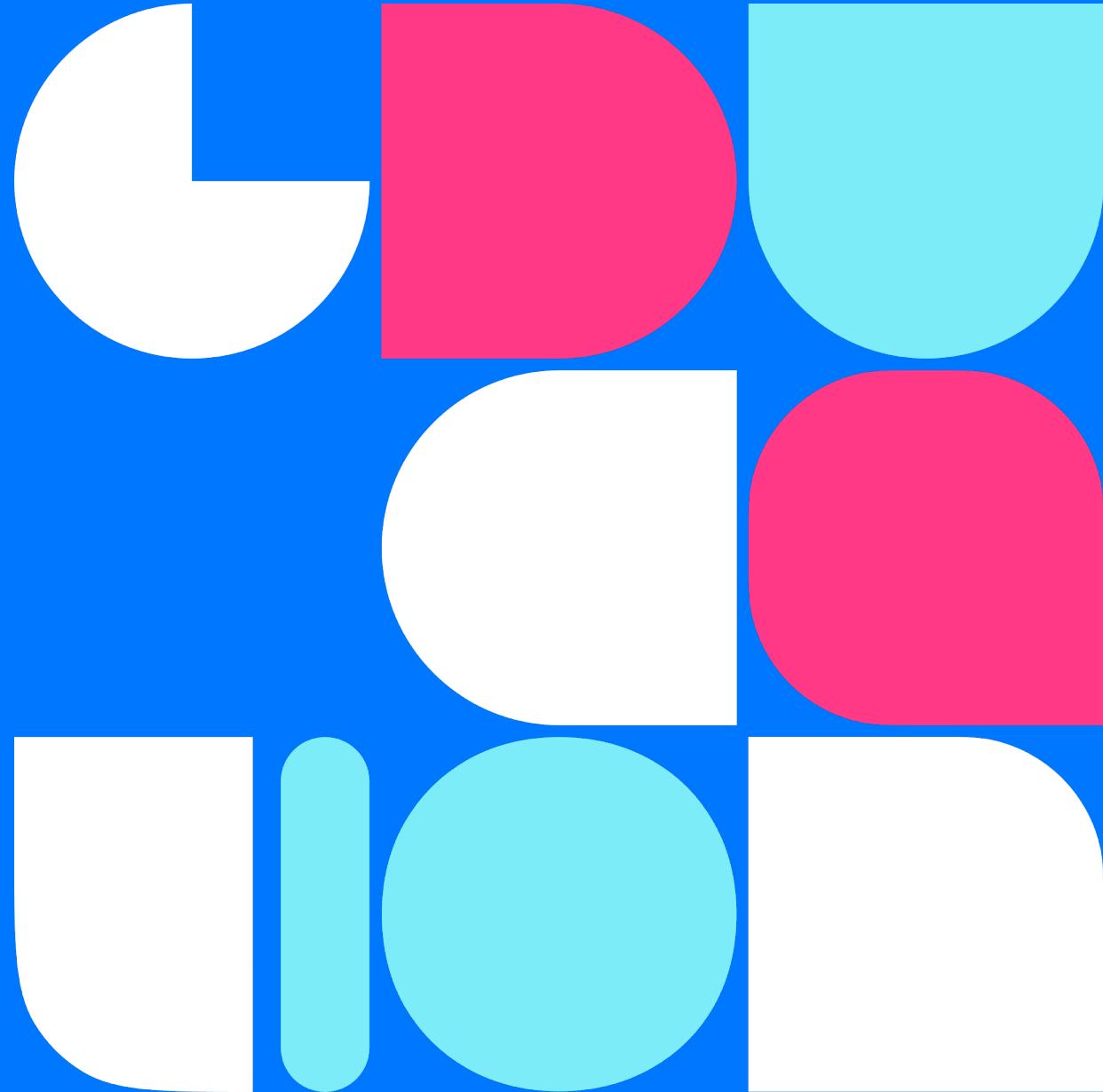




Transformers and beyond

Как развивались трансформеры

Егор Спирин

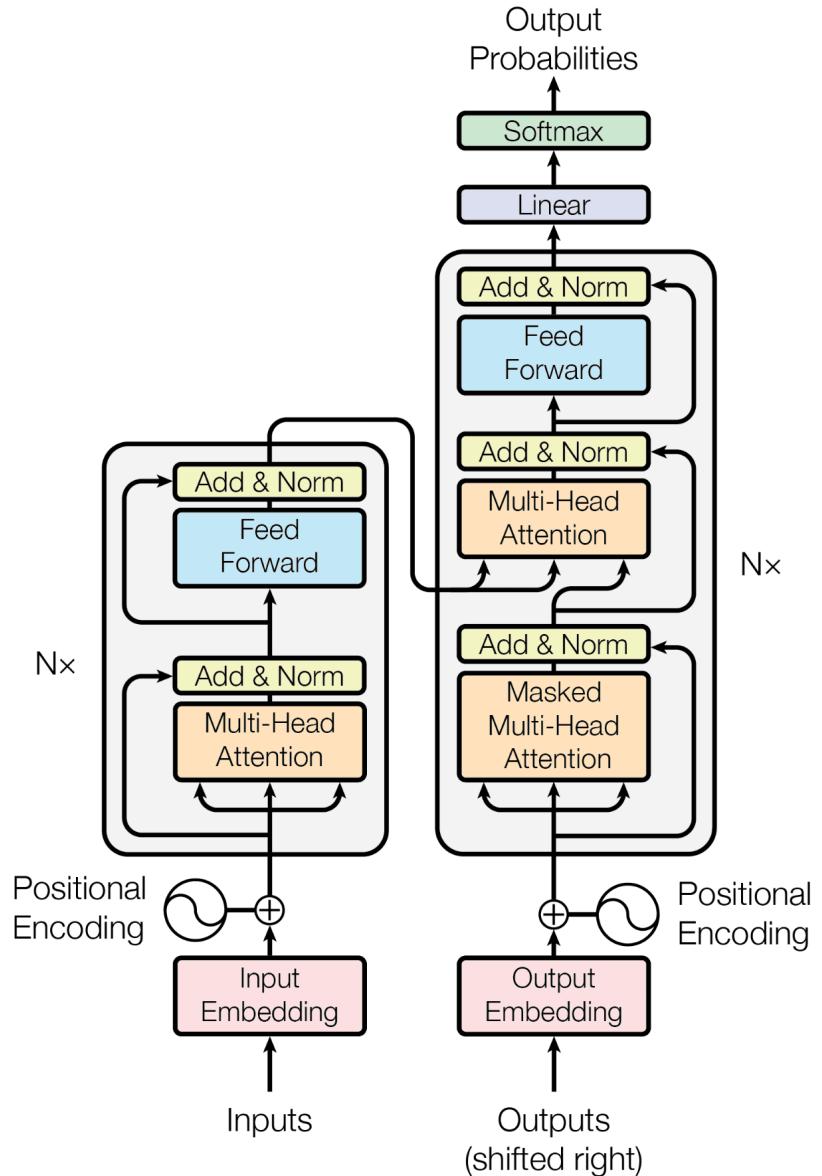


Quick recap



Attention is all you need

- 👉 Transformer — новая архитектура для естественного языка, проверка на машинном переводе
- 👉 Заточена под масштабирование на большие объемы данных — уходим от рекуррентной обработки данных
- 👉 Состоит из двух частей — Encoder и Decoder



