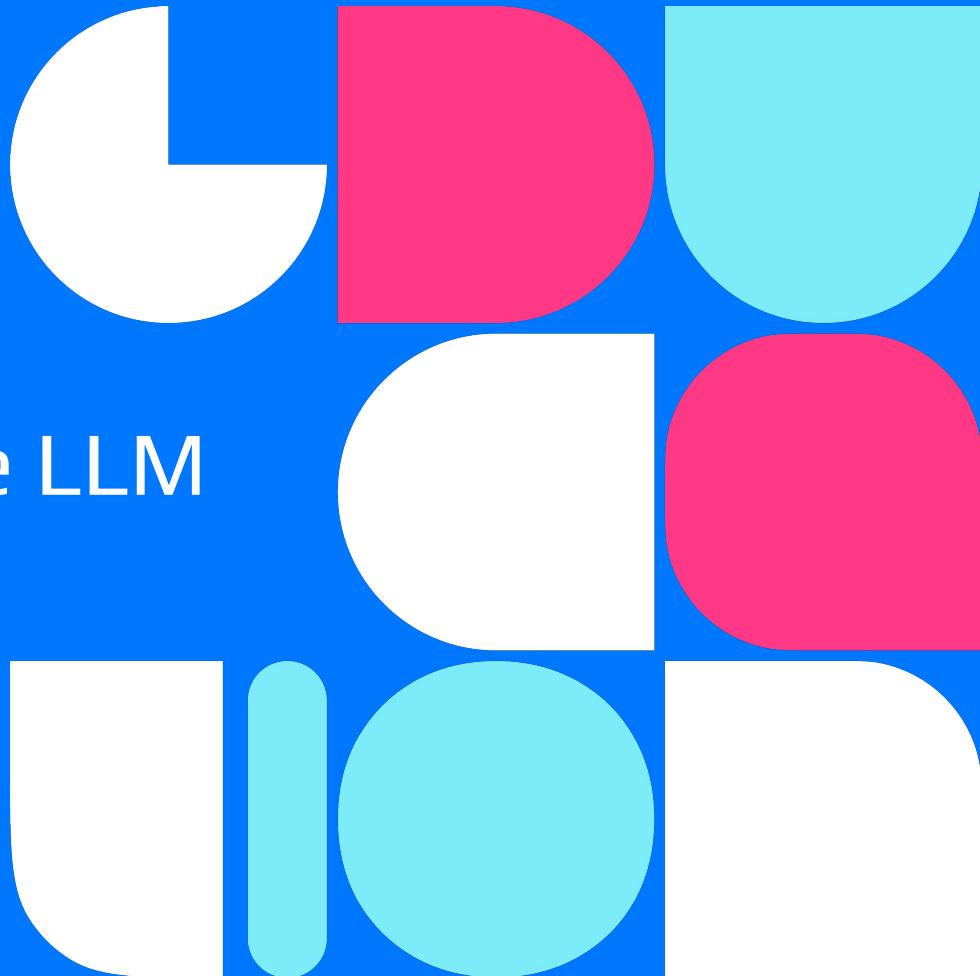




# Лекция 7. Ускорение LLM

Макаренко Владимир



# О чём сегодня поговорим

Зачем оптимизировать обучение и инференс?

Квантизация модели

Parameter-efficient fine-tuning (PEFT)



# 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

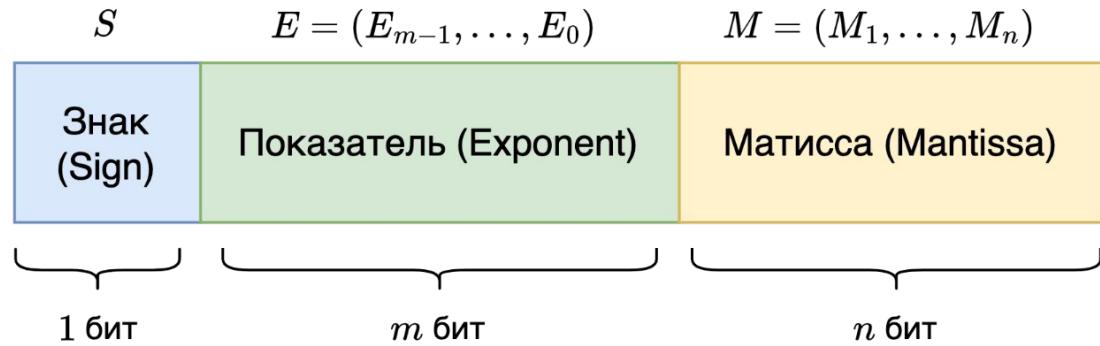
Во время обучения:

- Параметры модели
- Состояния оптимизатора
- Градиенты
- Активации

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

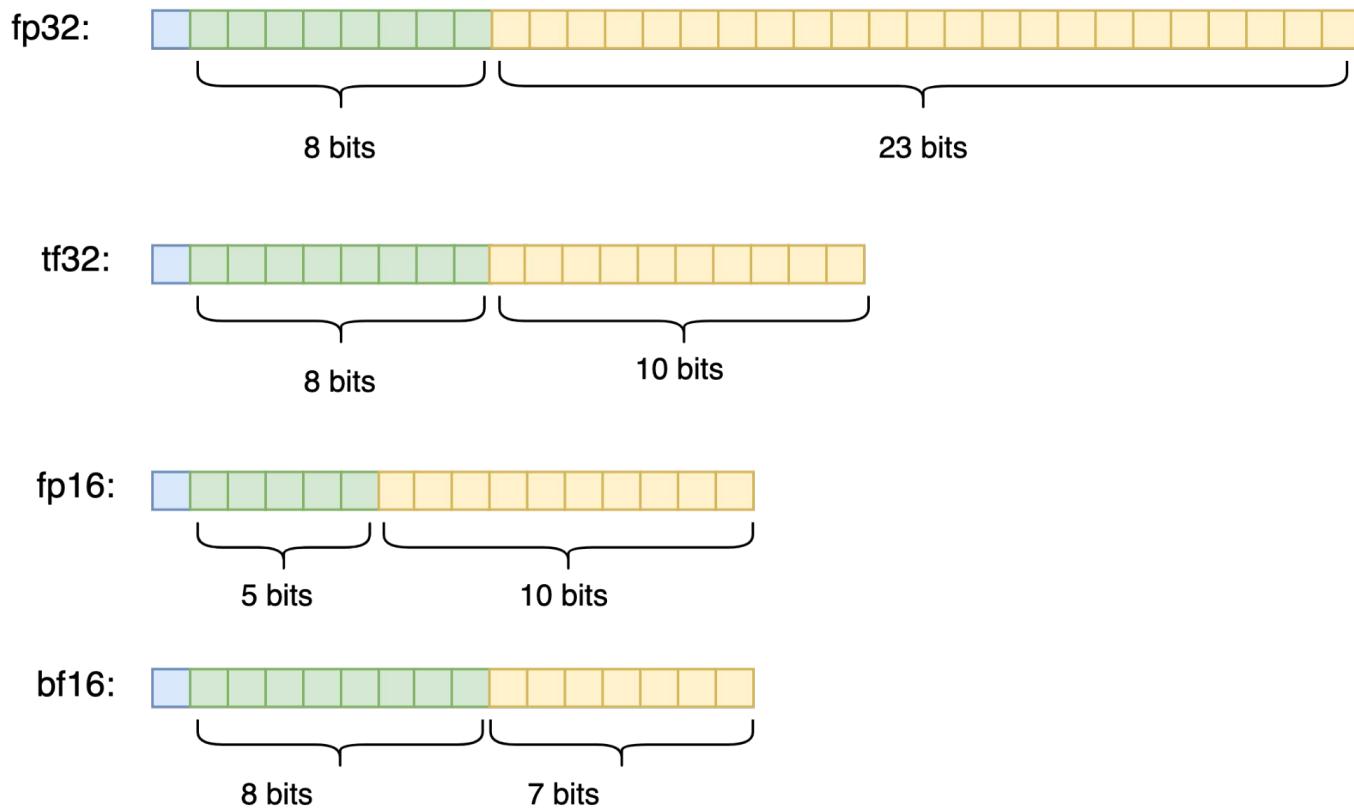
## **Представление числа:**

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

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

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

# 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:**

$$X = \begin{pmatrix} 1.5 & 0 & -1.1 \\ 2 & 0.5 & -0.1 \\ 1.2 & 1.7 & -0.5 \end{pmatrix}, \quad X_q = \begin{pmatrix} 95 & 0 & -70 \\ 127 & 32 & -6 \\ 76 & 108 & -32 \end{pmatrix}, \quad X_{dq} = \begin{pmatrix} 1.496 & 0 & -1.102 \\ 2 & 0.504 & -0.094 \\ 1.197 & 1.701 & -0.504 \end{pmatrix}$$

# 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

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

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

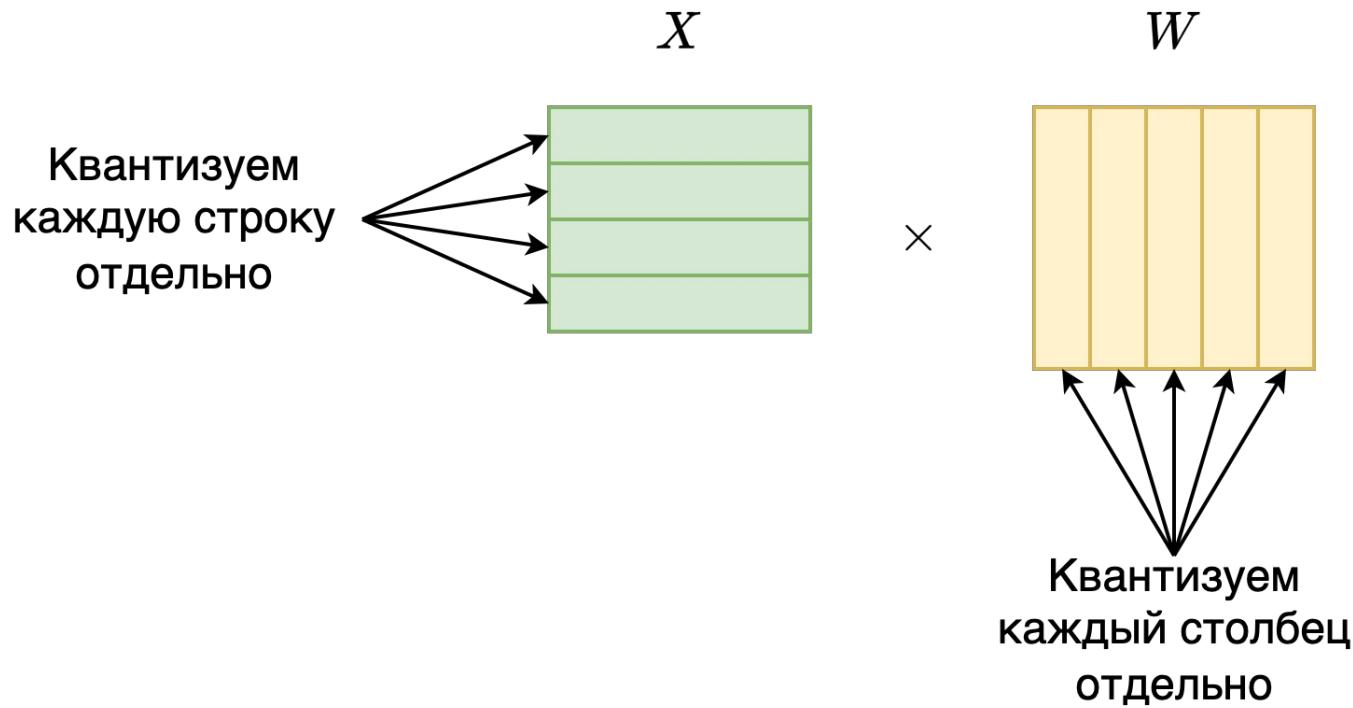
## 8-битное умножение матриц (zeropoint)

