

Prompt Engineering на русском

Оригинал: [Prompt Engineering v7.pdf](#)

Автор: Lee Boonstra

Благодарности

Авторы контента

Michael Sherman
Yuan Cao
Erick Armbrust
Anant Nawalgaria
Antonio Gulli
Simone Cammel

Кураторы и редакторы

Antonio Gulli
Anant Nawalgaria
Grace Mollison

Технический писатель

Joey Haymaker

Дизайнер

Michael Lanning

Содержание

- [Введение](#)
- [Инженерия промптов](#)
- [Конфигурация вывода LLM](#)
 - [Длина вывода](#)
 - [Управление выборкой](#)
 - [Температура](#)
 - [Top-K и top-P](#)
 - [Объединяя всё вместе](#)
- [Техники промптинга](#)
 - [Общий промптинг / zero shot](#)
 - [One-shot и few-shot](#)
 - [Системный, контекстный и ролевой промптинг](#)
 - [Системный промптинг](#)
 - [Ролевой промптинг](#)
 - [Контекстный промптинг](#)
 - [Step-back промптинг](#)
 - [Chain of Thought \(CoT\)](#)
 - [Самосогласованность](#)
 - [Tree of Thoughts \(ToT\)](#)
 - [ReAct \(reason & act\)](#)
 - [Автоматическая инженерия промптов](#)
 - [Промптинг кода](#)
 - [Промпты для написания кода](#)
 - [Промпты для объяснения кода](#)
 - [Промпты для перевода кода](#)
 - [Промпты для отладки и проверки кода](#)
 - [А как насчет мультимодального промптинга?](#)
- [Лучшие практики](#)
 - [Предоставляйте примеры](#)
 - [Проектируйте с простотой](#)

- [Будьте конкретны в отношении вывода](#)
- [Используйте инструкции вместо ограничений](#)
- [Контролируйте максимальную длину токенов](#)
- [Используйте переменные в промптах](#)
- [Экспериментируйте с форматами ввода и стилями написания](#)
- [Для few-shot промптинга с задачами классификации, смешивайте классы](#)
- [Адаптируйтесь к обновлениям модели](#)
- [Экспериментируйте с форматами вывода](#)
- [Восстановление JSON](#)
- [Работа со схемами](#)
- [Экспериментируйте вместе с другими инженерами промптов](#)
- [Лучшие практики CoT](#)
- [Документируйте различные попытки промптов](#)
- [Резюме](#)
- [Примечания](#)

Вам не нужно быть специалистом по данным или инженером машинного обучения – каждый может написать промпт.

Введение

Рассматривая ввод и вывод большой языковой модели, текстовый промпт (иногда сопровождаемый другими модальностями, такими как изображения) является входными данными, которые модель использует для прогнозирования конкретного вывода. Вам не нужно быть специалистом по данным или инженером машинного обучения – каждый может написать промпт. Однако создание наиболее эффективного промпта может быть сложным. Многие аспекты вашего промпта влияют на его эффективность: используемая модель, обучающие данные модели, конфигурации модели, выбор слов, стиль и тон, структура и контекст – всё это имеет значение. Поэтому инженерия промптов – это итеративный процесс. Плохие промпты могут привести к неоднозначным, неточным ответам и могут затруднить способность модели предоставлять осмысленный вывод.

Когда вы общаетесь с [чат-ботом Gemini](#), вы, по сути, пишете промпты, однако этот документ фокусируется на написании промптов для модели Gemini в рамках Vertex AI или с использованием API, поскольку при прямом обращении к модели у вас будет доступ к конфигурации, такой как температура и т.д.

Этот документ подробно обсуждает инженерию промптов. Мы рассмотрим различные техники промптинга, чтобы помочь вам начать работу, и поделимся советами и лучшими практиками, чтобы стать экспертом в промптинге. Мы также обсудим некоторые проблемы, с которыми вы можете столкнуться при создании промптов.

Инженерия промптов

Помните как работает LLM; это движок предсказаний. Модель принимает последовательный текст в качестве входных данных, а затем предсказывает, каким должен быть следующий токен, основываясь на данных, на которых она была обучена. LLM настроена делать это снова и снова, добавляя ранее предсказанный токен в конец последовательного текста для предсказания следующего токена. Предсказание следующего токена основано на взаимосвязи между тем, что содержится в предыдущих токенах, и тем, что LLM видела во время обучения.

Когда вы пишете промпт, вы пытаетесь настроить LLM на предсказание правильной последовательности токенов. Инженерия промптов — это процесс разработки высококачественных промптов, которые направляют LLM на создание точных выходных данных. Этот процесс включает в себя экспериментирование для поиска лучшего промпта, оптимизацию длины промпта и оценку стиля написания и структуры промпта в отношении задачи. В контексте обработки естественного языка и LLM, промпт — это входные данные, предоставляемые модели для генерации ответа или предсказания.

Эти промпты могут использоваться для решения различных задач понимания и генерации, таких как суммаризация текста, извлечение информации, вопросы и ответы, классификация текста, перевод языка или кода, генерация кода, а также документирование кода или рассуждения.

Пожалуйста, не стесняйтесь обращаться к [руководствам Google по промптингу](#), с простыми и эффективными примерами промптов.

При инженерии промптов вы начнете с выбора модели. Промпты могут нуждаться в оптимизации для вашей конкретной модели, независимо от того, используете ли вы языковые модели Gemini в Vertex AI, GPT, Claude или модель с открытым исходным кодом, такую как Gemma или LLaMA.

Помимо промпта, вам также потребуется экспериментировать с различными конфигурациями LLM.

Конфигурация вывода LLM

После выбора модели вам нужно будет определить конфигурацию модели. Большинство LLM поставляются с различными параметрами конфигурации, которые управляют выводом LLM. Эффективная инженерия промптов требует оптимальной

настройки этих конфигураций для вашей задачи.

Длина вывода

Важным параметром конфигурации является количество токенов, которые нужно сгенерировать в ответе. Генерация большего количества токенов требует больше вычислений от LLM, что приводит к более высокому энергопотреблению, потенциально более медленному времени отклика и более высоким затратам.

Уменьшение длины вывода LLM не заставляет LLM становиться более стилистически или текстуально лаконичной в создаваемом выводе, оно просто заставляет LLM прекратить предсказывать больше токенов после достижения лимита. Если ваши потребности требуют короткой длины вывода, вам, возможно, также потребуется соответствующим образом настроить ваш промпт.

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

Имейте в виду, что генерация большего количества токенов требует больше вычислений от LLM, что приводит к более высокому энергопотреблению и потенциально более медленному времени отклика, что приводит к более высоким затратам.

Управление выборкой

LLM формально не предсказывают один токен. Скорее, LLM предсказывают вероятности того, каким может быть следующий токен, причем каждый токен в словаре LLM получает вероятность. Затем эти вероятности токенов выбираются для определения того, каким будет следующий произведенный токен. Температура, top-K и top-P являются наиболее распространенными параметрами конфигурации, которые определяют, как обрабатываются предсказанные вероятности токенов для выбора одного выходного токена.

Температура

Температура контролирует степень случайности при выборе токенов. Более низкие температуры хороши для промптов, которые ожидают более детерминированного ответа, 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 обратите внимание, что если два токена имеют одинаковую наивысшую предсказанную вероятность, в зависимости от того, как реализовано разрешение конфликтов, вы не всегда можете получить одинаковый вывод при температуре 0).

Температуры, близкие к максимальным, как правило, создают более случайный вывод. И по мере повышения температуры все токены становятся одинаково вероятными для следующего предсказанного токена.

Управление температурой Gemini можно понимать аналогично функции softmax, используемой в машинном обучении. Настройка низкой температуры отражает низкую температуру softmax (T), подчеркивая одну предпочтительную температуру с высокой определенностью. Более высокая настройка температуры Gemini подобна высокой температуре softmax, делая более приемлемым более широкий диапазон температур вокруг выбранной настройки. Эта повышенная неопределенность подходит для сценариев, где жесткая, точная температура может быть не столь важна, например, при экспериментах с творческими выводами.

Top-K и top-P

Top-K и top-P (также известные как [nucleus sampling](#)) — это два параметра выборки, используемые в LLM для ограничения предсказанного следующего токена токенами с наивысшими предсказанными вероятностями. Как и температура, эти параметры выборки контролируют случайность и разнообразие генерируемого текста.

- **Top-K** выборка выбирает K наиболее вероятных токенов из предсказанного 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** выборка выбирает верхние токены, кумулятивная вероятность которых не превышает определенное значение (P). Значения P варьируются от 0 (жадное декодирование) до 1 (все токены в словаре LLM).

Лучший способ выбрать между top-K и top-P — это экспериментировать с обоими методами (или с обоими вместе) и посмотреть, какой из них дает результаты, которые вы ищете.

Объединяя всё вместе

Выбор между top-K, top-P, температурой и количеством токенов для генерации зависит от конкретного приложения и желаемого результата, и все настройки влияют друг на друга. Также важно убедиться, что вы понимаете, как выбранная вами модель комбинирует различные параметры выборки.

