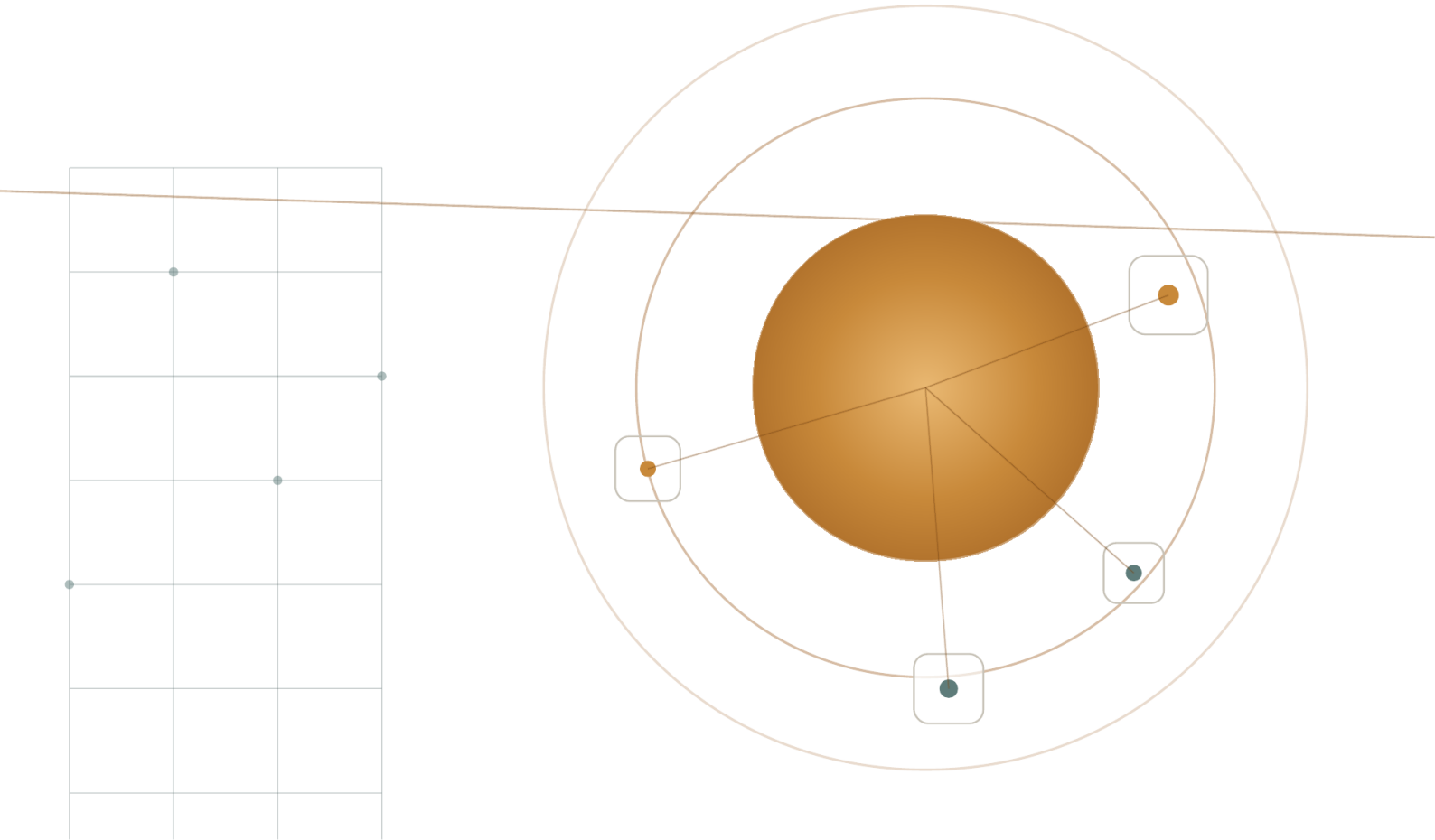
ПОД КАПОТОМ

Как стать ИИ-инженером в 2026

Честный маршрут для тех, кто хочет внедрять ИИ в бизнес, а не защищать диссертацию.



Under the Hood · Илья Утов
@gorilla_under_hood

Содержание

Как пользоваться этим гайдом

Список курсов ищи в другом месте. Здесь карта решения: что выбрать и в каком порядке. Читай по порядку:

1. Сначала пойми, **какая из двух профессий** тебе вообще нужна (раздел 1). Их путают все, и из-за этого люди годами учат не то.
2. Потом пройди **развилку ролей** и честно ответь, кто ты сейчас (раздел 2). От этого зависит весь маршрут.
3. Дальше бери **карту навыков** (раздел 3), пойми **способ работы** (раздел 4) и собери **первый проект** (раздел 6) как руководство к действию.
4. Раздел 7 это пошаговый план под твою стартовую точку.
5. Разделы 9-11 про найм, деньги и про то, что из всего этого хайп, а что реальная ценность.

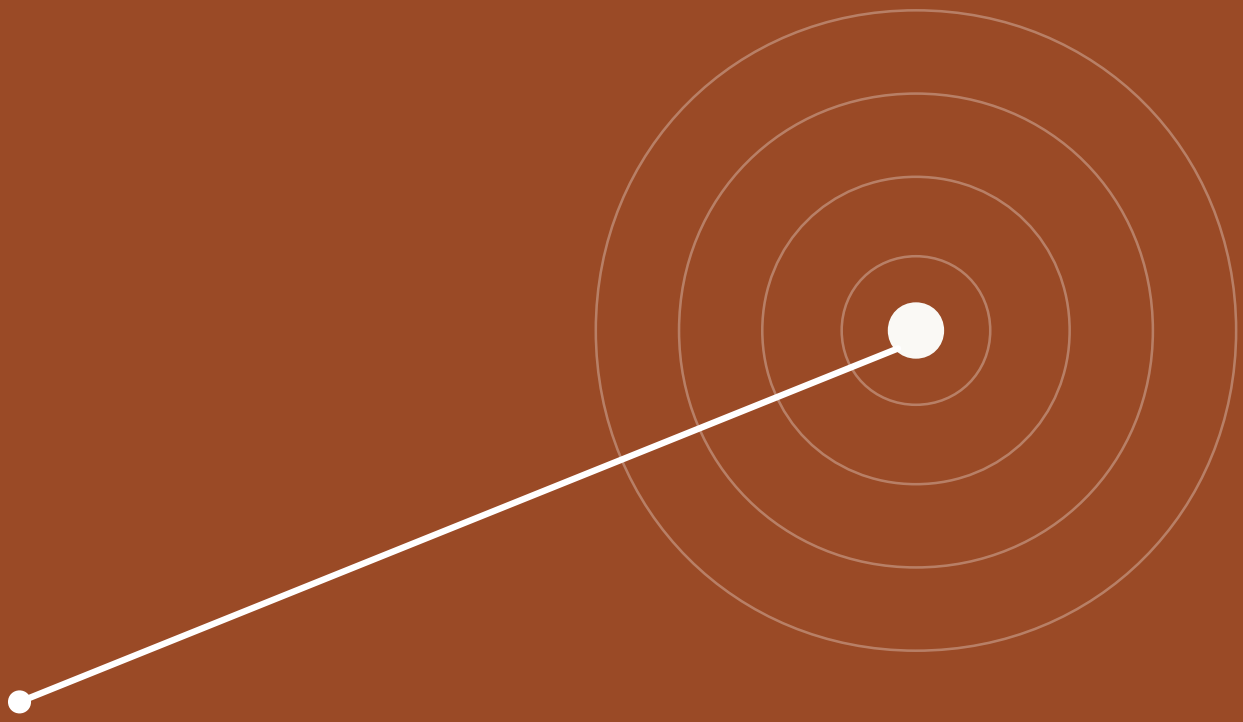
Если времени мало, читай разделы 1, 2, 4, 6 и чек-лист в конце. Этого хватит, чтобы не свернуть не туда.

ЧАСТЬ · I

I

Куда целиться

Две профессии под одним словом и развилка ролей. С чего начинается выбор.



1. Две плоскости: кто ДЕЛАЕТ ИИ и кто ВНЕДРЯЕТ

Самая дорогая ошибка новичка: услышать «ИИ-инженер» и решить, что надо учить машинное обучение, матанализ и нейросети с нуля. В большинстве случаев это не так.

Под одним названием прячутся две совершенно разные профессии.

Плоскость №1: те, кто делает ИИ. Они обучают модели. Это исследователи и ML-инженеры: PhD, тяжёлая математика, эксперименты на GPU-кластерах за миллионы долларов. Таких людей в мире очень мало. По оценке swyx (эссе 2023 года), серьёзных LLM-исследователей примерно 5 000 человек на всю планету. Вход сюда долгий, академический и почти закрыт без научного бэкграунда.

Плоскость №2: те, кто внедряет ИИ. Они строят продукты поверх уже готовых моделей, обращаясь к ним через API (Claude, GPT, Gemini и другие). Они не обучают модель, они её используют как электричество из розетки. Потенциально это все программисты мира, а их около 50 миллионов. Ни одного PhD не требуется.

Термин «AI Engineer» в его нынешнем смысле задал инженер по имени swyx (Shawn Wang) в эссе «The Rise of the AI Engineer». На него ссылается вся индустрия. Его формулировка:

> «software engineering породит новую поддисциплину, специализирующуюся на применении ИИ через новый стек инструментов».

