

How GPT, Claude, and Gemini are actually trained and served - Reiner Pope

@DwarkeshPatel · 2026-04-29 · <https://www.youtube.com/watch?v=xmkSf5IS-zw>



Райнер Поуп (CEO стартапа MatX, ранее разрабатывавший архитектуру TPU в Google) и Дваркеш Пател разбирают на доске фундаментальную экономику и математику инференса и обучения LLM.

Главная мысль выпуска: понимание того, как модели работают на уровне физического железа (пропускная способность памяти, FLOPs, батчинг, топология кластеров), объясняет всё — от архитектуры современных нейросетей до ценообразования по API и причин замедления или ускорения прогресса в ИИ. Все рассуждения строятся на строгих физических лимитах чипов и пропускной способности сетей.

Базовая математика инференса: Roofline-модель

Анализ производительности через Roofline-модель 02:10: Время генерации одного токена ограничено либо скоростью вычислений (compute), либо скоростью работы с памятью (memory bandwidth). Итоговое время выполнения шага инференса — это всегда максимум из этих двух значений: $t = \max(t_{\text{compute}}, t_{\text{mem}})$. Для анализа берется кластер уровня стойки NVIDIA Blackwell NVL72 (72 GPU).

Формула времени вычислений (t_{compute}) 03:19: Вычисления состоят из умножения матриц весов на контекст. Время вычисляется как $t_{\text{compute}} = B \cdot N_{\text{active}} / \text{FLOPS}$, где B — размер батча (количество параллельно обслуживаемых пользователей), N_{active} — количество активных параметров модели при генерации одного токена (например, у DeepSeek V3 это 37 млрд из 700 млрд

общих), а FLOPS — вычислительная мощность чипа. Временем вычисления самого механизма внимания (attention) на этом этапе можно пренебречь, так как оно мало по сравнению с умножением матриц.

Фактчек: DeepSeek V3: 37 млрд активных из ~700 млрд, активируются 32 эксперта из 256, множитель разреженности 8 — □: Параметры верны (37B из 671B), но роутер выбирает 8 из 256 экспертов (+1 общий), и $N_{total}/N_{active} \approx 18$, а не 8. Оценка «оптимальный батч ≈ 2400 » из этого не следует (при множителе 18 выходит ~ 5400 , порядок тот же); GPT дополнительно указывает, что при батче 1 читать веса невыбранных экспертов не нужно, так что допущение «читаем все веса» уместно только на большом батче.

Формула времени работы с памятью (t_{mem}) 05:00: Память тратится на две задачи: загрузку весов модели и загрузку KV-кэша. Формула выглядит как $t_{mem} = (N_{total} + B \cdot L_{context} \cdot Bytes_per_token) / Mem_BW$. Здесь N_{total} — общий размер всех параметров модели (их нужно прочитать целиком), $L_{context}$ — длина контекста, $Bytes_per_token$ — вес одного токена в байтах, а Mem_BW — пропускная способность памяти (байт/сек). Загрузка KV-кэша линейно зависит от размера батча, так как для каждого пользователя нужно подтягивать его уникальную историю контекста.

Зависимость задержки (latency) от размера батча 08:19: Если построить график задержки от размера батча, видно жесткое нижнее ограничение. Даже при батче равном 1 (один пользователь), задержка не может быть меньше времени, необходимого для считывания всех весов модели (N_{total}) из памяти чипа. По мере роста батча время вычислений и время загрузки KV-кэша растут линейно, и в какой-то момент график задержки начинает идти вверх.

Зависимость стоимости от размера батча 13:08: Стоимость инференса на один токен — это время аренды GPU (t), деленное на количество токенов в батче (B). При батче равном 1 стоимость стремится к бесконечности, так как огромные затраты на загрузку весов (N_{total}) не амортизируются. По мере роста батча стоимость загрузки весов на одного пользователя падает по параболе, пока график не упирается в константу — базовую стоимость самих вычислений ($t_{compute} / B$) и загрузки уникального KV-кэша. Именно поэтому провайдеры не могут сделать сверхдешевый «медленный режим» (Slow Mode) — стоимость упирается в физический лимит вычислений и памяти на конкретного пользователя.

Фактчек: При батче 1 стоимость стремится к бесконечности, падает «по параболе», дешёвый Slow Mode невозможен — □: Стоимость при $B=1$ конечна (= полное чтение весов), а зависимость $1/B$ — гипербола. Батчинг амортизирует чтение весов до физического пола (compute на токен + уникальный KV-кэш), но отложенная обработка вполне работает — Batch API со скидкой 50% у провайдеров и есть тот «медленный режим», который экономика допускает.

