ГЛАВА 1

Введение

Генеративный искусственный интеллект (GenAI) — мощный и в то же время простой в применении инструмент: чтобы создать на его основе прототип перспективного приложения, не нужно обладать техническими знаниями. Однако при переходе в продакшен вы непременно столкнетесь с трудностями, поскольку GenAI-модели не слишком надежны: склонны к галлюцинациям, по-разному отвечают на одинаковые вопросы, а также не могут выходить за рамки того, чему были обучены. Паттерны проектирования, представленные в книге, отражают передовые практики и решения этих и других проблем, с которыми вы, вероятно, столкнетесь при создании реально работающих приложений на основе GenAI-моделей.

Паттерны проектирования для GenAI

Как хорошо известно разработчикам ПО, *паттерны проектирования* (design patterns) — это проверенные решения типовых проблем, которые часто встречаются при разработке и проектировании программных систем, а также накопленные со временем и стандартизированные лучшие практики. По сути, паттерны проектирования стали для разработчиков языком эффективного общения, они помогают им повышать качество, сопровождаемость и масштабируемость приложений.

Это понятие возникло под сильным влиянием книги архитектора Кристофера Александера (Christopher Alexander), который ввел архитектурные паттерны в книге «*A Pattern Language*[1]» (изд-во Oxford University Press, 1977). В разработке ПО паттерны проектирования приобрели широкую известность с выходом книги «*Design Patterns: Elements of Reusable Object-Oriented Software*[2]» Эрика Гаммы (Erich Gamma), Ричарда Хелма (Richard Helm), Ральфа Джонсона (Ralph Johnson) и Джона Влиссидеса (John Vlissides) (изд-во Addison-Wesley) — авторов, которые стали известны как «Банда четырех». С того времени каталоги паттернов появились и для других предметных областей, например приложений на Java Enterprise и машинного обучения.

В наши дни при разработке ИИ-продуктов все чаще используются не кастомные ML-модели, которые приходится обучать с нуля, а *базовые* (foundational) модели

[1] Александер К. «Язык шаблонов».

[2] Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. «Паттерны объектно-ориентированного проектирования». Санкт-Петербург, издательство «Питер».

GenAI (в том числе GPT-4, Gemini, Claude, Llama, DeepSeek, Qwen и Mistral), обученные на больших, не связанных с конкретными приложениями датасетах. В книге Чип Хьюен (Chip Huyen) «*AI Engineering*»[1] (изд-во O'Reilly) этот подход назван *ИИ-инженерией* (AI engineering), а те, кто его практикует, — *ИИ-инженерами*.

ИИ-инженерия широко применяется, например, при обработке естественного языка (natural language processing, NLP), генерации текстов, толковании кода, распознавании изображений и синтезе видео. Благодаря ей больших успехов удалось добиться в создании ИИ-ассистентов, автоматизации рабочих процессов и робототехнике.

Как ИИ-инженер, вы можете поручить базовой модели сгенерировать необходимый приложению контент, отправив ей небольшой текст, называемый *промптом* (prompt). Однако при этом вы столкнетесь с рядом распространенных проблем: к примеру, полученный контент может не соответствовать нужному стилю, в нем могут отсутствовать узкоспециальные сведения, не известные модели.

Поэтому в этой книге мы составим подробный каталог из 32 проверенных решений для подобных типовых проблем, систематизируем научные достижения и опыт экспертов в рекомендации, которые можно легко применить на практике. В каждой главе предлагается набор паттернов для решения конкретной проблемы, часто возникающей при работе с ИИ. Так, например, в главе 3 мы обсудим, что можно сделать, когда необходимо генерировать контент с использованием секретных корпоративных сведений, которые не известны базовой модели, поскольку обучавшие ее провайдеры не имеют к ним доступа. В каждом пункте вы найдете описание проблемы, проверенное решение, работоспособный пример, а также обсуждение альтернатив и других аспектов реализации паттерна.