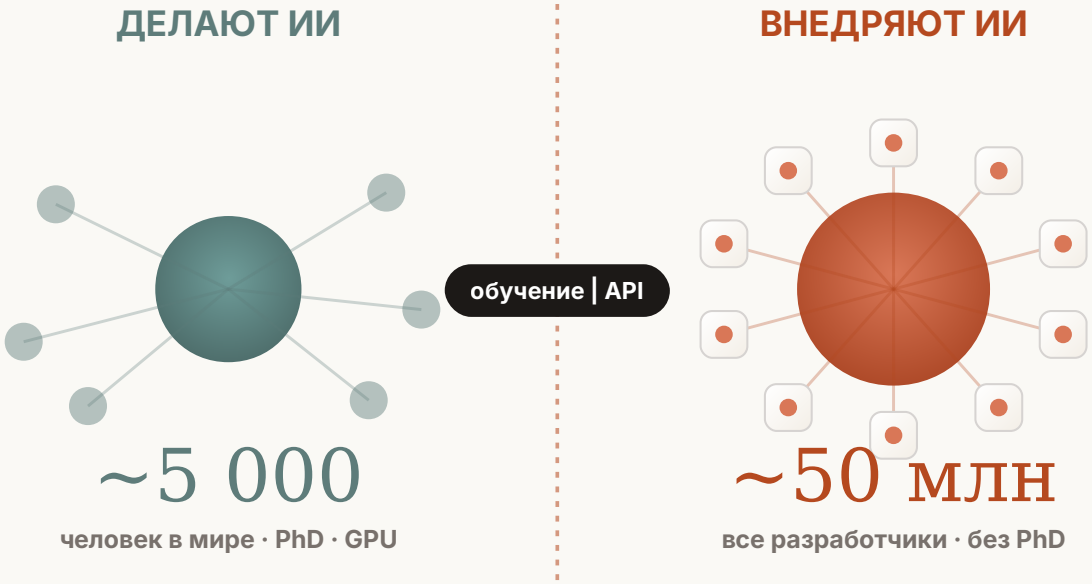
И ключевая цитата, которую стоит запомнить, от Андрея Карпати (одного из самых известных людей в ИИ):

> «В этой роли можно быть весьма успешным, ни разу ничего не обучив».

Граница между двумя плоскостями ровно одна:

> **Обучаешь ты модель или потребляешь её через API.**

Всё, чем пугают новичка (RAG, агенты, векторные базы, эмбеддинги, evals), целиком живёт во второй плоскости. Никакой науки, чистая инженерия поверх готового.

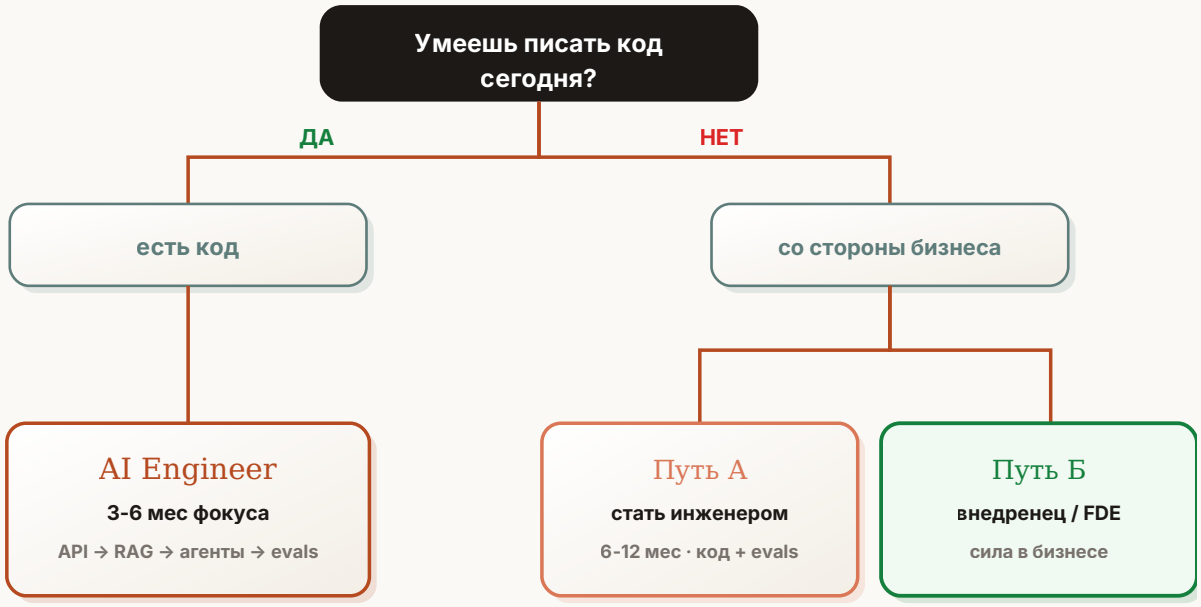


swyx прогнозирует (в эссе 2023 года, горизонт около 2028), что соотношение вакансий «ML-инженер» к «AI-инженер» перевернётся в пользу вторых. Карпати с этим тезисом согласился публично. Простыми словами: рынок труда массово смещается во вторую плоскость. Ты заходишь на растущую волну, а не на тонущую.

Вывод раздела: если твоя цель «автоматизировать бизнес и внедрять ИИ», тебе нужна плоскость №2. Забудь про «выучить ML». Это другая профессия.

2. Какая роль тебе нужна на самом деле (развилка)

Внутри второй плоскости тоже есть варианты, и выбрать правильный важнее, чем выбрать язык программирования. Ответь себе честно на один вопрос: **умеешь ли ты писать код сегодня?**



Если ты разработчик (backend, frontend, аналитик с кодом)

Тебе нужна роль **AI Engineer / AI-инженер-внедренец.** У тебя уже есть код и продакшен-мышление. Не хватает только верхнего слоя: LLM API, RAG, агенты, evals. Это самый быстрый путь, реалистично 3-6 месяцев фокусной практики. Иди прямо в раздел 3.

Если ты со стороны бизнеса, без кода

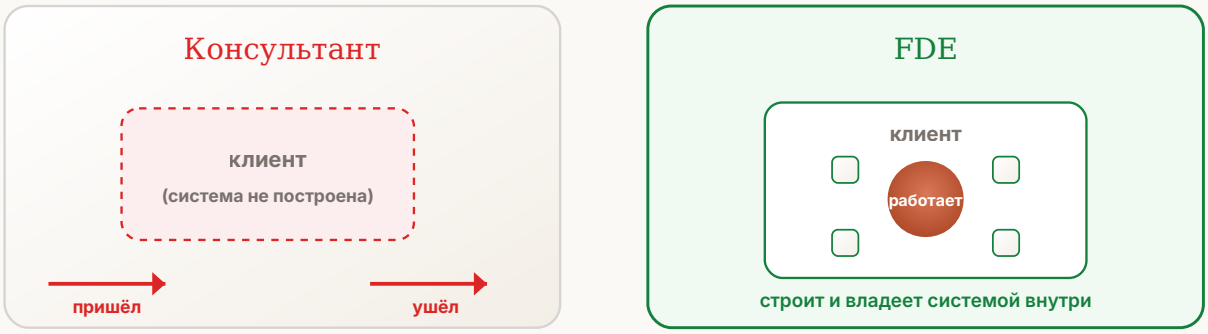
Вот здесь главная развилка, и большинство гайдов её замалчивают. У тебя не один путь, а два, и они ведут в разные стороны.

Путь А: всё-таки стать инженером. Придётся освоить код (Python на базовом уровне), а затем стек: API, RAG, агенты, evals. No-code инструменты (n8n, Make, Zapier) хороши как вход, чтобы руками пощупать автоматизацию и понять логику. Но у них низкий потолок: качество и измеримость на них не построить, а платят как раз за это. Срок честно длинный: считай от 6 до 12 месяцев, если начинаешь с нуля.

Путь Б: целиться не в «инженера», а во внедренца. Это роль для человека, чья сила в бизнесе, а не в коде. Ты понимаешь задачу клиента, владеешь его данными, отвечаешь за результат в проде. Код подтягиваешь по дороге как инструмент, но он не твой главный актив. В мире эту роль уже оформили официально, и у неё есть имя: **Forward-Deployed Engineer.**

Не пытайся идти обоими путями сразу. Выбери тот, где у тебя уже есть фора.

Forward-Deployed Engineer (FDE): внедренец в чистом виде



Это самый «бизнесовый» конец второй плоскости, и для человека из бизнеса это часто лучшая цель.

FDE это не консультант. Консультант приходит, отдаёт презентацию с рекомендациями и уходит. FDE приходит и **строит работающую систему прямо внутри клиента**, а потом ею владеет и отвечает за неё. Модель придумала компания Palantir ещё в начале 2010-х. Любопытный факт: какое-то время у Palantir было больше forward-deployed инженеров, чем обычных разработчиков.

В 2026 эту роль массово нанимают сами лаборатории: OpenAI, Anthropic (у них это называется Applied AI), Google. Не нишевая экзотика: за такими людьми лидеры рынка прямо охотятся.

Стек FDE в 2026:

- RAG-пайплайны (превращение документов клиента в базу знаний для ИИ),
- eval-фреймворки (как ловить галлюцинации и проверять качество),
- агентные фреймворки (LangGraph, CrewAI и подобные),
- observability (мониторинг того, что система делает в проде),
- безопасность и комплаенс для систем внутри клиента,

› продакшен-промптинг.

