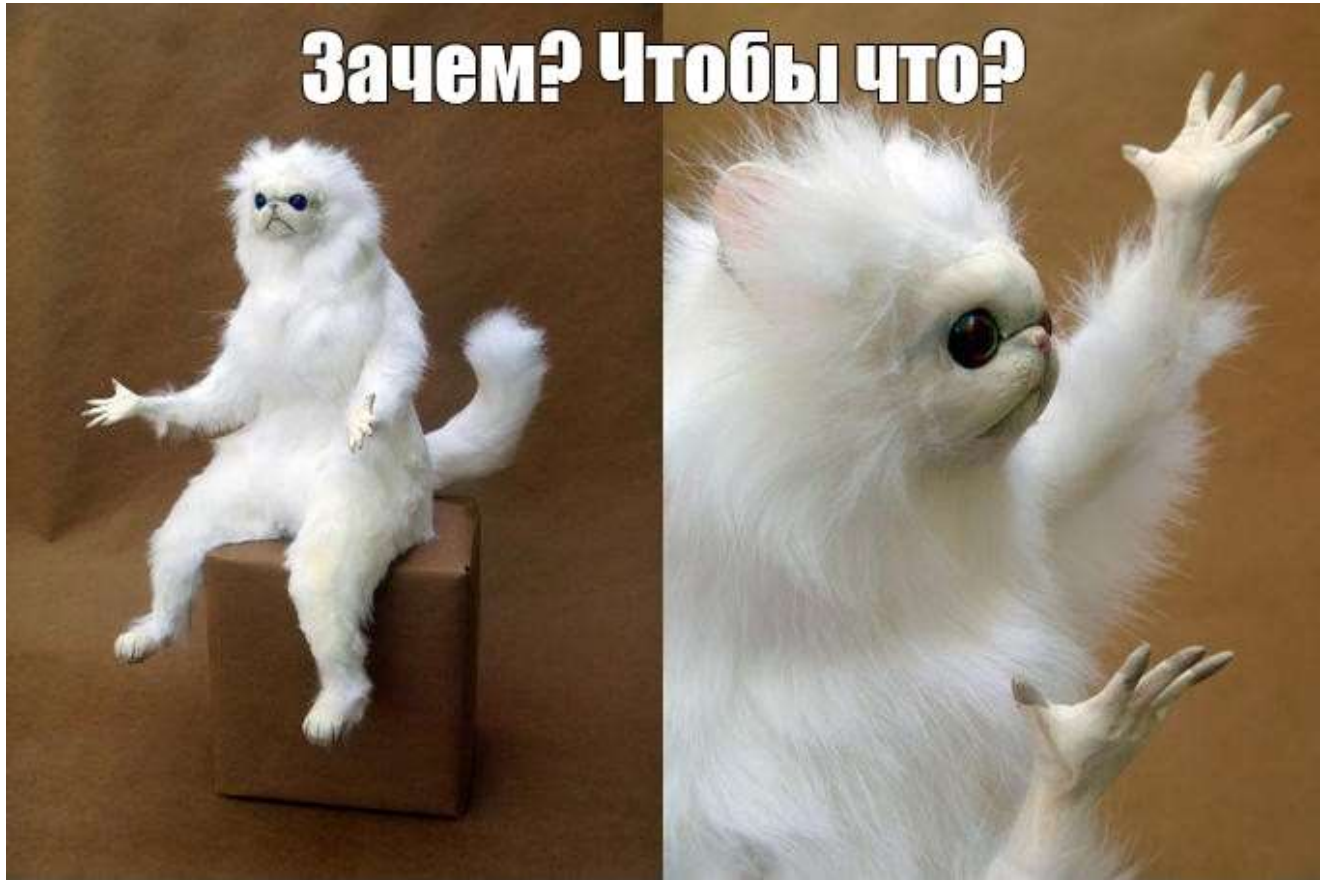
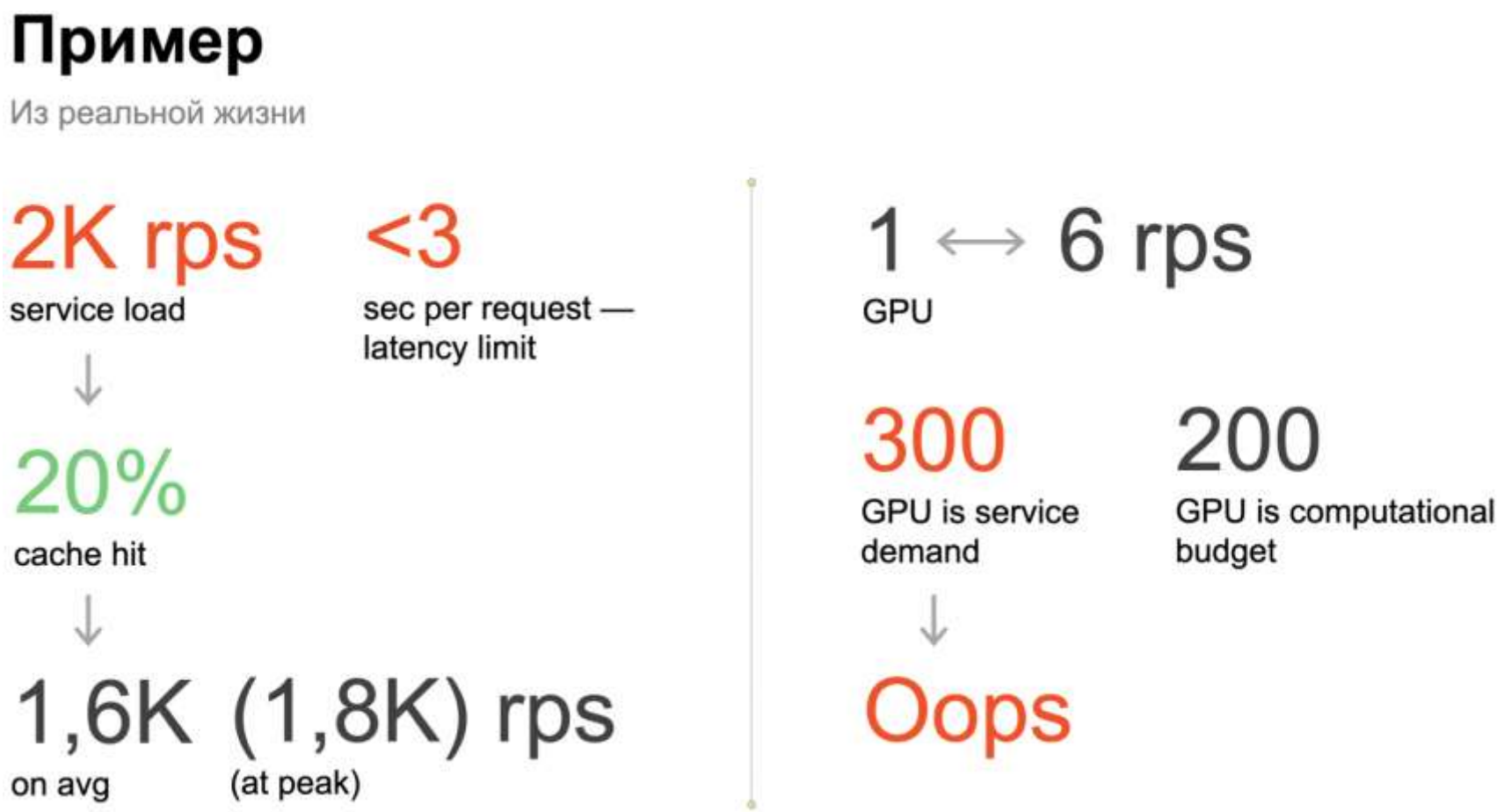
Ускорение LLM

Мотивация



Наглядный пример



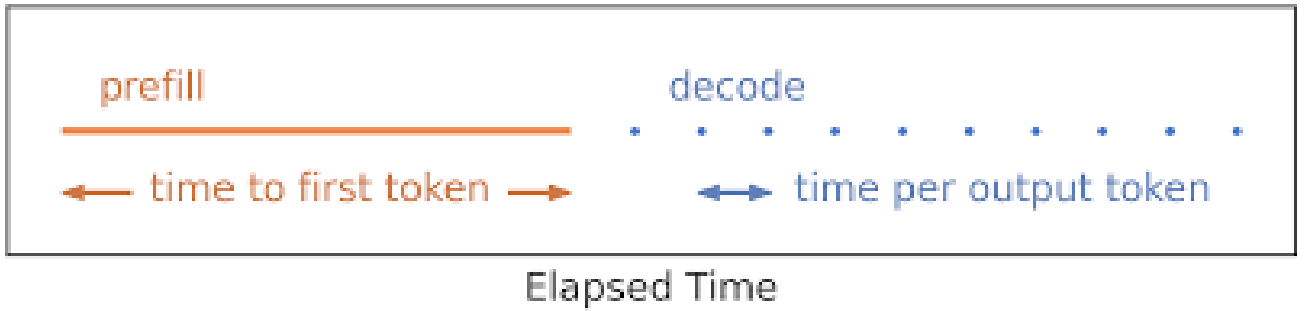
Ускорение инференса LLM

Цели ускорения LLM

- Снижение задержки - **latency**
- Увеличение пропускной способности – **throughput**
- Обработка длинного контекста
- Сокращение стоимости на запрос