Перед ИИ-инженерами часто ставятся задачи, которые слишком сложны для того, чтобы базовая модель могла решить их за один раз, а потому их разделяют на небольшие подзадачи. Предназначенные для их решения программные компоненты на базе ИИ принято называть *агентами*. Сегодня агенты становятся все более автономными, могут использовать модели GenAI, чтобы планировать последовательность операций, выбирать необходимые для работы инструменты бэкенда, восстанавливаться после ошибок, а также оценивать результаты своей работы. Приложения, собранные из набора агентов, называются *агентными*. Паттерны, о которых мы будем говорить в этой книге, позволят вам не только справляться с неизбежно возникающими при работе с ИИ проблемами, но и создавать более эффективных агентов и агентные приложения.

Приложения на основе базовых моделей

В этом пункте мы кратко рассмотрим основы ИИ-инженерии — вопросы, к которым больше не будем возвращаться в последующих главах. Более подробно

[1] Хьюен Ч. «AI-инженерия. Построение приложений с использованием базовых моделей».

о приложениях на основе GenAI вы можете прочитать, например, в книгах Омара Сансевьеро (Omar Sanseviero) «*Hands-On Generative AI with Transformers and Diffusion Models*» (изд-во O'Reilly), Криса Фрегли (Chris Fregly) «*Generative AI on AWS*» (изд-во O'Reilly) и Леонида Кулигина (Leonid Kuligin) «*Generative AI on Google Cloud with LangChain*» (изд-во Packt), в которой описаны, соответственно, основные технологии базовых моделей, работа с гиперскейлером, а также фреймворк с открытым исходным кодом для работы с GenAI.

Замечание о моделях и фреймворках

Во всех примерах в книге «*Machine Learning Design Patterns*»[1] мы, в целях единообразия, применяли два фреймворка (scikit-learn и TensorFlow) и один гиперскейлер (Google Cloud Platform, GCP). Однако многим читателям эта идея не показалась удачной. Поэтому в этот раз мы не будем привязывать материал к каким-то определенным моделям, фреймворкам и гиперскейлерам. Напротив, на этих страницах вам встретятся технологии самых разных поставщиков: большие языковые модели (large language model, LLM) от OpenAI, Anthropic, Google, Alibaba и Meta, фреймворки LangChain, Pydantic AI, Hugging Face и DSPy, гиперскейлеры Amazon Web Services (AWS), Azure, GCP и Oracle Cloud Infrastructure.

Поскольку с большой вероятностью вам предстоит использовать и другие технологии, примеры кода из этой книги необходимо рассматривать лишь как отправную точку для вашей реализации — они нуждаются в дальнейшей адаптации под нужные вам модели, фреймворки и гиперскейлеры.

Промпт и контекст

Обычно приложения на основе GenAI обращаются к серверным базовым моделям при помощи API, предоставленного ее разработчиком. Такое обращение может происходить напрямую или же через фреймворк, который поддерживает несколько моделей и позволяет легко переключаться между ними.

Взаимодействуя с моделью, приложение отправляет ей *промпты* и получает от нее *ответы*. Вы, безусловно, хорошо знакомы с этим принципом по веб-интерфейсу, к примеру, ChatGPT (http://chatgpt.com), при помощи которого можно ввести промпт в текстовом виде. Допустим, вы просите:

```
Нарисуй карандашный эскиз в стиле Дега: семья с двумя детьми играет в настольную игру.
```

[1] Лакшманан В., Робинсон С., Мунн М. «Машинное обучение. Паттерны проектирования».

Простейший промпт представляет собой *инструкцию*, по которой модель сгенерирует заданный контент, в данном случае — изображение в указанном стиле (рис. 1.1)[1]. Однако как промпты, так и ответы могут быть *мультимодальными*, то есть состоять из текста, изображений, видео и аудио.



Рис. 1.1. Промпт и ответ в диалоге с базовой моделью

Более сложный промпт может содержать не только инструкции, но и *контекст*, который модель должна использовать при ответе. К примеру, мы можем попросить

[1] Посмотрите на этот эскиз внимательно. Вам ничего не кажется странным? У всех персонажей положенное число пальцев? Лицо у девочки симметрично? И что за игра у них на столе? Увы, на момент написания этой книги модели — генераторы изображений все еще не в ладах с симметрией, счетом и реалистичностью.

Claude (сайт Claude.ai) представить себя в какой-то определенной роли, а также учесть дополнительные сведения:

> Ты — опытный маркетолог, хорошо знакомый с книжным рынком университетских городов Германии.
>
> «Завет воды» — роман, в котором повествуется о трех поколениях семьи православных христиан из церкви апостола Фомы в Керале.
>
> Напиши краткую аннотацию к этой книге для клиентов книжного магазина из Геттингена.