Заметь: половина этого не про код в классическом смысле, а про **понимание задачи и владение качеством**. Поэтому для бизнес-человека это достижимая и при этом высокооплачиваемая цель.

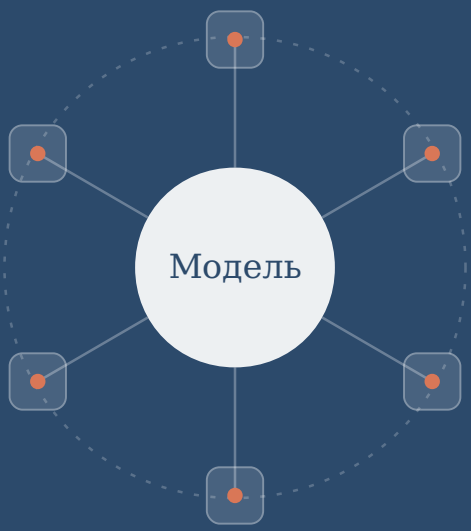
Если коротко: определись, кто ты. Разработчик идёт в AI Engineer. Бизнес-человек выбирает между долгим путём в инженеры и более естественным путём во внедренцы (FDE). Дальше гайд работает для обоих, но бизнес-человеку стоит читать с акцентом на разделы про evals, бизнес-контекст и владение данными, а не на код.

ЧАСТЬ · II

II

Как это работает

Навыки, способ работы и главный навык, который отличает профи: evals.



3. Карта навыков внедренца 2026

Это полный список того, что реально нужно, по слоям. Почти всё про интеграцию и API, а не про обучение моделей. Колонка «приоритет» подскажет, на что налегать.

Базовый код	минимум
LLM API	ядро
Промпт / context engineering	ядро
Эмбеддинги + Vector DB	высокий
RAG	высокий
Агенты · tool use · MCP	растёт
Evals · измерение качества	отличает профи
Observability	почти обязателен
Данные под RAG	недооценён
Экономика · cost / latency	растёт в проде
Fine-tuning	редко (57% не делают)

Несколько слоёв распакую подробнее.

Базовый код

Тебе не нужен senior-уровень. Нужно уметь читать и писать на Python настолько, чтобы дёргать API, обрабатывать данные и склеивать части. Это недели практики, не годы. Бизнес-человеку этого слоя достаточно на уровне «понимаю, что происходит, и могу поправить».

LLM API

Это твоё «электричество». Ты отправляешь модели текст (промпт) и получаешь ответ. Научись работать хотя бы с одним провайдером (Claude или OpenAI), понимать токены, температуру, системные промпты и структурированный вывод (когда модель отвечает строго в нужном формате, например JSON).

RAG: самый частый паттерн внедрения

RAG это способ заставить модель отвечать по твоим документам, а не по абстрактным знаниям из интернета. Логика простая:

1. Берёшь документы клиента (инструкции, базу знаний, договоры).
2. Режешь их на куски (чанкинг) и превращаешь в эмбеддинги (числовые представления смысла).
3. Складываешь в векторную базу.
4. Когда приходит вопрос, находишь самые подходящие куски и подсовываешь их модели вместе с вопросом.
5. Модель отвечает, опираясь на найденное.

С этого паттерна начинается большинство реальных внедрений. Если освоишь только RAG и evals к нему, ты уже полезен бизнесу.

Агенты

Агент это модель, которой дали инструменты (поиск, калькулятор, доступ к базе, отправку письма) и право самой решать, каким инструментом воспользоваться. Здесь важны вызов функций (function calling / tool use) и протокол MCP, который стал стандартом подключения инструментов к моделям. Агенты это растущая часть рынка, но строить их без следующего слоя бессмысленно.

Evals: навык, который отделяет профи от любителя

Это самое важное в гайде, поэтому ему посвящён отдельный раздел 5. Если коротко: eval это измерение качества твоего ИИ. Без него ты не знаешь, работает твой бот или врёт. Именно качество, а не выбор модели, мешает командам выводить ИИ в прод.

Fine-tuning: почему его можно почти игнорировать

Дообучение модели звучит круто, но на практике большинству оно не нужно. По опросу LangChain, **57% команд вообще не делают fine-tuning**, опираясь на базовые модели плюс промптинг и RAG. Это специализированная и недешёвая практика. Новичку трогать её не надо. Сэкономь силы.

Подтверждённые данные рынка (опрос LangChain «State of Agent Engineering», 1340 респондентов, конец 2025):

57%	32%	89%	57%
команд уже держат агентов в проде	барьер прода это качество, не модель	внедрили observability	НЕ делают fine-tuning

- **57% организаций уже держат ИИ-агентов в проде** (год назад было 51%). У крупных компаний (10 000+ сотрудников) это 67%.
- **Главный барьер вывода в прод это качество (32% назвали первой причиной).** На втором месте скорость ответа (20%). Не модель. Качество.
- **Observability почти обязательна:** 89% команд внедрили какую-то форму мониторинга, 62% делают полный трейсинг.
- **57% не делают дообучение.** Базовая модель плюс промптинг и RAG закрывают большинство задач.

Оговорка про честность: это опрос аудитории LangChain, людей, которые уже строят агентов. Выборка смещена в сторону энтузиастов, поэтому цифры скорее «потолок» отрасли, чем средняя температура по экономике. Тренд они показывают верно, абсолют завышен.

4. Как на самом деле устроена работа: harness, промптинг, эксперименты, исследования

Карта навыков выше это ингредиенты. А вот то, что отличает человека, который их перечисляет, от того, кто реально хорош: **способ работы.** Это самая недооценённая часть профессии, и именно её обычно не показывают в роадмапах.

Классическая разработка детерминирована: ты пишешь логику, программа делает ровно то, что написано. ИИ-инженерия другая. Модель недетерминирована: на один и тот же запрос она может ответить по-разному, и «правильную» конфигурацию нельзя вывести рассуждением. Её **находят экспериментом.** Из этого вытекает несколько вещей, которых нет в обычном бэкенде: каркас вокруг модели (harness), ремесло управления самой моделью (промпт- и context-инжиниринг), эксперименты и постоянное исследование.

4.1. Harness (обвязка вокруг модели)

Harness это весь код **вокруг** модели, который превращает сырую модель в работающую систему. Агентный цикл, управление контекстом, маршрутизация инструментов, повторные попытки при сбое, проверочные ворота, хранение состояния между сессиями, песочница, запись трейсов. Модель это двигатель. Harness это вся остальная машина.

Ключевая мысль, которую Anthropic повторяет в инженерных материалах: для реальных задач harness значит столько же, сколько сама модель, а часто и больше. Модель ты арендуешь, она у всех одна и та же. Твоё преимущество это обвязка, которую ты построил вокруг неё. В их разборе продакшен-harness состоит из управляющего слоя, сборщика контекста, маршрутизатора инструментов, песочницы, трейсинга и слоя оценки.

Для долгоживущих агентов (задачи на часы и дни, которые не влезают в одно контекстное окно) harness вообще становится продуктом: важнее не одна удачная формулировка промпта, а устойчивое состояние, проверочные ворота и чёткие границы ответственности между шагами.

Практически: когда ты собираешь своего RAG-бота, вся логика поиска, повторов, форматирования и ограничителей это и есть твой harness. Учись думать в терминах обвязки, а не «магического промпта». Магического промпта не существует.

Источник: Anthropic, «Effective harnesses for long-running agents».

4.2. Промпт- и context-инжиниринг (как управлять моделью)

Это ремесло, которым ты на самом деле управляешь поведением модели. В нём два уровня. Промпт-инжиниринг это как ты формулируешь конкретный запрос. Context engineering это что в принципе попадает модели в окно. Первое про слова, второе про отбор. Профи владеет обоими.

Промпт-инжиниринг: проверенные приёмы. Они кажутся очевидными, но именно их несоблюдение даёт большую часть плохих ответов.

- **Будь ясным и конкретным.** Модель не угадывает намерение. Чем точнее сказано, что нужно, в каком виде и для кого, тем лучше ответ. «Сделай хорошо» не работает.
- **Давай примеры (few-shot).** Один-три примера в формате «вход, желаемый ответ» задают планку надёжнее, чем абзац объяснений. Это самый сильный по отдаче приём.
- **Дай модели подумать.** Для задач с рассуждением попроси разложить ход мысли по шагам перед ответом. Модель, которой дали место порассуждать,

ошибается реже. В современных моделях это ещё и режим расширенного рассуждения.

- **Структурируй промпт.** Раздели инструкцию, данные и вопрос явными границами: заголовками или XML-тегами. Модель перестаёт путать, где команда, а где материал.
- **Задай роль через системный промпт.** «Ты внимательный технический редактор» меняет фокус ответа сильнее, чем кажется. Роль задаётся отдельно от запроса пользователя.
- **Проси формат явно.** Нужен список, JSON или таблица, скажи об этом прямо. Для строгого формата есть структурированный вывод по схеме.
- **Говори, что делать, а не только чего не делать.** Позитивная инструкция («ответь в три предложения») работает надёжнее запрета («не пиши много»).
- **Разбивай сложное на цепочку.** Одна большая задача в одном промпте часто проигрывает цепочке из нескольких простых шагов, где выход одного идёт на вход следующего (prompt chaining).

