ИЗДАНИЕ
2026

AI-АГЕНТ ЗА 3 НЕДЕЛИ

67 шаблонов, команд и чек-листов

Павел Новопольцев

PAVELNSYSTEMS

Павел Новопольцев

AI-агенты за 3 недели 67 шаблонов, команд и чек-листов для надёжного исполнителя.

«Автор»
2026

Новопольцев П.

AI-агенты за 3 недели 67 шаблонов, команд и чек-листов для надёжного исполнителя. / П. Новопольцев — «Автор», 2026

Модель рапортует «готово», а в продакшене тишина и битые файлы. Знакомо? Эта книга рабочий маршрут от разочарования в чат-боте до агента, который доводит задачи до конца.Что внутри:— 13 глав пошаговой сборки на Hermes Agent Framework;— 3 ключевых файла запуска: AGENTS.md, feature_list.json, progress.md;— 5 частей harness: память, инструменты, каналы, безопасность, тестирование;— 67 готовых шаблонов, команд и чек-листов; — трёхнедельный план от идеи до запуска.Для разработчиков, уставших от «преждевременного успеха» LLM, и для экспертов, желающих собрать своего агента без найма команды. Автор Павел Новопольцев (PavelNSystems), практик и автор YouTube-канала про AI.После прочтения у вас будет не демо, а рабочий исполнитель на своём стеке, под свои задачи. Соберите его за 3 недели.

Содержание

Павел Новопольцев
AI-агенты за 3 недели 67 шаблонов, команд и чек-листов для надёжного исполнителя.

Что внутри

Эта книга — не пересказ документации и не обзор фреймворков. Это маршрут: от первого разочарования в «умном чат-боте», который врёт о завершении задачи, до рабочего агента, который реально доводит дело до конца.

Я написал её для тех, кто уже пробовал LLM-чат и столкнулся с тремя привычными бедами: модель говорит «готово», а по факту ничего не готово; агент забывает, что делал пять минут назад; а чуть отвлечёшься — и он уже выполнил rm -rf в неправильной директории. Все эти беды лечатся не «возьму модель помощнее», а системой — тем, что в книге называется harness, обвязкой.

По мере чтения вы пройдёте путь: от понимания, чем агент отличается от чат-бота, до выбора фреймворка, настройки безопасности и трёхнедельного плана, который выводит к первому рабочему агенту. В конце — практический блок с чек-листами, шаблонами файлов и командами, которые пригодятся в первый же день работы.

Кому это читать

Начинающим разработчикам, которые уже пробовали LLM, но хотят собрать первого автономного агента — а не очередного чат-бота с кнопками.

Инженерам, уставшим от «преждевременного успеха»: модель говорит, что задача выполнена, а проверка показывает обратное. Эта книга объясняет, почему так происходит и как заставить агента реально проверять результат.

Техлидам и тимлидам, которые присматриваются к агентам для рутинных задач команды — ревью pull-реквестов, настройки CI/CD, работы с логами — и хотят понять, где тут реальная польза, а где хайп.

Самозанятым и небольшим командам, которым нужен всегда доступный ассистент в Telegram или Discord, который не «теоретически может», а реально делает.

Не нужно быть ML-инженером. Базового умения читать код и работать в терминале достаточно: всё остальное вы получите по дороге.

Маршрут читателя

Книга построена по принципу «халва по граммам»: одна идея — одна порция. Главы идут по зависимостям, а не по темам, поэтому я рекомендую читать их по порядку. Каждая глава опирается на то, что объяснено в предыдущих, и заранее вводит крючок к следующей.

Внутри глав вы встретите несколько повторяющихся блоков. «Важно» — это то, что часто упускают и за что потом платят часами отладки. «Пример» — живая ситуация, которая делает абстрактное конкретным. «Что сделать на практике» — конкретный шаг, который можно сделать сразу после прочтения. «Итог главы» — три-четыре мысли, которые стоит запомнить, плюс крючок к следующей главе.

