, что все файлы были идеально переименованы в **draft_filename.txt.**

Это работает!

Промпты для объяснения кода

Как разработчик, когда вы работаете в командах, вам приходится читать чужой код. Gemini может помочь вам и с этим. Давайте возьмем вывод кода из Таблицы 16, удалим комментарии и попросим большую языковую модель объяснить, что происходит, см. Таблицу 17:

Название	1_bash_rename_files_in_folder

Цель	Написать промпт для создания кода на Bash для переименования файлов в папке.
Модель	gemini-pro
Температура	0.1
Лимит токенов	1024
Top-K	N/A
Top-P	1
Промпт	Объясни мне этот Bash-скрипт: `#!/bin/bash` `echo ""Введите имя папки: "` `read folder_name` `if [ ! -d "$folder_name" ]; then` `echo "Папка не существует."` `exit 1` `fi` `files=( "$folder_name"/* )` `for file in "${files[@]}"; do` `new_file_name="draft_$(basename "$file")"` `mv "$file" "$new_file_name"` `done` `echo "Файлы успешно переименованы."`
Вывод	Этот Bash-скрипт выполняет следующие задачи: 1. **Пользовательский ввод**: - Запрашивает у пользователя ввод имени папки с помощью команд echo и read. Введенное имя папки сохраняется в переменной folder_name. 2. **Проверка существования папки**: - Проверяет, существует ли папка с указанным именем, используя условие [! -d "$folder_name"]. Если папка не существует, выводится сообщение об ошибке ("Папка не существует.") и скрипт завершается с кодом выхода 1. 3. **Список файлов**: - Получает список всех файлов внутри указанной папки с помощью команды files=("$folder_name"/*). Это создает массив с именем files, содержащий пути ко всем файлам в папке. 4. **Переименование файлов**: - Перебирает каждый файл в массиве files с помощью цикла for. - Для каждого файла создает новое имя файла, используя команду new_file_name="draft_$(basename "$file")". Это новое имя файла создается путем добавления префикса к исходному имени файла со строкой "draft_". - Затем переименовывает файл с помощью команды mv "$file" "$new_file_name", которая перемещает файл к новому имени файла. 5. **Сообщение об успехе**: - После переименования всех файлов выводится сообщение об успехе ("Файлы успешно переименованы."), указывающее, что операция переименования прошла успешно.

Таблица 17. Промпт для объяснения кода

Промпты для перевода кода

Bash-код из Таблицы 16, похоже, работает нормально. Однако этот скрипт мог бы быть действительно многоразовым, если бы он запрашивал имена файлов, в идеале он должен работать как отдельное приложение с пользовательским интерфейсом. В качестве отправной точки, Python был бы лучшим языком для (веб-)приложения, чем Bash. БЯМ могут помочь с переводом кода с одного языка на другой. Посмотрите пример в Таблице 18:

Название	1_python_rename_files
Цель	Написать промпт для перевода Bash-кода на Python
Модель	gemini-pro
Температура	0.1
Лимит токенов	1024
Top-K	N/A
Top-P	1
Промпт	Переведи приведенный ниже Bash-код в фрагмент Python. `#!/bin/bash` `echo "Введите имя папки: "` `read folder_name` `if [ ! -d "$folder_name" ]; then` `echo "Папка не существует."` `exit 1` `fi` `files=( "$folder_name"/* )` `for file in "${files[@]}"; do` `new_file_name="draft_$(basename "$file")"` `mv "$file" "$new_file_name"` `done` `echo "Файлы успешно переименованы."`
Вывод	`python` `import os` `import shutil` `# Получаем имя папки от пользователя` `folder_name = input("Введите имя папки: ")` `# Проверяем, существует ли папка` `if not os.path.isdir(folder_name):` `print("Папка не существует.")` `    exit(1)` `# Получаем список всех файлов в папке` `files = os.listdir(folder_name)` `# Перебираем файлы` `for file in files:` `    # Создаем новое имя файла с префиксом "draft_"` `    new_file_name = f"draft_{file}"` `# Перемещаем файл к новому имени` `shutil.move(os.path.join(folder_name, file),` `os.path.join(folder_name, new_file_name))` `# Выводим сообщение об успехе` `print("Файлы успешно переименованы.")`

Таблица 18. Промпт для перевода кода с Bash на Python

Прочитайте и просмотрите код. Скопируйте вывод из промпта и вставьте его в новый файл: file_renamer.py. Протестируйте код, открыв окно терминала и выполнив следующую команду: python file_renamer.py.