Если доступны температура, top-K и top-P (как в Vertex Studio), токены, которые соответствуют критериям как top-K, так и top-P, являются кандидатами на следующий предсказанный токен, а затем применяется температура для выборки из токенов, прошедших критерии top-K и top-P. Если доступны только top-K или top-P, поведение такое же, но используется только одна настройка top-K или P.

Если температура недоступна, то из токенов, соответствующих критериям top-K и/или top-P, случайным образом выбирается один для создания следующего предсказанного токена.

При экстремальных настройках одного значения конфигурации выборки этот параметр выборки либо отменяет другие параметры конфигурации, либо становится неактуальным.

- Если вы установите температуру на 0, top-K и top-P становятся неактуальными — наиболее вероятный токен становится следующим предсказанным токеном. Если вы установите чрезвычайно высокую температуру (выше 1 — обычно в десятки), температура становится неактуальной, и из токенов, прошедших критерии top-K и/или top-P, случайным образом выбирается следующий предсказанный токен.
- Если вы установите top-K на 1, температура и top-P становятся неактуальными. Только один токен проходит критерии top-K, и этот токен является следующим предсказанным токеном. Если вы установите чрезвычайно высокий top-K, например, размером со словарь LLM, любой токен с ненулевой вероятностью быть следующим токеном будет соответствовать критериям top-K, и ни один не будет отсеян.
- Если вы установите top-P на 0 (или очень маленькое значение), большинство реализаций выборки LLM будут рассматривать только наиболее вероятный токен как соответствующий критериям top-P, делая температуру и top-K неактуальными. Если вы установите top-P на 1, любой токен с ненулевой вероятностью быть следующим токеном будет соответствовать критериям top-P, и ни один не будет отсеян.

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

ПРИМЕЧАНИЕ: С большей свободой (более высокая температура, top-K, top-P и выходные токены) LLM может генерировать текст, который менее релевантен.

ПРЕДУПРЕЖДЕНИЕ: Вы когда-нибудь видели ответ, заканчивающийся большим количеством слов-заполнителей? Это также известно как "ошибка цикла повторения", которая является распространенной проблемой в больших языковых моделях, когда модель застревает в цикле, многократно генерируя одно и то же (заполняющее) слово, фразу или структуру предложения, часто усугубляемую неподходящими настройками температуры и top-k/top-p. Это может происходить 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. И наоборот, при высоких температурах вывод модели становится чрезмерно случайным, увеличивая вероятность того, что случайно выбранное слово или фраза, по случайности, приведет обратно к предыдущему состоянию, создавая цикл из-за огромного количества доступных вариантов. В обоих случаях процесс выборки модели "застревает", что приводит к монотонному и бесполезному выводу, пока окно вывода не заполнится. Решение этой проблемы часто требует тщательной настройки значений температуры и top-k/top-p, чтобы найти оптимальный баланс между детерминизмом и случайностью.

Техники промптинга

LLM настроены на выполнение инструкций и обучены на больших объемах данных, чтобы они могли понимать промпт и генерировать ответ. Но LLM не идеальны; чем яснее ваш текст промпта, тем лучше для LLM предсказать следующий вероятный текст. Кроме того, конкретные техники, которые используют особенности обучения и работы LLM, помогут вам получить релевантные результаты от LLM.

Теперь, когда мы понимаем, что такое инженерия промптов и что для этого требуется, давайте рассмотрим некоторые примеры наиболее важных техник промптинга.

Общий промптинг / zero shot

Zero-shot промпт — это самый простой тип промпта. Он предоставляет только описание задачи и некоторый текст, с которого LLM может начать. Этот ввод может быть чем угодно: вопросом, началом истории или инструкциями. Название zero-shot означает "без примеров".

Давайте использовать Vertex AI Studio (для языка) в [Vertex AI](#), который предоставляет площадку для тестирования промптов. В Таблице 1 вы увидите пример zero-shot промпта для классификации отзывов о фильмах. Формат таблицы, используемый ниже, — отличный способ документирования промптов. Ваши промпты, вероятно, пройдут через множество итераций, прежде чем попадут в кодовую базу, поэтому важно отслеживать вашу работу по инженерии промптов дисциплинированным, структурированным способом. Подробнее об этом формате таблицы, важности отслеживания работы по инженерии промптов и процессе разработки промптов — в разделе "Лучшие практики" далее в этой главе ("Документируйте различные попытки промптов").

Температура модели должна быть установлена на низкое значение, поскольку творчество не требуется, и мы используем значения top-K и top-P по умолчанию для gemini-pro, которые эффективно отключают оба параметра (см. "Конфигурация вывода LLM" выше). Обратите внимание на сгенерированный вывод. Слова "тревожный" и "шедевр" должны сделать предсказание немного более сложным, поскольку 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	Лимит токенов	5
Тор-K	N/A	Тор-P	1
Промпт	Классифицируйте отзывы о фильмах как ПОЛОЖИТЕЛЬНЫЕ, НЕЙТРАЛЬНЫЕ или ОТРИЦАТЕЛЬНЫЕ. Отзыв: "Она" – это тревожное исследование, раскрывающее направление, в котором движется человечество, если ИИ будет позволено продолжать развиваться бесконтрольно. Я хотел бы, чтобы было больше фильмов, подобных этому шедевру. Настроение:		
Вывод	ПОЛОЖИТЕЛЬНЫЙ		

Таблица 1. Пример zero-shot промптинга

Когда zero-shot не работает, вы можете предоставить демонстрации или примеры в промпте, что приводит к "one-shot" и "few-shot" промптингу.

One-shot и few-shot

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

One-shot промпт предоставляет один пример, отсюда и название one-shot. Идея в том, что у модели есть пример, который она может имитировать для наилучшего выполнения задачи.

Few-shot промпт предоставляет модели несколько примеров. Этот подход показывает модели шаблон, которому она должна следовать. Идея аналогична one-shot, но несколько примеров желаемого шаблона увеличивают шансы, что модель будет следовать шаблону.

Количество примеров, необходимых для few-shot промптинга, зависит от нескольких факторов, включая сложность задачи, качество примеров и возможности используемой вами модели генеративного ИИ (gen AI). В качестве общего правила, вы должны использовать по крайней мере от трех до пяти примеров для few-shot промптинга. Однако вам может потребоваться использовать больше примеров для более сложных задач или меньше из-за ограничения длины ввода вашей модели.

Таблица 2 показывает пример few-shot промпта, давайте использовать те же настройки конфигурации модели gemini-pro, что и раньше, за исключением увеличения лимита токенов для размещения более длинного ответа.

Цель	Разбор заказов пиццы в JSON		
Модель	gemini-pro		
Температура	0.1	Лимит токенов	250
Тор-K	N/A	Тор-P	1
Промпт	<p>Разберите заказ пиццы клиента в действительный JSON:</p> <p>ПРИМЕР: Я хочу маленькую пиццу с сыром, томатным соусом и пепперони. JSON-ответ: ```\n{\n "size": "small",\n "type": "normal",\n "ingredients": [\n ["cheese", "tomato sauce", "peperoni"]\n]\n}\n```\n</p> <p>ПРИМЕР: Можно мне большую пиццу с томатным соусом, базиликом и моцареллой {\n "size": "large",\n "type": "normal",\n "ingredients": [\n ["tomato sauce", "baze", "mozzarella"]\n]\n}\n</p> <p>Теперь я хотел бы большую пиццу, с первой половиной сыр и моцарелла. А другая половина томатный соус, ветчина и ананас.</p> <p>JSON-ответ:</p>		
Вывод	<pre>```\n{\n "size": "large",\n "type": "half-half",\n "ingredients": [\n ["cheese", "mozzarella"],\n ["tomato sauce", "ham", "pineapple"]\n]\n}\n```\n</pre>		

Таблица 2. Пример few-shot промптинга

Когда вы выбираете примеры для своего промпта, используйте примеры, которые соответствуют задаче, которую вы хотите выполнить. Примеры должны быть разнообразными, высокого качества и хорошо написанными. Одна маленькая ошибка может запутать модель и приведет к нежелательному выводу.

Если вы пытаетесь генерировать вывод, который устойчив к различным входным данным, то важно включить граничные случаи в ваши примеры. Граничные случаи — это входные данные, которые необычны или неожиданны, но с которыми модель все равно должна уметь справляться.

Системный, контекстный и ролевой промптинг

Системный, контекстный и ролевой промптинг — это все техники, используемые для направления генерации текста LLM, но они фокусируются на разных аспектах:

- **Системный промптинг** устанавливает общий контекст и цель для языковой модели. Он определяет "общую картину" того, что модель должна делать, например, переводить язык, классифицировать отзыв и т.д.
- **Контекстный промптинг** предоставляет конкретные детали или фоновую информацию, относящуюся к текущему разговору или задаче. Он помогает модели понять нюансы того, о чем спрашивают, и соответствующим образом адаптировать ответ.
- **Ролевой промптинг** назначает языковой модели конкретный характер или идентичность для принятия. Это помогает модели генерировать ответы, которые соответствуют назначенной роли и связанным с ней знаниям и поведению.