В конце книги — практический блок: чек-листы, шаблоны AGENTS.md, feature_list.json, progress.md, системного промпта и сводка команд Hermes. Если вы предпочитаете «сначала сделать, потом понять» — можно начать с практического блока, собрать агента по шаблонам, а потом вернуться к главам за пониманием, почему шаблоны именно такие.

Моё обещание вам

Если вы хотя бы раз просили ChatGPT или Claude «настроить CI/CD для проекта» — вы знаете, как это обычно заканчивается. Модель уверенно рассказывает, что нужно сделать. Показывает куски конфигурации. Объясняет, какой шаг за каким идёт. Вы киваете, закрываете вкладку и идёте делать всё руками — потому что модель не может ни создать файл, ни запустить тесты, ни проверить, что получилось.

Это не баг модели.

Это её нормальное состояние. LLM по своей природе — это собеседник, а не исполнитель. Она генерирует текст в ответ на текст. И сколько бы вы её ни просили «действовать», она не откроет файл, не запустит команду и не отправит pull-реквест, потому что у неё нет для этого рук. Чтобы LLM стала агентом, ей нужна обвязка: инструменты, память, правила, верификация. Без обвязки вы получаете умного собеседника. С обвязкой — исполнителя.

Эта книга о том, как собрать такую обвязку. Не о моделях — их сделают за вас большие лаборатории, и каждую неделю выходит что-то новое. А о системе вокруг модели, которая превращает «говорящую голову» в рабочего ассистента: того, кто понимает задачу, выбирает инструмент, помнит контекст, проверяет результат и не сжёг ваш сервер к концу первого дня.

Я обещаю: к концу книги у вас в голове сложится модель — не в смысле «нейросеть», а в смысле «картина мира». Вы будете понимать, какие компоненты у агента есть, зачем нужен каждый, почему сильная модель без правил ведёт себя непредсказуемо, как организовать память так, чтобы агент не «забывал» предыдущие шаги, и как настроить безопасность, чтобы не бояться давать агенту доступ к терминалу. Этого хватит, чтобы собрать первого агента самостоятельно — на любом фреймворке, с любой моделью, под любую задачу.

И ещё одно. По дороге вы поймёте главное: надёжность агента — это свойство системы, а не модели. Сильная LLM без harness — это ракета без системы наведения. Слабая LLM с хорошим harness спокойно доведёт задачу до конца. Эта мысль проходит через всю книгу, и к тринадцатой главе она станет очевидной.

Глава 1. Почему ваш «умный чат-бот» бросает задачу на полпути

Представьте знакомую ситуацию. Вы ставите задачу: «настрой CI/CD для этого проекта». Чат-бот — неважно, какой именно — отвечает развёрнуто. Рассказывает про GitHub Actions, показывает yaml-конфигурацию, объясняет, какие шаги нужны: линтер, тесты, сборка Docker-образа, деплой на staging. Вы читаете, киваете, закрываете вкладку — и идёте делать всё это сами. Потому что бот ничего из перечисленного не сделал. Он только рассказал.

Это не дефект конкретной модели. Это её природа. Большая языковая модель по определению — это система, которая продолжает текст. Она предсказывает следующий токен на основе предыдущих. Когда вы спрашиваете «как настроить CI/CD», она отвечает текстом — потому что для неё это просто продолжение диалога. У неё нет ни терминала, чтобы запустить команду, ни файловой системы, чтобы записать конфиг, ни рук, чтобы нажать кнопку. Она не «не хочет» — она не может.

Поэтому, когда вы жалуетесь, что «агент снова бросил задачу», стоит уточнить: вы общались с агентом или с чат-ботом? Эти слова часто путают, а разница между ними — вся суть этой книги.

ОПРЕДЕЛЕНИЕ

Чат-бот отвечает на вопросы. AI-агент выполняет задачи. Чат-бот генерирует текст в ответ на текст. Агент читает файлы, запускает команды, ищет информацию, принимает решения в рамках заданных правил и докладывает о результате. У чат-бота есть только голос. У агента есть голос, руки, память и правила поведения.