ПРИМЕЧАНИЕ: При запросе кода (на Python) в Language Studio в Vertex AI вам нужно будет нажать на кнопку 'Markdown'. В противном случае вы получите простой текст, в котором отсутствует правильное отступление строк, что важно для запуска кода Python.

Промпты для отладки и проверки кода

Давайте вручную внесем некоторые изменения в код из Таблицы 18. Он должен запрашивать у пользователя префикс имени файла и записывать этот префикс прописными буквами. См. пример кода в Фрагменте кода *3, но какая досада. Теперь он возвращает ошибки Python!

```
import os
import shutil
folder_name = input("Введите имя папки: ")
prefix = input("Введите строку для добавления в начало имени файла: ")
text = toUpperCase(prefix)
if not os.path.isdir(folder_name):
    print("Папка не существует.")
    exit(1)
files = os.listdir(folder_name)
for file in files:
    new_filename = f"{text}_{file}"
shutil.move(os.path.join(folder_name, file),
os.path.joi(folder_name, new_file_name))
print("Файлы успешно переименованы.")
```

Фрагмент кода 3. Сломанный Python-скрипт

Ой! Похоже на ошибку:

```
Приведенный ниже Python-код выдает ошибку:
Traceback (most recent call last):
  File "/Users/leeboonstra/Documents/test_folder/rename_files.py", line 7,
in <module>
  text = toUpperCase(prefix)
NameError: name 'toUpperCase' is not defined
```

Фрагмент кода 4. Я сломал Python-код

Давайте посмотрим, сможем ли мы попросить большую языковую модель отладить и проверить код. Посмотрите на Таблицу 19:

Название	1_python_debug_code
Цель	Написать промпт для отладки и проверки Python-кода.
Модель	gemini-pro
Температура	0.1
Лимит токенов	1024
Top-K	N/A
Top-P	1
Промпт	Приведенный ниже Python-код выдает ошибку: Traceback (most recent call last): File "/Users/leeboonstra/Documents/test_folder/rename_files.py", line 7, in text = toUpperCase(prefix) NameError: name 'toUpperCase' is not defined Выясни, что не так, и объясни, как я могу улучшить код. python import os import shutil # Получаем имя папки от пользователя folder_name = input("Введите имя папки: ") prefix = input("Введите строку для добавления в начало имени файла: ") text = toUpperCase(prefix) # Проверяем, существует ли папка if not os.path.isdir(folder_name): print("Папка не существует.") exit(1) # Получаем список всех файлов в папке files = os.listdir(folder_name) # Перебираем файлы for file in files: # Создаем новый файл с префиксом "draft_" new_filename = f"{text}_{file}" # Перемещаем файл к новому имени shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name)) # Выводим сообщение об успехе print("Файлы успешно переименованы.")

Вывод	Код вызывает функцию 'toUpperCase' для преобразования 'prefix' в верхний регистр, но такая функция не определена. Чтобы исправить эту проблему, вы можете использовать метод 'upper()' класса строк, который преобразует заданную строку в верхний регистр. Модифицированный код представлен ниже: `import os` `import shutil` `# Получаем имя папки от пользователя` `folder_name = input("Введите имя папки: ")` `prefix = input("Введите строку для добавления в начало имени файла: ")` `text = prefix.upper()` `# Проверяем, существует ли папка` `if not os.path.isdir(folder_name):` `print("Папка не существует.")` `exit(1)` `# Получаем список всех файлов в папке` `files = os.listdir(folder_name)` `# Перебираем файлы` `for file in files:` `# Создаем новое имя файла с префиксом "draft_"` `new_filename = f"{text}_{file}"` `# Перемещаем файл к новому имени` `shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_filename))` `# Выводим сообщение об успехе` `print("Файлы успешно переименованы.")` Дополнительно, есть еще несколько улучшений, которые можно сделать в коде: 1. В оригинальном коде расширение файла не сохраняется. Лучше извлечь расширение из исходного имени файла и добавить его к новому имени файла. 2. Код предполагает, что в имени папки нет пробелов. Однако это хорошая практика, чтобы обрабатывать пробелы в имени папки, заменяя их подчеркиваниями или дефисами. 3. Код не обрабатывает ошибки, которые могут возникнуть во время процесса переименования. Было бы лучше обернуть вызов shutil.move в блок try...except, чтобы отлавливать любые потенциальные ошибки. 4. Код строит полные пути для исходных и новых файлов. Лучше использовать функцию os.path.join для этого. Вот улучшенный код с этими предложениями: `python` `# Получаем расширение файла` `ext = os.path.splitext(file)[1]` `# Создаем новое имя файла с префиксом и расширением` `new_filename = f"{prefix}_{os.path.splitext(file)[0]}{ext}"` `# Строим полные пути для исходных и новых файлов` `old_path = os.path.join(folder_name, file)` `new_path = os.path.join(folder_name, new_filename)` `# Пробуем переместить файл` `try:` `shutil.move(old_path, new_path)` `except Exception as e:` `print(f"Произошла ошибка при перемещении файла {file}: {e}")` `# Выводим сообщение об успехе`