Оптимальный размер батча и влияние разреженности (Sparsity)

Точка баланса между памятью и вычислениями 16:00: Оптимальный размер батча достигается там, где время вычислений равно времени загрузки памяти ($t_{\text{compute}} = t_{\text{mem}}$). Если отбросить KV-кэш для упрощения, уравнение сводится к аппаратно-программной пропорции: $B = (\text{FLOPS} / \text{Mem_BW}) \cdot (N_{\text{total}} / N_{\text{active}})$.

Аппаратная константа и реальный размер батча 18:51: Отношение FLOPS к пропускной способности памяти (Mem_BW) — это аппаратная константа, которая остается удивительно стабильной от поколения к поколению (A100, H100, B100) и составляет около 300. Отношение N_{total} к N_{active} — это параметр разреженности (sparsity) модели. Для DeepSeek, где активируются 32 эксперта из 256, этот множитель равен 8. Итоговый оптимальный батч: $B \approx 300 \times 8 = 2400$ уникальных последовательностей. На практике берут чуть больше (x2-x3), чтобы компенсировать неидеальную утилизацию железа.

Фактчек: Отношение FLOPS/Mem_BW — аппаратная константа ≈ 300 , стабильная от A100 до B100 — Δ : Для BF16 у H100 (~ 295) и B200 (~ 280) сходится, но у A100 ≈ 156 , а в FP8 у H100 уже ~ 590 . «Около 300» верно для актуального поколения в BF16, а не универсальная константа; сама roofline-модель — идеализированная нижняя граница без учёта коммуникаций и синхронизации.

[20:57-21:58] рекламная интеграция Jane Street.

Механика запуска батчей («расписание поездов») 21:58: Инференс работает как поезд, отправляющийся по строгому расписанию. Время эвакуации и заполнения всей HBM-памяти (Capacity / Bandwidth) на современных чипах составляет около 15-20 миллисекунд (например, 288 ГБ / 20 ТБ/с на поколении Rubin). Каждые 20 мс запускается новый цикл (forward pass). Если запрос пользователя пришел сразу после старта, он ждет 20 мс до следующего «поезда» и еще 20 мс едет. Худшая задержка в очереди — 40 мс.

Глобальный трафик и конкуренция 26:13: Батч в 2000 пользователей при цикле в 20 мс (около 64 итераций в секунду) означает пропускную способность системы около 128 000 токенов в секунду. Для сравнения, глобальный трафик Gemini оценивается в сотни миллионов токенов в секунду. Чтобы экономика инференса сходилась и была конкурентной, провайдер должен обслуживать хотя бы 1/1000 от трафика Gemini.

Экономика Mixture of Experts (MoE) 27:50: Увеличение разреженности (sparsity) позволяет радикально снизить объем вычислений (t_{compute}). Однако, чтобы сохранить качество модели, приходится экспоненциально увеличивать общее число параметров (N_{total}). Например, модель с 64 экспертами (370 млн активных параметров) дает качество плотной модели на 1.3 млрд параметров, но требует огромного объема памяти. Экономически это выгодно только в том

случае, если у провайдера есть гигантский поток пользователей (огромный батч), чтобы амортизировать загрузку этих раздутых весов.

Топология кластеров и виды параллелизма

Распределение MoE-слоя по GPU (Expert parallelism) 32:08: В слое MoE токен проходит через роутер и отправляется к нужным экспертам. Лучший способ разложить это на железе — Expert parallelism, когда разные эксперты лежат на разных GPU. Для модели с 256 экспертами на стойке из 64 GPU на каждый чип ложится по 4 эксперта. Это порождает паттерн трафика all-to-all: любой GPU должен иметь возможность мгновенно отправить данные любому другому GPU.

Ограничения физического размера стойки (rack) 36:52: Внутри стойки (scale-up network, например NVLink) все GPU соединены коммутаторами напрямую (в два хопа) с колоссальной пропускной способностью. Выход за пределы стойки (scale-out network) работает примерно в 8 раз медленнее. Сделать стойку на 1000 GPU невозможно из-за физических ограничений: плотности кабелей, радиуса их изгиба, веса металла, энергообеспечения и охлаждения. Увеличение числа GPU в стойке требует квадратичного роста числа кабелей.

Фактчек: Ограничения стойки: scale-out в ~8 раз медленнее, число кабелей растёт квадратично — [1]: Общепринятые инженерные оценки; расхождений у рецензентов нет.

