



# Лекция 7. Подходы к оптимизации обучения и инференса моделей

Макаренко Владимир



# О чём сегодня поговорим

Зачем оптимизировать обучение и инференс?

Распределенное обучение и инференс

Квантизация модели

Parameter-efficient fine-tuning (PEFT)

Прочие методы оптимизации обучения и инференса



# Зачем оптимизировать обучение и инференс?



## Типичные проблемы при работе с LLM

- Для обучения больших моделей не хватает памяти.
- Обновление параметров занимает слишком много времени.
- Модели занимают слишком много места на диске.
- Скорость применения больших моделей оставляет желать лучшего.

## Что можно оптимизировать?

- Потребление памяти
- Скорость обучения
- Скорость применения

## На что тратится память во время применения

Во время применения:

- Параметры модели
- Активации

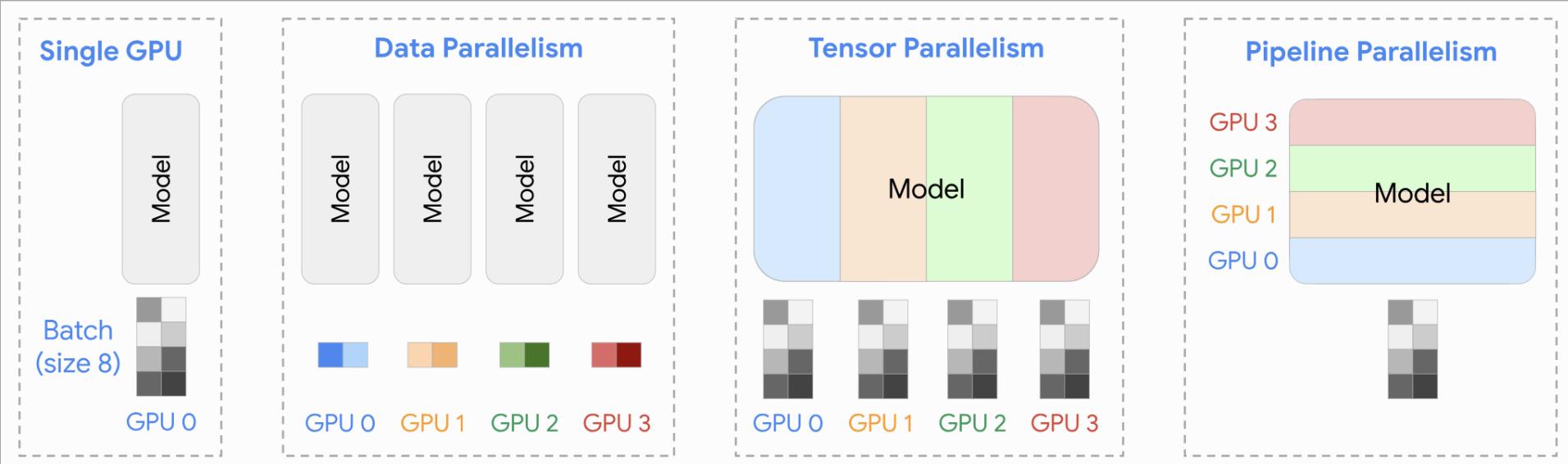
Во время обучения:

- Параметры модели
- Состояния оптимизатора
- Градиенты
- Активации

# Распределенное обучение и инференс



# Распределенное обучение



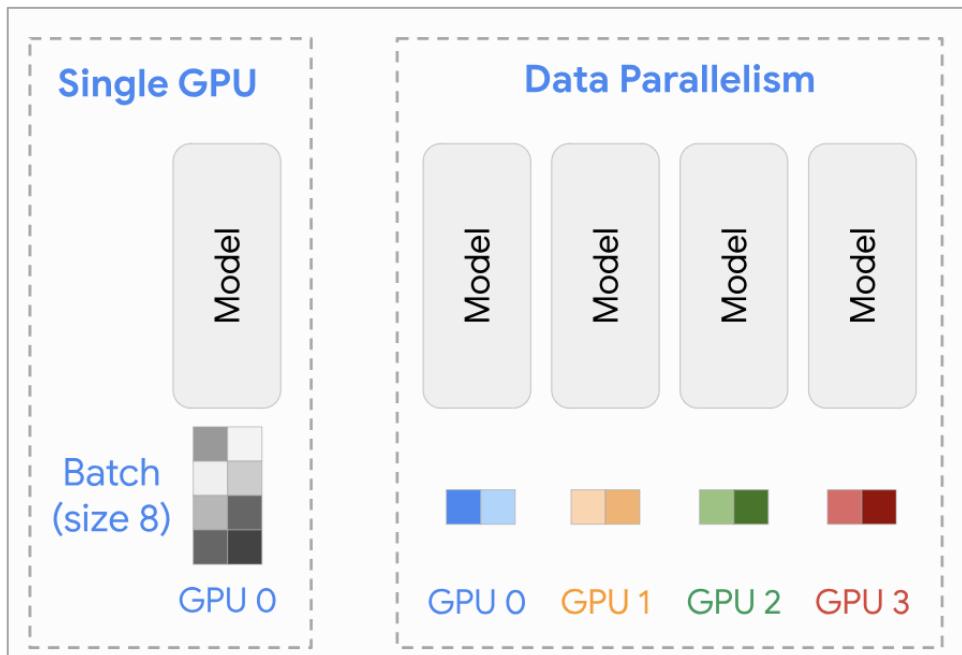
[https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial\\_notebooks/scaling/JAX/pipeline\\_parallel\\_simple.html](https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/scaling/JAX/pipeline_parallel_simple.html)

# Data parallelism

Сначала загружаем копию модели на каждую из GPU.

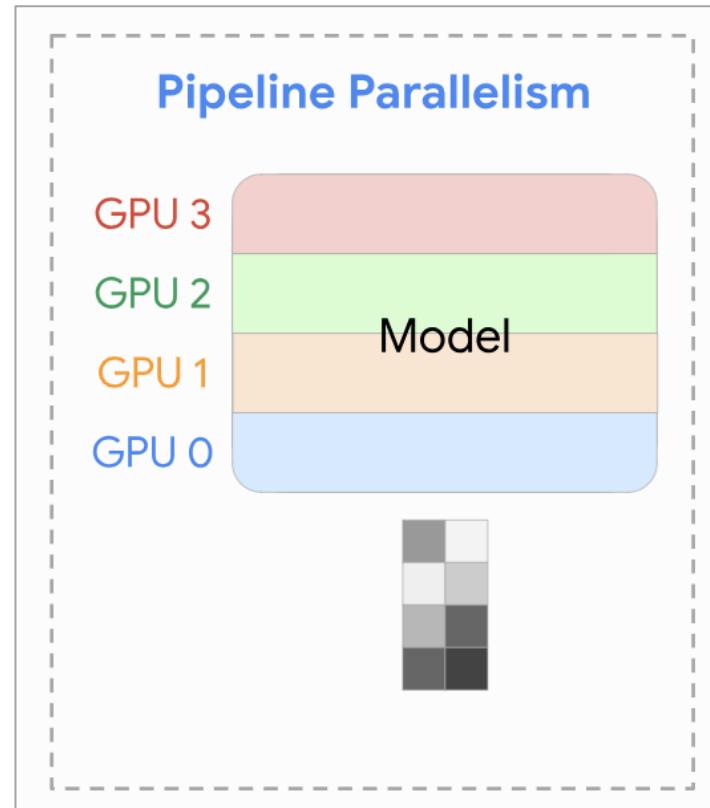
Одна итерация обновления весов:

- Каждая GPU принимает свой кусок данных (раскидываем batch равномерно по всем GPU)
- Каждая GPU считает градиенты функции потерь по параметрам
- Все GPU обмениваются градиентами друг с другом, после чего обновляют веса (одинаково)



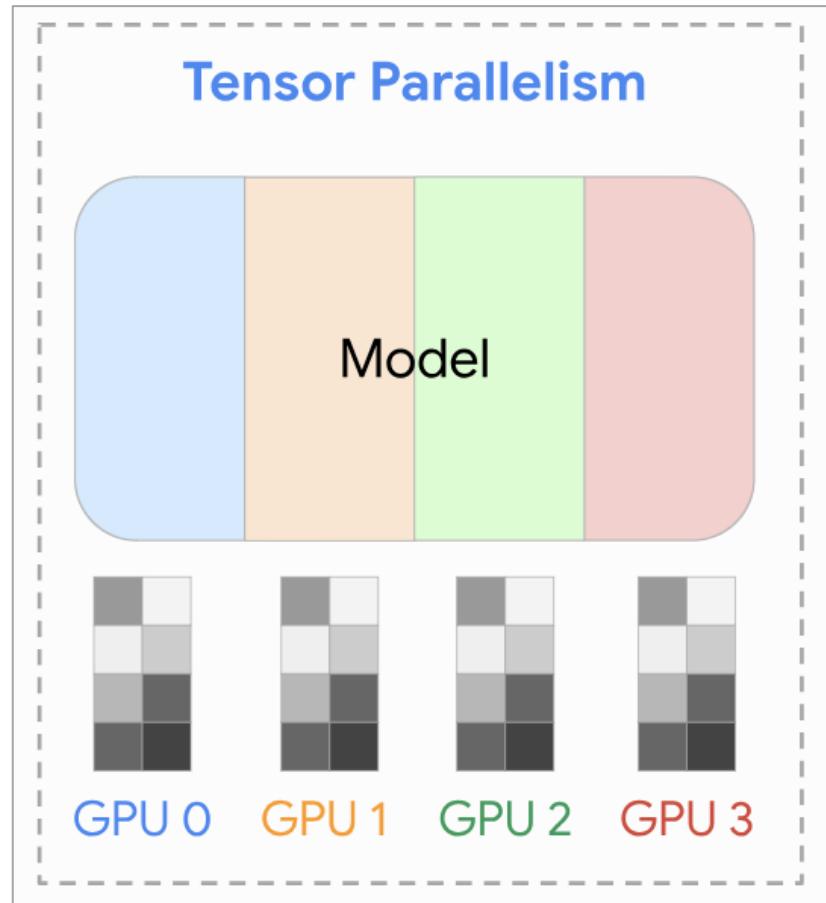
# Pipeline parallelism

Каждый слой модели  
(+соответствующие градиенты и  
состояния оптимизатора) разносим  
по разным GPU