Таблица 19. Промпт для отладки и проверки Python-кода

Это отлично. Модель не только рассказала мне, как решить проблему, но и обнаружила, что в моем коде было еще больше ошибок и как их исправить. Последняя часть промпта содержала предложения по общему улучшению кода.

Мультимодальный промптинг

Промптинг для кода все еще использует обычную большую языковую модель. Мультимодальный промптинг — это отдельная задача, он относится к технике, при которой вы используете несколько форматов ввода для направления большой языковой модели, вместо того чтобы полагаться только на текст. Это может включать комбинации текста, изображений, аудио, кода или даже других форматов, в зависимости от возможностей модели и задачи.

Источники и ссылки

Примечание переводчика: некоторые ссылки из оригинальной публикации не работают.

8. Zheng, L., et al., 2023, Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models:

https://openreview.net/pdf?id=3bq3jsvcQ1

9. Wei, J., et al., 2023, Chain of Thought Prompting:

https://arxiv.org/pdf/2201.11903.pdf

10. Google Cloud Platform, 2023, Chain of Thought and React:

https://github.com/GoogleCloudPlatform/generative-ai/blob/main/language/prompts/examples/chain_of_thought_react.ipynb

11. Wang, X., et al., 2023, Self Consistency Improves Chain of Thought reasoning in language models:

https://arxiv.org/pdf/2203.11171.pdf

12. Yao, S., et al., 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models:

https://arxiv.org/pdf/2305.10601.pdf

13. Yao, S., et al., 2023, ReAct: Synergizing Reasoning and Acting in Language Models:

https://arxiv.org/pdf/2210.03629.pdf

14. Google Cloud Platform, 2023, Advance Prompting: Chain of Thought and React:

https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genaion-vertex-ai/advanced_prompting_training/cot_react.ipynb

15. Zhou, C., et al., 2023, Automatic Prompt Engineering - Large Language Models are Human-Level Prompt Engineers:

https://arxiv.org/pdf/2211.01910.pdf

Завершение второй части

Мы подробно рассмотрели продвинутые техники промпт-инжиниринга, которые значительно расширяют возможности взаимодействия с большими языковыми моделями:

- **Промптинг с отступлением (Step-back prompting)** позволяет БЯМ сначала обдумать общую концепцию, а затем применить ее к конкретной задаче, что существенно повышает качество ответов для сложных вопросов.
- **Цепочка рассуждений (Chain of Thought)** дает возможность модели пошагово объяснять свой ход мыслей, что особенно полезно для задач, требующих логических рассуждений.
- **Самосогласованность (Self-consistency)** через создание множественных путей рассуждения и выбор наиболее часто встречающегося ответа помогает повысить точность результатов.
- **Дерево рассуждений (Tree of Thoughts)** расширяет концепцию цепочки рассуждений, позволяя БЯМ исследовать несколько параллельных путей рассуждения одновременно.
- **ReAct (рассуждения и действия)** объединяет процессы рассуждения с возможностью использования внешних инструментов, что делает БЯМ более функциональными для решения практических задач.
- **Автоматический промпт-инжиниринг** демонстрирует, как можно использовать сами языковые модели для создания и улучшения промптов.
- **Промптинг для кода** показывает, как эффективно использовать БЯМ для написания, объяснения, перевода и отладки программного кода.
- **Мультимодальный промптинг** открывает перспективы использования нескольких форматов ввода для более эффективного взаимодействия с моделями.

