ГЛАВА 1

Знакомство с агентами

Мы являемся свидетелями глубокого технологического преобразования, движущей силой которого стали автономные агенты — интеллектуальные программные системы, способные независимо рассуждать, принимать решения и эффективно взаимодействовать с динамическим окружением. В отличие от традиционных программных продуктов, автономные агенты интерпретируют контексты, адаптируются к изменяющимся ситуациям и выполняют нетривиальные действия с минимальным контролем со стороны человека.

Что такое ИИ-агенты?

Автономные агенты — интеллектуальные системы, спроектированные для независимого анализа данных, интерпретации своего окружения и принятия решений с учетом контекста. С ростом популярности слова «агент» его смысл размывается, так что термин часто применяется к системам, в которых отсутствует подлинная автономия. На практике концепция агента существует в некотором смысловом спектре. Истинно автономные агенты демонстрируют принятие осмысленных решений, рассуждение с учетом контекста и адаптивное поведение. В противоположность этому, многие системы, которые называются «агентами», могут просто выполнять детерминированные скрипты или жестко контролируемые рабочие потоки. Проектирование действительно автономных, адаптивных агентов — достаточно сложная задача, которая заставляет многие команды искать более простые пути для достижения быстрых результатов. Таким образом, ключевым критерием истинного агента становится его способность принимать полноценные решения, вместо того чтобы выполнять статические скрипты.

Быстрая эволюция автономных агентов прежде всего обусловлена достижениями в области фундаментальных моделей (foundation model) и обучения с подкреплением (reinforcement learning). Если традиционные сценарии использования с фундаментальными моделями ориентировались на генерацию результатов, понятных для человека, последние разработки позволяют этим моделям генерировать структурированные сигнатуры функций и выбирать параметры. Затем оркестрирующие фреймворки могут выполнять эти функции: давать возможность агентам искать данные, взаимодействовать с внешними системами и выполнять конкретные действия. В этой книге мы будем использовать термин «агентная система» для описания полной поддерживающей функциональности, позволяющей агенту эффективно выполняться, включая инструментарий, память, фундаментальную модель, средства оркестрации и инфраструктуру поддержки.

С ростом количества таких протоколов, как протокол контекста модели (MCP, model context protocol) (рассматриваемый в главе 4) и протокол «агент — агент» (рассматриваемый в главе 8), эти агенты смогут пользоваться удаленными инструментами и сотрудничать с другими агентами для решения проблем. При этом открываются колоссальные возможности для сложной автоматизации, но также появляется серьезная ответственность за проектирование, оценку и продуманное управление этими системами, гарантирующее, что действия агентов будут соответствовать человеческим ценностям и требованиям безопасности, предъявляемым к работе в сложных динамических средах.

Революция в предварительном обучении

Хотя традиционное машинное обучение (ML) — невероятно мощная методология, обычно она ограничивается количеством и качеством датасетов. Скорее всего, практик ML скажет, что он проводит бóльшую часть своего времени не за обучением моделей, а за сбором и очисткой датасетов, которые он сможет использовать для обучения. Невероятный успех генеративных моделей, обученных на больших объемах данных, показал, что в наши дни одиночные модели могут адаптироваться к широкому спектру задач без дополнительного обучения. В этом отношении они меняют многолетнюю практику. Для построения приложения, использующего ML, ранее приходилось нанимать ML-инженеров или специалистов data science, которые собирали данные, а затем развертывали полученную модель. С новейшими достижениями в области больших, предварительно обученных (pretrained) генеративных моделей для получения качественной модели, достаточно хорошо работающей во многих практических ситуациях, может быть достаточно одного вызова к уже готовой модели без какого-либо самостоятельного обучения или развертывания. Таким образом кардинально снижаются затраты и сложность создания приложений, использующих машинное обучение и искусственный интеллект.

Последние достижения в области больших языковых моделей (large language models, LLM) — таких как GPT-5, Claude (Anthropic), Llama (Meta), Gemini Ultra (Google) и V3 (DeepSeek) — еще сильнее увеличили производительность в широком спектре сложных вопросов, расширив область задач, решаемых при помощи предварительно обученных моделей. Такие фундаментальные модели обеспечивают стабильное понимание естественного языка и генерацию контента, наращивая функциональность агентов благодаря следующим возможностям:

- *Понимание естественного языка* (natural language understanding).
- Интерпретация и интуитивное реагирование на пользовательский ввод.
- *Контекстно-осведомленное взаимодействие* (context-aware interaction).
- Удержание контекста для получения релевантных и точных ответов в рамках продолжительных диалогов.
- *Генерация структурированного контента* (structured content generation).

Создание текста, кода и структурированного вывода, необходимых для аналитических и творческих задач.

