**ОСНОВЫ ПРОМТ-ИНЖИНИРИНГА**

**Промт** — текстовый запрос или инструкция, которую пользователь передаёт языковой модели, чтобы получить от неё определённый ответ или действие. Вы также можете встретить написание «промпт». Оба варианта образованы от англ. prompt — «запрос», «подсказка».

**Промтинг** — процесс создания и использования промтов для взаимодействия с языковыми моделями.

**Промт-инжиниринг** — техника составления промтов, которые позволят получать от модели стабильные и контролируемые результаты. Поскольку поведение модели не всегда предсказуемо, важно задавать не только тему, но и стиль, формат, ограничения и цели ответа.

Промт-инжинирингом вы и будете заниматься, проходя эти уроки.

Писать простые запросы можно обычным текстом. Но лучше всего делать это на языке, который лучше понимает нейросеть. Есть несколько форматов ввода запросов.

**ФОРМАТЫ ВВОДА ЗАПРОСОВ**

**Самой простой**

Для простых запросов, особенно если задача небольшая и однозначная:

Разделять запрос на части с помощью заголовков.

Например:

Задача:

Отсортируйте список чисел по возрастанию и напечатайте результат.

Подсказка:

Используйте встроенную функцию sorted() или метод .sort() для списков.

Прекод:

numbers = [42, 17, 68, 3, 99]

# < напишите код здесь >

Ожидаемый результат:

[3, 17, 42, 68, 99]

Для сложных задач, лучше использовать **Markdown или XML**.

Преимущества Markdown и XML:

Чёткая структура

* Разделы, списки, таблицы — легко отделить условия, данные и ожидаемый результат.
* Например, можно выделить Input, Output, Constraints и т.д.

Уменьшает неоднозначность

* LLM проще понять, что именно делать, когда всё структурировано явно.
* Можно использовать кодовые блоки python или <code> в XML для прекода.

Легче масштабировать

* Когда задач много или они сложные (ML, SQL, большие датафреймы), Markdown/XML помогают моделям корректно интерпретировать данные.

**Markdown**

Самый простой и удобный формат для промтов — Markdown. Он позволяет оформлять заголовки, списки, выделения, ссылки и код с помощью специальных символов, но без сложной разметки. Это помогает легко структурировать запросы и делать их понятными для модели.

Вот пример самого простого Markdown-промта.

Ты — \*\*backend-разработчик\*\*. Предложи три способа валидации email-адреса в Python.

С помощью заголовков, списков и фрагментов кода можно создать более сложный структурированный промт. Например:

# Проведение ревью кода

## Синтаксис

…описание правил кодстайла…

## Принципы именования переменных

…описание принципов именования переменных…

## Примеры кода

### Плохо

…пример плохого кода…

### Хорошо

…пример хорошего кода…

**XML**

Когда промты включают в себя несколько компонентов — например, контекст, инструкции и примеры — можно использовать более строгое форматирование запросов — XML. Это текстовый формат для хранения и передачи структурированных данных, где каждая часть информации обёрнута в понятные теги вроде <имя> или <описание>. Он позволяет ИИ-моделям более точно анализировать запросы и выдавать более качественные результаты.

XML-разметку для промтов можно использовать в нескольких случаях:

Важно чётко разделять части запроса. Теги вроде <роль>, <задача>, <пример> делают структуру запроса более явной.

Нужна высокая точность. Уменьшается риск того, что модель перепутает инструкцию с комментарием или пользовательским вводом.

Планируется изменять или переиспользовать запрос. XML-разметку легко редактировать, поэтому можно заменить одну часть, не переписывая весь промт.

Вот пример XML-промта.

<запрос>

<роль>backend-разработчик</роль>

<задача>предложи три способа валидации email-адреса в python</задача>

<результат>верни примеры кода</результат>

</запрос>

Если вам нужен простой и удобный формат для промтов, то будет достаточно Markdown-разметки. Если в вашем промте много частей или вы хотите задать чёткую и однозначную структуру, то XML-разметка может дать более качественные результаты. Всегда имеет смысл попробовать оба варианта.

Помимо разметки, на понятность и эффективность промта может влиять язык общения с моделью. Современные модели хорошо понимают многие языки, включая русский, и могут решать технические задачи без необходимости переходить на английский.

В большинстве случаев стоит использовать тот язык, на котором вам проще и быстрее формулировать мысли. Но запросы на английском могут дать значительное преимущество в некоторых видах задач. Например, если вы переводите или редактируете тексты на английском с помощью ИИ или работаете с терминологией, у которой нет точных аналогов.

Языки можно смешивать — например, формулировать инструкции на русском, а код или названия переменных оставлять на английском. Главное, чтобы модель могла корректно интерпретировать промт. Если вы видите, что ответы становятся странными или неполными, попробуйте изменить язык или уточнить формулировки.

**КАК ВЫГЛЯДИТ ХОРОШИЙ ПРОМТ**

Промты бывают разными: короткими и длинными, формальными и неформальными, с примерами и без. Но что делает их действительно эффективными? На что обращать внимание при формулировке? Рассказываем, какие принципы делают запрос понятным и результативным.