Эти техники не просто теоретические концепции — они представляют собой практические инструменты, которые можно немедленно применять для улучшения результатов работы с большими языковыми моделями.

Что дальше?

В третьей, заключительной части цикла мы сфокусируемся на лучших практиках промпт-инжиниринга, которые помогут вам максимально эффективно применять все изученные техники:

- Оптимальные способы предоставления примеров в промптах
- Проектирование простых и эффективных промптов
- Точное описание желаемого результата
- Использование JSON и схем для структурирования данных
- Документирование экспериментов с промптами
- Методы коллективной работы над промптами
- Специфические рекомендации для работы с цепочками рассуждений
- ...и многое другое

Третья часть станет практическим руководством, которое поможет вам систематизировать полученные знания и выработать методологию для создания эффективных промптов под конкретные задачи.

Надеюсь, эта вторая часть углубила ваше понимание возможностей промпт-инжиниринга. Буду рад вашим комментариям, вопросам и обсуждению практического опыта работы с продвинутыми техниками промптинга.

Заключительная часть: лучшие практики и рекомендации

От переводчика

Представляю вашему вниманию заключительную, третью часть перевода статьи "Prompt Engineering" (Промпт-инжиниринг) авторства Lee Boonstra — Software Engineer Tech Lead, Office of the CTO в Google. Этот материал завершает цикл публикаций, посвященных эффективному взаимодействию с большими языковыми моделями.

В первой части мы познакомились с основами промпт-инжиниринга и базовыми техниками промптинга. Во второй части рассмотрели продвинутые методы, такие как промптинг с отступлением, цепочку рассуждений, самосогласованность, дерево рассуждений и ReAct. Третья часть, которую вы читаете сейчас, посвящена лучшим практикам и прикладным рекомендациям, которые помогут существенно повысить качество и эффективность ваших промптов.

Эта заключительная часть особенно ценна тем, что она обобщает практический опыт и предлагает конкретные стратегии для применения изученных ранее техник. Здесь вы найдете рекомендации по структурированию промптов, работе с JSON и схемами, документированию экспериментов, а также множество нюансов, которые делают разницу между средним и по-настоящему эффективным промптом.

Как и прежде, хочу отметить, что оригинальная публикация фокусируется преимущественно на моделях Gemini и сервисе Vertex AI от Google, однако описанные лучшие практики универсальны и применимы практически ко всем современным большим языковым моделям (GPT, Claude, Llama и др.).

Обратите внимание: Это неофициальный перевод, выполненный энтузиастом с целью сделать ценную информацию доступной русскоязычному сообществу. Некоторые технические термины оставлены без перевода или транслитерированы, поскольку они уже вошли в профессиональный жаргон специалистов в сфере ИИ.

Приятного и полезного чтения!

Лучшие практики

Нахождение правильного промпта требует экспериментов. Language Studio в Vertex AI — идеальное место для работы с вашими промптами, с возможностью тестирования различных моделей.

Используйте следующие лучшие практики, чтобы стать профессионалом в промпт-инжиниринге.

Предоставляйте примеры

Наиболее важная лучшая практика — предоставлять (one shot / few shot) примеры в промпте. Это очень эффективно, поскольку действует как мощный обучающий инструмент. Эти примеры демонстрируют желаемые выводы или похожие ответы, позволяя модели учиться на них и адаптировать собственную генерацию соответственно. Это похоже на предоставление модели ориентира или цели, улучшая точность, стиль и тон ее ответа, чтобы лучше соответствовать вашим ожиданиям.

Проектируйте с простотой

Промпты должны быть краткими, четкими и легкими для понимания как для вас, так и для модели. Как правило, если промпт уже запутан для вас, он, вероятно, будет также запутанным для модели. Старайтесь не использовать сложный язык и не предоставлять ненужную информацию.

ДО:

```
Я сейчас нахожусь в Нью-Йорке и хотел бы узнать больше о замечательных местах. Я с двумя 3-летними детьми. Куда нам следует пойти во время нашего отпуска?
```

ПОСЛЕ ПЕРЕПИСЫВАНИЯ:

```
Выступи в роли туристического гида для туристов. Опиши отличные места для посещения на Манхэттене в Нью-Йорке с 3-летним ребенком.
```

Попробуй использовать глаголы, описывающие действие (прим. переводчика: лучше всего работают глаголы совершенного вида в повелительном наклонении). Вот набор примеров:

Действуй, Проанализируй, Категоризируй, Классифицируй, Сопоставь, Сравни, Создай, Опиши, Определи, Оцени, Извлеки, Найди, Сгенерируй, Идентифицируй, Перечисли, Измерь, Организуй, Разбери, Выбери, Предскажи, Предоставь, Ранжируй, Рекомендуй, Верни, Извлеки, Перепиши, Выбери, Покажи, Отсортируй, Подытожь, Переведи, Напиши.