Хотя сами по себе эти модели очень функциональны, они также могут использоваться для принятия решений в четко определенных областях, адаптироваться к новой информации и пользоваться инструментами для выполнения реальной работы. Интеграция с нетривиальными оркеструющими фреймворками позволяет моделям напрямую взаимодействовать с внешними системами и выполнять практические задачи. В частности, имеются следующие возможности:

Контекстная интерпретация и принятие решений

Разрешение неоднозначных ситуаций без утомительного предварительного программирования.

Использование инструментов

Вызовы других программных продуктов для извлечения информации или выполнения действий.

Адаптивное планирование

Планирование и автономное выполнение сложных многоэтапных действий.

Обобщение информации

Быстрая обработка объемных документов и формирование ключевых выводов упрощают юридический анализ, позволяют объединить результаты исследований и обеспечить отбор контента.

Управление неструктурированными данными

Интерпретация и интеллектуальное реагирование на неструктурированные тексты (например, сообщения электронной почты, документы, журналы и отчеты).

Генерация кода

Написание и выполнение кода и создание юнит-тестов.

Автоматизация рутинных задач

Эффективная обработка повторяющихся действий в рабочих процессах технической поддержки и администрирования, чтобы работники-люди могли сосредоточиться на более сложных задачах.

Объединение мультимодальной информации

Выполнение сложного анализа графических, звуковых и видеоданных в большом масштабе.

Такая гибкость позволяет автономным агентам эффективно справляться со сложными и динамическими ситуациями, которые обычно не могут быть решены при помощи статических ML-моделей.

Типы агентов

С ростом популярности термина «агент» его смысл расширился и стал охватывать широкий спектр систем с поддержкой ИИ. Все это приводит к путанице относительно того, что же в действительности собой представляет ИИ-агент. Интернет-ресурс The Information делит агентов на семь практических типов (https://oreil.ly/99_ZV) в соответствии с тем, как эти технологии применяются в наши дни:

Агенты бизнес-задач (business-task agents)

Такие агенты автоматизируют предварительно определенные рабочие потоки бизнес-задач (например, автоматизация роботизированных процессов UiPath, low-code-процессы в Microsoft Power Automate или интеграции приложений Zapier). Они выполняют последовательности детерминированных действий, которые обычно инициируется по событиям и требуют минимального контекстного анализа.

Разговорные агенты (conversational agents)

К этой категории относятся чат-боты и агенты поддержки клиентов, общающиеся с пользователями через интерфейсы естественных языков. Они оптимизированы для управления диалогом, распознавания намерений и чередования высказываний в разговоре (например, виртуальные ассистенты, встроенные в платформы поддержки клиентов).

Исследовательские агенты (research agents)

Исследовательские агенты занимаются сбором информации, синтезом и обобщением задач. Они сканируют документы, базы знаний или веб-ресурсы для получения структурированных результатов, упрощающих работу аналитиков. Примеры — Perplexity AI и Elicit.

Аналитические агенты (analytics agents)

Аналитические агенты (такие как Power BI Copilot или Glean) сосредоточены на интерпретации структурированных наборов данных и генерации выводов, дашбордов и отчетов. Они часто интегрируются с корпоративными хранилищами данных (data warehouses), позволяя пользователям формулировать сложные запросы на естественном языке.

Агенты разработки (developer agents)

Такие инструменты, как Cursor, Windsurf и GitHub Copilot, упрощают задачу разработчика своими возможностями генерации, рефакторинга и объяснения кода. Они интегрируются в рабочие процессы IDE для повышения эффективности разработки.

Агенты предметных областей (domain-specific agents)

Эти агенты оптимизированы для специализированных профессиональных областей — как, например, юридические (Harvey), медицинские (Hippocratic AI) или финансовые агенты. Они объединяют знания, относящиеся к конкретной

предметной области, со структурированными рабочими потоками для предоставления целенаправленной поддержки экспертного уровня.

Браузерные агенты (browser-using agents)

Эти агенты способны перемещаться по веб-сайтам, взаимодействовать с ними, извлекать информацию и совершать действия без участия человека. В отличие от традиционной автоматизации процессов, которая следует заранее заданным сценариям, современные браузерные агенты объединяют понимание языка, визуальное восприятие и динамическое планирование, чтобы адаптироваться к изменениям в реальном времени.

Кроме этих семи категорий агентов, также важны голосовые и видеоагенты. Предполагается, что в ближайшие годы они получат более широкое распространение.

Голосовые агенты (voice agents)

Эти агенты, работающие на основе полного распознавания и генерации речи, делают возможной автоматизацию разговоров в таких областях, как поддержка клиентов, планирование встреч и даже обработка заказов в реальном времени.

Видеоагенты (video agents)

Эти агенты представляют пользователям видеоответы с использованием аватаров, объединяя речь с синхронизацией движений губ, выражения лица и жестов. Сейчас они быстро развиваются в областях продаж, обучения, инструктирования новых клиентов, маркетинга и средств виртуального присутствия, делая возможным масштабируемое, персонализируемое видеообщение без участия человека.