**Принципы хорошего промта**

**Целенаправленность** — промт должен отражать конкретную цель. Недостаточно просто указать направление — важно описать, что именно нужно получить и в каком виде. Это особенно важно при генерации кода, ответов, документации или анализа.

|  |  |  |
| --- | --- | --- |
| **Оценка** | **Промт** | **Комментарий** |
| - | Объясни мне этот код | Неясно, какой код, какие части объяснять, на каком уровне должно быть объяснение и для кого. Ответ может быть слишком общим |
| + | Объясни простыми словами для начинающего разработчика, что делает функция validateEmail. Не углубляйся в объяснение регулярных выражений, сосредоточься на общей логике | Чётко указано, какую функцию объяснить, какой должен быть стиль ответа и уровень детализации. Ответ будет полезным |

**Недвусмысленность** — если формулировка допускает несколько трактовок, модель может выбрать не ту, что вы ожидали. В отличие от человека, она не задаст уточняющих вопросов. Поэтому любые двусмысленные, расплывчатые или обтекаемые формулировки лучше сразу заменить на точные.

|  |  |  |
| --- | --- | --- |
| **Оценка** | **Промт** | **Комментарий** |
| - | Добавь логирование для функции createInvoice | Неясно, что требуется логировать, каким образом и какие есть инструменты |
| + | Добавь в функцию createInvoice логирование входных параметров и ошибок внутри неё. Используй console.error для ошибок и console.log для параметров | Чётко указано, что и где логировать, какими инструментами. Такой промт снижает риск ошибок |

**Структурированность** — даже короткий промт должен быть логически организован. Это особенно важно при генерации кода, тестов или документации.

|  |  |  |
| --- | --- | --- |
| **Оценка** | **Промт** | **Комментарий** |
| - | Напиши мне тест на функцию calculateTax. Проверь все случаи | Не указано, для какой функции, какие кейсы и библиотеку использовать. Результат, весьма вероятно, будет далёк от желаемого |
| + | Напиши unit-тест на функцию calculateTax с использованием Jest. Проверь: amount = 1000, taxRate = 0.2, результат — 200 | Чёткая цель, конкретная библиотека, значения параметров и ожидаемый результат. Задача понятна, результат предсказуем |

**СТРУКТУРА ПРОМТА**

Чтобы получить точный и полезный ответ от модели, важно задать контекст, описать задачу и определить желаемый образ результата.

Формулу хорошего промта можно описать так.

Хороший промт = контекст + роль + задача + формат

Рассмотрим каждый элемент подробнее.

**Контекст** — это информация, которая помогает модели понять, с чем она работает: что уже есть, в какой системе или проекте мы находимся, какие ограничения или правила важны.

|  |  |  |
| --- | --- | --- |
| **Оценка** | **Промт** | **Комментарий** |
| - | Оптимизируй функцию fetchUserData для работы с сетью | Можно придумать много оптимизаций для работы с сетью. Вероятно, модель пойдёт не тем путём, который вы ожидаете |
| + | У нас есть функция fetchUserData, которая делает три последовательных HTTP-запроса. Это замедляет работу. Оптимизируй код так, чтобы запросы выполнялись параллельно. Используем axios | Указан код, проблема, что подразумевается под оптимизацией, выбран инструмент. Модель получит нужную информацию и выдаст валидный ответ |

**Роль модели** - важная часть контекста: кто она, в каком стиле отвечает, какой уровень сложности выбирает. Можно выдать ИИ любую роль.

Ты опытный преподаватель, который объясняет программирование простыми словами для студентов первого курса.

Роль влияет на тон, структуру объяснений и глубину разбора. В этом примере при работе с моделью мы получим не обычный сухой ответ, а подробное объяснение по шагам — как от настоящего учителя.

Кроме преподавателя, полезными ролями для разработчика могут быть:

* Ревьюер кода — находит ошибки и предлагает улучшения.
* Технический интервьюер — задаёт уточняющие и наводящие вопросы.
* Партнёр по брейншторму — помогает генерировать и развивать идеи.

Правильно выбранная роль делает общение с моделью более продуктивным и приближает её поведение к поведению живого коллеги.

**Задача** — это то, что вы хотите получить от модели: объяснить сложную логику, сгенерировать необходимый код, найти ошибку, придумать название для переменной и многое другое. Чем чётче задача, тем точнее будет результат.

|  |  |  |
| --- | --- | --- |
| **Оценка** | **Промт** | **Комментарий** |
| - | Перепиши код внутри функции getUserAge | Непонятно, для чего нужно переписать код: для читаемости, производительности или соответствия стилю проекта |
| + | Перепиши функцию getUserAge так, чтобы она была более понятной для младшего разработчика. Разбей длинные выражения, добавь переменные и необходимые комментарии | Задача ясна: нужно улучшить читаемость, указана аудитория и подход к переписыванию |

**Формат** — это желаемый вид ответа: список, таблица, Markdown-разметка, код на определённом языке программирования и многое другое. Это особенно важно указать, если вы планируете сразу вставить ответ в проект или документацию.

|  |  |  |
| --- | --- | --- |
| **Оценка** | **Промт** | **Комментарий** |
| - | Скажи, какие есть способы маппинга DTO в TypeScript | Ответ может быть неструктурированным, с произвольным числом примеров |
| + | Опиши три примера способов маппинга DTO в TypeScript: ручной маппинг, class-transformer, zod. Для каждого приведи краткий пример в Markdown и пояснение в 1–2 предложениях | Задано число вариантов, структура, формат примеров и объём пояснений. Ответ легко читать и копировать |

**ОСНОВНЫЕ ТЕХНИКИ ПРОМТИНГА**

Чтобы получать от модели точные и полезные ответы, важно не только следовать ключевым принципам, но и уметь выбирать самый подходящий способ формулировки запроса. Таких способов много. Их называют **техниками** или **паттернами промтинга**. В зависимости от задачи вы можете использовать разные техники: от простых команд без примеров до сложных цепочек рассуждений.

Чтобы вы могли лучше разобраться в том, как работают эти техники, посмотрим, как можно по-разному решить одну и ту же задачу.