Выполняя инструкцию, Claude Sonnet выступит в указанной роли и примет во внимание контекст:

> **«Завет воды» — литературное путешествие в Кералу**
>
> В древнем Геттингене, где академические традиции сочетаются с любовью к литературе, мы представляем мастерски написанное эпическое творение Абрахама Вергезе «Завет воды». Эта захватывающая сага повествует о трех поколениях православных христиан из церкви апостола Фомы, живущих в Керале (Индия) — регионе, чье сложное религиозное наследие может найти отклик у студентов-теологов из нашего университета. Сюжет этой книги разворачивается среди живописных керальских заводей, так что читатели, знакомые с историей Геттингена и его отношениями с водой — от священного фонтана Гензелизель до извилистой реки Лайне, — обнаружат удивительные параллели в том, как география влияет на судьбу. Вергезе описывает историю семейных тайн, медицинских загадок и связей между поколениями, которая выходит за рамки культурных границ и предлагает международному сообществу ученых и тех, кто всю жизнь стремится к знаниям, глубокое исследование того, как вода — будь то муссоны Кералы или тихие дожди Нижней Саксонии — становится одновременно метафорой и заветом, объединяя людей во времени и пространстве.

Работа с API провайдера модели

Предназначенные для обращений к базовым моделям API доступны на множестве языков, среди которых наибольшей популярностью пользуются Python, Go, Java и TypeScript. Для простоты в этой книге мы будем использовать Python.

Для работы с Claude Sonnet 3 при помощи Anthropic API на Python необходимо загрузить пакет Anthropic и написать, например, следующий код (полная версия находится на GitHub (https://oreil.ly/NauRN)):

```
import anthropic
client = anthropic.Anthropic(
  # По умолчанию - os.environ.get("ANTHROPIC_API_KEY")
  api_key="YOUR_ANTHROPIC_API_KEY",
)

completion = client.messages.create(
  model="claude-3-7-sonnet-latest",
  system="Ты - разработчик-эксперт на Python.",
  messages=[
    {
      "role": "user",
```

```
      "content": [
        {
          "type": "text",
          "text": """
Напиши код, вычисляющий медианное значение списка целых чисел.
"""
        }
      ]
    }
  ]
)

print(completion.content[0].text)
```

В этом обращении к API промпт состоит из двух частей: *системного промпта*, который формируется разработчиком и определяет общее поведение модели (в данном случае — предписывает ей конкретную роль), и *пользовательского промпта*, описывающего задачу, которую должна выполнить модель, то есть содержащего контекст и инструкции.

Работа с универсальным фреймворком

Чтобы решить ту же задачу при помощи фреймворка PydanticAI, потребуется следующий код (здесь мы предполагаем, что ключ для доступа к API хранится в переменной окружения):

```
from pydantic_ai import Agent
agent = Agent('anthropic:claude-3-7-sonnet-latest',
              system_prompt="Ты - разработчик-эксперт на Python.")

result = agent.run_sync(
           "Напиши код, вычисляющий медианное значение списка целых чисел.")
print(result.data)
```

Этот подход дает важное преимущество: мы можем перейти на другую базовую модель, заменив строку с обращением к Claude, скажем, на `openai:gpt-4o-mini`, `google-vertex:gemini-2.0-flash` или `groq:llama3-70b-8192`. Полный список поддерживаемых моделей можно найти в документации к Pydantic (https://oreil.ly/Kg6jE).

Для обращения к базовой модели в Pydantic API используется класс под названием `Agent`. Мы будем говорить об агентах в следующем разделе, а пока закончим обсуждение способов обращения к базовым моделям.

Локальный запуск модели

Чтобы использовать модель наподобие Llama 3 на локальном компьютере, необходимо загрузить и запустить ее при помощи клиента Ollama (https://ollama.com):

```
ollama run llama3.2
```

Ollama предоставляет доступ к моделям с открытыми весами через OpenAI API. Поэтому можно использовать следующий код:

```
from pydantic_ai.models.openai import OpenAIModel
from pydantic_ai.providers.openai import OpenAIProvider

model = OpenAIModel(
  model_name='llama3.2',
  provider=OpenAIProvider(base_url='http://localhost:11434/v1')
)
```

Как создаются базовые модели

В отличие от традиционных систем машинного обучения, ИИ-приложения, как правило, строятся не с нуля, а на основе базовых моделей общего назначения, уже способных выполнять разнообразные задачи. Поэтому такие приложения обычно не нужно обучать, а их разработчикам не приходится разбираться в том, как работают базовые модели. Все, что мы будем обсуждать в этом разделе, необходимо нам для понимания важной терминологии.