Важно, что количество и разнообразие типов агентов стремительно растет, и, наверняка, вскоре мы будем наблюдать новые виды агентов во многих областях по мере развития самой области и технологий, на которых она базируется. В этой книге основное внимание будет уделяться базовой категории агентов, построенных на основе языковых моделей, особенно использующих текст и код. Хотя в книге будут затронуты темы автоматизации бизнес-задач, голосовых и видеоагентов, в последующих главах на первый план выходят агенты, построенные на базе языковых моделей, — их архитектуры, способности к рассуждениям и пользовательский опыт (UX).

После обсуждения развивающихся категорий агентов возникает следующий важный вопрос: какую модель следует выбрать в качестве основы для вашего агента? Выбор модели — сложная и быстро изменяющаяся область. Как показано в следующем разделе, чтобы сделать правильный выбор, приходится выдерживать баланс таких факторов, как сложность будущих задач, поддержка модальности, ограничения по задержке и затратам, а также требования к интеграции.

Выбор модели

Сейчас появилось множество мощных моделей как от коммерческих поставщиков, так и от сообщества с открытым исходным кодом. OpenAI, Anthropic, Google, Meta и DeepSeek предоставляют самые современные фундаментальные модели с впечатляющими возможностями общего назначения. В то же время модели с открытыми весами — такие как Llama, Mistral и Gemma — расширяют наши представления о том, чего можно достичь локальным развертыванием или тонкой настройкой. Еще более поразительным стало стремительное развитие малых и средних моделей. Новые методы очистки, квантования и генерации синтетических данных позволяют компактным моделям наследовать впечатляющие уровни функциональности от своих более крупных собратьев.

Стремительно расширяющийся выбор можно только приветствовать: конкуренция приводит к ускорению инноваций, улучшению быстродействия и снижению затрат. Впрочем, она также создает дилемму: как выбрать подходящую модель для вашей агентной системы? Истина в том, что одного ответа на все случаи жизни не существует. Один из самых разумных начальных вариантов — просто воспользоваться новейшей моделью общего назначения от какого-либо из ведущих поставщиков (например, OpenAI или Anthropic). Как показано в таблице 1.1, эти модели обеспечивают высокую производительность в изначальном варианте, требуют минимальной настройки и во многих случаях позволяют добиться невероятных результатов. По совокупности характеристик (на август 2025 г.) лидером является GPT-5 mini с общим средним рейтингом 0,819, за ним вплотную следуют o4-mini (0,812) и o3 (0,811). Проприетарные модели и модели с открытым доступом (такие как Qwen3, Grok 4, Claude 4 и Kimi K2) практически не отстают от них.

Впрочем, эти варианты не всегда являются самыми эффективными. Для многих задач — особенно для четко определенных, чувствительных к затратам и обладающих низкой задержкой — менее крупные модели могут показывать практически эквивалентную производительность при существенно меньших расходах. Это привело к формированию нарастающей тенденции: автоматическому выбору модели. Некоторые платформы сейчас перенаправляют простые запросы быстрым и недорогим малым моделям, оставляя большие и дорогие для более сложных рассуждений. Такая динамическая оптимизация оказалась эффективной; вероятно, в будущем мультимодельные системы станут нормой.

Ключевой вывод: вы можете потратить огромные усилия на оптимизацию выбора модели ради малозаметного выигрыша. Однако если масштаб задачи позволяет, можно начинать с самого простого. Часто стоит поэкспериментировать с небольшими моделями, тонкой настройкой или добавлением механизмов поиска для повышения производительности и сокращения затрат. Главное — помнить, что будущее почти наверняка за мультимодельными решениями, и если вы сегодня проектируете с учетом гибкости, то позже это точно окупится.

Таблица 1.1. Рейтинг HELM Core Scenarion (август 2025 г.) Сравнительные показатели производительности 10 ведущих моделей для задач, требующих рассуждений, и оценочных задач: MMLU-Pro, GPQA, IFEval, WildBench иand Omni-MATH

Модель	Средний показатель	MMLU-Pro — COT correct	GPQA — COT correct	IFEval — IFEval Strict Acc	WildBench — WB Score	Omni-MATH — Acc
GPT-5 mini (2025-08-07)	0,819	0,835	0,756	0,927	0,855	0,722
o4-mini (2025-04-16)	0,812	0,82	0,735	0,929	0,854	0,72
o3 (2025-04-16)	0,811	0,859	0,753	0,869	0,861	0,714
GPT-5 (2025-08-07)	0,807	0,863	0,791	0,875	0,857	0,647
Qwen3 235B A22B Instruct 2507 FP8	0,798	0,844	0,726	0,835	0,866	0,718
Grok 4 (0709)	0,785	0,851	0,726	0,949	0,797	0,603
Claude 4 Opus (20250514, extended thinking)	0,78	0,875	0,709	0,849	0,852	0,616
gpt-oss-120b	0,77	0,795	0,684	0,836	0,845	0,688
Kimi K2 Instruct	0,768	0,819	0,652	0,85	0,862	0,654
Claude 4 Sonnet (20250514, extended thinking)	0,766	0,843	0,706	0,84	0,838	0,602

От синхронных операций к асинхронным

Традиционные программные системы обычно выполняют задачи синхронно, шаг за шагом, дожидаясь завершения каждого действия, прежде чем начать следующее. И хотя этот механизм выполнения самый простой, он может оказаться крайне неэффективным — особенно при ожидании внешнего ввода или обработке больших объемов данных.

