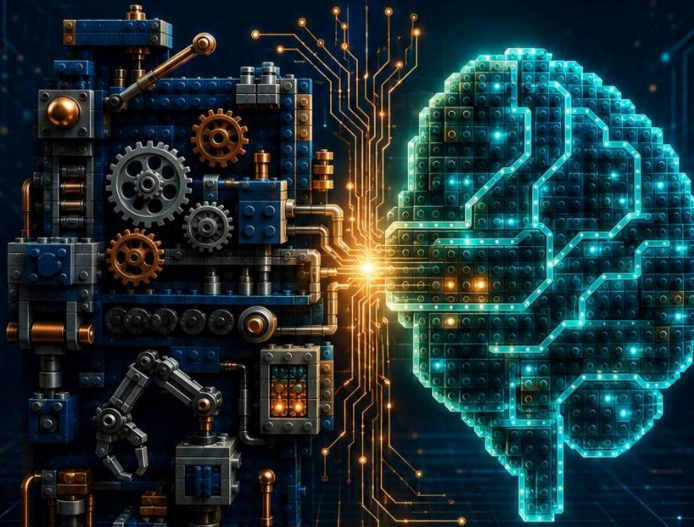
ПОЛЕВОЙ ГИД ПО АРХИТЕКТУРЕ СОВРЕМЕННОГО AI-АГЕНТА

Десять исследований arXiv,
которые изменят ваш AI-продукт в 2026 году



Павел Новопольцев

Полевой гид по архитектуре современного AI-агента

https://litres.ru/74300188
SelfPub; 2026

Аннотация

Почему один и тот же искусственный интеллект в руках одной команды работает, а в руках другой нет?Не потому что одна команда покупает более дорогую модель и не потому что у нее есть секретные слова для промптов. Дело в обвязке.

Обвязка - это всё, что окружает языковую модель в работающей системе: системный промпт, набор инструментов, способ хранения контекста между сессиями, цикл обратной связи, в котором модель проверяет собственные ответы и пересобирает стратегию

Десять свежих исследований об архитектуре AI-агентов, переведённые в практику. JSON умирает, Python-стабы побеждают.

Промпт с «подумай глубже» сжигает бюджет в 18 раз.

Полная история диалога ломает агента.

Память, которая помнит нужное, повышает успех с 79 до 96 процентов.

Книга для тех, кто внедряет AI-агентов в продакшн и хочет перестать палить деньги.

Содержание

Павел Новопольцев
Полевой гид по архитектуре современного AI-агента

Введение

Эта книга о том, почему один и тот же искусственный интеллект в руках одной команды работает, а в руках другой нет. Не потому что одна команда покупает более дорогую модель. Не потому что у нее лучше промпт-инженеры. И уж точно не потому, что она знает секретные слова, которые заставляют нейросеть «стараться сильнее».

Дело в обвязке.

Обвязка - это все, что окружает языковую модель в работающей системе: системный промпт, набор инструментов, способ хранения контекста между сессиями, цикл обратной связи, в котором модель проверяет собственные ответы и пересобирает стратегию. Модель - это мозг. Обвязка это руки, инструменты, память и рабочий стол. И, как показывают свежие исследования, разница между «способной моделью с плохой обвязкой» и «средней моделью с хорошей» в 2026

году огромна.

Эту книгу я писал для тех, кто внедряет AI-агентов в реальные продукты. Для инженеров, которые видят, что агент «вроде работает», но жжёт бюджет и проваливает простые задачи. Для продакт-менеджеров, которые не понимают, почему замена модели на более дорогую не дала прироста. Для основателей небольших студий, которые хотят разговаривать с разработчиками на одном языке, когда речь идет об архитектуре агентской системы.

Здесь нет кода, который нельзя запустить, и нет формул, которые нельзя объяснить. Здесь есть десять исследований, отобранных по одному критерию: применимо ли это завтра в вашем продукте. Каждое из них опубликовано на arXiv в июле или августе 2026 года, и каждое меняет что-то в том, как нужно строить агентов.