Декомпозиция latency

- **TTFT (Time to First Token) – Prefill фаза** – обработка входных токенов для генерации первого выходного токена
- **Decode фаза** – генерирует токены последовательно, один за другим
- **Prefill latency** – задержка до первого выходного токена
- **Decode latency** – задержка одного шага декодирования



Prefill фаза

- Все входные токены обрабатываются параллельно
- Фаза является **compute-bound** и эффективно использует GPU
- Вычислительная сложность: $O(L^2d)$ для attention и $O(Ld^2)$ для FFN*
- Все входные токены обрабатываются параллельно, поэтому производительность выше при больших батчах

*L – длина последовательности, d – размер внутреннего представления модели

Decode фаза

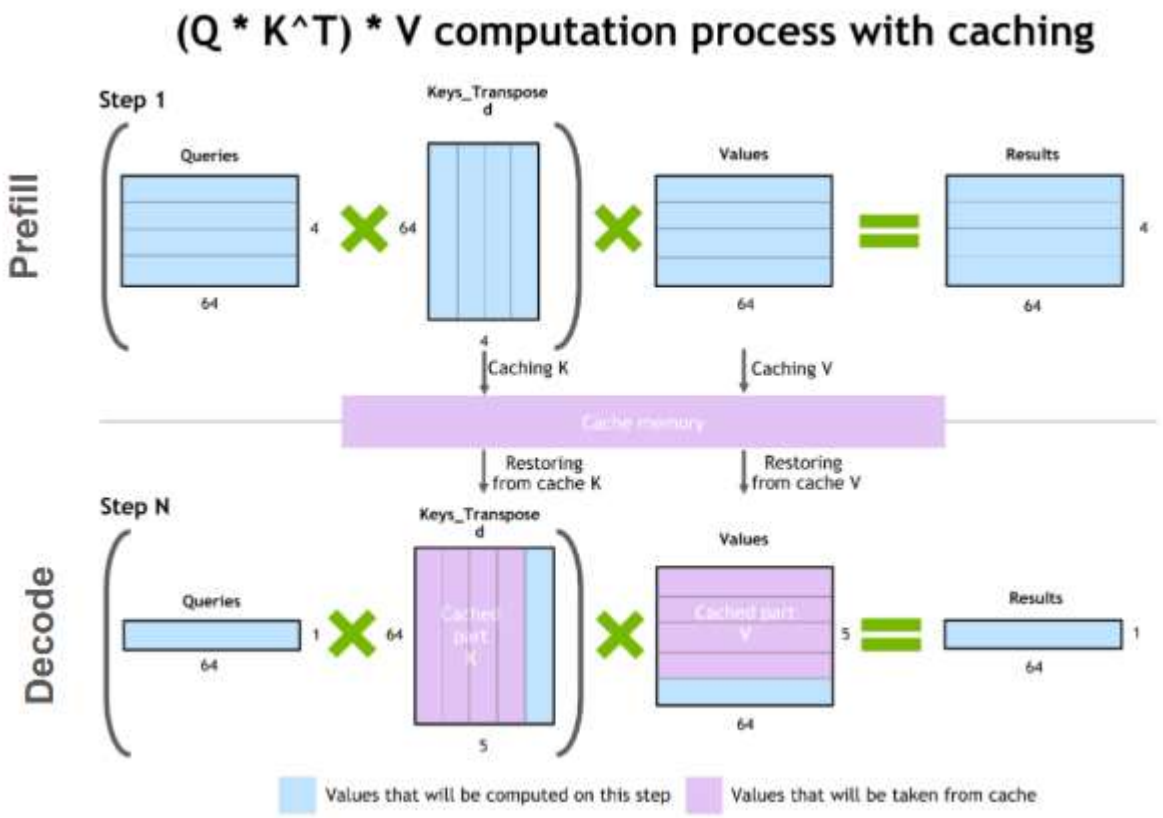
- Query length = 1 (новый токен), context length = L (все предыдущие токены)
- Доминируют матричные умножения и обращения к **KV-кешу**
- Фаза является **memory-bound** - ограничена пропускной способностью памяти
- По мере увеличения длины выходной последовательности, decode фаза занимает до 80-100% общего времени
- Вычислительная сложность: $O(Ld)$ для attention и $O(d^2)$ для FFN*
- Вычисления идут по одному токену, что снижает степень параллелизма

KV-cache

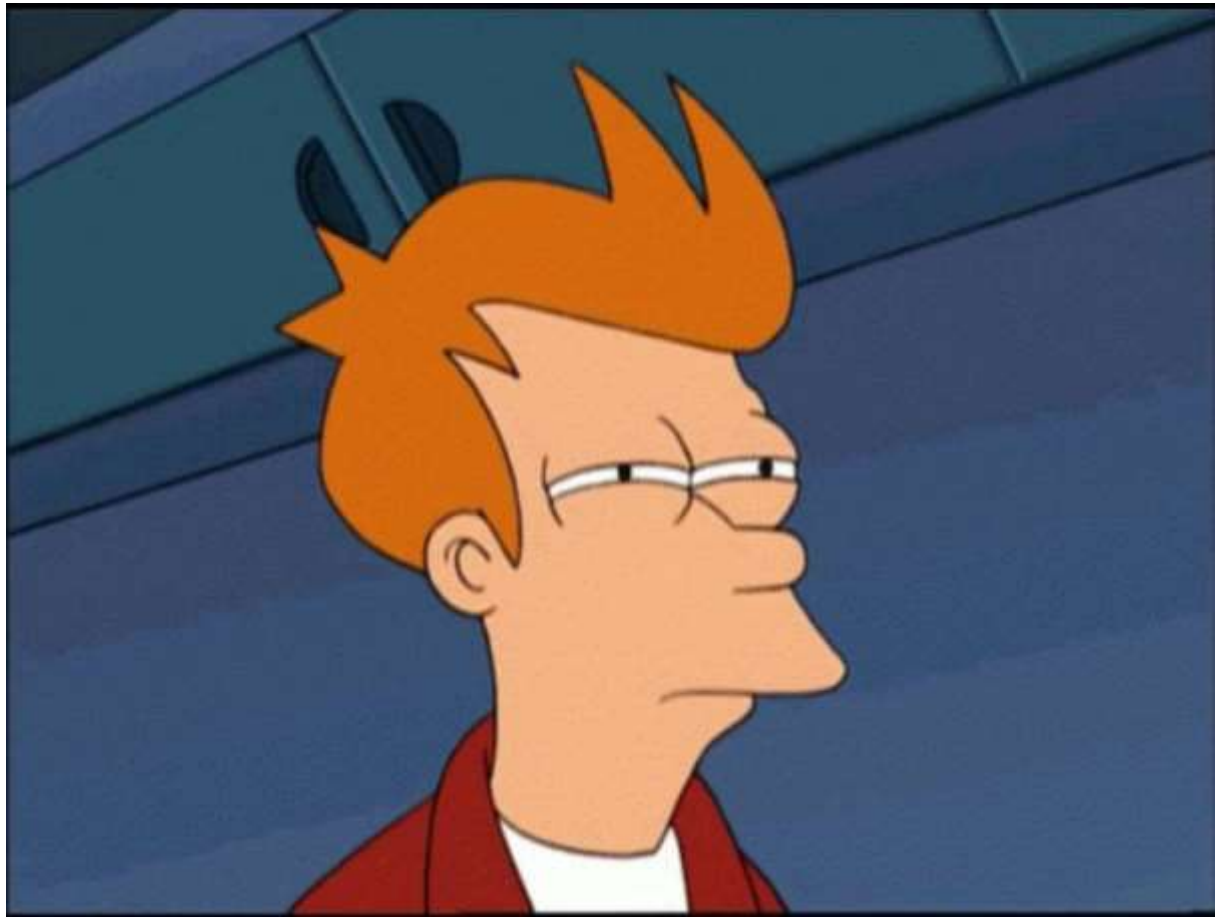
Во время генерации текста LLM работает рекурсивно: для каждого нового токена вычисляется механизм внимания по всей предыдущей последовательности.