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

Имеем

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

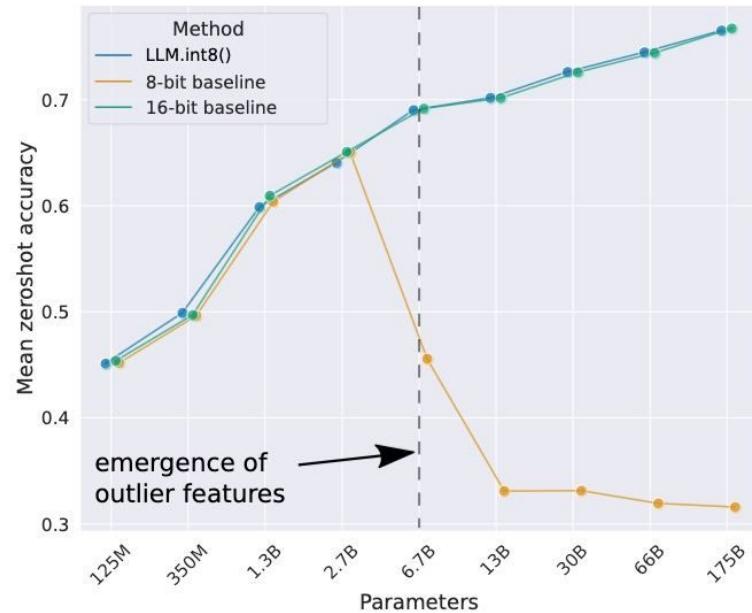
# Vector-wise quantization



# Проблема выбросов

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

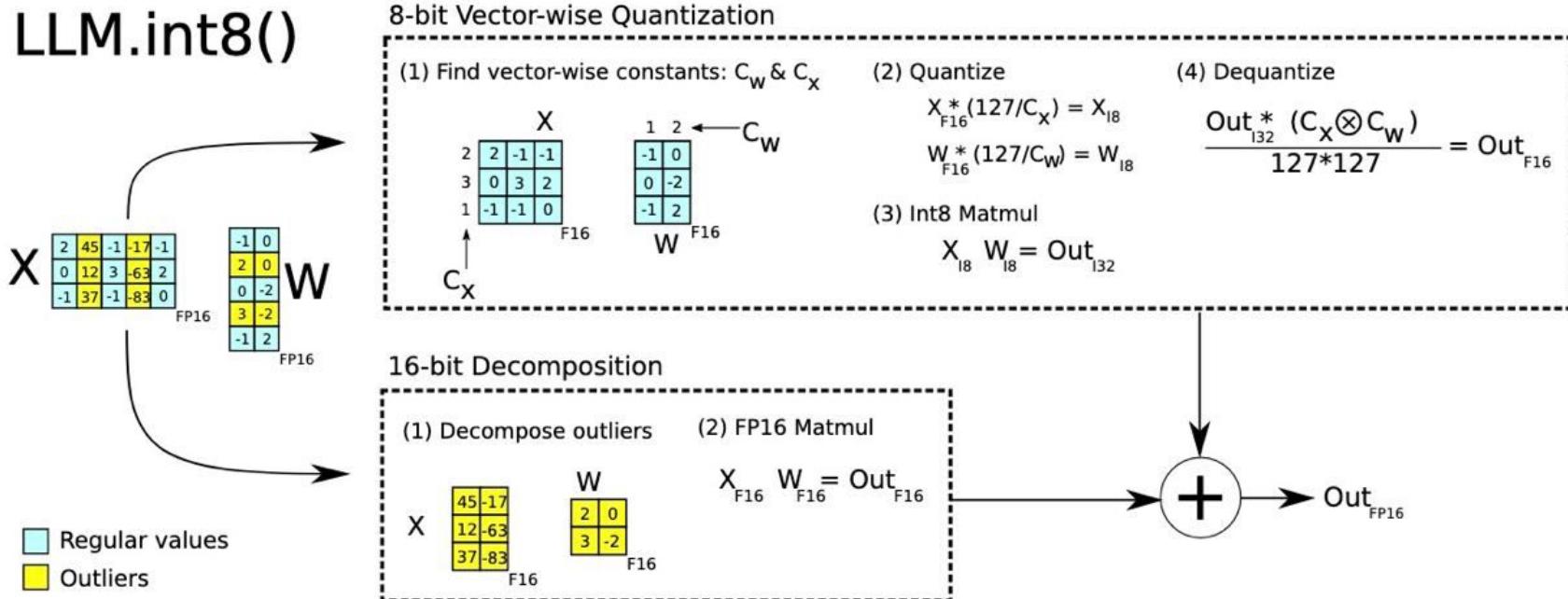
Большие значения портят качество.



# LLM.int8()

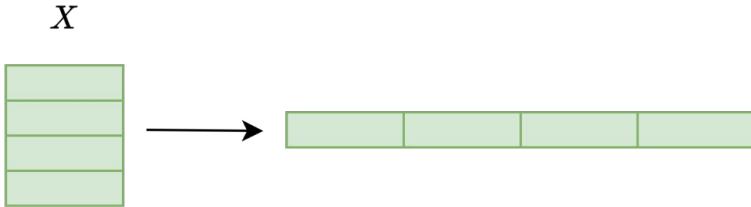
Используется absmax quantization и отдельная обработка выбросов.