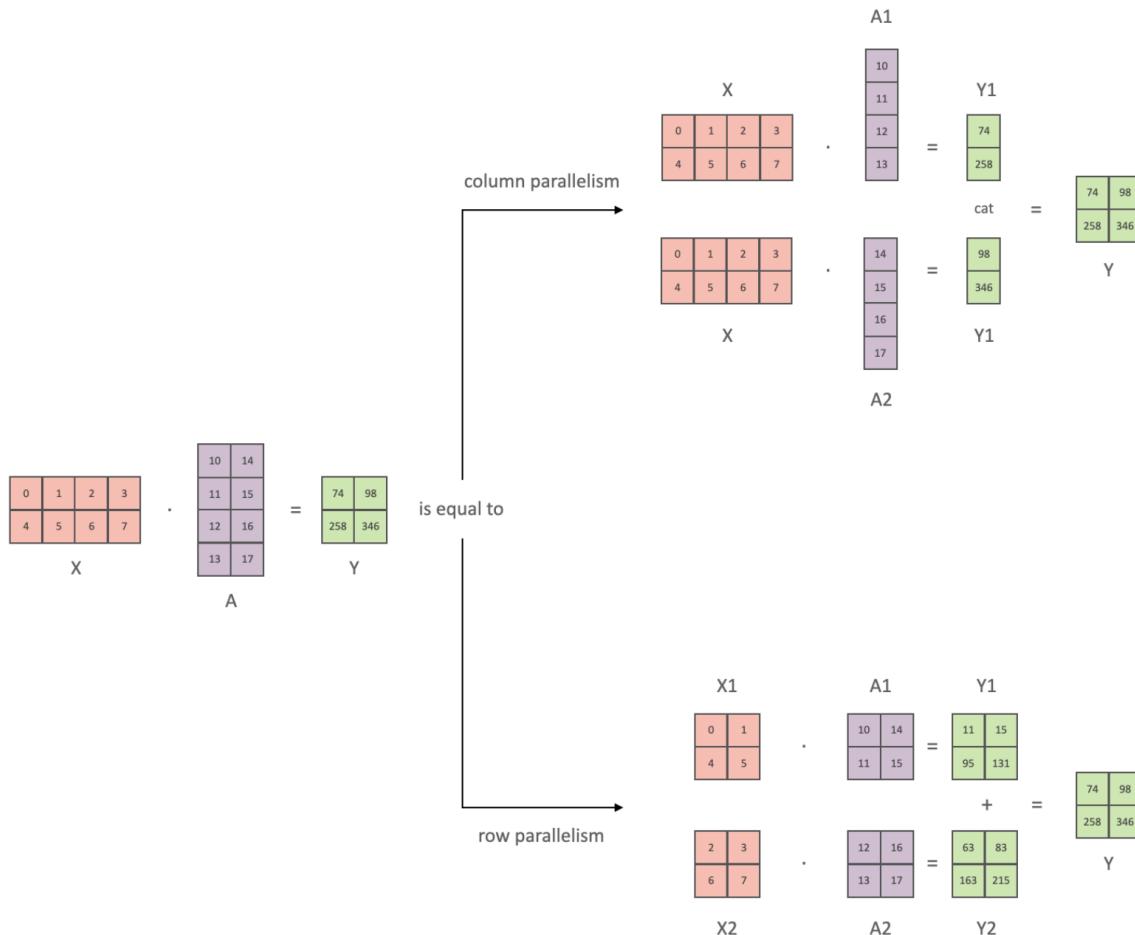


# Tensor parallelism

- Каждый тензор модели (матрица весов) и соответствующие градиенты и состояния оптимизатора разделяем между разными GPU
- Обмениваемся информацией между GPU, когда нужно



# Tensor parallelism





Deepspeed — библиотека от Microsoft с открытым исходным кодом, совместимая с PyTorch.

Позволяет:

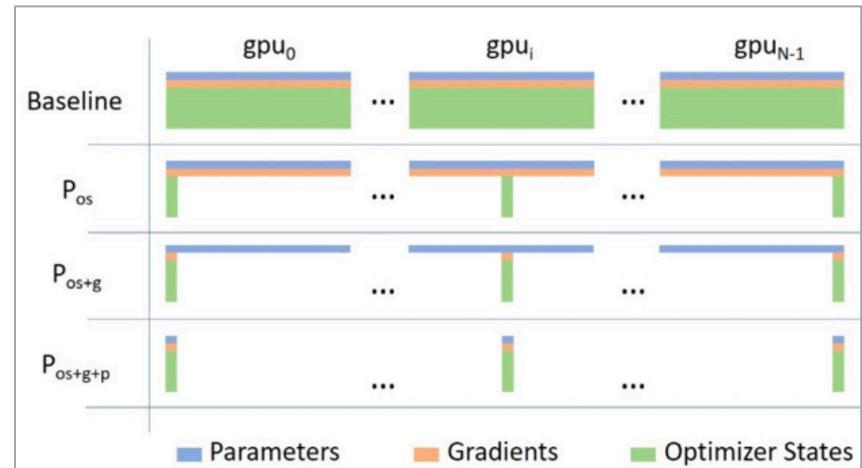
- Снизить расход памяти при обучении и применении;
- Ускорить обучение и применение.

В основе лежит технология, которая называется ZeRO (Zero Redundancy Optimizer). Она сокращает потребление памяти путем разделения состояний оптимизатора, градиентов и параметров модели между параллельными процессами вместо репликации.

# ZeRO (Zero Redundancy Optimizer)

Трехуровневая оптимизация:

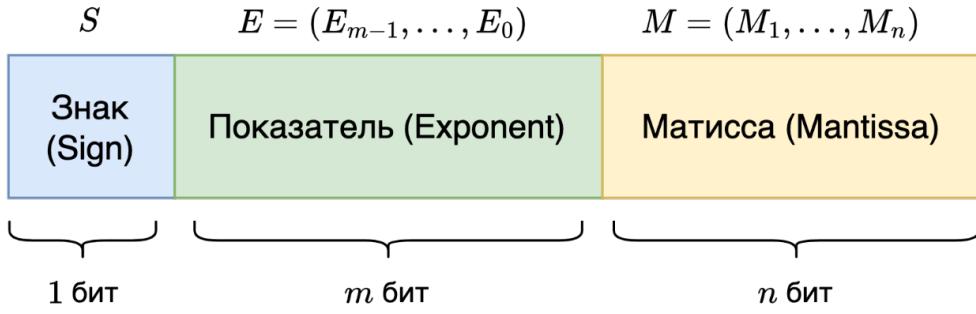
- Optimizer State Partitioning (OS).  
Сокращает используемую память в 4 раза.
- + Gradient Partitioning (OS + G).  
Сокращает используемую память в 8 раз.
- + Parameter Partitioning (OS + G + P).  
Уменьшает использование памяти пропорционально количеству ГПУ.