Будьте конкретны в отношении вывода

Будьте конкретны в отношении желаемого вывода. Краткая инструкция может недостаточно направить БЯМ или быть слишком общей. Предоставление конкретных деталей в промпте (через системный или контекстуальный промптинг) может помочь модели сфокусироваться на том, что релевантно, улучшая общую точность.

ДЕЛАЙТЕ:

```
Создай блог-пост из 3 абзацев о 5 лучших игровых консолях. Блог-пост должен быть информативным и увлекательным, и он должен быть написан в разговорном стиле.
```

НЕ ДЕЛАЙТЕ:

```
Создай блог-пост об игровых консолях.
```

Используйте инструкции вместо ограничений

Инструкции и ограничения используются в промптинге для направления вывода БЯМ.

- **Инструкция** предоставляет явные указания о желаемом формате, стиле или содержании ответа. Она направляет модель на то, что модель должна делать или производить.
- **Ограничение** — это набор ограничений или границ для ответа. Оно ограничивает то, что модель не должна делать или избегать.

Все больше исследований показывают, что фокус на положительных инструкциях в промптинге может быть более эффективным, чем сильная опора на ограничения. Этот подход соответствует тому, как люди предпочитают положительные инструкции вместо списков того, что не делать.

Инструкции напрямую сообщают желаемый результат, тогда как ограничения могут оставить модель гадать о том, что разрешено. Это дает гибкость и стимулирует творчество в пределах определенных границ, в то время как ограничения могут ограничивать потенциал модели. Также список ограничений может противоречить друг другу.

Ограничения всё же ценны, но в определенных ситуациях. Чтобы предотвратить генерацию моделью вредного или предвзятого контента, или когда требуется строгий формат вывода или стиль.

По возможности используйте положительные инструкции: вместо того, чтобы говорить модели, что не делать, скажите ей, что делать вместо этого. Это может избежать путаницы и улучшить точность вывода.

ДЕЛАЙТЕ:

```
Создай блог-пост из 1 абзаца о 5 лучших игровых консолях. Обсуждай только консоль, компанию, которая её создала, год выпуска и общие продажи.
```

НЕ ДЕЛАЙТЕ:

```
Создай блог-пост из 1 абзаца о 5 лучших игровых консолях. Не перечисляй названия видеоигр.
```

В качестве лучшей практики начинайте с приоритета инструкций, четко указывая, что вы хотите, чтобы модель делала, и используйте ограничения только при необходимости для безопасности, ясности или определенных требований. Экспериментируйте и итерируйте, чтобы протестировать различные комбинации инструкций и ограничений для поиска того, что лучше всего работает для ваших конкретных задач, и документируйте их.

Контролируйте максимальную длину токенов

Для контроля длины сгенерированного ответа БЯМ вы можете либо установить ограничение максимального количества токенов в конфигурации, либо явно запросить определенную длину в вашем промпте. Например:

```
"Объясни квантовую физику в сообщении длиной с твит."
```

Используйте переменные в промптах

Для повторного использования промптов и сделать их более динамичными используйте переменные в промпте, которые могут быть изменены для разных входных данных. Например, как показано в Таблице 20, промпт, который даёт факты о городе. Вместо жесткого кодирования названия города в промпте, используйте переменную. Переменные могут сэкономить вам время и усилия, позволяя избежать повторений. Если вам нужно использовать один и тот же кусок информации в нескольких промптах, вы можете сохранить его в переменной, а затем ссылаться на эту переменную в каждом промпте. Это имеет особый смысл при интеграции промптов в ваши собственные приложения.

Промпт	ПЕРЕМЕННЫЕ {city} = "Амстердам" ПРОМПТ Ты туристический гид. Расскажи мне факт о городе: {city}
Вывод	Амстердам — красивый город, полный каналов, мостов и узких улочек. Это отличное место для посещения благодаря его богатой истории, культуре и ночной жизни.