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

Однако каждый тип промпта служит немного разной основной цели:

- Системный промпт: Определяет фундаментальные возможности модели и всеобъемлющую цель.
- Контекстный промпт: Предоставляет непосредственную, специфичную для задачи информацию для направления ответа. Он очень специфичен для текущей задачи или ввода, который является динамическим.
- Ролевой промпт: Формирует стиль и голос вывода модели. Он добавляет слой специфичности и индивидуальности.

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

Давайте углубимся в эти три разных вида промптов.

Системный промптинг

Таблица 3 содержит системный промпт, где я указываю дополнительную информацию о том, как возвращать вывод. Я увеличил температуру, чтобы получить более высокий уровень креативности, и указал более высокий лимит токенов. Однако из-за моей четкой инструкции о том, как возвращать вывод, модель не вернула дополнительный текст.

Цель	Классифицировать отзывы о фильмах как положительные, нейтральные или отрицательные.		
Модель	gemini-pro		
Температура	1	Лимит токенов	5
Тор-K	40	Тор-P	0.8
Промпт	Классифицируйте отзывы о фильмах как положительные, нейтральные или отрицательные. Только верните метку в верхнем регистре. Отзыв: "Она" — это тревожное исследование, раскрывающее направление, в котором движется человечество, если ИИ будет позволено продолжать развиваться бесконтрольно. Это настолько тревожно, что я не мог смотреть. Настроение:		
Вывод	ОТРИЦАТЕЛЬНЫЙ		

Таблица 3. Пример системного промптинга

Системные промпты могут быть полезны для генерации вывода, который соответствует конкретным требованиям. Название "системный промпт" на самом деле означает "предоставление дополнительной задачи системе". Например, вы можете использовать системный промпт для генерации фрагмента кода, который совместим с конкретным языком программирования, или вы можете использовать системный промпт для возврата определенной структуры. Взгляните на Таблицу 4, где я возвращаю вывод в формате JSON.

Цель	Классифицировать отзывы о фильмах как положительные, нейтральные или с JSON.	
Модель	gemini-pro	
Температура	1	Лимит токенов
Тор-K	40	Тор-P
Промпт	<div>Классифицируйте отзывы о фильмах как положительные, нейтральные или отрицательные. Верните действительный JSON:</div> <div>Отзыв: "Она" — это тревожное исследование, раскрывающее направление, в котором движется человечество, если ИИ будет позволено процветать бесконтрольно. Это настолько тревожно, что я не мог смотреть дальше."</div> <div>Схема:</div> <div>MOVIE:</div> <div>{</div> <div> "sentiment": String "POSITIVE" "NEGATIVE" "NEUTRAL",</div> <div> "name": String</div> <div>}</div> <div>MOVIE REVIEWS:</div> <div>{</div> <div> "movie_reviews": [MOVIE]</div> <div>}</div> <div>JSON-ответ:</div>	
Вывод	<div>{</div> <div> "movie_reviews": [<div>{</div><div> "sentiment": "NEGATIVE",</div><div> "name": "Her"</div><div>}</div><div>]</div><div>}</div></div>	

Таблица 4. Пример системного промптинга с форматом JSON

Есть некоторые преимущества в возврате объектов JSON из промпта, который извлекает данные. В реальном приложении мне не нужно вручную создавать этот формат JSON, 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 создавать структуру и ограничивать галлюцинации.

Системные промпты также могут быть действительно полезны для безопасности и токсичности. Чтобы контролировать вывод, просто добавьте дополнительную строку к вашему промпту, например: "Вы должны быть уважительны в своем ответе".

Ролевой промптинг

Ролевой промптинг — это техника в инженерии промптов, которая включает в себя назначение конкретной роли модели генеративного ИИ. Это может помочь модели генерировать более релевантный и информативный вывод, поскольку модель может формировать свои ответы в соответствии с конкретной ролью, которая ей была назначена.

Например, вы можете назначить модели генеративного ИИ роль редактора книги, воспитателя детского сада или мотивационного спикера. После того, как модели назначена роль, вы можете давать ей промпты, специфичные для этой роли. Например, вы можете попросить учителя создать план урока, который они затем могут просмотреть. Взгляните на Таблицу 5, где модель принимает роль туристического гида.

Цель	Действовать как туристический гид и предоставить 3 туристических предложения		
Модель	gemini-pro		
Температура	1	Лимит токенов	1024
Тор-K	40	Тор-P	0.8
Промпт	Я хочу, чтобы вы выступили в роли туристического гида. Я напишу вам о своем местоположении, и вы предложите 3 места для посещения рядом со мной. В некоторых случаях я также укажу тип мест, которые я хочу посетить. Мое предложение: "Я в Амстердаме и хочу посетить только музеи." Туристические предложения:		
Вывод	1. Рейксмузеум: Всемирно известный музей, известный голландскими шедеврами, такими как "Ночной дозор" Рембрандта. Не пропустите невероятную коллекцию картин и артефактов, демонстрирующих 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. Городской музей Амстердама (Стеделейк): Откройте для себя современное и современное искусство со всего мира. Коллекция музея включает работы Пикассо, Кандинского и Ритвельда,		

Приведенный выше пример показывает пример принятия роли туристического агента. Если бы вы изменили роль на учителя географии, вы бы заметили, что получите другой ответ.

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

Вот некоторые стили, из которых вы можете выбрать, которые я считаю эффективными:

Конфронтационный, Описательный, Прямой, Формальный, Юмористический, Влиятельный, Неформальный, Вдохновляющий, Убедительный

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

Промпт	Я хочу, чтобы вы выступили в роли туристического гида. Я напишу вам о своем местоположении, и вы предложите 3 места для посещения рядом со мной в юмористическом стиле. Мое предложение: "Я на Манхэттене." Туристические предложения:
Вывод	1. Созерцайте Имперское состояние ума: Поднимитесь на головокружительные высоты Эмпайр-стейт-билдинг и насладитесь великолепием манхэттенского горизонта. Приготовьтесь почувствовать себя Кинг-Конгом на вершине Большого Яблока, минус банан размером с гигантскую обезьяну. 2. Станьте артистично-фартистичным в МоМА: Раскройте своего внутреннего ценителя искусства в Музее современного искусства (МоМА). Взгляните на 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 розничной терапией на знаменитой Пятой авеню. Приготовьтесь к шоку от ценников, когда будете рассматривать витрины (или действительно делать покупки) в дизайнерских бутиках, которые заставят ваш кошелек плакать. Но эй, вы на Манхэттене, так

Контекстный промптинг

Предоставляя контекстные промпты, вы можете помочь обеспечить максимально плавное и эффективное взаимодействие с ИИ. Модель сможет быстрее понять ваш запрос и сможет 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	Лимит токенов	1024
Тор-K	40	Тор-P	0.8
Промпт	<p>Контекст: Вы пишете для блога о ретро-аркадных видеоиграх 80-х.</p> <p>Предложите 3 темы для написания статьи с несколькими строками описания того, что должна содержать эта статья.</p>		
Вывод	<p>1. Эволюция дизайна аркадных автоматов: Эта статья исследовала бы эволюцию дизайна аркадных автоматов, от ранних деревянных и металлических корпусов 1970-х годов до гладких, неоновых-освещенных дизайнов 1980-х годов и далее.</p> <p>2. Взрыв из прошлого: Культовые аркадные игры 80-х: Эта статья представила бы взгляд на историю, влияние и наследие некоторых из самых культовых и влиятельных аркадных игр 1980-х годов, подчеркивая их инновации, популярные механики и непреходящее очарование.</p> <p>3. Подъем и ретро-возрождение пиксельного искусства: Эта статья углубилась бы в эволюцию пиксельного искусства как определяющего визуального стиля</p>		

Step-back промптинг

Step-back промптинг — это техника для улучшения производительности путем побуждения LLM сначала рассмотреть общий вопрос, связанный с конкретной задачей, а затем использования ответа на этот общий вопрос в последующем промпте для конкретной задачи. Этот "шаг назад" позволяет LLM активировать соответствующие фоновые знания и процессы рассуждения перед попыткой решить конкретную проблему.

Рассматривая более широкие и основополагающие принципы, LLM могут генерировать более точные и содержательные ответы. Step-back промптинг побуждает LLM мыслить критически и применять свои знания новыми и творческими способами. Он изменяет финальный промпт, выполняющий задачу, используя больше знаний в параметрах LLM, чем было бы задействовано при прямом промптинге LLM.

Это может помочь смягчить предвзятость в ответах LLM, фокусируясь на общих принципах вместо конкретных деталей, step-back промптинг.