Context engineering: уровень выше отдельного промпта. У модели конечный бюджет внимания. Чем больше лишнего в окне, тем выше шанс, что важное утонет. Поэтому цель не «дать побольше информации», а собрать наименьший набор по-настоящему сигнальных токенов. Четыре операции: что положить, что убрать, когда сжать историю (компакция), как структурировать. В простом запросе это разовая настройка. В агенте контекст растёт на каждом шаге, и управление им становится постоянным процессом, которым занимается harness из пункта 4.1.

Как это связано с остальным. Промпт и контекст не угадывают, а подбирают экспериментом и проверяют на evals (пункт 4.3 и раздел 5). Красивая формулировка без измерения это вкусовщина. Формулировка, которая на твоём eval-наборе подняла качество с 70 до 90 процентов, это инженерия.

Частые ошибки. Расплывчатые инструкции; ноль примеров там, где два примера решили бы всё; стена текста без границ между инструкцией и данными; запихнуть в окно всё подряд на всякий случай; держать полную историю диалога, пока окно не переполнится.

Где набить руку. У Anthropic есть бесплатный интерактивный туториал по промпт-инжинирингу (GitHub: anthropics/prompt-eng-interactive-tutorial), раздел про промптинг в официальной документации и инженерная статья про context engineering. Это лучший способ перейти от теории к навыку.

Источники: документация Anthropic по промпт-инжинирингу (platform.claude.com) и статья «Effective context engineering for AI agents».

4.3. Эксперименты (эмпирический цикл вместо «один раз решил»)

Ты не «знаешь» правильный размер чанка, правильную модель или правильный промпт. Ты выдвигаешь гипотезу, прогоняешь, измеряешь, сравниваешь. Каждая настройка (размер чанка, сколько кусков доставать, какая модель, какой вариант промпта) это переменная для теста, а не решение, принятое раз и навсегда.

Важный принцип отсюда: продакшен-harness не только добавляет, он ещё и удаляет. Это и есть смысл абляций: выкинуть элемент и измерить, стало хуже или нет. Если не измеряешь, ты не инженер, а гадалка.

Именно поэтому существуют evals (следующий раздел): они и есть измерительный прибор твоих экспериментов. Эксперименты без evals это мнения. Эксперименты с evals это инженерия.

Практически: заведи журнал прогонов. Меняй по одной переменной. Делай много мелких дешёвых экспериментов, а не один большой. Это научный метод, применённый к ИИ-продуктам.

4.4. Исследования (оставаться в курсе это часть работы, а не хобби)

Поле меняется буквально еженедельно: новые модели, новые техники (протокол MCP появился и стал стандартом за считанные месяцы). Вспомни цифру из раздела про рынок: навыки в ИИ-профессиях меняются на 66% быстрее обычных. Перевод на человеческий: **читать и пробовать новое это часть оплачиваемой работы**, а не то, что делаешь по выходным.

Практически:

- следи за первоисточниками (инженерные блоги Anthropic и OpenAI, Latent Space от swyx, ключевые статьи), а не за пересказами в ленте;

- каждую новую модель прогоняй через свои собственные evals. Твой eval-набор это объективный судья: он отвечает на вопрос «эта новая модель реально лучше для моей задачи» вместо «в твиттере написали, что она топ».

И сразу про дисциплину: исследовательская грамотность это не погоня за каждой новой игрушкой. Навык в том, чтобы проверять новое на своём бенчмарке, а не внедрять, потому что хайп. Здесь research, эксперименты и harness замыкаются в один контур: читаешь новое → проверяешь экспериментом → встраиваешь в обвязку, если evals подтвердили выигрыш.

Что ещё часто забывают

Короткий список того, что не попадает в роадмапы, но всплывает в реальном проде:

- **Экономика токенов (cost и latency).** В проде важно не только «работает», но и «сколько стоит запрос» и «как быстро отвечает». Отсюда роутинг моделей (дорогая для сложного, дешёвая для простого), кэширование, контроль бюджета токенов.
- **Выбор и смена модели.** Не одна модель навсегда. Под разные задачи разные модели, и при каждом релизе стоит перепроверять выбор на своих evals.
- **Данные.** RAG хорош ровно настолько, насколько хороши данные под ним. Подготовка, чистка и структурирование данных клиента это отдельный навык, который недооценивают.
- **Безопасность агентов (prompt injection).** Пока агент просто отвечает текстом, это теория. Как только у него появляются инструменты и доступ к данным, инъекция промпта становится реальной атакой. Для FDE, который ставит систему внутрь клиента, это обязательный пункт.
- **Продуктовое мышление под недетерминированность.** Ты проектируешь систему, которая иногда ошибается. Значит, нужны мягкие отказы, человек в контуре на критичных шагах и работа с доверием пользователя. Это ближе к продакту, чем к коду, и это снова сильная сторона внедренца.

◆

5. Главный навык: evals (измерение качества)

Если ты запомнишь из всего гайда одну вещь, пусть это будет evals.

Anthropic в материале «Demystifying Evals for AI Agents» формулирует проблему прямо: без оценок команды застревают в реактивном цикле. Они ловят проблемы только в проде, где починка одного места ломает другое. После стадии прототипа разработка агента без evals начинает разваливаться.

> «building without evals starts to break down».

Почему это и есть водораздел между профессионалом и человеком, который «обмотал API в Zapier»: обмотать API может кто угодно за вечер. А вот доказать, что система отвечает правильно в 95 случаях из 100, и удержать это качество после каждого изменения, умеют единицы. За это и платят.

Как делать evals на старте (без переусложнения)

Anthropic советует начинать с малого, и это хорошая новость для новичка:

- **Начни с 20-50 простых задач**, собранных из реальных сбоев твоего агента. На ранней стадии эффект от изменений велик, и маленькой выборки достаточно.
- Дальше выборку расширяешь по мере роста.

Есть три типа «грейдеров» (тех, кто ставит оценку ответу):

На практике комбинируют. И ещё один критический навык: **чтение трейсов.** Трейс это полная запись прогона агента (что он выводил, какие инструменты звал, как рассуждал). Грейдерам нельзя слепо доверять, пока ты сам не просмотрел руками десятки трейсов и не понял, где система реально ошибается.

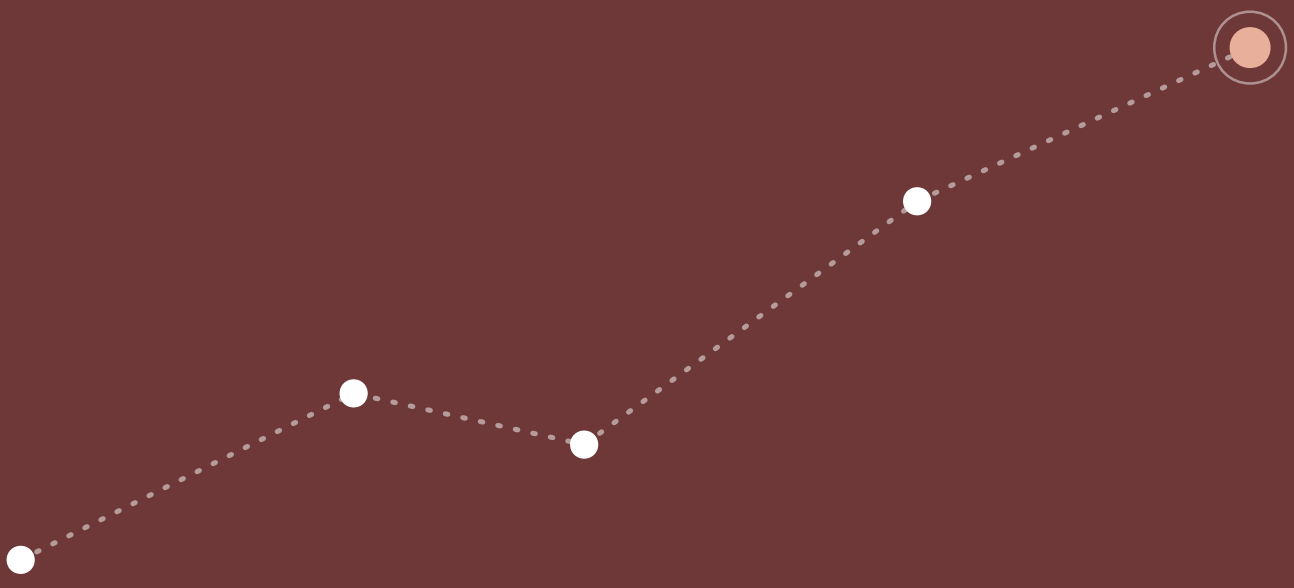
Освой этот раздел глубже остальных. Он не модный, но именно он делает тебя нанимаемым.

ЧАСТЬ · III

III

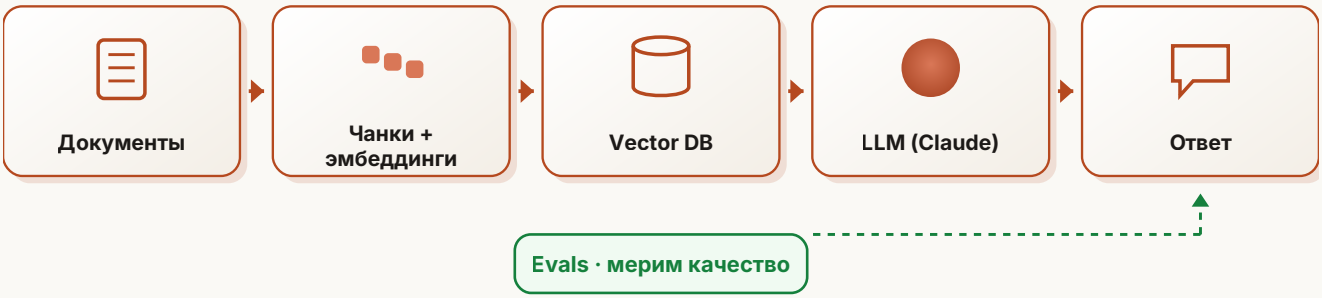
Путь в профессию

Первый проект, пошаговый план и где учиться бесплатно.