На момент написания этой книги больше всего сведений об обучении своей модели представили создатели DeepSeek. Скорее всего, при разработке GPT, Gemini, Claude и Llama специалисты из OpenAI, Google, Anthropic и Meta действовали немного иначе, но в целом похожим образом. Поэтому мы разберем основные этапы создания базовых моделей (рис. 1.2), опираясь на данные о DeepSeek.

Прототип LLM DeepSeek прошел предварительное обучение на многоплановом высококачественном корпусе из примерно 14.8 триллиона токенов. (Для сравнения, в произведениях Шекспира содержится около 1.2 миллиона токенов, а значит, обучающий датасет для DeepSeek равен 12 миллионам шекспировских текстов.) Токены — короткие последовательности знаков — используются для обучения современных LLM и позволяют им, в отличие от более ранних предшественниц, обучавшихся на наборах слов, осваивать термины, не входящие в обычные словари, например имена собственные. При этом значение имеет не только количество: специалисты DeepSeek говорят (https://oreil.ly/5hjPn), что высоким качеством их модели обязаны тщательному отбору данных, в том числе строгим процессам дедупликации.

В результате *предварительного обучения* (pretraining) — этап 1 на рис. 1.2 — модель, анализируя большой датасет (токенов), приобретает способность понимать естественный язык. При этом главная цель разработчиков — научить LLM предугадывать следующий токен на основе контекста — предшествующих токенов (за что LLM нередко называют *предсказателями токенов*). Однако это лишь первый этап обучения.

За предварительным обучением следует *тонкая настройка с учителем* (supervised fine-tuning, SFT) — этап 2 на рис. 1.2, во время которого модели учатся более точно следовать инструкциям, а качество ответов повышается. Для этого разработчики