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

Цель	Написать сюжетную линию для уровня шутера от первого лица.		
Модель	gemini-pro		
Температура	1	Лимит токенов	1024
Тор-K	40	Тор-P	0.8
Промпт	Напишите сюжетную линию в один абзац для нового уровня шутера от первого лица, который будет сложным и увлекательным.		
Вывод	<p>Уровень начинается с того, что отряд игрока попадает в засаду в густонаселенном городском районе, устроенную безжалостной вражеской фракцией. Игроки должны пробиться через лабиринт задних переулков и обветшалых 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 делая расчетливые ходы, чтобы выжить. По пути они обнаруживают разведданные, указывающие на то, что враг планирует разрушительную атаку, и игрок должен наперегонки со временем собрать доказательства и</p>		

Когда вы устанавливаете температуру на 1, вы можете получить всевозможные творческие тексты для сюжетной линии, но они также довольно случайны и обобщены. Так что давайте сделаем шаг назад:

Промпт	Основываясь на популярных шутерах от первого лица, какие 5 вымышленных ключевых сеттингов способствуют созданию сложной и увлекательной сюжетной линии уровня в шутере от первого лица?
Вывод	<p>1. **Заброшенная военная база**: Обширный, постапокалиптический военный комплекс, кишачий мутировавшими солдатами и роботами-изгоями, идеально подходящий для сложных огневых боев.</p> <p>2. **Киберпанк-город**: Неоново-освещенная, футуристическая городская среда с возвышающимися небоскребами и густыми переулками, с кибернетически улучшенными врагами и механикой взлома.</p> <p>3. **Инопланетный космический корабль**: Огромный инопланетный корабль, застрявший на Земле, с жуткими коридорами, секциями с нулевой гравитацией и внеземными существами для встречи.</p> <p>4. **Город, зараженный зомби**: Опустошенный город, наводненный ордами агрессивных зомби, с интенсивными боями в ближнем бою и решением головоломок для поиска безопасного прохода.</p> <p>5. **Подводный исследовательский комплекс**: Глубоководная лаборатория, затопленная водой, наполненная мутировавшими водными существами и требующая навыков скрытности и подводного исследования для выживания.</p>

Таблица 9. Пример промптинга для самосогласованности

Да, эти темы кажутся хорошо подходящими для шутера от первого лица. Давайте вернемся к исходному промпту, но на этот раз включим ответ на шаг назад в качестве контекста и посмотрим, что он вернет.

Промпт	<p>Контекст: 5 увлекательных тем для шутера от первого лица:</p> <p>1. **Заброшенная военная база**: Обширный, постапокалиптический военный комплекс, кишачий мутировавшими солдатами и роботами-изгоями, идеально подходящий для сложных огневых боев.</p> <p>2. **Киберпанк-город**: Неоново-освещенная, футуристическая городская среда с возвышающимися небоскребами и густыми переулками, с кибернетически улучшенными врагами и механикой взлома.</p> <p>3. **Инопланетный космический корабль**: Огромный инопланетный корабль, застрявший на Земле, с жуткими коридорами, секциями с нулевой гравитацией и внеземными существами для встречи.</p> <p>4. **Город, зараженный зомби**: Опустошенный город, наводненный ордами агрессивных зомби, с интенсивными боями в ближнем бою и решением головоломок для поиска безопасного прохода.</p> <p>5. **Подводный исследовательский комплекс**: Глубоководная лаборатория, затопленная водой, наполненная мутировавшими водными существами и требующая навыков скрытности и подводного исследования для выживания.</p> <p>Возьмите одну из тем и напишите сюжетную линию в один абзац для нового уровня шутера от первого лица, который будет сложным и увлекательным.</p>
Вывод	<p>В сердце мрачной бездны лежит обветшалый подводный исследовательский комплекс, стоящий как свидетельство человеческих амбиций и их катастрофических последствий. Окутанный тьмой, пульсирующий гулом неисправной техники и преследуемый эхом утонувших душ, комплекс теперь стал рассадником кошмара, рожденного из глубин моря. Игрок, элитный морской пехотинец, оснащенный продвинутым снаряжением для дайвинга и экспериментальным оружием, получает задание отправиться в это водное царство ужаса. Вооруженный стойкостью и находчивостью, он должен ориентироваться в опасных коридорах, решать загадочные головоломки и противостоять ужасным морским чудовищам, которые скрываются в глубинах. От борьбы с сокрушительным давлением глубокого моря до перехитрения хитрых водных хищников, каждый момент в этом неизведанном подземном мире испытывает пределы человеческой выносливости и мужества.</p>

Таблица 10. Пример промптинга для самосогласованности

Это выглядит как интересная видеоигра! Используя техники step-back промптинга, вы можете повысить точность ваших промптов.

Цепочка размышлений (CoT)

Цепочка размышлений (CoT) — это техника промптинга для улучшения способностей рассуждения LLM путем генерации **промежуточных шагов рассуждения**. Это помогает LLM генерировать более точные ответы. Вы можете комбинировать ее с few-shot промптингом для получения лучших результатов на более сложных задачах, требующих рассуждения перед ответом, поскольку это является проблемой при zero-shot цепочке размышлений.

CoT имеет множество преимуществ. Прежде всего, это требует мало усилий, при этом очень эффективно и хорошо работает с готовыми LLM (так что нет необходимости в дополнительной настройке). Вы также получаете интерпретируемость с CoT промптингом, поскольку можете учиться на ответах LLM и видеть шаги рассуждения, которым следовали. Если есть неисправность, вы сможете ее идентифицировать. Цепочка размышлений, по-видимому, улучшает устойчивость при переходе между различными версиями LLM. Это означает, что производительность вашего промпта должна меньше дрейфовать между различными LLM, чем если бы ваш промпт не использовал цепочки рассуждений. Конечно, есть и недостатки, но они в некотором роде интуитивно понятны.

Ответ LLM включает рассуждение цепочки размышлений, что означает больше выходных токенов, что означает, что предсказания стоят больше денег и занимают больше времени.

Чтобы объяснить следующий пример в Таблице 11, давайте сначала попробуем создать промпт, который не использует CoT промптинг, 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, который пытается решить математическую задачу

Ой. Это явно неправильный ответ. На самом деле, LLM часто испытывают трудности с математическими задачами и могут давать неправильные ответы – даже для такой простой задачи, как умножение двух чисел. Это потому, что они обучены на больших объемах текста, а математика может требовать другого подхода. Так что давайте посмотрим, улучшат ли промежуточные шаги рассуждения вывод.

Промпт	Когда мне было 3 года, мой партнер был в 3 раза старше меня. Сейчас мне 20 лет. Сколько лет моему партнеру? Давайте подумаем шаг за шагом.
Вывод	<div>1. Предположим, что мой текущий возраст составляет "x" лет.</div> <div>2. Когда мне было 3 года, мой возраст составлял 3 года.</div> <div>3. Возраст моего партнера в то время был в 3 раза больше моего возраста, что означает, что возраст моего партнера составлял $3 * 3 = 9$ лет.</div> <div>4. Сейчас мне 20 лет, что означает, что мой возраст увеличился на $20 - 3 = 17$ лет с тех пор, как мне было 3 года.</div> <div>5. Поскольку возраст моего партнера составлял 9 лет, когда мне было 3 года, и мой возраст увеличился на 17 лет, возраст моего партнера также должен был увеличиться на 17 лет.</div> <div>6. Следовательно, текущий возраст моего партнера составляет $9 + 17 = 26$ лет.</div> <div>Итак, моему партнеру 26 лет.</div>

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

Таблица 12 — это пример 'zero-shot' цепочки размышлений. Промптинг с цепочкой размышлений может быть очень мощным в сочетании с single-shot или few-shot, как вы можете видеть в Таблице 13:

Промпт	<div>В: Когда моему брату было 2 года, я был в два раза старше его. Сейчас мне 40 лет. Сколько лет моему брату? Давайте подумаем шаг за шагом.</div> <div>О: Когда моему брату было 2 года, мне было $2 * 2 = 4$ года. Это разница в возрасте 2 года, и я старше. Сейчас мне 40 лет, поэтому моему брату $40 - 2 = 38$ лет. Ответ: 38.</div> <div>В: Когда мне было 3 года, мой партнер был в 3 раза старше меня. Сейчас мне 20 лет. Сколько лет моему партнеру? Давайте подумаем шаг за шагом.</div> <div>О:</div>
Вывод	Когда мне было 3 года, моему партнеру было $3 * 3 = 9$ лет. Это разница в возрасте 6 лет, и мой партнер старше. Сейчас мне 20 лет, поэтому моему партнеру $20 + 6 = 26$ лет. Ответ: 26.

Цепочка размышлений может быть полезна для различных случаев использования. Подумайте о генерации кода, о разбиении запроса на несколько шагов и сопоставлении их с конкретными строками кода. Или для создания синтетических данных, когда у вас есть какое-то начало, например, "Продукт называется XYZ, напишите описание, направляя модель через предположения, которые вы бы сделали на основе данного названия продукта". В общем, любая задача, которую можно решить, "проговаривая", является хорошим кандидатом для цепочки размышлений. Если вы можете объяснить шаги для решения проблемы, попробуйте цепочку размышлений.

Пожалуйста, обратитесь к [блокноту](#), размещенному в репозитории Github GoogleCloudPlatform, который более подробно рассмотрит CoT промптинг:

В разделе лучших практик этой главы мы узнаем некоторые лучшие практики, специфичные для промптинга с цепочкой размышлений.

