Глава 13. Промт инжиниринг для улучшения RAG

Это продолжение перевода книги Кит Борн. Раскрытие потенциала данных с помощью генеративного ИИ и технологии RAG. Быстрый вопрос: что вы используете для генерации контента с помощью большой языковой модели (LLM)? Промт! Очевидно, что промт является ключевым элементом любого приложения на основе генеративного ИИ, а значит, и любого приложения RAG. Системы RAG сочетают в себе возможности поиска информации и генеративных языковых моделей, чтобы повысить качество и релевантность сгенерированного текста. Инженерия промтов в этом контексте означает стратегическое формулирование и корректировку входных промтов для улучшения поиска информации, что, в свою очередь, повышает качество генерации.

Инженерия промтов — это настолько обширная тема, что о ней можно написать целые книги. Существует множество стратегий, направленных на разные аспекты работы с промтами, которые помогают улучшить результаты работы LLM. Однако в этой главе мы сосредоточимся на стратегиях, наиболее применимых к RAG-приложениям. В этой главе мы рассмотрим:

- Ключевые концепции и параметры инженерии промтов
- Основы проектирования и оптимизации промтов для RAG-приложений
- Адаптацию промтов для различных LLM, не только моделей OpenAI
- Лаборатория кода 13.1 — создание пользовательских шаблонов промтов
- Лаборатория кода 13.2 — варианты промтов

К концу этой главы вы получите крепкую базу знаний по инженерии промтов для RAG, а также освоите практические методы оптимизации промтов для улучшения поиска информации, генерации качественного текста и адаптации под конкретные сценарии использования.

Предыдущая глава Содержание Следующая глава

Код для этой главы размещен в репозитории GitHub.

Мы начнем обсуждение с ключевых концепций промтов, начиная с параметров промтов.

Параметры промтов

Существует множество параметров, характерных для большинства LLM, но мы рассмотрим только те, которые оказывают наибольшее влияние на RAG-приложения:

- Температура (temperature)
- Вероятностное сужение (top-p)
- Зерно случайности (seed)

Температура

Если представить выходные данные LLM как последовательность токенов, то модель предсказывает следующий токен на основе предоставленного текста и ранее сгенерированных токенов. Выбор следующего слова зависит от распределения вероятностей, где некоторые слова имеют значительно более высокие шансы быть выбранными, чем другие. Однако модель всегда может выбрать менее вероятное слово.

Во многих случаях вероятность некоторых слов будет значительно выше, чем у остальных, но модель все же может выбрать менее вероятное слово. Температура — это параметр, который определяет, насколько вероятно, что модель выберет слово с низкой вероятностью из распределения. Другими словами, этот параметр позволяет управлять степенью случайности вывода модели. Температуру можно передавать в LLM в качестве параметра. Он необязателен. Если его не указывать, по умолчанию используется значение 1. Можно задать температуру в диапазоне от 0 до 2. Более высокие значения делают вывод более случайным, что означает, что модель будет чаще учитывать менее вероятные слова, тогда как более низкие значения, наоборот, уменьшают степень случайности.

Рассмотрим простой пример распределения вероятностей следующего слова, чтобы проиллюстрировать, как работает температура. Допустим, у нас есть предложение The dog ran, и мы ждем, пока модель предскажет следующее слово. Пусть, основываясь на обучении модели и всей другой информации, которую она учитывает при предсказании, распределение условных вероятностей выглядит так:

P("следующее слово" | "The dog ran") = {"down": 0.4, "to": 0.3, "with": 0.2, "away": 0.1}

Сумма всех вероятностей составляет 1. Наиболее вероятное слово — "down" (0.4), второе по вероятности — "to" (0.3). Однако это не означает, что слово "away" (0.1) никогда не будет выбрано. Модель использует вероятностный механизм выбора, и в некоторых случаях менее вероятное слово может быть выбрано случайно. В некоторых сценариях это может быть преимуществом для RAG-приложения, а в других — недостатком. Если температура установлена в 0, модель всегда выбирает наиболее вероятное слово. Если температура равна 2, модель значительно чаще рассматривает все возможные варианты и может случайным образом выбрать менее вероятное слово. Другими словами, увеличение температуры делает модель более случайной.

Мы использовали температуру с самого начала, устанавливая ее в 0. Вот строка, которую мы добавили:

llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

Цель такого подхода — сделать результаты кода более предсказуемыми, чтобы при каждом запуске они были схожими. Ваши результаты могут немного отличаться, но при нулевой температуре шансы на их повторяемость значительно выше.

Однако не всегда стоит использовать температуру 0. Например, если вам нужен более креативный вывод от LLM, температуру можно использовать в свою пользу при разработке RAG-приложения.

Температура и вероятностное сужение (top-p) связаны между собой, так как оба параметра управляют случайностью вывода модели. Однако они работают по-разному. Давайте разберем, что такое top-p и в чем отличие от температуры.

Вероятностное сужение (top-p)

Как и температура, top-p помогает влиять на случайность вывода модели. Однако если температура определяет общий уровень случайности, top-p управляет тем, какая часть вероятностного распределения будет использоваться.

Рассмотрим наш предыдущий пример:

P("следующее слово" | "The dog ran") = {"down": 0.4, "to": 0.3, "with": 0.2, "away": 0.1}

Мы уже отмечали, что общая сумма вероятностей здесь равна 1.0. С помощью top-p можно указать, какая часть вероятностей должна учитываться. Например, если top-p = 0.7, модель учтет только первые два слова ("down" и "to"), так как их суммарная вероятность достигает 0.7. С температурой такого точечного контроля нет — она равномерно воздействует на все вероятности.

Параметр top-p является необязательным. Если не задавать его, используется значение по умолчанию = 1, то есть модель учитывает все возможные варианты. Может возникнуть соблазн использовать одновременно температуру и top-p, но это может сделать поведение модели слишком сложным и непредсказуемым. Поэтому обычно рекомендуется использовать либо один, либо другой параметр, но не оба сразу.