AgentExecutor и E3:
Проектирование Сверхэффективных ИИ-Агентов
ТРЕХФАЗНЫЙ ПРОЦЕСС AgentExecutor
ИЗОЛИРОВАННОЕ ОКРУЖЕНИЕ
ГЕНЕРАТОРЫ ПРЕФИКСОВ
ФАЗА 1: EnxAgent — Подготовиа среды
Агент нннциализирует навлированное коружение, устенqвчввист псаяскмости (перез ит) и сотдает недостающие ресурсы.
ФАЗА 2: RofAgent — Итеративное уточнение
Данемическое исследовение путей выполнения к использованиеу «дружолюбной» обротний сяязи и стеяченивм контекста (proning).
ФАЗА 3: EvoAgant — Снитез гeператоров
Синтетирует программу-танератор, которея вихтемитически парвбирехт прчшие варнанты префиясов для достижених мохсимального покрытия.
СТРАТЕГИЯ E3: ЭКОНОМНОЕ МЫШЛЕНИЕ
ESTIMATE (Dцениа) — Определение сложности
Агент иодниваят необходнмый объем информации поред, нечезаю работы, нхбегвл стратегия «максимального контекста» для престых задач.
EXECUTE (Выполнение) — Минимальный путь
Запуси камого керетного недаиного пути вынолнения. Это появвпахт звсномить до 91% токенов на првстых нрвскех неда.
EXPAND (Росширение) — Адаптивный роот
Область пияхка и контевст раgширяются тольке тегда, когда провкрао мнивмегвного нсти саберннкь наудачей.
РЕЗУЛЬТАТЫ И ЭФФЕКТИВНОСТЬ
94%
ПОКРЫТИЕ КОДА
Рекардный уровень, +19.9% к Treefix
СНИЖЕНИЕ ЗАТРАТ НА 80%
-80%
ПРЕДЫДУЩИЙ Treefix
AgentExecutor + E3
Краже оыполнения: -Б0.3%
Денежные аатраты API: -Б6.5%
<1.0
бесполезная работа
полезная работа
ACRR — Кохффициент когинтивной иабыточности
Матрина, измераницах обьам «беспелканей» работы агенга. Методы E3 к AgentExecutor стрематся свести этот показатель к минимуму.
Gemini Notebook

Что вы получите

Книга состоит из трех частей.

В первой части мы разберемся, почему обвязка важнее модели, и увидим два конкретных способа, которыми плохая обвязка буквально сжигает деньги. Это главы о JSON-вызовах инструментов, которые проигрывают Python-стабам, и о промптах, которые заставляют агента работать в семь раз больше, чем нужно.

Во второй части перейдем к архитектуре. Научим агента понимать, насколько задача сложная, и не читать весь репозиторий ради однострочной правки. Построим память, которая помнит нужное и не забывает лишнее. Защитимся от ситуации, когда старая история диалога ломает агента, и научимся делать retrieval так, чтобы контекст не раздувался.

В третьей части поговорим о циклах, которые улучшают сами себя. Агент, который переписывает собственную обвязку в рантайме. Система из нескольких агентов, которые эволюционируют общую обвязку на своих данных и делятся находками. И, наконец, метрики, по которым вообще можно понять, что обвязка стала лучше, а не просто «вроде работает».

Как читать эту книгу

Книгу можно читать последовательно, от введения до заключения. Главы выстроены так, что каждая опирается на предыдущую и готовит следующую. Но если вы опытный инженер и вам нужна конкретная тема, можно начать с нее каждая глава самодостаточна, и в конце есть мостик к следующей, чтобы вы не потерялись.

Если вы читаете книгу, чтобы внедрить что-то конкретное в понедельник утром, в каждой главе есть блок «Что сделать прямо сейчас» с конкретными шагами. Если вы читаете, чтобы понять, как устроено современное AI-внедрение, эти блоки можно пропустить.