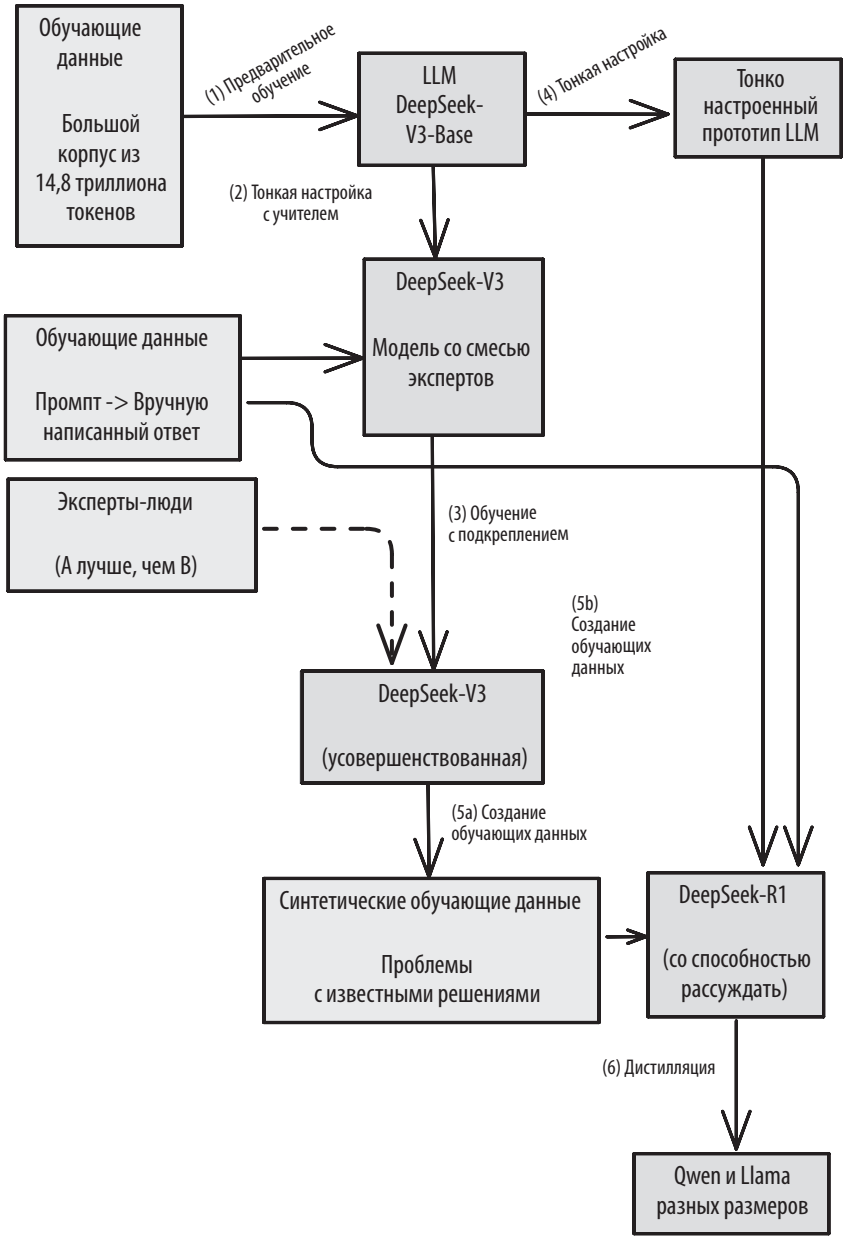

Рис. 1.2. Этапы обучения модели DeepSeek-R1

используют тщательно отобранные датасеты, составленные из подготовленных вручную материалов. Так, например, компания Cohere (https://oreil.ly/qQh-Zu) привлекает к работе лицензированных врачей, финансовых аналитиков

и бухгалтеров — специалистов, которые, как считается, могут дать на промпт идеальный ответ. Как результат, DeepSeek-V3 представляет собой модель со *смесью экспертов* (mixture of experts, MoE) — оптимизированную систему, которая при выборе каждого из токенов использует не весь обширный набор параметров, а лишь определенное их количество. К примеру, имея в общей сложности 671 миллиард параметров, DeepSeek-V3 использует только 37 миллиардов на токен (https://arxiv.org/abs/2412.19437). Благодаря этому модель может выполнять инструкции разных типов разными способами.

При *обучении с подкреплением* (reinforcement learning) — этап 3 на рис. 1.2 — LLM совершенствуется на основе человеческих предпочтений. При этом выполняется так называемое *обучение с подкреплением на основе обратной связи от человека* (reinforcement learning with human feedback, RLHF): эксперты-люди сравнивают пары выданных моделью ответов и выбирают те, которые им больше понравились. Такая *настройка под предпочтения* (preference tuning) помогает привести ответы модели в соответствие с ожиданиями и ценностями людей. Поэтому данный этап иногда называют *оптимизацией под предпочтения* (preference optimization).

При обучении DeepSeek-V3 использовался относительно небольшой объем подготовленных людьми материалов. Вместо этого, чтобы создать полноценную модель DeepSeek-R1, разработчики прибегли к следующему многоэтапному процессу (https://oreil.ly/djxoR):

1. *Холодный старт*: тонкая настройка прототипа (DeepSeek-V3-Base) с использованием начального датасета.
2. *Чистое обучение с подкреплением* (pure reinforcement learning, RL — https://oreil.ly/9G-r8) для улучшения способности к рассуждению.
3. *Выборка с отбраковкой* (rejection sampling — https://oreil.ly/b7-jt), во время которой отбирались наилучшие образцы среди данных, сгенерированных при последнем успешном обучении с подкреплением.
4. SFT, в ходе которой произошло слияние синтетических данных и данных, контролируемым образом полученных от модели DeepSeek-V3-Base. Настройка привела к улучшениям в таких областях, как самопознание, умение излагать материал и отвечать на фактические вопросы.
5. *Итоговое обучение с подкреплением* с применением разнообразных промптов и сценариев.