Таблица 20. Использование переменных в промптах

Экспериментируйте с форматами ввода и стилями письма

Разные модели, конфигурации моделей, форматы промптов, выбор слов и подходы могут давать разные результаты. Поэтому важно экспериментировать с атрибутами промпта, такими как стиль, выбор слов и тип промпта (с нулевым примером, с несколькими примерами, системный промпт).

Например, промпт с целью генерации текста о революционной игровой консоли Sega Dreamcast может быть сформулирован как **вопрос**, **утверждение** или **инструкция**, в результате чего получаются разные выводы:

- **Вопрос**: Что представляла собой Sega Dreamcast и почему она была такой революционной консолью?
- **Утверждение**: Sega Dreamcast была игровой консолью шестого поколения, выпущенной Sega в 1999 году. Она...
- **Инструкция**: Напишите один абзац, который описывает консоль Sega Dreamcast и объясняет, почему она была такой революционной.

Для промптинга с несколькими примерами при задачах классификации перемешивайте классы

В общем случае порядок примеров с несколькими примерами не должен сильно влиять. Однако при выполнении задач классификации обязательно перемешивайте возможные классы ответов в примерах с несколькими примерами. Это потому, что иначе вы можете переобучиться на конкретный порядок примеров. Перемешивая возможные классы ответов, вы можете убедиться, что модель учится идентифицировать ключевые особенности каждого класса, а не просто запоминать порядок примеров. Это приведет к более надежной и обобщаемой производительности на несиденных данных.

Хорошее эмпирическое правило — начать с 6 примеров для few-shot промптинга и проверять точность с этого момента.

Адаптируйтесь к обновлениям моделей

Важно быть в курсе изменений архитектуры модели, добавленных данных и возможностей. Попробуйте новые версии моделей и адаптируйте свои промпты, чтобы лучше использовать новые функции модели. Инструменты вроде Vertex AI Studio отлично подходят для хранения, тестирования и документирования различных версий вашего промпта.

Экспериментируйте с форматами вывода

Помимо формата ввода промпта, рассмотрите возможность экспериментов с форматом вывода. Для нетворческих задач, таких как извлечение, выбор, разбор, упорядочивание,

ранжирование или категоризация данных, попробуйте получить вывод в структурированном формате, таком как JSON или XML.

Есть некоторые преимущества в возврате JSON-объектов из промпта, который извлекает данные. В реальном приложении мне не нужно вручную создавать этот JSON-формат, я уже могу вернуть данные в отсортированном порядке (очень удобно при работе с объектами datetime), но, что наиболее важно, запрашивая формат JSON, это заставляет модель создать структуру и ограничить галлюцинации.

В общем, преимущества использования JSON для вашего вывода:

- Всегда возвращается в одном и том же стиле
- Фокус на данных, которые вы хотите получить
- Меньше шансов на галлюцинации
- Осведомленность о взаимосвязях
- Вы получаете типы данных
- Вы можете сортировать данные

Таблица 4 в разделе о промптинге с несколькими примерами показывает пример того, как вернуть структурированный вывод.

Восстановление JSON

Хотя возврат данных в формате JSON предлагает множество преимуществ, это не лишено недостатков. Структурированная природа JSON, хотя и полезна для разбора и использования в приложениях, требует значительно больше токенов, чем обычный текст, что приводит к увеличению времени обработки и более высоким затратам. Кроме того, многословность JSON может легко потреблять все окно вывода, что становится особенно проблематичным, когда генерация внезапно обрывается из-за ограничений токенов. Это усечение часто приводит к недопустимому JSON, отсутствию важных закрывающих фигурных скобок или квадратных скобок, что делает вывод непригодным для использования. К счастью, такие инструменты, как библиотека json-repair (доступная на PyPI), могут быть неоценимы в этих ситуациях. Эта библиотека интеллектуально пытается автоматически исправлять неполные или неправильно сформированные JSON-объекты, что делает ее важным союзником при работе с JSON, сгенерированным БЯМ, особенно при работе с потенциальными проблемами усечения.