## LLM.int8()

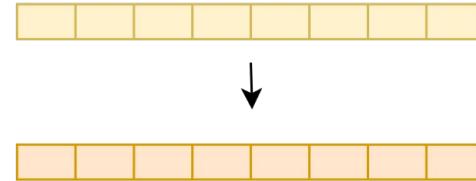


# Блоковая квантизация

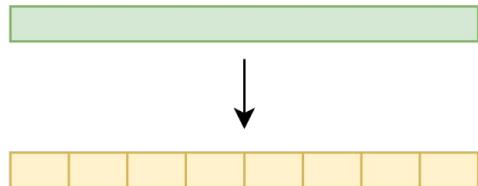
1) Вытягиваем в вектор



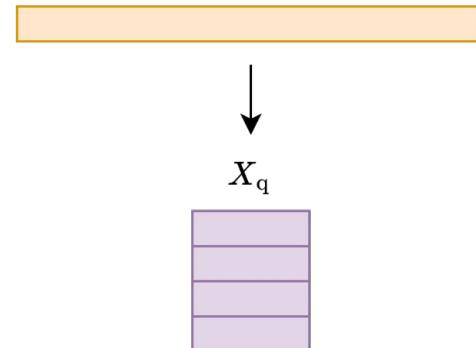
3) Квантизуем каждый блок отдельно



2) Нарезаем на блоки одинакового размера

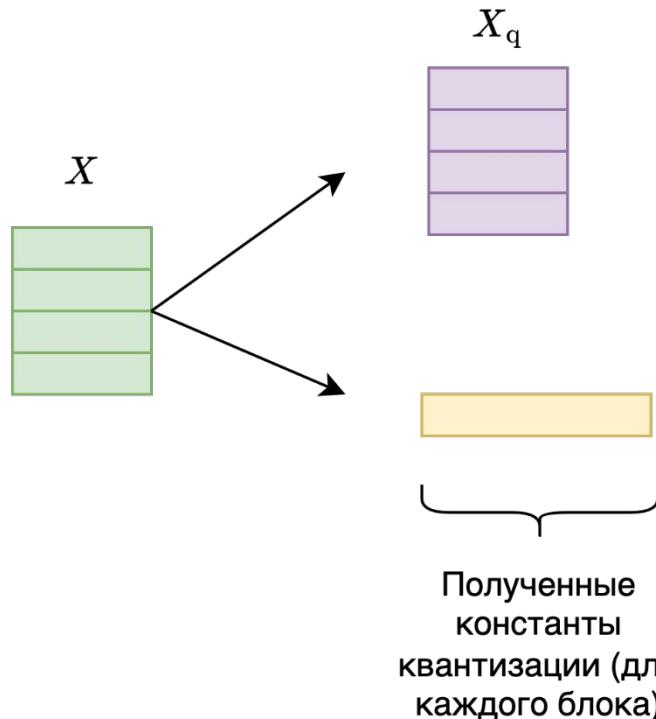


4) Собираем обратно в матрицу

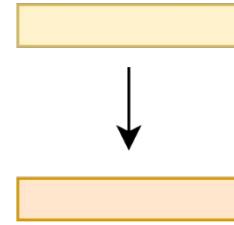


# Double quantization

1) Выполняем блочную квантизацию



2) Выполняем блочную квантизацию вектора констант квантизации



# Optimal Brain Surgeon (OBS)

**Цель:** удалить несколько параметров нейросети, скорректировав при этом остальные веса так, чтобы ошибка на тренировочной выборке увеличилась как можно меньше.

**Задача оптимизации (удаление одного параметра)**

$$\operatorname{argmin}_{\Delta w} L(w^* + \Delta w), \quad \text{s.t.} \quad e_q^T \cdot \Delta w = -w_q.$$

**Обозначения**

Параметры нейросети (вектор):  $w \in \mathbb{R}^n$

Функция потерь:  $L(w)$

Параметры, полученные в результате обучения:

$$w^* = \operatorname{argmin}_w L(w)$$

Естественный базис:  $e_1, e_2, \dots, e_n \in \mathbb{R}^n$ , где

$$e_k[j] = \begin{cases} 1, & j = k, \\ 0, & j \neq k \end{cases}$$

# Optimal Brain Surgeon (OBS)

**Упрощение:** воспользуемся формулой Тейлора.

$$L(w^* + \Delta w) \approx \overbrace{\frac{\partial L}{\partial w}(w^*)}^{\text{0}} \cdot \Delta w + \frac{1}{2} \cdot \Delta w^T \cdot H \cdot \Delta w = \frac{1}{2} \cdot \Delta w^T \cdot H \cdot \Delta w,$$

где

$$H_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}.$$

# Optimal Brain Surgeon (OBS)

## Решение

$$\Delta w = -\frac{w_q}{[H^{-1}]_{q,q}} H^{-1} \cdot e_q = -\frac{w_q}{[H^{-1}]_{q,q}} H_{:,q}^{-1},$$

$$L_q \equiv L(w^* + \Delta_w) \approx \frac{1}{2} \frac{w_q^2}{[H^{-1}]_{q,q}}$$

По значению  $L_q$  можем оценить, как изменится функция потерь, если мы обнулим параметр  $w_q$ , скорректировав остальные веса.

Удалить несколько параметров можно, например, используя жадный алгоритм: каждый раз выбирать вес, удаление которого меняет функцию потерь меньше всего, а затем все пересчитывать.

# Optimal Brain Quantizer (OBQ)

**Идея:** применить аналог OBS для каждого слоя.

**Функция потерь**

$$L(\hat{W}) = \|WX - \hat{W}X\|_2^2$$

**Задача оптимизации**

$$\hat{W} = \operatorname{argmin}_{\hat{W}} L(\hat{W}), \quad \text{s.t.} \quad \hat{W}_{i,j} \in G$$

**Обозначения**

Параметры слоя:  $W$

Квантизованные параметры слоя:  $\hat{W}$

Датасет (матрица входов к слою) для адаптации:  $X$

Сетка квантизации (допустимые значения): множество  $G$

**Наблюдение:** можем решать задачу независимо для каждой строки матрицы  $\hat{W}$ .

# Optimal Brain Quantizer (OBQ)

Для каждой строки запускаем аналог OBS: теперь цель - не занулить вес, а округлить (сдвинуть к соответствующему значению).

OBS (обнуление):

$$\Delta w = -\frac{w_q}{[H^{-1}]_{q,q}} H_{:,q}^{-1}$$

Аналог OBS (квантизация):

$$\Delta w = -\frac{w_q - \text{quant}(w_q)}{[H^{-1}]_{q,q}} H_{:,q}^{-1}$$

Как и раньше, для квантизации жадно выбираем вес, для которого изменение функции потерь минимально; корректируем остальные параметры (которые еще не квантованы).

# Optimal Brain Quantizer (OBQ)

**Проблема:** если часть весов являются выбросами, то они, как правило, будут обрабатываться в конце, и уже будет очень мало свободных параметров, с помощью которых можно компенсировать большую ошибку квантизации. Кроме того, в процессе квантизации, часть весов может вылезти за границы сетки, что тоже скажется на качестве позже.

**Решение:** обрабатывать выбросы вне очереди.

## GPTQ

- Авторы заметили, что OBQ работает примерно с тем же качеством, если выбирать веса для квантизации не жадно, а произвольно. Это позволило авторам обрабатывать сразу все строки одновременно, квантизую один и тот же столбец (одна и та же позиция внутри каждой строки).
- Авторы предложили дополнительные модификации для ускорения и устойчивости

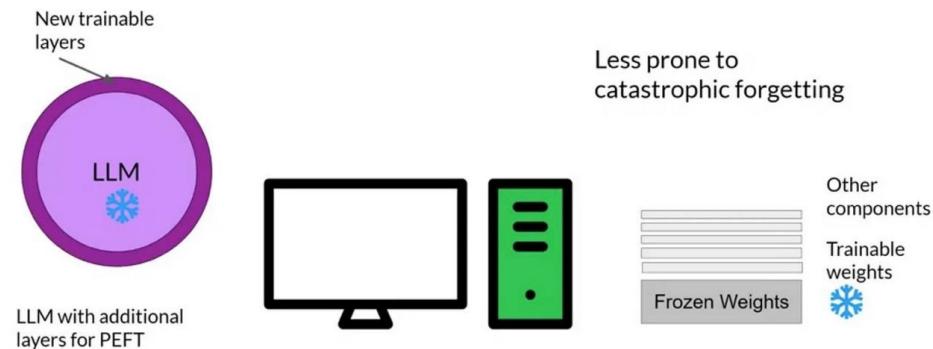
[GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers \(2022\)](#)