В противоположность этому, автономные агенты проектируются для асинхронных операций. Они могут управлять несколькими задачами параллельно, быстро адаптироваться к новой информации и динамически назначать приоритеты действий в зависимости от изменяющихся условий. Асинхронная обработка радикально повышает эффективность, сокращая простои и оптимизируя использование вычислительных ресурсов.

Практические последствия этого перехода весьма существенны. Например:

- Сообщения электронной почты могут приходить с уже подготовленными черновиками ответов.
- Счета могут уже включать детали платежа.

- Разработчики могут получать тикеты, к которым прилагаются код для решения и юнит-тесты для проверки.
- Сотрудники отдела поддержки пользователей могут получать предлагаемые ответы и рекомендуемые действия.
- Аналитики в области безопасности могут получать оповещения, которые уже были автоматически проверены и дополнены актуальной информацией об угрозах.

В каждом из этих случаев агенты не просто ускоряют рутинные рабочие потоки — они изменяют природу самой работы. Такое развитие преобразует роль человека: теперь он не сам выполняет задачу, а следит за ее выполнением. Вместо того чтобы тратить время на повторяющиеся или рутинные действия, люди могут сосредоточиться на стратегическом надзоре, ревью и принятии важных решений. Это усилит творческий и оценочный потенциал человека, пока агенты будут заниматься техническими деталями. С такими агентами человек сможет действовать проактивно, а не просто реагировать на происходящее.

Практические применения и сценарии использования

Универсальность автономных агентов дает возможность применять их во множестве различных отраслей. Я хотел, чтобы эта книга базировалась на четких и конкретных сценариях использования, поэтому разместил семь реальных агентов с системами оценки в общедоступном репозитории GitHub (https://oreil.ly/GitHub-scenarios). Мы будем часто возвращаться к этим примерам при изучении ключевых аспектов агентных систем:

Агент поддержки пользователей

Поддержка пользователей — одна из самых частых областей применения автономных агентов. Они обрабатывают типичные запросы, осуществляют возврат средств, обновляют заказы и передают сложные проблемы специалистам-людям, обеспечивая поддержку 24/7, и это сопровождается повышением удовлетворенности клиентов и сокращением операционных издержек.

Агент финансовых сервисов

В банковских и финансовых сервисах агенты помогают с управлением счетами, обработкой кредитных заявок, расследованием попыток мошенничества и изменением баланса инвестиционного портфеля. Они упрощают поддержку клиентов, ускоряют обработку операций и повышают безопасность за счет обнаружения подозрительных операций в реальном времени.

Агент для приема пациентов и выбора приоритетности оказания медицинской помощи

Эти агенты поддерживают работу учреждений здравоохранения: они регистрируют новых пациентов, проверяют страховые полисы, анализируют симптомы

для назначения приоритетов оказания помощи, записывают на прием у врача, ведут истории болезни и координируют направления к специалистам. Таким образом повышается эффективность рабочего потока и результативность лечения.

Агент ИТ-службы поддержки

Агенты ИТ-службы поддержки управляют пользовательским доступом, занимаются диагностикой и устраняют неполадки в работе сети и систем, развертывают обновления программного обеспечения, реагируют на инциденты безопасности и передают нерешенные проблемы специалистам. Быстрое решение стандартных технических проблем способствует повышению производительности.

Агент для анализа юридических документов

Юридические агенты проверяют контракты, проводят правовые исследования, обрабатывают запросы клиентов и проверяют конфликты интересов, оценивают документы на соответствие законодательным нормам, вычисляют размер ущерба и отслеживают сроки. Это помогает оптимизировать рабочие потоки и повышать точность в юридических операциях.

Агент-аналитик центра обеспечения безопасности (Security Operations Center, SOC)

Агенты-аналитики SOC исследуют оповещения о безопасности, собирают данные о киберугрозах, проверяют журналы событий, назначают приоритеты инцидентам, изолируют скомпрометированные хосты и предоставляют обновления командам безопасности. Они ускоряют реакцию на инциденты и укрепляют позицию организации в сфере информационной защиты.