Если выполнять эти вычисления "в лоб", модель пересчитывает K и V для каждого токена заново, что приводит к избыточным операциям и квадратичной сложности по времени.

KV-cache позволяет сохранять рассчитанные K и V вектора для всех предыдущих токенов, повторно используя их для новых шагов и уменьшая сложность до линейной.



А что там по памяти?

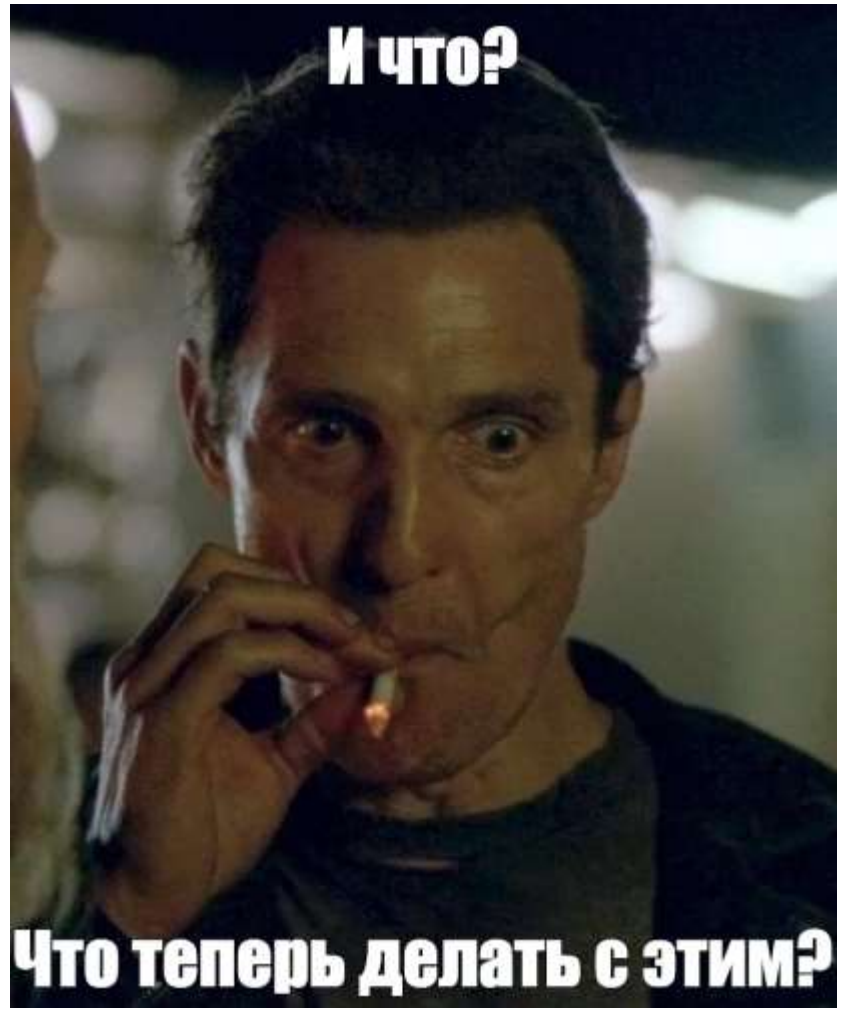


KV-cache memory footprint

KV-cache ≈ 2 * batch_size * seq_len * num_layers * n_heads * head_dim (hidden_size / n_heads) * precision

Для GPT-2 XL:

KV-cache per token ≈ 2 * 16 * 1000 * 48 * 25 * (1600 / 25 = 64) * 2 = 4.5GB


И что?
Что теперь делать с этим?

Оптимизация KV-cache

- MQA/GQA
- Квантизация
- Offloading & paging

Внешние факторы роста latency

- Частота стриминга
- Задержка сети (особенно для распределенного инференса)
- Overhead на предобработку

Ключевые пользовательские метрики

- **TTFT (Time-To-First-Token)** - отражает prefill + инициализацию KV-cache.
- **TPS (Tokens Per Second)** - скорость генерации decode фазы
- **E2E Latency (End-to-End latency)** - полное время на ответ при генерации N токенов. Зависит от TTFT + TPS × N
- Перцентили latency **(P50, P95, P99)**
- **Стоимость токена -** GPU-секунды × цена / количество токенов

Ключевые системные метрики

- **GPU SM occupancy** (Utilization of Streaming Multiprocessors) - процент времени, когда вычислительные ядра GPU активно выполняют инструкции.

Высокая загрузка говорит о том, что ядра GPU эффективно заняты, а низкая — что есть зависимость от памяти, передачи данных или плохая оптимизация.

- **Memory Bandwidth Utilization** - доля пропускной способности шины памяти, используемой при чтении и записи данных в/из GPU памяти.

Высокий memory BW utilization показывает, что пропускная способность памяти — узкое место.

- **HtoD** (Host-to-Device) / **DtoH** (Device-to-Host) Transfer Time - время передачи данных между CPU (host) и GPU (device)

Инструменты профилирования

GPU-уровень

- NVIDIA Nsight Systems / Nsight Compute – детальный анализ kernel’ов
- nvidia-smi – базовые метрики SM, memory BW

PyTorch-уровень

- PyTorch Profiler (torch.profiler) – вывод op-level timing trace.
- TensorBoard plugin – визуализация профилей

Production stack

- vLLM profiling tools – встроенные метрики по throughput, batching, cache hit rate
- Triton Inference Server Metrics – latency, throughput, GPU-утилизация
- Prometheus / Grafana – обеспечение мониторинга в реальном времени

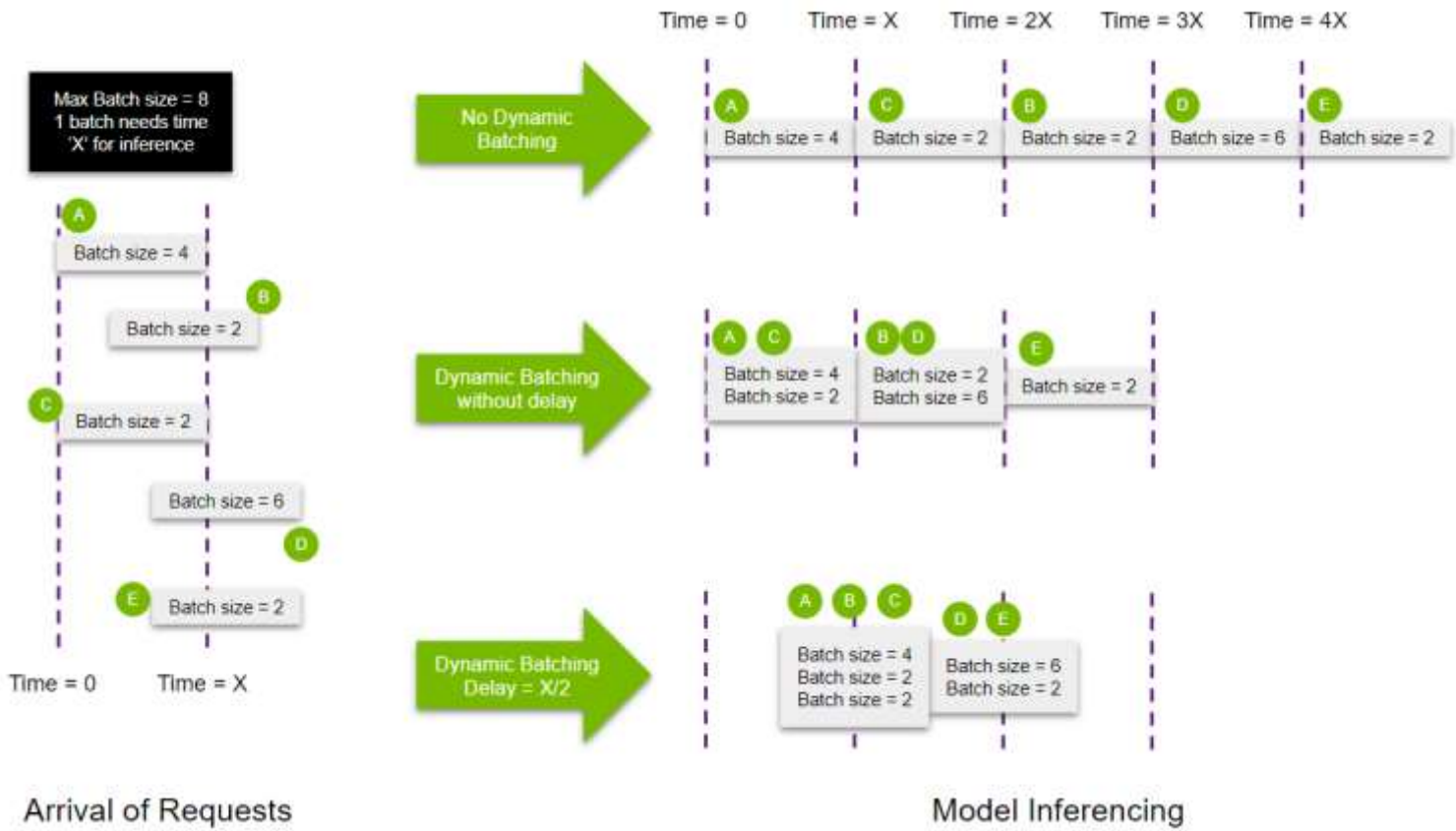
Оптимизации для Compute-bound задач

- AMP и квантизация
- Fusion Kernels (слияние операций CUDA)
- Оптимизированные реализации Attention (FlashAttention, …)
- Параллелизм по тензорам (Tensor Parallelism) и пайплайнам (Pipeline Parallelism)
- Аккуратная работа с batch size / sequence length