**Задача**

Использовать ИИ как персонального помощника, который будет анализировать изменения в коде (diff) и на их основе генерировать понятные, аккуратные коммит-сообщения.

Рассмотрим несколько основных подходов к её решению.

**Zero-Shot (без примеров)**

Zero-Shot-промтинг — это подход, при котором вы формулируете задачу и не даёте модели никаких примеров. Модель постарается понять, что от неё требуется, только опираясь на текст инструкции и собственные «знания».

Рассмотрим пример такого запроса.

Проанализируй следующий diff и сгенерируй короткое и понятное commit-сообщение. Оформи сообщение в императивном наклонении, на английском.

diff --git a/src/utils/calc.js b/src/utils/calc.js

index e69de29..4b825dc 100644

--- a/src/utils/calc.js

+++ b/src/utils/calc.js

@@

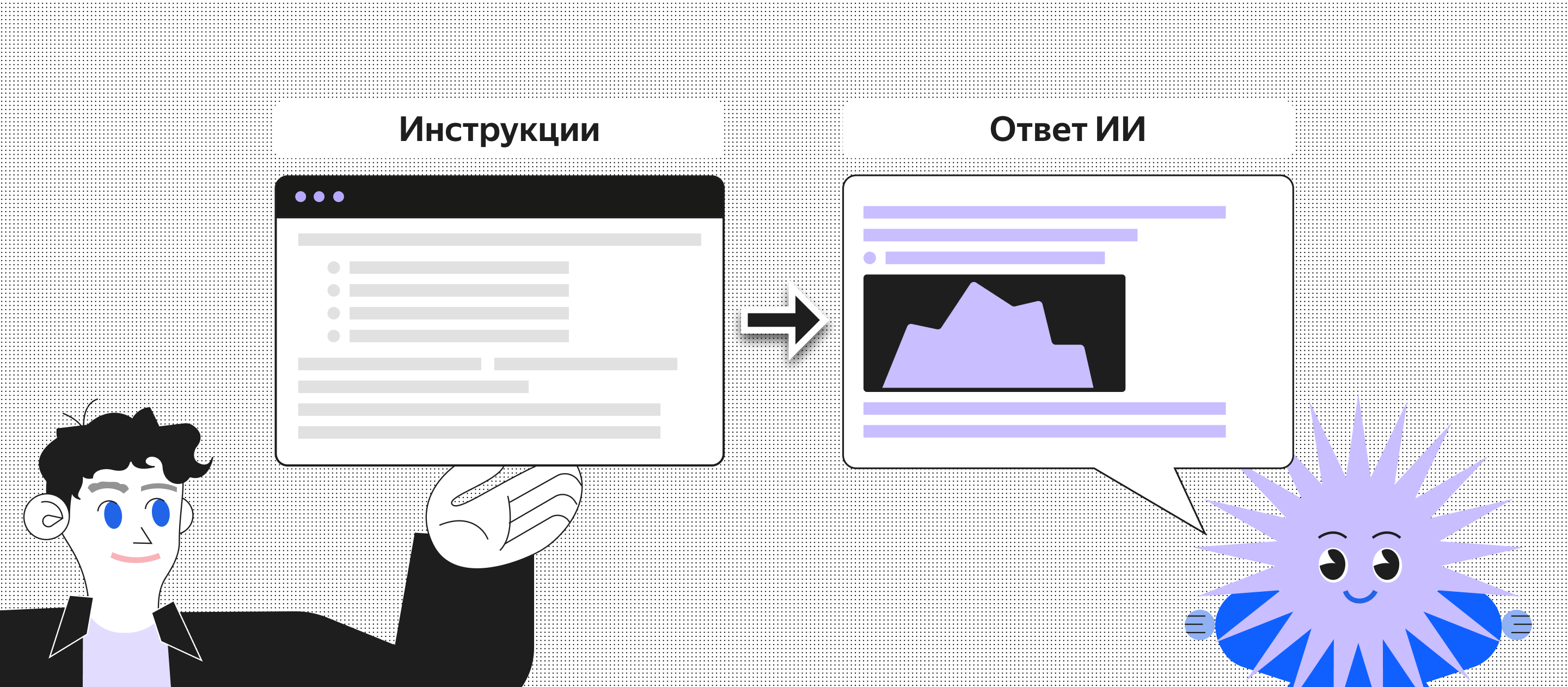
+export function sum(a, b) {

+ return a + b;

+}

Когда применять эту технику:

* Задача стандартная и понятная.
* Модель может опереться на общее знание, накопленное во время обучения.
* Нужно быстро получить результат без подготовки.



**Few-Shot (с примерами)**

Few-Shot-промтинг — это подход, при котором вы показываете модели 1–3 примера, прежде чем задать основную задачу. Модель использует эти примеры как шаблон и пытается продолжить в том же формате. Это помогает задать структуру, стиль и ожидаемый результат. За счёт этого ответ от модели, как правило, будет более релевантным.

Рассмотрим пример.

Сгенерируй короткое commit-сообщение по следующему diff. Используй английский язык, стиль — императив, без точки на конце.