Параметр LLM	Результат
Температура (temperature)	Общая случайность
Вероятностное сужение (top-p)	Локализованная случайность
Температура + top-p	Непредсказуемая сложность

Рис. 13.1. Влияние различных параметров LLM на результат

Теперь разберемся, как использовать top-p в модели. Этот параметр передается не напрямую, а как часть переменной model_kwargs:

llm = ChatOpenAI(model_name="gpt-4o-mini", model_kwargs={"top_p": 0.5})

Переменная model_kwargs позволяет передавать параметры, которые не встроены в LangChain напрямую, но поддерживаются API LLM. Top-p является параметром этой версии ChatGPT, но в других моделях он может называться иначе или вовсе отсутствовать. Перед использованием проверьте документацию API вашей модели, чтобы убедиться, что передаете параметры правильно.

Теперь, когда мы разобрались с настройками случайности, давайте рассмотрим параметр seed, который позволяет контролировать случайность.

Зерно случайности (seed)

По умолчанию ответы LLM недетерминированы, то есть результаты генерации могут отличаться при каждом запросе. Однако во многих задачах требуется повторяемость и детерминированность. Эти требования кажутся противоречивыми, но OpenAI и другие разработчики добавили механизм контроля — параметр seed и поле ответа system_fingerprint. Seed — это широко используемый параметр в системах, связанных с генерацией случайных чисел или данных.

При использовании seed генерация остается случайной, но при этом можно получать одни и те же результаты при повторных запросах с одинаковым seed. Это дает возможность (в основном) детерминированных результатов при вызовах API. Вы можете задать seed любым целым числом и использовать одно и то же значение в разных запросах, чтобы получать повторяемые результаты. Более того, если включен seed, даже при использовании случайных настроек (таких как temperature или top-p), вы все равно будете (в большинстве случаев) получать идентичные ответы.

Важно понимать, что даже при использовании seed результаты могут отличаться. Это связано с тем, что API взаимодействует с сервисом, который постоянно обновляется, и эти изменения могут влиять на ответы со временем.

Некоторые модели, такие как ChatGPT, предоставляют в своих ответах поле system_fingerprint. Его можно использовать для отслеживания изменений в системе, которые могут вызывать различия в ответах. Если значение system_fingerprint изменилось с момента последнего вызова LLM API (даже при использовании того же seed), значит, OpenAI внес изменения в систему, и это могло повлиять на итоговый результат.

Параметр seed является необязательным и не включен в стандартные параметры LLM в LangChain. Поэтому, как и в случае с top-p, его необходимо передавать через model_kwargs:

```
optional_params = {
   "top_p": 0.5,
   "seed": 42
}
llm = ChatOpenAI(model_name="gpt-4o-mini", model_kwargs=optional_params)
```

Здесь мы добавляем параметр seed вместе с top-p в словарь параметров, который затем передаем в model_kwargs.

Существует много других параметров, которые можно использовать в различных LLM, и мы рекомендуем изучить их самостоятельно. Однако именно эти параметры (temperature, top-p, seed) оказывают наибольшее влияние на RAG-приложение.

Теперь перейдем к следующей ключевой концепции, связанной с промтами — количеству примеров (shot concept), которое определяет, сколько фоновой информации вы передаете LLM.

Выбор количества примеров

Термины no-shot, single-shot, few-shot и multi-shot часто встречаются в обсуждениях стратегий промтов. Все они связаны с одной и той же концепцией: *shot* — это пример, который вы даете LLM, чтобы помочь ей понять, как отвечать на ваш запрос. Если это пока не совсем понятно, то можно привести пример. А, подождите, именно в этом и заключается суть shot-концепции!

Вы можете передать 0 примеров (no-shot), 1 пример (single-shot) или несколько примеров (few-shot или multi-shot). Пример single-shot промта для LLM:

> Дай мне шутку, в которой используется животное и действие, которое оно выполняет.
> Используй этот пример как ориентир:
> Вопрос-шутка: Почему курица перешла дорогу?
> Ответ-шутка: Чтобы попасть на другую сторону.

Смысл подхода в том, что, предоставляя этот пример, вы помогаете LLM понять, в каком формате должен быть ответ. В RAG-приложениях вы часто включаете примеры в контекст запроса. Однако это не всегда обязательно — иногда контекст содержит просто дополнительные данные. Если же вы

предоставляете примеры вопросов и ответов в контексте с целью направить LLM на ответ в аналогичной манере, то это и есть shot-метод. Некоторые RAG-приложения предпочитают многошаговый (multi-shot) подход, но это зависит от задач приложения и доступных данных. Примеры (shots) — не единственная важная концепция в промтах. Также важно понимать различные подходы к формированию промтов, о которых мы поговорим далее.

Промты, их проектирование и инженерия

В разделе терминологии главы 1 мы уже обсуждали эти три концепции и их взаимосвязь. Мы тогда определили их следующим образом:

- Промтинг (Prompting) — это процесс отправки запроса (промта) в LLM.
- Проектирование промтов (Prompt design) — это стратегия, которую вы используете для разработки промта перед отправкой его в LLM. Существует множество стратегий, каждая из которых работает в разных сценариях.
- Инженерия промтов (Prompt engineering) — это технический процесс оптимизации промта, чтобы улучшить выходные данные модели. Например, сложный запрос можно разбить на несколько отдельных взаимодействий с LLM, что позволит получить более точные результаты.

Мы обещали вернуться к этим темам в главе 13, и вот мы выполняем обещание! Здесь мы не только снова рассмотрим эти концепции, но и покажем, как они реализуются в коде. Так как промтинг — это простая концепция, мы сосредоточимся на проектировании и инженерии промтов.

Подходы к проектированию и инженерии промтов

Когда выше мы говорили о shot-подходах, это относилось к проектированию промтов. Однако, когда мы подставляли в шаблон промта данные из других частей RAG-системы, это уже был пример инженерии промтов. Этот процесс называется *гидратацией* (hydrating) промта, и он является конкретным методом инженерии промтов. Таким образом, проектирование и инженерия промтов сильно пересекаются, и эти термины часто используются взаимозаменяемо.