Агент цепочек поставок и логистики

В области управления цепочками поставок агенты оптимизируют запасы, отслеживают доставку, оценивают поставщиков, координируют складские операции, прогнозируют спрос, устраняют возможные нарушения и следят за соблюдением комплаенса. Эти возможности помогают поддерживать необходимую устойчивость и эффективность в глобальных логистических сетях.

Автономные агенты обладают значительным потенциалом в различных сценариях использования, от поддержки клиентов и выполнения функций личного помощника до юридических услуг и рекламы. Интегрируя этих агентов в свои операции, организации могут добиться большей эффективности, повысить качество обслуживания и открыть новые возможности для инноваций и роста. По мере изучения возможностей и практического применения автономных агентов становится очевидно, что их влияние будет весьма значительным и распространится на многие отрасли.

Итак, мы представили несколько примеров реальных агентов. В следующем разделе рассмотрим некоторые ключевые аспекты проектирования агентных систем.

Рабочие потоки и агенты

Во многих реальных проектах выбор между простым скриптом, детерминированным рабочим потоком, традиционным чат-ботом, генерацией, дополненной поиском (RAG, retrieval-augmented generation), или полноценным автономным агентом может определять различия между элегантной реализацией и переусложненным, трудным в сопровождении беспорядочным решением. Чтобы выбор стал более понятным, рассмотрим четыре ключевых фактора: изменчивость входных данных, сложность необходимой логической обработки или рассуждений (reasoning), возможные ограничения по производительности или комплаенсу и бремя текущего сопровождения.

Прежде всего, когда можно обойтись без фундаментальной модели — и вообще любого ML-компонента? Если ваш ввод полностью предсказуем и каждый возможный результат может быть описан заранее, несколько строк процедурного кода часто оказываются быстрее, дешевле и намного проще в тестировании, чем пайплайн на базе ML. Например, разбор файла журнала, в котором всегда соблюдается формат «*ГГГГ-ММ-ДД ЧЧ:ММ:СС — сообщение*», может надежно осуществляться небольшим парсером с регулярными выражениями, написанным на Python или Go. Аналогичным образом, если ваше приложение требует задержки миллисекундного уровня (например, чтобы встроенная система могла реагировать на информацию от датчиков в реальном времени), на вызов API языковой модели попросту не хватит времени. В таких случаях стоит выбирать традиционный код. Наконец, области с сильным регламентированием (медицинское оборудование, аэронавтика, некоторые финансовые системы) часто требуют полностью детерминированной, пригодной для аудита логики принятия решений — нейронные модели по принципу «черного ящика» не пройдут сертификацию. Если действуют какие-либо из этих условий (детерминированный ввод, жесткие требования по производительности или объяснимости, статичная предметная область), простой код почти всегда лучше фундаментальной модели.

Затем рассмотрите детерминированные или полуавтоматизированные рабочие потоки. В них логика может выражаться конечной последовательностью шагов или ветвей, и вы заранее знаете, в каких точках понадобится человеческое вмешательство или дополнительная обработка ошибок. Допустим, вы загружаете счета от небольшого набора поставщиков, и каждый счет поступает в одном из трех известных форматов: CSV, JSON или PDF. Можно построить рабочий поток, который направляет каждый формат подходящему парсеру, проверяет возможные несоответствия и прерывает работу для вмешательства человека, если какие-либо поля не согласуются, — никакое глубокое семантическое понимание не требуется. Аналогичным образом, если ваша система должна повторять неудачные шаги с экспоненциальной задержкой или приостанавливаться для одобрения менеджером, движок рабочего потока (например, Airflow, AWS Step Functions или структурированный набор скриптов) обеспечивает более понятный контроль над ошибочными путями, чем большая языковая модель.

Применение детерминированных рабочих потоков оправданно там, где вы можете заранее перечислить все ветви принятия решений и вам требуется жесткий, пригодный для аудита контроль над каждой ветвью. В таких сценариях рабочие потоки масштабируются более естественно, чем большие ситуативные скрипты, но при этом запуск агентного пайплайна прост и экономичен.

Традиционные чат-боты или RAG-системы занимают следующий уровень сложности; они добавляют понимание естественного языка и загрузку документов, но не доходят до автономного многошагового планирования. Если вам прежде всего нужно предоставить пользователям возможность обращаться с вопросами к базе знаний (допустим, провести поиск в руководстве по продукту, в юридическом архиве или корпоративных вики), RAG-система может встроить документы в векторное хранилище, загрузить соответствующие фрагменты по запросу и сгенерировать связные, учитывающие контекст ответы. Например, внутренняя ИТ-служба поддержки может использовать RAG для получения ответа на вопрос «Как мне сбросить свои учетные данные VPN?», для чего она загружает новейшее руководство по диагностике и обобщает релевантные действия. В отличие от автономных агентов, RAG-системы не принимают независимых решений по дальнейшим действиям (например, оформление тикета или планирование обратного звонка); они просто извлекают информацию. Решение с традиционным чат-ботом или RAG имеет смысл, когда в основном требуются ответы на вопросы по структурированному или неструктурированному контенту с ограниченной необходимостью внешних вызовов API или координации решений. Затраты на сопровождение будут ниже, чем для агентов, и будут связаны с поддержанием актуальности внедренных документов и уточнением промптов, но вы лишаетесь способности агентов планировать многошаговые рабочие потоки или обучаться на циклах обратной связи.

