# ГЛАВА 1

# Введение в промт-инжиниринг

Языковая модель ChatGPT была выпущена в конце ноября 2022 года. К январю следующего года количество постоянных пользователей приложения достигло планки в 100 млн, что сделало его самым быстрорастущим в истории (для сравнения: TikTok потребовалось девять месяцев, чтобы набрать 100 млн пользователей, а Instagram — 2,5 года), и, как вы легко можете убедиться, эта популярность заслуженна. Большие языковые модели, как та, на которой базируется ChatGPT, кардинально меняют наш подход к работе. Вместо того чтобы искать ответы через традиционный веб-поисковик вроде Google, вы можете просто попросить LLM рассказать о какой-либо теме. Вместо чтения Stack Overflow или тематических блогов достаточно попросить LLM дать персонализированную консультацию по вашему запросу, а затем задать ей дополнительные вопросы. Вместо повторения привычных шагов по написанию кода библиотеки программирования можно заметно ускорить процесс, объединившись с LLM-ассистентом, который будет постепенно улучшать и автоматически дополнять ваш код в процессе написания.

И вы, *будущие* читатели, наверняка станете использовать большие языковые модели так, как нам в 2024 году даже сложно представить. Если текущие тенденции сохранятся, скорее всего, вы будете общаться с LLM не один раз в день: с голосовым ассистентом техподдержки при отсутствии доступа в Интернет, с приветливым банкоматом на углу и даже с пугающе реалистичным роботом автообзвона. Будут и другие варианты использования. LLM отберут для вас новости, объединят заголовки потенциально интересных вам статей и удалят (а может, и *добавят*) предвзятые комментарии. LLM будут помогать вам в коммуникации: писать и анализировать письма, а домашние и офисные ассистенты даже смогут взаимодействовать с внешним миром от вашего имени. В течение дня ваш персональный ИИ-ассистент сможет как выполнять роль турагента, взяв на себя работу по составлению плана поездки, бронированию авиабилетов и отеля, так и помогать с поиском и приобретением товаров.

Почему большие языковые модели такие поразительные? Как говорил футурист Артур Кларк, «любая достаточно развитая технология неотличима от магии». Люди воспринимают машину, с которой можно пообщаться, как что-то волшебное, однако цель нашей книги — развеять магию. Мы намерены продемонстрировать, что, какими бы сверхъестественными, проницательными и человекоподобными ни казались большие языковые модели, по своей сути они просто предугадывают следующее слово в блоке текста — не более! Важно понимать, что LLM — всего лишь инструмент, помогающий пользователям с выполнением задач, и для работы с ним потребуется освоить составление *промптов*, фрагментов текста, которые LLM нужно завершить. Вот что мы называем *промт-инжинирингом*. Книга предоставляет практическую основу для промт-инжиниринга и в конечном счете для создания приложений на базе LLM, которые станут по-настоящему волшебными для пользователей.

Данная глава представляет собой небольшой исторический экскурс в мир LLM, ознакомившись с которым мы уже приступим к изучению промт-инжиниринга. Но сначала позвольте рассказать, как мы, авторы книги, открыли эту магию для себя.

## Большие языковые модели — это волшебство

Мы оба были одними из первых разработчиков продукта по автодополнению кода — GitHub Copilot. Альберт состоял в команде основателей, а Джон подключился, когда Альберт уже перешел на другие зарождающиеся исследовательские проекты LLM.

Альберт впервые открыл для себя «волшебство» LLM в середине 2020 года. Вот как он это описывает.

> Примерно раз в полгода на наших групповых встречах, где мы обсуждали идеи применения машинного обучения в программировании (ML-on-code), кто-то обязательно поднимал вопрос о синтезе (генерации) кода. И ответ всегда был одинаков: когда-нибудь случится прорыв, но этот день настанет не раньше чем через пять лет. Для нас это было чем-то вроде холодного ядерного синтеза в физике — так же недостижимо.
>
> Так все и оставалось, пока мне в руки не попал ранний прототип LLM, позже ставший OpenAI Codex. Я увидел, что будущее уже наступило: холодный синтез наконец-то стал реальностью.
>
> Я сразу понял, что эта модель полностью отличалась от того, что мы видели раньше. Она могла не только предугадать следующее слово, но и генерировать целые предложения и функции из всего-навсего строки документации. И что самое поразительное — функции работали!