# Квантизация модели



## Представление числа с плавающей точкой (IEEE 754)



## Пример:

$$S = 0,$$

$$\tilde{E} = 127 - (128 - 1) = 0,$$

$$\widetilde{M} = \frac{1}{2},$$

$$x = (-1)^0 \cdot 2^0 \cdot \left(1 + \frac{1}{2}\right) = 1.5$$

**Обозначения:**  $E_k, M_k, S \in \{0, 1\}$

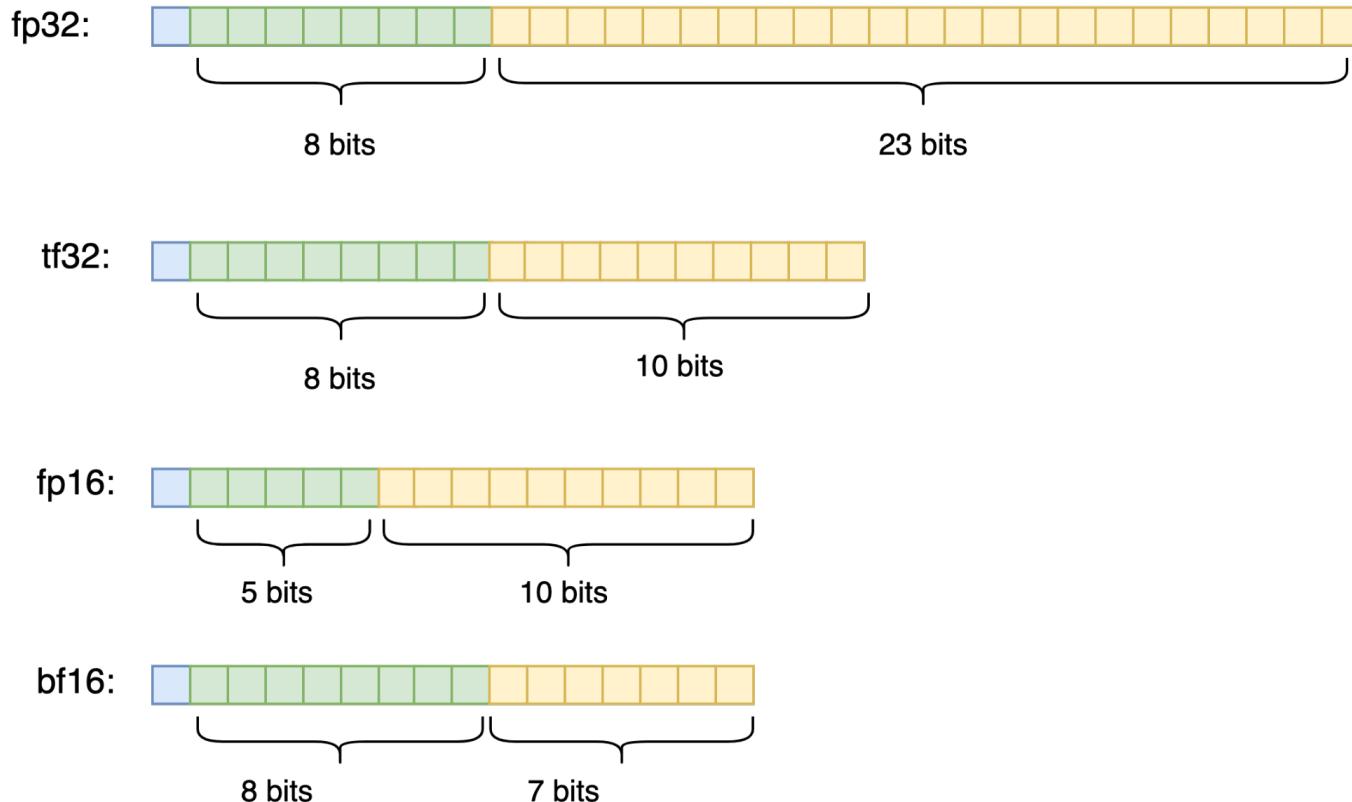
## Представление числа:

$$x = (-1)^S \cdot 2^{\widetilde{E}} \cdot (1 + \widetilde{M}),$$

$$\tilde{E} = \sum_{j=0}^{m-1} 2^j E_j - (2^{m-1} - 1),$$

$$\widetilde{M} = \sum_{j=1}^n \frac{M_j}{2^j}$$

# Числовые форматы



# Простейшая 8-битная квантизация (absmax)

**Цель:** масштабированием отобразить значения матрицы  $X$  в промежуток  $[-127, 127]$

**Уравнение:**

$$s \cdot \max_{i,j} |X_{i,j}| = 127, \text{ откуда } s = \frac{127}{\max_{i,j} |X_{i,j}|}$$

**Квантизация:**

$$X_q = \text{round}(s \cdot X)$$

**Деквантизация:**

$$X_{dq} = \frac{X_q}{s}$$

**Пример:**

$$X = \begin{pmatrix} 1.5 & 0 & -1.1 \\ 2 & 0.5 & -0.1 \\ 1.2 & 1.7 & -0.5 \end{pmatrix}, \quad X_q = \begin{pmatrix} 95 & 0 & -70 \\ 127 & -2 & 36 \\ 76 & 108 & -32 \end{pmatrix}, \quad X_{dq} = \begin{pmatrix} 1.496 & 0 & -1.102 \\ 2 & 0.504 & -0.094 \\ 1.197 & 1.701 & -0.504 \end{pmatrix}$$

# Простейшая 8-битная квантизация (zeropoint)

**Цель:** линейно отобразить значения матрицы  $X$  в промежуток  $[-128, 127]$

**Уравнения:**

$$\begin{cases} s \min_{i,j} X_{i,j} + z = -128, \\ s \max_{i,j} X_{i,j} + z = 127 \end{cases} \iff \begin{cases} s = \frac{255}{\max_{i,j} X_{i,j} - \min_{i,j} X_{i,j}}, \\ z = -s \min_{i,j} X_{i,j} - 128 \end{cases}$$

**Квантизация:**

$$X_q = \text{round}(s \cdot X + \tilde{z}), \quad \tilde{z} = \text{round}(z)$$

**Деквантизация:**

$$X_{dq} = \frac{X_q - \tilde{z}}{s}, \quad \tilde{z} = \text{round}(z)$$

**Пример:**

$$X = \begin{pmatrix} 1.5 & 0 & -1.1 \\ 2 & 0.5 & -0.1 \\ 1.2 & 1.7 & -0.5 \end{pmatrix}, \quad X_q = \begin{pmatrix} 85 & -38 & -128 \\ 127 & -3 & -46 \\ 61 & 102 & -79 \end{pmatrix}, \quad X_{dq} = \begin{pmatrix} 1.495 & 0 & -1.094 \\ 2.006 & 0.498 & -0.097 \\ 1.204 & 1.702 & -0.498 \end{pmatrix}$$

## 8-битное умножение матриц (absmax)

Пусть  $X \in \mathbb{R}^{m \times p}$ ,  $W \in \mathbb{R}^{p \times n}$ .

Имеем

$$\begin{aligned}(X_{\text{dq}} W_{\text{dq}})_{i,j} &= \sum_{k=1}^p (X_{\text{dq}})_{i,k} (W_{\text{dq}})_{k,j} = \sum_{k=1}^p \frac{(X_{\text{q}})_{i,k} (W_{\text{q}})_{k,j}}{s_x s_w} = \\ &= \frac{1}{s_x s_w} \sum_{k=1}^p (X_{\text{dq}})_{i,k} (W_{\text{dq}})_{k,j},\end{aligned}$$

то есть  $X_{\text{dq}} W_{\text{dq}} = \frac{1}{s_x s_w} \cdot X_{\text{q}} W_{\text{q}}$ .

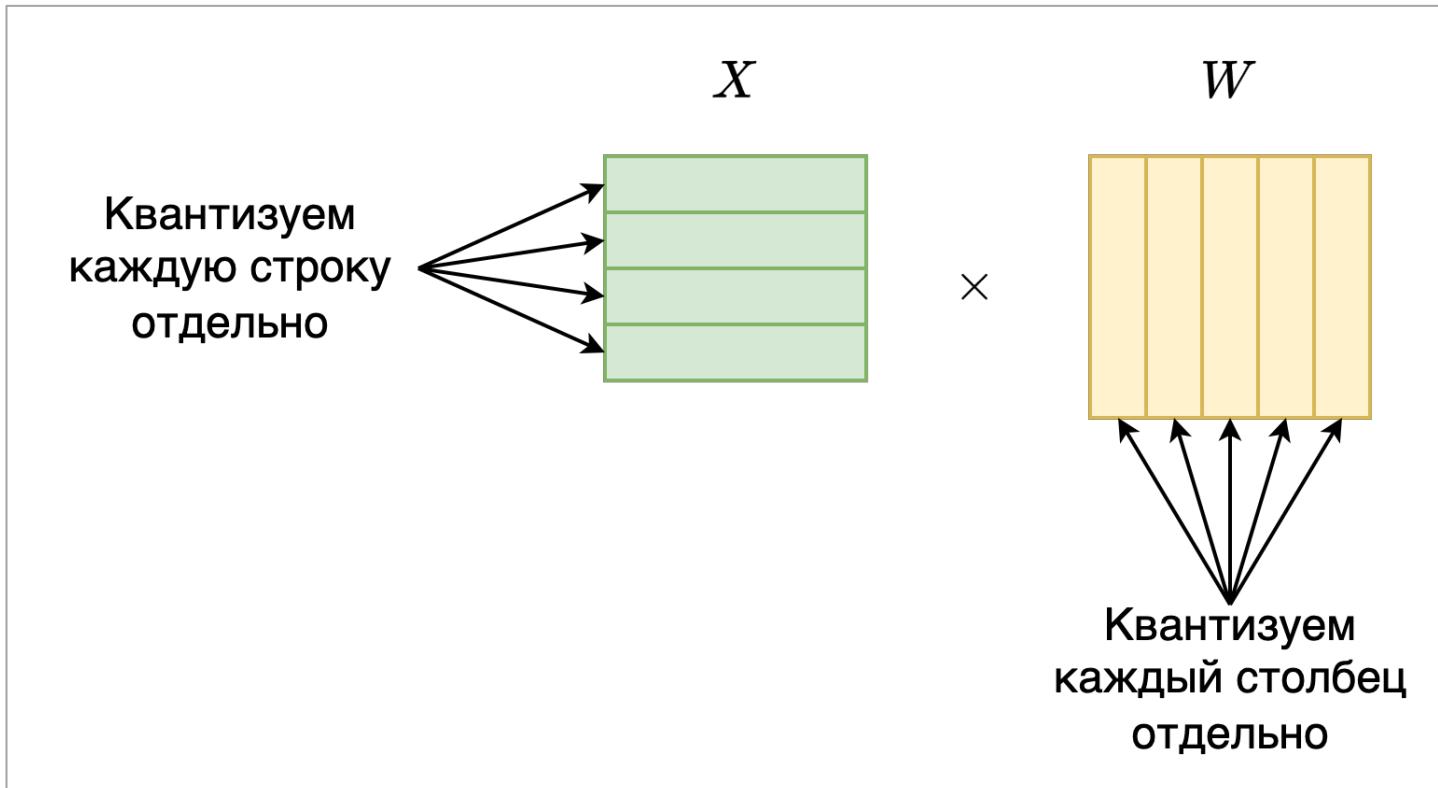
## 8-битное умножение матриц (zeropoint)

Пусть  $X \in \mathbb{R}^{m \times p}$ ,  $W \in \mathbb{R}^{p \times n}$ .

