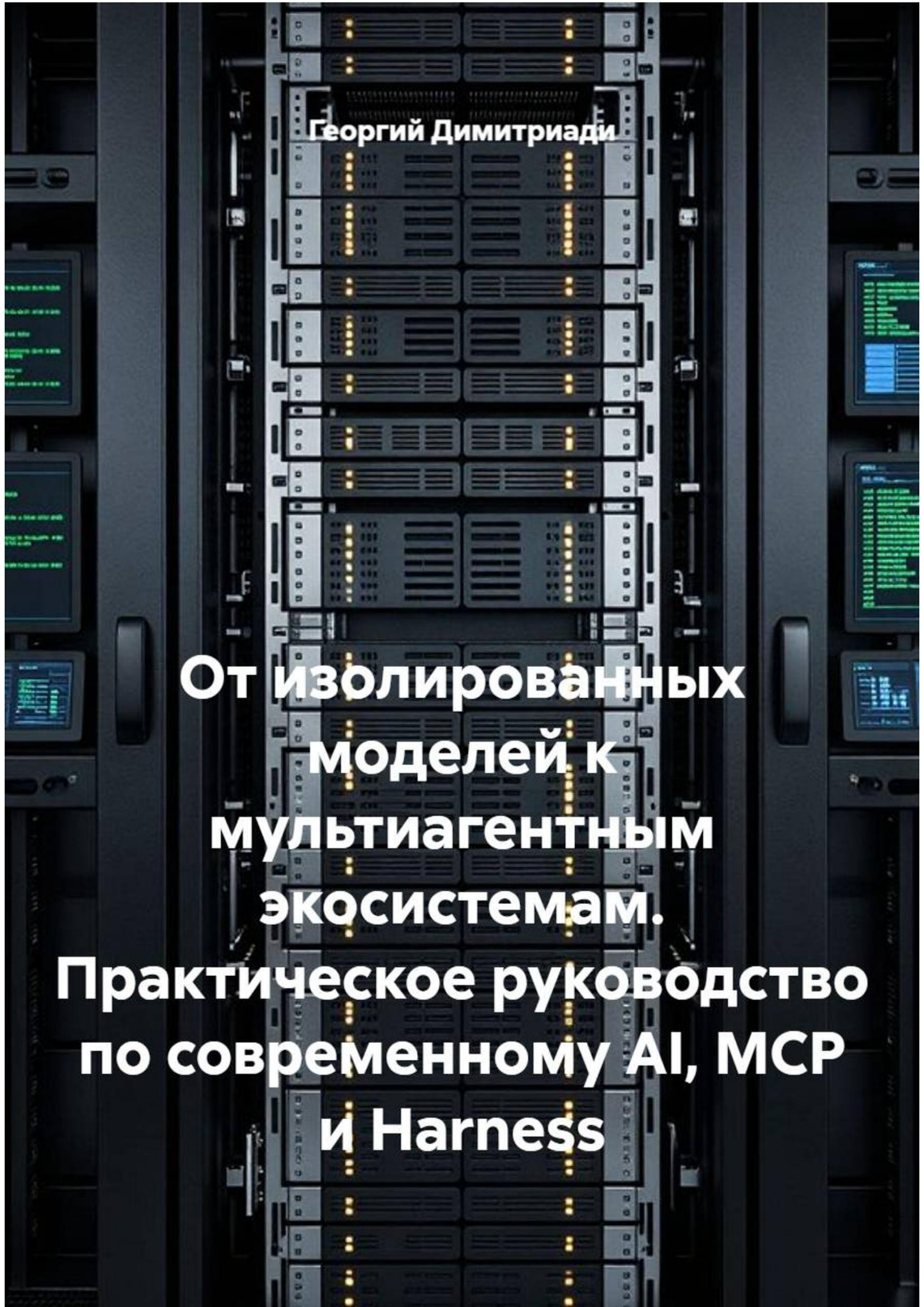




Георгий Димитриади

От изолированных
моделей к
мультиагентным
экосистемам.

Практическое руководство
по современному AI, MCP
и Harness

Георгий Димитриади

**От изолированных моделей к
мультиагентным экосистемам.
Практическое руководство по
современному AI, MCP и Harness**

«Автор»

2026

Димитриади Г.