Transformer Encoder

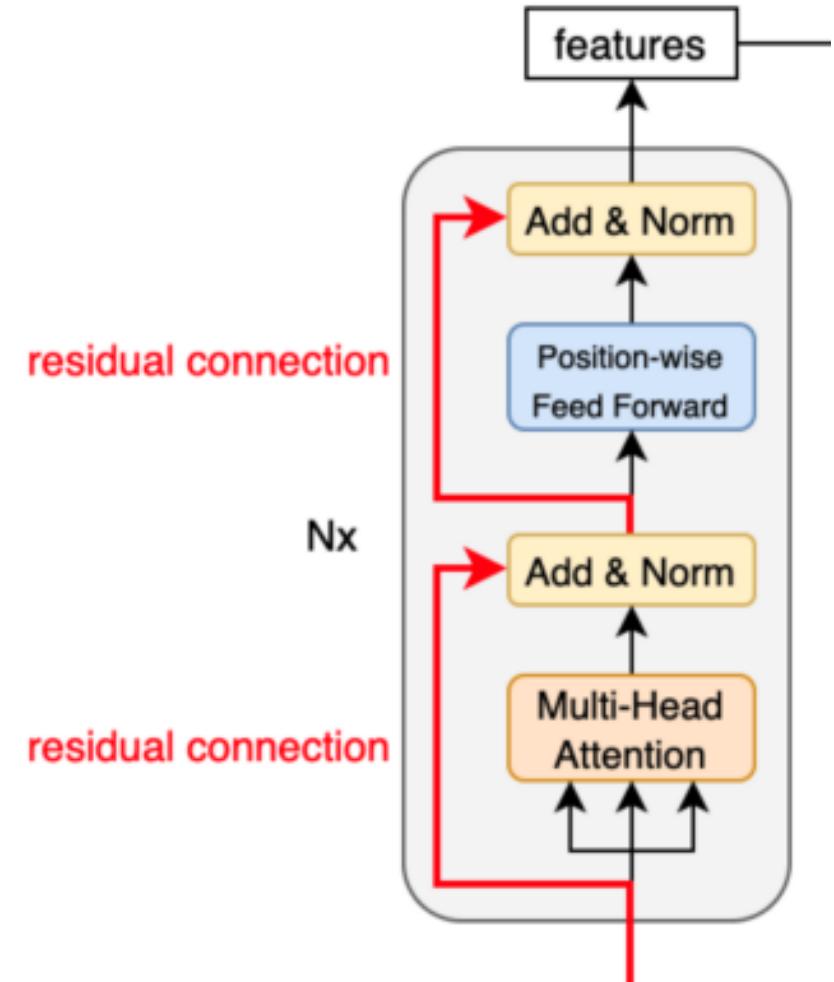
Transformer Encoder — состоит из последовательных
Transformer Encoder Layers

Transformer Encoder Layer:

- 👉 Self-Attention — механизм внимания, “изучаем” связь между элементами последовательности
- 👉 Feed-Forward — “запоминаем” признаки
- 👉 Слои нормализации и residual connection

Inputs: [batch size; seq len; hidden dim]

Features: [batch size; seq len; hidden dim]



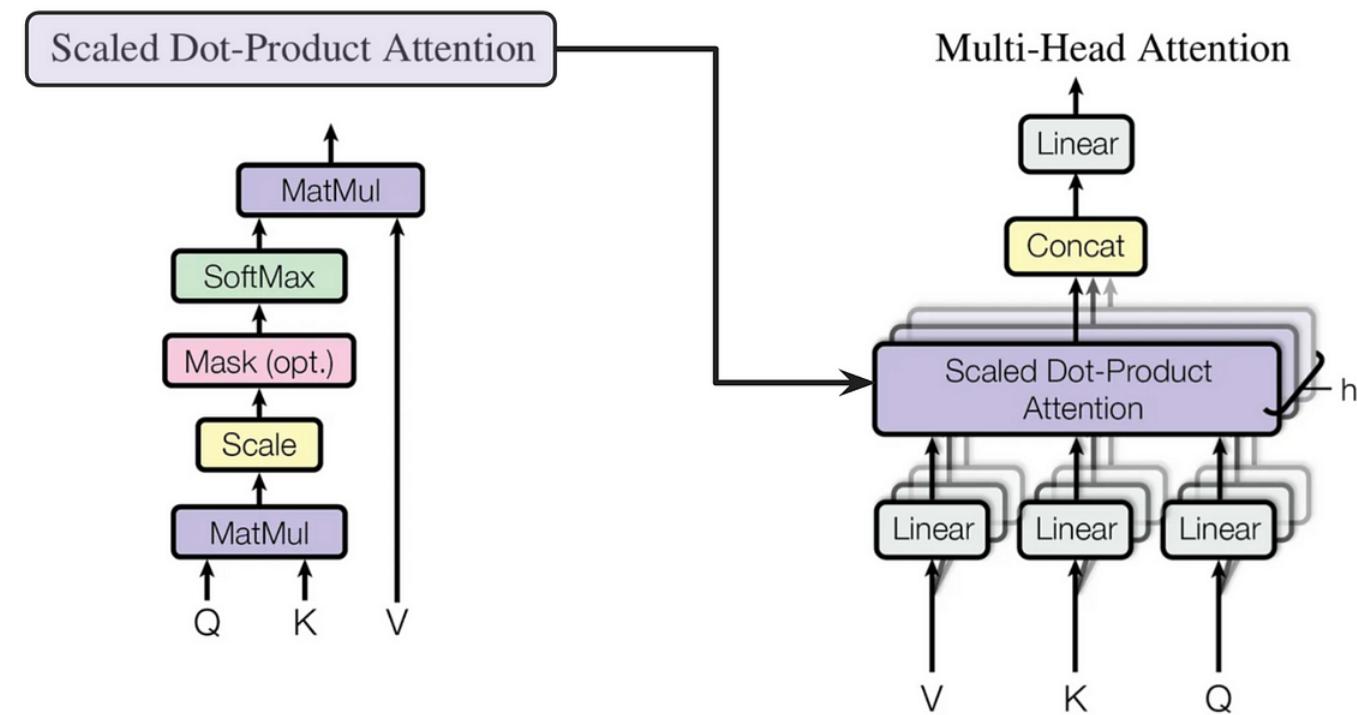
Scaled Dot-Product Attention

Механизм внимания, вдохновленный ассоциативным хранилищем

1. $\langle K; V \rangle$ — хранилище ключей и значений
2. Q — запросы поиска: для q найти похожие k и взять соответствующие v с весами "схожести"

В мире трансформеров:

- 👉 Self-attention — одинаковый вход для Q , K и V , но разные матрицы параметров
- 👉 QK^T — attention map, отражает взаимосвязь между токенами



Transformer Decoder

Transformer Decoder — состоит из последовательных **Transformer Decoder Layers**

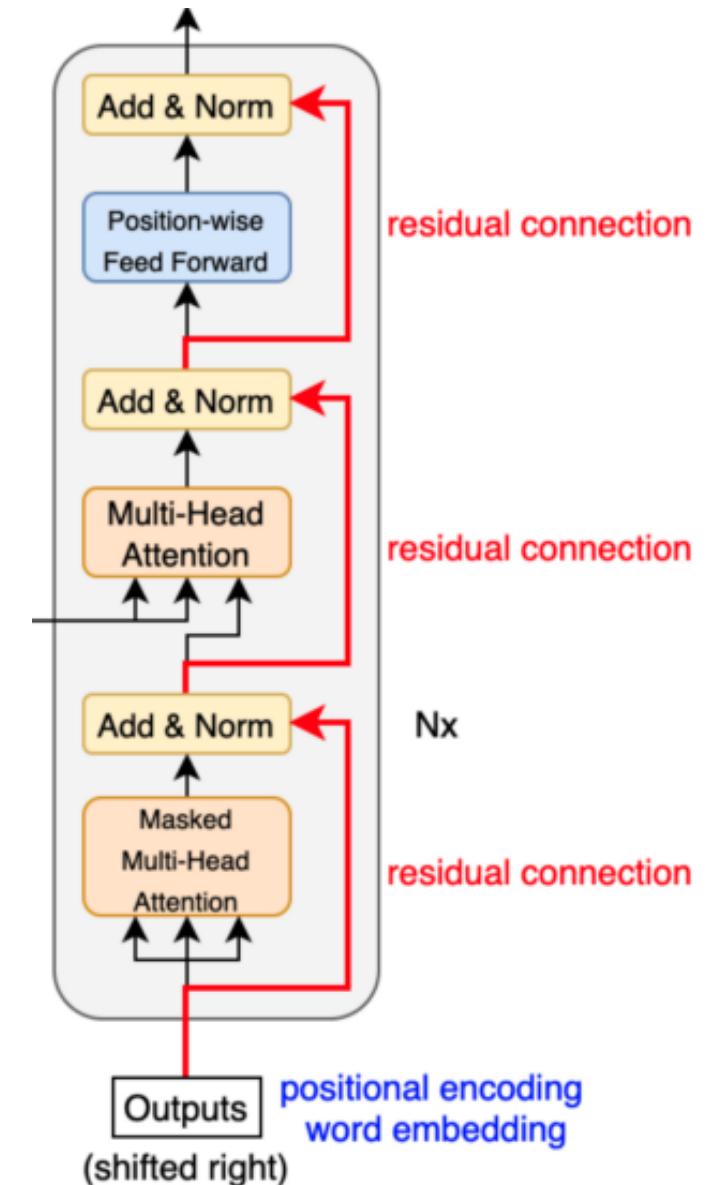
Transformer Decoder Layer:

- 👉 Практически полностью повторяет Transformer Encoder Layer
- 👉 Между Self-Attention и Feed-Forward добавляется Cross-Attention
- 👉 В Self-Attention запрещаем токенам смотреть в будущее через зануление части QK^T — применяется каузальная маска