Наконец, остаются автономные агенты — в ситуациях, где не подойдет ни простой код, ни жесткие рабочие потоки, ни RAG, потому что ввод не структурирован, неизвестен заранее или отличается высокой изменчивостью; потому что вам требуется динамическое многошаговое планирование (multistep planning) или непрерывное обучение на обратной связи (continuous learning from feedback). Представьте центр поддержки пользователей, который получает по электронной почте сообщения в свободной форме, от «аккумулятор моего ноутбука вздулся и скоро лопнет» до «с меня продолжают списывать оплату за услуги, которые я не заказывал». Рабочий поток, основанный на правилах, или поиск в FAQ с использованием RAG не справится с таким разнообразием, но агент, работа которого основана на фундаментальной модели, сможет определить намерения, извлечь актуальные сущности, обратиться к базе знаний, построить черновик подходящего ответа и даже передать вопрос человеку, если потребуется, — и все это без определения всех возможных ветвей заранее. Аналогичным образом при управлении цепочкой поставок агент, который получает в реальном времени данные складских запасов, сроки поставки и прогнозы продаж, может динамически перепланировать графики поставок; детерминированному рабочему потоку потребуются постоянные ручные обновления для обработки новых исключительных случаев.

Агенты также отлично проявляют себя в ситуациях, когда много подзадач должны выполняться параллельно, например, агент безопасности одновременно опрашивает API обнаружения угроз, сканирует сетевую телеметрию и выполняет в песочнице анализ подозрительных двоичных файлов. Так как агенты работают асинхронно и переназначают приоритеты на основании данных реального времени, они лишены хрупкости, присущей пошаговому подходу рабочих потоков или RAG-систем. Для оправдания более высоких затрат на вычисления и сопровождение, связанных с использованием фундаментальной модели, необходим именно такой уровень контекстного рассуждения и умозаключений, координации параллельных задач или непрерывного самосовершенствования. Это сценарии, когда жесткий код, рабочие потоки или чат-боты будут слишком хрупкими или затратными для сопровождения.

Таблица 1.2. Отличия рабочих потоков и агентов от традиционного кода

Характеристика	Традиционный код	Рабочий поток	Автономный агент
Структура ввода	Полностью предсказуемые схемы	В основном предсказуем с конечным числом ветвей	Неструктурированный или заранее неизвестный ввод
Объяснимость	Полная прозрачность; простой аудит	Явный след аудита «ветвь за ветвью»	Компоненты по принципу «черного ящика», требующие дополнительной инструментации
Задержка	Сверхнизкая задержка	Умеренная задержка	Высокая задержка
Адаптируемость и обучение	Нет	Ограничены	Высокие (с обучением на обратной связи)

Каждый путь подразумевает некоторые компромиссы. Традиционный код дешев и быстро работает, но ему не хватает гибкости; рабочие потоки обеспечивают контроль, но не справляются с сильно изменчивым вводом; традиционные чат-боты или RAG отлично подходят для получения ответов на вопросы по документам, но не могут заниматься оркестрацией многошаговых действий; агенты обладают выдающейся мощью, но предъявляют высокие требования к облачным вычислениям и инженерным усилиям по мониторингу, настройке и управлению. Прежде чем сделать окончательный выбор, спросите себя: является ли в моем случае ввод неструктурированным или непредсказуемым? Необходимо ли мне многошаговое планирование, адаптирующееся к промежуточным результатам? Достаточно ли системы получения документов для информационных потребностей моих пользователей или же система должна принимать решения и действовать автономно? Хочу ли я, чтобы эта система совершенствовалась со временем при минимальном вмешательстве со стороны человека? И приемлема ли задержка и затраты на сопровождение фундаментальной модели?

Короче говоря, если ваша задача сводится к фиксированному детерминированному преобразованию, напишите простой код. Если в обработке имеется ряд известных ветвлений, и вам нужны явные контрольные точки обработки ошибок, используйте детерминированный рабочий поток. Если вам прежде всего нужны

ответы на вопросы по корпусу текстов, задаваемые на естественном языке, выберите традиционного чат-бота или архитектуру RAG. Но если вы сталкиваетесь с высокой изменчивостью, необходимостью открытых логических рассуждений и динамического планирования или требованиями к непрерывному обучению, будьте готовы инвестировать в разработку автономного агента. Сознательное принятие этого решения гарантирует, что вы получите правильное соотношение простоты, производительности и адаптируемости, чтобы ваше решение оставалось как эффективным, так и простым в сопровождении при эволюции требований.