6. Первый проект: пошагово (а не «учи ML полгода»)

Теория без проекта мертва. Вот конкретный маршрут, который можно пройти на одной реальной задаче. Результат этого маршрута это не сертификат, а работающая штука на GitHub, которую можно показать на собеседовании.



Шаг 0. Выбери живую задачу

Не абстрактную. Возьми реальный бизнес-процесс: свой, знакомого предпринимателя, отдела. Например: «бот, который отвечает сотрудникам по внутренним регламентам компании» или «бот, который отвечает клиентам по базе товаров». Главное, чтобы были реальные документы и реальные вопросы.

Шаг 1. Собери RAG-бота

Минимальная сборка:

- загрузи документы,
- разрежь на куски и сделай эмбеддинги,
- положи в векторную базу (для старта подойдёт Chroma или Qdrant, они бесплатные),
- подключи LLM API (Claude или GPT),
- сделай так, чтобы на вопрос бот находил нужные куски и отвечал по ним.

Не гонись за красотой. Цель этого шага: рабочий прототип, который отвечает по твоим данным.

Шаг 2. Прикрути evals

Вот где ты обгоняешь 90% новичков:

- собери 20-50 реальных вопросов, на которые бот обязан отвечать правильно,
- для каждого запиши, какой ответ считается верным,
- прогони бота по всем вопросам и посчитай, на скольких он соврал или ушёл не туда,
- почини худшие случаи и прогони снова.

Теперь у тебя есть цифра качества и способ её улучшать. Это и есть профессиональная работа.

Шаг 3. Добавь наблюдаемость

Подключи простой трейсинг (например, LangSmith или аналог), чтобы видеть, что бот делает внутри на каждом запросе. Научись читать эти записи и находить, где он спотыкается.

Шаг 4. Выложи на GitHub и опиши

Создай репозиторий, положи код, напиши понятный README: какую задачу решает, как устроено, какие цифры качества получились, что было сложно. На найме это **бьёт любой диплом и любой дорогой сертификат**. Рынок смотрит на то, что ты запустил, а не на бейдж.

Когда этот маршрут пройден, ты уже говоришь на языке роли. На собеседовании 2026 спрашивают не «знаешь ли PyTorch», а «как ты мерил качество и как дебажил галлюцинации». На оба вопроса у тебя будет реальный ответ.

7. Пошаговый план: от первого вызова до отклика

Это последовательность шагов, а не календарь. Иди по порядку и переходи к следующему шагу, когда предыдущий реально заработал. Скорость у всех своя.

Базовый трек (если уже пишешь код)

- **Шаг 1. LLM API и промптинг.** Прогони простые вызовы, освой системные промпты и структурированный вывод.
- **Шаг 2. RAG.** Собери первого бота по своим документам (Шаг 1 из раздела 6).
- **Шаг 3. Evals и observability.** Прикрути измерение качества и трейсинг (Шаги 2-3 из раздела 6).
- **Шаг 4. Агенты.** Добавь боту инструменты (tool use, MCP), сделай его агентным.
- **Шаг 5. Витрина.** Оформи проект на GitHub, напиши разбор, начни откликаться на вакансии.

Если ты аналитик / data без продакшена

Перед Шагом 1 добавь инженерную часть: работа с API, базовый деплой, окружение. У тебя есть данные и метрики, не хватает «довести до прода».

Если ты со стороны бизнеса, без кода

- **Сначала no-code.** Собери пару автоматизаций в n8n или Make, чтобы понять логику пайплайнов. Параллельно подтяни базовый Python.
- **Дальше по базовому треку.** Перенеси одну автоматизацию в код, собери RAG-бота (раздел 6), прикрути evals.
- **Целься в роль внедренца / FDE**, где ценят владение задачей, данными и качеством, а не в чистого инженера.

8. Инструменты, экосистема Anthropic и где учиться

Чтобы не тонуть в выборе, вот рабочий минимум. Бесплатного достаточно для старта.

- **Модели / API:** Claude (Anthropic), GPT (OpenAI), Gemini (Google). Возьми один, освой, потом сравнивай.
- **Векторные базы:** Chroma и Qdrant (бесплатные, для старта), Pinecone и Weaviate (когда нужен масштаб).
- **RAG-фреймворки:** LangChain и LlamaIndex. Не обязательны, но ускоряют.
- **Агентные фреймворки:** LangGraph, CrewAI, DSPy. Бери один, когда дойдёшь до агентов.
- **Evals и observability:** LangSmith, плюс встроенные инструменты провайдеров. Главное не инструмент, а привычка мерить.
- **MCP:** протокол подключения инструментов к моделям, стал стандартом. Стоит понимать.
- **Карта обучения:** roadmap.sh/ai-engineer, бесплатная и честная.

Правило: не собирай весь зоопарк сразу. Один инструмент на слой, проект важнее коллекции технологий.

Экосистема Anthropic (как пример зрелого стека)

Навыки переносимы на любого провайдера, но на Claude они удобно собраны в один стек, и Anthropic заметно вкладывается в обучение. Что стоит знать:

- **Линейка моделей по уровням.** Дешёвая и быстрая (Haiku) для простого: классификация, маршрутизация, извлечение. Рабочая лошадка (Sonnet) для примерно 80% задач. Самая мощная (Opus) для глубокого рассуждения и длинных агентных задач. Грамотный роутинг (простое на дешёвую модель, сложное на дорогую) экономит большую часть бюджета. Линейка обновляется часто, поэтому актуальные версии и цены сверяй в документации, а не по памяти.
- **Tool use (вызов инструментов).** Фундамент любого агента: модель сама решает, какой инструмент дёрнуть.

- **MCP (Model Context Protocol).** Открытый стандарт подключения данных и инструментов к моделям, своего рода «USB-C для ИИ». Вместо десяти кастомных интеграций один протокол. Стал индустриальным стандартом, и в вакансиях это самый быстрорастущий навык.
- **Prompt caching.** Кэширование повторяющейся части запроса (системный промпт, документы). Чтение из кэша стоит примерно в 10 раз дешевле обычного. Главный рычаг экономии в проде.
- **Claude Code и Claude Agent SDK.** Claude Code это агент для работы с кодом в терминале. Agent SDK это тот же агентный движок, который можно программно встроить в свой продукт (Python и TypeScript). Рекомендуемый способ строить продакшен-агентов, не изобретая цикл заново.
- **Agent Skills.** Папка с инструкциями (файл SKILL.md) плюс скрипты и шаблоны, которая расширяет возможности агента и подгружается только когда нужна. В вакансиях Anthropic это прямо указано как ожидаемый артефакт, наряду с MCP-серверами и субагентами.
- **Инженерный канон Anthropic.** Перед тем как строить агента, прочитай их статьи: «Building Effective Agents», «Writing tools for AI agents», «Effective context engineering», «Demystifying Evals», «Effective harnesses for long-running agents». Сквозная мысль: не тащи тяжёлые фреймворки, используй простые композируемые паттерны и вызывай API напрямую.

Где учиться бесплатно: Anthropic Academy

У Anthropic есть бесплатная обучающая платформа (Anthropic Academy на Skilljar) с сертификатами. Рабочий маршрут для новичка-внедренца:

1. **Claude Platform 101.** Вход для тех, кто сделал пару вызовов API или только пользовался чатом.
2. **Building with the Claude API.** Флагманский глубокий курс (84 урока, 8+ часов видео): API, продвинутый промптинг, tool use, RAG, мультимодальность.
3. **Интерактивный туториал по промпт-инжинирингу** (GitHub: anthropics/prompt-eng-interactive-tutorial). 9 глав с упражнениями.
4. **Туториал «Build a tool-using agent»** в официальных доках. Собираешь агента по шагам, сам пишешь агентный цикл.
5. **Introduction to Model Context Protocol.** Строишь MCP-сервер и клиент с нуля на Python.

6 **Claude Cookbooks** (GitHub: anthropics/claude-cookbooks). Готовые рецепты: RAG, tool use, авто-evals, prompt caching.

7 **Claude Code 101 плюс Agent Skills и Subagents.** Агентные рабочие процессы.

Плюс roadmap.sh/ai-engineer как общий бесплатный каркас и официальная документация (platform.claude.com) как справочник.

Честная оговорка для русскоязычного читателя: доступ к Anthropic Academy, консоли и API из России может быть ограничен. Проверь доступность до того, как строить на этом весь план обучения. Туториалы и Cookbook на GitHub при этом читаются откуда угодно.