От изолированных моделей к мультиагентным экосистемам.

Практическое руководство по современному AI, MCP и Harness /
Г. Димитриади — «Автор», 2026

Как превратить нейросеть из «умного собеседника» в автономного сотрудника? Эта книга - инженерный путеводитель по созданию промышленных ИИ-агентов. Вы освоите MCP (Model Context Protocol) - новый стандарт («USB-C для ИИ»), позволяющий безопасно подключать модели к базам данных, API и файлам без написания сотен адаптеров. Вы научитесь проектировать мультиагентные системы, где десятки специалистов работают сообща, и внедрять Eval Harness - систему жесткого тестирования, которая делает поведение ИИ предсказуемым и безопасным. В фокусе книги: архитектура протоколов, контроль затрат на токены, безопасность (OAuth 2.1) и наблюдаемость систем. Практические примеры адаптированы для российских инфраструктурных реалий. Это руководство для тех, кто хочет строить не просто умные чаты, а надежные цифровые экосистемы для бизнеса.

© Димитриади Г., 2026

© Автор, 2026

Содержание

Краткое содержание	5
Введение. Конец эпохи чат-ботов	8
Глава 1. Анатомия большой языковой модели (LLM)	10
Глава 2. От статики к динамике: RAG и инструменты	12
Глава 3. Что такое ИИ-агент	14
Глава 4. Кризис N×M: Хаос интеграций	16
Конец ознакомительного фрагмента.	17

Георгий Димитриади

От изолированных моделей к мультиагентным экосистемам.

Практическое руководство по современному AI, MCP и Harness

Краткое содержание

Введение. Конец эпохи чат-ботов

Эволюция ИИ: от поисковых запросов к действию.

Почему генерация текста - это лишь 10% возможностей современных систем.

Для кого эта книга: разработчики, архитекторы, продакт-менеджеры.

Часть 1. Фундамент: Современный искусственный интеллект

Глава 1. Анатомия большой языковой модели (LLM). Токены, контексты, трансформеры. Почему модель «галлюцинирует» и как это связано с вероятностной природой ИИ.

Глава 2. От статики к динамике: RAG и инструменты. Как научить модель работать с актуальными данными (Retrieval-Augmented Generation) и вызывать внешние функции (Function Calling). Ограничения классического подхода.

Глава 3. Что такое ИИ-агент. Петля ReAct (Reason + Act). Разница между моделью, которая отвечает на вопрос, и агентом, который решает задачу. Компоненты агента: мозг (LLM), память (краткосрочная и долгосрочная), инструменты.

Пример: Создание простого агента на Python, который умеет проверять погоду и сохранять заметки в локальный файл.

Часть 2. Проблема масштаба: Хаос интеграций

Глава 4. Кризис N×M. Почему подключение 10 агентов к 10 корпоративным системам (CRM, базы данных, Slack, файловое хранилище) превращается в неподдерживаемый кошмар. Проблема фрагментации API.

Глава 5. Знакомство с MCP (Model Context Protocol).

Теория: MCP как «USB-C для ИИ». История создания Anthropic и передача в Linux Foundation (Agentic AI Foundation) в декабре 2025 года.

Архитектура: Host, Client, Server. Транспорты (stdio для локальных задач и Streamable HTTP для облака).

Примитивы: Tools (действия), Resources (данные), Prompts (готовые шаблоны).

Отличие от Function Calling: MCP не просто вызывает функцию, он позволяет агенту динамически обнаруживать доступные инструменты и управлять сессиями через JSON-RPC 2.0.

Пример: Переписывание агента из Главы 3. Теперь он не имеет жестко зашитого кода для работы с файлами, а подключается к MCP-серверу файловой системы. Агент из коробки получает доступ к новому MCP-серверу базы данных без изменения своей логики.

Часть 3. Индустриальный стандарт: Глубокое погружение в MCP

Глава 6. Жизненный цикл MCP-соединения. Рукопожатие (handshake), листинг возможностей, вызов инструмента, обработка ошибок.