Имеем

$$\begin{aligned}(X_{\text{dq}} W_{\text{dq}})_{i,j} &= \sum_{k=1}^p (X_{\text{dq}})_{i,k} (W_{\text{dq}})_{k,j} = \\&= \sum_{k=1}^p \frac{(X_{\text{q}})_{i,k} - \tilde{z}_x}{s_x} \cdot \frac{(W_{\text{q}})_{k,j} - \tilde{z}_w}{s_w} = \\&= \frac{1}{s_x s_w} \left[ \sum_{k=1}^p (X_{\text{dq}})_{i,k} (W_{\text{dq}})_{k,j} - \tilde{z}_w \sum_{k=1}^p (X_{\text{q}})_{i,k} - \tilde{z}_x \sum_{k=1}^p (W_{\text{q}})_{k,j} + p \tilde{z}_x \tilde{z}_w \right]\end{aligned}$$

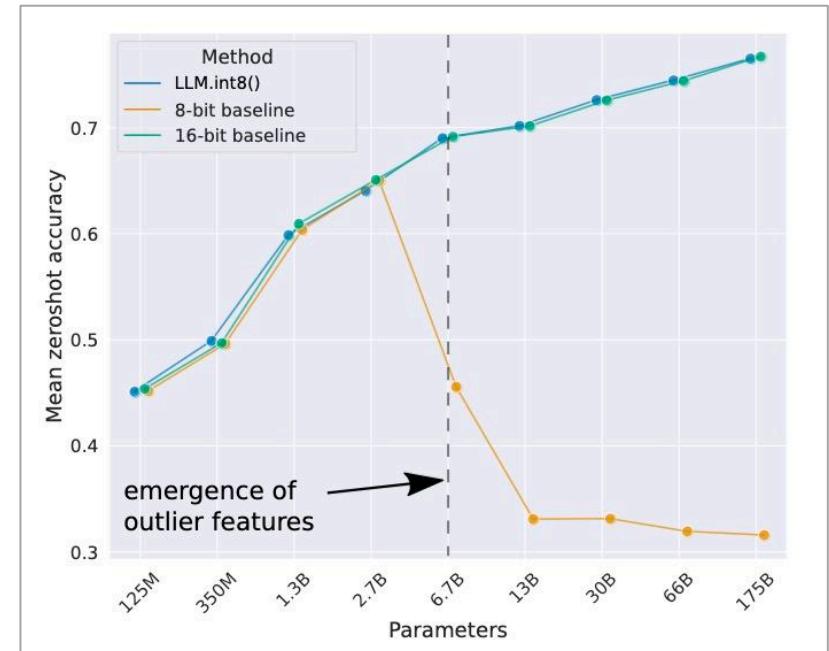
## Vector-wise quantization



# Проблема выбросов

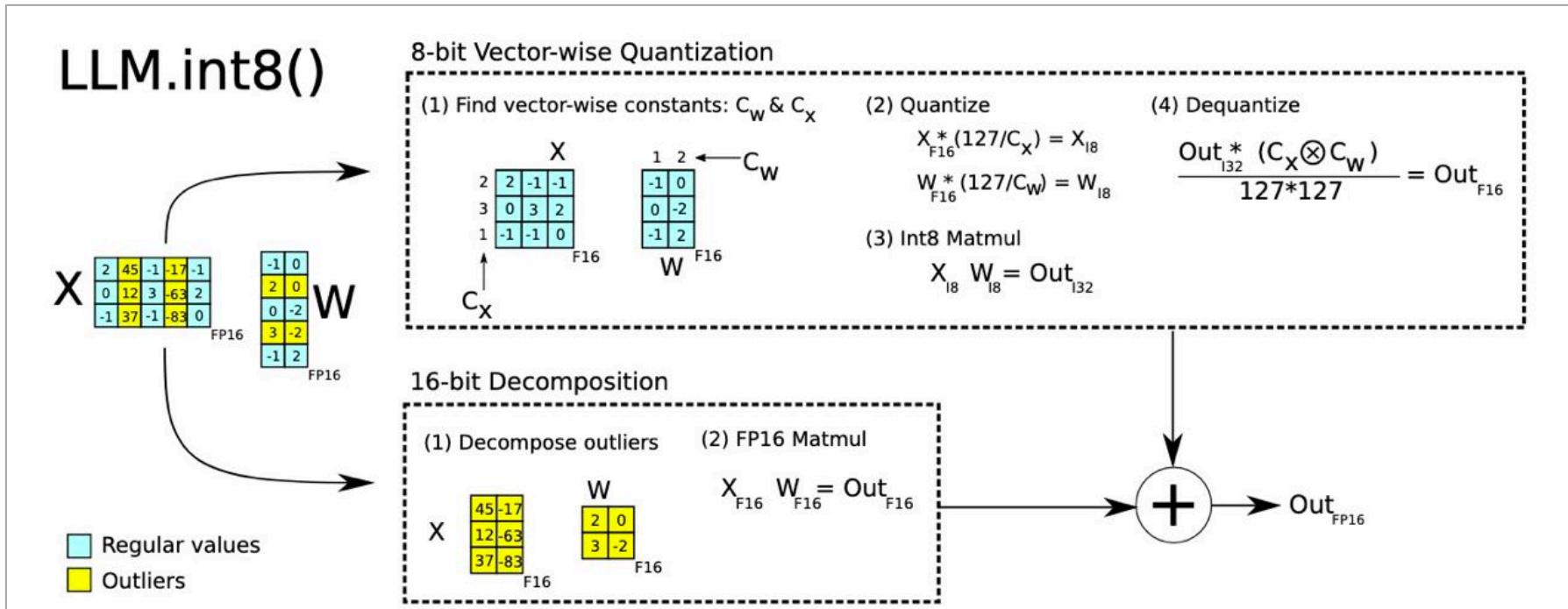
Чем больше параметров у модели, тем больше возникает больших значений во входах к слоям.

Большие значения портят качество.



# LLM.int8()

Используется absmax quantization и отдельная обработка выбросов.