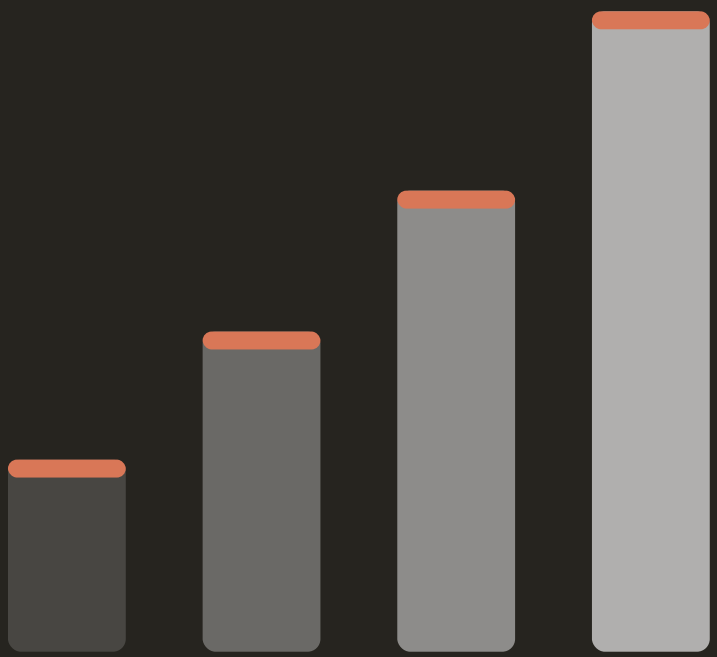
◆

ЧАСТЬ · IV

IV

Рынок и реальность

Что просят работодатели, сколько платят и где здесь хайп, а где ценность.



9. Портфолио и найм

Что показывать

- Реальные проекты на GitHub: RAG-бот, агент с инструментами, eval-пайплайн с цифрами качества.
- Разбор того, как ты мерил качество и чинил ошибки. Это сигнал зрелости.

Что спрашивают на собеседовании 2026

- Реальные кейсы RAG и агентов: что строил, какие были проблемы.
- Как выбирал модель под задачу.
- **Как мерил качество (evals).**
- Как дебажил галлюцинации.

Сертификаты: что ценится, что мусор

- **Ценится:** портфолио реальных проектов (бьёт диплом), короткие практические курсы (DeepLearning.AI), официальные материалы Anthropic и OpenAI.
- **Сомнительно:** дорогие «сертификации AI Engineer» без практики. Они дают словарь, но не навык. Рынок смотрит, что ты запустил, а не какой бейдж купил.

Что реально просят в вакансиях 2026 (по живым постингам)

Чтобы это были не общие слова, вот что прямо написано в реальных вакансиях лабораторий.

Anthropic — без PhD

Forward Deployed Engineer

$200–300K

база в год

- 3+ года · бакалавр
- промптинг · агенты · evals
- MCP · субагенты · Skills

Anthropic — без PhD

Applied AI Engineer

$280–300K

база в год

- 4+ года · бакалавр
- context engineering
- агентные архитектуры

OpenAI — без PhD

Forward Deployed Engineer

~$265K

база в год

- 5+ лет · до 50% разъездов
- Python · JavaScript
- полный стек

Anthropic, Forward Deployed Engineer. База 200-300 тысяч долларов в год. Опыт от 3 лет в технической роли с клиентом. Образование: бакалавр или эквивалентный опыт, **PhD не требуется.** Навыки прямо из постинга: продвинутый промпт-инжиниринг, разработка агентов, eval-фреймворки, деплой в масштабе. Python обязателен. Ожидаемые артефакты: MCP-серверы, субагенты, Agent Skills. Вопрос-фильтр в анкете: «Выводил ли ты агентов в продакшен?»

Anthropic, Applied AI Engineer (Beneficial Deployments). База 280-300 тысяч. Опыт от 4 лет. Снова бакалавр или эквивалент, без PhD. Навыки: промпт и context engineering, агентные архитектуры, eval-фреймворки, оптимизация затрат, бенчмарки, MCP, Agent Skills.

OpenAI, Forward Deployed Engineer. База около 265 тысяч. Опыт от 5 лет. До 50% командировок. Полный стек: Python и JavaScript.

Google Cloud, FDE по GenAI. RAG, оркестрация инструментов, мультиагентные системы (LangGraph, CrewAI, Google ADK), паттерны вроде ReAct. Любопытная деталь: только в одной старшей гугловской вакансии всплывает требование магистра или PhD, и даже там речь про прикладные навыки, а не про науку. В остальных бакалавр или эквивалент.

Сквозной список навыков из множества вакансий (что повторяется чаще всего): Python, промпт-инжиниринг, RAG плюс векторные базы, оркестрация агентов (LangChain лидирует), интеграция MCP (самый быстрорастущий навык), **дизайн evals (его называют главным сигналом реального опыта с LLM)**, observability, оптимизация затрат.

Что показывает разбор интервью (вторичные данные, но детальные). На технических этапах не гоняют по алгоритмическим задачкам, а просят собрать что-то LLM-прикладное: scorer для retrieval, распределитель бюджета токенов, оркестратор tool use. На онсайте системный дизайн крутится вокруг **eval-харнесса, а не вокруг архитектуры RAG.** И отдельный этап: симуляция разговора с клиентом, который, по слухам, отсеивает больше половины тех, кто прошёл код. Перевод на простой язык: код это входной билет, но решают evals и умение говорить с бизнесом.

Заметь, как это совпадает со всем гайдом: PhD не нужен, evals главный сигнал, harness и MCP в центре, а во внедренческих ролях бизнес-разговор весит не меньше кода.

Что на самом деле тестируют (а не пишут в вакансии)

Вакансия перечисляет базу, которую заявляют все: Python, PyTorch, SQL, «опыт с LLM», облака. На собесе проверяют другое:

- **Evals выше умения обучать модели.** Почти все провальные LLM-продукты роднит одно: не выстроили систему оценки. На любой RAG ждут eval-харнесс.
- **Рассуждение про стоимость и латентность.** Бюджеты токенов, цена за запрос, роутинг моделей. Прикидка «100K юзеров × 10 обращений × 2K токенов = 2 млрд токенов в день» отличает продакшен-мышление от прототипа.
- **Беглость в trade-off.** Не «что такое RAG?», а «когда RAG НЕ нужен?»: скорость поиска vs длина контекста, fine-tuning vs промптинг, цена GPU vs латентность.
- **Системное мышление циклами:** retrieval, генерация, обратная связь. Не пайплайны, а жизненные циклы.
- **Observability с первого дня:** логи, трейсинг, отслеживание галлюцинаций, цена на юзера.
- **Безопасность:** prompt injection, утечки данных, небезопасный вызов инструментов. Пропустил, сигнал слабого продакшен-опыта.
- **Глубина Python:** async, GIL, конкурентность vs параллелизм. На фронтир-лабах ещё и алгоритмы (у Anthropic 90-минутный CodeSignal на идеальную корректность) и реализация attention/LoRA по памяти.
- **Full-stack.** Многие AI-роли по сути full-stack: спросят про event loop, очереди, выбор БД наряду с GenAI.

По грейдам: junior, фундамент кода и базовый ML; middle, end-to-end система, RAG, продакшен-осознанность; senior, trade-off, дизайн под масштаб, разбор failure modes, оптимизация затрат; staff+, техлидерство и влияние.

Что отличает кандидата

- **Первые 5 минут решают.** Начинай с результата, а не с названий моделей.
- **Говори как строитель, не как исследователь:** «пробовали fine-tuning, но галлюцинировал, перешли на гибридный RAG».
- **Осознанность по деньгам, суперсила.** Один инженер показал before/after на 70% экономии OpenAI и получил оффер на следующий день.
- **Честность бьёт блеф.** «С LangSmith не работал, но расскажите, как у вас устроены метрики, разберусь» превращалось в оффер.

- **Не нужно быть единорогом:** берут сильных дженералистов с глубиной в 1-2 областях.

Правило 90/10: 90% успеха собеса это прошлые карьерные решения (проекты, компании, связи), и только 10%, стратегия отклика и переговоры. Вывод: вкладывайся в проекты и репутацию заранее, а не в «лайфхаки собеса».

Частые ошибки на собесе

Прыгать к fine-tuning (дефолт это промптинг плюс RAG). Считать LLM источником истины вместо заземления retrieval'ом и цитатами. Пропускать оценку и мониторинг. Сыпать названиями без trade-off. Игнорировать failure modes. Переинженерить с старта вместо «сначала рабочее». Блефовать на пробелах: «дайте подсказку» сильнее блефа.

Раунд AI system design

Отдельная категория собеса (уже у Anthropic, OpenAI, Google, Cohere). 45-60 минут, ты ведёшь разговор: уточняешь требования (~5 мин), высокоуровневая архитектура (10-15), погружение в компоненты (15-20), trade-off и узкие места (5-10). Типовые вопросы: спроектируй чат-ассистента; RAG по документам; голосовой ассистент для клиники (шум, приватность, латентность); «1 млн запросов в день, как оптимизировать стоимость». Это в основном senior-раунд: ждут рассуждения в недетерминированности и ясности, как течёт информация. Твой eval-харнесс это козырь именно здесь.

Подготовка: 8-12 недель

- **Недели 1-2:** фундамент алгоритмов (паттерны, не зубрёжка).
- **Недели 3-4:** реализация LLM/ML руками (attention, LoRA).
- **Недели 5-8:** 2-3 end-to-end проекта (RAG, агент, что-то задеплоенное), evals (Ragas, DeepEval, LLM-as-judge), продакшен (Docker, CI/CD, мониторинг).
- **Недели 9-12:** репетируй вслух (проговори последний проект за 60 секунд), отработай self-presentation на 2-3 областях, для take-home документируй решения и прикладывай Loom-видео.