Разница не семантическая, а архитектурная. Чтобы LLM стала агентом, вокруг неё собирают обвязку — набор компонентов, которые превращают «говорящую голову» в исполнителя. Эта обвязка и есть то, что я в этой книге называю harness. Без harness — чат-бот. С harness — агент.

Теперь вернёмся к нашему CI/CD. Если ту же задачу — «настрой CI/CD для этого проекта» — дать настоящему агенту, произойдёт другое. Агент прочитает структуру проекта. Поймёт, что это Python-репозиторий с pytest. Создаст файл .github/workflows/ci.yml. Прогонит тесты локально, чтобы убедиться, что они проходят. Запушит ветку. Проверит, что pipeline в GitHub Actions зелёный. И только тогда сообщит: «готово, пайплайн работает, вот ссылка». Разница не в качестве ответа — разница в том, что ответ вообще есть.

ПРИМЕР

Представьте, что вы попросили ассистента «настрой CI/CD». Он уверенно отвечает: «Сделано! Пайплайн настроен». Вы открываете репозиторий — файла нет. Спрашиваете снова — он показывает кусок yaml и говорит: «Вот, добавьте это в репозиторий». Вы добавляете. Запускаете. Падает. Спрашиваете — он извиняется и предлагает другой кусок. И так три часа.

Это и есть чат-бот, замаскированный под агента. У него нет инструмента для записи файла. У него нет инструмента для запуска тестов. У него нет верификации — он не может проверить, что его совет сработал. Поэтому он рапортует об успехе, которого не было. Это не «глупая модель» — это модель без рук.

ВАЖНО

Главный симптом, что вы общаетесь с чат-ботом, а не с агентом: модель говорит «готово», но вы не можете проверить это, не открыв ещё одну вкладку и не сделав что-то руками. Если каждое «сделано» требует от вас ручной проверки и доделки — у вашей модели нет рук. Ей нужен harness.

Откуда тогда берётся уверенность модели, что «всё готово»? Это интересный момент. LLM обучена продолжать текст так, как будто она компетентный собеседник. Когда её спрашивают «настроил?», она, не имея реального опыта, отвечает в стиле «да, конечно» — потому что в её обучающей выборке именно так отвечают ассистенты в подобных диалогах. Это не ложь в человеческом смысле. Это текстовая реконструкция поведения, которое модель никогда не выполняла. Поэтому верить модели на слово о завершении задачи — самый дорогой источник проблем в работе с агентами.

Хорошая новость: всё это лечится. Чтобы превратить чат-бот в агента, нужно добавить пять компонентов. Они появляются в следующей главе — и сразу станет видно, какого именно не хватает в вашей текущей неудачной попытке.

ЧТО СДЕЛАТЬ НА ПРАКТИКЕ

Вспомните последнюю задачу, которую вы пытались поручить LLM и которая провалилась. Запишите её. Рядом запишите, что именно пошло не так: модель соврала о завершении? Не смогла записать файл? Забыла контекст? Выполнила опасное действие? Эта запись пригодится в следующих главах — вы будете сверять её с компонентами и увидите, какого именно не хватило.

ИТОГ ГЛАВЫ

- Чат-бот отвечает на вопросы, агент — выполняет задачи. Разница в наличии рук, памяти и правил, а не в качестве модели.
- LLM по природе — это текстовый движок. У неё нет инструментов, чтобы действовать в мире. Без обвязки она остаётся собеседником.
- «Готово» из уст модели без верификации — это не отчёт, а текстовая реконструкция успеха. Верьте только проверяемому результату.
- Превращение чат-бота в агента — это инженерная задача, а не молитва «дай нам модель помощнее».