# PEFT

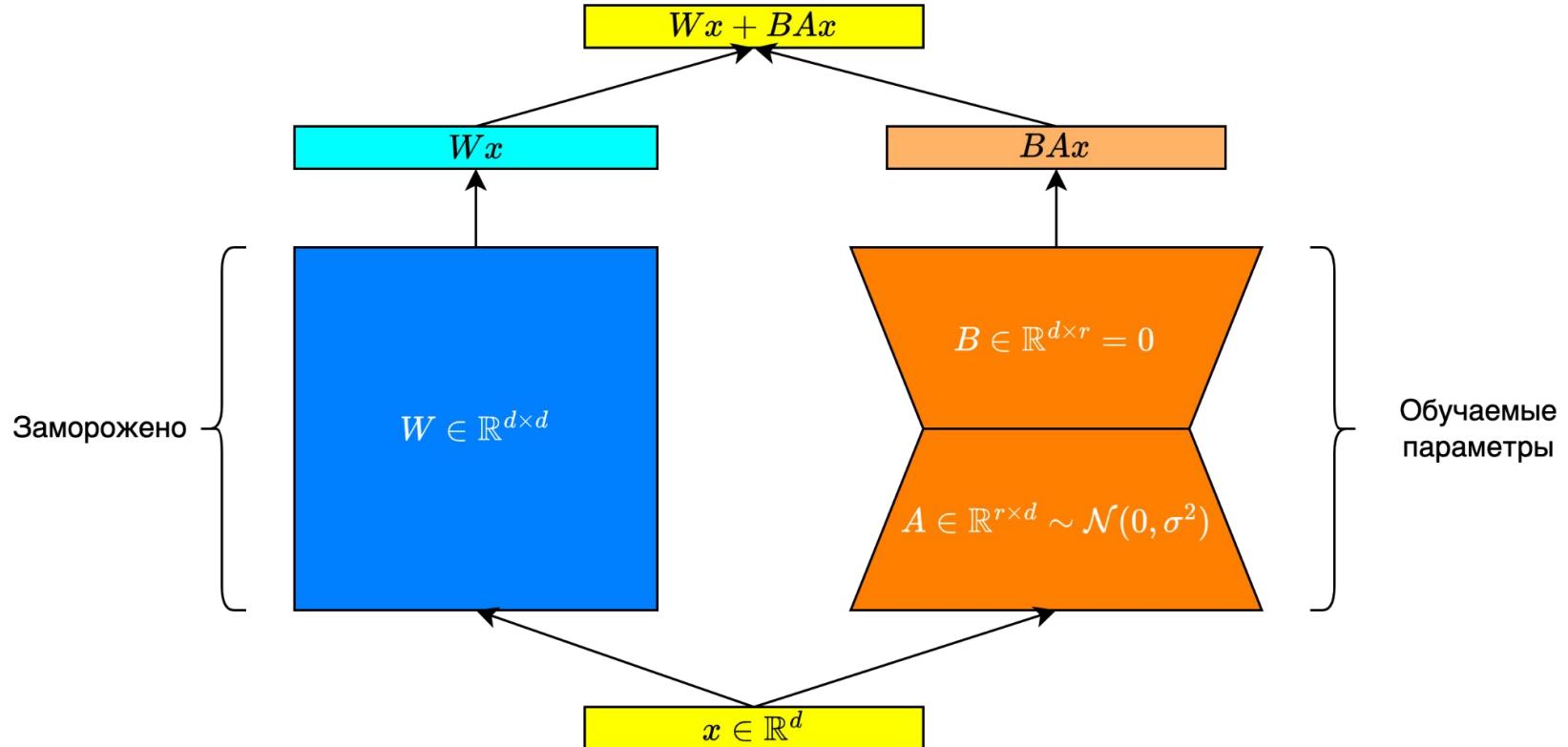
## (Parameter-Efficient Fine-Tuning)

# PEFT (Parameter-Efficient Fine-Tuning)

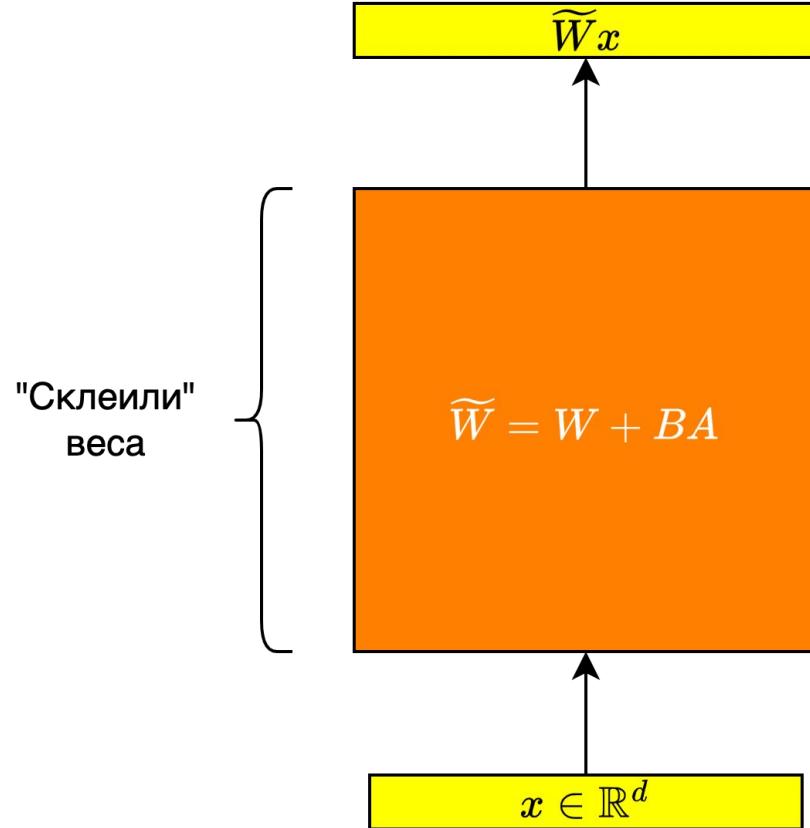
- Учим не все параметры модели, а небольшое количество (возможно, новых);
- Обновление весов происходит быстрее;
- Меньше потребление памяти;
- Легковесные адаптеры занимают мало места на диске, что позволяет обучать одну и ту же модель под разные задачи;
- Боремся с катастрофическим забыванием.