Самосогласованность

Хотя большие языковые модели показали впечатляющие успехи в различных задачах NLP, 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 размышлений, модель можно побудить генерировать шаги рассуждения, как человек, решающий проблему. Однако CoT использует простую стратегию "жадного декодирования", ограничивающую его эффективность. **Самосогласованность** объединяет выборку и мажоритарное голосование для генерации разнообразных путей рассуждения и выбора наиболее согласованного ответа. Это улучшает точность и согласованность ответов, генерируемых LLM.

Самосогласованность дает псевдо-вероятностную вероятность правильности ответа, но, очевидно, имеет высокие затраты.

Она следует следующим шагам:

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

Давайте рассмотрим пример системы классификации электронной почты, которая классифицирует электронное письмо как ВАЖНОЕ или НЕ ВАЖНОЕ. Zero-shot промпт с цепочкой размышлений будет отправлен в LLM несколько раз, чтобы увидеть, отличаются ли ответы после каждой отправки. Обратите внимание на дружелюбный тон, выбор слов и сарказм, которые были использованы в письме. Все это может обмануть LLM.

Промпт	EMAIL: ```Привет, Я заметил, что вы используете Wordpress для своего веб-сайта. Отличная система управления контентом с открытым исходным кодом. Я тоже использовал ее в прошлом. Она поставляется с множеством отличных пользовательских плагинов. И ее 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 вызв4л. Но в остальном это отличный веб-сайт. Мне нравится его читать. Не стесняйтесь оставить ошибку на веб-сайте, потому что это дает мне больше интересных вещей для чтения. С уважением, Хакер Гарри. ```Классифицируйте вышеуказанное электронное письмо как ВАЖНОЕ или НЕ ВАЖНОЕ. Давайте подумаем шаг за шагом и объясним почему.
Вывод Попытка 1	<p>**Шаг 1: Определить цель электронного письма.**</p> <p>Цель электронного письма — информировать получателя об ошибке в контактной форме на его веб-сайте. Отправитель также упоминает, что ему нравится читать веб-сайт и что он оставляет ошибку на месте на данный момент.</p> <p>**Шаг 2: Оценить потенциальное влияние ошибки.**</p> <p>Ошибка в контактной форме может позволить злоумышленнику выполнять произвольный JavaScript-код на веб-сайте. Это может привести к тому, что злоумышленник сможет взять под контроль веб-сайт, украсть конфиденциальную информацию или испортить веб-сайт.</p> <p>**Шаг 3: Рассмотреть достоверность отправителя.**</p> <p>Отправитель электронного письма утверждает, что он хакер по имени Гарри. Однако нет способа проверить личность отправителя. Возможно, отправитель является законным хакером, который пытается помочь получателю, но также возможно, что отправитель является злоумышленником, который пытается обмануть получателя, заставив его нажать на вредоносную ссылку или загрузить вредоносный файл.</p> <p>**Заключение: ВАЖНОЕ**</p> <p>Основываясь на потенциальном влиянии ошибки и достоверности отправителя, электронное письмо следует классифицировать как **ВАЖНОЕ**. Получатель должен немедленно принять меры для исправления ошибки и защиты своего веб-сайта от атаки.</p>
Вывод Попытка 2	<p>**Объяснение:**</p> <p>1. **Отсутствие срочности:** Электронное письмо не передает никакого чувства срочности или временной чувствительности. Оно начинается с непринужденного приветствия и представляет информацию в тоне констатации факта.</p> <p>2. **Некритичное сообщение об ошибке:** Ошибка, о которой сообщается в контактной форме, описывается как интересная, но не критичная. Отправитель электронного письма даже предлагает владельцу веб-сайта оставить ошибку на месте.</p> <p>3. **Отсутствие личного влияния:** Электронное письмо не указывает на какое-либо прямое или личное влияние на владельца веб-сайта. Оно просто упоминает, что отправителю нравится читать веб-сайт.</p> <p>4. **Отсутствие запроса на действие:** Электронное письмо явно не запрашивает никаких действий от владельца веб-сайта. Оно не просит исправить ошибку или не предоставляет каких-либо конкретных предложений по улучшению.</p> <p>5. **Намерение отправителя:** Намерение отправителя, кажется, больше сосредоточено на обмене наблюдениями и выражении интереса к веб-сайту, чем на поднятии серьезной проблемы или запросе помощи.</p> <p>**Заключение: НЕ ВАЖНОЕ**</p>
	<p>**Шаг 1: Определить цель электронного письма.**</p>

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

Генерируя множество цепочек размышлений и выбирая наиболее часто встречающийся ответ ("ВАЖНОЕ"), мы можем получить более стабильно правильный ответ от LLM.

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

Дерево мыслей (ToT)

Теперь, когда мы знакомы с цепочкой размышлений и промптингом с самосогласованностью, давайте рассмотрим [Дерево мыслей \(ToT\)](#). Оно обобщает концепцию CoT промптинга, поскольку позволяет LLM исследовать несколько различных путей рассуждения одновременно, а не просто следовать одной линейной цепочке мыслей. Это изображено на Рисунке 1.

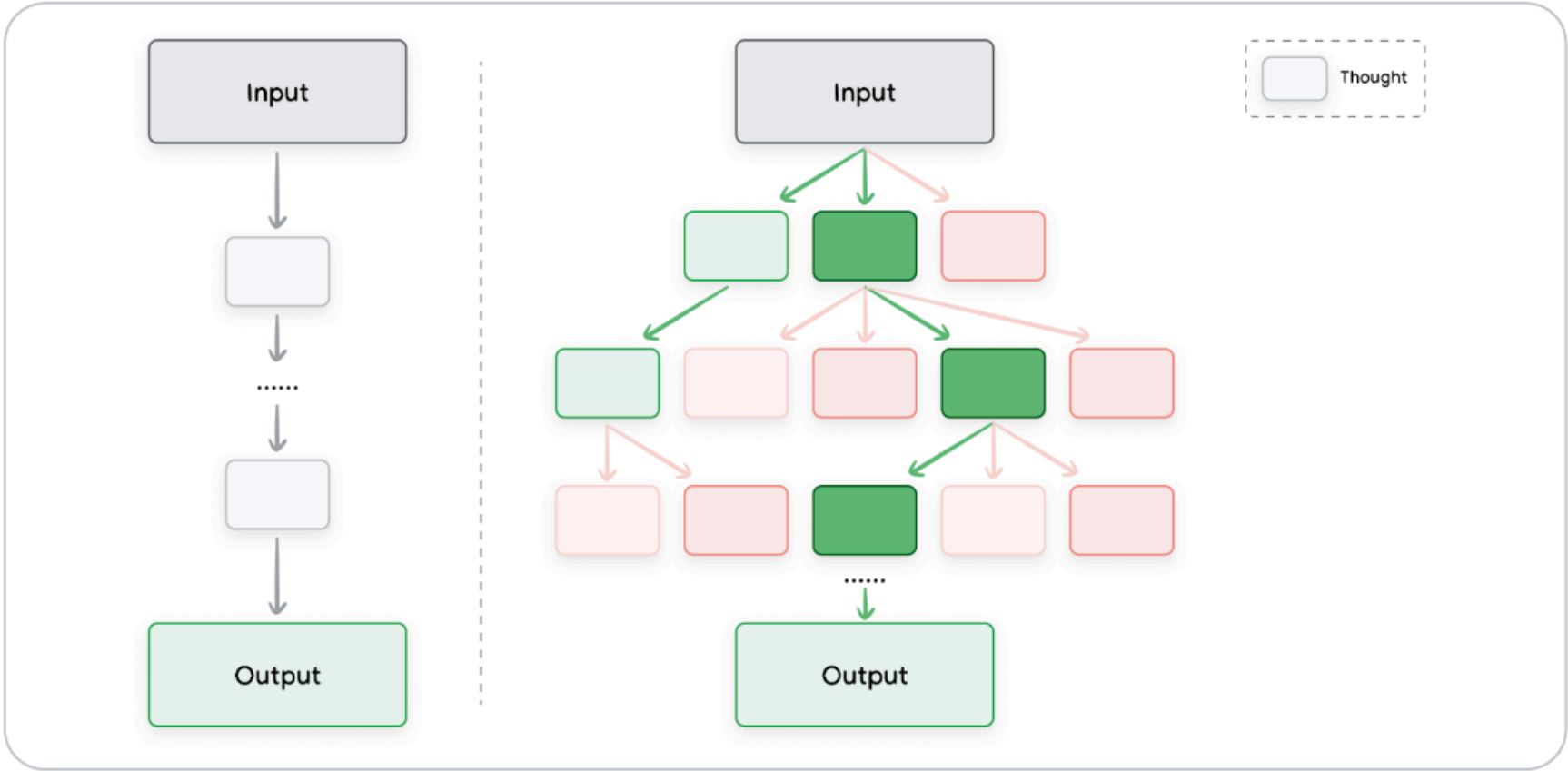


Рисунок 1. Визуализация промптинга с цепочкой размышлений слева против промптинга с деревом мыслей справа

Этот подход делает ToT особенно хорошо подходящим для сложных задач, требующих исследования. Он работает путем поддержания дерева 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 в дереве.

Есть отличный блокнот, который более подробно показывает Дерево мыслей (ToT), основанное на статье 'Large Language Model Guided Tree-of-Thought'.