Принципы построения эффективных агентных систем

Создание успешных автономных агентов требует подхода, который выводит на передний план масштабируемость, модульность, непрерывное обучение, гибкость и защиту от устаревания (future-proofing).

Масштабируемость

Убедитесь в том, что агенты справятся с растущими рабочими нагрузками и разнообразными задачами за счет использования распределенных архитектур, облачной инфраструктуры и эффективных алгоритмов, поддерживающих параллельную обработку и оптимизацию ресурсов. Пример: агент поддержки пользователей, обрабатывающий 10 заявок в минуту, может зависнуть или аварийно завершиться при всплеске трафика до 1000 заявок, если такие всплески не поддерживаются инфраструктурой автоматического масштабирования.

Модульность

Проектируйте агентов с независимыми взаимозаменяемыми компонентами, соединяемыми через понятные интерфейсы. Такой модульный подход упрощает сопровождение, способствует гибкости и упрощает быструю адаптацию к новым требованиям или технологиям. Пример: агент с плохой модульностью, в котором все инструменты жестко запрограммированы в его агентном сервисе, потребует полного повторного развертывания каждый раз, когда в инструменты вносятся небольшие добавления или изменения.

Непрерывное обучение

Оснастите своих агентов механизмами обучения через опыт (такими как контекстное обучение — in-context learning). Интегрируйте обратную связь пользователей для уточнения поведения агентов и следите за производительностью по мере эволюции задач. Пример: агенты, игнорирующие циклы обратной связи, могут снова и снова совершать одни и те же ошибки — такие как неправильная классификация условий контракта или неспособность к эскалации критических проблем с поддержкой.

Эластичность

Разрабатывайте надежные эластичные архитектуры, способные корректно справляться с ошибками, угрозами безопасности, тайм-аутами и неожиданными

условиями. Внедряйте тщательную обработку ошибок, строгие меры безопасности и избыточность для обеспечения надежной и непрерывной работы агентов. Пример: агенты без логики повторных попыток или резервного механизма могут полностью утратить работоспособность при сбое единственного вызова API, а пользователь будет ждать ответа, не понимая, что происходит.

Защита от устаревания

Стройте агентные системы на базе открытых стандартов и масштабируемой инфраструктуры, развивайте инновационную культуру для быстрой адаптации к появляющимся технологиям и развивающимся ожиданиям пользователей. Пример: жесткая привязка агента к одному проприетарному формату промпта от конкретного поставщика может существенно усложнить переход на другую модель и ограничить возможности для экспериментирования.

Соблюдение этих принципов позволяет организациям разрабатывать автономных агентов, которые остаются эффективными и актуальными, легко адаптируются к технологическим достижениям и изменяющимся рабочим средам.

Организационные основы успешного создания агентных систем

Доступность фундаментальных моделей через простые вызовы API стала причиной обширных экспериментов с агентными системами во многих организациях. Команды часто запускают независимые проверки жизнеспособности концепций, приводящие к ценным открытиям и инновационным идеям. С другой стороны, простота экспериментирования часто приводит к фрагментации — перекрытию проектов, дублированию усилий и росту числа незавершенных экспериментов в организации. Наоборот, преждевременная стандартизация может подавлять творчество и загонять компании в ловушки жестких фреймворков или решений, привязанных к конкретному поставщику. Чтобы ваша система была успешной, необходимо выдержать баланс гибкости для экспериментов с достаточным потенциалом для масштабируемости и связности.

На ранних фазах разработки агентов организации должны активно поощрять исследовательские усилия, давая возможность командам свободно тестировать разные архитектуры, рабочие потоки и модели. Со временем будут выявлены успешные паттерны и лучшие практики, и стратегическая ориентация начнет играть критическую роль. Реализация стратегии «один стандарт на большую группу» может эффективно сбалансировать эту необходимость. В конкретных подразделениях или функциональных областях команды могут создавать стандарты, основанные на общепринятых инструментах и методологиях и упрощающие сотрудничество без ограничения инноваций в более широком масштабе.

Другой важный аспект успеха — отсутствие привязок к конкретному поставщику за счет принятия открытых стандартов (таких как OpenAPI) и стремления

к модульности системных дизайнов. Такие практики способствуют обеспечению гибкости и сокращению зависимости от одной технологии или поставщика, а это обеспечивает адаптируемость в будущем.

Эффективный обмен знаниями также играет важную роль. Уроки, извлеченные как из успешных, так и из проваленных экспериментов, должны широко распространяться на внутренних форумах, в общих репозиториях и в подробной документации. Стратегия, ориентированная на сотрудничество, ускорит организационное обучение, сведет к минимуму избыточные усилия и стимулирует коллективные улучшения.