В нашем случае мы обсудим их вместе, особенно в контексте улучшения RAG-приложений. Я встречал разные описания этих концепций, и пока что в нашей области не сложилось четких границ между ними. Для понимания их в рамках этой книги можно сказать, что **инженерия промтов** — это более широкое понятие, включающее не только разработку промта, но и оптимизацию, тонкую настройку и организацию взаимодействия пользователя с LLM.

Существует множество техник проектирования промтов, которые в теории могут улучшить RAG-приложение. Важно знать доступные варианты и понимать, в каких сценариях они применимы. Обычно требуется экспериментировать с разными подходами, чтобы определить, какой лучше всего подходит для вашего приложения. Не существует универсального решения для проектирования промтов. Мы приведем короткий список примеров, но рекомендуем изучить другие источники, чтобы найти подходы, полезные именно для ваших задач.

- *Проектирование примеров (Shot design).* Отправная точка при разработке стратегии промтов. Включает тщательную проработку начального промта с использованием примеров, чтобы направить модель к желаемому результату. Может сочетаться с другими техниками проектирования для улучшения качества и релевантности сгенерированного контента.
- *Промтинг с разбиением на этапы (Chain-of-thought prompting).* Разбивает сложные задачи на меньшие, более управляемые шаги, запрашивая у LLM промежуточные рассуждения на каждом этапе. Улучшает качество ответов, обеспечивая пошаговый логический процесс, что повышает точность и понимание ответа.
- *Персоны (персонализированные роли) (Personas / role prompting).* Заключается в создании вымышленного персонажа, соответствующего целевой аудитории (с именем, профессией, демографией, историей, проблемами и потребностями). Помогает адаптировать выходные данные к потребностям аудитории, придавая контенту стиль и характер. Мощный инструмент для настройки LLM под конкретные потребности пользователей.
- *Цепочка плотности (суммаризация) (Chain of density – summarization).* Контролирует, насколько полно и качественно модель суммировала текст, следит за тем, чтобы ключевая информация не была упущена. Оценивает *плотность сущностей* в процессе суммаризации, чтобы важные элементы были включены в итоговый текст.

- *Дерево мыслей (Tree of thoughts – exploration over thoughts).* Начинается с базового промта, который генерирует несколько возможных вариантов рассуждений. Затем модель поэтапно отбирает лучшие для следующего шага. Позволяет глубже и шире исследовать идеи, прежде чем сформировать окончательный ответ.
- *Графовый промтинг (Graph prompting).* Новый подход, разработанный специально для работы с графовыми структурами данных. Позволяет LLM анализировать и генерировать текст, учитывая связи между объектами в графе.
- *Дополнение знаниями (Knowledge augmentation).* Улучшает качество генерации за счет обогащения промтов дополнительной релевантной информацией. Может реализовываться через RAG, который включает внешние знания в контекст запроса.
- *Промты "Покажи мне" vs. "Расскажи мне" (Show Me vs. Tell Me prompts).* Два разных способа давать инструкции генеративным моделям: *Show Me* — передача примеров и демонстраций. *Tell Me* — передача явных текстовых инструкций или документации. Использование обоих подходов дает большую гибкость и может повысить точность модели, в зависимости от контекста и сложности задачи.

Этот список лишь поверхностно затрагивает множество методов, применяемых в инженерии промтов. Так как эта область продолжает развиваться, в будущем появятся новые инновационные техники, которые еще больше улучшат возможности генеративных моделей ИИ.

Далее мы рассмотрим основные принципы проектирования промтов, которые могут быть полезны для RAG-приложений.

Основы проектирования промтов

При создании промтов для RAG-приложений важно учитывать следующие принципы:

- *Будьте краткими и точными.* Четко определите задачу, которую должна выполнить модель, и указывайте только необходимую информацию. Например, запрос "Пожалуйста, проанализируйте данный контекст и ответьте на вопрос, учитывая всю релевантную информацию и детали" менее ясен, чем "На основе приведенного контекста ответьте на следующий вопрос: [конкретный вопрос]".
- *Формулируйте одну задачу за раз.* Разбивайте сложные задачи на более мелкие и управляемые части. Вместо запроса "Суммируйте основные моменты, определите ключевые сущности и ответьте на вопрос", лучше разделить его на несколько промтов: Суммируйте основные моменты следующего контекста: [контекст]. Определите ключевые сущности, упомянутые в этом резюме: [резюме из предыдущего промта]. Используя контекст и выявленные сущности, ответьте на следующий вопрос: [конкретный вопрос]
- *Преобразовывайте генеративные задачи в задачи классификации.* Когда возможно, перефразируйте открытые генеративные запросы в формат классификации с ограниченным числом вариантов. Например, вместо "Какую эмоциональную окраску имеет контекст?", лучше сказать: "Классифицируйте тональность текста как положительную, отрицательную или нейтральную".
- *Используйте примеры для повышения качества ответа.* Это продолжает концепцию single-shot и few-shot prompting. Вместо "Ответьте на следующий вопрос на основе приведенного контекста", лучше сказать: "Используя примеры ниже как ориентир, ответьте на следующий вопрос на основе контекста", затем добавить примеры. Пример 1: [вопрос] [контекст] [ответ]. Пример 2: [вопрос] [контекст] [ответ]. Текущий вопрос: [вопрос]. Контекст: [контекст].
- *Начинайте с простых промтов и постепенно уточняйте их.* Сначала используйте простые формулировки, а затем постепенно добавляйте элементы для улучшения результата. Например, "Суммируйте основные моменты статьи, определите ключевые сущности и ответьте на вопрос" будет менее эффективным, чем постепенное усложнение промтов: Итерация 1: Суммируйте основные моменты следующей статьи: [текст статьи]. Итерация 2: Суммируйте основные моменты и определите ключевые сущности в статье: [текст статьи]. Итерация 3: На основе резюме и сущностей ответьте на следующий вопрос: [вопрос].
- *Размещайте инструкции в начале промта.* Четко формулируйте инструкции в начале, используя разделители (например, ###). "Используйте приведенный выше контекст для ответа на следующий вопрос: [вопрос]" будет менее эффективным, чем: Инструкция:

Используя приведенный ниже контекст, ответьте на вопрос кратко. ### Контекст: [контекст]. Вопрос: [вопрос].