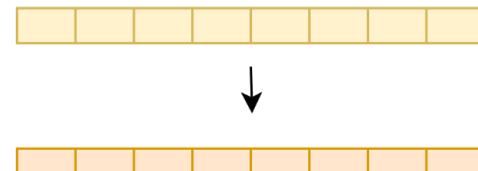


# Блоковая квантизация

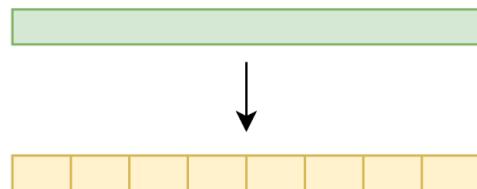
1) Вытягиваем в вектор



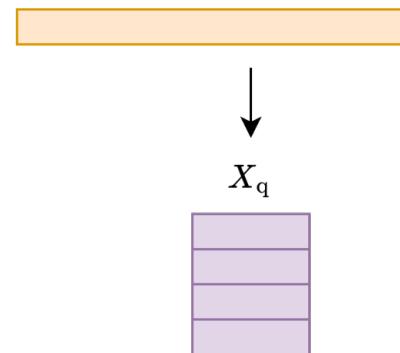
3) Квантизуем каждый блок отдельно



2) Нарезаем на блоки одинакового размера

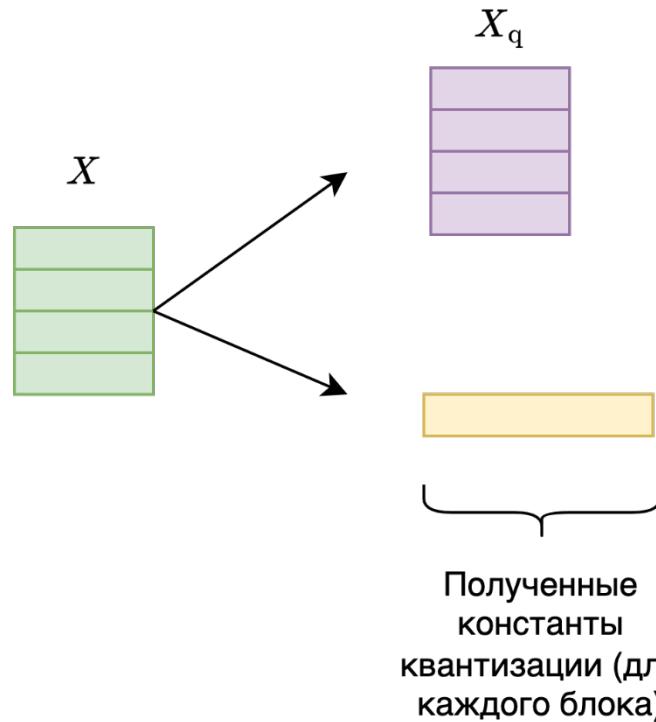


4) Собираем обратно в матрицу

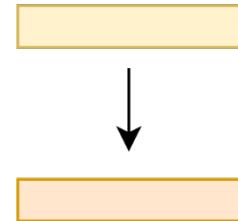


# Double quantization

1) Выполняем блочную квантизацию



2) Выполняем блочную квантизацию вектора констант квантизации

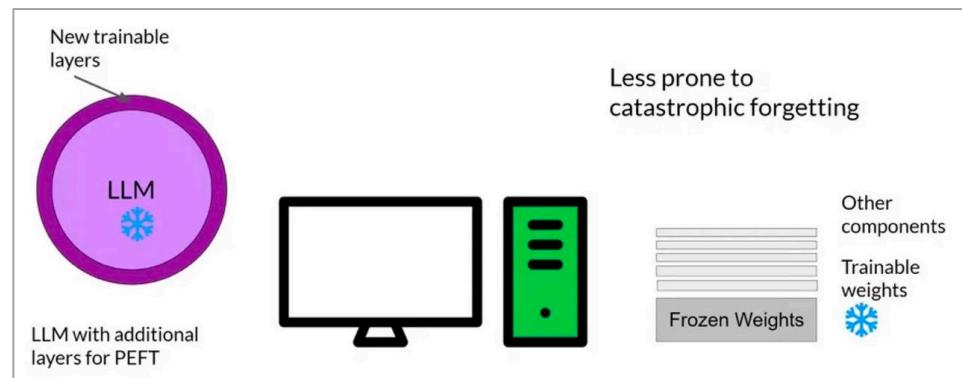


# PEFT (Parameter-Efficient Fine-Tuning)

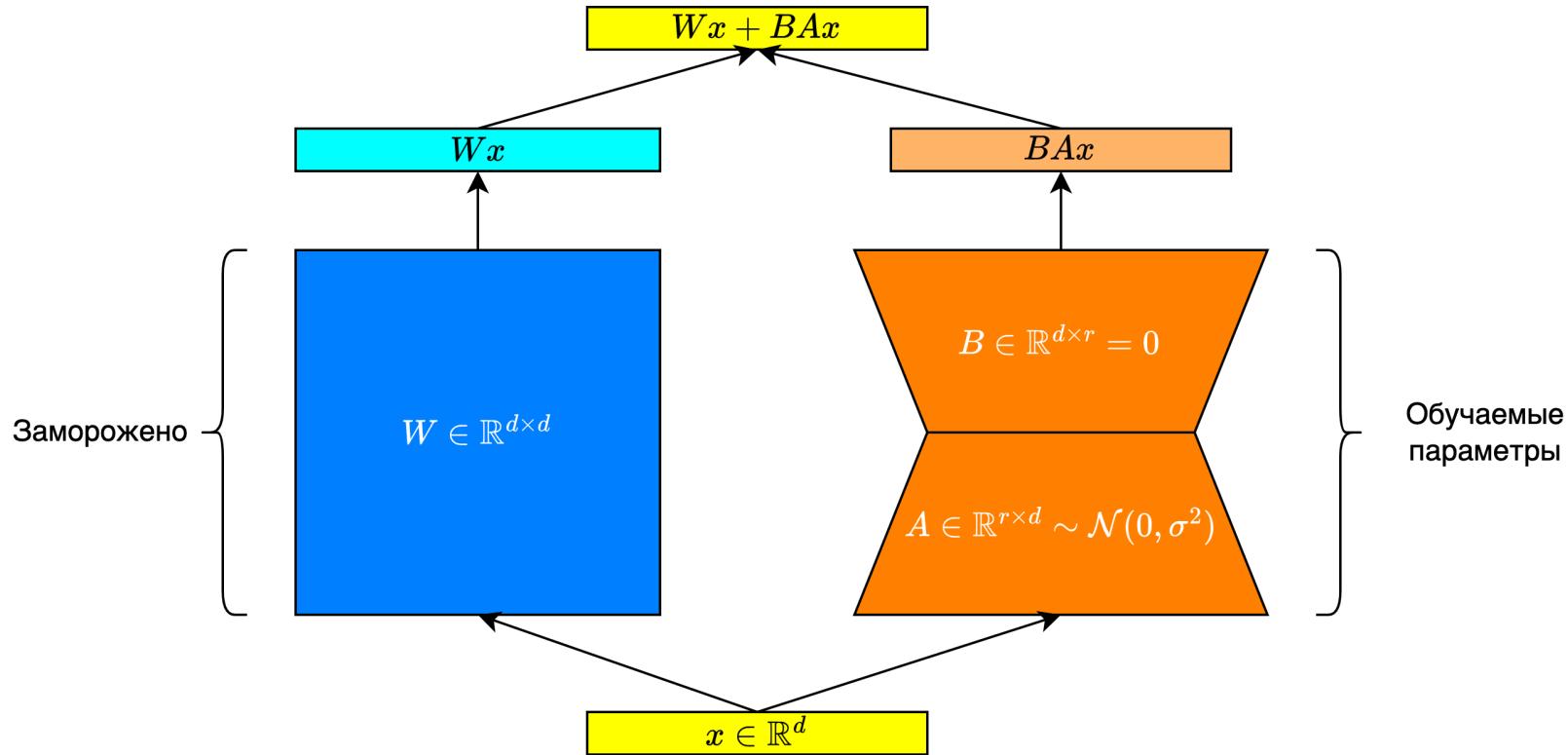


# PEFT (Parameter-Efficient Fine-Tuning)

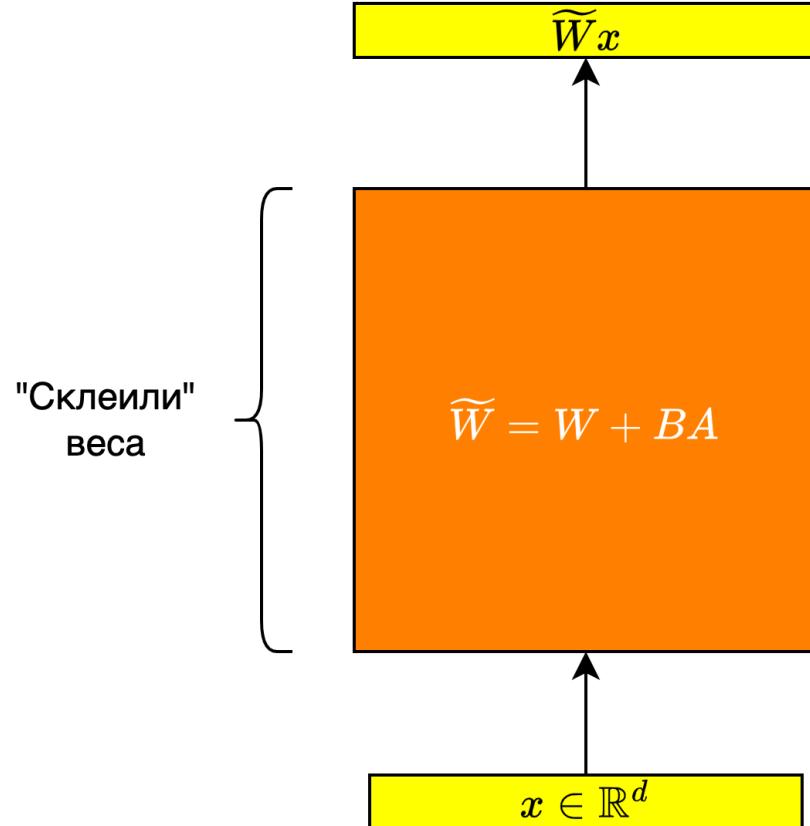
- Учим не все параметры модели, а небольшое количество (возможно, новых);
- Обновление весов происходит быстрее;
- Меньше потребление памяти;
- Легковесные адаптеры занимают мало места на диске, что позволяет обучать одну и ту же модель под разные задачи;
- Борется с катастрофическим забыванием.



# LoRA (Обучение)



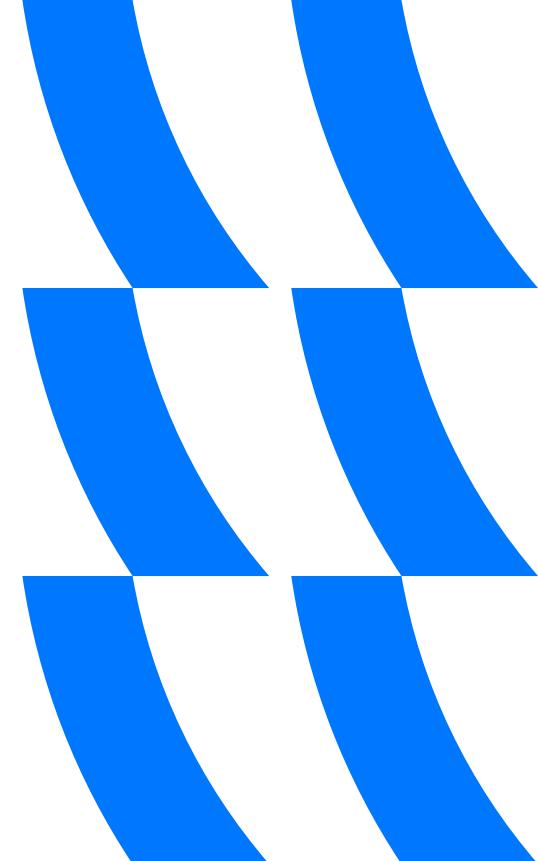
# LoRA (Инферес)