В конце книги есть заключение, которое называется «Что делать в понедельник утром». В нем собран один список из десяти шагов, ранжированных по приоритету. Если у вас есть пятнадцать минут и работающий AI-агент, начните оттуда.

Чего в этой книге нет

Здесь нет гимна возможностям LLM. Здесь нет новостей про очередную модель, обогнавшую предыдущую. Здесь нет обещаний, что через год агенты заменят всех разработчиков.

Здесь есть трезвая инженерная работа. Архитектура. Метрики. Циклы обратной связи. Экономика токенов. И десять конкретных исследований, каждое из которых улучшит вашу систему, если вы внедрите его результаты.

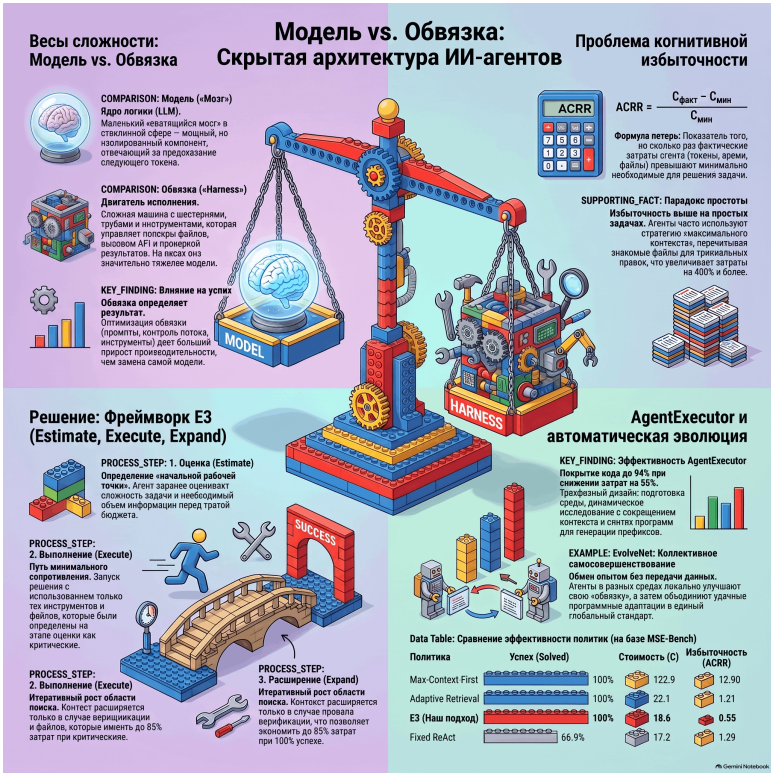
Давайте начнем.

Глава 1. Горький урок 2026 модель это тридцать процентов, обвязка семьдесят

В августе 2026 года исследователи из Scale AI опубликовали работу под названием HarnessOpt-Bench.

Они поставили простой, но неприятный вопрос если дать разным языковым моделям одинаковую задачу и одинаковую обвязку, кто выиграет?

А если дать одинаковую модель, но разные обвязки, кто выиграет тогда?



Ответ оказался таким обвязка важнее.

В эксперименте участвовали пять frontier-моделей: Claude Opus 5, Claude Sonnet 5, Kimi K3, GPT-5.6-Sol и GPT-5.6-Terra. Все они решали четыре разные задачи GAIA (многошаговое исследование с веб-инструментами), OfficeQA Pro (корпоративное рассуждение над документа-

ми), BrowseComp-Plus (*глубокий web-research*) и Terminal-Bench 2.0 (задачи command-line).

Каждая модель работала в двух режимах: под общей coding-обвязкой и под своей нативной обвязкой. Сто одиннадцать зачетных прогонов. И результат, который стоит запомнить.

Когда меняли модель, эффект на качество был огромным.