Наконец, управляющие фреймворки должны оставаться гибкими и легкими, они должны отдавать предпочтение направляющим принципам перед жесткими предписаниями. Оптимизированная структура управления позволяет командам уверенно заниматься инновациями без ущерба для общих целей организации.

Успешные процессы на базе агентных систем имеют итеративную природу. Организации должны постоянно переоценивать свои стратегии для поддержки динамического баланса между исследованиями и стандартизацией. Культивируя среду, в которой ценятся эксперименты, совместное обучение и открытые стандарты, организации могут эффективно преобразовывать агентные системы из изолированных экспериментов в масштабируемые, готовые к преобразованиям решения, которые глубоко интегрируются в их рабочие процессы.

Агентные фреймворки

В настоящее время существуют многочисленные фреймворки для разработки автономных агентов, каждый из которых направлен на некоторую критическую функциональность: интеграция навыков, управление памятью, планирование, оркестрация, обучение через опыт и мультиагентная координация. Ниже перечислены наиболее известные фреймворки (безусловно, этот список не полон).

LangGraph

Достоинства

Модульный фреймворк оркестрации, основанный на направленных графах, вершины которых содержат отдельные единицы логики (часто — обращения к фундаментальным моделям), а ребра — потоки данных в сложных, потенциально циклических рабочих потоках; сильная эргономика разработчика; встроенная поддержка асинхронных рабочих потоков и повторных попыток.

Компромиссы

Требует собственной логики для нетривиального планирования и управления памятью; низкий уровень встроенной поддержки для мультиагентного взаимодействия.

Кому подходит

Командам, занимающимся построением надежных одноагентных или несложных мультиагентных систем с явным и контролируемым управлением потоками.

AutoGen

Достоинства

Мощные возможности мультиагентной оркестрации; динамическое назначение ролей; гибкие взаимодействия между агентами на базе сообщений.

Компромиссы

Даже для простых сценариев использования решения могут быть тяжеловесными или сложными; ограниченный выбор схем взаимодействий между агентами.

Кому подходит

Исследовательским и производственным системам, требующим диалога между агентами (например, руководитель/работник, циклы самоанализа).

CrewAI

Достоинства

Простота в изучении и освоении; быстрая подготовка для построения прототипов; полезные абстракции типа «команды» или «задачи».

Компромиссы

Ограниченные возможности настройки и контроля над внутренними механизмами оркестрации; уступает по зрелости LangGraph или AutoGen для сложных рабочих потоков.

Кому подходит

Разработчикам, которые хотят побыстрее взяться за построение практичных агентов, ориентированных на человека (таких как помощники или агенты поддержки).

OpenAI Agents Software Development Kit (SDK)

Достоинства

Глубокая интеграция с экосистемой инструментов OpenAI; безопасный и простой вызов функций; примитивы управления памятью и передача управления инструментам.

Компромиссы

Тесная привязка к инфраструктуре OpenAI; может быть менее гибким или менее портируемым для нестандартных агентных стеков или инструментариев с открытым кодом.

Кому подходит

Командам, которые уже используют OpenAI API и ищут возможности быстрого создания безопасных агентов, способных пользоваться инструментами с минимумом вспомогательного кода (scaffolding).

Хотя каждый фреймворк обладает уникальным набором достоинств и ограничений, ожидается, что непрерывные инновации и конкуренция в этой области будут стимулировать дальнейшую эволюцию. Если вы создаете ранние прототипы, то CrewAI и OpenAI Agents SDK позволят вам быстро взяться за дело. Для масштабируемых систем, рассчитанных на реальную эксплуатацию, LangGraph и AutoGen предоставляют больше возможностей для контроля и более сложной функциональности. Эти фреймворки не являются необходимыми, и многие команды предпочитают строить проекты прямо на базе API от поставщиков модели. В этой книге основное внимание уделяется фреймворку LangGraph, который был выбран за простой, но мощный подход к разработке агентных систем. При помощи подробных объяснений, практических примеров и реальных сценариев мы продемонстрируем, как LangGraph эффективно решает проблемы сложности и динамики, необходимые для современных интеллектуальных агентов.

Заключение

Автономные агенты способны выполнять сложные динамические задачи с высокой степенью автономности. В этой главе были изложены фундаментальные концепции агентов, выделены их преимущества перед традиционными ML-системами, а также рассмотрены практические применения и ограничения. По мере более глубокого изучения проектирования и реализации таких систем становится ясно, что осмысленная интеграция агентов в различные области приведет к прорывным инновациям и повышению эффективности.

Хотя различные подходы к проектированию автономных агентов, описанные в этой главе, продемонстрировали значительные возможности и потенциал, они также дают представление о сложности и проблемах, связанных с построением эффективных и адаптируемых систем. Каждый метод, от систем, основанных на правилах, до современных когнитивных архитектур, обладает уникальными достоинствами, но ему также присущи определенные внутренние ограничения. В этой книге я постараюсь восполнить этот пробел в понимании.