ReAct (рассуждение и действие)

Рассуждение и действие (ReAct) [10] — это парадигма промптинга, позволяющая LLM решать сложные задачи, используя рассуждения на естественном языке в сочетании с внешними инструментами (поиск, интерпретатор кода и т.д.), позволяя LLM 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 хорошо работает по сравнению с другими подходами к инженерии промптов в различных областях.

ReAct промптинг работает путем объединения рассуждения и действия в цикл мысль-действие. LLM сначала рассуждает о проблеме и генерирует план действий. Затем она выполняет действия в плане и наблюдает результаты. Затем LLM использует наблюдения для обновления своих рассуждений и генерации нового плана действий. Этот процесс продолжается до тех пор, пока LLM 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 pip google-search-results.

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

Далее давайте напишем некоторый код Python с задачей для LLM выяснить: *Сколько детей имеют известные отцы, выступающие в группе Metallica.*

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

prompt = "How many kids do the band members of Metallica have?"

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 1. Создание ReAct агента с LangChain и VertexAI

Фрагмент кода 2 показывает результат. Обратите внимание, что ReAct делает цепочку из пяти поисков. На самом деле, LLM скрапит результаты поиска Google, чтобы выяснить имена участников группы. Затем она перечисляет результаты как

наблюдения и связывает мысль для следующего поиска.

```
> Entering new AgentExecutor chain...
Metallica has 4 members.
Action: Search
Action Input: How many kids does James Hetfield have?
Observation: three children
Thought: 1/4 Metallica band members have 3 children
Action: Search
Action Input: How many kids does Lars Ulrich have?
Observation: 3
Thought: 2/4 Metallica band members have 6 children
Action: Search
Action Input: How many kids does Kirk Hammett have?
Observation: Hammett has been married to his second wife Lani since 1998.
They have two sons, Angel (b. September 29, 2006) and Vincenzo (b. June
28, 2008).
Thought: 3/4 Metallica band members have 8 children
Action: Search
Action Input: How many kids does Robert Trujillo have?
Observation: 2
Thought: 4/4 Metallica band members have 10 children
Final Answer: 10
```

Фрагмент 2. Выполнение ReAct агента в терминале

Фрагмент кода 2 выясняет, что в группе Metallica четыре участника. Затем он ищет каждого участника группы, чтобы запросить общее количество детей и суммировать общее количество. Наконец, он возвращает общее количество детей как окончательный ответ.

ReAct промптинг на практике требует понимания того, что вы постоянно должны повторно отправлять предыдущие промпты/ответы (и обрезать лишний сгенерированный контент), а также настраивать модель с соответствующими примерами/инструкциями. Пожалуйста, обратитесь к [блокноту](#), размещенному в репозитории Github GoogleCloudPlatform, который более подробно показывает фактические входные и выходные данные LLM с более сложным примером.

Автоматическая инженерия промптов

На этом этапе вы можете осознать, что написание промпта может быть сложным. Не было бы здорово автоматизировать это (написать промпт для написания промптов)? Что ж, есть метод: [Автоматическая инженерия промптов \(APE\)](#). Этот метод не только снижает потребность в человеческом вводе, но и улучшает производительность модели в различных задачах.

Вы будете побуждать модель генерировать больше промптов. Оценивать их, возможно, изменять хорошие. И повторять.

Например, вы можете использовать автоматическую инженерию промптов, которая поможет с обучением чат-бота для интернет-магазина футболок с атрибутикой групп. Мы хотим выяснить все различные способы, которыми клиенты могут 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 атрибутикой групп, и для обучения чат-бота нам нужны различные способы заказа: "Одна футболка Metallica размера S". Сгенерируйте 10 вариантов с той же семантикой, но сохраняя то же значение.
Вывод	1. Я хотел бы приобрести футболку Metallica размера S. 2. Могу я заказать футболку Metallica маленького размера? 3. Я хотел бы купить одну футболку Metallica размера S. 4. Одну футболку Metallica, размер S, пожалуйста. 5. Я хочу одну футболку Metallica размера S. 6. Я хотел бы разместить заказ на одну футболку Metallica размера S. 7. Футболка Metallica размера S, одну, пожалуйста. 8. Я ищу одну футболку Metallica размера S. 9. Можете достать мне одну футболку Metallica размера S? 10. Одну футболку Metallica размера S, пожалуйста.

Таблица 15. Использование LLM для генерации промптов

2. Оцените всех кандидатов на инструкции, 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 на инструкцию с наивысшей оценкой. Этот кандидат будет окончательным промптом, который вы можете использовать в своем программном приложении или чат-боте. Вы также можете настроить выбранный промпт и оценить снова.

Промптинг кода

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

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 займет 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 публичном 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 в том, что вы можете настроить температуру и т.д.

Название	1_bash_rename_files_in_folder		
Цель	Написать промпт для написания кода на Bash для переименования файлов в папке.		
Модель	gemini-pro		
Температура	0.1	Лимит токенов	1024
Тор-K	N/A	Тор-P	1
Промпт	Напишите фрагмент кода на Bash, который запрашивает имя папки. Затем он берет содержимое папки и переименовывает все файлы внутри, добавляя в начало имени файла слово draft.		
Вывод	<pre>#!/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" в начало имени файла for file in "\${files[@]}; do new_file_name="draft_\${basename "\$file"}" mv "\$file" "\$new_file_name" done echo "Файлы успешно переименованы."</pre>		

Таблица 16. Использование Gemini для программирования кода

Это кажется мне хорошим кодом - он даже документирован! Однако, поскольку LLM не могут рассуждать и повторяют обучающие данные, важно сначала прочитать и протестировать ваш код.

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

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

1. Скопируйте вывод из Таблицы 16 (без обертки текста ````bash````) и вставьте его в новый файл под 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, кажется, работает нормально. Вы увидите сообщение: **Файлы успешно переименованы.**
Когда вы посмотрите в тестовую папку, заметьте, что все файлы были идеально переименованы в **draft_filename.txt**.

Это сработало!

Промпты для объяснения кода

Как разработчик, когда вы работаете в команде, 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_explain_the_code		
Цель	Написать промпт для объяснения кода на Bash		
Модель	gemini-pro		
Температура	0.1	Лимит токенов	1024
Тор-K	N/A	Тор-P	1
Промпт	<p>Объясните мне следующий код на Bash:</p> <pre>#!/bin/bash echo "Enter the folder name: " read folder_name if [! -d "\$folder_name"]; then echo "Folder does not exist." exit 1 fi files=("\$folder_name"/*) for file in "\${files[@]}"; do new_file_name="draft_\$(basename "\$file")" mv "\$file" "\$new_file_name" done echo "Files renamed successfully."</pre>		
Вывод	<p>Этот скрипт на Bash выполняет следующие задачи:</p> <p>1. Пользовательский ввод:</p> <p>- Он предлагает пользователю ввести имя папки, используя команды <code>echo</code> и <code>read</code> . Введенное имя папки сохраняется в переменной <code>folder_name</code> .</p> <p>2. Проверка существования папки:</p> <p>- Он проверяет, существует ли папка с указанным именем, используя условие <code>[! -d "\$folder_name"]</code> . Если папка не существует, он отображает сообщение об ошибке ("Папка не существует.") и завершает скрипт с кодом выхода 1.</p> <p>3. Список файлов:</p> <p>- Он перечисляет все файлы внутри указанной папки, используя команду <code>files=("\$folder_name"/*)</code> . Это создает массив с именем <code>files</code> , содержащий пути ко всем файлам в папке.</p> <p>4. Переименование файлов:</p> <p>- Он перебирает каждый файл в массиве <code>files</code> , используя цикл <code>for</code> . - Для каждого файла он генерирует новое имя файла, используя команду <code>new_file_name="draft_\$(basename "\$file")"</code> . Это новое имя файла создается путем добавления префикса "draft_" к исходному имени файла. - Затем он переименовывает файл, используя команду <code>mv "\$file" "\$new_file_name"</code> , которая перемещает файл в новое имя файла.</p>		

Промпты для перевода кода

Код на Bash из Таблицы 16, кажется, работает нормально. Однако этот скрипт действительно мог бы быть многоразовым, если бы он запрашивал у меня 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. LLM могут помочь с переводом кода с одного языка на другой. См. пример в Таблице 18:

Название	1_python_rename_files		
Цель	Написать промпт для перевода кода с Bash на Python		
Модель	gemini-pro		
Температура	0.1	Лимит токенов	1024
Топ-K	N/A	Топ-P	1
Промпт	<div>Переведите следующий код на Bash в фрагмент Python. #!/bin/bash echo "Enter the folder name: " read folder_name if [! -d "\$folder_name"]; then echo "Folder does not exist." exit 1 fi files=("\$folder_name"/*) for file in "\${files[@]}"; do new_file_name="draft_\$(basename "\$file")" mv "\$file" "\$new_file_name" done echo "Files renamed successfully."</div>		
Вывод	<div>import os import shutil # Get the folder name from the user folder_name = input("Enter the folder name: ") # Check if the folder exists if not os.path.isdir(folder_name): print("Folder does not exist.") exit(1) # Get a list of all files in the folder files = os.listdir(folder_name) # Iterate over the files for file in files: # Create a new file name with the prefix "draft_" new_file_name = f"draft_{file}" # Move the file to the new name shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name)) # Print a success message print("Files renamed successfully.")</div>		

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 (Python) кода в Language Studio в Vertex AI вам нужно будет нажать на кнопку 'Markdown'. В противном случае вы получите обычный текст, в котором отсутствует правильный отступ 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 заглавными буквами. Посмотрите на пример кода в Фрагменте *3, но какая неприятность. Теперь он возвращает ошибки Python!