5 проектов для портфолио

Конкретнее, чем «собери бота». Наниматель смотрит на README с цифрами качества, продакшен-практики в пет-проекте и оригинал, а не туториал.

1. **RAG-ассистент поддержки** по реальной базе знаний + eval на 20-50 вопросах с цифрой точности.
2. **Агент с инструментами** (tool use / MCP): календарь, поиск, БД, с разбором failure modes.
3. **Маркетплейс с AI-предзаполнением** карточек из фото или описания.
4. **Личный knowledge-бот** по своим заметкам и документам.
5. **Мультиагентная система** на реальный процесс (ревью, рассылки, планирование) с границами автономии.

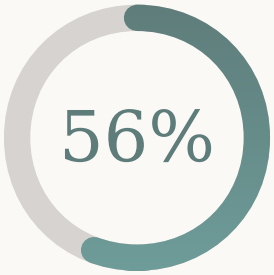
Одна реальная задача важнее пяти туториалов. Веди в проде хотя бы один проект. На собесе сможешь рассказать, как мерил качество и чинил галлюцинации, а это и есть сигнал найма.

Главный сигнал найма в 2026: портфолио важнее диплома, публичные проекты важнее резюме.

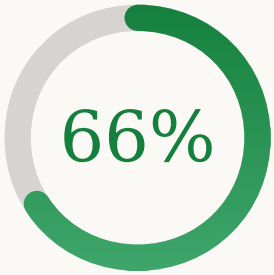
10. Рынок и деньги

Тренд подтверждён независимо несколькими крупными источниками (LinkedIn, PwC, WEF).

новых ИИ-рабочих мест за 2 года



премия за ИИ-навык к зарплате



быстрее меняются навыки

- **1,3 млн новых ИИ-связанных рабочих мест** добавлено в мировую экономику за два года (к январю 2026, данные LinkedIn). Среди них роли AI Engineer, Forward-Deployed Engineer и другие.
- **AI Engineer одна из самых быстрорастущих профессий на LinkedIn** за последние три года.
- **Премия за ИИ-навыки около 56%** к зарплате на той же должности (по данным PwC AI Jobs Barometer), и она выросла за год.
- **Навыки в ИИ-профессиях меняются примерно на 66% быстрее** обычных. Перевод: постоянное переобучение это часть профессии, а не разовое усилие.
- **WEF Future of Jobs 2025** относит специалистов по ИИ и ML к самым быстрорастущим ролям, а 86% работодателей ждут, что ИИ изменит их бизнес к 2030.

Про зарплаты честно: в США AI Engineer и FDE стоят примерно 130-280 тысяч долларов в год плюс опционы (а у лабораторий вроде Anthropic и OpenAI база FDE доходит до 265-300 тысяч). Для русскоязычного рынка эти цифры не ориентир, у нас другая база. Но важна не абсолютная сумма, а два факта, которые подтверждены независимо: спрос на навык растёт, и за навык платят премию. Это работает и на локальном рынке, и на удалёнке.

Что это значит для русскоязычного читателя

Несколько честных мыслей, без приукрашивания.

- **Профессия по своей природе удалённая и глобальная.** Работа идёт онлайн, документация и сообщество в основном англоязычные, портфолио на GitHub география-нейтрально. Это значит, что твой потолок не ограничен локальным рынком: при хорошем английском реально целиться в удалённые и зарубежные команды, не переезжая.
- **Английский это рычаг, а не опция.** Первоисточники, доки, вакансии, собеседования в этой сфере на английском. Подтянутый технический английский напрямую расширяет, куда ты можешь устроиться.
- **Доступ к инструментам проверяй заранее.** Часть платформ, консолей, курсов и платёжных систем может быть недоступна из России напрямую. Это решаемо (альтернативные провайдеры, доступ через другие юрисдикции), но закладывай это в план обучения, а не упирайся в стену в середине пути. Открытые материалы на GitHub при этом доступны откуда угодно.
- **Локальный рынок тоже растёт.** Российские компании внедряют ИИ, спрос на внедренцев и консультантов есть, хотя названия ролей и зрелость отстают от мировых. Здесь часто выгоднее целиться не в чистого «инженера», а во внедренца или консультанта, который приносит бизнесу измеримый результат. Это та же модель, что и FDE, только на локальном рынке.
- **Не ориентируйся на абсолютные суммы из США.** Ориентируйся на тренд: навык дефицитный, премия за него реальна, и она проявляется в любой валюте.

Реальные цифры по РФ (данные hh.ru, январь 2026):

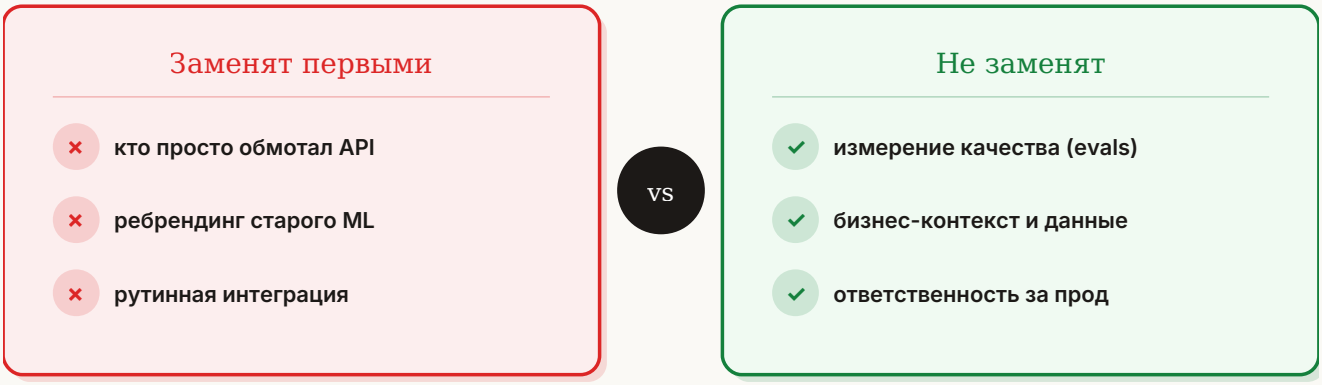
- Медиана **AI-инженера ≈ 220 000 ₽/мес**.
- **ML-инженер: 184-345K** (middle около 160K, senior 300K+).
- А вот **промпт-инженер обычно ≤100K** (60-90K). Это и есть честный сигнал: «промпт-инженер без кода» и «AI-инженер», разные лиги по деньгам и потолку. Кто обещает большие деньги за «нейросети без кода», тот лукавит.
- Число ИИ-вакансий за 2025 выросло **вдвое** к 2024. Внедряют Сбер (GigaChat), Яндекс, Т-Банк, финтех, e-com.

Стартовать можно на отечественном стеке без обходов: GigaChat / GigaCode, YandexGPT, Yandex Cloud доступны легально и их учат на OTUS и Практикуме. Карьеру реально начать полностью на доступных в РФ инструментах, а Claude/OpenAI подключить позже.

Оговорка: медианы это hh.ru/Ведомости (янв 2026, Tier 1); вилки по грейдам зависят от источника; данные по прямому удалённому найму из РФ тонкие, поэтому дано осторожно (реалистичнее РФ-компании, аутстафф или релокация).

11. Скептический угол: где хайп, а где ценность

Без этого раздела гайд был бы рекламой. А честность это то, за что подписываются.



- **«AI Engineer» как ребрендинг.** Часть вакансий это переупакованный старый инженер на волне моды. Рекрутеры путают роли: зовут на «AI Engineer», а по факту гоняют по классическому ML. Тайтлы в 2026 превратились в кашу.
- **Разрыв между описанием вакансии и реальностью.** По историям практиков работа часто оказывается на 80% промпт-инжинирингом, склейкой и мониторингом, хотя в вакансии красиво написали про RAG и эмбеддинги.
- **Риск, что ИИ-инструменты съедят часть самой роли.** Pragmatic Engineer (опрос 900+ инженеров, 2026) фиксирует: роль смещается к оркестрации. Инженеры всё больше управляют ИИ-агентами, а не пишут всё руками. Это трансформация, а не исчезновение профессии.

Что останется ценным через 3-5 лет. Не «написать интеграцию», это всё больше делают сами агенты. А вот это не коммодитизируется:

- умение измерять качество (evals),

- понимание бизнес-контекста,
- владение данными клиента,
- ответственность за продакшен.

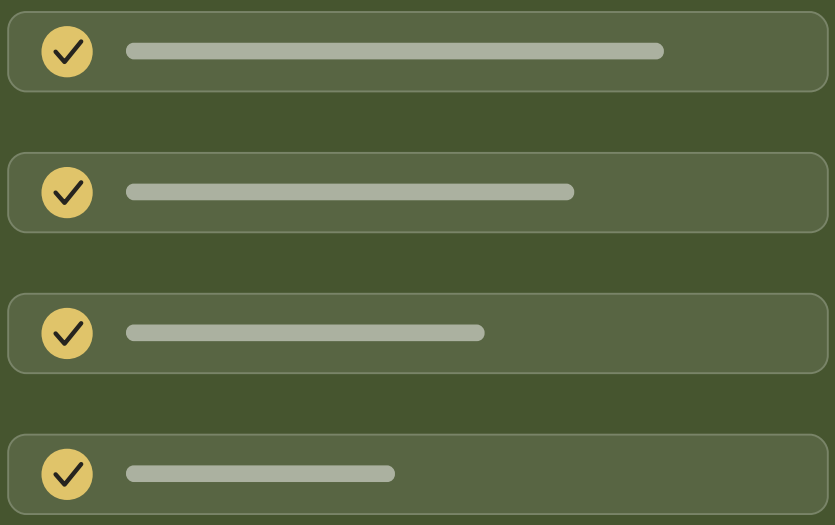
Именно поэтому FDE это про владение системой, а не про строчки кода. Качество и доверие не автоматизируются.