# LoRA (Обучение)



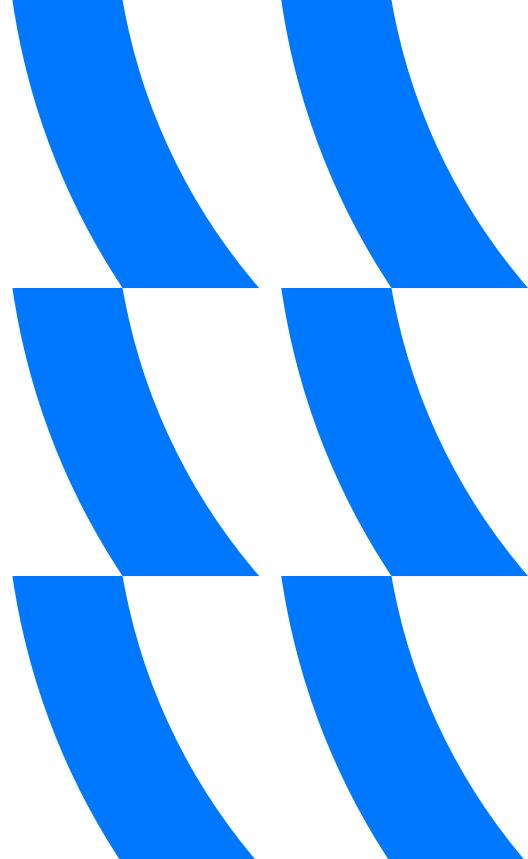
# LoRA (Инференс)



# LoRA

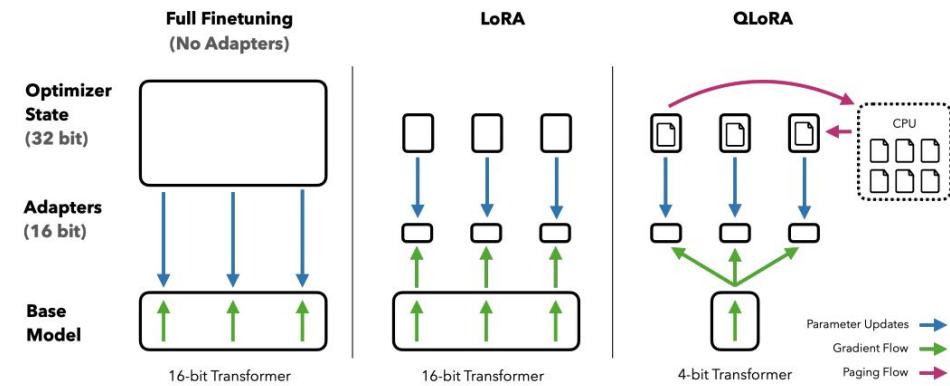
Преимущества:

- 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

- 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).



# DoRA

**Проблема:** LoRA показывает качество хуже, чем fine-tuning.

Авторы предположили, что проблема может быть связана не только с низким рангом (которого недостаточно для полной адаптации), и исследовали, как меняются “вклады” в итоговое изменение:

- **magnitude** variations
- **directional** variations

# DoRA

Пусть  $W \in \mathbb{R}^{d \times k}$ . Тогда ее можно представить в виде

$$W = m \frac{V}{\|V\|_c} = \|W\|_c \frac{W}{\|W_c\|},$$

где  $m \in \mathbb{R}^{1 \times k}$  - длины столбцов (**magnitude vector**),  $V \in \mathbb{R}^{d \times k}$  - матрица (**directional matrix**). Здесь умножение и деление выполняется покомпонентно для каждой строки.

В рамках такого представления получается, что матрица  $\frac{V}{\|V\|_c}$  состоит из столбцов единичной длины.

Обозначим через  $W_0, W_{\text{FT}}, W_{\text{LoRA}}$ , соответственно матрицы весов после предобучения (pre-training), дообучения всех весов модели (fine-tuning) и дообучения с помощью LoRA. Для них можно записать разложение:

$$W_0 = m_0 \frac{V_0}{\|V_0\|_c}, \quad W_{\text{FT}} = m_{\text{FT}} \frac{V_{\text{FT}}}{\|V_{\text{FT}}\|_c}, \quad W_{\text{LORA}} = m_{\text{LORA}} \frac{V_{\text{LORA}}}{\|V_{\text{LORA}}\|_c},$$