# LoRA

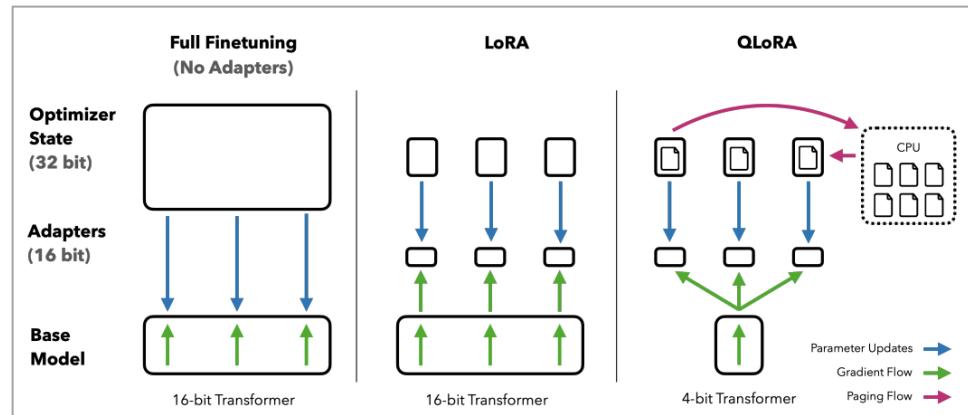
Преимущества:

- Количество обучаемых параметров обычно на несколько порядков меньше, чем у исходной модели.
- Для обучения нужно меньше памяти.
- Обучение происходит быстрее.
- Нет накладных расходов при инференсе.
- Можно учить отдельный адаптер под каждую задачу, а на инференсе менять их на лету.
- Можно использовать в связке с другими адаптерами



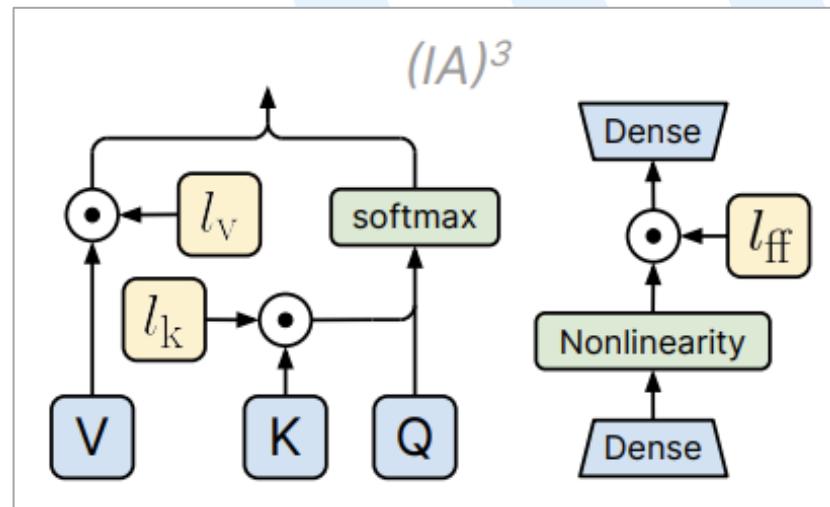
# QLoRA: Efficient Finetuning of Quantized LLMs (2023)

- Веса модели квантизуются с использованием блочной квантизации 4-bit + double quantization, а, когда нужно что-то посчитать, деквантизуются в bfloat16.
- Используются paged optimizers (если не хватает памяти, то состояние оптимизатора выгружаются с ГПУ на ЦПУ).



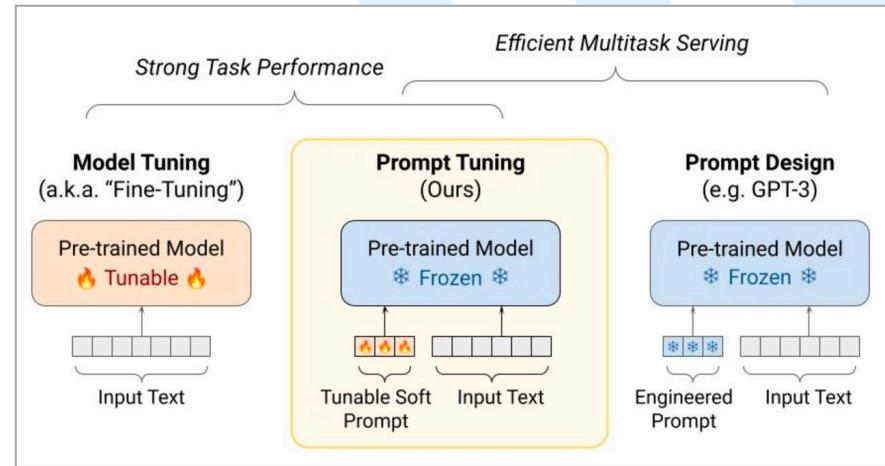
## IA3 (Infused Adapter by Inhibiting and Amplifying Inner Activations)

- На каждом слое обучаем 3 вектора, на которые нужно покомпонентно умножить соответственно keys, values и выходы нелинейного преобразования.
- Еще меньше обучаемых параметров, чем в LoRA.



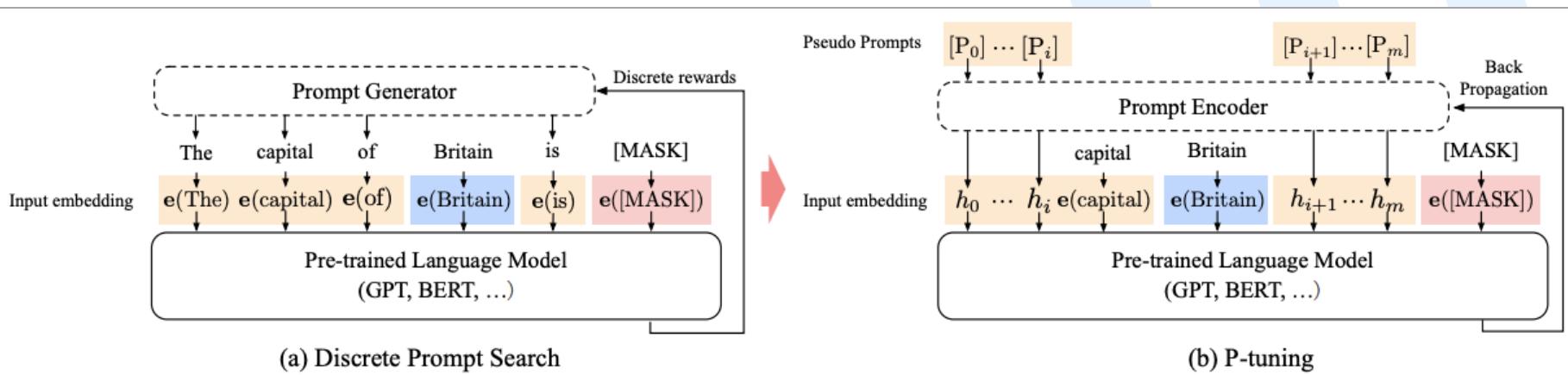
# Prompt tuning

- Замораживаем веса модели;
- Ко входу модели (последовательность эмбеддингов) приклеиваем слева несколько обучаемых векторов и учим только их параметры;
- Можно инициализироваться из некоторого текста (использовать в качестве начальной инициализации уже выученные вектора для данного текста).



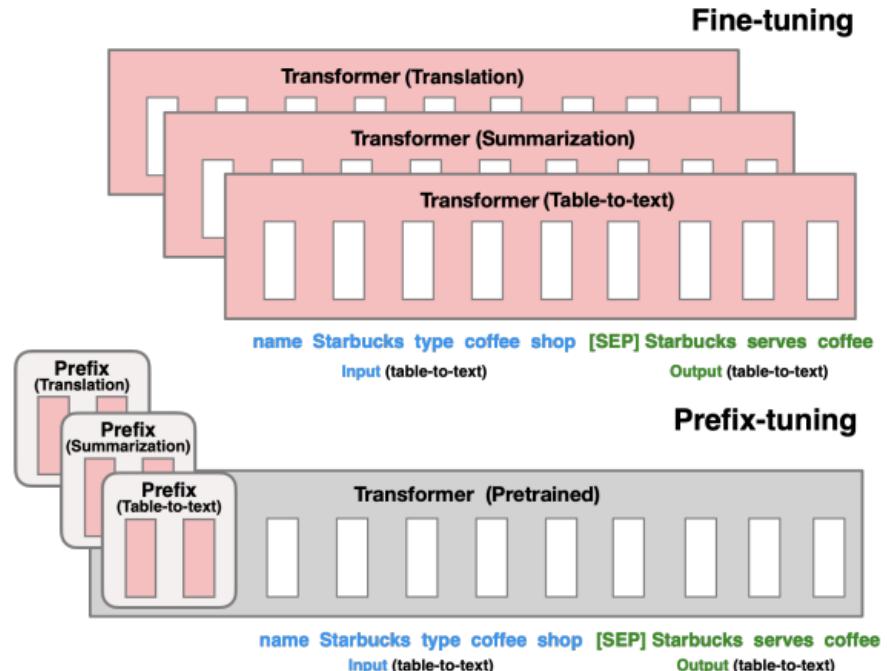
# P-tuning

- Как prompt-tuning, но можно вставлять вектора не только в начало.
- Векторное представление обучается при помощи BiLSTM.