Хотя основные принципы проектирования промтов дают надежную основу, важно помнить, что разные языковые модели могут требовать адаптации для достижения оптимальных результатов.

Далее мы обсудим, как подстраивать промты под разные LLM.

Адаптация промтов для разных LLM

По мере развития ИИ пользователи все реже ограничиваются только OpenAI для своих задач по обработке языка. Другие компании, такие как Anthropic с их моделями Claude, приобрели популярность благодаря поддержке длинных контекстов. Google также разрабатывает и выпускает мощные языковые модели, а сообщество open-source догоняет лидеров, предлагая альтернативы, такие как Llama. Однако промты не всегда можно просто перенести с одной модели на другую. Каждая LLM может требовать своего подхода и использовать уникальные техники обработки промтов.

Например, Claude-3 предпочитает XML-кодирование, а Llama3 использует специальные теги, такие как SYS и INST, для разметки частей промта. Пример промта для моделей Llama:

<SYS> Вы — ИИ-ассистент, созданный для предоставления полезных и информативных ответов на вопросы пользователей. </SYS>

<INST> Проанализируйте вопрос пользователя и дайте четкий, лаконичный ответ, используя свою базу знаний. Если вопрос недостаточно ясен, запросите уточнение. Вопрос пользователя: "Каковы основные преимущества использования возобновляемых источников энергии по сравнению с ископаемым топливом?" </INST>

В этом примере:

- Тег <SYS> определяет роль модели как ассистента, который дает полезные ответы.
- Тег <INST> содержит инструкцию, указывая, как обрабатывать вопрос пользователя.

Тег SYS — это сокращение от system message, а INST — от instructions.

При проектировании промтов для RAG-приложений важно учитывать особенности и лучшие практики выбранной модели, чтобы добиться оптимальных результатов. У всех популярных LLM есть документация по промтам, где подробно объясняются их особенности.

Теперь переходим к практической части: реализуем изученные концепции в коде!

Лаборатория кода 13.1 — Пользовательский шаблон промта

PromptTemplate — это класс в LangChain, который используется для управления и применения промтов. Как и в большинстве шаблонов, в нем есть текст с переменными, которые выступают в роли входных данных. Использование PromptTemplate позволяет эффективно интегрировать промты в LangChain, обеспечивая совместимость с другими компонентами экосистемы.

Этот код основан на лабораторной работе 8.3 из главы 8 и находится в репозитории GitHub.

Вот промт, который мы использовали чаще всего:

```
prompt = hub.pull("jclemens24/rag-prompt")
```

Вывод этого промта на экран выглядит следующим образом:

```
You are an assistant for question-answering tasks. Use the following
pieces of retrieved context to answer the question. If you don't know
the answer, just say that you don't know.
Question: {question}
Context: {context}
Answer:
```

Этот шаблон сохраняется в объекте *PromptTemplate*, который можно вывести на экран. Если это сделать, результат будет выглядеть примерно так:

```
ChatPromptTemplate(input_variables=['context',
'question'], metadata={'lc_hub_owner': 'jclemens24',
```

'lc_hub_repo': 'rag-prompt', 'lc_hub_commit_hash':
'1a1f3ccb9a5a92363310e3b130843dfb2540239366ebe712ddd94982acc06734'},
messages=[HumanMessagePromptTemplate(prompt=PromptTemplate(input_
variables=['context', 'question'], template="You are an assistant
for question-answering tasks. Use the following pieces of retrieved
context to answer the question. If you don't know the answer, just
say that you don't know.\nQuestion: {question} \nContext: {context} \
nAnswer:"))])

Итак, как мы видим, объект PromptTemplate включает в себя не только текст и переменные. Прежде всего, можно сказать, что это конкретная версия объекта PromptTemplate, называемая ChatPromptTemplate, что указывает на то, что он предназначен главным образом для чат-сценариев. Входные переменные — context и question, которые затем используются в строке шаблона. Вскоре мы создадим собственный шаблон, но данный шаблон загружен из LangChain hub. Здесь отображаются метаданные, содержащие информацию о владельце, репозитории и хеше коммита,[1] связанном с хабом.

Давайте начнем лабораторную работу с замены этого шаблона промта на наш собственный.

Мы уже импортировали LangChain ранее:

from langchain_core.prompts import PromptTemplate

Дополнительные импорты не нужны Теперь заменим старый промт…

prompt = hub.pull("jclemens24/rag-prompt")

… на новый…

```
prompt = PromptTemplate.from_template(
"""
You are an environment expert assisting others in
understanding what large companies are doing to
improve the environment. Use the following pieces
of retrieved context with information about what
a particular company is doing to improve the
environment to answer the question.
If you don't know the answer, just say that you don't know.
Question: {question}
Context: {context}
Answer:"""
)
```

Здесь мы модифицировали шаблон, сделав его более подходящим под тематику нашего набора данных (экологический отчет Google). Мы используем шаблон персонализированных ролей (Personas Prompt Design), чтобы задать LLM определенную роль, что помогает настроить генерацию в соответствии с нашей тематикой.

Шаблоны промтов принимают в качестве входных данных словарь, где каждый ключ представляет переменную, которую необходимо подставить в шаблон. Результатом работы объекта PromptTemplate является переменная PromptValue, которую можно передавать в LLM или экземпляр ChatModel напрямую либо использовать в качестве шага в конвейере LCEL.