Пример 1:

diff:

+ add README.md with basic project description

commit:

add project README

Пример 2:

diff:

+ function sum(a, b) { return a + b; }

commit:

add sum utility function

Теперь обработай следующий diff:

diff:

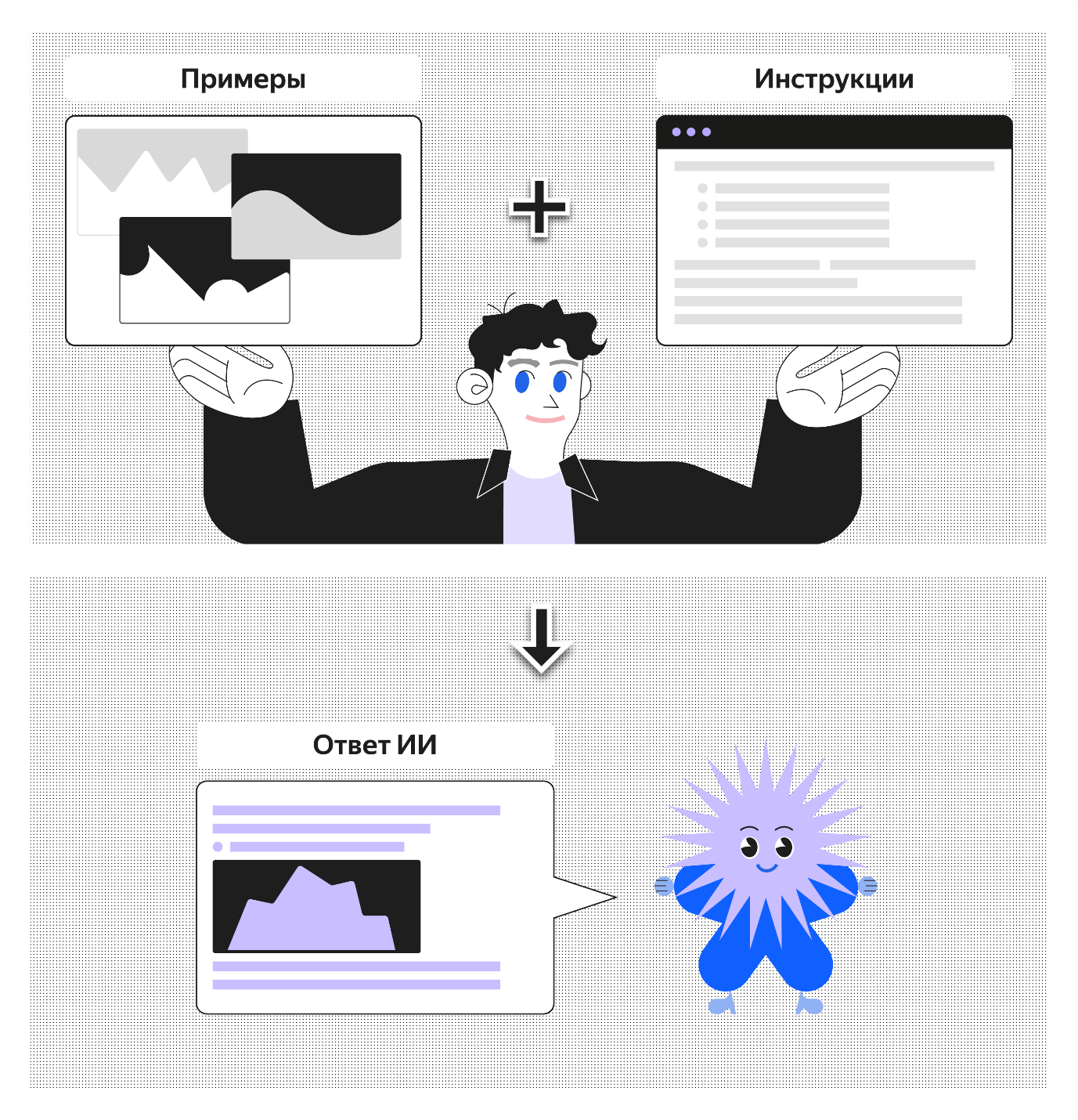
+ export function formatDate(date) {

+ return date.toISOString().split('T')[0];

+ }

Когда применять:

* Когда важно задать определённый тон или формат ответа.
* При нестандартных задачах, где у модели может быть несколько вариантов интерпретации.
* Когда требуется структурированный, шаблонный или предсказуемый вывод.



**Chain-of-Thought (цепочка рассуждений)**

Chain-of-Thought — это техника, при которой вы просите модель не просто дать ответ, а сначала пошагово объяснить своё мышление. Такой подход помогает улучшить качество вывода, особенно в задачах, где требуется логика или интерпретация.

Вот как может выглядеть такой запрос.

Проанализируй diff, пошагово объясни, какие изменения были внесены в код и с какой целью. Затем на основе анализа сгенерируй короткое commit-сообщение (на английском, в императивной форме, без точки).