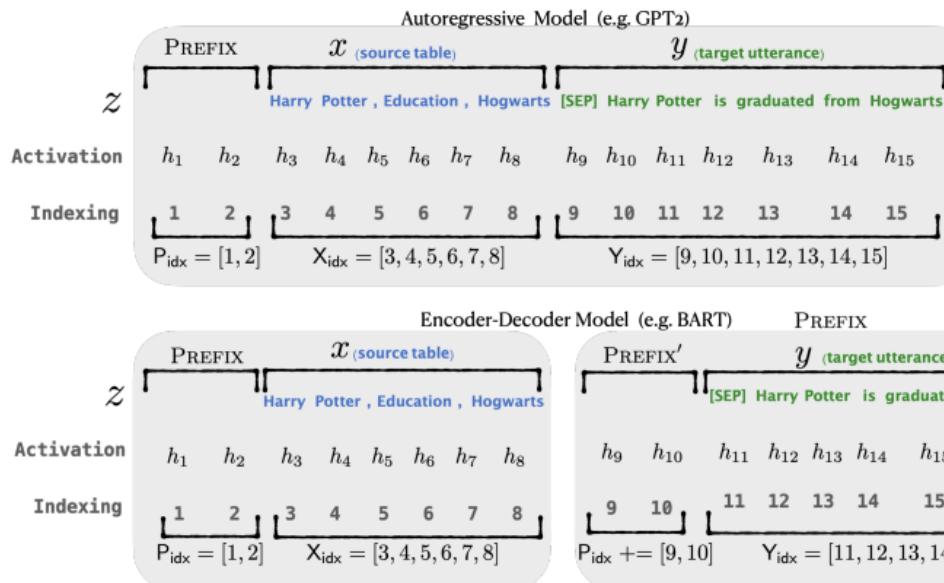
# Prefix-tuning

- Как prompt-tuning, но вставляем обучаемые вектора (keys, values) на каждом слое.
- Моделируем их с помощью fcnn с bottleneck.



[Prefix-Tuning: Optimizing Continuous Prompts for Generation](#) (2021)

# Prefix-tuning



**Summarization Example**

Article: Scientists at University College London discovered people tend to think that their hands are wider and their fingers are shorter than they truly are. They say the confusion may lie in the way the brain receives information from different parts of the body. Distorted perception may dominate in some people, leading to body image problems ... [ignoring 308 words] could be very motivating for people with eating disorders to know that there was a biological explanation for their experiences, rather than feeling it was their fault."

Summary: The brain naturally distorts body image – a finding which could explain eating disorders like anorexia, say experts.

**Table-to-text Example**

Table: name[Clowns] customer-rating[1 out of 5] eatType[coffee shop] food[Chinese] area[riverside] near[Clare Hall]

Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5 . They serve Chinese food .

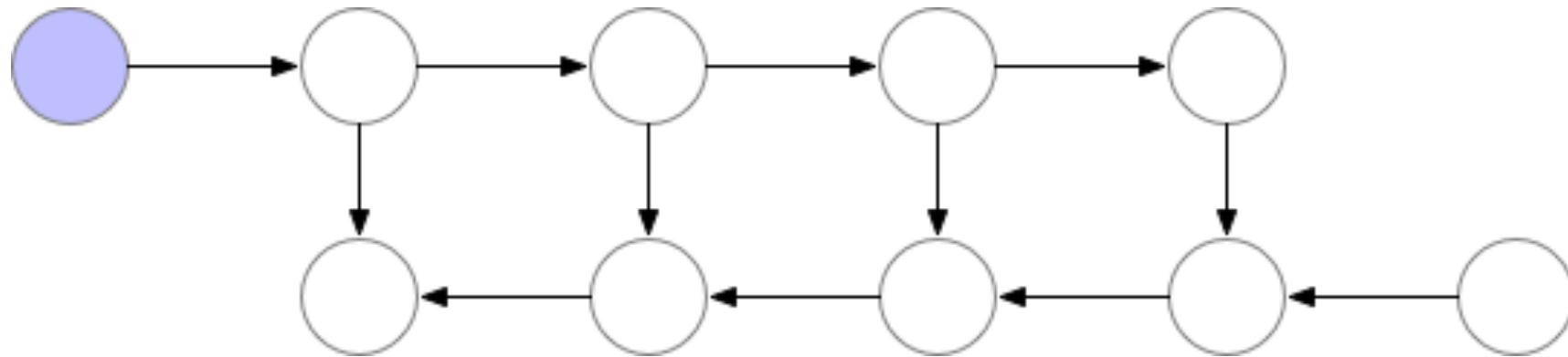
# Прочая оптимизация



# Gradient checkpointing

## Классический backpropagation:

- Память:  $O(n)$  по числу слоев
  - Вычисления:  $O(n)$  по числу слоев

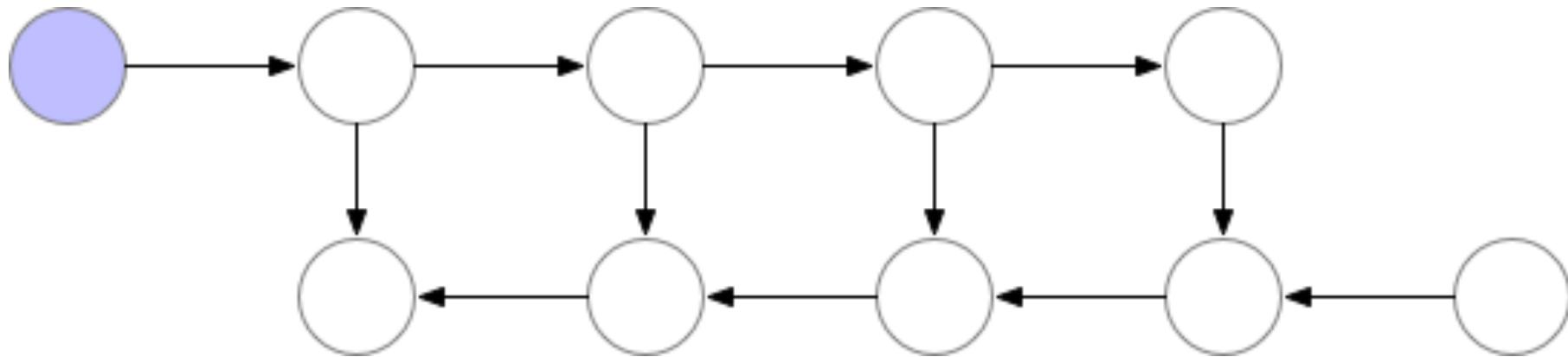


## Анимация

# Gradient checkpointing

Сокращение потребления памяти за счет существенного увеличения вычислений:

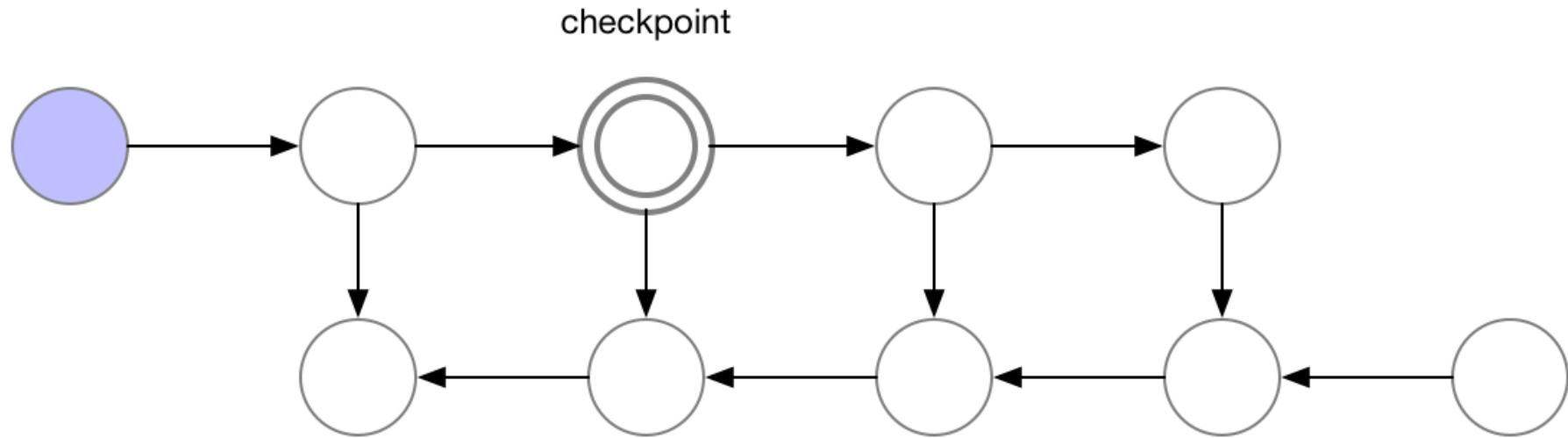
- Память:  $O(1)$  по числу слоев
- Вычисления:  $O(n^2)$  по числу слоев



[Анимация](#)

# Gradient checkpointing

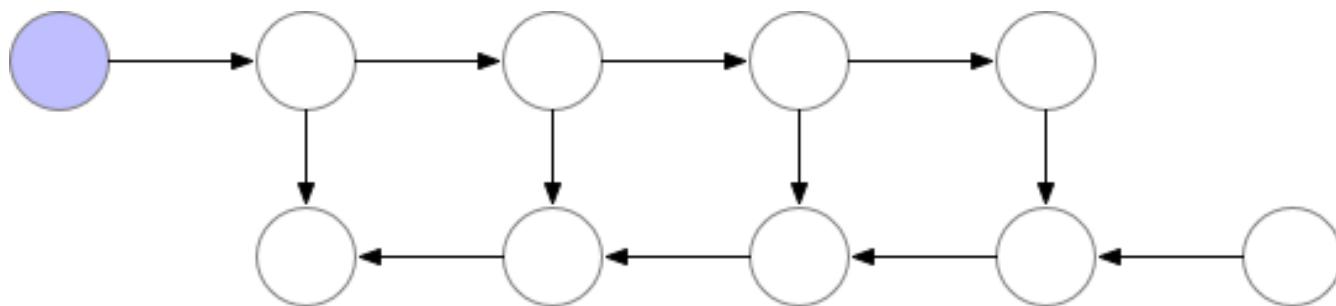
Будем хранить активации лишь в некоторых слоях