Выведите объект промта с помощью следующего кода:

print(prompt)

Результат будет следующим:

---

[1] *Commit* (коммит) — это сохраненное изменение в системе управления версиями, такой как Git. Он фиксирует состояние файлов в определенный момент времени, позволяя отслеживать историю изменений, возвращаться к предыдущим версиям и работать в команде. Каждый коммит имеет уникальный хеш (commit hash), который идентифицирует его в репозитории.

```
input_variables=['context', 'question'] template="\n You are
an environment expert assisting others in \n understanding what
large companies are doing to \n improve the environment. Use
the following pieces \n of retrieved context with information
about what \n a particular company is doing to improve the
\n environment to answer the question. \n \n If you don't
know the answer, just say that you don't know.\n \n Question:
{question} \n Context: {context} \n \n Answer:"
```

Мы видим, что входные переменные были автоматически определены, без необходимости указывать их явно:

```
input_variables=['context', 'question']
```

Можно вывести только текст шаблона, используя следующую строку:

```
print(prompt.template)
```

Это представит более читаемую версию предыдущего вывода, содержащую только сам текст промта. Обратите внимание, что сразу под этим уже находится пользовательский шаблон промта, ориентированный на определение оценки релевантности:

```
# Relevance check prompt
relevance_prompt_template = PromptTemplate.from_template(
    """
    Given the following question and retrieved context, determine if the context is relevant to the question.
    Provide a score from 1 to 5, where 1 is not at all relevant and 5 is highly relevant.
    Return ONLY the numeric score, without any additional text or explanation.

    Question: {question}
    Retrieved Context: {retrieved_context}

    Relevance Score:"""
)
```

Если запустить оставшуюся часть кода, можно увидеть, как этот шаблон влияет на конечный результат RAG-приложения. Этот шаблон промта относится к типу String prompt template, что означает, что он создается в виде обычной строки, содержащей текст промта с заполнителями для динамического контента (например, {question} и {retrieved_context}). Также можно использовать объект ChatPromptTemplate, который предназначен для форматирования списка сообщений и сам по себе состоит из списка шаблонов.

Шаблоны промтов играют важную роль в оптимизации производительности RAG-систем. В оставшихся лабораторных работах этой главы мы будем использовать их в качестве основного инструмента.

Следующая лабораторная работа будет посвящена форматированию промтов.

Лаборатория кода 13.2 — Варианты промтов

Файл с кодом.

При проектировании промтов существует ряд ключевых концепций, которые следует учитывать. К ним относятся итерация, суммаризация, трансформация и расширение. Каждая из этих концепций имеет различные области применения, и их можно комбинировать. Понимание этих принципов поможет улучшить RAG-приложения, а также грамотно разрабатывать промты.

В этой лабораторной работе мы рассмотрим различные подходы к промтингу на практическом примере: разработке промтов для маркетингового отдела компании. Начнем с итерации.

Итерация

Итерация означает постепенное улучшение промта для получения лучших результатов. Редко бывает так, что первый вариант промта оказывается идеальным. Этот раздел посвящен базовым техникам и концепциям, которые помогут быстро адаптировать и оптимизировать промты для RAG-приложений.

Изменение тона

Ваш начальник только что позвонил. Он сообщил, что маркетинговый отдел хочет использовать результаты RAG-приложения в маркетинговых материалах, но оформление должно соответствовать формату маркетингового информационного листа.

Без проблем! Мы можем настроить промт так, чтобы он выдавал нужный формат. Для этого добавим второй промт после первого.

```
# Prompt for marketing speak
prompt2 = PromptTemplate.from_template(
  """
  Your task is to help a marketing team create a
  description for the website about the environmental
  initiatives our clients are promoting.

  Write a marketing description based on the information
  provided in the context delimited by triple backticks.

  If you don't know the answer, just say that you don't know.

  Question: {question}
  Context: ```{context}```

  Answer:
  """
)
```

Затем необходимо изменить промт в цепочке rag_chain_from_docs на prompt2. Посмотрите чуть ниже строки с RunnablePassthrough().

```
rag_chain_from_docs = (
…
          "answer": (
            RunnablePassthrough()
            | prompt2 # <- изменить здесь
            | llm
            | str_output_parser
          )
      }
  )
  | RunnablePassthrough().assign(final_answer=conditional_answer)
)
```

Затем перезапустите код, начиная с prompt2, чтобы получить следующий результат:

```
Google is at the forefront of environmental sustainability, leveraging
its technological prowess to drive impactful initiatives across
various domains. Here are some of the key environmental initiatives
that Google is promoting:
Empowering Individuals: Google aims to help individuals make more
sustainable choices through its products. In 2022, Google reached its
goal of assisting 1 billion people in making eco-friendly decisions.
This was achieved through features like eco-friendly routing in Google
Maps, energy-efficient settings in Google Nest thermostats, and carbon
emissions information in Google Flights. By 2030, Google aspires to
help reduce 1 gigaton of carbon equivalent emissions annually.
…[TRUNCATED FOR BREVITY]…
By organizing information about the planet and making it actionable
through technology, Google is helping to create a more sustainable
```

future. The company's efforts span from individual empowerment to global partnerships, all aimed at reducing environmental impact and fostering a healthier planet.

Можно заметить, что вывод действительно больше соответствует маркетинговому стилю. Возможно, именно этого и добивается маркетинговая команда.

Однако вы только что вспомнили, что начальник также упомянул, что этот текст будет размещен в небольшом квадрате на сайте, который вмещает не более 50 слов!