→ *Чтобы агент мог «сделать», а не «рассказать», ему нужны пять компонентов. Каких именно — в следующей главе. Спойлер: один из них — модель, но далеко не первый по важности.*

Глава 2. Пять частей, без которых агента не существует

В предыдущей главе мы выяснили: чтобы LLM стала агентом, ей нужна обвязка. Теперь посмотрим, из чего эта обвязка состоит. У любого агента — от простейшего CLI-бота до корпоративного ассистента в Telegram — есть пять компонентов. Если убрать любой из них, агент перестаёт быть агентом и превращается во что-то другое: в чат-бот, в скрипт, в галлюцинирующую модель или в закрытую систему без связи с внешним миром.

Я перечислю все пять, а потом разберём каждый по «грамму» — что это, зачем нужно и что случится, если его не станет.

ПЯТЬ КОМПОНЕНТОВ АГЕНТА

1. Языковая модель (LLM) — мозг, обрабатывающий запросы и генерирующий ответы. 2. Память — способность сохранять контекст между обращениями и сессиями. 3. Инструменты — действия во внешнем мире: чтение и запись файлов, терминал, веб-поиск. 4. Harness (обвязка) — явные правила и ограничения, которые задают границы поведения. 5. Каналы связи — интерфейсы, через которые вы общаетесь с агентом: CLI, мессенджер, веб-консоль.

1. Языковая модель — мозг

Это то, что у всех на слуху: GPT-4o, Claude, Qwen, DeepSeek, локальная Llama через Ollama. Модель принимает текст на вход и генерирует текст на выход. Без неё агент не существует — нечего «обвязывать». Но она не делает работу сама; она принимает решения внутри системы, которую вы построили.

Минимальное требование к модели для агента — контекстное окно не менее 64 000 токенов. Меньше — и агент не сможет вести многошаговый диалог с инструментами: каждый вызов инструмента добавляет в контекст его результат, и при коротком окне агент быстро забывает, с чего начал. Современные облачные модели дают 128K–200K, чего хватает для большинства задач.

Выбор модели зависит от трёх вещей: бюджета, требований к приватности и доступности в вашем регионе. Для первого проекта проще всего начать с облачной модели — OpenAI, Anthropic, DeepSeek, Alibaba Cloud Qwen — а потом, если понадобится полная приватность, перейти на локальную через Ollama или vLLM.

2. Память — почему агент не начинает с нуля

Без памяти каждый диалог с агентом начинается с чистого листа. Вы объяснили задачу — агент начал работать. Закрыли вкладку, открыли снова — и агент снова спрашивает, что вы от него хотите. Для коротких диалогов это приемлемо. Для задач, которые занимают часы или дни, — убийственно: каждое возвращение к работе превращается в пересказ предыдущих шагов.

Память бывает нескольких уровней: краткосрочная (в рамках одной сессии — это контекстное окно модели), среднесрочная (прогресс задач, сохраняемый в файл progress.md), долгосрочная (база знаний и история прошлых взаимодействий). В шестой главе мы разберём её подробно — там есть свои тонкости. Пока достаточно запомнить: память — это отдельный компонент, который нужно проектировать, а не «он сам появится».

3. Инструменты — руки агента

Это то, что превращает агента из собеседника в исполнителя. Без инструментов модель может только генерировать текст. С инструментами — читать и писать файлы, выполнять команды в терминале, искать информацию в интернете, работать с базами данных, отправлять запросы к API.

Инструменты бывают двух родов. Встроенные — терминал, файлы, браузер — обычно входят в любой фреймворк из коробки. Внешние подключаются через протокол MCP (Model Context Protocol) — это стандарт, который позволяет стыковать агента со сторонними серверами инструментов: GitHub, Slack, базами данных, Kubernetes. Отдельно стоят навыки (Skills) — многошаговые сценарии, которые объединяют несколько инструментов под одну цель: «создать Pull Request» — это чтение изменений, генерация описания, создание ветки и отправка PR через Git. Мы разберём это в седьмой главе.

4.

Harness

— правила игры

Это центральная тема книги, и ей посвящены отдельные главы — с третьей по пятую. Пока — определение.

ОПРЕДЕЛЕНИЕ

Harness (обвязка) — набор явных правил и ограничений, которые определяют, что агент может и чего не может делать, как он проверяет свою работу и как сохраняет контекст. Harness не делает модель умнее. Он создаёт для неё замкнутую рабочую систему с чёткими границами, верификацией результатов и наблюдаемостью.