[44:24-44:58] рекламная интеграция Gemma 4 / Cursor.

Pipeline-параллелизм между стойками 44:58: Если модель не влезает в одну стойку, используется Pipeline parallelism — разрезание модели по слоям. Первая стойка считает первые слои, затем передает данные во вторую стойку по медленной сети (scale-out). Это не становится узким местом, потому что между стойками передается только размер батча (B), тогда как внутри стойки при all-to-all коммуникации передается батч, умноженный на число активированных экспертов и число слоев ($B \times N_{\text{experts}} \times N_{\text{layers}}$).

Проблема «пузырей» (pipeline bubbles) при обучении 54:33: При пайплайнинге во время обучения возникает проблема простоя. Чтобы сделать обратное распространение ошибки (backward pass), нужно дождаться прохождения всего батча через все стойки. Это создает огромные «пузыри» (idle time), когда часть стоек ничего не делает. Для решения используются микро-батчи и сложные схемы вроде 1F1B (one-forward-one-backward), переплетающие прямые и обратные проходы.

Бесполезность пайплайнинга для ускорения инференса 1:00:58: В инференсе пайплайнинг не улучшает ни задержку, ни пропускную способность.

Он не экономит время вычислений или памяти, а просто переносит часть работы на другую стойку. Единственный его плюс — снижение требований к объему памяти (capacity) на одной стойке, так как веса размазываются по нескольким.

Проблема памяти и масштабирование моделей

[1:02:36-1:03:36] рекламная интеграция Cursor.

Потребление емкости памяти (Memory Capacity) 1:03:36: Требования к памяти состоят из весов (N_{total}) и KV-кэша ($B \cdot L_{context} \cdot Bytes_per_token$). Пайплайнинг (P) делит веса на количество стоек, но не спасает от KV-кэша. Чтобы стойки не простаивали, при пайплайнинге приходится увеличивать количество микро-батчей (глобальный батч растет пропорционально числу стадий P). В итоге множитель P сокращается, и объем памяти под KV-кэш на каждом GPU остается неизменным. Поэтому для инференса пайплайнинг почти не используют — выгоднее держать всю модель в пределах одной стойки (scale-up domain).

Истинная причина скачка в размерах моделей 1:13:30: Долгое время размеры моделей стагнировали, пока не появились стойки с огромным scale-up доменом (переход от Hopper с 8 GPU к Blackwell с 72 GPU). Главная польза большой стойки — не суммарный объем памяти (памяти хватало и раньше за счет пайплайнинга), а суммарная пропускная способность памяти (Memory Bandwidth). 72 GPU, читающие веса параллельно, радикально снижают t_{mem} , позволяя обслуживать большие батчи и длинные контексты с приемлемой задержкой.

Сетевые задержки при межузловом взаимодействии 1:15:06: Прыжок между стойками добавляет физическую задержку (около нескольких миллисекунд на хоп). При генерации токена эти задержки суммируются. 4 стадии пайплайна добавляют ~10 мс на каждый токен, что критично замедляет авторегрессионный вывод.

Экономика обучения: Chinchilla, RL и переобучение

Балансировка стоимости претрейна, RL и инференса 1:18:53: Оптимизация общих затрат на ИИ-модель сводится к уравниванию стоимости разных этапов. Эвристическое правило гласит: общая стоимость минимизируется там, где затраты на претрейн, RL-генерацию и инференс примерно равны (по 33% бюджета на каждый этап).

Фактчек: Скачок размеров моделей вызван переходом к NVL72 — □: Логика про суммарную пропускную способность памяти корректна для NVIDIA-железа, но TPU-поды Google с большим scale-up доменом существовали задолго до Blackwell — тезис не объясняет всю индустрию.

Вычислительная стоимость разных этапов 1:22:13: Стоимость претрейна на один параметр — 6 FLOPS (прямой + обратный проход). Стоимость инференса — 2 FLOPS (только прямой). Стоимость RL (где генерируются траектории, но градиентный шаг делается не всегда) колеблется от 2 до 6 FLOPS. С учетом неэффективности утилизации железа при генерации в RL, для равенства затрат объем токенов в претрейне должен быть примерно равен объему токенов в RL и объему токенов, которые модель сгенерирует за весь свой жизненный цикл на проде.