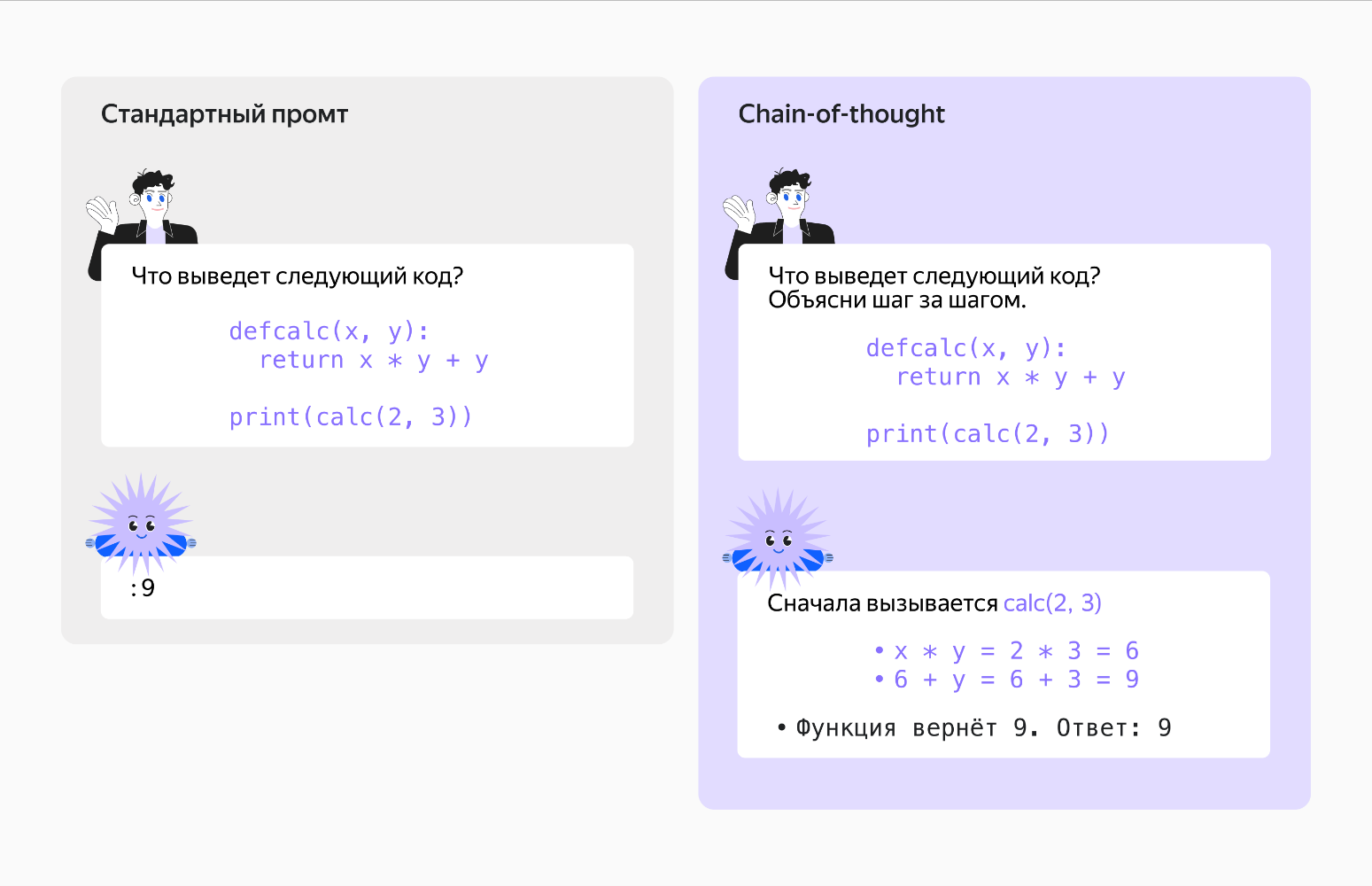
diff:

- const PORT = 3000;

+ const PORT = process.env.PORT || 3000;

Когда применять:

* Когда задача требует логического анализа или пояснения.
* Если нужно понять, почему модель делает тот или иной вывод.
* При разборе кода, данных или принятии решений, где важна прозрачность рассуждений.



Это основные техники промтинга, которые помогают достичь результатов в большинстве случаев. Если проблема более сложная, объёмная или предыдущие техники не привели к желаемому результату, можно применять более продвинутые техники. О них расскажем дальше.

**Продвинутые техники промтинга**

Продвинутые техники используют в сложных задачах. Например, если разработчику нужно создать сложный API с несколькими эндпоинтами и системой авторизации, которая требует детального планирования архитектуры, пошагового написания кода и тестирования функциональности.

Давайте рассмотрим некоторые из таких техник.

**Prompt Chaining (цепочка промтов)**

Prompt Chaining — это техника, при которой задача разбивается на несколько этапов, и каждый следующий шаг использует результат предыдущего. Пользователь уточняет запрос итерациями и получает более точные и полезные ответы.

Рассмотрим пример.

Промт 1:

Шаг 1. Определи, какие изменения произошли в коде:

diff:

- const PORT = 3000;

+ const PORT = process.env.PORT || 3000;

Запрос:

Опиши, что изменилось в коде и почему это важно.

Ответ ИИ:

Вместо жёстко заданного порта (3000) теперь используется значение из переменной окружения, с запасным вариантом.

Это позволяет адаптировать приложение к разным окружениям, например, продакшену или staging.

Промт 2:

Шаг 2. Сформулируй commit-сообщение на основе этого объяснения.

Запрос:

На основе предыдущего объяснения сгенерируй commit-сообщение.

Пиши кратко, на английском, в императивной форме, без точки.

Ответ ИИ:

commit:

make server port configurable via environment variable

Когда применять:

* Когда задача состоит из нескольких логических шагов.
* Если нужен промежуточный контроль за выводом модели на каждом этапе.
* При сложных или чувствительных задачах, где важно разбить процесс на понятные стадии (например, анализ → генерация → верификация).

**Ролевой промтинг**

Ролевой промтинг — это техника, при которой вы назначаете модели конкретную роль: специалиста, преподавателя, эксперта, редактора. Это помогает получить более релевантный, уверенный и стилистически выверенный ответ, особенно в профессиональных или прикладных задачах.

Например, в задаче миграции кода можно использовать ИИ в разных ролях: в качестве аналитика для поиска устаревшего синтаксиса, как разработчика — для предложения современных решений, как тестировщика — для проверки результата.

Пример:

Ты технический писатель, который помогает команде разработчиков формулировать понятные и лаконичные commit-сообщения. Твоя задача — проанализировать изменения в коде (diff), понять суть правок и сформулировать сообщение. Пиши на английском, в императивной форме, без точки в конце.

Вот diff:

+ export function validateEmail(email) {

+ return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);

+ }

Когда применять:

Когда вы хотите задать стиль или профессиональный контекст (например, архитектор, разработчик, тестировщик).