Без harness сильная модель — это мощный автомобиль без руля и тормозов: она может, но не знает куда. Без harness агент часто объявляет задачу выполненной до того, как реально её завершил, или «забывает» о предыдущих шагах, или выполняет опасные команды, потому что никто не сказал «не делай так». Harness — это то, что отличает «работает» от «иногда работает».

5. Каналы связи — где вы общаетесь с агентом

Самый простой канал — CLI, командная строка. Для постоянной работы удобнее мессенджер: Telegram, Discord, Slack, WhatsApp. Для командной работы — веб-консоль. Фреймворки вроде Hermes и QwenPaw поддерживают несколько каналов одновременно, а компонент gateway маршрутизирует сообщения между ними и ядром агента.

Канал — это не «только интерфейс». От выбора канала зависит, как агент вписан в вашу жизнь. CLI требует, чтобы вы открыли терминал. Мессенджер — что агент всегда на связи, как коллега. Веб-консоль — что у агента есть «кабинет», куда вы заходите для серьёзных задач. Это разные модели использования, и их стоит подбирать под задачу.

ВАЖНО

Все пять компонентов незаменимы. Уберите модель — нечего обвязывать. Уберите память — агент забывает. Уберите инструменты — модель не может действовать. Уберите harness — модель врёт о завершении и не знает границ. Уберите каналы связи — нет способа с ней общаться. Компоненты не заменяют друг друга: их можно только усилить или ослабить.

Если вы сейчас работаете с LLM и вам кажется, что «агент не справляется», попробуйте применить этот чек-лист. Модель у вас есть? Скорее всего. Память? Если контекст сбрасывается между сессиями — её нет. Инструменты? Если модель только отвечает текстом — их нет. Harness? Если модель врёт о завершении — его нет. Каналы? Если вы общаетесь через обычный веб-чат OpenAI — это не канал агента, это интерфейс собеседника.

Чаще всего у начинающих есть модель и есть канал (чат). Не хватает инструментов, памяти и harness. Эти три компонента — и есть то, что вы собираете, когда «делаете агента». Мозг уже есть. Нужно дать ему руки, память и правила.

ЧТО СДЕЛАТЬ НА ПРАКТИКЕ

Нарисуйте пять колонок: модель, память, инструменты, harness, каналы. Напротив каждой — что у вас сейчас есть в проекте, с которым вы работаете. Большинство обнаружит, что три из пяти пустые. Это нормально. Это и есть зона роста: следующие главы заполнят их по очереди.

ИТОГ ГЛАВЫ

- Любой агент состоит из пяти компонентов: LLM, память, инструменты, harness, каналы связи.
- Компоненты незаменимы: убрать любой — агент перестаёт быть агентом.
- Минимальное контекстное окно модели — 64K токенов, иначе многошаговые диалоги не работают.
- Чаще всего у начинающих есть только модель и канал. Не хватает инструментов, памяти и harness.

→ *Теперь, когда вы знаете пять компонентов, самое время разрушить главный миф начинающих: «возьму модель помощнее — и агент заработает». Почему это не работает — в следующей главе.*

Глава 3. Главная ошибка: верить, что мощная модель заменит правила

Когда агент не справляется, первая реакция — сменить модель. GPT-4o вместо GPT-4o-mini. Claude Opus вместо Sonnet. DeepSeek-R1 вместо V3. Логика простая: если модель «глупее», чем нужно, возьмём «умнее» — и всё заработает. Иногда это действительно помогает — но чаще нет. И вот почему.

Проблемы агента делятся на два класса. Первый — модель действительно недостаточно умна для задачи: не понимает нюансы, путается в логике, не справляется с длинным контекстом. Здесь смена модели на более сильную помогает напрямую. Второй класс — модель достаточно умна, но у неё нет правильной обвязки: она не верифицирует результат, не помнит предыдущие шаги, не знает границ. Здесь смена модели не помогает вообще. Точнее, она помогает временно и случайно — потому что новая модель, скорее всего, тоже не будет верифицировать, не будет помнить и не будет знать границ.