Оценка объема данных для претрейна через трафик инференса 1:28:08: Если модель уровня GPT-5 будет генерировать 50 млн токенов в секунду и проработает до устаревания 2 месяца, она выдаст около 200 триллионов токенов. Значит, в ее претрейн тоже должно быть вложено около 200 триллионов токенов. По законам масштабирования Chinchilla, для модели на 100 млрд активных параметров оптимальный объем данных — 2 трлн токенов. Вывод: современные frontier-модели переобучаются (over-trained) примерно в 100 раз сверх оптимума Chinchilla, чтобы сделать инференс экономически выгодным.

Фактчек: Chinchilla: 100 млрд параметров → 2 трлн токенов; frontier-модели переобучены в ~100× — 📄: Правило ~20 токенов на параметр — корректное чтение Chinchilla. «100×» — правдоподобная, но непроверяемая оценка: у Llama 3 405B ~2× сверх оптимума, у Llama 3 8B ~100×, для закрытых моделей данных нет, а для MoE понятие «активных параметров» в Chinchilla применяется нестрогое. Overtraining здесь не означает overfitting.

Реверс-инжиниринг цен на API (Gemini, Claude, GPT)

Скачок цены при длинном контексте (на примере Gemini > 200k) 1:33:03: В API Gemini цена вырастает на 50% при превышении контекста в 200 000 токенов. Это точка перегиба на графике Roofline-модели. До 200k стоимость ограничена вычислениями (t_{compute}), которые не зависят от длины контекста. После 200k доминирующим фактором становится время загрузки KV-кэша из памяти (t_{mem}), которое растет линейно вместе с контекстом.

Фактчек: Цена Gemini растёт на 50% при контексте >200k из-за перегиба compute→memory-bound — ⚠: У Gemini 2.5 Pro выход дорожает в 1.5× (10→15), но вход — в 2× (1.25→2.50). Объяснение через KV-кэш разумно, но это гипотеза: коммерческий тариф — не измерение себестоимости и не доказательство архитектуры.

Оценка размера KV-кэша на токен 1:38:20: Зная точку перегиба (200k токенов), можно вычислить размер KV-кэша. Приравняв t_{compute} и t_{mem} для KV-кэша, получаем, что один токен весит около 1.6 КБ. Это достигается за счет плотного внимания с переиспользованием контекста (MQA/GQA), где на слой приходится, например, 8 KV-голов с размерностью 128, либо за счет алгоритмов разреженного внимания (sparse attention).

Фактчек: Размер KV-кэша ≈ 1.6 КБ на токен при 8 KV-головах $\times$ 128 на слой — $\square$: 8 голов $\times$ 128 $\times$ 2 (K и V) $\times$ 2 байта = 4096 байт на слой, а не 1.6 КБ на всю модель; без специальных схем сжатия (MLA, sparse attention) итог надо умножать на число слоёв. Расчёт из пересказа не сходится по размерности.

Разница в цене между Prefill (ввод) и Decode (вывод) 1:41:42: В API обработка входящих токенов (prefill) стоит в 3-5 раз дешевле, чем генерация (decode). При prefill модель обрабатывает весь промпт параллельно за один проход. Загрузка весов из памяти происходит один раз и амортизируется на тысячи токенов промпта. Процесс упирается исключительно в дешёвые вычисления (compute-bound). При decode модель вынуждена заново грузить все веса ради генерации каждого единственного токена, что делает процесс жестко ограниченным пропускной способностью памяти (memory-bandwidth bound).

Фактчек: Prefill в 3-5 раз дешевле decode — $\square$: По порядку величины верно, реальный разброс шире — 4-8 $\times$ (Claude Sonnet 5 $\times$, GPT-4o 4 $\times$, Gemini 2.5 Pro 8 $\times$). Объяснение через memory-bound decode корректно.

Ценообразование кэширования контекста (Cache hits) 1:48:29: Провайдеры предлагают скидки (до 10 $\times$) за попадание в кэш контекста. Удержание KV-кэша в памяти стоит денег, так как занимает емкость (Capacity), не давая обслуживать других пользователей. Альтернатива — удалить кэш и пересчитать его заново (Rematerialization), потратив GPU-время (вычисления).

Фактчек: Скидка до 10 $\times$ за попадание в кэш; часовой тариф означает сброс на HDD — $\square$ по скидке, $\square/\square$ по HDD (расхождение рецензентов): Тарифы Anthropic 5 мин / 1 час и чтение из кэша за 0.1 $\times$ подтверждаются. Gemini считает вывод про HDD подтверждённым практикой многоуровневого хранения; Claude — красивой, но не подтверждённой гипотезой; GPT — некорректным: TTL определяется вероятностью повторного обращения и стоимостью пересчёта, а не drain time носителя. Итог: тип носителя из прайса не выводится.