Inputs: [batch size; seq len; hidden dim]

Conditions: [batch size; condition len; hidden dim]

Outputs: [batch size; seq len; hidden dim]



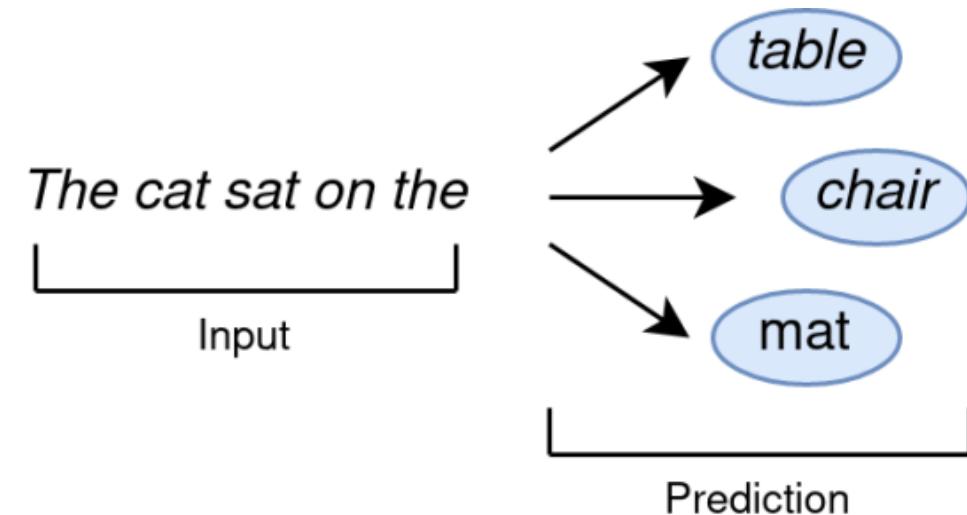
Языковое моделирование

Авторегрессивность — текущее значение последовательности зависит от предыдущих её значений

Языковая модель — авторегрессивная модель, предсказывает следующее слово в последовательности, основываясь на предыдущих словах, контексте

GPT and beyond:

- 👉 За основу взят Transformer Encoder
- 👉 Каузальная маска в Self-Attention
- 👉 Features передаются в LM голову, классификатор по словарю



Что было
далше

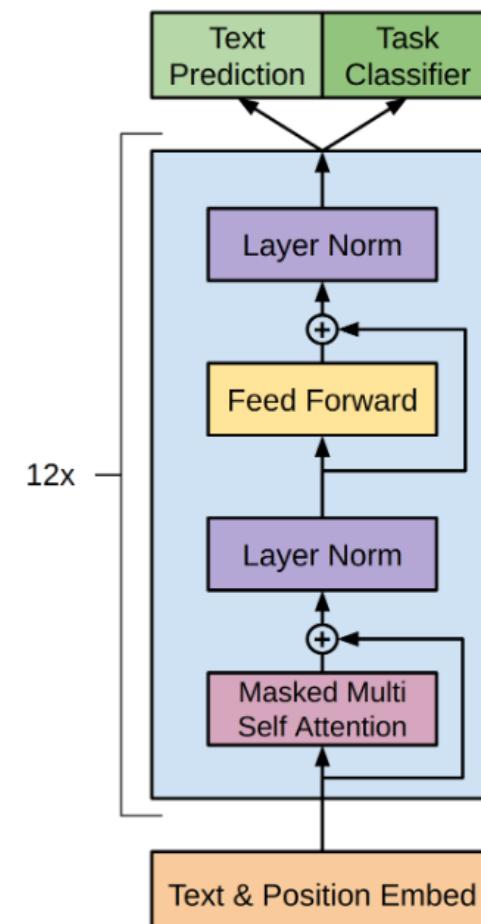


Transformers – не конкретная модель

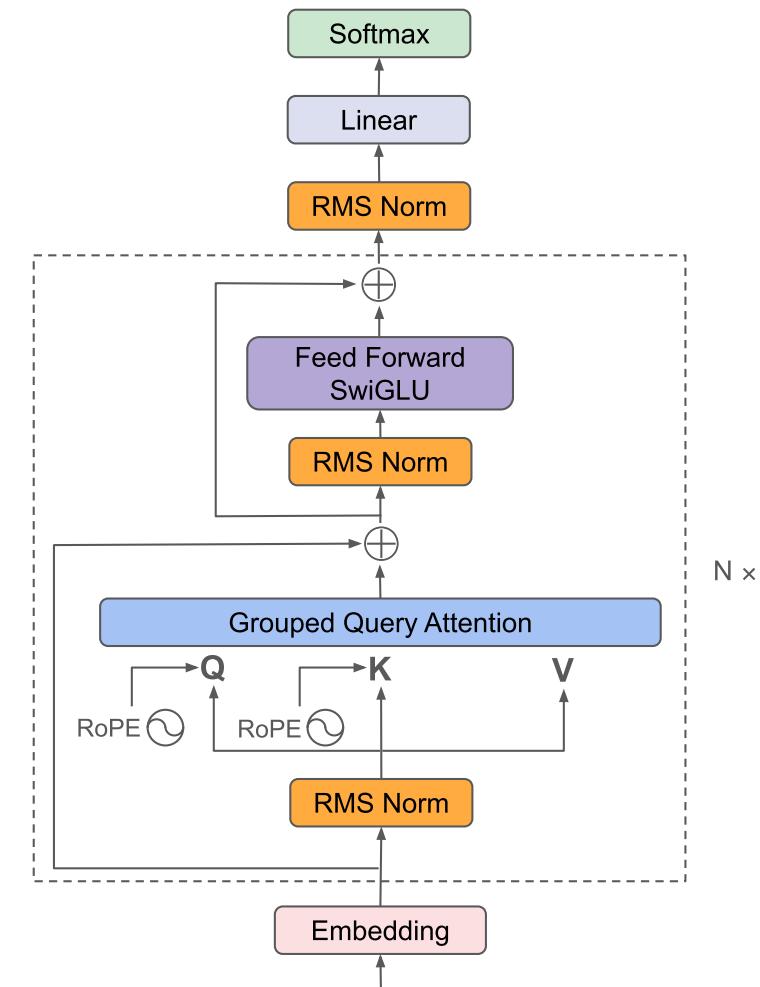
С 2017 модели сильно продвинулись вперед, они стали более стабильными и оптимизированными

Неизменным остались

- 👉 Механизм внимания для "замешивания" информации
- 👉 Блок нелинейности для "извлечения" и "хранения" информации



GPT

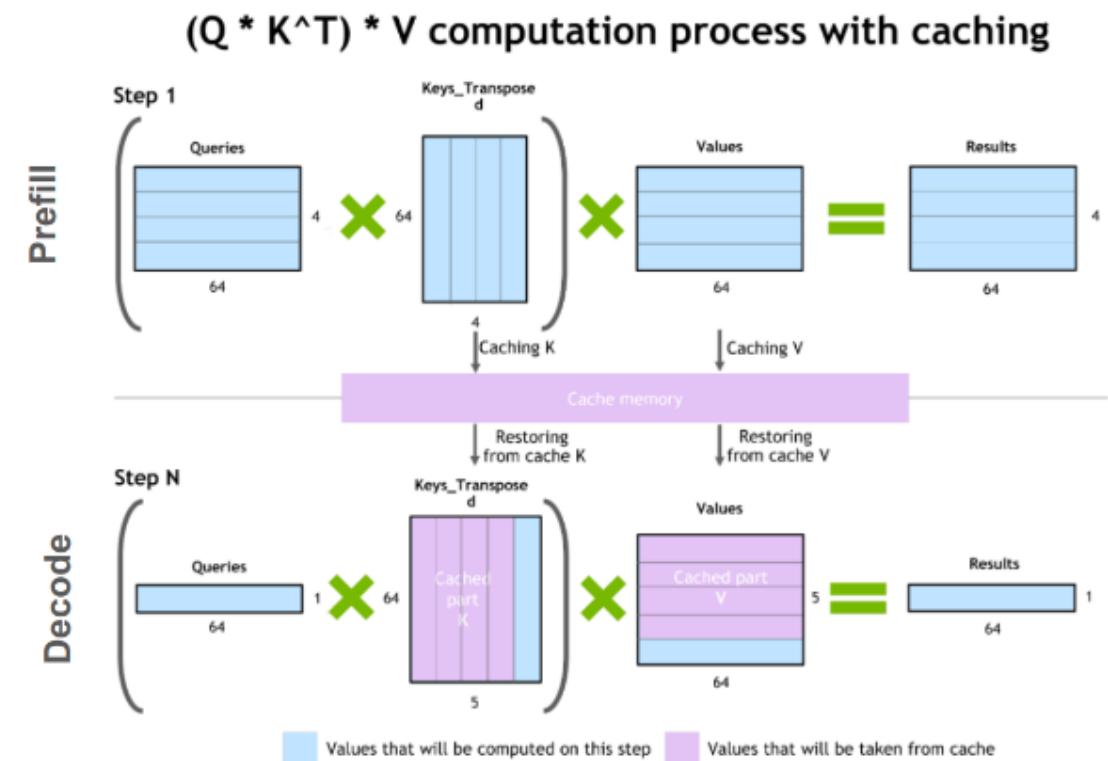


LLAMA-3

KV-caching

Поговорим немного про инференс — необходимо сгенерировать последовательность X , уже есть $x_{1\dots k}$ и надо предсказать x_{k+1}