Если в задаче нужно соблюдать определённый тон, подход или уровень детализации.

Когда нужен экспертный ответ с учётом специфики роли или нескольких ролей.

**Метапромтинг**

Метапромтинг — это техника, при которой вы не задаёте основную задачу напрямую, а просите модель придумать, как правильно её сформулировать. Такой подход полезен, когда вы не уверены в формулировке или хотите получить более точный и эффективный промт с учётом внутреннего «опыта» модели.

Пример:

Ты система метапромтинга. Придумай оптимальный промт для генерации commit-сообщения на основе `diff`. В промте должны быть указаны:

- стиль commit-сообщения (язык, тон, формат);

- формат входных данных (diff);

- ожидаемый формат ответа.

Опиши этот промт полностью и чётко.

Ответ ИИ:

Сгенерируй короткое commit-сообщение на английском языке. Используй императивную форму, без точки в конце. Вход — diff. Ответ должен быть одной строкой.

Пример:

diff:

+ function sum(a, b) { return a + b; }

commit:

add sum utility function

Когда применять:

Когда неясно, как лучше задать вопрос или оформить задачу.

Для генерации промтов, которые можно переиспользовать или параметризовать.

При построении собственных агентов, тулов или цепочек, где важна гибкость в формулировке.

**Допромтинг и итеративное улучшение**

Даже при правильно выбранной технике результат не всегда получается с первого раза. Модель может неправильно интерпретировать задачу, пропустить важные детали или выбрать неподходящий стиль ответа. Поэтому важно уметь работать с ИИ как с собеседником: уточнять, переформулировать, добавлять контекст.

Такой подход называется **допромтинг** — вы начинаете с простого запроса, оцениваете ответ и дорабатываете его через уточнение промта. Этот процесс можно повторять, пока результат не станет полезным и точным.

Предположим, вы задали нейросети промт.

Когда применять:

Когда неясно, как лучше задать вопрос или оформить задачу.

Для генерации промтов, которые можно переиспользовать или параметризовать.

При построении собственных агентов, тулов или цепочек, где важна гибкость в формулировке.

Допромтинг и итеративное улучшение

Даже при правильно выбранной технике результат не всегда получается с первого раза. Модель может неправильно интерпретировать задачу, пропустить важные детали или выбрать неподходящий стиль ответа. Поэтому важно уметь работать с ИИ как с собеседником: уточнять, переформулировать, добавлять контекст.

Такой подход называется допромтинг — вы начинаете с простого запроса, оцениваете ответ и дорабатываете его через уточнение промта. Этот процесс можно повторять, пока результат не станет полезным и точным.

Предположим, вы задали нейросети промт.

Ответ ИИ:

Added formatDate function

Такой ответ не подходит, так как стиль не соответствует нашим требованиям — не императив, сообщение слишком общее.

Улучшаем промт:

Сформулируй commit-сообщение в императивной форме, на английском языке. Без точки на конце.

Ответ ИИ:

add formatDate function

Этот ответ лучше, но сообщение всё ещё слишком общее. Дорабатываем промт:

Сформулируй короткое commit-сообщение по этому diff. Стиль — английский, императив, без точки.

Уточни, что функция форматирует дату в формате YYYY-MM-DD.

Ответ ИИ:

add function to format date as YYYY-MM-DD

Этот ответ уже может вас устроить. Уточняйте и корректируйте свой промт сколько угодно раз, пока не получите результат, который вас удовлетворяет.

**КАК ФОРМУЛИРОВАТЬ ЭФФЕКТИВНЫЕ ПРОМТЫ**

Вы уже знаете принципы написания хороших промтов, которые позволяют получить от ИИ наиболее полезные и точные ответы:

* указание контекста;
* чёткая постановка задачи;
* указание формата ответа.

Посмотрите, как применять эти принципы для решения конкретной задачи — поиска и объяснения информации.

**Указывать контекст**

Чтобы ИИ мог лучше понять запрос, нужно дать ему дополнительное описание: ваш уровень знаний, цель и область применения. Это поможет адаптировать ответ под ваши потребности.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Плохой запрос** | **Хороший запрос** |  |  |
|  | *Что такое регрессия?* | *Я начинающий аналитик, и мне нужно понять, что такое линейная регрессия в статистике. Можешь объяснить простыми словами?* |  |  |
|  | В этом случае ИИ сам решит, насколько подробным и сложным должно быть описание. Далеко не факт, что его ответ нас устроит | Здесь ответ ИИ будет гораздо более предсказуемым, полезным и понятным |  |  |

**Ставить конкретную задачу**

Чётко указывайте, что именно вы хотите узнать, и давайте дополнительные пояснения, если это необходимо.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Плохой запрос** | **Хороший запрос** |  |  |
|  | *Как в SQL объединить данные?* | *Есть две таблицы: Clients и Movies. Таблица Clients содержит информацию о клиентах и имеет следующие колонки: id, name, mail, favorite\_movie. Таблица Movies хранит информацию о фильмах, и в ней есть колонки id, title, director и year. Нужно объединить эти таблицы так, чтобы в каждой строке содержалась информация о клиенте и о его любимом фильме. <…>* |  |  |
|  | У ИИ слишком много вариантов, как на это ответить: привести пример с конкретными таблицами, пересказать теорию или выдать лекцию об эффективности разных подходов | Точная формулировка сразу направит ИИ на решение вашей задачи |  |  |

**Указывать формат ответа**

По умолчанию ИИ часто представляет ответ в виде развёрнутого текста с небольшими примерами. Если это не то, что вам нужно, можно явно указать в запросе, в каком виде вы хотите получить информацию: текстом в 2–3 предложения, таблицей, списком, примером кода и так далее.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | **Плохой запрос** | **Хороший запрос** |  |  |
|  | *Покажи, как объединять таблицы в SQL* | *Покажи пример кода SQL для объединения таблиц с использованием LEFT JOIN* |  |  |
|  | Если нас интересует инструкция с описанием, такой запрос может сработать. Но если нужен только код, лучше написать об этом явно | С таким запросом гораздо больше шансов получить то, что нужно |  |  |

ИИ может не только подстроить ответ под указанный формат, но и адаптировать его стиль. Вот несколько примеров того, как можно задать стиль в запросе:

* Объясни, как учитель первокурсникам.
* Объясни простыми словами, без технических терминов.
* Объясни просто и с юмором, используя жизненные аналогии.
* Дай краткое объяснение в трёх предложениях.
* Объясни подробно с примерами кода.