Определение типа памяти по времени хранения 1:59:32: В API есть тарифы на хранение кэша на 5 минут и на 1 час. Оптимальное время хранения в конкретном типе памяти равно времени полного чтения/записи этого объема памяти (drain time). Для HBM (видеопамяти) это 20 миллисекунд. Для DDR (оперативной памяти хоста) — секунды. Для Flash-накопителей (SSD) — около минуты. Для жестких дисков (HDD) — около часа. Наличие часового тарифа прямо указывает на то, что провайдеры сбрасывают «холодный» контекст на дешёвые медленные диски.

Фактчек: Оптимум: претренин, RL и инференс по $\sim 33\%$ бюджета — $\triangle$: Принцип «уравнивай предельную отдачу статей» разумен, но автоматическое «по трети» не установлено — оптимум зависит от прироста качества на каждом этапе и от будущего спроса.

Нейросети и криптография: конвергентная эволюция

Сходство задач криптографии и глубокого обучения 2:03:48: Обе сферы занимаются перемешиванием информации. Криптография берет структурированные данные и превращает их в белый шум (лавинный эффект, где малейшее изменение входа меняет весь выход). Нейросети делают обратное — берут шум (пиксели, текст) и извлекают структуру. При этом случайная инициализация нейросети работает как неплохой шифр.

Сеть Фейстеля (Feistel network) в нейросетях 2:08:34: В криптографии давно используется конструкция Фейстеля, позволяющая сделать любую неинвертируемую функцию обратимой за счет сохранения части входа ($x, y \rightarrow x+f(y), y$). В 2017 году эту идею перенесли в машинное обучение в виде архитектуры Reversible Networks (RevNets).

Фактчек: RevNets (2017) по мотивам сети Фейстеля позволяют не хранить активации —  Reversible Residual Networks делают слои обратимыми, входы пересчитываются на backward pass; размен памяти на вычисления подтверждается.

Reversible Networks (RevNets) для экономии памяти 2:10:48: При стандартном обучении трансформера все промежуточные активации слоев (forward pass) должны сохраняться в HBM-памяти, чтобы позже использовать их для вычисления градиентов (backward pass). Это сжирает гигантский объем памяти. Использование обратимых слоев (RevNets) позволяет вообще не хранить активации. Во время обратного прохода сеть буквально вычисляется задом наперед, восстанавливая состояния на лету. Это разменивает дополнительные вычисления на радикальную экономию памяти — подход, прямо противоположный KV-кэшированию (где память тратится ради экономии вычислений).

ЛИЧНО ДЛЯ ТЕБЯ

Управляй счётом за API через физику: генерация в 4–8х дороже ввода, контекст свыше ~200k дорожает скачком, кэш префикса даёт до 10х. Практически: стабильные инструкции и документы — в начало промпта и не менять, задача — в конец; просить короткие ответы и меньше «думания»; фоновые задачи гонять через Batch API (–50%). Проверь фактические cached tokens в биллинге, а не полагайся на теорию.

Дешёвого до нуля инференса не будет — калибруй ожидания при любых расчётах, где AI — переменная себестоимость: цена токена упирается в compute на пользователя и его уникальный KV-кэш. Удешевление идёт через разреженность, квантизацию и рост батча, а не через «подождём, станет бесплатно».

Сравнивай AI-инструменты по трём метрикам, а не по «токенам в секунду»: время до первого токена (интерактив), скорость потока и стоимость решённой задачи. Для правки кода в диалоге критична отзывчивость, для фонового ревью — цена и качество результата.

Не читай прайс-лист как рентген модели: размер модели, KV-кэша или тип накопителя из тарифов не выводится — половина «реверс-инжиниринга» в выпуске не сошлась по цифрам. Для решений о деньгах собственные замеры надёжнее красивых выкладок на доске.

КОНСЕНСУС РЕЦЕНЗЕНТОВ

Все трое сошлись: концептуальный каркас (roofline, compute- vs memory-bound, prefill/decode, Chinchilla, RevNets) верен и полезен, а конкретные числа в пересказе местами перепутаны (DeepSeek 8 vs 32 экспертов, размер KV-кэша, «константа 300»). Разошлись в строгости: Gemini подтверждает почти всё, включая HDD и «100× переобучение»; Claude и GPT считают выводы из прайс-листов (HDD, 200k-перегиб, 33/33/33) гипотезами, а GPT дополнительно отвергает «бесконечную стоимость при $B=1$ » и невозможность медленного режима.