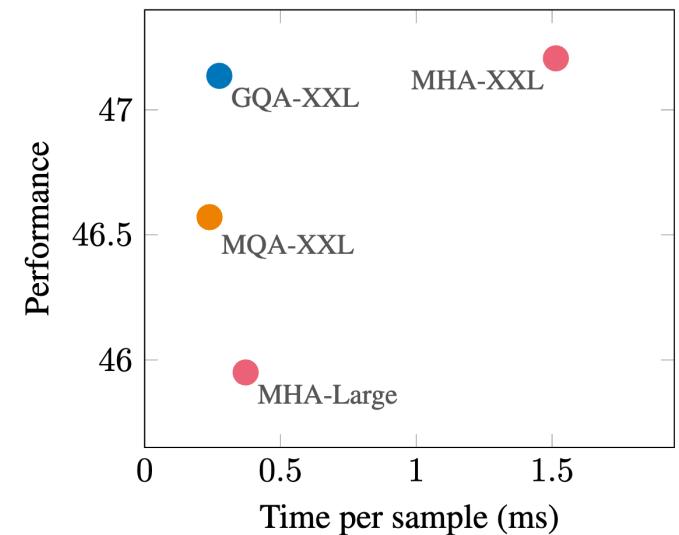
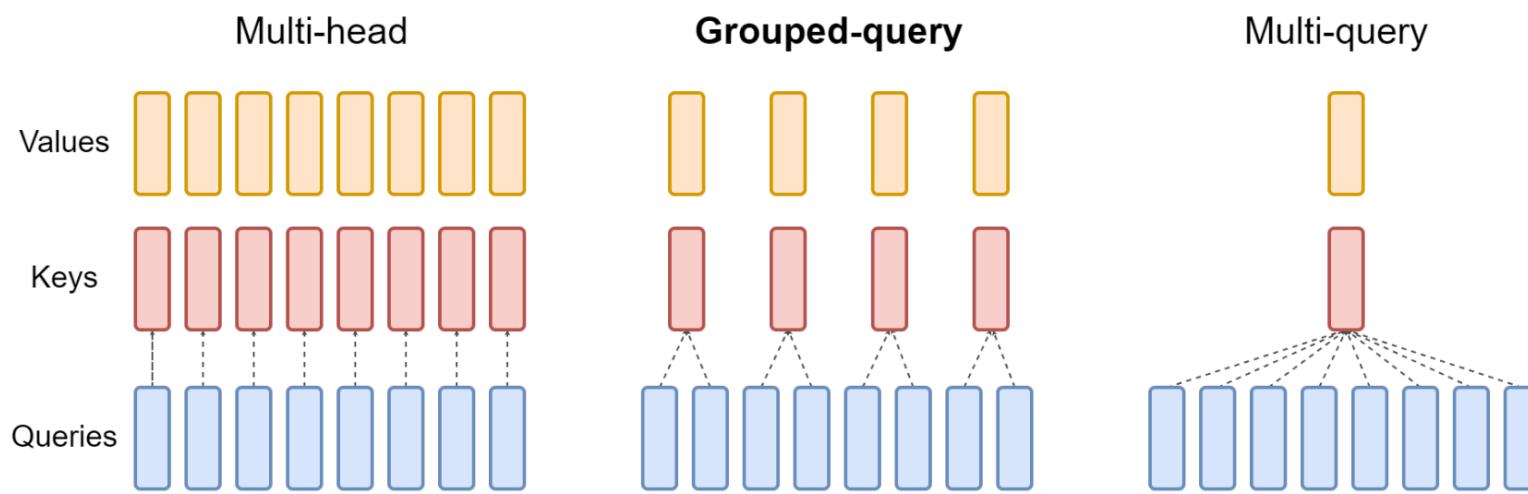
1. На шаге k внутри внимания посчитали QK^T
2. На шаге $k + 1$ появляется новый токен — $x_{k'}$, для него считаем q_{new}
3. Для q_{new} надо посчитать "связь" с ключами — $K' \in \mathbb{R}^{d \times (k+1)}$
4. Но $K' = [K; W_k x_k] \Rightarrow$ переиспользуем матрицу ключей, дописываем в нее столбец
5. Аналогично для матрицы значений V



Grouped Query Attention

KV-cache — дорогое удовольствие, требует $O(seq_len \cdot n_head \cdot head_dim)$ памяти

- 👉 MQA — используем всего одну голову, значительно выигрываем в скорости декодинга
- 👉 GQA — промежуточный вариант между МНА и МQA, баланс качества и памяти/скорости



[2] — Fast Transformer Decoding: One Write-Head is All You Need, Noam Shazeer, 2019

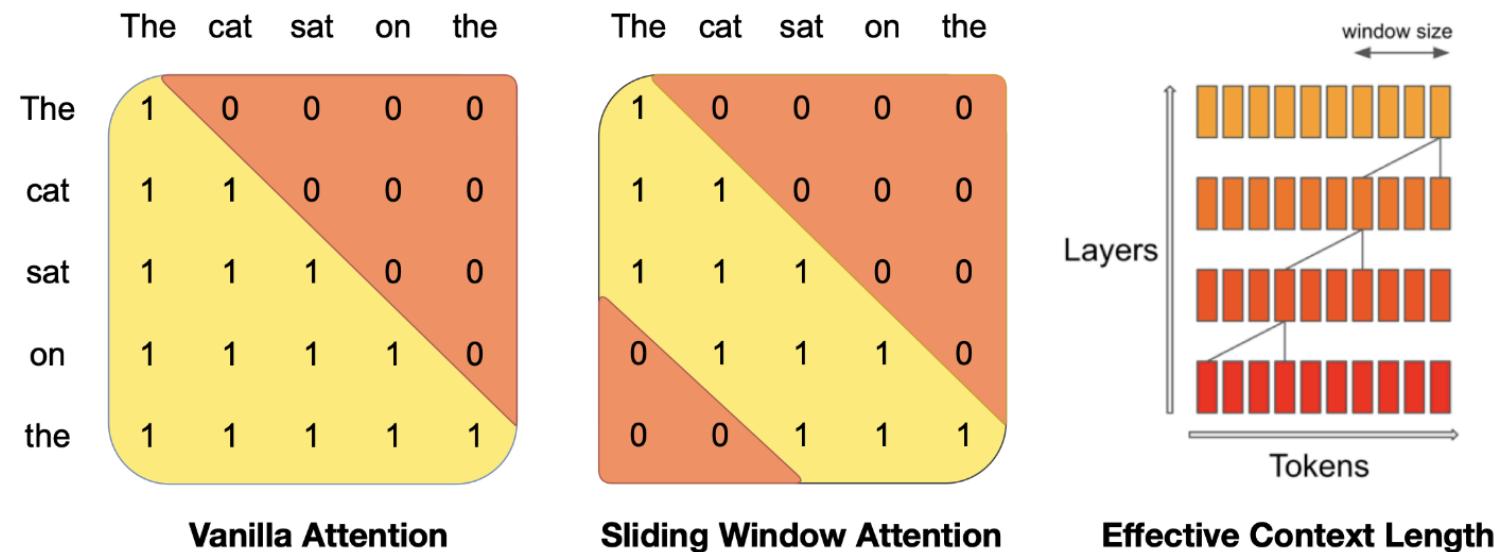
[3] — GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints, J. Ainslie, J. Lee-Thorp, M. De Jong et al., EMNLP'23

Sliding Window Attention

💡 Ограничиваляем seq_len в рамках одного слоя внимания.