Глава 7. Безопасность и авторизация в MCP. Внедрение OAuth 2.1 в спецификацию 2025 года. Как ограничить агенту доступ только к нужным таблицам или каналам. Stateless-ядро протокола.

Глава 8. Экосистема MCP. Официальный реестр серверов. Тысячи готовых коннекторов (Google Drive, Slack, GitHub, корпоративные ERP). Как опубликовать свой MCP-сервер.

Пример: Сборка корпоративного хаба. Агент технической поддержки через MCP подключается одновременно к Jira (тикеты), Confluence (база знаний) и внутренней CRM, используя единый протокол авторизации.

Часть 4. Гарантия качества: Harness для агентов

Глава 9. Почему агенты непредсказуемы. Проблема оценки неструктурированного вывода. Что такое Harness (обязка) в контексте ИИ.

Глава 10. Анатомия Eval Harness.

Инфраструктура как данные: описание окружения, лимитов времени и токенов.

Rule-based checks: детерминированные проверки (формат JSON, regex, наличие полей).

LLM-as-a-judge: использование второй модели для оценки семантики, читаемости и отсутствия галлюцинаций по рубрикам.

Глава 11. Harness Bench и воспроизводимость. Как создать тестовый набор (бенчмарк), который можно прогонять при каждой смене промпта или версии модели. Гибкая архитектура, отделяющая логику задачи от правил оценки.

Пример: Пишем Harness для агента-программиста. Автотест проверяет, скомпилировался ли код, прошел ли он unit-тесты, и оценивает чистоту архитектуры с помощью LLM-судьи.

Часть 5. Синергия: Мультиагентные среды

Глава 12. Один агент - хорошо, а сеть - лучше. Пределы когнитивных способностей одной модели. Делегирование задач.

Глава 13. Паттерны взаимодействия.

Координатор (Orchestrator) - Исполнители (Workers).

Peer-to-peer: агенты договариваются между собой.

Иерархия и рынок: агенты как MCP-серверы для других агентов. Агент может предоставлять свои навыки (например, навык сложного анализа данных) другим агентам через протокол MCP.

Глава 14. Коммуникация и память в мультиагентных системах. Протоколы обмена сообщениями. Общая «рабочая доска» (blackboard). Разрешение конфликтов.

Пример: Разработка системы для вывода нового продукта на рынок.

Агент-исследователь собирает данные через MCP-серверы аналитики.

Агент-копирайтер пишет тексты, запрашивая у дизайнера (другого агента) мокапы.

Агент-менеджер (Orchestrator) распределяет задачи и контролирует бюджет, общаясь с агентами через MCP.

Часть 6. Производство и этика

Глава 15. Мониторинг и наблюдаемость. Как понять, на каком шаге мультиагентная цепочка пошла не так. Трассировка вызовов MCP.

Глава 16. Экономика агентов. Подсчет стоимости токенов в сложных цепочках. Оптимизация.

Глава 17. Этика, безопасность и автономия. Проблема «вредоносных инструкций» во внешних данных, к которым агент получает доступ через MCP. Кто несет ответственность за действие агента.

Заключение. Будущее агентного ИИ

От инструментов к полноценным цифровым сотрудникам.

Тренды 2026 года и далее: физическое воплощение агентов (роботы), децентрализованные сети.

Приложения

Глоссарий (MCP Host, Harness, ReAct, LLM-as-a-judge).

Чек-лист запуска мультиагентной системы в продакшен.

Список ключевых открытых библиотек (MCP SDK, Inspect AI, Databricks Evaluation Framework).

Введение. Конец эпохи чат-ботов

Осенью 2022 года мир сошел с ума от больших языковых моделей. Мы научились общаться с компьютерами на естественном языке, и это казалось вершиной технологического прогресса. Мы спрашивали - нейросеть отвечала. Мы просили написать код - она выдавала сниппет. Это было похоже на магию.

Но магия быстро столкнулась с суровой реальностью. Выяснилось, что модель, способная написать безупречное эссе о квантовой физике, не может забронировать вам отель, обновить статус в Jira или найти ошибку в логах сервера за вчерашний день. Она заперта в 2022 или 2024 году, в зависимости от даты среза ее знаний. Она не знает о ваших корпоративных документах, не видит вашего экрана и не имеет права нажать кнопку «Оплатить» на сайте.