Наибольшим прорывом в этом процессе стал этап 2 — обучение с подкреплением, — который проводился прототипом (DeepSeek-V3-Base) *без* предварительного обучения с учителем. Такой подход позволил модели освоить рассуждения на основе *цепочки рассуждений* (chain-of-thought, CoT) и применять их для решения сложных проблем. Так появилась версия DeepSeek-R1-Zero (https://oreil.ly/lCLIZ). Оказалось, что, вопреки давно сложившемуся мнению ученых, способности LLM к рассуждению можно развить только путем обучения с подкреплением, без применения SFT. Отказ от дорогостоящих обучающих материалов, подготовленных

людьми, позволил тренировать модель дольше и на значительно более разнообразном наборе задач.

Чтобы сделать возможности DeepSeek-R1 более доступными, разработчики подготовили несколько упрощенных версий модели (https://oreil.ly/9G-r8) на основе архитектур Qwen и Llama разных размеров (1.5B, 7B, 8B, 14B и др.)[1]. Такие упрощенные модели можно запускать на более слабых компьютерах без уменьшения способности к рассуждению.

Текущее состояние базовых моделей

Экосистема базовых моделей GenAI постоянно развивается. В ней появились категории моделей с четкой специализацией, предназначенные для решения определенных задач.

Поскольку чисто научные параметры давно перестали быть показателем, а также могут быть сфальсифицированы, оценивать модели GenAI лучше всего путем попарных слепых тестов. Обширным сравнением моделей занимается платформа LMArena (https://lmarena.ai). Так, например, на рис. 1.3 показаны результаты за апрель 2025 года.

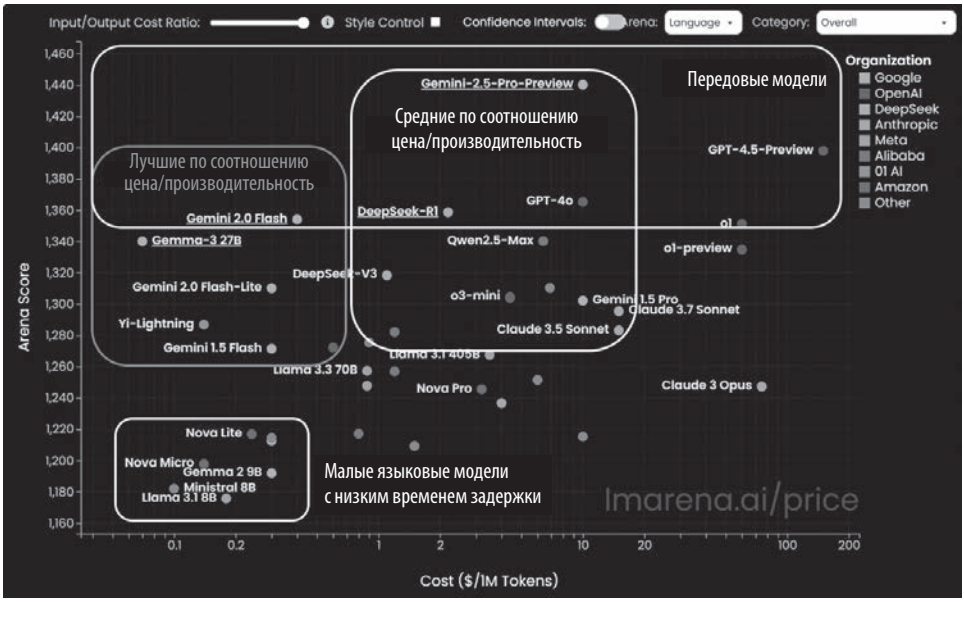

Рис. 1.3. Рейтинг LMArena от 6 апреля 2025 года (с аннотациями авторов книги)

[1] B (от billion — «миллиард») — это общепринятое обозначение для количества параметров в больших языковых моделях. — *Примеч. ред.*

На этом рисунке по вертикальной оси отложен рейтинг моделей, а по горизонтальной — их стоимость. По этим данным можно выбрать модель, оптимальную по бюджету, а также определить минимальные затраты на модель с требуемым рейтингом. Обратите внимание: масштаб по обеим осям — логарифмический[1].

На момент, когда мы сделали этот скриншот, самый высокий рейтинг был у Gemini 2.5 Pro Preview, GPT-4.5 (Preview), GPT-4o и DeepSeek-R1. Чуть ниже по вертикальной оси стояли Gemini 2.0 Flash и OpenAI o1. Цифры изменяются каждый день: появляются новые модели, делаются новые оценки. Однако в целом картина остается прежней: флагманские модели от Google, OpenAI и Anthropic всегда наверху. Совокупно их называют *передовыми моделями* (frontier models).