Сокращение длины

Для prompt3 нам нужно просто добавить небольшую правку: *Используй не более 50 слов*:

```
# Prompt for shorter output
prompt3 = PromptTemplate.from_template(
    """
    Your task is to help a marketing team create a
    description for the website about the environmental
    initiatives our clients are promoting.

    Write a marketing description based on the information
    provided in the context delimited by triple backticks.

    If you don't know the answer, just say that you don't know.

    Use at most 50 words.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Обновите промт в цепочке rag_chain_from_docs, заменив на prompt3. Запустите оставшийся код, и получите следующий результат:

```
Google's environmental initiatives include promoting electric
vehicles, sustainable agriculture, net-zero carbon operations, water
stewardship, and a circular economy. They aim to help individuals and
partners reduce carbon emissions, optimize resource use, and support
climate action through technology and data-driven solutions.
```

Маркетинговая команда в восторге от вашей работы!

Проходит время... Через месяц принимается решение изменить акцент. Вместо того чтобы фокусироваться на всех экологических инициативах клиента, лучше сосредоточиться на технологических аспектах.

Изменение фокуса

Теперь нам нужно направить LLM на аспекты, связанные с технологиями. Новый текст предназначен для брошюры, поэтому ограничение по количеству слов можно убрать. Создадим prompt4, который изменит фокус генерации:

```
# Prompt for focus on area
prompt4 = PromptTemplate.from_template(
    """
    Your task is to help a marketing team create a
    description for the website about the environmental
    initiatives our clients are promoting.
```

```
    Write a marketing description based on the information
    provided in the context delimited by triple backticks.

    The description is intended for a technology audience,
    so this should focus on only the aspects of the company's
    efforts that relate to using technology.

    If you don't know the answer, just say that you don't know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Не забудьте обновить промт в цепочке на prompt4, а затем запустить оставшуюся часть кода, чтобы получить результат:

```
Google is at the forefront of leveraging technology to drive
environmental sustainability. Here are some of the key initiatives
that highlight their commitment to using technology for a greener
future:
Eco-Friendly Product Features:
Google Maps: Introduced eco-friendly routing, which has helped prevent
over 1.2 million metric tons of carbon emissions since its launch.
Google Nest: Energy efficiency features in Google Nest thermostats
help users reduce their energy consumption.
Code lab 13.2 – Prompting options 269
Google Flights: Provides carbon emissions information to help
travelers make more sustainable choices.
…TRUNCATED FOR BREVITY…
Sustainability-Focused Accelerators:
Google for Startups Accelerator: Supports early-stage innovations
aimed at tackling sustainability challenges, fostering the growth of
technologies that can positively impact the planet.
Google's comprehensive approach to environmental sustainability
leverages their technological expertise to create significant positive
impacts. By integrating sustainability features into their products,
optimizing their operations, and collaborating with partners, Google
is driving forward a more sustainable future.
```

Мы сократили вывод здесь, но если посмотреть результат в коде, он четко ориентирован на технологические аспекты. Маркетинговая команда впечатлена!

Этот пример выглядел забавно, но он вполне реалистичен для разработки подобных систем. В реальных условиях итераций будет гораздо больше, но поэтапный подход к проектированию промтов помогает добиться наилучших результатов в RAG-приложениях, не менее чем другие части системы.

Теперь поговорим о том, как преобразовать большой объем данных в компактный формат — то есть о суммаризации.

Суммаризация

Компании могут обрабатывать большие объемы внутренних данных и преобразовывать их в краткую и удобную форму, что помогает повысить продуктивность. Это особенно полезно в сферах, которые зависят от информации или работают с быстро меняющимися данными. Мы уже видели, как задать ограничение по количеству слов (в prompt3). Сейчас мы сфокусируемся на суммаризации, а не на генерации экспертного мнения или маркетингового текста. Изменим код промат:

```
# Prompt for shorter output with a summary
prompt5 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    Summarize the retrieved context below, delimited by
    triple backticks, in at most 30 words.

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Обновите цепочку до prompt5 и запустите код. Результат:

```
Google's environmental initiatives include achieving net-zero carbon,
promoting water stewardship, supporting a circular economy, and
leveraging technology to help partners reduce emissions.
```

Отлично! Кратко и по делу. Только факты!

Суммаризация с фокусом

Следующий пример покажет, как можно направить LLM на определенный аспект при суммаризации. Для prompt6 мы оставим большую часть предыдущего промта, но добавим акцент на экологические свойства продуктов:

```
# Prompt for shorter output with a summary and a focus
prompt6 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    Summarize the retrieved context below, delimited by
    triple backticks, in at most 30 words, and focusing
    on any aspects that mention the eco-friendliness of
    their products.

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Обновите цепочку до prompt6 и запустите код. Результат:

```
Google's environmental initiatives include eco-friendly routing in
Google Maps, energy-efficient Google Nest thermostats, and carbon
emissions information in Google Flights.
```

Кратко и по существу!

Если сравнить с более развернутыми описаниями, видно, что LLM выделил только продукты, представленные в PDF. Результат получился неплохим, но иногда, даже если запрос сфокусирован, LLM может включать ненужную информацию. Чтобы этого избежать, используем метод извлечения данных вместо суммаризации.

Извлечение вместо суммаризации

Если при суммаризации LLM включает лишнюю информацию, попробуйте использовать извлечение (extract), а не суммаризацию (summarize). Это кажется незначительным изменением, но оно может сильно повлиять на результат. Извлечение заставляет модель искать конкретные данные, а не обобщать весь текст. LLM понимает эту разницу, и этот прием может помочь вам избежать проблем с суммаризацией. Создадим prompt7, учитывая этот нюанс:

```
# Prompt for shorter output using extract and focus
prompt7 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    From the retrieved context below, delimited by
    triple backticks, extract the information focusing
    on any aspects that mention the eco-friendliness of
    their products. Limit to 30 words.

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Обновите цепочку до prompt7 и запустите код. Результат:

```
Google's environmental initiatives include eco-friendly routing in
Google Maps, energy efficiency features in Google Nest thermostats,
and carbon emissions information in Google Flights to help users make
sustainable choices.
```

Ответ немного отличается от prompt6, но уже в предыдущем случае он был хорошо сфокусирован. Если LLM добавляет ненужные данные в суммаризацию, попробуйте этот метод для улучшения результатов.

Итерация и суммаризация — это не единственные концепции, которые помогут вам оптимизировать промты. Далее мы рассмотрим, как использовать RAG-приложение для извлечения выводов из имеющихся данных.

Выведение (Inference)

В основе выведения (inference) лежит запрос к модели проанализировать данные и предоставить дополнительную информацию. Это может включать извлечение меток, имен, тем или даже определение тональности текста. Такие возможности значительно расширяют применение RAG, поскольку позволяют автоматизировать задачи, ранее доступные только человеку.

Начнем с простого анализа тональности (sentiment analysis) в формате положительный/отрицательный (Boolean-style analysis):

```
# Sentiment analysis
prompt8 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
```

```
    company is doing to improve the environment.