Промышленность уперлась в потолок. Чтобы сделать следующий шаг, модели нужно было дать руки.

Так наступила эра агентного искусственного интеллекта (ИИ). Мы перестали просить ИИ *говорить* и начали просить его *делать*. ИИ-агент - это уже не просто нейросеть, это система, обладающая петлей обратной связи: она может проанализировать задачу, составить план, обратиться к внешнему инструменту, получить результат, скорректировать свои действия и попробовать снова. Агент не просто генерирует текст - он решает проблему в реальном мире.

Однако, как только мы начали массово создавать таких агентов, индустрия погрузилась в хаос.

Представьте крупную компанию. У вас есть десять ИИ-агентов: один пишет код, второй отвечает клиентам, третий анализирует финансы. И есть двадцать корпоративных систем: база данных PostgreSQL, CRM Salesforce, файловое хранилище, Slack, GitHub, ERP 1С. Чтобы агент мог работать, разработчику приходилось писать уникальный, хрупкий «клей» - адаптер - для каждой пары «агент - система». Если систем 20, а агентов 10, вам потенциально нужно 200 различных интеграций. Любое изменение API в Salesforce ломало агента по продажам. Обновление базы данных обрушивало аналитика. Разработка превратилась в кошмар поддержки.

В ноябре 2024 года этот хаос начал уходить в прошлое. Компания Anthropic представила протокол **Model Context Protocol (MCP)**- открытый стандарт, который индустрия мгновенно окрестила «USB-C для искусственного интеллекта».

MCP делает для ИИ то же самое, что когда-то HTTP сделал для веба. Он предлагает единый, универсальный язык, на котором любой агент может общаться с любым внешним инструментом. Больше не нужно писать 200 адаптеров. Вы создаете один MCP-сервер для вашей базы данных и один MCP-клиент для вашего агента. После этого любой совместимый агент - будь то модель от OpenAI, Anthropic или открытая LLM, запущенная на вашем сервере - получает доступ к этим данным мгновенно и безопасно. В декабре 2025 года Anthropic передала MCP в дар фонду Agentic AI Foundation под эгидой Linux Foundation, навсегда сделав его нейтральным общественным стандартом.

Но дать агентам инструменты - это только половина дела. Когда система обретает автономию и начинает совершать реальные действия (менять код, переводить деньги, удалять файлы), цена ошибки возрастает до стоимости бизнеса. Как гарантировать, что агент не «галлюцинирует» в процессе выполнения критической операции? Как протестировать цепочку из пяти разных инструментов? Как убедиться, что после обновления модели система не начнет нарушать регламент?

Для этого появился промышленный стандарт **Harness**- инженерная обвязка, которая превращает непредсказуемое поведение нейросети в повторяемый, измеряемый и тестируемый процесс. Harness прогоняет агента через фиксированные сценарии, проверяет не только фор-

мат его ответа, но и смысловую корректность, используя для этого как жесткие правила, так и других ИИ-судей.

Эта книга - ваш путеводитель по новому стеку технологий, который определит развитие IT на ближайшие десять лет. Мы не будем просто обсуждать теорию трансформеров. Мы пройдем путь от создания первого простейшего агента до проектирования сложных мультиагентных экосистем, где десятки специализированных цифровых сотрудников работают сообща, обмениваясь данными и навыками через протокол MCP.

Вы узнаете, как устроена архитектура MCP на уровне JSON-RPC, как правильно проектировать безопасную авторизацию через OAuth 2.1, как писать Harness для автоматической оценки качества агентов и как собирать из этого конструктора системы, которые еще три года назад казались научной фантастикой.

Эпоха, где ИИ был лишь умным собеседником, закончилась. Начинается эра ИИ, который работает. Пристегните ремни.

Глава 1. Анатомия большой языковой модели (LLM)

Прежде чем мы научим искусственный интеллект управлять нашими компьютерами, базами данных и бизнес-процессами, мы должны безжалостно развеять главную иллюзию, с которой сталкивается каждый новичок. Большая языковая модель - это не разумное существо. У нее нет убеждений, желаний или сознания. В своей основе LLM - это чудовищно сложная, но математически предсказуемая машина по угадыванию следующего слова.