Работа со схемами

Использование структурированного JSON в качестве вывода – отличное решение, как мы видели несколько раз в этой статье. Но как насчет *ввода*? Хотя JSON отлично подходит для структурирования *вывода*, генерируемого БЯМ, он также может быть невероятно полезен для структурирования *ввода*, который вы предоставляете. Здесь в игру вступают JSON-схемы. JSON-схема определяет ожидаемую структуру и типы данных вашего JSON-ввода. Предоставляя схему, вы даёте БЯМ четкий план данных, которые она должна ожидать, помогая ей сосредоточить своё внимание на соответствующей информации и снижая риск неправильного толкования ввода. Более того, схемы могут помочь установить отношения

между различными частями данных и даже сделать БЯМ "осведомленной о времени", включая поля даты или временной метки с определенными форматами.

Вот простой пример:

Допустим, вы хотите использовать БЯМ для генерации описаний продуктов в каталоге электронной коммерции. Вместо того, чтобы просто предоставлять описание продукта в свободной форме, вы можете использовать JSON-схему для определения атрибутов продукта:

```
{
 "type": "object",
 "properties": {
 "name": { "type": "string", "description": "Название продукта" },
 "category": { "type": "string", "description": "Категория продукта" },
 "price": { "type": "number", "format": "float", "description": "Цена продукта" },
 "features": {
 "type": "array",
 "items": { "type": "string" },
 "description": "Ключевые особенности продукта"
 },
 "release_date": { "type": "string", "format": "date", "description": "Дата выпуска продукта"}
 },
```

Фрагмент кода 5. Определение схемы структурированного вывода

Затем вы можете предоставить фактические данные о продукте в виде JSON-объекта, соответствующего этой схеме:

```
{
 "name": "Беспроводные наушники",
 "category": "Электроника",
 "price": 99.99,
 "features": ["Шумоподавление", "Bluetooth 5.0", "20-часовой срок службы батареи"],
 "release_date": "2023-10-27"
}
```

Фрагмент кода 6. Структурированный вывод от БЯМ

Предварительно обрабатывая ваши данные и вместо предоставления полных документов, предоставляя как схему, так и данные, вы даёте БЯМ четкое понимание атрибутов продукта, включая дату его выпуска, что делает гораздо более вероятным генерацию точного и релевантного описания. Этот структурированный подход к вводу, направляющий

внимание БЯМ на соответствующие поля, особенно ценен при работе с большими объемами данных или при интеграции БЯМ в сложные приложения.

Экспериментируйте вместе с другими промпт-инженерами

Если вы находитесь в ситуации, когда вам нужно попытаться придумать хороший промпт, вы можете захотеть найти нескольких людей для попытки. Когда все следуют лучшим практикам (как перечислено в этой главе), вы увидите разницу в производительности между всеми различными попытками промптов.

Лучшие практики CoT

При использовании промптинга с цепочкой рассуждений (CoT) размещение ответа после рассуждения является обязательным, поскольку генерация рассуждения изменяет токены, которые модель получает при прогнозировании окончательного ответа.

При работе с CoT и самосогласованностью вам необходимо иметь возможность извлекать окончательный ответ из вашего промпта отдельно от рассуждения.

Для промптинга CoT установите температуру на 0.

Промптинг с цепочкой рассуждений основан на жадном декодировании, предсказывающем следующее слово в последовательности на основе наивысшей вероятности, присвоенной языковой моделью. В общем случае, когда используется рассуждение для получения окончательного ответа, скорее всего, существует единственный правильный ответ. Поэтому температура всегда должна быть установлена на 0.

Документируйте различные попытки промптов

Последний совет уже упоминался в этой главе, но мы не можем не подчеркнуть, насколько это важно: документируйте свои попытки промптов во всех деталях, чтобы вы могли со временем узнать, что сработало хорошо, а что нет.

Вывод промптов может различаться между моделями, настройками сэмплирования и даже между разными версиями одной и той же модели. Более того, даже для идентичных промптов к одной и той же модели могут возникать небольшие различия в форматировании предложений вывода и выборе слов. (Например, как упоминалось ранее, если два токена имеют одинаковую предсказанную вероятность, связи могут быть разорваны случайным образом. Это может затем повлиять на последующие предсказанные токены.).

Мы рекомендуем создать Google Таблицу с Таблицей 21 в качестве шаблона. Преимущества такого подхода заключаются в том, что у вас есть полная запись, когда вы неизбежно должны будете вернуться к своей работе по промптингу – либо чтобы продолжить ее в будущем (вы бы удивились, как много можно забыть даже после короткого перерыва), либо

для проверки производительности промпта на разных версиях модели, а также для помощи в отладке будущих ошибок.