Claude Opus 5 стабильно превосходил остальных. Когда меняли обвязку, эффект тоже был, но меньше. Но и это критическая оговорка оптимизация обвязки, выполненная правильно, давала прирост, **сопоставимый** с разницей между лучшей и худшей моделью.

Переведем на человеческий. Если у вас есть средняя модель и хорошая обвязка, вы обгоняете тех, у кого топовая модель и плохая обвязка. Если у вас топовая модель и плохая обвязка, вы проигрываете.

Почему это важно

Последние пять лет мы жили в парадигме «купить более умную модель». Бенчмарки росли, новости выходили, маркетологи продавали. Но между «модель хорошо решает задачу в лаборатории» и «агент хорошо решает задачу в продакшне» — пропасть. Эта пропасть и заполняется обвязкой.

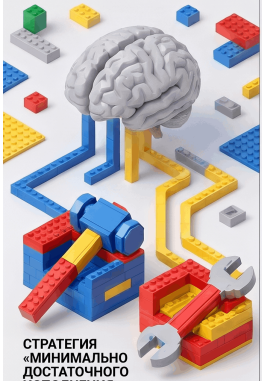
Обвязка делает три вещи, которые модель сама по себе делать не умеет.

Во-первых, обвязка превращает абстрактные способности модели в конкретные действия. Модель умеет рассуждать. Но чтобы рассуждение превратилось в ответ клиенту, нужны: правильный формат вывода, проверка через инструмент, переспрос при неоднозначности, логирование для отладки. Все это — обвязка.

Во-вторых, обвязка ограничивает то, что модель может сломать. Модель умеет генерировать SQL. Но чтобы SQL не угробил базу клиента, нужны белый список таблиц, проверка синтаксиса, тестовый прогон на копии, откат при ошибке. Все это обвязка.

В-третьих, обвязка создает экономику. Модель стоит одинаково на любом запросе. Но архитектура вызова может сделать этот запрос в десять раз дешевле или в десять раз дороже. И выбрать дешевый способ — это инженерное решение, а не магия модели.

ЭФФЕКТИВНОСТЬ ИИ-АГЕНТОВ: КАК ИЗБЕЖАТЬ «ИЗБЫТОЧНОГО МЫШЛЕНИЯ»



СТРАТЕГИЯ «МИНИМАЛЬНО ДОСТАТОЧНОГО ИСПОЛНЕНИЯ»

Агент сначала оценивает сложность задачи и строит кратчайший путь, прежде чем тратить бюджет.

ДИНАМИЧЕСКОЕ ИССЛЕДОВАНИЕ СРЕДЫ

Инструменты AgentExecutor позволяют агенту самостоятельно создавать файлы и настраивать зависимости «на лету».

ИНСТРУМЕНТАЛЬНЫЙ СТЕК АГЕНТА

Использование Python-скриптов и Bash-команд для активного взаимодействия с кодом.

ПРОГРЕССИВНОЕ РАСШИРЕНИЕ (EXPAND)

Область поиска и контекст расширяются только в случае неудачи начального плана.

ВЫБОРОЧНАЯ ПАМЯТЬ (SELECTIVE MEMORY)

Система сохраняет спецификации задач и схем данных, но отбрасывает лишние логи рассуждений.

ЗАЩИТА ОТ «ЗАСОРЕНИЯ» ИСТОРИИ

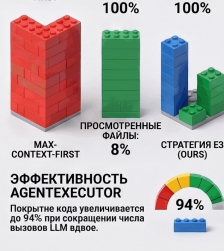
Удаление нерелевантных шагов предотвращает деградацию качества ответов при длительных сессиях.

ACRR: КОЭФФИЦИЕНТ ИЗБЫТОЧНОСТИ

Избыточность мышления наиболее высока на самых простых задачах.

СНИЖЕНИЕ ЗАТРАТ НА 85%

Фреймворк E3 сокращает расходы на токены и время выполнения без потери успеха.