Что такое токен и почему модель не видит букв

Когда вы пишете модели фразу «Привет, как дела?», она не читает ее так, как это делает человек. Модель не понимает, что «Привет» - это приветствие, а знак вопроса означает запрос на ответ. Для нее ваш текст - это просто последовательность чисел.

Сначала текст разбивается на **токены**. Токен - это не всегда целое слово. Популярные алгоритмы (например, Byte Pair Encoding) дробят слова на наиболее частые слоги или сочетания символов. Слово «приветствие» может быть разбито на токены: прив, ет, ствие. Английское слово *unbelievable* превратится в un, believ, able.

Каждому токenu из огромного словаря модели присваивается уникальный идентификатор (число). Ваша фраза превращается в массив чисел, например: [1234, 5678, 90, 12]. Именно с этими числами и работает нейросеть.

Архитектура: Как числа превращаются в смыслы

В основе современных LLM лежит архитектура **Трансформер** (Transformer), представленная в 2017 году. Ее главный секретный ингредиент - механизм **внимания** (self-attention).

Представьте, что вы читаете предложение: «Я положил кошку в сумку, потому что она была тяжелой». Человек понимает, что «она» относится к сумке, а не к кошке. Механизм внимания позволяет модели устанавливать связи между токенами на огромном расстоянии друг от друга. При обработке каждого токена модель «взвешивает» важность всех остальных токенов в контексте. Она понимает, что в данном случае слово «тяжелой» имеет гораздо более сильную связь со словом «сумка», чем со словом «кошка».

Именно благодаря этому механизму модель улавливает синтаксис, контекст и базовые причинно-следственные связи, которые мы привыкли называть «смыслом».

Контекстное окно: Память с золотыми слитками

У модели есть оперативная память, которую называют **контекстным окном**. Это максимальное количество токенов (ваших слов + слов ответа модели), которые она может удерживать в «голове» одновременно.

В 2023 году контекстное окно в 4 000 токенов казалось прорывом. В 2025 году стандартом стали окна в 128 000, 200 000 и даже 1 миллион токенов.

Однако контекст не бесплатен. Чем больше информации вы загружаете в начало диалога, тем меньше места остается для ответа. Кроме того, существует «эффект края»: модель лучше всего помнит начало и конец контекста, но имеет тенденцию «забывать» или игнорировать информацию из середины длинного документа. Умение инженера упаковать самую важную информацию в начало или конец запроса - это уже половина успеха в работе с агентами.

Почему модель «галлюцинирует»

Галлюцинации - это не программная ошибка. Это фундаментальное свойство архитектуры.

Когда модель заканчивает анализировать ваш контекст, ее единственная задача - предсказать **вероятностное распределение** следующего токена. Она не знает правильного ответа. Она знает, какое слово статистически должно идти следующим на основе миллиардов прочитанных текстов.

Если в данных для обучения было мало информации о редком научном термине или о ваших внутренних корпоративных регламентах, модель все равно выдаст самый вероятный, по ее мнению, токен. Она будет врать уверенно, логично и безупречным литературным языком, потому что с точки зрения математики она просто продолжает наиболее вероятную последовательность слов.

Именно поэтому агентный подход, который мы начнем разбирать со следующей главы, так важен. Агент не просто верит ответу модели. Он использует LLM для генерации плана, а затем проверяет каждый шаг этого плана с помощью внешних инструментов (через тот самый MCP), сверяясь с реальностью.

Температура и детерминизм

На последнем этапе, когда модель уже посчитала вероятности всех возможных следующих слов, в дело вступает параметр **температуры**.

Температура 0: Модель всегда будет выбирать самый вероятный токен. Это делает ее предсказуемой, сухой и идеальной для написания кода или заполнения JSON-структур. Если вы дадите ей один и тот же запрос дважды, вы получите идентичный ответ.

Температура выше 0 (например, 0.7): Модель начинает иногда выбирать менее вероятные, но более творческие варианты. Ответы становятся разнообразными, живыми, но возрастает риск ошибок.

Для классических чат-ботов креативность приветствуется. Для строительных блоков агентных систем (где модель должна сгенерировать строгое название функции или точное число) температура всегда должна быть опущена до нуля.