> Прежде чем решить, что мы можем построить на базе этой модели (спойлер: впоследствии она превратится в GitHub Copilot), мы хотели оценить, насколько она хороша. Так что мы привлекли группу добровольцев из состава инженеров GitHub, чтобы они снабдили нас самодостаточными задачами по программированию. Некоторые из представленных ими задач были сравнительно легкими, но, поскольку писали их бывалые программисты, они все равно оказывались довольно интересными. Одни задачи были такими, с которыми младший разработчик обратился бы к Google за помощью, а другие заставили бы даже старшего разработчика заглянуть на Stack Overflow. И все равно с нескольких попыток модель решила большинство из них. Тогда мы поняли — это «двигатель», который откроет новую эру написания кода. Все, что нам оставалось, — построить вокруг него правильный «автомобиль».

Для Джона волшебный момент наступил пару лет спустя, в начале 2023 года, когда он тестировал этот «автомобиль». Вот как он это вспоминает.

> Я включил запись экрана и дал команду разобраться с непростой задачей по программированию: создать функцию, которая возвращает текстовую форму целого числа. Например, если ввести 10, на выходе должно быть «десять», если ввести 1 004 712, должно получиться «один миллион четыре тысячи семьсот двенадцать». Это сложнее, чем кажется, из-за обилия нелогичных исключений в естественном языке. Текстовые версии чисел между 10 и 20 образуются по иному шаблону, нежели числа в других десятках. Да и в разряде десятков шаблон постоянно ломается: например, если 90 — «девяносто», то почему 80 — «восемьдесят», а 40 — вообще «сорок»? Но главный подвох был в том, что я хотел реализовать решение на языке, с которым никогда не работал, — Rust. Справится ли с этой задачей Copilot?
>
> Обычно, изучая новый язык программирования, я задавал стандартные вопросы: как создать переменную? Как создать список? Как перебирать элементы списка? Как написать условный оператор? Но в этот раз я начал просто со строки документации (docstring):

```
// ЦЕЛЬ: создать функцию, которая выводит строковое
// представление любого числа, переданного в нее.
// 1 -> "один"
// 2034 -> "две тысячи тридцать четыре"
// 11 -> "одиннадцать"
fn
```

> Copilot заметил `fn` и поспешил помочь:

```
fn number_to_string(number: i32) -> String {
```

> Отлично! Я не знал, как присваивать типы входным аргументам или возвращаемым значениям функций, но по ходу нашей совместной работы направлял высокоуровневые рабочие процессы с помощью комментариев наподобие «Разбей входящее число на трехзначные группы», а Copilot учил меня

программным конструкциям. К примеру, он показал, как создавать векторы и присваивать их переменным: `let mut number_string_vec = Vec::new();` а также научил создавать циклы: `while number > 0 {`.

Это был замечательный опыт. Я продвигался вперед и изучал язык, не отвлекаясь на учебники: мой проект и был моим учебником. Затем, спустя 20 минут эксперимента, Copilot просто перевернул мое сознание. Я набрал комментарий и начал следующий цикл управления.

```
// пройди циклом по number_string_vec, собери название числа
// для каждого порядка величины и соедини их в number_string
for
```

После небольшой паузы Copilot выдал 30 строк кода! На записи даже слышен мой вздох от удивления (https://oreil.ly/4ZYWY). Код был синтаксически корректен, он успешно скомпилировался и запустился. Полученный ответ был немного странным: введенное число 5 034 012 на выходе превратилось в строку «пять тридцать четыре тысячи двенадцать миллионов», но, знаете, я бы и от человека не ждал идеального результата с первой попытки, а эту ошибку было легко найти и исправить. К концу сорокаминутной парной сессии программирования я сделал невозможное — создал нетривиальный код на языке, абсолютно мне незнакомом! Copilot помог мне освоить базовый синтаксис языка Rust, продемонстрировал понимание моих целей и несколько раз вмешался, чтобы помочь мне разобраться с деталями. Если бы я попробовал выполнить задачу самостоятельно, это заняло бы несколько часов.