ЭФФЕКТИВНОСТЬ AGENTEXECUTOR

Покрытие кода увеличивается до 94% при сокращении числа вызовов LLM вдвое.

Цифры, которые изменят ваш подход

Возьмите эти числа и запомните. Они из разных исследований, но все сходятся в одном.

Claude Opus 5 под плохой обвязкой проигрывает Claude Sonnet 5 под хорошей обвязке. Это не исключение. Это правило.

Хорошая обвязка может сократить расход токенов в десять раз без потери качества. Или улучшить качество в два раза при том же расходе. Или и то и другое одновременно.

Пять-тридцать раз -это разброс стоимости одной успешной задачи в зависимости от обвязки, которую вы выбрали. Те же модели, те же задачи — разная обвязка, разная экономика.

Это значит, что бюджет на AI-продукт нужно планировать не как «закупка API», а как «инвестиции в обвязку». Если вы тратите сто тысяч рублей в месяц на API и не тратите ничего на обвязку, вы делаете ровно наоборот.

Что считать обвязкой

Чтобы дальше в книге не было путаницы, давайте зафиксируем, что входит в это понятие.

Обвязка - это программа, которая вызывает языковую модель, передает ей контекст, получает ответ, проверяет его и либо возвращает пользователю, либо переспрашивает модель. Звучит просто, но в реальности обвязка — это десятки и сотни решений.

-Какой системный промпт использовать.

-Какие инструменты дать модели.

-В каком формате она должна вернуть ответ.

-Как обрабатывать ошибки.

-Как хранить историю диалога.

-Как выбирать между несколькими моделями.

-Как считать бюджет. Как логировать.

-Как тестировать.

-Как версионировать.

-Как катить в продакшн.

-Как откатывать, если сломалось.

Каждое из этих решений - это строка в счете за API.

Каждое решение -это потенциальный баг. Каждое решение - это точка, где ваш агент либо работает, либо нет.

Большинство команд, которые внедряют AI-агентов, тратят девяносто процентов времени на выбор модели и десять

процентов на обвязку. Это перевернутая пропорция. Правильная - наоборот.

Что вы узнаете в следующих главах

В следующей главе мы разберем одну конкретную часть обвязки вызов инструментов.

И увидим, что JSON-вызов, который сейчас использует большинство, проигрывает Python-стабам в одиннадцати из четырнадцати моделей.

Это не «иногда лучше» это «почти всегда лучше».

В третьей главе мы посмотрим, как плохо написанный промпт может заставить агента тратить в восемнадцать раз больше токенов без улучшения качества.

И как одно предложение в системном промпте это лечит.

Вторая часть книги -про архитектуру думающего агента. Мы научим агента понимать, насколько задача сложная, и не делать лишней работы. Построим память, которая помнит нужное. Защитимся от ситуации, когда старая история диалога ломает агента.

Третья часть - про циклы, которые улучшают сами себя. Агент, который переписывает собственную обвязку. Система из нескольких агентов, которые эволюционируют общую обвязку. Метрики, по которым это вообще можно измерить.

Все это не теория, все это результаты, опубликованные в 2026 году, с цифрами, бенчмарками и кодом, который можно скачать и запустить.

Все это применимо к вашему агенту в понедельник утром.

Что сделать прямо сейчас

Прежде чем мы двинемся дальше, одна конкретная вещь, которую вы можете сделать сегодня.

Откройте свой текущий проект с AI-агентом.

Найдите место, где вы вызываете языковую модель. Посмотрите, сколько строк кода вокруг этого вызова. Это и есть ваша текущая обвязка. Если там меньше пятидесяти строк, у вас, скорее всего, нет обвязки. У вас есть тонкий слой поверх API.

Это нормально, если вы только начали. Это проблема, если вы внедряете в продакшн. Потому что продакшн -это про обвязку. Модель дает вам базовую способность. Обвязка превращает способность в продукт.

Запомните этот тезис. Мы будем возвращаться к нему в каждой главе.