Если первый ответ не совсем ясен или требуется уточнение, смело задавайте дополнительные вопросы:

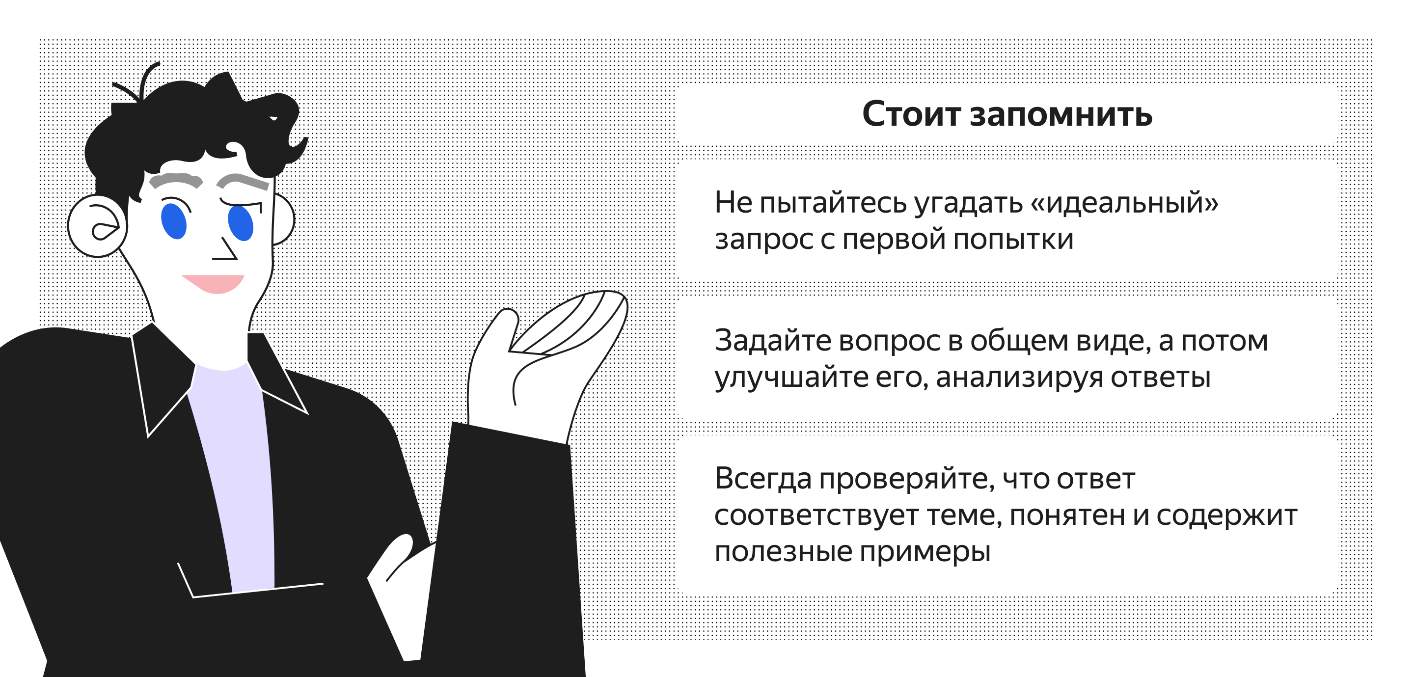
Мне не до конца понятно, как работает метод наименьших квадратов. Можешь объяснить это с примером?

ИИ тут же уточнит свой ответ и объяснит метод ещё более простыми словами.

**Примеры хороших запросов**

Вот несколько примеров эффективных промтов для разных задач.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Для объяснения концепта** | Я хорошо знаю SQL и работаю с большими таблицами. Напиши три совета по оптимизации SELECT-запросов в PostgreSQL, объясни каждую рекомендацию на примере, покажи, как меняется время выполнения запроса, и коротко прокомментируй, почему это важно |  |  |
|  | **Для описания инструмента** | Я только начинаю учить SQL и хочу понять, как извлекать данные из таблицы. Объясни, как работает оператор SELECT, на простом примере с таблицей employees (колонки: id, name, salary). Приведи пример запроса, короткое объяснение и ожидаемый результат в виде таблицы |  |  |
|  | **Для решения задачи** | У меня есть таблица sales с колонками product\_id, sale\_date и revenue. Нужно найти топ-5 товаров по выручке за последний месяц. Покажи SQL-запрос с комментариями, поясняющими каждый шаг |  |  |
|  | **Для решения проблемы** | Есть таблица users с колонками user\_id, name и email. Есть SQL-запрос для этой таблицы: "SELECT \* FROM users;". При выполнении запроса я получаю ошибку: "ERROR: syntax error at or near "SELECT". Почему происходит ошибка и как её исправить? |  |  |
|  | **Для сравнения подходов** | Я анализирую временные ряды продаж. Сравни методы скользящего среднего и экспоненциального сглаживания с точки зрения точности прогноза, сложности расчёта и требований к объёму данных |  |  |



**КАК ПРОВЕРИТЬ ОТВЕТ ИИ НА ДОСТОВЕРНОСТЬ**

Модели умеют писать подробные и правдоподобные тексты, но это не гарантия достоверности. Ошибки бывают разные: от устаревших фактов и синтаксических неточностей в формулах до галлюцинаций, когда ИИ описывает несуществующие понятия или инструменты. Если не проверять ответы, можно запомнить неправильную теорию или потратить время на неработающий код.