Понимание этой анатомии - что модель считает токены, теряет контекст в середине окна и просто угадывает следующее слово - критически важно. Это лишает ИИ мистического ореола и превращает его из «цифрового бога» в мощный, но требующий точной настройки инженерный инструмент. Теперь, когда мы знаем, как работает мозг, мы готовы собрать вокруг него тело - автономного агента.

Глава 2. От статики к динамике: RAG и инструменты

Если первая глава убедила вас в том, что большая языковая модель - это просто очень продвинутый автодополнитель текста, то перед вами неизбежно встанет вопрос: как заставить эту статистическую машину приносить реальную пользу? Как сделать так, чтобы она знала о ваших новых продуктах, выпущенных вчера, или о пункте 4.2 в регламенте, который вы обновили час назад?

Ответ кроется в переходе от статической памяти к динамическому взаимодействию с внешним миром. В 2025 году существует два фундаментальных способа расширить возможности «голой» модели: **Retrieval-Augmented Generation (RAG)** и **Function Calling (вызов инструментов)**.

RAG: Дешевая память для гиганта

RAG - это технология, позволяющая модели отвечать на вопросы, опираясь на конкретную базу знаний, не переобучая саму нейросеть. Обучение или дообучение LLM стоит миллионы долларов и занимает недели. RAG работает за миллисекунды и стоит копейки.

Процесс состоит из двух этапов, скрытых от пользователя:

Индексация (происходит заранее): Вы берете свои документы (PDF-файлы, инструкции, страницы Confluence, записи в базе данных), разбиваете их на небольшие смысловые куски (чанки) и с помощью специальной нейросети (эмбединг-модели) превращаете каждый кусок в вектор - длинный список чисел, описывающий смысл текста. Эти векторы сохраняются в специальной векторной базе данных (Vector DB).

Генерация (происходит в момент запроса):

Когда вы задаете вопрос, модель превращает ваш вопрос в такой же вектор.

Система ищет в векторной базе данных куски текста, чьи векторы максимально близки к вектору вашего вопроса (семантический поиск). Это гораздо умнее обычного поиска по ключевым словам: модель поймет, что «как вернуть товар» и «процедура обмена» - это об одном и том же.

Найденные куски текста (релевантный контекст) автоматически подклеиваются к вашему исходному запросу.

Только после этого вся эта конструкция отправляется в основную LLM.

В промте, который видит модель, скрыто негласное правило: *«Вот контекст, ты обязан отвечать только на его основе. Если ответа в контексте нет - скажи, что не знаешь»*.

Ограничение RAG: RAG отлично справляется с ролью энциклопедии. Но модель по-прежнему остается пассивным наблюдателем. Она может сказать вам, сколько денег на счету по состоянию на вчерашний вечер (если эти данные есть в базе), но она не может сама перевести эти деньги или забронировать на них билет.

Function Calling: Рождение рук

Чтобы агент мог действовать, миру понадобился **Function Calling** (вызов функций). Это механизм, который позволил LLM выйти за пределы генерации текста.

Технически это выглядит так: разработчик описывает модели доступные функции на языке JSON. Например: `send_email(to, subject, body)` или `get_exchange_rate(currency)`.

Когда вы просите агента «Напиши письмо Ивану о том, что встреча переносится», происходит следующее:

LLM анализирует запрос и понимает, что для выполнения задачи нужно не писать текст ответа вам, а вызвать функцию `send_email`.

Вместо обычного текста модель возвращает структурированный JSON-объект: `{ "name": "send_email", "arguments": { "to": "ivan@example.com", "subject": "Встреча", "body": "Коллеги, встреча переносится..." } }`.

Ваша программная обвязка (код, в котором живет агент) перехватывает этот JSON, выполняет реальную функцию в операционной системе или API почтового сервиса, получает результат (например, `success: true`) и возвращает его модели.

Только теперь модель пишет вам человеческим языком: «Готово, письмо отправлено».