SWA, Dilated SWA, Global + SWA, ... — разные способы брать токены из последовательности

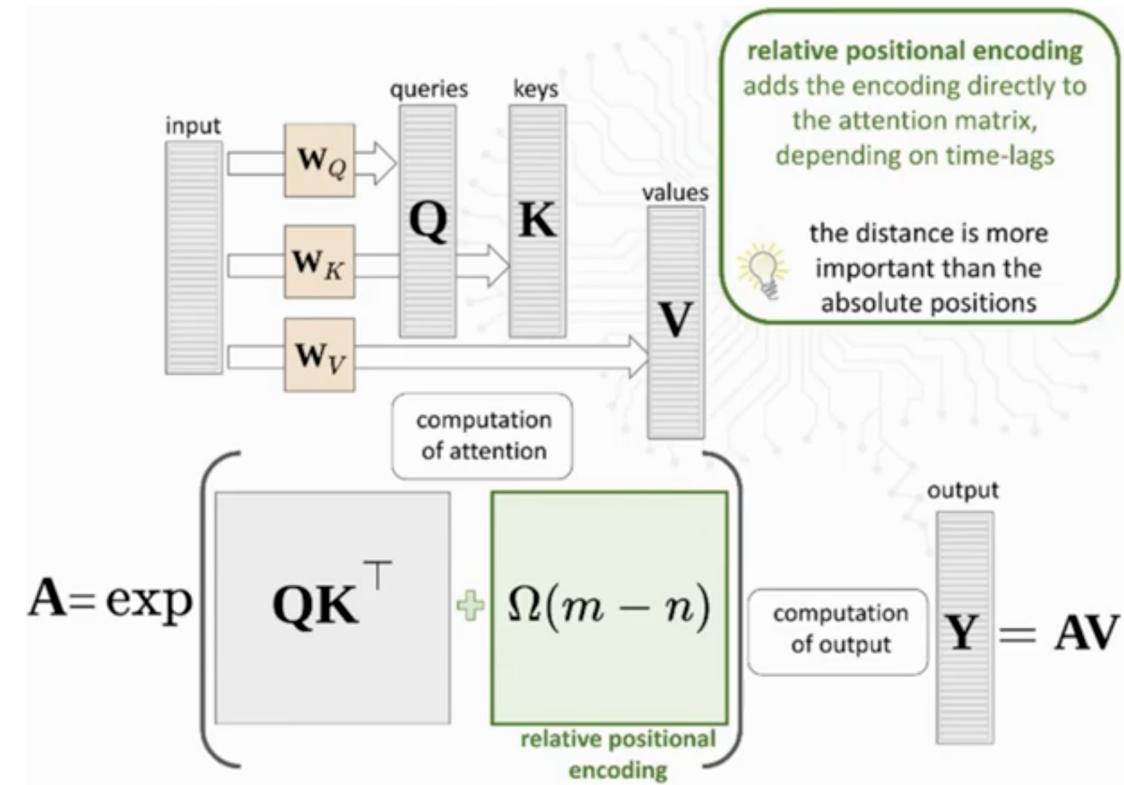
- 👉 Для окна W и k слоев, эффективный контекст $W \cdot k$
- 👉 В KV-cache храним значения последних W токенов
- 👉 Mistral-7B: для контекста в 16К и окна 4096 ускорение работы в 2 раза



Positional embeddings

Механизм внимания инвариантен к позициям токенам — QK^T хранит семантические связи, не учитывает где эти токены расположены в последовательности

1. Абсолютные позиционные эмбеддинги — добавляем информацию о позиции в эмбеддинг токена, механизм внимания остается без изменения
2. Относительные позиционные эмбеддинги — добавляем информацию о разнице позиций между токенов, работаем с QK^T



Rotary Positional Embedding

$\langle q_m; k_n \rangle$ — обеспечивает передачу знаний между токенами на позициях m и n

Чтобы начать учитывать относительную позицию между токенами x_m и x_n скалярное произведение q_m и k_n может проходить где-то внутри $g(x_m, x_n, m - n)$

 **RoPE:** хотим найти такие $f_q(x_m, m)$ и $f_k(x_n, n)$, чтобы

$$\langle f_q(x_m, m); f_k(x_n, n) \rangle = g(x_m, x_n, m - n)$$

RoPE: 2D case

Пусть $x \in \mathbb{R}^2$ и определим $f_{\{q,k\}}(x_m, m) = (W_{\{q,k\}} x_m) \cdot e^{im\theta}$ — поворот вектора на угол $m\theta$

Такой поворот можно записать в матричной форме

$$f_{\{q,k\}}(x_m, m) = R_{m\theta} W_{\{q,k\}} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

$$\text{где } R_{m\theta} = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$$

В таком случае $g(x_m, x_n, m - n) = \langle f_q(x_m, m); f_k(x_n, n) \rangle = \operatorname{Re} [(W_q x_m) \cdot (W_k x_n)^* \cdot e^{i(m-n)\theta}]$

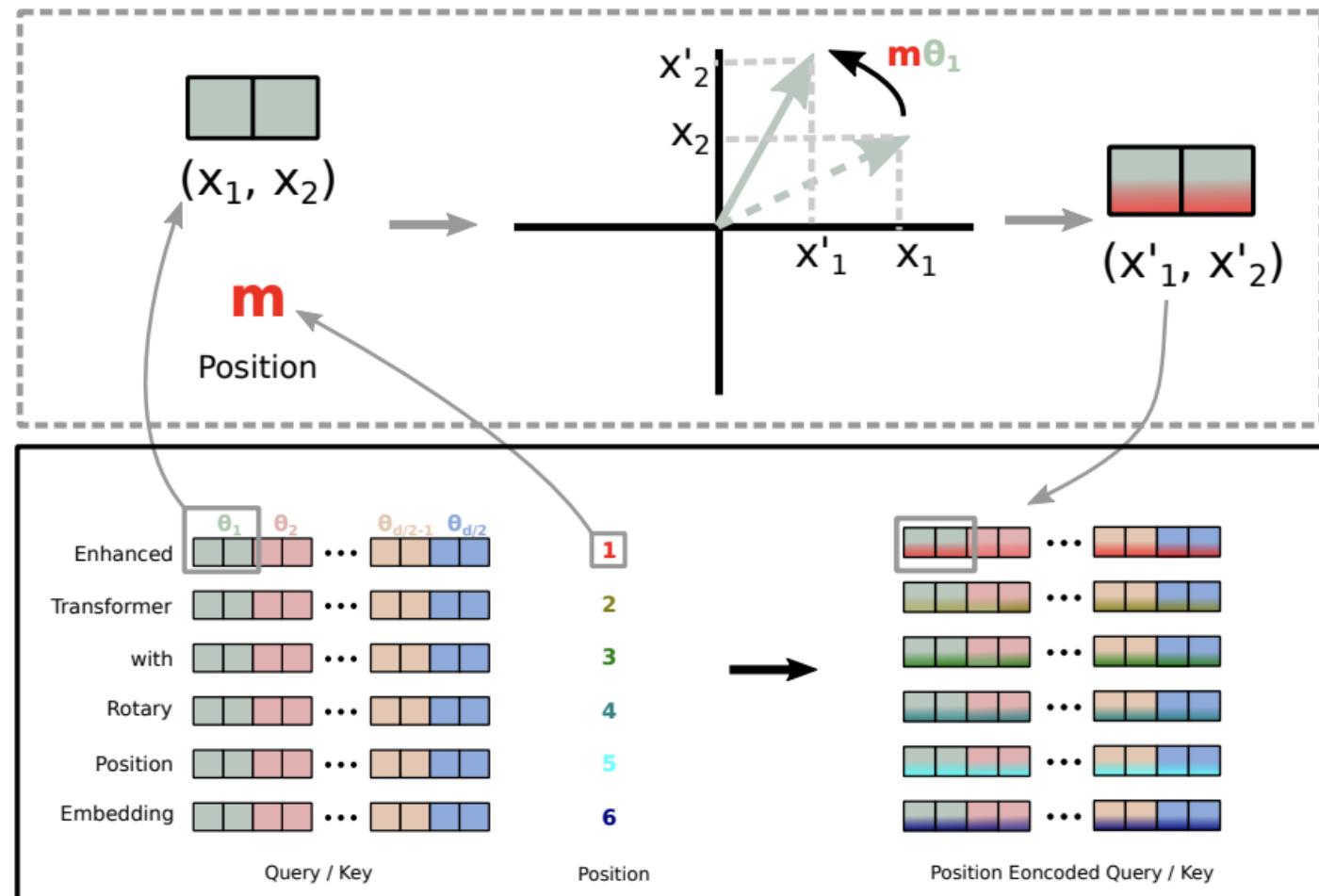
👉 $\operatorname{Re}[\dots]$ — вещественная часть, а $*$ — сопряженная матрица