Пропускать шаг с проверкой ответа нельзя, даже если кажется, что текст звучит убедительно. Особенно это опасно в новых темах — там заметить ошибки труднее всего.

Рассмотрим несколько способов верификации — от самого простого до самого надёжного.

**Задать дополнительный запрос**

Самый простой способ — попросить ИИ оценить его собственный ответ. Например, можно вставить SQL-запрос и попросить: Проверь этот код на ошибки и совместимость с PostgreSQL.

Этот метод называется кросс-валидацией, или перекрёстной проверкой. Получив команду проанализировать информацию, ИИ может заметить ошибки, которые допустил в предыдущем ответе.

Для более точной оценки лучше проводить перекрёстную проверку в другом диалоге с ИИ.

**Сравнить ответы разных ИИ или повторить запрос**

Можно задать один и тот же вопрос разным моделям или повторить запрос к тому же ИИ, но с другой формулировкой. Если ответы совпадают, им можно доверять — хотя это и не абсолютная гарантия.

**Провести практическую проверку**

Если вы используете ИИ для написания кода, есть простой способ проверить его работу. Запустите код в «живой» среде, например, в SQL-редакторе или Python-ноутбуке. Если код работает и выдаёт осмысленный результат, всё хорошо. Если нет — вы сразу обнаружите проблему.

**Поискать в официальной документации**

Если речь идёт о конкретном инструменте (например, Excel), скорее всего, у него есть подробная документация. В ней можно найти информацию о правильном использовании функций и синтаксиса.

Если вы только начинаете изучать новую тему, работа с документацией может быть сложной. Необязательно читать её полностью: достаточно выделить ключевые термины из ответа ИИ и проверить в документации их наличие и определение. Также можно сравнить примеры из документации с теми, что предлагает ИИ.

**ГЕНЕРАЦИЯ УЧЕБНЫХ ЗАДАЧ ДЛЯ ПРАКТИКИ**

**Шаблон:**

Задача

Уровень сложности

Учебные цели

Стиль изложения

**Пример:**

Сгенерируй 1 учебную задачу по теме «SQL: группировки и агрегаты» для подготовки продуктового аналитика.

Уровень сложности: Junior.

Учебные цели: научиться корректно отвечать на вопросы ревьюера по теме «Группировки и агрегаты».

Формат заданий: разбей решение на 3–4 последовательных шага и покажи дополнительные вопросы от интервьюера.

Стиль изложения: «живая» беседа с интервьюером.

**Пример:**

Представьте, что вы учитесь на продуктового аналитика и начали изучать статистику. Ваша задача — с помощью ИИ составить три теста на тему «Дисперсия для начинающего специалиста» с акцентом на практические примеры из аналитики.

Сгенерируй 3 учебных задачи по теме «Дисперсия» для начинающего продуктового аналитика.

Уровень сложности: Junior.

Учебные цели: понять, что такое дисперсия, научиться её вычислять на простых примерах, интерпретировать результат.

Формат заданий: тестовые вопросы с вариантами ответов.

Стиль изложения: простой и понятный, с акцентом на практические примеры из аналитики.

**КАК СОЗДАВАТЬ ЭФФЕКТИВНЫЕ ПРОМТЫ ДЛЯ ПЛАНИРОВАНИЯ**

**Шаблон:**

Проект: чёткое описание задачи и бизнес-контекста.

Роль и уровень: ваша экспертиза и ролевая модель (DA, BA, SA).

Ограничения: время, ресурсы, данные, инструменты.

Результат: конечный артефакт и критерии успеха.

Формат: как именно оформить вывод.

**Пример:**

Помоги составить детальный план работ для аналитического проекта: [краткое название проекта].

Проект:

- [Подробное описание бизнес-проблемы, цели анализа и ключевые вопросы, на которые нужно ответить].

- [Описание имеющихся данных: источники, объём, известные проблемы качества].

Роль и уровень:

- Роль: [Бизнес-аналитик / Data Analyst / системный аналитик].

- Технический стек: [уверенный SQL, Python (pandas, sklearn), Tableau].

- Предметный опыт: [отрасль, домен, например: e-commerce, финтех, проведение A/B-тестов].

Ограничения:

- Срок: [общее время на проект, например: 2 недели].

- Ресурсы: [доступное время в день/неделю, бюджет, команда].

- Технические ограничения: [доступные инструменты, лимиты на вычисления, ограничения доступа к данным].

Результат:

- Конечные артефакты: [например: «Презентация для стейкхолдеров на 10 слайдов», «Дашборд в Tableau», «Технический отчёт с кодом на GitHub»].

- Критерии успеха: [например: «Выявлены 3 основные гипотезы падения метрики», «Предложены 2 приоритетные рекомендации к внедрению»].

Формат:

Представь план в виде таблицы с разбивкой по этапам. Для каждого этапа укажи:

1. Задачи и подзадачи.

2. Оценочные трудозатраты (в часах или днях).

3. Необходимые инструменты и технологии.

4. Потенциальные риски и способы их минимизации.

5. Критерии завершённости этапа.

Начало формы