Сильная модель не означает надёжное исполнение. Даже лучшие модели ошибаются, галлюцинируют и преждевременно объявляют об успехе.

—Harness Engineering, базовый тезис

Это не теоретическое утверждение. Это рабочая реальность каждого, кто собирал агентов. Самые частые проблемы — не «модель не смогла понять задачу», а «модель сказала, что выполнила, а по факту нет» или «модель потеряла контекст на пятнадцатом шаге» или «модель выполнила rm -rf в корневой директории, потому что ей никто не запретил». Смена модели с GPT-4o на Claude Opus в этих случаях не лечит ничего: новая модель точно так же будет рапортовать об успехе без верификации, терять контекст и выполнять опасные команды — просто, возможно, чуть реже или чуть аккуратнее.

ПРИМЕР

Представьте двух плотников. Один — новичок, но ему дали чертёж, рулетку, карандаш и правило «семь раз отмерь, один раз отрежь». Второй — мастер с тридцатилетним стажем, но без чертежа и инструментов — только руки и пила. Кто точнее отрежет доску? Скорее всего, первый — потому что у него есть система. Мастер может «сделать на глаз», но ошибка будет. И не потому, что он плохо режет, а потому что без системы даже мастер не гарантирует результат.

Так же и с моделью. Сильная LLM без harness — это мастер без рулетки. Слабая LLM с harness — новичок с чертежом. Второй надёжнее.

В этом месте часто возражают: «но ведь GPT-4o умнее, значит, меньше галлюцинирует и лучше понимает задачу». Это отчасти правда — для чат-бота. У чат-бота нет инструментов, и вся его работа — это текст. Там «умнее» значит «лучше отвечает». У агента работа состоит не из ответов, а из последовательности действий, и каждое действие должно быть проверено. «Умнее» в этом контексте значит «лучше понимает, что делать» — но не «гарантированно делает».

Здесь мы подходим к ключевой мысли всей книги. Запомните её.

ВАЖНО

Надёжность агента — это свойство системы, а не модели. Сильная LLM без harness ведёт себя непредсказуемо. Слабая LLM с хорошим harness спокойно доводит задачу до конца.

Harness не заменяет модель и не делает её умнее — он создаёт условия, при которых любая достаточно умная модель работает надёжно.

Из этого тезиса следуют два практических вывода. Первый: не пытайтесь компенсировать слабый harness мощной моделью. Если агент врёт о завершении, проблема не в модели — добавьте правило «всегда запускать тесты перед завершением» и проверку «файл существует и непустой». Если агент забывает контекст — добавьте progress.md и требование записывать шаги. Если агент выполняет опасные команды — добавьте tool guardrails. Каждая из этих проблем решается harness-ом, а не сменой модели.

Второй вывод: не пытайтесь заменить инструменты промптами. Это ещё одна частая попытка. Если агенту нужно прочитать файл, а у него нет инструмента чтения, никакой промпт «пожалуйста, прочитай файл» не поможет — модели физически нечем его прочитать. Если нужно проверить, прошёл ли тест, нужен инструмент запуска тестов. Промпт не заменяет инструмент, потому что промпт — это текст, а инструмент — это действие. Текстом можно описать, что нужно сделать, но нельзя сделать.

ПРИМЕР

Представьте, что вы просите модель «прочитай файл main.py и скажи, что в нём». Модель отвечает: «Не могу прочитать файл напрямую, но если вы вставите содержимое сюда, я помогу разобрать». Это не упрямство модели — это честное признание, что у неё нет инструмента. Если вы вместо этого напишете длинный промпт «пожалуйста, ну пожалуйста, представь, что ты прочитал main.py», модель начнёт галлюцинировать содержимое файла. Это и есть источник знаменитых «галлюцинаций» — модель достраивает реальность текстом, потому что у неё нет инструментов её проверить.