👉 Кодируя абсолютные позиции, явно интегрируем относительные позиции

RoPE: общий случай

Для $x \in \mathbb{R}^d$ — делим на $\frac{d}{2}$ частей и применяем 2D подход с $\theta_1, \dots, \theta_{d/2}$

$\Theta = \{\theta_i = 10000^{-2(i-1)/d}\}$ — совсем как в оригинальном позиционном кодировании



RoPE: на практике

Инициализируем тензор константных Θ , никаких обучаемых параметров, в будущем на лету можно увеличивать число позиций

- 👉 Считаем Q и $K \rightarrow$ применяем RoPE — поворачиваем матрицы \rightarrow считаем МНА
- 👉 $R_{\Theta,m}^d \in \mathbb{R}^{d \times d}$ — блочно-диагональная матрица из $R_{m\theta_i}$
- 👉 Работает с любой реализацией МНА: `flash-attn`, `torch.sdp`, ...
- 👉 Работает и со многими видами линейного механизма внимания

$$R_{\Theta,m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

GPT-J, GPT-NeoX, XPos, ... — доработали идею RoPE до современного вида

ALiBi

Делаем позиционное кодирование с возможностью увеличить длину контекста на инференсе

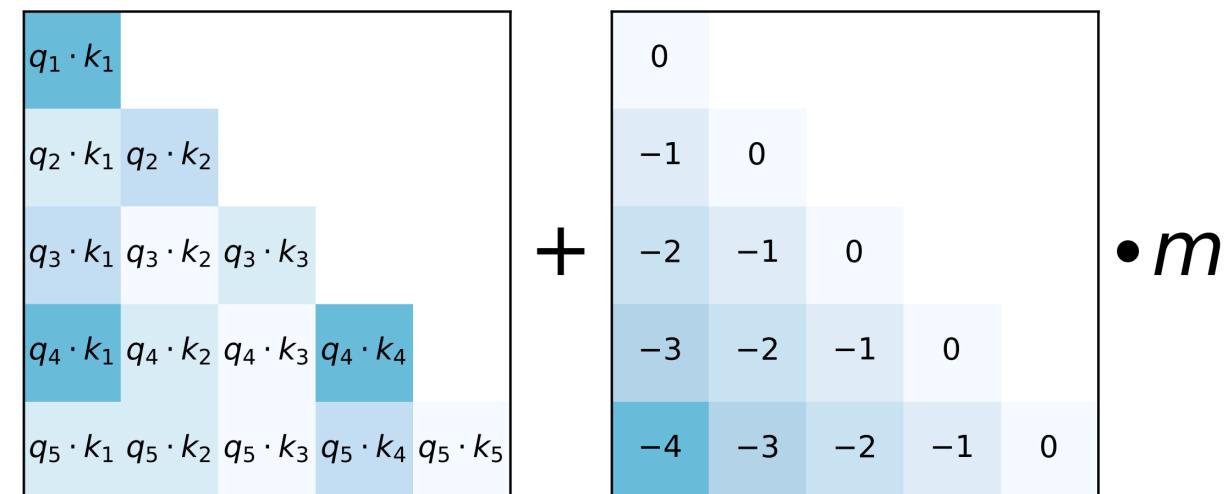
💡 **ALiBi (Attention with Linear Bias)** — добавляем статичный не-обучаемый баес

m — head-specific константа

👉 Для 8 голов: $1/2^1, \dots, 1/2^8$

👉 Для 16 голов интерполируем значения
 $1/2^{0.5}, 1/2^1, \dots, 1/2^{7.5}, 1/2^8$

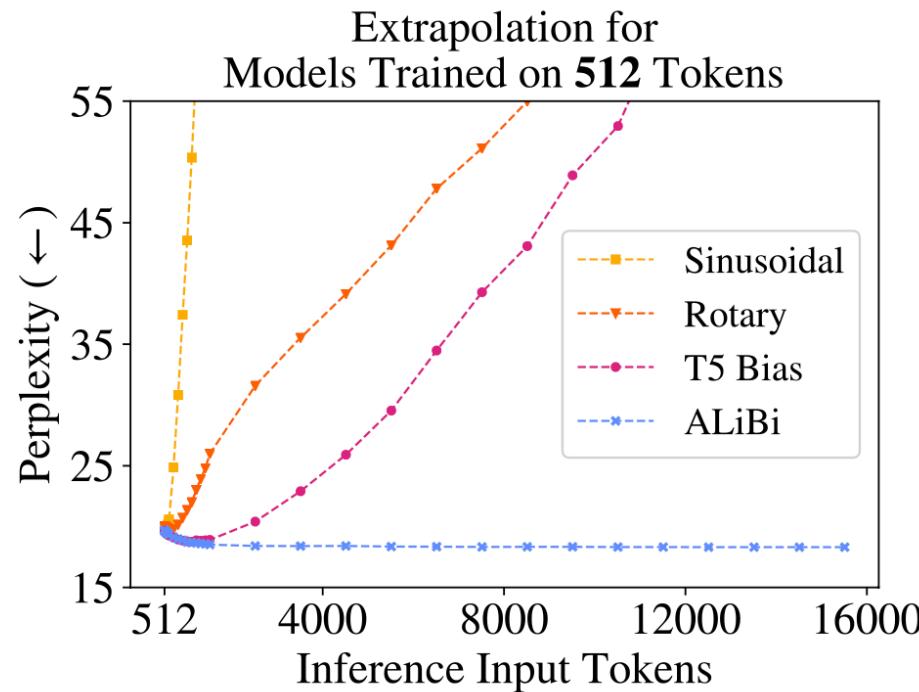
👉 Для n голов начинаем с $2^{-8/n}$ и
двигаемся с таким же шагом



Что выбрать?

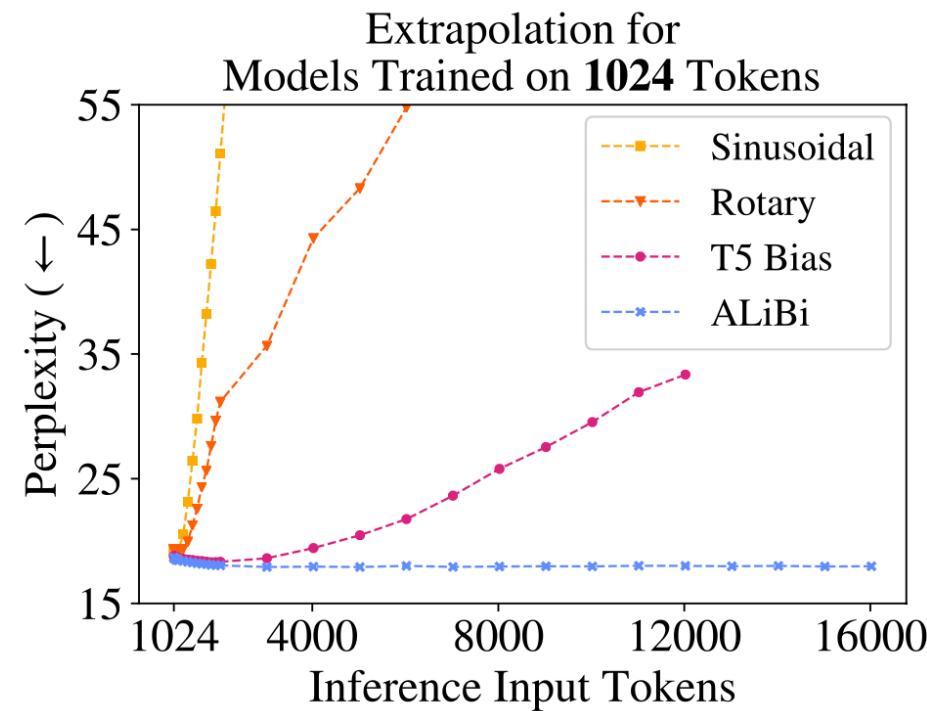
RoPE

- 👉 Интегрируется в любую реализацию МНА
- 👉 LLaMa, Gemma, Phi, Nemotron, ...