Дальше - глава 2, в которой мы разберемся, почему классический JSON-вызов инструментов уже проиграл, и что с этим делать.

Глава 2. Конец JSON как Python-стабы обходят классический function calling

В шестом августа 2026 года вышла работа под названием The Bitter Lesson of Tool Calling. Название говорящее.

Авторы Ишан Патель, Сахил Сен, Элиас Лумер и Вамсе Кумар Суббия из PricewaterhouseCoopers сравнили два способа дать модели доступ к инструментам.

Результат оказался настолько убедительным, что теперь, спустя несколько недель после публикации, любой инженер, который продолжает использовать старый подход, просто теряет деньги клиента.

Эволюция ИИ-агентов: От жесткого JSON к программному коду

Сравнение подходов: Старая школа vs. Новая эра

Старая школа

Многоходовые задержки

Каждый шаг требует отдельного цикла работы LLM, что замедляет сложные цепочки действий.

Чрезмерный расход токенов

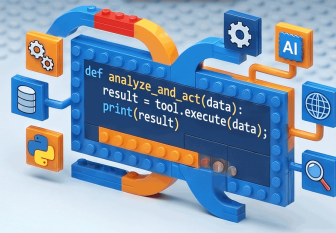
13-кратная когнитивная избыточность

Традиционные агенты переходят весь контекст, увеличивая расходы на **90%** без пользы для результата.

Обрыв вызовов при N > 70

Нативные JSON-парадигмы часто теряют данные при высокой параллельности задач.

Новая эра



Код как универсальный интерфейс

Использование Python-заглушек повышает точность длинных цепочек на **18.8%**.

Стратегия E3 (Оценка → Выполнение → Расширение)

Экономия затрат на 85%

Агент сначала оценивает минимально необходимый контекст и расширяет его только при сбоях.

Фреймворк AgentExecutor

Автономная подготовка окружения и эволюция префиксов сокращают время работы на **80%**.

Эффективность новых систем

Два способа позвать инструмент

Первый способ, который сейчас используют все, называется JSON tool calling. Работает так: вы передаете модели JSON-схему, описывающую каждый инструмент. Модель возвращает структурированный JSON-объект, в котором указано имя функции и аргументы.

Ваш код ловит этот объект, вызывает функцию, возвращает результат обратно в модель. На каждом вызове отдельный ход инференса.

Если нужно вызвать пять инструментов подряд, это пять ходов. Если нужно распараллелить семь вызовов, модель должна эмитировать семь JSON-объектов за один ответ, и на больших числах она начинает их терять.

Второй способ - programmatic tool calling, или PTC.

Здесь вы передаете модели не схему, а исходный код Python-модуля со стабами функций.

Стаб - это функция с типизированной сигнатурой, которая при вызове делает ровно то, что вы хотите. Модель пишет обычный Python-скрипт, импортирует нужные стабы, вызывает их с аргументами, обрабатывает результаты.

На все семь параллельных вызовов один скрипт, один subprocess, один ход инференса.

Звучит как косметическое отличие. На практике это разница в одиннадцать из четырнадцати протестированных мо-

делей.

Цифры, которые изменят ваш подход к инструментам

Авторы протестировали четырнадцать языковых моделей на бенчмарке BFCL v4, который специально создан для оценки **function calling**.

Среди моделей - пять моделей семейства Claude (включая Opus 5 и Sonnet 5), несколько поколений GPT, модели от Google и DeepSeek.

Каждая модель запускалась в двух режимах нативный **JSON tool calling** и PTC с Python-стабами.

Одиннадцать из четырнадцати моделей в режиме PTC показали результаты не хуже, чем в режиме JSON. Семейство GPT-5.6 дало прибавку в десять и шесть десятых процента. Тринадцать из четырнадцати моделей не уступали PTC в режиме параллельного fan-out то есть когда нужно одновременно вызвать много инструментов.

Конец ознакомительного фрагмента.