Итак, выводы главы. Когда агент не справляется, сначала ищите проблему в обвязке, а не в модели. Поменяйте модель только если убедились, что обвязка правильная, но модели не хватает ума для конкретной задачи. Не пытайтесь заменить инструменты промптами. И помните: harness — это не «костыль для слабых моделей», это нормальная инженерная практика, без которой не работает ни один надёжный агент.

В следующей главе мы наконец посмотрим, как именно устроен harness — что в нём есть, какие правила он задаёт и как выглядит цикл его работы. Это центральная концепция книги, поэтому я разберу её по «грамму»: один элемент за раз.

ЧТО СДЕЛАТЬ НА ПРАКТИКЕ

Если у вас сейчас есть агент, который «иногда работает», запишите три последние проблемы с ним. Для каждой поставьте галочку: это «модель недостаточно умна» или «обвязка недостаточна». Подсказка: если проблема в том, что модель врёт, забывает или делает не то — это обвязка. Если проблема в том, что модель не понимает задачу даже после подробного объяснения — это модель. Большинство проблем окажется в обвязке.

ИТОГ ГЛАВЫ

- Сильная модель не заменяет harness. Надёжность агента — свойство системы, а не модели.
- Смену модели имеет смысл делать только после того, как harness выстроен, но модели не хватает ума для конкретной задачи.
- Инструменты нельзя заменить промптами. Промпт — текст, инструмент — действие.
- Галлюцинации часто возникают там, где у модели нет инструмента проверить реальность, и она достраивает её текстом.

→ Теперь, когда понятно, что harness — не аксессуар, а необходимость, пора разобрать его устройство. Из чего он состоит и как работает цикл — в следующей главе.

Глава 4. Harness — руль и тормоза вашего агента

Мы подошли к центральной концепции книги. Если вы поймёте эту главу, остальное станет почти очевидным. Если не поймёте — остальное превратится в набор рецептов без причины. Поэтому я разберу harness медленно, по «грамму».

Вернёмся к аналогии из третьей главы: сильная модель без harness — это мощный автомобиль без руля и тормозов. Двигатель работает, колёса крутятся, но куда едет машина — зависит от уклона дороги и случайных препятствий. Может, доедет. Может, в кювет. Зависит не от мощности двигателя, а от того, что его направляет. Harness — это руль, тормоза и приборная панель в одном комплекте.

Что именно harness делает? Я бы выделил четыре функции, без которых он не harness.

ЧТО ДЕЛАЕТ HARNESS

1. Задаёт границы. Явно прописывает, что агент может делать, а что — нет. Не «будь аккуратнее», а «не запускай команды с sudo». 2. Верифицирует результат. Требует от агента проверять свою работу: запустить тесты, проверить файл, подтвердить статус — прежде чем объявить задачу выполненной. 3. Сохраняет контекст. Заставляет агента записывать каждый значительный шаг, чтобы не потерять нить между сессиями. 4. Даёт наблюдаемость. Позволяет вам — оператору — видеть, что агент делает в каждый момент, через логи и файлы прогресса.

Каждая из этих функций отвечает за конкретный класс ошибок. Границы — за опасные действия. Верификация — за «преждевременный успех». Контекст — за забывание. Наблюдаемость — за «непонятно, что он вообще делает». Если убрать любую — соответствующий класс ошибок возвращается.

Теперь — самое важное про harness. Я уже говорил это в третьей главе, но повторю: harness не делает модель умнее. Это не «обучение», не «fine-tuning», не «улучшенный промпт». Harness — это инженерное окружение, в котором модель работает. Модель остаётся той же; меняется система вокруг неё. Поэтому harness можно ставить на любую модель — от маленькой локальной до GPT-4o — и эффект будет один и тот же: агент становится надёжнее.

Цикл работы harness

Хороший harness устроен циклически. Это не файл, который написали один раз и забыли, — это петля, которая прогоняется на каждом шаге работы агента. Цикл состоит из пяти фаз.

Конец ознакомительного фрагмента.