Наши «волшебные» переживания не уникальны. Если вы читаете эту книгу, скорее всего, у вас тоже были захватывающие взаимодействия с языковыми моделями. Может, вы впервые узнали о возможностях LLM через ChatGPT, или ваш первый опыт был связан с приложениями первого поколения, появившимися в начале 2023 года: поисковыми помощниками Microsoft Bing и Google Bard или помощником по работе с документами, таким как расширенный набор программ Microsoft Copilot.

Однако достигнуть этого переломного момента удалось не за один день. Чтобы по-настоящему понять большие языковые модели, нужно проследить весь путь развития технологии.

## Языковые модели: как мы к этому пришли

Чтобы понять, как мы добрались до этой крайне интересной точки в истории технологий, сначала нужно разобраться, что же такое языковая модель и что она делает. Кому же еще задать этот вопрос, как не самой популярной большой языковой модели: приложению ChatGPT (рис. 1.1)?

**Вы**
Что такое языковая модель?

**ChatGPT**
Языковая модель — это тип системы искусственного интеллекта, обученный понимать и генерировать тексты, похожие на человеческие. Она изучает языковую структуру, грамматику и семантику, обрабатывая большие объемы текстовых данных. Основная задача языковой модели — предсказание вероятности появления слова или последовательности слов в заданном контексте.

**Рис. 1.1.** Что такое языковая модель

Видите? Все так, как мы говорили в начале главы: главная задача языковой модели — предсказать вероятность следующего слова. Вы уже видели такую функциональность раньше: это та самая панель со словами, появляющаяся над клавиатурой вашего смартфона, когда вы набираете сообщение (рис. 1.2). Возможно, вы никогда не обращали на нее внимания... *потому что она не такая уж и полезная*. И если это все, на что способны LLM, как же тогда они сейчас переворачивают мир?

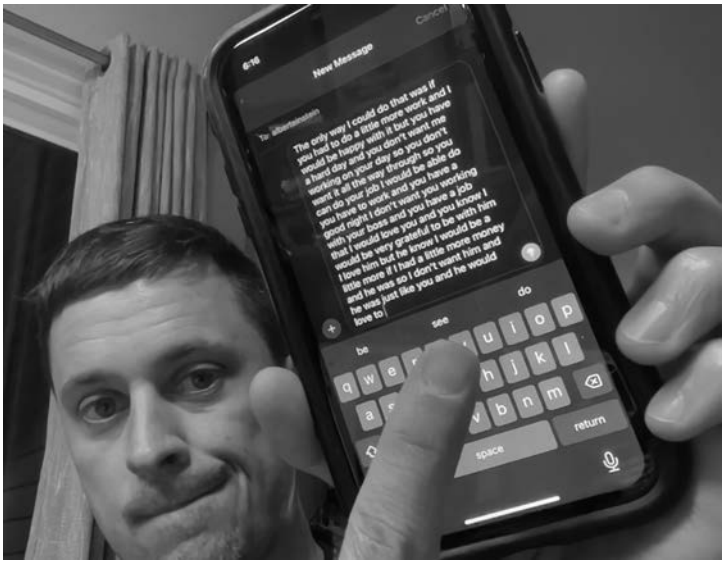


**Рис. 1.2.** Джон показывает панель автодополнения на своем телефоне

## Первые языковые модели

Языковые модели существуют уже давно. Если вы читаете эту книгу вскоре после публикации, то языковая модель, которая позволяет iPhone угадывать следующее слово, скорее всего основана на модели естественного языка Маркова, появившейся еще в 1948 году (https://oreil.ly/D6Q3U). Однако встречаются и более ранние языковые модели, которые заложили фундамент для ИИ-революции, происходящей сейчас.

До 2014 года самые мощные языковые модели были основаны на архитектуре обработки последовательностей данных (seq2seq) от Google (https://arxiv.org/abs/1409.3215). Seq2seq — это рекуррентная нейронная сеть, которая в теории идеально подходила для обработки текстов, так как она обрабатывает один токен в единицу времени, рекуррентно обновляя свое внутреннее состояние. Это позволяет seq2seq работать с произвольно длинными текстовыми последовательностями. С помощью специализированных архитектур и обучения seq2seq могла справляться с несколькими различными типами заданий на естественном языке: классификацией, извлечением объектов, переводом, пересказом и многими другими. Но у таких моделей был один недостаток, который ограничивал их возможности, — потеря информации.