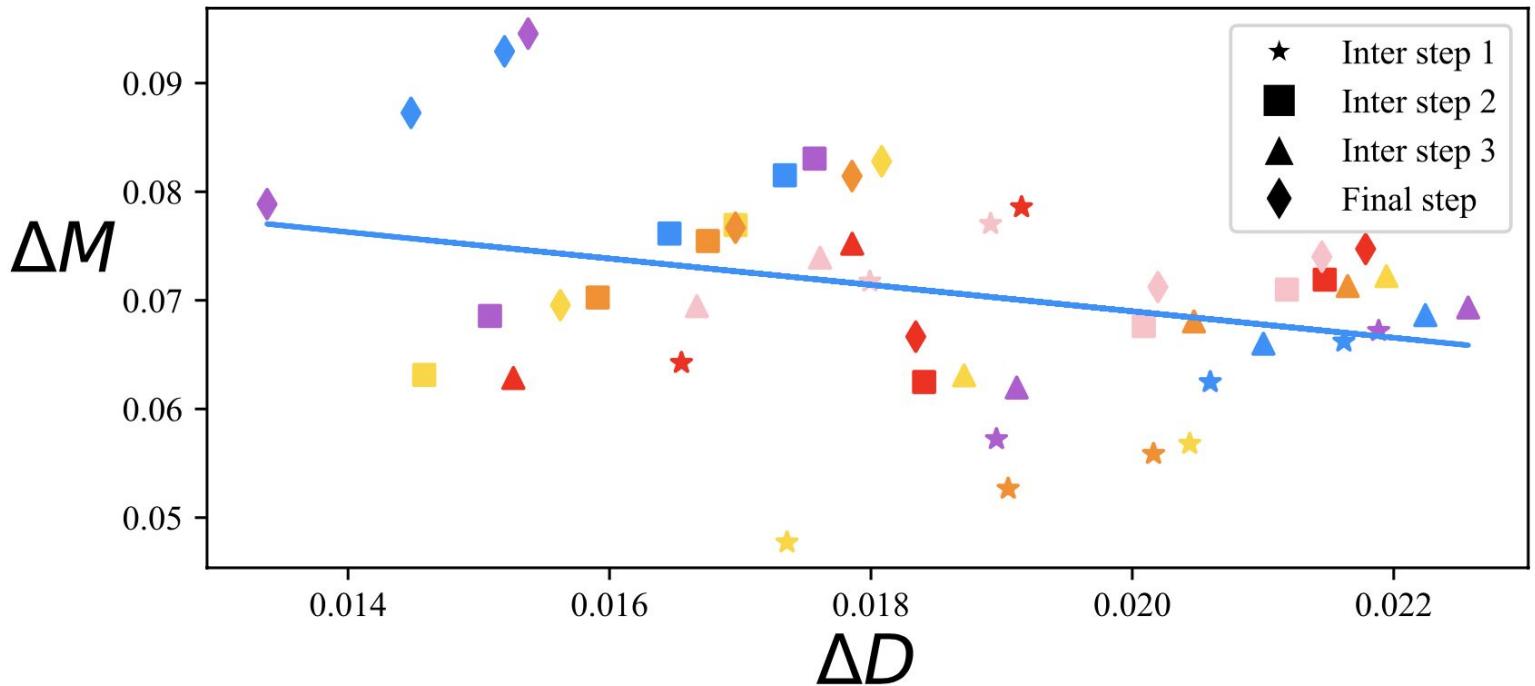
# DoRA

Обозначим

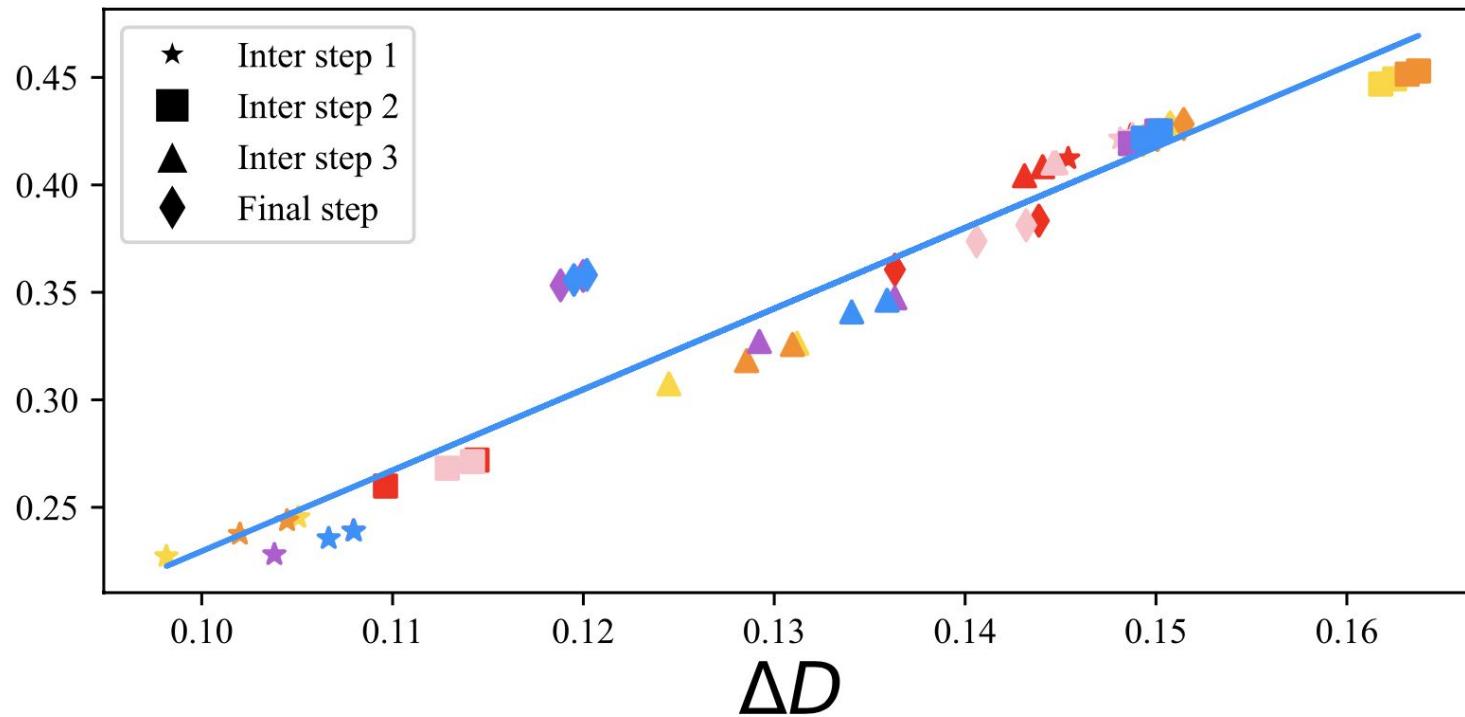
$$\Delta M_{\text{type}}^t = \frac{\sum_{n=1}^k |m_{\text{type}}^{n,t} - m_0^n|}{k}, \quad \Delta D_{\text{type}}^t = \frac{\sum_{n=1}^k (1 - \cos(V_{\text{type}}^{n,t}, W_0^n))}{k}, \quad \text{type} \in \{\text{FT}, \text{LORA}\},$$

где  $t$  - итерация обучения.