Передовые модели, такие как GPT 5 и Gemini 2.5 Pro, воплощая в себе последние достижения языковых моделей, являются лучшими по уровню знаний, способности рассуждать и выполнять мультимодальные задачи. Для них характерны расширенные окна контента (у некоторых моделей до двух миллионов токенов). Однако для их работы необходимо много ресурсов и денег, а в силу размеров и проприетарности нельзя запускать их локально. Поэтому их используют в приложениях корпоративного масштаба, где наибольшую важность имеют способности к рассуждению, а скорость отходит на второй план.

Упрощенные версии передовых моделей сочетают в себе производительность и эффективность, предлагая разумный уровень возможностей при меньших затратах и меньшем времени отклика. Самые яркие примеры таких моделей — Gemini Flash, Claude Sonnet и GPT-4o-mini. Как правило, они достаточно быстро справляются с распространенными задачами, например генерацией и реферированием контента, эффективны и экономичны для крупномасштабных приложений. Если учесть логарифмический масштаб по горизонтальной оси на графиках с рис. 1.3, разница между Gemini Pro и Gemini Flash по стоимости применения очевидна: Flash в 20 раз дешевле.

Параметры моделей *с открытыми весами* (open-weight), например Llama, Mistral, DeepSeek, Qwen и Falcon, общедоступны, что обеспечивает прозрачность, возможность настройки и усовершенствования. Эти модели, хотя и отстают от передовых, отличаются высокой производительностью и могут быть тонко настроены (дообучены) при помощи проприетарных датасетов. Тем не менее для хостинга таких моделей требуется определенная квалификация. Поэтому существуют специализированные платформы для хостинга LLM (например, Together.ai), а гиперскейлеры предоставляют для доступа к ним полностью управляемые эндпоинты API.

Локально запускаемые модели, к примеру Llama 8B и Gemma 2B, оптимизированные версии которых доступны через NVIDIA NIM (https://developer.nvidia.com/nim), предназначены для работы на потребительском или корпоративном оборудовании без подключения к облаку. Это обеспечивает полную конфиденциальность

[1] В данном случае это рейтинг Эло: разница в 400 пунктов говорит о том, что вероятность победы игрока с более высоким показателем составляет 10:1.

(данные не покидают локальные устройства) и отсутствие затрат на API. Однако такие модели сильно ограничены по возможностям по сравнению с облачными. Спрос на передовые модели в изолированных, отключенных от интернета системах привел к тому, что некоторые провайдеры стали предлагать подобные услуги. К примеру, Gemini может работать локально в среде Google Distributed Cloud (https://oreil.ly/HFIut), а OpenAI — в локальных развертываниях Azure (https://oreil.ly/-XSfk).

Агентный искусственный интеллект

Для обращения к моделям Claude в Pydantic API служит класс `Agent`:

```
from pydantic_ai import Agent
agent = Agent('anthropic:claude-3-7-sonnet-latest',
  ...
```

Что такое *агент*? Уже давно в computer science под этим термином понимают программную сущность, которая действует от лица пользователя или другой программы. При обращении к базовой модели вы задаете ей роль, приводите контекст и просите что-то сделать. С этой точки зрения модель выступает вашим агентом.

Вот, например, как можно создать агент для управления складскими запасами магазина (полная версия кода находится на GitHub: https://oreil.ly/d2kyK):

```
agent = Agent(
  f"anthropic:{MODEL_ID}",
  system_prompt="Ты - менеджер по складу, который делает заказы по мере
надобности.",
  ...
)
```

Автономность

Используя ИИ в качестве агента, мы ожидаем от него некоторой автономности. В качестве мозга у него LLM, а значит, мы не должны объяснять ему, как управлять складскими запасами, а только поставить *цель*: делать заказы «по мере надобности».