    From the retrieved context below, delimited by
    triple backticks, extract the information focusing
    on any aspects that mention the eco-friendliness of
    their products. Limit to 30 words.

    After this summary, determine what the sentiment
    of context is, providing your answer as a single word,
    either "positive" or "negative".

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

В этом коде мы дополняем предыдущий промт: теперь LLM не только суммаризирует текст, но и анализирует его тональность. В данном случае модель определяет тональность как положительную:

```
Google is enhancing eco-friendliness through features like ecofriendly
routing in Maps, energy-efficient Nest thermostats,
and carbon emissions data in Flights, aiming to reduce emissions
significantly.
Sentiment: positive
```

Извлечение ключевых данных

Еще одна распространенная аналитическая задача — извлечение конкретной информации из контекста. Допустим, вам поручили выявить, какие именно продукты клиент (Google) упоминает в документации в контексте экологических инициатив. У компании Google много продуктов, но в этом документе упоминаются только некоторые. Как быстро их выделить? Попробуем сделать это с помощью промта:

```
# Product name extraction
prompt9 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    From the retrieved context below, delimited by
    triple backticks, extract the information focusing
    on any aspects that mention the eco-friendliness of
    their products. Limit to 30 words.

    After this summary, determine any specific products
    that are identified in the context below, delimited
    by triple backticks.  Indicate that this is a list
    of related products with the words 'Related products: '
    and then list those product names after those words.

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
```

```
    Context: ```{context}```

    Answer:
    """
)
```

В этом коде мы продолжаем развивать предыдущие промты, но вместо анализа тональности теперь запрашиваем список продуктов, фигурирующих в тексте. Модель GPT-4o-mini успешно выполняет задачу, выводя конкретные названия продуктов:

```
Google is enhancing eco-friendliness through products like ecofriendly
routing in Google Maps, energy efficiency features in Google
Nest thermostats, and carbon emissions information in Google Flights.
Related products: Google Maps, Google Nest thermostats, Google Flights
```

Выведение тем

Иногда требуется понять, о чем текст в целом, а не просто извлечь конкретные данные. Это можно рассматривать как экстремальный случай суммаризации: мы берем тысячи слов и преобразуем их в короткий список тем.

```
# Topic extraction
prompt10 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    From the retrieved context below, delimited by
    triple backticks, extract the information focusing
    on any aspects that mention the eco-friendliness of
    their products. Limit to 30 words.

    After this summary, determine eight topics that are
    being discussed in the context below delimited
    by triple backticks.
    Make each item one or two words long.
    Indicate that this is a list of related topics
    with the words 'Related topics: '
    and then list those topics after those words.

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Здесь мы используем тот же подход, что и в предыдущих примерах, но теперь запрашиваем список как минимум из восьми тем, связанных с текстом. GPT-4o-mini снова справляется с задачей, выдавая список релевантных тем, охваченных в тексте:

```
Google is enhancing eco-friendliness through products like ecofriendly
routing in Google Maps, energy-efficient Google Nest
thermostats, and carbon emissions information in Google Flights.
Related topics:
1. Electric vehicles
2. Net-zero carbon
```

3. Water stewardship
4. Circular economy
5. Supplier engagement
6. Climate resilience
7. Renewable energy
8. AI for sustainability

Мы рассмотрели итерацию, суммаризацию и выведение — все эти техники существенно улучшают работу с промтами. Теперь перейдем к еще одной важной концепции — трансформации данных.

Трансформация

Трансформация — это процесс преобразования данных в другой формат или состояние. Один из наиболее распространенных примеров — перевод текста. Однако есть и другие формы трансформации, например: преобразование данных в JSON или HTML, проверка орфографии и грамматики.

Начнем с перевода.

Языковая трансформация (перевод)

Маркетинговый отдел снова выходит на связь. Работа идет отлично, но теперь компания выходит на международный рынок! Мы выбрали испанский и французский как первые целевые языки. Кроме того, новый инвестор — фанат всего, что связано с пиратами, так что да, нам придется добавить пиратский диалект! Хотя это формально трансформация, чаще используется термин *перевод*.

Приступим:

```
# Language transformation
prompt11 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    From the retrieved context below, delimited by
    triple backticks, extract the information focusing
    on any aspects that mention the eco-friendliness of
    their products. Limit to 30 words.

    Translate the summary into three additional languages,
    Spanish, French, and English Pirate:
    labeling each language with a format like this:

    English: [summary]
    Spanish: [summary]
    French: [summary]
    English pirate: [summary]

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

В этом коде мы используем предыдущий промт, но теперь LLM генерирует четыре версии краткого описания: английскую, испанскую, французскую, английскую (пиратский диалект). Очевидно, пиратский диалект — самый забавный вариант!

English: Google enhances eco-friendliness through features like ecofriendly routing in Maps, energy-efficient Nest thermostats, and carbon emissions info in Flights, helping reduce carbon emissions significantly.
Spanish: Google mejora la eco-amigabilidad con funciones como rutas ecológicas en Maps, termostatos Nest eficientes en energía e información de emisiones de carbono en Flights, ayudando a reducir significativamente las emisiones de carbono.
French: Google améliore l'éco-responsabilité avec des fonctionnalités telles que les itinéraires écologiques dans Maps, les thermostats Nest économes en énergie et les informations sur les émissions de carbone dans Flights, aidant à réduire significativement les émissions de carbone.
English pirate: Google be makin' things greener with eco-routes in Maps, energy-savin' Nest thermostats, and carbon info in Flights, helpin' to cut down on carbon emissions mightily.

Перевод — одно из популярных применений RAG, но есть и другие полезные трансформации. Рассмотрим изменение тона текста.