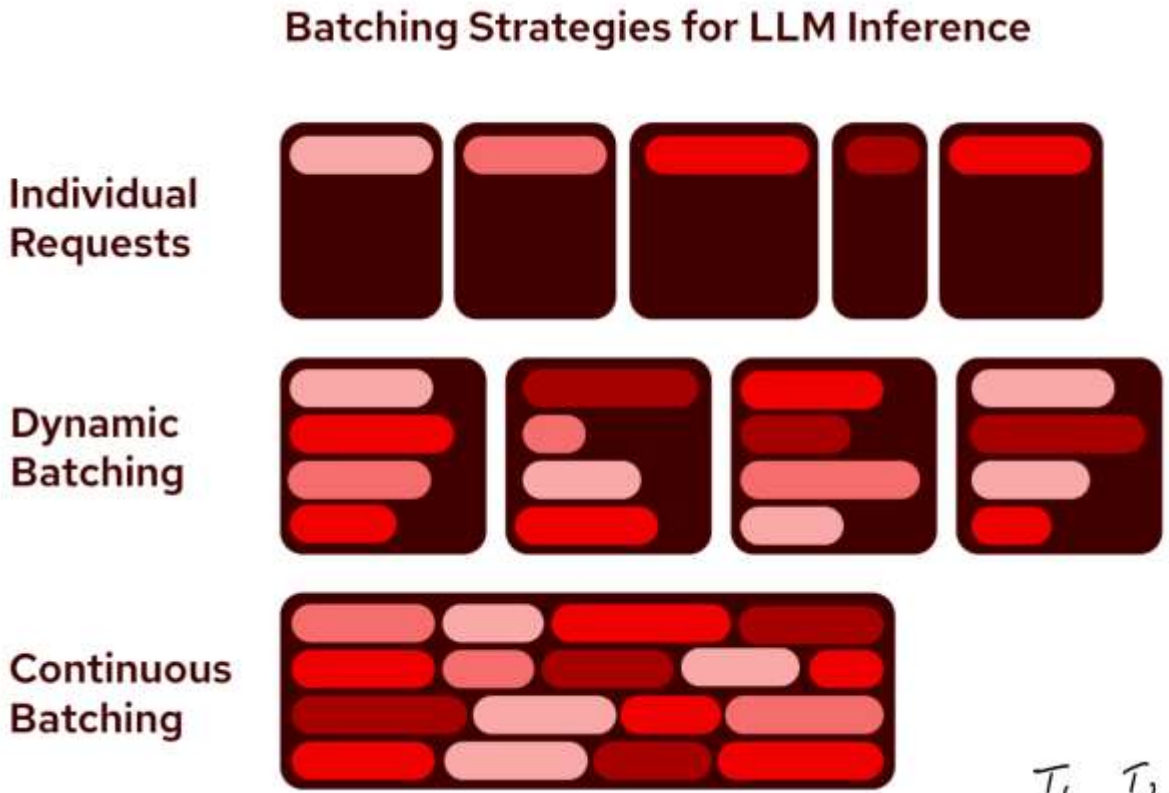
Оптимизации для Memory-bound задач

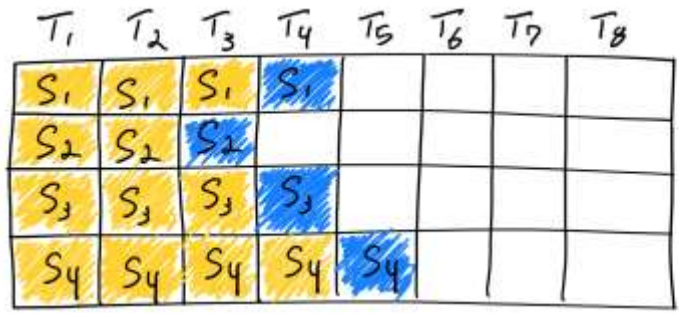
- KV-cache и квантизация, сжатие KV-cache
- Paging и Offloading KV-cache
- Оптимизация доступа к памяти
- Оптимизация transfer time (HtoD, DtoH)
- Использование архитектур с меньшим числом голов или моделей с эффективными attention-механизмами

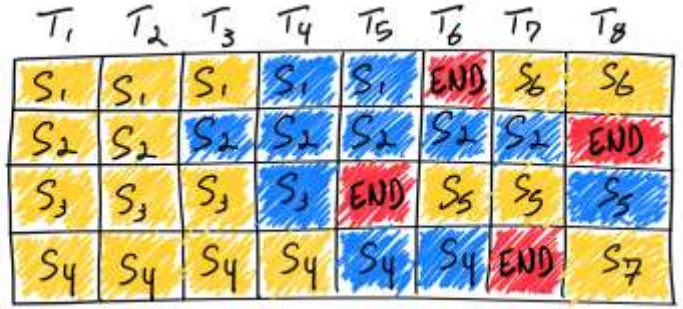
Dynamic Batching



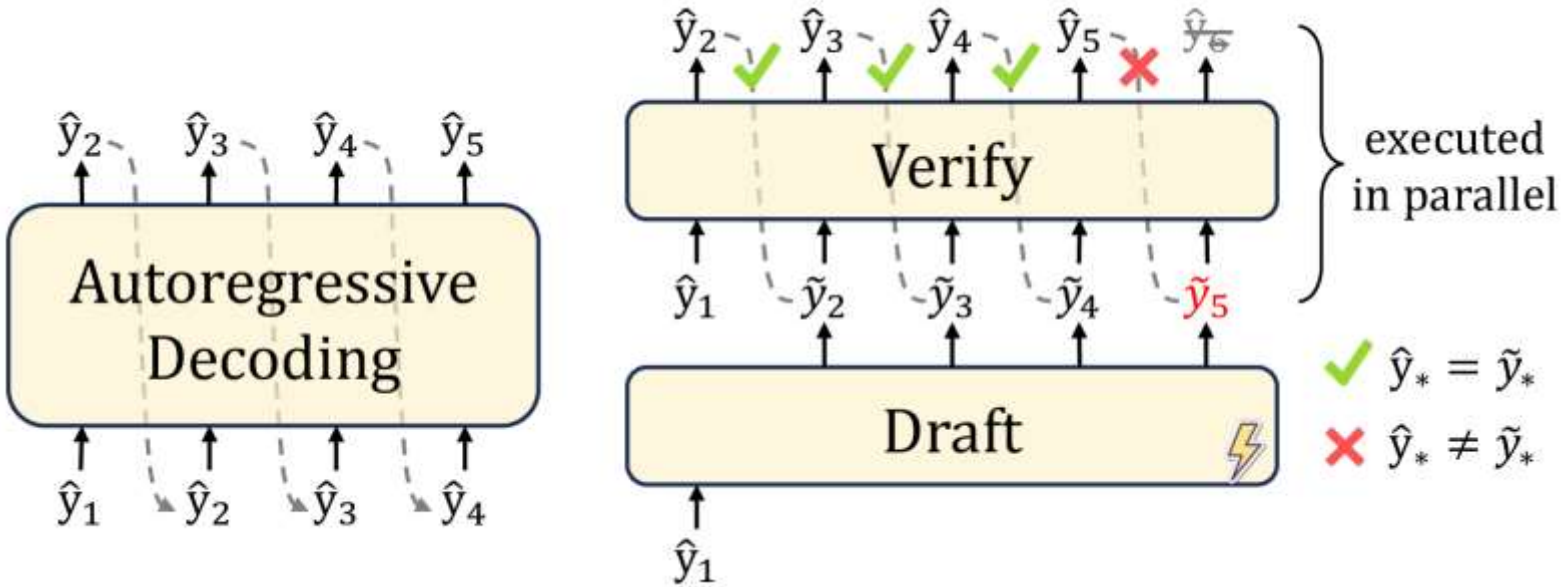
Continuous Batching







Спекулятивный декодинг



Спекулятивный декодинг - алгоритм

Speculative Sampling (SpS) with Auto-Regressive Target and Draft Models

Given lookahead K and minimum target sequence length T.