Допустим, у нас есть список товаров, по каждому из которых известно, как хорошо он продается и сколько длится доставка:

```
@dataclass
class InventoryItem:
  name: str
  quantity_on_hand: int
  weekly_quantity_sold_past_n_weeks: [int]
  weeks_to_deliver: int

items = [
  InventoryItem("ТоварА", 300, [50, 70, 80, 100], 2),
```

```
  InventoryItem("ТоварB", 100, [70, 80, 90, 70], 2),
  InventoryItem("ТоварC", 200, [80, 70, 90, 80], 1)
]
```

Передадим этот список агенту, и он подскажет нам, что нужно дозаказать:

```
result = agent.run_sync(f"""
Определи, какие из этих товаров нужно дозаказать на этой неделе.

**Товары**
{items}
""")
```

В результате получим, в том числе:

> ТоварB
>
> **quantity_to_order=300** reason_to_reorder='Текущий запас (100) недостаточен для покрытия планируемого спроса за время доставки. На основании данных о недавних недельных продажах (70–90 шт.) можно заключить, что следует **сделать заказ на объем, достаточный на две недели срока доставки** плюс резервный запас'.

Сравните это с традиционным программированием, где нужно писать код, объясняя компьютеру, как управлять складом. Подобная *автономность*, — то есть умение работать независимо, без постоянных указаний со стороны человека, без явно прописанного кода, — и есть основное отличие ИИ-агентов от традиционных программ.

Свойства агентов

Помимо автономности агенты обычно отличаются следующими свойствами:

Ориентация на цель

Агенты не просто отвечают на промпты, но стремятся к достижению определенных целей. Заказ товаров по мере необходимости — цель, которую мы поставили перед агентом-менеджером по складу, используя системный промпт.

Планирование и рассуждение

Смотрите, сколько работы выполнил наш агент: он вычислил, сколько товаров продано за последнее время, сделал прогноз на период доставки, определил необходимое количество товаров, а затем рассчитал, сколько их нужно дозаказать. И все это без явного программирования.

Восприятие и воздействие

Агенты могут собирать необходимые данные («воспринимать», perceive) и воздействовать на внешнее окружение. Как правило, мы можем дать им такую возможность, разрешив прибегать к внешним функциям (к примеру, калькулятору, базе данных или же поиску в интернете). Используя паттерн «Вызов инструментов» (паттерн 21 из главы 7), мы можем выйти за

рамки явного указания списка товаров: агент будет сам выяснять, сколько товаров на складе, а сколько продано, узнавая об этом из внутренней базы данных. И более того, он может не просто передавать нам список заказов, но сам обращаться к нужному API, к примеру, на сайте поставщика, для их размещения.

Адаптивность и обучение

Откуда нам знать, что заказать товар B в количестве 300 штук — верное решение? Будь менеджер по складу человеком, он посмотрел бы данные по продажам и сопоставил имеющийся запас с предполагаемым спросом. Все это по силам и агенту. В главах 6 и 9 мы познакомимся с «Рефлексией» (паттерн 18) и «Самопроверкой» (паттерн 31), при помощи которых агент может оценить проделанную работу и самостоятельно скорректировать результат.

На момент написания этой книги агентное поведение приложений на основе базовых моделей все еще является амбициозной целью: созданию полностью автономных ИИ-агентов мешают галлюцинации, недетерминированность и другие сбои. Возьмем, к примеру, недетерминированность: при каждом обращении мы будем получать разные списки товаров для заказа — и по составу, и по количеству (попробуйте!). Планирование работает только в самых простых случаях. Поэтому в главе 5 мы познакомимся с «Цепочкой рассуждений» (паттерн 13), которая улучшает способности агента планировать и рассуждать. Многие паттерны из рассмотренных в этой книге делают поведение ИИ-приложений более агентным или, по крайней мере, расширяют их возможности.

Детальное управление

Базовые модели предоставляют детальный контроль над генерацией контента за счет управления выборкой и лучевым поиском (оба процесса обсуждаются далее в этой главе). В настройках LLM есть мощные средства, которые позволяют найти компромисс между высоким качеством и детерминированностью получаемых ответов, с одной стороны, и их креативностью и разнообразием — с другой. Понимание математической основы этих настроек, от логитов до стратегий выборки, сделает управление работой модели проще, чем с помощью паттернов проектирования, о которых мы будем говорить во всех остальных главах.

Логиты

Языковые модели состоят из сотен слоев, самый последний из которых предсказывает следующее слово для продолжения генерации. При этом мы получаем не один, а сразу несколько возможных токенов, каждый из которых с определенной вероятностью может стать следующим.