У архитектуры seq2seq есть два главных компонента: кодировщик и декодировщик (еще называют энкодером и декодером) (рис. 1.3). Процесс обработки начинается с отправки кодировщику потока токенов, которые обрабатываются по одному. По мере поступления токенов он обновляет скрытый вектор состояния, который накапливает информацию из входящей последовательности. Как только последний токен обработан, окончательное значение скрытого состояния, называемое вектором промежуточного представления (thought vector, иногда называют «вектором мысли»), передается декодировщику. Декодировщик, в свою очередь, использует информацию из этого вектора для генерации выходных токенов. Проблема в том, что вектор промежуточного представления (вектор состояния) фиксирован и конечен. Он часто «забывает» важную информацию из длинных фрагментов текста, оставляя декодировщику недостаточно данных для работы — это и есть проблема потери информации.

Модель, представленная на рис. 1.3, работает следующим образом.

1. Токены из исходного языка последовательно передаются в кодировщик, конвертируются в эмбеддинг (векторное представление), а затем обновляют внутреннее состояние кодировщика.
2. Внутреннее состояние упаковывается в вектор промежуточного представления и отправляется в декодировщик.

3. В декодировщик посылается специальный «стартовый» токен, обозначая начало выходных токенов.
4. Состояние декодировщика обновляется в зависимости от значения вектора, и выдается выходной токен на целевом языке.
5. Выходной токен служит следующей информацией, вводимой в декодировщик. С этого момента процесс рекуррентно повторяется от шага 4 к шагу 5.
6. В конце декодировщик выдает специальный «конечный» токен, сигнализирующий о завершении декодирования. Ограниченный вектор состояния может передать только ограниченное количество информации в декодировщик.

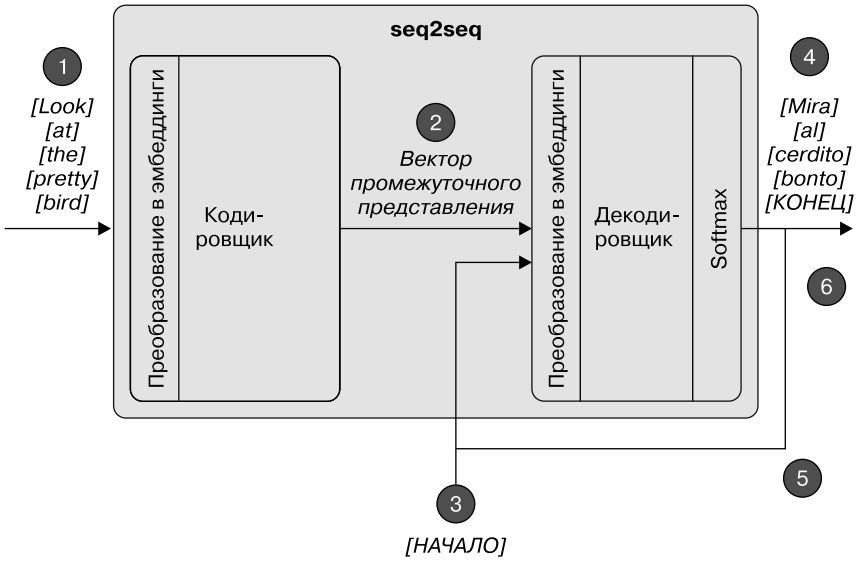


**Рис. 1.3.** Модель перевода seq2seq

В статье 2015 года *Neural Machine Translation by Jointly Learning to Align and Translate* (https://arxiv.org/abs/1409.0473) был представлен новый подход к решению проблемы потери информации. Кодировщик не пополняет единственный вектор состояния, а сохраняет все векторы скрытых состояний, сгенерированные для каждого токена, и затем позволяет декодировщику совершить «мягкий поиск» (поиск с учетом контекста) по всем векторам. Исследование демонстрирует, что использование мягкого поиска в модели перевода с английского на французский значительно улучшает качество результата. Такой метод вскоре стал известен как механизм внимания.