Given auto-regressive target model $q(.|.)$ and auto-regressive draft model $p(.|.)$, initial prompt sequence $x0, \ldots, xt$.

Initialise $n \leftarrow t$.

while $n < T$ **do**

  for $t = 1 : K$ **do**

    Sample draft auto-regressively $x_t \sim p(x|, x_1, \ldots, x_n, x_1, \ldots x_{t-1})$

  end for $t = 1 : K$ **do**

  In parallel, compute $K + 1$ sets of logits from drafts $x_{t-1}, \ldots, x_K$:

$$q(x|, x_1, \ldots, x_n),\ q(x|, x_1, \ldots, x_n, x_1),\ \ldots\ q(x|, x_1, \ldots, x_n, x_1, \ldots x_K)$$

  for $t = 1 : K$ **do**

    Sample $r \sim U[0, 1]$ from a uniform distribution.

    if $r < \min\left(1, \frac{q(x|x_1, \ldots, x_{n+t-1})}{p(x|x_1, \ldots, x_{n+t-1})}\right)$ **then**

      Set $x_{n+t} \leftarrow x_t$ и $n \leftarrow n + 1$.

    else

      sample $x_{n+t} \sim (q(x|x_1, \ldots, x_{n+t-1}), - \ldots + p(x|x_1, \ldots, x_{n+t-1}))_+$ and exit for loop.

    end if

  end for

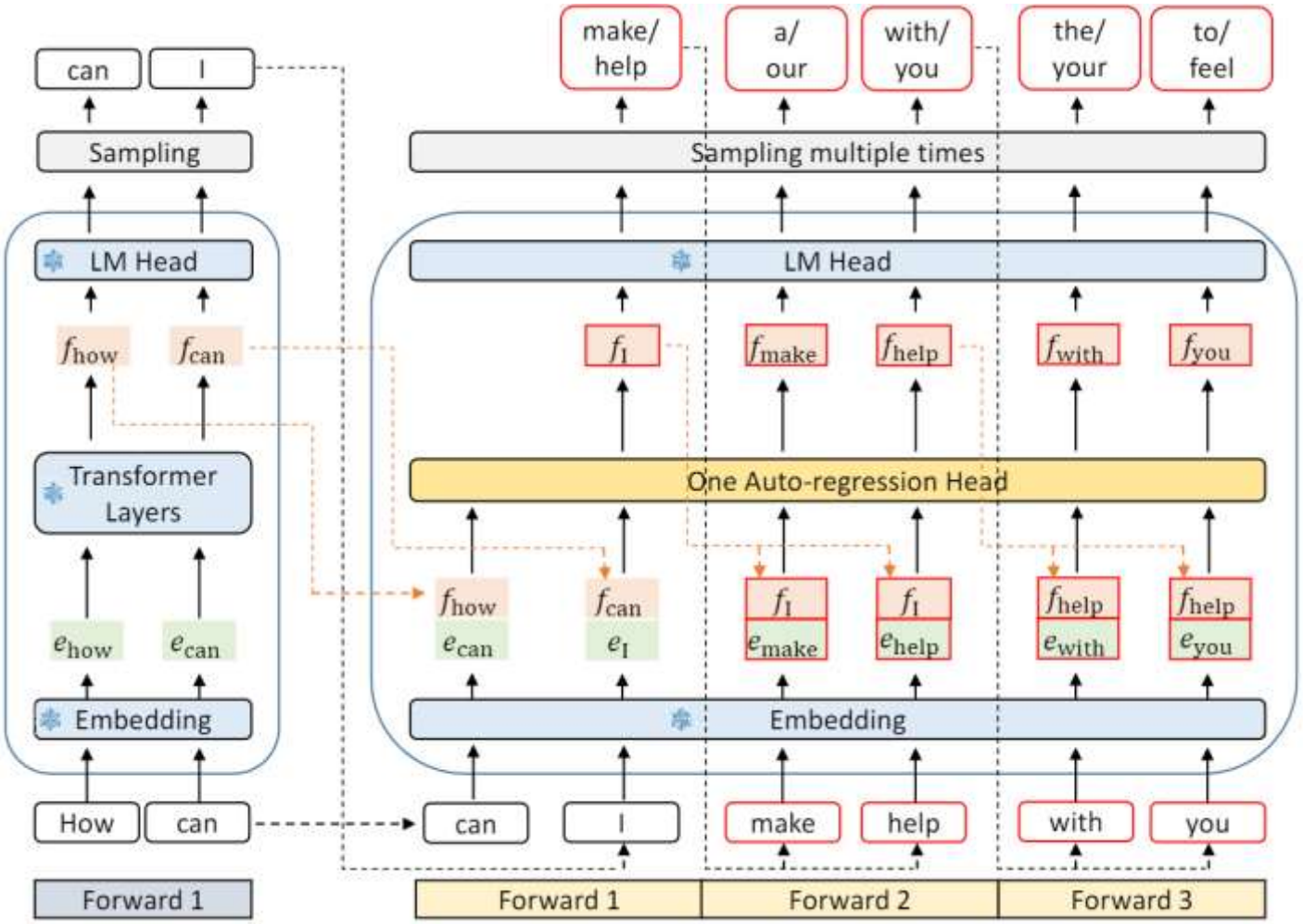
  If all tokens $x_{n+t}, \ldots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \ldots, x_n, x_{n+K})$ and set $n \leftarrow n + 1$.

end while

Спекулятивный декодинг - EAGLE

Черновая» модель заранее порождает несколько будущих токенов не линейной цепочкой, а деревом (несколько веток, несколько шагов вперёд).

Основная модель проверяет эти предложения и максимально длинным префиксом принимает их, а при первом несовпадении корректно пересэмплирует, чтобы итоговое распределение строго совпадало с распределением большой модели