Главный честный тезис. Хайп сидит в слове «инженер». Ценность сидит под капотом: умение довести ИИ до работающего в проде и измеримого результата. Тех, кто просто обмотал API, заменят первыми. Тех, кто умеет держать качество, не заменят.

ЧАСТЬ · V

Справочник

Короткие ответы, чек-лист готовности и приложения.



12. FAQ

Нужен ли мне PhD? Нет. Для внедрения ИИ нужны инженеры, а не исследователи. Это прямая цитата из канонического эссе о профессии.

Нужна ли тяжёлая математика? Для второй плоскости (внедрение) нет. Линейную алгебру и статистику полезно понимать на уровне интуиции, но это не вход.

Хватит ли no-code (n8n, Zapier, Make)? Как точка входа да, чтобы понять логику. Как профессия нет: на no-code не построить качество и evals, а именно за них платят. Это потолок, а не финиш.

За сколько реально войти? Разработчику 3-6 месяцев фокуса. Аналитику чуть дольше. Бизнес-человеку с нуля от 6 до 12 месяцев, либо целиться во внедренца, где код не главный актив.

Нужно ли дообучать модели? Почти всегда нет. 57% команд не делают этого вовсе. Базовая модель плюс промптинг и RAG закрывают большинство задач.

Что важнее, диплом или GitHub? GitHub. Портфолио реальных проектов бьёт диплом и сертификаты на найме 2026.

С какого проекта начать? RAG-бот по реальным документам плюс evals на 20-50 вопросах. Это и учебный проект, и портфолио одновременно.

Что важнее, модель или то, как я её обвязал? Чаще обвязка (harness). Модель у всех одна и та же, ты её арендуешь. Преимущество это система вокруг неё: контекст, инструменты, проверки и измерение качества.

Нужно ли постоянно читать про новые модели? Да, это часть работы. Но не гонись за хайпом: проверяй новое на своих собственных evals, а не по постам в ленте.

13. Чек-лист: ты готов претендовать на роль

Отметь, что уже умеешь:

- [ ] Понимаю разницу между «обучать модель» и «использовать через API», и знаю, что я во второй плоскости.
- [ ] Могу на базовом Python дёрнуть LLM API и обработать ответ.
- [ ] Умею писать системные промпты и получать структурированный вывод.
- [ ] Собрал хотя бы один RAG-бот по реальным документам.
- [ ] Сделал eval-набор из 20-50 задач и измерил качество своего бота.
- [ ] Умею читать трейсы и находить, где система ошибается.
- [ ] Думаю о системе как об обвязке вокруг модели (harness), а не как о «магическом промпте».
- [ ] Отношусь к настройкам (модель, чанк, промпт) как к экспериментам: меняю по одной и измеряю.
- [ ] Регулярно читаю первоисточники и проверяю новые модели на своих evals.
- [ ] Добавил боту хотя бы один инструмент (tool use).
- [ ] Выложил проект на GitHub с понятным README и цифрами качества.
- [ ] Могу на собеседовании рассказать, как мерил качество и чинил галлюцинации.

Если отмечено 6 и больше пунктов, ты уже сильнее половины тех, кто называет себя ИИ-инженерами. Иди откликаться.

Приложение А. Каркас первого проекта (эскиз)

Чтобы «собери RAG-бота плюс evals» из раздела 6 стало осязаемым, вот скелет потока в псевдокоде. Это не готовый к проду код, а карта: видно, из

каких шагов состоит работа. Конкретные библиотеки подставляются по вкусу.

```
# 1. Готовим знания: режем документы на куски и считаем эмбеддинги
chunks = split_into_chunks(load_documents("docs/"))      # чанкинг
index = VectorStore()
for chunk in chunks:
    index.add(embed(chunk), chunk)                        # эмбеддинг -> векторная база

# 2. Отвечаем на вопрос по найденным кускам. Это и есть RAG
def answer(question):
    found = index.search(embed(question), top_k=5)        # поиск похожего
    context = "\n\n".join(found)
    prompt = f"Ответь строго по контексту.\nКонтекст:\n{context}\n\nВопрос: {question}"
    return llm(prompt)                                     # вызов модели через API

# 3. Меряем качество: прогон по эталонному набору. Это и есть eval
eval_set = load_eval_set("eval.jsonl")   # 20-50 пар: вопрос + признак верного ответа
passed = 0
for case in eval_set:
    out = answer(case["question"])
    if grade(out, case["expected"]):     # грейдер: код, модель-судья или человек
        passed += 1
print(f"Качество: {passed}/{len(eval_set)}")
```

Когда этот скелет у тебя работает на реальных документах и реальном eval-наборе, ты уже умеешь то, что спрашивают на собеседовании. Дальше наращиваешь: reranking в шаге 2, наблюдаемость вокруг шага 1, инструменты для превращения бота в агента.

Приложение Б. Мини-глоссарий

- **LLM**: большая языковая модель, на которой строят продукты.
- **Токен**: кусочек текста, в токенах считают объём и цену запроса.
- **Промпт**: то, что отправляется модели на вход.
- **Системный промпт**: инструкция о роли и поведении модели.
- **Контекстное окно**: максимум текста, который модель держит в работе за раз.
- **Context engineering**: управление тем, что попадает в контекстное окно.
- **Tool use**: вызов инструментов: модель сама решает, какую функцию дёрнуть.
- **Эмбеддинг**: числовое представление смысла текста для поиска по близости.

- **Векторная база**: хранилище эмбеддингов с быстрым поиском похожего.
- **RAG**: паттерн, когда модель отвечает по найденным в твоих данных кускам.
- **Агент**: модель с инструментами и правом самой решать, что вызвать.
- **Harness**: обвязка вокруг модели, превращающая её в работающую систему.
- **MCP**: открытый стандарт подключения данных и инструментов к моделям.
- **Eval**: измеримая проверка качества ответов системы.
- **Трейс**: полная запись одного прогона: ввод, вызовы инструментов, вывод.
- **Observability**: возможность видеть, что система делает внутри в проде.
- **Prompt caching**: переиспользование повторяющейся части запроса ради цены и скорости.
- **Fine-tuning**: дообучение модели, прикладным командам нужно редко.
- **Prompt injection**: атака через вредную инструкцию, спрятанную во входных данных.
- **FDE**: Forward-Deployed Engineer, внедренец, который строит систему внутри клиента.

Источники

ПЕРВИЧНЫЕ · КАНОНИЧНЫЕ

1. **swyx · The Rise of the AI Engineer** · latent.space
2. **LangChain · State of Agent Engineering 2025** · langchain.com
3. **Anthropic · Demystifying Evals for AI Agents** · anthropic.com
4. **Anthropic · Effective context engineering for AI agents** · anthropic.com
5. **Anthropic · Effective harnesses for long-running agents** · anthropic.com
6. **Anthropic · Building Effective Agents** · anthropic.com
7. **Anthropic Academy · бесплатные курсы** · anthropic.skilljar.com

8. **Интерактивный туториал по промпт-инжинирингу** · github.com
9. **Claude Cookbook · готовые рецепты** · github.com
10. **Model Context Protocol · документация** · modelcontextprotocol.io
11. **WEF · AI added 1.3M jobs (данные LinkedIn)** · weforum.org
12. **PwC · AI Jobs Barometer (премия за навык)** · pwc.com
13. **Pragmatic Engineer · Impact of AI on Software Engineers 2026** · pragmaticengineer.com
14. **Anthropic Careers · Forward Deployed / Applied AI** · greenhouse.io
15. **OpenAI Careers · Forward Deployed Engineer** · openai.com

ВТОРИЧНЫЕ · ПРАКТИЧЕСКИЕ

16. **roadmap.sh/ai-engineer · карта навыков** · roadmap.sh
17. **O'Reilly · The AI Agents Stack, 2026 edition** · oreilly.com
18. **marktechpost · What is a Forward Deployed Engineer (2026)** · marktechpost.com

Заметка о достоверности

Числа, помеченные как подтверждённые (доля команд с агентами в проде, барьер качества, доля без fine-tuning, премия за навык, число новых рабочих мест), сверены с первоисточником. Опрос LangChain отражает аудиторию энтузиастов, поэтому его абсолютные доли стоит читать как «потолок отрасли», а не среднюю по экономике; тренд при этом достоверен. Зарплатные вилки относятся к рынку США и приведены как ориентир тренда, а не локальный бенчмарк. Опровергнутых источниками утверждений в гайде нет.

ПОД КАПОТОМ · 2026

Спасибо, что дочитал.

Этот гайд это часть канала «Under the Hood»: разборы ИИ-индустрии без хайпа, с источниками и цифрами. Если зашло, забирай продолжение.

Under the Hood

@gorilla_under_hood

Автор	Илья Утов
Telegram	**@wh_zt**
LinkedIn	**linkedin.com/in/ilyautov**