Механизм внимания вызвал большой интерес среди ИИ-сообщества, что в 2017 году привело к публикации статьи Google Research *Attention is all you need* (https://arxiv.org/abs/1706.03762), где была представлена архитектура трансформера (рис. 1.4).

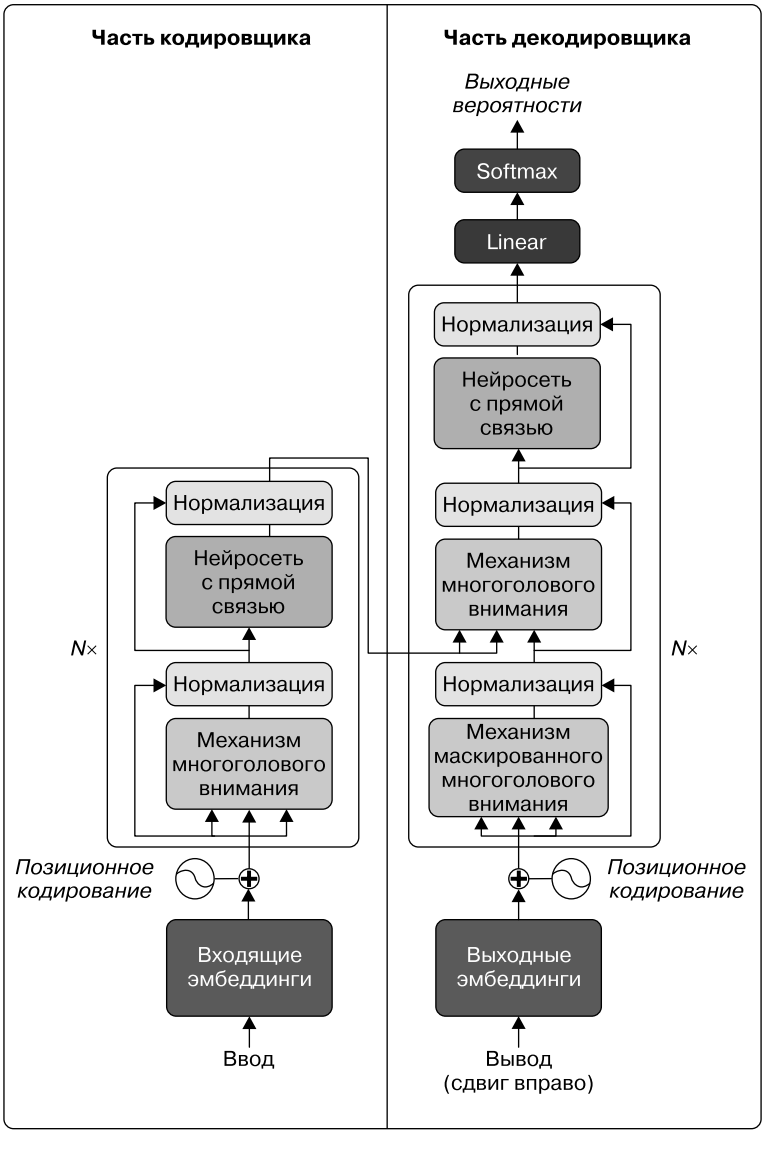


**Рис. 1.4.** Архитектура трансформера

Сохранив высокоуровневую структуру предшественника, трансформер состоял из кодировщика, получающего токены на входе, и декодировщика, генерирующего выходные токены. Но в отличие от модели seq2seq все рекуррентные цепочки были удалены. Вместо этого трансформер целиком полагался на механизм внимания. Получившаяся архитектура была очень гибкой и гораздо лучше моделировала обучающие данные, чем seq2seq. Но если seq2seq могла обрабатывать последовательности любой длины, то трансформер справлялся только с фиксированной, конечной последовательностью входных и выходных данных. А поскольку трансформер — прямой предшественник моделей ChatGPT, нам приходится бороться с этим ограничением до сих пор.