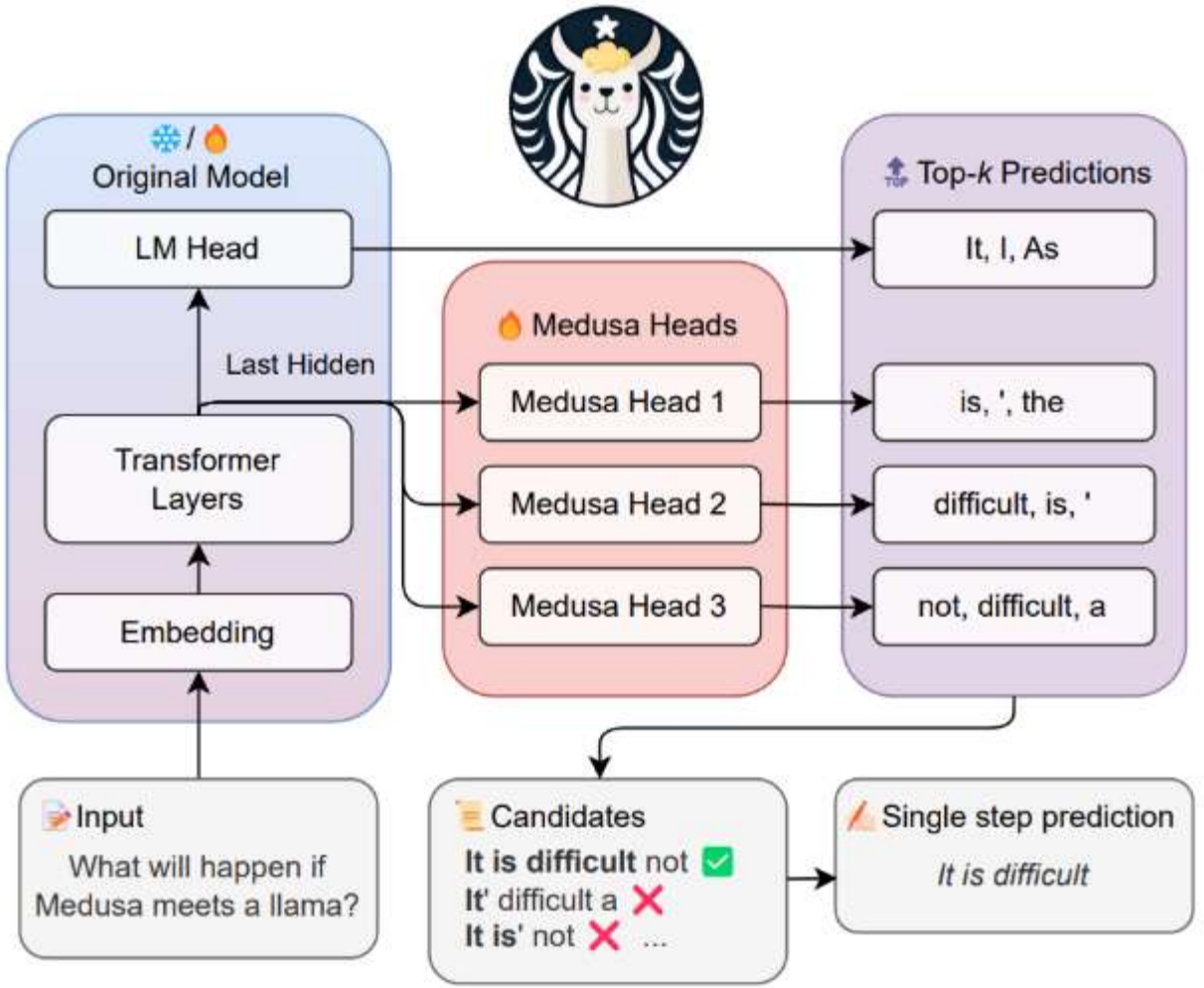


EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty

Спекулятивный декодинг - Medusa

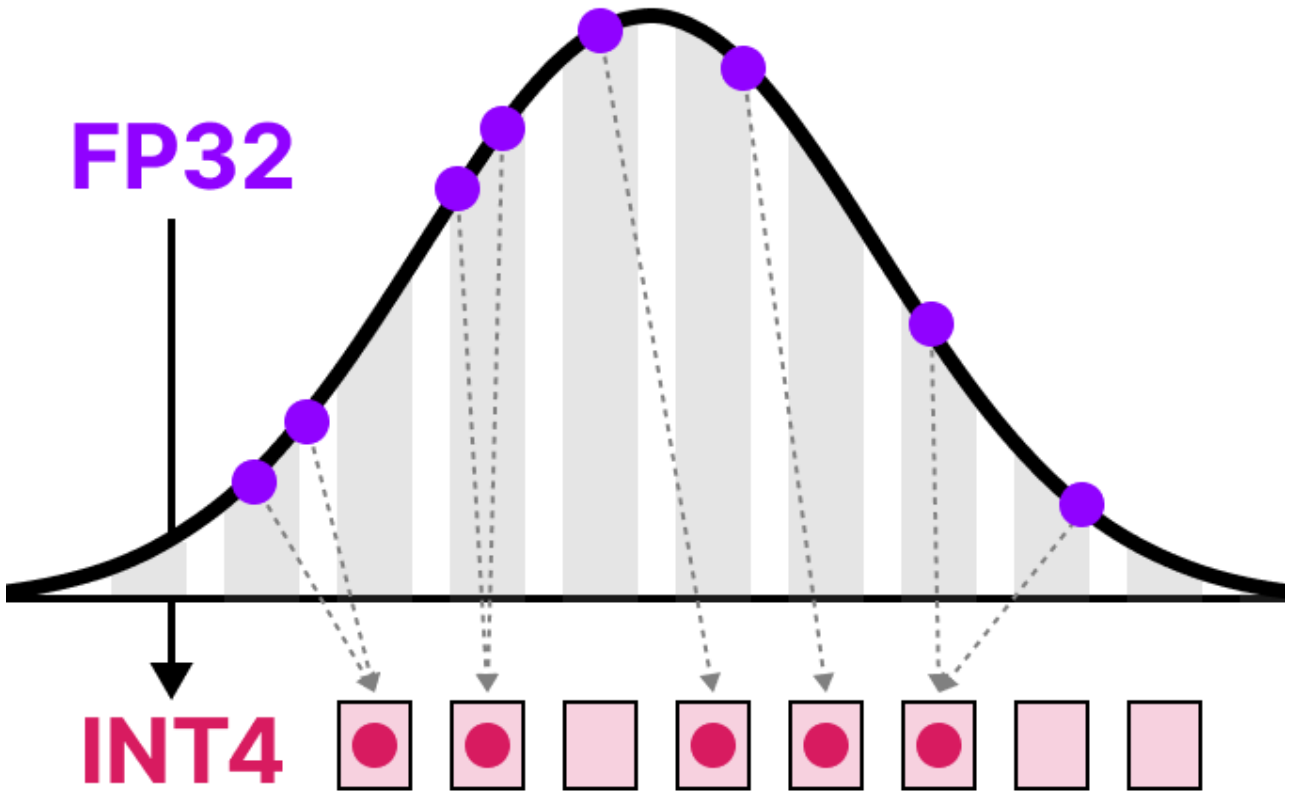
Medusa представляет инновационный подход, добавляющий множественные головы декодирования поверх последних скрытых состояний LLM:

- Каждая голова Medusa реализована как однослойная feed-forward сеть с residual connection
- Обучаются только головы, основная модель остается замороженной
- Для параллельной верификации множественных кандидатов используется tree attention

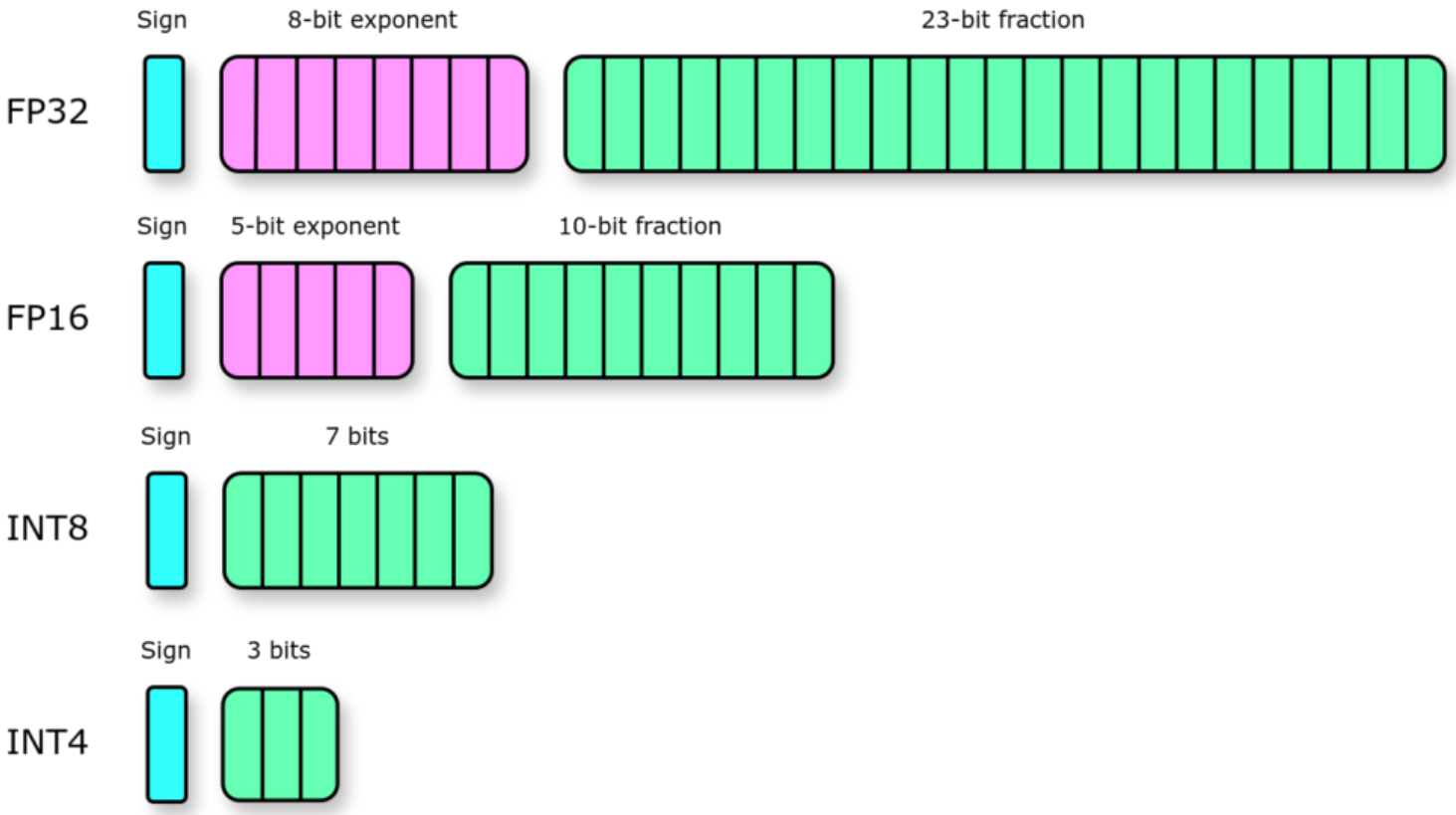


[Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads]