Python

```
import os
import shutil

folder_name = input("Enter the folder name: ")
prefix = input("Enter the string to prepend to the filename: ")
text = toUpperCase(prefix)

if not os.path.isdir(folder_name):
    print("Folder does not exist.")
    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("Files renamed successfully.")
```

Фрагмент 3. Сломанный скрипт Python

Ой! Это похоже на ошибку:

Следующий код Python 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 4. Я сломал код Python

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

Название	1_python_debug_code		
Цель	Написать промпт для отладки и проверки кода Python.		
Модель	gemini-pro		
Температура	0.1	Лимит токенов	1024
Тор-К	N/A	Тор-P	1
Промпт	<p>Следующий код Python выдает ошибку:</p> <pre>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</pre> <p>Отладьте, что не так, и объясните, как я могу улучшить код.</p> <pre>import os import shutil # Get the folder name from the user folder_name = input("Enter the folder name: ") prefix = input("Enter the string to prepend to the filename: ") text = toUpperCase(prefix) # Check if the folder exists if not os.path.isdir(folder_name): print("Folder does not exist.") exit(1) # Get a list of all files in the folder files = os.listdir(folder_name) # Iterate over the files for file in files: # Create a new file name with the prefix "draft_" new_filename = f"{text}_{file}" # Move the file to the new name shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_filename)) # Print a success message print("Files renamed successfully.")</pre>		
	<p>Код вызывает функцию <code>toUpperCase</code> для преобразования <code>prefix</code> в верхний регистр, но эта функция не определена. Чтобы исправить эту проблему, вы можете использовать метод <code>upper()</code> класса <code>string</code>, который преобразует заданную строку в верхний регистр. Измененный код показан ниже:</p> <pre>import os import shutil # Get the folder name from the user folder_name = input("Enter the folder name: ") prefix = input("Enter the string to prepend to the filename: ") text = prefix.upper() # Check if the folder exists if not os.path.isdir(folder_name): print("Folder does not exist.") exit(1) # Get a list of all files in the folder files = os.listdir(folder_name) # Iterate over the files for file in files: # Create a new file name with the prefix "draft_" new_filename = f"{text}_{file}" # Move the file to the new name shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_filename))</pre>		

Вывод	<pre># Print a success message print("Files renamed successfully.")</pre> <p>Кроме того, можно внести несколько других улучшений в код:</p> <ol style="list-style-type: none">1. Расширение файла новых имен файлов 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 хорошей практикой является корректная обработка пробелов, заменяя их подчеркиваниями или дефисами.3. Использование синтаксиса строки <code>f</code> для интерполяции строк более читабельно и лаконично, чем традиционный оператор <code>+</code>.4. Код не обрабатывает ошибки, которые могут возникнуть в процессе переименования. Было бы лучше обернуть вызов <code>shutil.move</code> в блок <code>try...except</code>, чтобы перехватить любые потенциальные ошибки. <p>Вот улучшенный код с этими предложениями:</p> <pre>import os import shutil # Get the folder name from the user folder_name = input("Enter the folder name: ").replace(" ", "_") prefix = input("Enter the string to prepend to the filename: ") text = prefix.upper() # Check if the folder exists if not os.path.isdir(folder_name): print("Folder does not exist.") exit(1) # Get a list of all files in the folder files = os.listdir(folder_name) # Iterate over the files for file in files: # Get the file extension ext = os.path.splitext(file)[1] # Create a new file name with the prefix and the file extension new_filename = f"{prefix}_{os.path.splitext(file)[0]}{text}" # Build the full paths for the original and new files old_path = os.path.join(folder_name, file) new_path = os.path.join(folder_name, new_filename) # Try to move the file try: shutil.move(old_path, new_path) except Exception as e: print(f"An error occurred while moving the file {file}: {e}") # Print a success message print("Files renamed successfully.")</pre>
-------	--

Это отлично. Он не только рассказал мне, как решить проблему, но и выяснил, что в моем коде было больше ошибок, и как их тоже решить. Последняя часть промпта дала предложения по улучшению кода в целом.

А как насчет мультимодального промптинга?

Промптинг для кода по-прежнему использует ту же обычную большую языковую модель. Мультимодальный промптинг — это отдельная задача, он относится к технике, при которой вы используете несколько форматов ввода для управления большой языковой моделью, 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 поставленной задачи.

Лучшие практики

Поиск правильного промпта требует экспериментов. Language Studio в Vertex AI — идеальное место для экспериментов с вашими промптами, с возможностью тестирования на различных моделях.

Используйте следующие лучшие практики, чтобы стать профессионалом в инженерии промптов.

Предоставляйте примеры

Самая важная лучшая практика — предоставлять (one shot / few shot) примеры в промпте. Это очень эффективно, потому что действует как мощный обучающий инструмент. Эти примеры демонстрируют желаемые выходные данные или похожие

ответы, позволяя модели учиться на них и соответственно адаптировать свою собственную генерацию. Это похоже на предоставление модели ориентира или цели, к которой нужно стремиться, улучшая точность, стиль и тон ее ответа, чтобы лучше соответствовать вашим ожиданиям.

Проектируйте с простотой

Промпты должны быть лаконичными, ясными и легкими для понимания как для вас, так и для модели. Как правило, если это уже запутанно для вас, то, вероятно, будет запутанно и для модели. Старайтесь не использовать сложный язык и не предоставлять ненужную информацию.

Примеры:

ДО:

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

ПОСЛЕ ПЕРЕПИСЫВАНИЯ:

Выступите в роли туристического гида. Опишите отличные места для посещения в Нью-Йорке на Манхэттене с 3-летним ребенком.

Попробуйте использовать глаголы, которые описывают действие. Вот набор примеров:

Действуйте, Анализируйте, Категоризируйте, Классифицируйте, Противопоставляйте, Сравнивайте, Создавайте, Описывайте, Определяйте, Оценивайте, Извлекайте, Находите, Генерируйте, Идентифицируйте, Перечисляйте, Измеряйте, Организуйте, Разбирайте, Выбирайте, Предсказывайте, Предоставляйте, Ранжируйте, Рекомендуйте, Возвращайте, Извлекайте, Переписывайте, Выбирайте, Показывайте, Сортируйте, Суммируйте, Переводите, Пишите.

Будьте конкретны в отношении вывода

Будьте конкретны в отношении желаемого вывода. Краткая инструкция может недостаточно направлять LLM или может быть слишком общей. Предоставление конкретных деталей в промпте (через системный или контекстный промптинг) может помочь модели сосредоточиться на том, что важно, улучшая общую точность.

Примеры:

ДЕЛАЙТЕ:

Сгенерируйте 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:

Сгенерируйте блог-пост об игровых консолях.

Используйте инструкции вместо ограничений

Инструкции и ограничения используются в промптинге для направления вывода LLM.

- **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 чего следует избегать.

Растущие исследования показывают, что фокусирование на положительных инструкциях в промптинге может быть более эффективным, чем сильное полагание на ограничения. Этот подход соответствует тому, как люди предпочитают положительные инструкции спискам того, чего не следует делать.

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

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

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

ДЕЛАЙТЕ:

Сгенерируйте блог-пост из 1 абзаца о 5 лучших игровых консолях. Обсуждайте только консоль, компанию, которая ее создала, год и общие продажи.

НЕ ДЕЛАЙТЕ:

Сгенерируйте блог-пост из 1 абзаца о 5 лучших игровых консолях. Не перечисляйте названия видеоигр.

В качестве лучшей практики начните с приоритизации инструкций, четко указывая, что вы хотите, чтобы модель делала, и используйте ограничения только при необходимости для безопасности, ясности или конкретных требований. Экспериментируйте и итерируйте, чтобы проверить различные комбинации инструкций и ограничений, чтобы найти то, что лучше всего работает для ваших конкретных задач, и документируйте их.

Контролируйте максимальную длину токенов

Чтобы контролировать длину сгенерированного ответа LLM, вы можете либо установить максимальный лимит токенов в конфигурации, либо явно запросить определенную длину в вашем промпте. Например:

"Объясните квантовую физику в сообщении длиной с твит."

Используйте переменные в промптах

Чтобы повторно использовать промпты 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 дает факты о городе. Вместо жесткого кодирования названия города в промпте, используйте переменную. Переменные могут сэкономить ваше время и усилия, позволяя избежать повторения. Если вам нужно использовать один и тот же фрагмент информации в нескольких промптах, вы можете сохранить его в переменной, а затем ссылаться на эту переменную в каждом промпте. Это имеет большой смысл при интеграции промптов в ваши собственные приложения.