Ограничение Function Calling: На этом этапе агент становится похож на сороку, таскающую в гнездо всё блестящее. Разработчики начинают подключать десятки функций: для работы с календарем, для запроса к SQL-базе, для публикации в Telegram, для парсинга сайтов. Логика выбора нужного инструмента начинает захламлять системный промт. Каждая новая интеграция требует написания уникального кода-переходника. Мы снова возвращаемся к проблеме хаоса $N \times M$, о которой говорили во введении.

Почему RAG и Function Calling - это только фундамент

В современной архитектуре агентов RAG и Function Calling - это базовые навыки, которые мы даем модели «из коробки». Но они не решают проблему *динамики масштаба*.

Представьте, что вашему агенту нужно работать с пятью разными базами данных, двумя CRM и корпоративным мессенджером.

В классическом подходе вам придется описать структуру каждой базы данных в промте и написать пять разных Python-функций для подключения.

При любом изменении схемы в базе данных ваш агент сломается, потому что его «карта мира» в голове перестанет соответствовать реальности.

Именно здесь пролегает граница между простым «умным чат-ботом с инструментами» и по-настоящему автономным агентом, готовым к промышленной эксплуатации. Чтобы пересечь эту границу, нам нужен не просто вызов функций, а **протокол** - единый стандарт, который позволит агенту не просто выполнять команды, а самостоятельно исследовать доступные ему инструменты, понимать их возможности и безопасно обмениваться данными.

В следующей главе мы разберем, как протокол MCP превращает разрозненные RAG-источники и функции в единую, масштабируемую и безопасную экосистему, где агент может подключаться к новым инструментам так же легко, как мы вставляем флешку в USB-порт.

Глава 3. Что такое ИИ-агент

В главах 1 и 2 мы разобрали мозг (LLM) и способы дать ему память и руки (RAG и Function Calling). Теперь пришло время собрать из этого полноценного робота. В экосистеме современного ИИ этот робот называется **агентом**.

Главное отличие агента от обычной языковой модели заключается в наличии **петли обратной связи**. Обычная модель работает по принципу «вопрос - ответ - конец». Агент работает по принципу «вопрос - план - действие - проверка - корректировка». Он не просто генерирует текст, он исполняет итеративный цикл, пока не достигнет поставленной цели.

Петля ReAct: Сердцебиение агента

Фундаментальным паттерном поведения любого современного агента является архитектура **ReAct**(Reason + Act - Рассуждение + Действие). Это внутренний монолог, который агент ведет сам с собой на каждом шаге.

Когда вы ставите перед агентом сложную задачу, например: *«Проанализируй вчерашние продажи, найди самый популярный товар и забронируй на складе 100 единиц его запаса»*, агент не пытается сделать это одним прыжком. Он дробит реальность на шаги:

Reason (Рассуждение): «Мне нужно сначала получить данные о продажах за вчера. У меня нет этих данных в контексте. Я должен использовать инструмент запроса к базе данных».

Act (Действие): Агент вызывает функцию `get_sales_data(date: "2025-10-03")`.

Observation (Наблюдение): Инструмент возвращает JSON с таблицей продаж. Агент получает этот результат.

Reason (Рассуждение): «В данных вижу, что товар "Термокружка X" продан в количестве 500 штук, это больше всего. Теперь мне нужно проверить остатки на складе».

Act (Действие): Вызов функции `check_inventory(item: "Термокружка X")`.

Observation (Наблюдение): «На складе 5000 единиц, лимит бронирования не превышен».

Act (Действие): Вызов функции `reserve_stock(item: "Термокружка X", amount: 100)`.

Final Answer (Финальный ответ): «Задача выполнена. Забронировано 100 единиц Термокружки X».

Этот цикл может повторяться десятки раз. Если на шаге 6 агент увидит, что на складе всего 10 единиц, его рассуждение изменится: «Товара мало. Я должен отменить задачу бронирования и сообщить пользователю, что запрос на 100 единиц невыполним». Именно эта способность остановиться, подумать и сменить стратегию делает систему агентом, а не просто скриптом.

Анатомия агента: Больше, чем просто модель

Чтобы петля ReAct работала в реальном мире, агенту недостаточно одной нейросети. Современный ИИ-агент - это комплексная система, состоящая из четырех ключевых компонентов:

Мозг (LLM): Координатор. Он отвечает за этап *Reason*. Он читает цель пользователя, интерпретирует результаты действий (*Observation*) и решает, какой инструмент вызвать следующим (*Act*).