# Gradient checkpointing

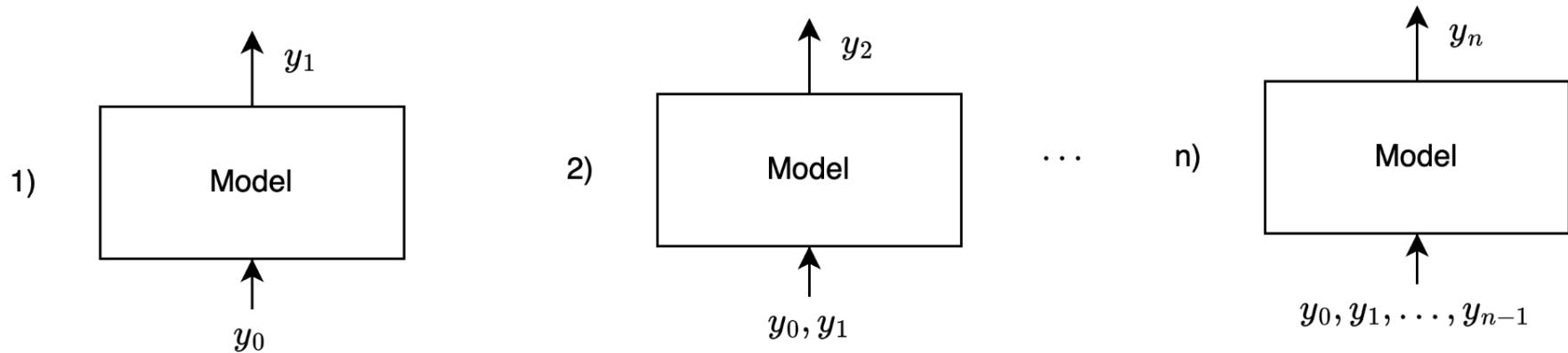
Можно в гибридном решении хранить активации у  $\sqrt{n}$  слоев, тогда

- Вычисления:  $O(n)$  по числу слоев
- Память:  $O(\sqrt{n})$  по числу слоев



[Анимация](#)

# Наивная генерация



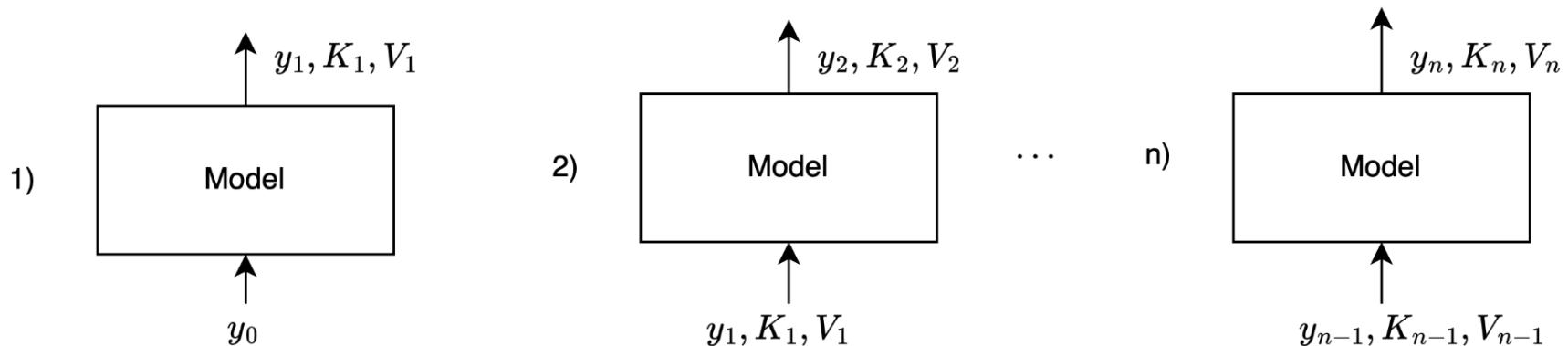
На очередном шаге прогоняем через модель всю уже сгенерированную последовательность, чтобы сгенерировать следующий токен.

Время работы всей генерации:  $O(n^3)$

# kv-кэширование

Внутреннее представление уже сгенерированных токенов не меняется при добавлении новых токенов справа (ведь токены не могут "смотреть вперед"), поэтому можно их не считать каждый раз заново, а запомнить.

Чтобы получить представление очередного добавленного токена, нужно знать ключи и значения всех предыдущих, поэтому будем их запоминать (key-value cache).



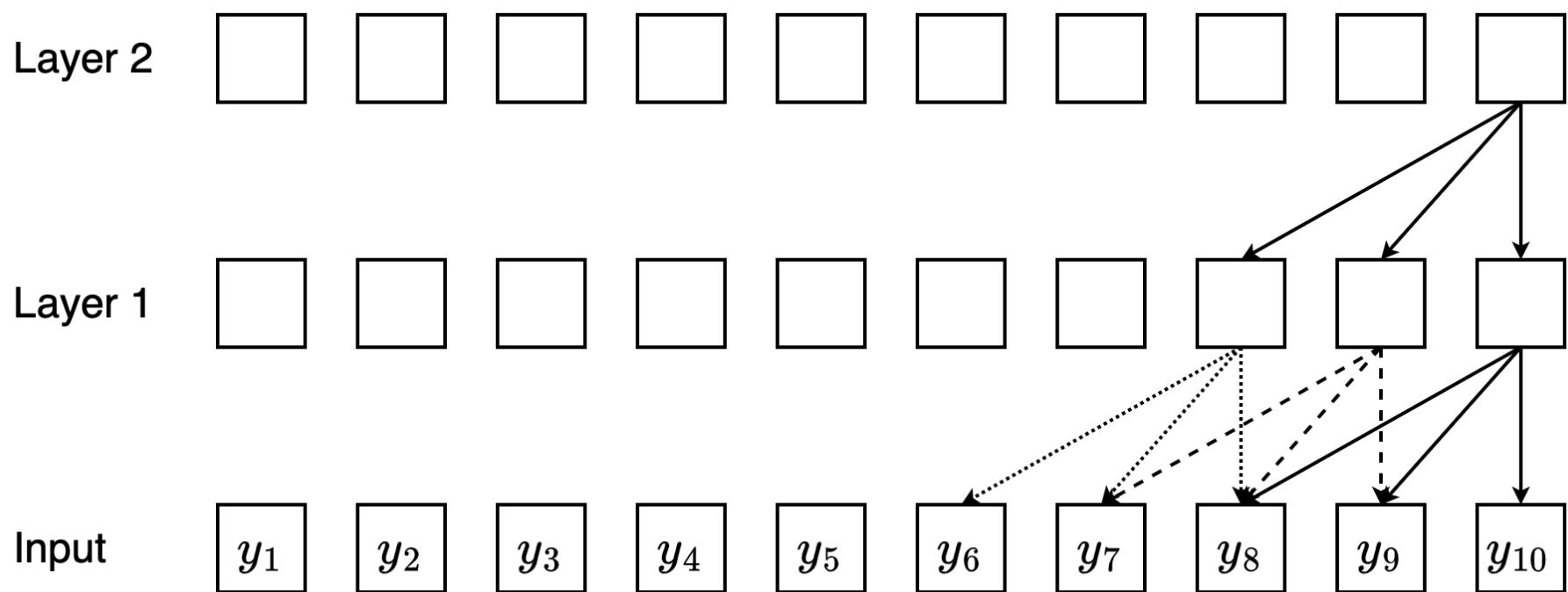
Время работы всей генерации:  $O(n^2)$

# Sliding window attention mask

- При декодировании смотрим лишь на несколько предыдущих токенов, а не на все уже сгенерированные.
- Линейное время работы всей генерации (если использовать kv-кэширование)
- kv-кэш фиксированного размера

1				
1	1			
1	1	1		
	1	1	1	
		1	1	1

# Длина контекста неявно увеличивается



# Что сегодня изучили?

- Обсудили, зачем оптимизировать обучение и инференс
- Поговорили о способах распределенного обучения
- Рассмотрели способы квантизации модели
- Изучили несколько методов PEFT:
  - LoRA
  - QLoRA
  - IA3
  - Prompt-tuning
  - Prefix-tuning
  - P-tuning
- Поговорили о еще нескольких способах оптимизации обучения/инференса:
  - Gradient checkpointing
  - kv-кэширование
  - Sliding window attention

## Что еще почитать?

- [ZeRO: Memory Optimizations Toward Training Trillion Parameter Models](#)
- [Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation](#)
- [LLM.int8\(\): 8-bit Matrix Multiplication for Transformers at Scale](#)
- [LoRA: Low-Rank Adaptation of Large Language Models](#)
- [QLoRA: Efficient Finetuning of Quantized LLMs](#)
- [Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning](#)
- [The Power of Scale for Parameter-Efficient Prompt Tuning](#)
- [GPT Understands, Too](#)
- [Prefix-Tuning: Optimizing Continuous Prompts for Generation](#)
- [Training Deep Nets with Sublinear Memory Cost](#)
- [Longformer: The Long-Document Transformer](#)
- [FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness](#)
- [Fast Transformer Decoding: One Write-Head is All You Need](#)
- [GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints](#)

# Спасибо за внимание!

Владимир Макаренко, старший разработчик-исследователь