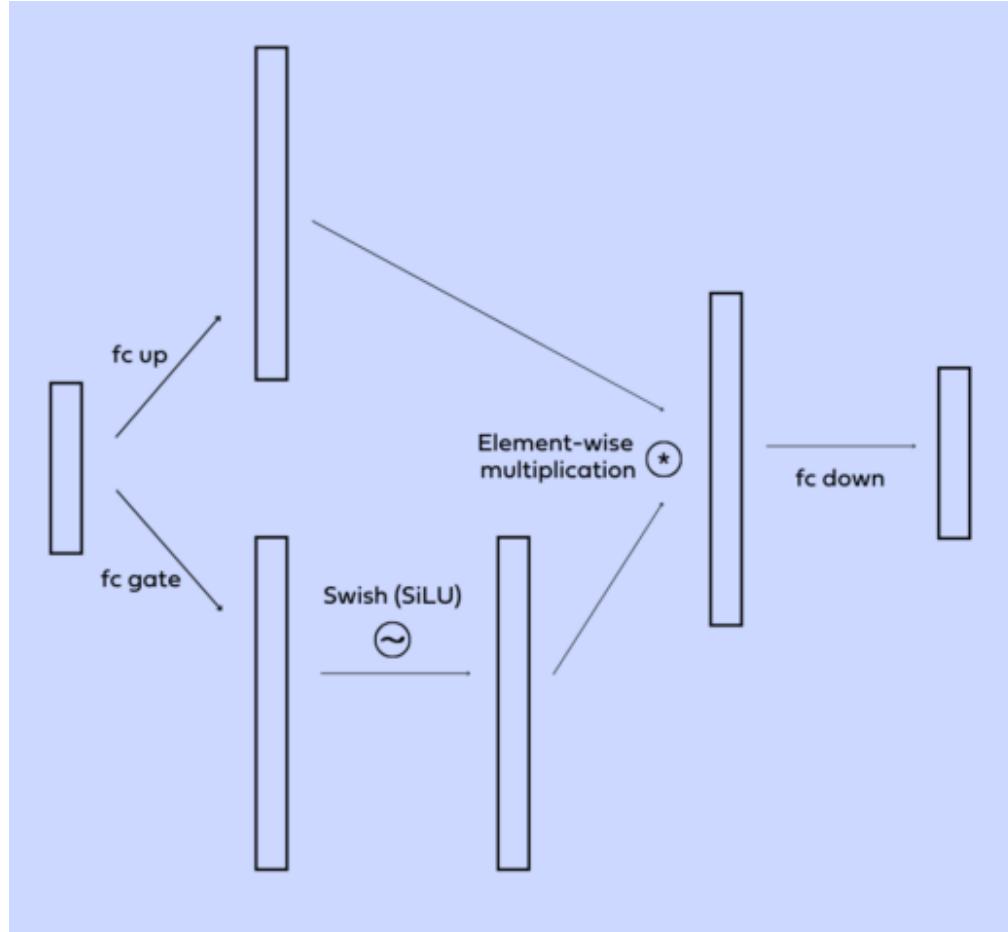


ALiBi

- 👉 Требуют внедрения в МНА, flash-attention поддержал только в 2.4 (Dec'23)
- 👉 Легче использовать на новых длинах
- 👉 Bloom



SwiGLU



Стандартный FFN: $y = W_2(\text{Act}(W_1x + b_1)) + b_2$

👉 В качестве активации ReLU, GeLU, Swish, ...

👉 Как правило, $d_{\text{model}} \rightarrow 4 \cdot d_{\text{model}} \rightarrow d_{\text{model}}$

SwiGLU (Swish + Gated Linear Unit):

$$u = W_2x + b_2$$

$$g = \text{Swish}(W_1x + b_1)$$

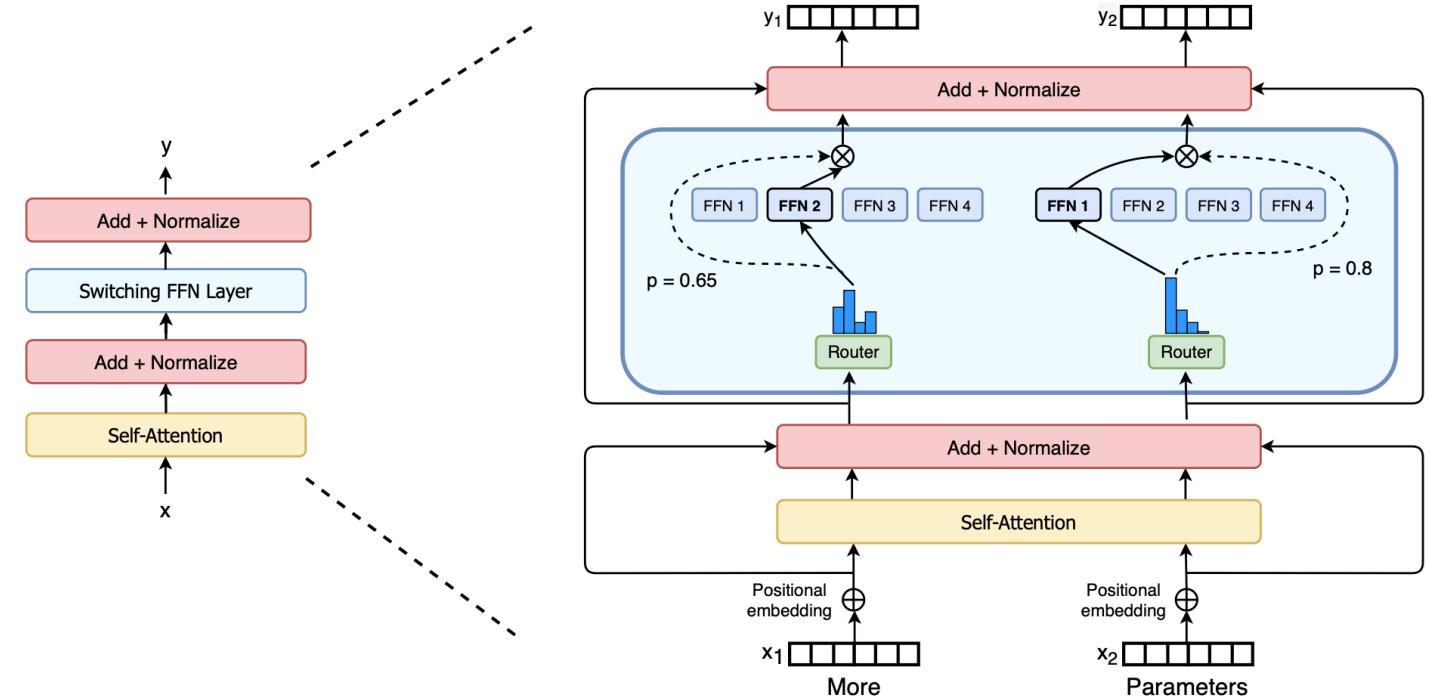
$$y = W_3(u \times g) + b_3$$

Изначально $8/3 \cdot d_{\text{model}}$ для баланса, но сейчас везде по-разному. Важно поддерживать кратность 64 или 128!

Mixture of Experts

MoE — sparse-слой, для каждого токена выбираются k из n экспертов. Позволяет эффективно наращивать число параметров — модель “знает” больше, но в моменте пользуется не всем

- 👉 Mixtral: 8 экспертов по 2 на каждый токен, из 47 млрд задействуется только 13
- 👉 Эксперт — любой FFN блок
- 👉 Роутинг — обучаемый слой для распределения токенов по экспертам