## GPT

Архитектура генеративного предварительно обученного трансформера была представлена в статье 2018 года под названием *Improving Language Understanding by Generative Pre-Training* (https://oreil.ly/vIiDJ). Данная архитектура не была какой-то особенной или новой. На деле это был тот же трансформер, но без кодировщика — только декодировщик. Тем не менее такое упрощение открыло новые неожиданные возможности, которые будут полностью реализованы только в следующие несколько лет. Именно эта архитектура генеративного предварительно обученного трансформера — GPT — вскоре совершит революцию в области искусственного интеллекта.

В 2018 году это еще не было очевидно. На тот момент стандартной практикой было *предварительно обучать* модели на неразмеченных данных: например, на обрывках текста из Интернета, а потом вносить изменения в архитектуру модели и применять специальную тонкую настройку (fine-tuning), чтобы в итоге модель могла эффективно решать *одну* конкретную задачу. Так же дела обстояли с архитектурой генеративного предварительно обученного трансформера. Статья 2018 года просто подчеркнула, что эта схема отлично работает с GPT — предварительное обучение на неразмеченном тексте и последующая контролируемая тонкая настройка для конкретной задачи помогли создать очень хорошие модели для разнообразных целей: классификации, сравнения документов, ответов на вопросы с выбором вариантов. Но мы должны отметить, что после тонкой настройки GPT хорошо справлялась только с задачей, для которой и была настроена.

GPT-2 стала просто более масштабной версией GPT. На момент ее появления в 2019 году исследователи уже начали догадываться, что архитектура GPT была особенной. Подтверждение этому можно найти во втором абзаце блога OpenAI, посвященном GPT-2 (https://oreil.ly/tv8t).

> Наша модель под названием GPT-2 (следующая версия GPT) была обучена на 40 Гбайт взятого из Интернета текста просто предсказывать следующее слово. Мы отменяем релиз этой обученной модели из-за опасений по поводу вредоносного использования лежащей в ее основе технологии.

Вот это да! Как эти два предложения оказались рядом друг с другом? Каким образом столь безобидная вещь вроде предсказания следующего слова — как это делает iPhone, когда вы пишете текстовое сообщение, — может привести к таким серьезным опасениям? Чтобы понять это, рекомендуем ознакомиться с соответствующей научной работой *Language Models Are Unsupervised Multitask Learners* (https://oreil.ly/QEeI9). У GPT-2 было 1,5 млрд параметров в сравнении с 117 млн у первой версии, она была обучена на 40 Гбайт текста, а GPT — на 4,5 Гбайт. Простое увеличение на порядок масштаба самой модели и размера данных для обучения привели к беспрецедентному взрывному росту качества — вместо тонкой настройки GPT-2 под конкретную задачу можно было использовать неподготовленную предварительно обученную модель и зачастую добиться лучшего результата, чем у передовых моделей, заточенных специально под этот вид задач. Оценка включала в себя тесты и бенчмарки для понимания двусмысленных местоимений, предсказывание отсутствующих в тексте слов, разметку частей речи и многое другое. И несмотря на то, что GPT-2 не была самой продвинутой технологией, она не только не отставала, но и справлялась с пониманием текстов, пересказом, переводом и ответами на вопросы не хуже специализированных моделей.

Но откуда столько беспокойства по поводу «вредоносного использования» этой модели? Дело в том, что она стала очень хорошо имитировать естественные тексты. Как отмечается в блоге OpenAI, эта способность может быть использована «для создания ложных новостных статей, выдачи себя за других людей в Сети, публикации оскорбительных или поддельных материалов в социальных сетях, а также автоматизации производства спама или фишинга». Сегодня данная угроза стала еще более реальной, чем в 2019 году.

GPT-3 претерпела еще одно значительное увеличение как в размере модели, так и в объеме обучающих данных, что привело к соответствующему скачку возможностей. В статье 2020 года под названием *Language Models Are Few-Shot Learners* (https://arxiv.org/abs/2005.14165) показано, что, получив несколько примеров задачи (few-shot examples), которую нужно выполнить модели, она способна точно воспроизвести шаблон входящих данных и в конечном счете справиться с практически любой языковой задачей, часто с удивительно качественными результатами. Именно тогда мы поняли, что можно изменить вводимую информацию — промт — и тем самым заставить модель выполнить требуемое задание. Так родился промт-инжиниринг.