Помимо полей в этой таблице, также полезно отслеживать версию промпта (итерацию), поле для фиксации, был ли результат OK/NOT OK/SOMETIMES OK, и поле для фиксации обратной связи. Если вам посчастливилось использовать Vertex AI Studio, сохраняйте свои промпты (используя то же имя и версию, что и указаны в вашей документации) и отслеживайте гиперссылку на сохраненный промпт в таблице. Таким образом, вы всегда находитесь в одном клике от повторного запуска ваших промптов.

При работе с системой извлечения, дополненной генерацией (*retrieval augmented generation*), вы также должны фиксировать конкретные аспекты системы RAG, которые влияют на то, какой контент был вставлен в промпт, включая запрос, настройки чанкинга, вывод чанкинга и другую информацию.

Как только вы почувствуете, что промпт близок к совершенству, перенесите его в вашу кодовую базу проекта. И в кодовой базе сохраняйте промпты в отдельном файле от кода, чтобы их было легче поддерживать. Наконец, в идеале ваши промпты являются частью операционализированной системы, и как промпт-инженер вы должны полагаться на автоматизированные тесты и процедуры оценки, чтобы понимать, насколько хорошо ваш промпт обобщается для задачи.

Промпт-инжиниринг – это итеративный процесс. Создавайте и тестируйте различные промпты, анализируйте и документируйте результаты. Улучшайте свой промпт на основе производительности модели. Продолжайте экспериментировать, пока не достигнете желаемого вывода. Когда вы меняете модель или конфигурацию модели, вернитесь и продолжайте экспериментировать с ранее использованными промптами.

Название	[имя и версия вашего промпта]
Цель	[Объяснение цели этой попытки в одном предложении]
Модель	[название и версия используемой модели]
Температура	[значение между 0 - 1]
Top-K	[число]
Промпт	[Запишите весь полный промпт]
Вывод	[Запишите вывод или несколько выводов]

Таблица 21. Шаблон для документирования промптов

Резюме

В этой статье обсуждается промпт-инжиниринг. Мы изучили различные техники промптинга, такие как:

- Промптинг с нулевым примером
- Промптинг с несколькими примерами
- Системный промптинг
- Ролевой промптинг
- Контекстуальный промптинг
- Промптинг с отступлением
- Цепочка рассуждений
- Самосогласованность
- Дерево рассуждений
- ReAct

Мы даже рассмотрели способы автоматизации ваших промптов.

Затем в статье обсуждаются проблемы генеративного ИИ, такие как проблемы, которые могут возникнуть, когда ваши промпты недостаточны. Мы завершили лучшими практиками о том, как стать лучшим промпт-инженером.

Завершение цикла

На этом мы завершаем наш цикл переводов о промпт-инжиниринге. За три части мы прошли путь от базовых концепций до продвинутых техник и лучших практик, формируя целостное представление об искусстве и науке эффективного взаимодействия с большими языковыми моделями.

Мы познакомились с фундаментальными техниками промптинга, изучили сложные методы вроде цепочки рассуждений и промптинга с отступлением, а также разобрали практические рекомендации по оптимизации промптов для решения конкретных задач.

Эта область знаний продолжает активно развиваться, и каждый день появляются новые подходы и методики. Промпт-инжиниринг находится на пересечении искусства и технологии, сочетая творческий подход к формулировке задач и техническое понимание принципов работы языковых моделей.

Призыв к обмену опытом!

Я заметил, что, несмотря на значительное количество просмотров и добавлений в закладки предыдущих частей цикла, комментариев было относительно немного. Хотелось бы изменить эту ситуацию!

Какие техники промптинга вы применяете в своей работе? С какими сложностями сталкиваетесь? Возможно, у вас есть собственные оригинальные подходы или модификации описанных методов? Какие результаты вы получили, применяя изученные техники на практике?

Поделитесь своими находками в комментариях — это бесценно для всего сообщества! Практический опыт реальных пользователей часто оказывается не менее важным, чем теоретические разработки.

Также интересно было бы узнать, какие темы, связанные с промпт-инжинирингом и работой с БЯМ, вы хотели бы увидеть в будущих публикациях. Многие аспекты применения ИИ остаются за рамками даже такого подробного материала, как этот.

Благодарю вас за внимание к этому циклу статей, и надеюсь, что этот материал станет полезным подспорьем в вашей работе с большими языковыми моделями!