Собираем
модели



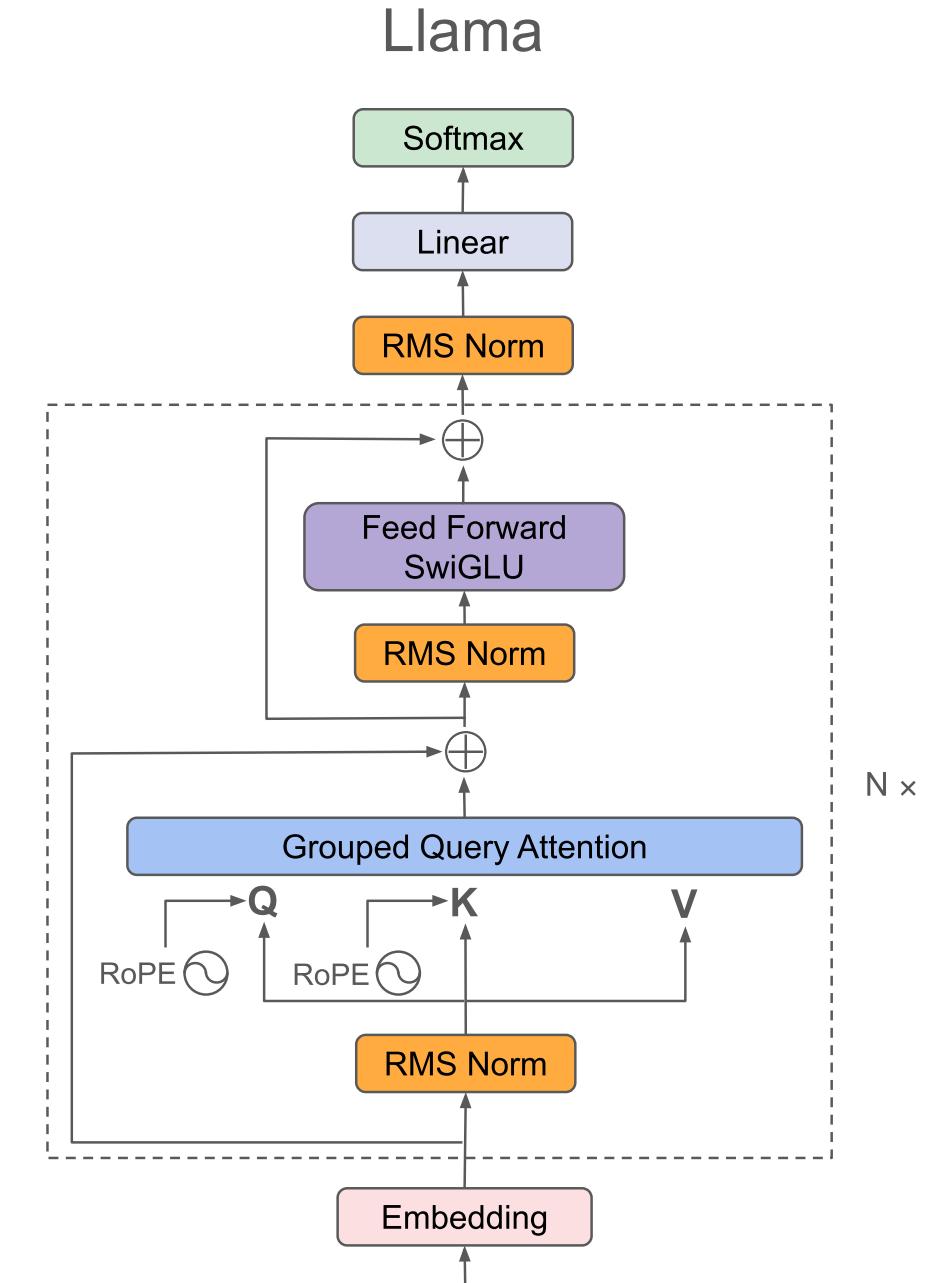
LLaMA-3

RMS Norm vs Layer Norm:

$$\bar{a}_i = \frac{a_i}{\text{RMS}(a)} g_i, \text{ где } \text{RMS}(a) = \sqrt{\frac{1}{n} \sum_{i=1}^n a_i^2}$$

Отказались от центрирования, но зато ускорили вычисления

- 👉 Линейные слои без использования bias
- 👉 Отсутствие dropout во время предобучения
- 👉 Fused-ядра для прямого и обратного прохода
- 👉 И еще много трюков для стабильного обучения



[12] — Root Mean Square Layer Normalization, Zhang et al., NeurIPS'19

[13] — Llama-3, Meta, 2024

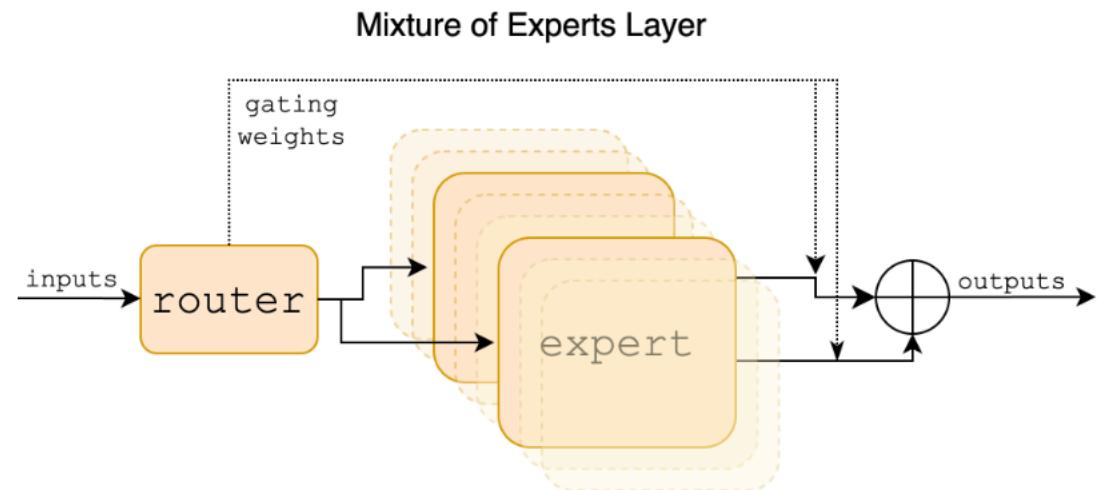
Mistral & Mixtral

Mistral 7B:

- 👉 GQA: 32 головы для query и по 8 для key и value
- 👉 SWA: 32 слоя и окно 4096
- 👉 RMSNorm, RoPE, SwiGLU

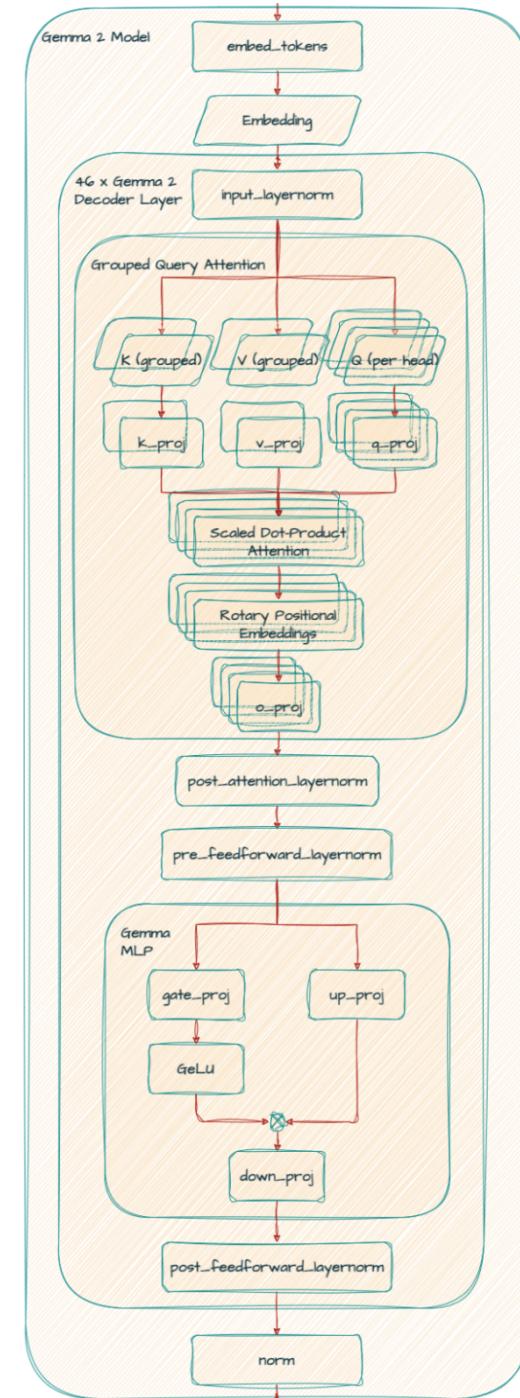
Mixtral 8x7B:

- 👉 MoE: 8 экспертов, каждый SwiGLU
- 👉 Для каждого токена выбирается 2 эксперта



Gemma-2

- 👉 RMSNorm, SwiGLU, RoPE, GQA, ...
- 👉 Немного другая работа с half-precision — хаки для стабильности, формально не отличается
- 👉 Local and Global Attention — SWA на каждом втором слое
- 👉 Logit Soft-Capping — запрещаем быть слишком уверенными в связи двух токенов и при предсказании следующего
Ограничиваю значения QK^T между $-\sigma$ и σ
$$QK^T \leftarrow \sigma \cdot \tanh\left(\frac{QK^T}{\sigma}\right)$$
- 👉 Вставляем нормализации перед и после FFN



Картиночка не самая точная, но официальная

Итоги



TL;DR

Спустя 7 лет появилось множество модификаций Transformer для более стабильного обучения, быстрого инференса и лучшего качества

Наиболее популярные модификации включают SwiGLU, RoPE, GQA

Но архитектурных модификаций недостаточно, смотрите в следующих сериях:

- 👉 Для больших моделей нужно много данных
- 👉 Подходы к обучению, подготовка моделей для диалогов
- 👉 Большая модель требует много ресурсов, как их эффективно утилизировать
- 👉 Большим моделям нужен быстрый инференс
- 👉 ...

На этом все



Егор Спирин — vk.com/boss
VK Lab — vk.com/lab