Тональная трансформация

До сих пор мы создавали суммаризации и маркетинговые тексты, но теперь их нужно адаптировать под другие каналы, например e-mail. Для этого нам понадобится трансформация тона.

```
# Tone transformation
prompt12 = PromptTemplate.from_template(
    """
    Your task is to generate a short summary of what a
    company is doing to improve the environment.

    From the retrieved context below, delimited by
    triple backticks, extract the information focusing
    on any aspects that mention the eco-friendliness of
    their products. Limit to 30 words.

    After providing the summary, translate the summary
    into an email format with a more friendly and
    casual tone.

    If you don't know the answer, just say that you don't
    know.

    Question: {question}
    Context: ```{context}```

    Answer:
    """
)
```

Здесь мы продолжаем суммаризацию, но затем просим LLM переписать текст в виде e-mail в неформальном стиле. Получаем:

```
Google is enhancing eco-friendliness through features like ecofriendly
routing in Google Maps, energy-efficient Google Nest
thermostats, and carbon emissions data in Google Flights.
Email Format:
Subject: Exciting Eco-Friendly Features from Google!
Hi [Recipient's Name],
I hope you're doing well! I wanted to share some cool updates from
```

Google about their efforts to help the environment. They've introduced some awesome features like eco-friendly routing in Google Maps, energy-efficient Google Nest thermostats, and even carbon emissions data in Google Flights. It's great to see such big steps towards a greener future!
Best, [Your Name]

Как мы видим на этих примерах, LLM можно использовать множеством способов для улучшения RAG-приложений.

Другие полезные трансформации, которые не так актуальны в данном примере, но часто применяются в других задачах: преобразование кода из одного формата в другой, проверка орфографии и грамматики.

Мы рассмотрели итерацию, суммаризацию, выведение и трансформацию. Осталась еще одна концепция, которую мы обсудим в завершение этой лабораторной работы — расширение (expansion).

Расширение (Expansion)

Цель расширения во многом противоположна целям, рассмотренным ранее, таким как суммаризация. При суммаризации мы сжимаем большой объем данных, сохраняя их смысл. При расширении мы, наоборот, берем небольшое количество информации и дополняем его, увеличивая объем. Рассмотрим пример расширения короткого текста.

Нам нужно взять экологические концепции и продвигать их ценность среди инвесторов. В следующем промте мы используем краткое резюме, предполагая, что это весь доступный контент. Мы попросим LLM расширить этот текст, акцентируя внимание на привлекательности для инвесторов. Давайте посмотрим, что у него получится:

```
# Expand on a short text
prompt13 = PromptTemplate.from_template(
  """
  Your task is to generate a short summary of what a
  company is doing to improve the environment.

  From the retrieved context below, delimited by
  triple backticks, extract the information focusing
  on any aspects that mention the eco-friendliness
  of their products. Limit to 30 words.

  After providing the summary, provide a broader
  description of what the company is doing to
  improve the environment and explain how this
  can be useful to investors in that company.

  For this broader description, do not use any of
  the data provided in the context below, using
  only the summary you have generated as the basis
  for this description.

  If you don't know the answer, just say that you don't
  know.

  Question: {question}
  Context: ```{context}```

  Answer:
  """
)
```

В реальной ситуации исходный контекст часто отсутствует. Поэтому мы имитируем этот сценарий, ограничивая источник информации только данным резюме. LLM справляется с задачей, создавая развернутое описание экологических инициатив, адаптированное для потенциальных инвесторов:

Summary: Google offers eco-friendly routing in Google Maps, energyefficient
Google Nest thermostats, and carbon emissions information in
Google Flights to help users make sustainable choices.
Broader Description: Google is actively enhancing the eco-friendliness
of its products by integrating features that promote sustainability.
For instance, Google Maps now includes eco-friendly routing options,
Google Nest thermostats are designed for energy efficiency, and Google
Flights provides carbon emissions information. These initiatives not
only help users make more environmentally conscious decisions but
also demonstrate Google's commitment to reducing its carbon footprint.
For investors, this focus on sustainability can be a significant
value proposition, as it aligns with the growing consumer demand for
eco-friendly products and can lead to long-term cost savings and
regulatory advantages. Additionally, it positions Google as a leader
in environmental responsibility, potentially enhancing its brand
reputation and market share.

Это лишь один пример того, как можно использовать расширение данных. Подумайте, в каких ситуациях ваш RAG-проект может извлекать пользу из этого метода.

Саммари

В этой главе мы рассмотрели ключевую роль инженерии промтов в повышении эффективности RAG-систем. Грамотное проектирование и оптимизация промтов позволяют улучшить процесс поиска информации и, как следствие, качество сгенерированного текста. Мы изучили различные техники проектирования промтов, включая:

- Shot design — проектирование примеров (shot-дизайн)
- Chain-of-thought prompting — пошаговое проектирование промта (цепочка рассуждений)
- Personas — персонализация (персонажи)
- Knowledge augmentation — дополнение знаний

Мы рассмотрели основные принципы проектирования промтов, такие как четкость, структурированность, итеративное улучшение и адаптация к разным LLM. В ходе лабораторной работы мы создали пользовательские шаблоны промтов с помощью PromptTemplate в LangChain и применили различные стратегии для улучшения RAG-систем:

- Итерация (Iterating) — постепенное улучшение промта.
- Суммаризация (Summarization) — сжатие информации.
- Выведение (Inference) — извлечение дополнительной информации.
- Трансформация (Transformation) — преобразование формата, тона или языка.
- Расширение (Expansion) — дополнение краткого текста.

Также мы изучили параметры управления генерацией (temperature, top-p, seed) для баланса случайности и детерминированности в LLM.

Применяя все рассмотренные техники, можно значительно повысить эффективность RAG-приложений, сделав их более точными, адаптивными и качественными.

Поскольку область инженерии промтов продолжает развиваться, важно следить за новыми техниками, чтобы оставаться в курсе лучших практик.

В следующей и заключительной главе мы разберем продвинутые методы, которые помогут вывести вашу RAG-систему на новый уровень!