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

Таблица 20. Использование переменных в промптах

Экспериментируйте с форматами ввода и стилями написания

Различные модели, конфигурации моделей, форматы промптов, выбор слов и отправки могут давать разные результаты. Поэтому важно экспериментировать с атрибутами промпта, такими как стиль, выбор слов и тип промпта (zero shot, few shot, системный промпт).

Например, промпт с целью генерации текста о революционной игровой консоли Sega Dreamcast может быть сформулирован как **вопрос**, **утверждение** или **инструкция**, что приведет к разным выводам:

- **Вопрос:** Что такое Sega Dreamcast и почему это была такая революционная консоль?
- **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 few-shot промптинга с задачами классификации, смешивайте классы

В общем, порядок ваших few-shot примеров не должен иметь большого значения. Однако при выполнении задач классификации убедитесь, что вы смешиваете возможные классы ответов в few-shot примерах. Это потому, что в противном случае вы можете переобучиться к конкретному порядку примеров. Смешивая возможные классы ответов, вы можете убедиться, что модель учится идентифицировать ключевые особенности каждого класса, а не просто запоминает порядок примеров. Это приведет к более надежной и обобщаемой производительности на невиданных данных.

Хорошее эмпирическое правило — начать с 6 few-shot примеров и начать тестировать точность оттуда.

Адаптируйтесь к обновлениям модели

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

Экспериментируйте с форматами вывода

Помимо формата ввода промпта, рассмотрите возможность экспериментирования 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 объектов JSON из промпта, который извлекает данные. В реальном приложении мне не нужно вручную создавать этот формат JSON, 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 создавать структуру и ограничивать галлюцинации.

В итоге, преимущества использования JSON для вашего вывода:

- Всегда возвращается в одном и том же стиле
- Фокус на данных, которые вы хотите получить
- Меньше шансов для галлюцинаций
- Делает его осведомленным о взаимосвязях
- Вы получаете типы данных
- Вы можете его сортировать

Таблица 4 в разделе few-shot промптинга показывает пример того, как возвращать структурированный вывод.

Восстановление JSON

Хотя возврат данных в формате JSON предлагает множество преимуществ, он не лишен недостатков. Структурированная природа JSON, хотя и полезна для анализа 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 резко обрывается из-за ограничений токенов. Это усечение часто приводит к недействительному JSON, отсутствию критически важных закрывающих 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 исправить неполные или неправильно сформированные объекты JSON, что делает ее важным союзником при работе с JSON, сгенерированным LLM, 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, который генерирует LLM, 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 даете LLM четкий план данных, которые он должен ожидать, помогая ему сосредоточить свое внимание на соответствующей информации и снижая риск неправильного толкования ввода. Кроме того, схемы могут помочь установить отношения между различными частями данных и даже сделать LLM "осведомленным о времени", включая поля даты или временной метки с определенными форматами.

Вот простой пример:

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

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string", "description": "Product name" },
```



```
    "category": { "type": "string", "description": "Product category" },
    "price": { "type": "number", "format": "float", "description": "Product price" },
    "features": {
      "type": "array",
      "items": { "type": "string" },
      "description": "Key features of the product"
    },
    "release_date": { "type": "string", "format": "date", "description": "Date the product was released" }
  },
}
```

Фрагмент 5. Определение схемы структурированного вывода

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

```
{
  "name": "Wireless Headphones",
  "category": "Electronics",
  "price": 99.99,
  "features": ["Noise cancellation", "Bluetooth 5.0", "20-hour battery life"],
  "release_date": "2023-10-27"
}
```

Фрагмент 6. Структурированный вывод от LLM

Предварительно обрабатывая ваши данные и вместо предоставления полных документов предоставляя только схему и данные, вы даете LLM 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 LLM на соответствующие поля, особенно ценен при работе с большими объемами данных или при интеграции LLM в сложные приложения.

Экспериментируйте вместе с другими инженерами промптов

Если вы находитесь в ситуации, когда вам нужно попытаться придумать хороший промпт, вы можете захотеть найти несколько человек 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

Для промптинга CoT размещение ответа после рассуждения необходимо, потому что генерация рассуждения изменяет токены, которые модель получает при прогнозировании окончательного ответа.

С CoT и самосогласованностью вам нужно иметь возможность извлечь окончательный ответ из вашего промпта, отделенный от рассуждения.

Для промптинга CoT установите температуру на 0.

Промптинг с цепочкой размышлений основан на жадном декодировании, предсказывающем следующее слово в последовательности на основе наивысшей вероятности, присвоенной языковой моделью. В общем, когда используется рассуждение для получения окончательного ответа, вероятно, есть только один правильный ответ. Поэтому температура всегда должна быть установлена на 0.

Документируйте различные попытки промптов

Последний совет был упомянут ранее в этой главе, но мы не можем достаточно подчеркнуть, насколько это важно: документируйте ваши попытки промптов во всех деталях, чтобы вы могли со временем узнать, что прошло хорошо, а что нет.

Выводы промптов могут отличаться в разных моделях, в разных настройках выборки и даже в разных версиях одной и той же модели. Более того, даже в идентичных промптах к одной и той же модели могут возникать небольшие различия в форматировании предложений 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 Sheet с Таблицей 21 в качестве шаблона. Преимущества этого подхода заключаются в том, что у вас есть полная запись, когда вам неизбежно придется пересмотреть вашу работу с промптами — либо чтобы продолжить ее в будущем (вы удивитесь, как много вы можете забыть после короткого перерыва), чтобы проверить производительность промпта на разных версиях модели, и чтобы помочь отладить будущие ошибки.

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

При работе над системой генерации с дополнением извлечения (RAG), вы также должны фиксировать конкретные аспекты системы RAG, которые влияют на то, какой контент был вставлен в промпт, включая запрос, настройки чанков, вывод чанков и другую информацию.

Как только вы почувствуете, что промпт близок к совершенству, перенесите его в кодовую базу вашего 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 инженер промптов вы должны полагаться на автоматизированные тесты и процедуры оценки, чтобы понять, насколько хорошо ваш промпт обобщается для задачи.

Инженерия промптов — это итеративный процесс. Создавайте и тестируйте различные промпты, анализируйте и документируйте результаты. Улучшайте ваш 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	[имя и версия используемой модели]		
Температура	[значение между 0 - 1]	Лимит токенов	[число]
Тор-К	[число]	Тор-Р	[число]
Промпт	[Напишите весь полный промпт]		
Вывод	[Выпишите вывод или несколько выводов]		

Таблица 21. Шаблон для документирования промптов

Резюме

В этой белой книге обсуждается инженерия промптов. Мы изучили различные техники промптинга, такие как:

- Нулевой промптинг (Zero prompting)
- Промптинг с несколькими примерами (Few shot prompting)
- Системный промптинг (System prompting)
- Ролевой промптинг (Role prompting)
- Контекстный промптинг (Contextual prompting)
- Промптинг с шагом назад (Step-back prompting)
- Цепочка размышлений (Chain of thought)
- Самосогласованность (Self consistency)
- Дерево мыслей (Tree of thoughts)
- ReAct (рассуждение и действие)

Мы даже рассмотрели способы автоматизации ваших промптов.

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

Примечания

1. Google, 2023, Gemini by Google. Доступно по адресу: <https://gemini.google.com> .
2. Google, 2024, Gemini for Google Workspace Prompt Guide. Доступно по адресу: <https://inthecloud.withgoogle.com/gemini-for-google-workspace-prompt-guide/dl-cd.html> .
3. Google Cloud, 2023, Introduction to Prompting. Доступно по адресу: <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. Доступно по адресу: 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. Доступно по адресу: <https://arxiv.org/pdf/2109.01652.pdf> .
6. Google Cloud, 2023, Google Cloud Model Garden. Доступно по адресу: <https://cloud.google.com/model-garden> .
7. Brown, T., et al., 2023, Few Shot - Language Models are Few Shot learners. Доступно по адресу: <https://arxiv.org/pdf/2005.14165.pdf> .
8. Zheng, L., et al., 2023, Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. Доступно по адресу: <https://openreview.net/pdf?id=3bq3jsvcQ1>
9. Wei, J., et al., 2023, Chain of Thought Prompting. Доступно по адресу: <https://arxiv.org/pdf/2201.11903.pdf> .
10. Google Cloud Platform, 2023, Chain of Thought and React. Доступно по адресу: 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. Доступно по адресу: <https://arxiv.org/pdf/2203.11171.pdf> .

12. Yao, S., et al., 2023, Tree of Thoughts: Deliberate Problem Solving with Large Language Models. Доступно по адресу: <https://arxiv.org/pdf/2305.10601.pdf> .
13. Yao, S., et al., 2023, ReAct: Synergizing Reasoning and Acting in Language Models. Доступно по адресу: <https://arxiv.org/pdf/2210.03629.pdf> .
14. Google Cloud Platform, 2023, Advance Prompting: Chain of Thought and React. Доступно по адресу: https://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genai-on-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. Доступно по адресу: <https://arxiv.org/pdf/2211.01910.pdf> .