Квантизация



Представление чисел

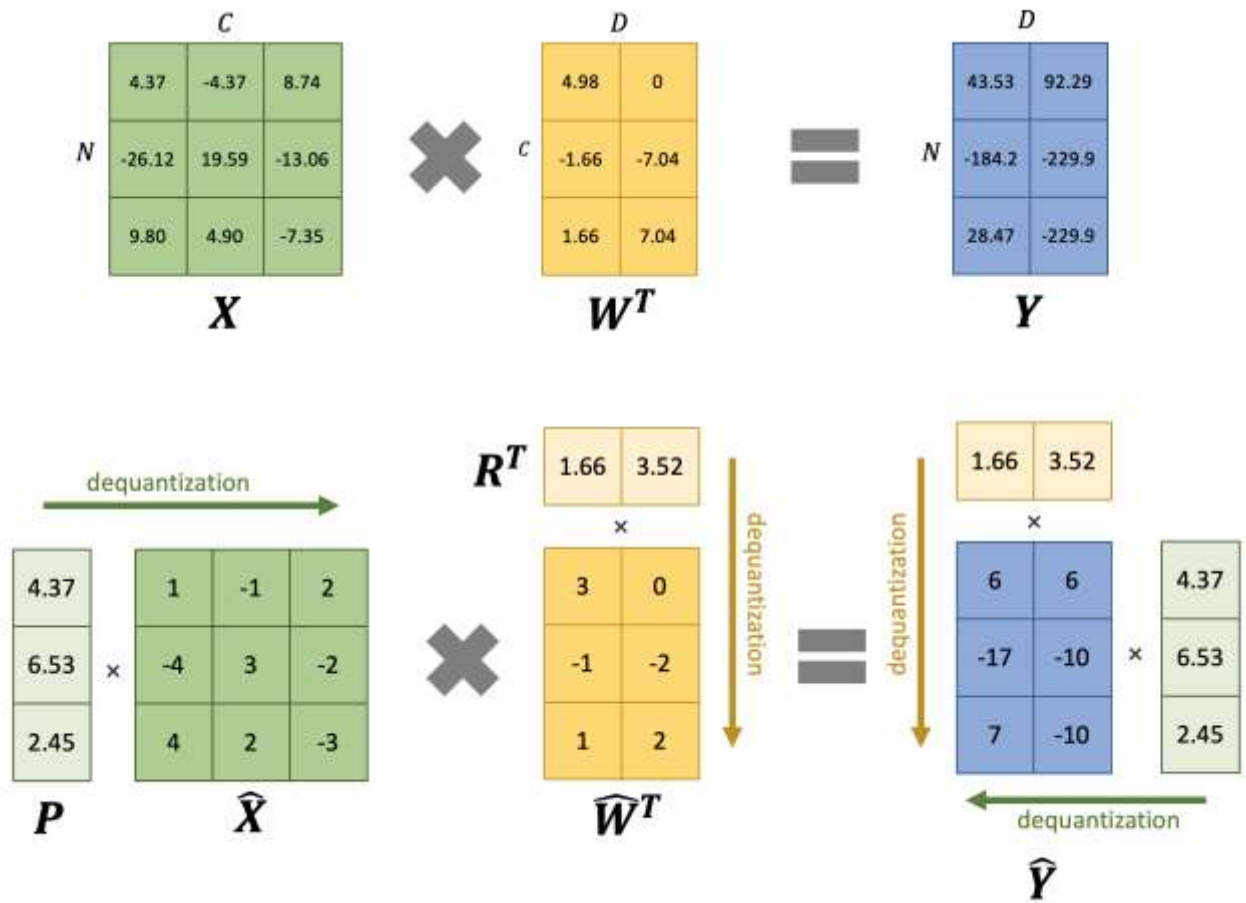


IEEE754

Квантизация - идея

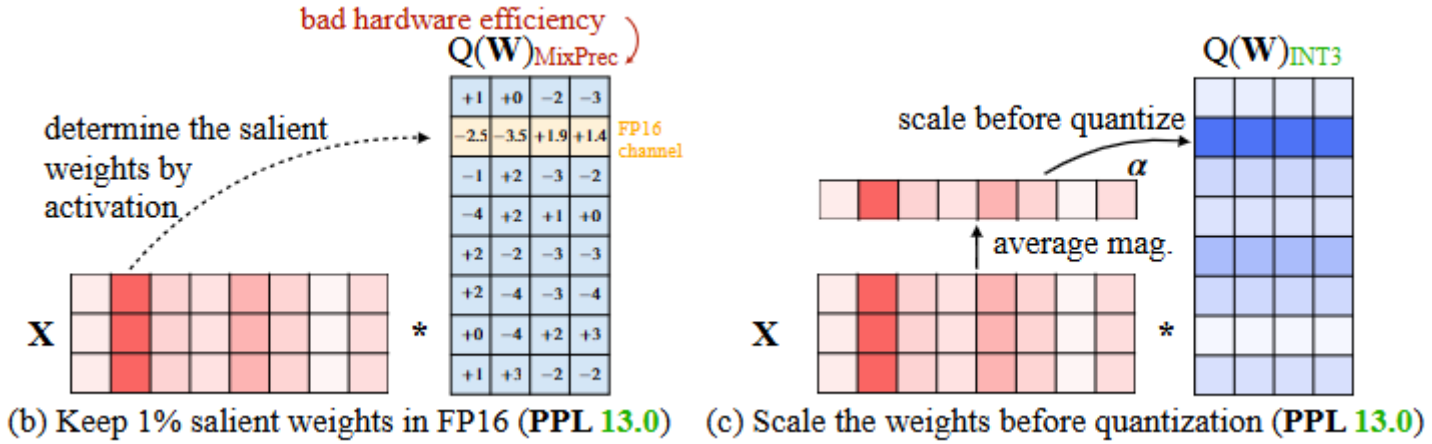
$$\mathbf{X}_{\text{quant}} = \text{round}\left(\text{scale} \cdot \mathbf{X} + \text{zeropoint}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\mathbf{X}_{\text{quant}} - \text{zeropoint}}{\text{scale}}$$



Квантизация – weight-only – AWQ

Идея: не все каналы/столбцы матриц весов одинаково важны для вывода модели; если измерить, какие входные каналы чаще «возбуждаются», можно изменить масштабирование весов и защитить «выбросы», чтобы 4-битная аппроксимация почти не портила ответ



AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration

Квантизация – weights-only – GPTQ

В отличие от простого округления, GPTQ минимизирует ошибку матричных умножений, чтобы результат работы слоя с квантизованными весами максимально соответствовал оригиналу

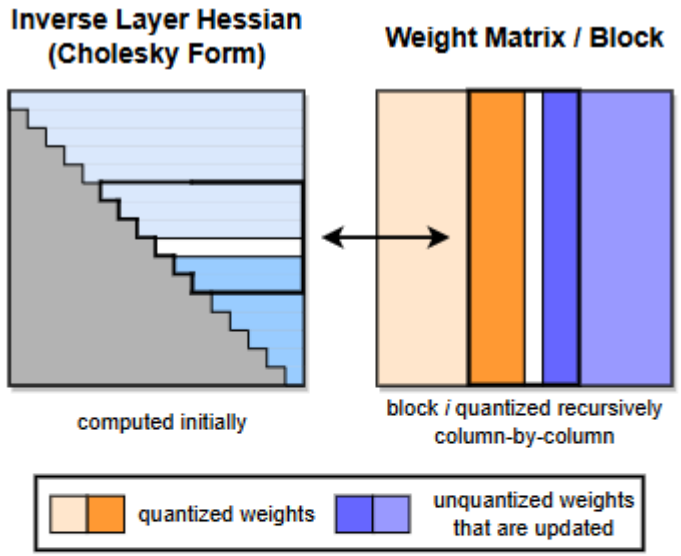


Figure 2: GPTQ quantization procedure. Blocks of consecutive *columns* (bolded) are quantized at a given step, using the inverse Hessian information stored in the Cholesky decomposition, and the remaining weights (blue) are updated at the end of the step. The quantization procedure is applied recursively inside each block: the white middle column is currently being quantized.

[GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers]

Квантизация – weights+activations – FP8

Операции (умножение, сложение) выполняются на аппаратных ядрах, оптимизированных под FP8, что снижает энергопотребление и повышает пропускную способность

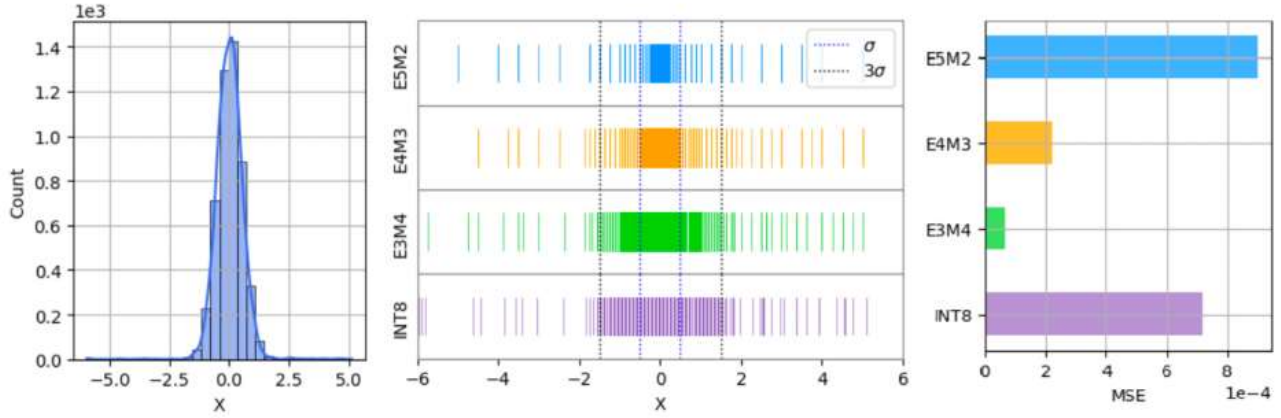
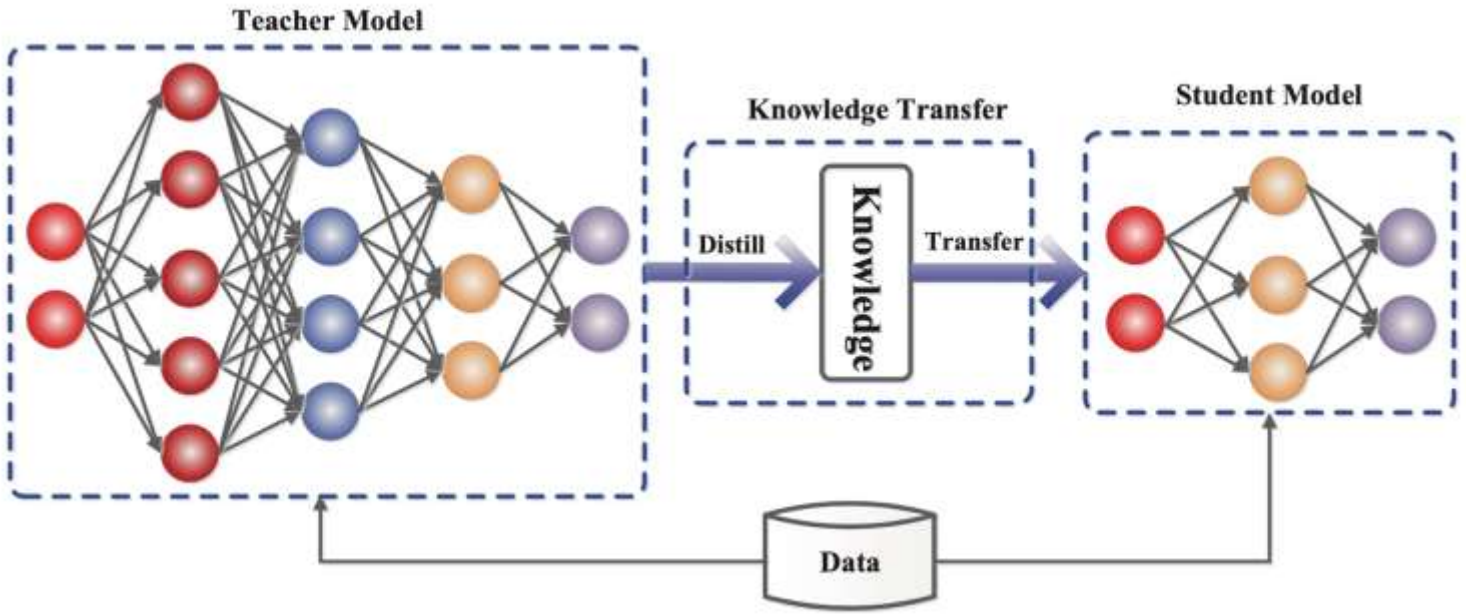


Figure 1. (**left**) Histogram of the tensor $X \sim \mathcal{N}(\mu = 0.0, \sigma^2 = 0.5)$, that contains a small number (1%) of outliers uniformly distributed between -6.0 to 6.0. (**center**) Distribution of quantized values represented by E5M2, E4M3, E3M4 and INT8 data formats. (**right**) Overall quantization error as measured by mean-square-error (MSE).

[Efficient Post-training Quantization with FP8 Formats]

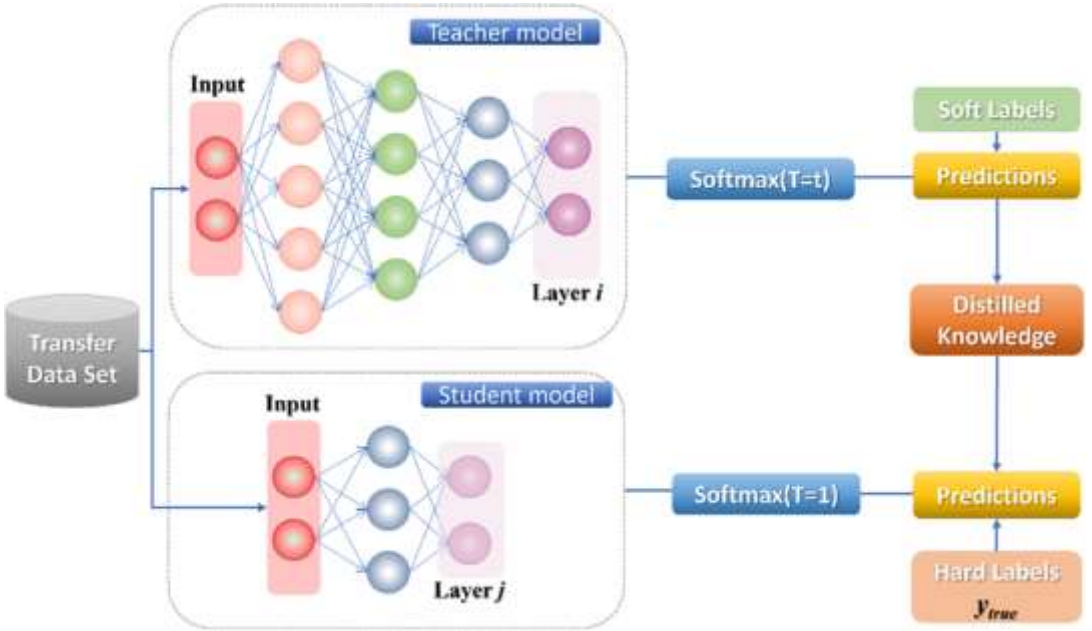
Дистилляция – hard label

Генерируем таргеты для выборки учителем, а затем обучаем на них студента в обычном режиме



Дистилляция – soft label

Вместо того, чтобы использовать на каждом шаге hard-метки, дистиллированная модель обучается так, чтобы соответствовать полному распределению вероятностей учителя



Дистилляция – on-policy

Решает проблему несоответствия между обучением и инференсом. Вместо использования фиксированного набора выходных последовательностей, студент генерирует свои собственные выходы и получает обратную связь от учителя на уровне токенов.

Что когда и как использовать

- Если compute-bound:
  - Перейти на BF16/FP16/FP8. Weight-only квантизация может увеличить compute!
  - FlashAttention, оптимизированные ядра (Triton LLM/TensorRT, vLLM), фьюзинг операций
  - Увеличить батч, включить continuous батчинг
  - Эксперименты со спекулятивным декодингом – EAGLE 2
  - Дистилляция, модель поменьше
- Если memory bound:
  - Квантизация – weight-only, weight+activations
  - KV-cache + квантизация, управление KV-cache
  - FlashAttention
  - Prefix Caching
  - continuous батчинг, sequence packing
  - Архитектура с GQA/MQA
  - Дистилляция
  - Уменьшение контекстного окна