Инструментарий (Tools): Руки и глаза. Это те самые функции и внешние API, к которым агент имеет доступ. Важно понимать: агент не знает, как работает код внутри функции. Он знает только название инструмента и описание того, что этот инструмент делает (его «спецификация»).

Память (Memory): Это самый сложный и важный компонент, который часто путают с контекстным окном модели. Память агента делится на три уровня:

Краткосрочная память:Стек сообщений в текущем диалоге. То, что мы обсуждаем прямо сейчас.

Рабочая память (Episodic):Результаты последних выполненных действий. «Я только что запросил данные о продажах, вот они».

Долгосрочная память:Внешние хранилища (базы данных, векторные базы вроде Pinecone или Chroma). Здесь агент хранит профиль пользователя, историю его прошлых запросов или важные факты о мире, которые не влезают в контекстное окно.

Планировщик (Planner):У продвинутых агентов между вашим запросом и первым шагом ReAct стоит еще один слой. Получив сложную задачу, модель сначала генерирует план из 5–10 высокоуровневых шагов, показывает его пользователю (или выполняет молча) и только затем начинает исполнять их по очереди. Это спасает от ситуации, когда агент уходит в бесконечный цикл неэффективных действий.

Пример: Агент-помощник «Марсианин»

Давайте разберем простейший агент, написанный на Python, который умеет делать две вещи: искать информацию в Википедии и складывать числа.

Пользователь пишет:«Кто был первым человеком на Марсе и сложи год его рождения с 10».

Внутренняя работа агента:

Шаг 1 (Анализ):Мозг (LLM) видит два запроса. Он понимает, что для первой части нужна Википедия, для второй - калькулятор.

Шаг 2 (Действие 1):Агент вызывает инструмент `search_wikipedia(query: "первый человек на Марсе")`.

Шаг 3 (Наблюдение 1):Инструмент возвращает текст: «На данный момент нога человека не ступала на поверхность Марса. В 1976 году на Марс сел аппарат "Викинг-1"».

Шаг 4 (Рассуждение):«Ага, человека там не было. Значит, года рождения нет. Я должен сообщить пользователю, что вторая часть задачи невыполнима, так как нет исходных данных».

Шаг 5 (Финальный ответ):«Первым человеком на Марсе никто не был, поэтому я не могу выполнить математическую операцию, так как отсутствует год рождения. Однако в 1976 году на планету сел аппарат "Викинг-1"».

Обратите на критическую деталь: агент не стал галлюцинировать имя (например, выдумывать «Алексея Иванова») и не стал ошибаться в логике. Благодаря циклу ReAct он честно признал отсутствие данных.

Именно этот переход - от «модели, которая пытается угадать ответ» к «агенту, который признает границы своих знаний и идет искать истину во внешний мир» - является главным качественным скачком в искусственном интеллекте последних лет.

Однако по мере того, как количество таких «Марсиан» в вашей компании растет, а количество инструментов (Slack, SQL, Google Drive, GitHub) увеличивается, вы столкнетесь с новой проблемой. Описание каждого инструмента в промте начнет занимать десятки тысяч токенов, а код для поддержки сотен API-коннекторов станет неподъемным. В следующей главе мы увидим, как протокол MCP спасает разработчиков от этого инфраструктурного ада, превращая зоопарк инструментов в единый, понятный стандарт.

Глава 4. Кризис N×M: Хаос интеграций

В предыдущей главе мы создали идеального цифрового сотрудника. Наш агент умеет рассуждать, пользоваться инструментами и хранить память. В рамках одного проекта или небольшого скрипта это выглядит как триумф инженерной мысли. Но как только мы пытаемся внедрить таких агентов в реальную, зрелую компанию, романтика заканчивается и начинается суровая математика инфраструктурного кошмара.

Этот кошмар имеет имя: **проблема N×M**.

Анатомия взрыва

Представьте типичную корпорацию 2025 года.

У вас есть Nразличных ИИ-инициатив (агентов). Например:

Агент-юрист, проверяющий договоры.

Агент-финансист, г

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.