FT

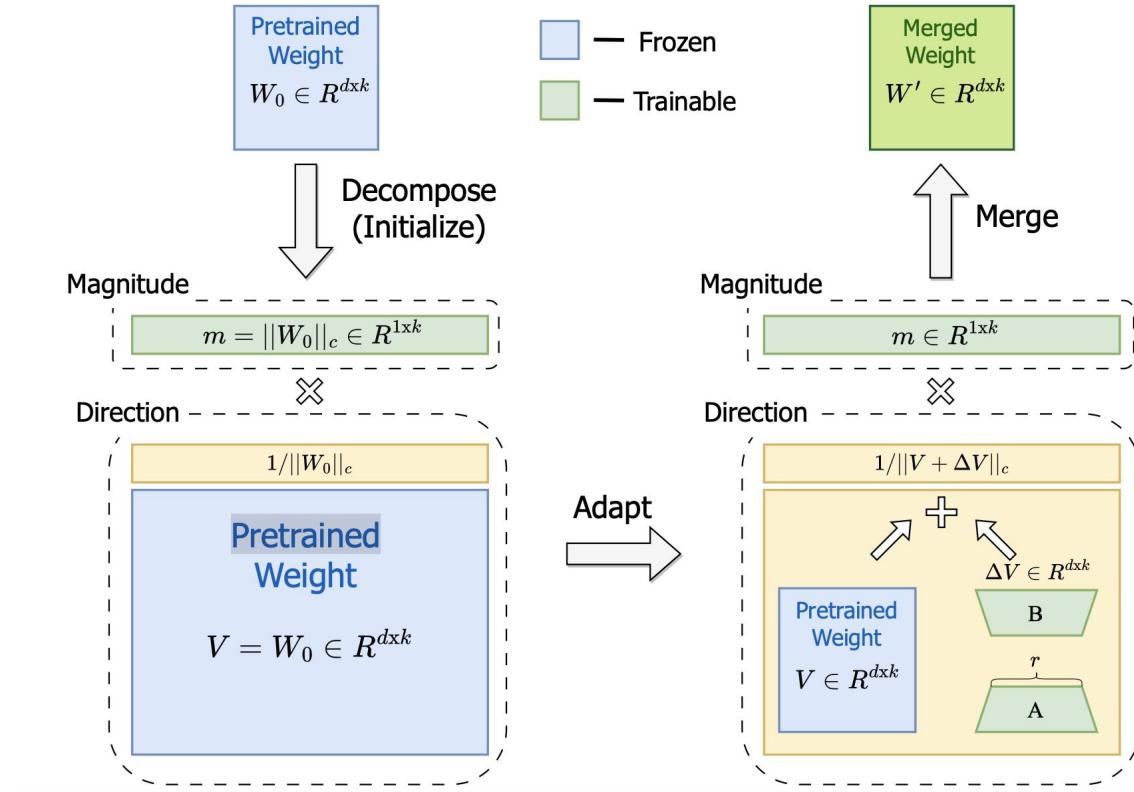


## LoRA

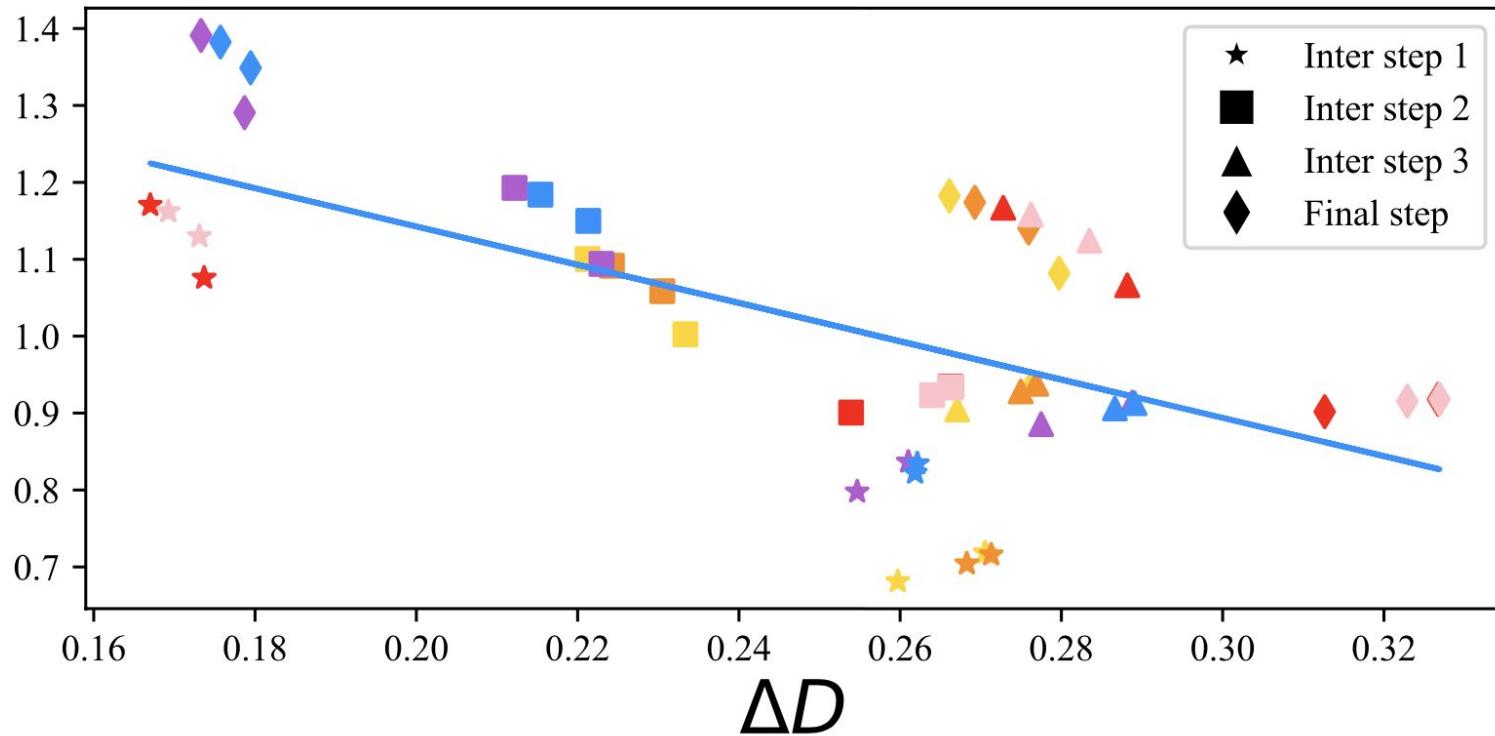


# DoRA

- Явно учим magnitude vector
- Для directional matrix используем LoRA



## DoRA



# DoRA

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA <sup>†</sup> (Ours)	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	<b>77.5</b>
	DoRA (Ours)	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	<b>78.4</b>
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA <sup>†</sup> (Ours)	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	<b>80.8</b>
	DoRA (Ours)	0.68	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	<b>81.5</b>
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA <sup>†</sup> (Ours)	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	<b>80.5</b>
	DoRA (Ours)	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	<b>79.7</b>
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA <sup>†</sup> (Ours)	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	<b>85.0</b>
	DoRA (Ours)	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	<b>85.2</b>

# RoSA

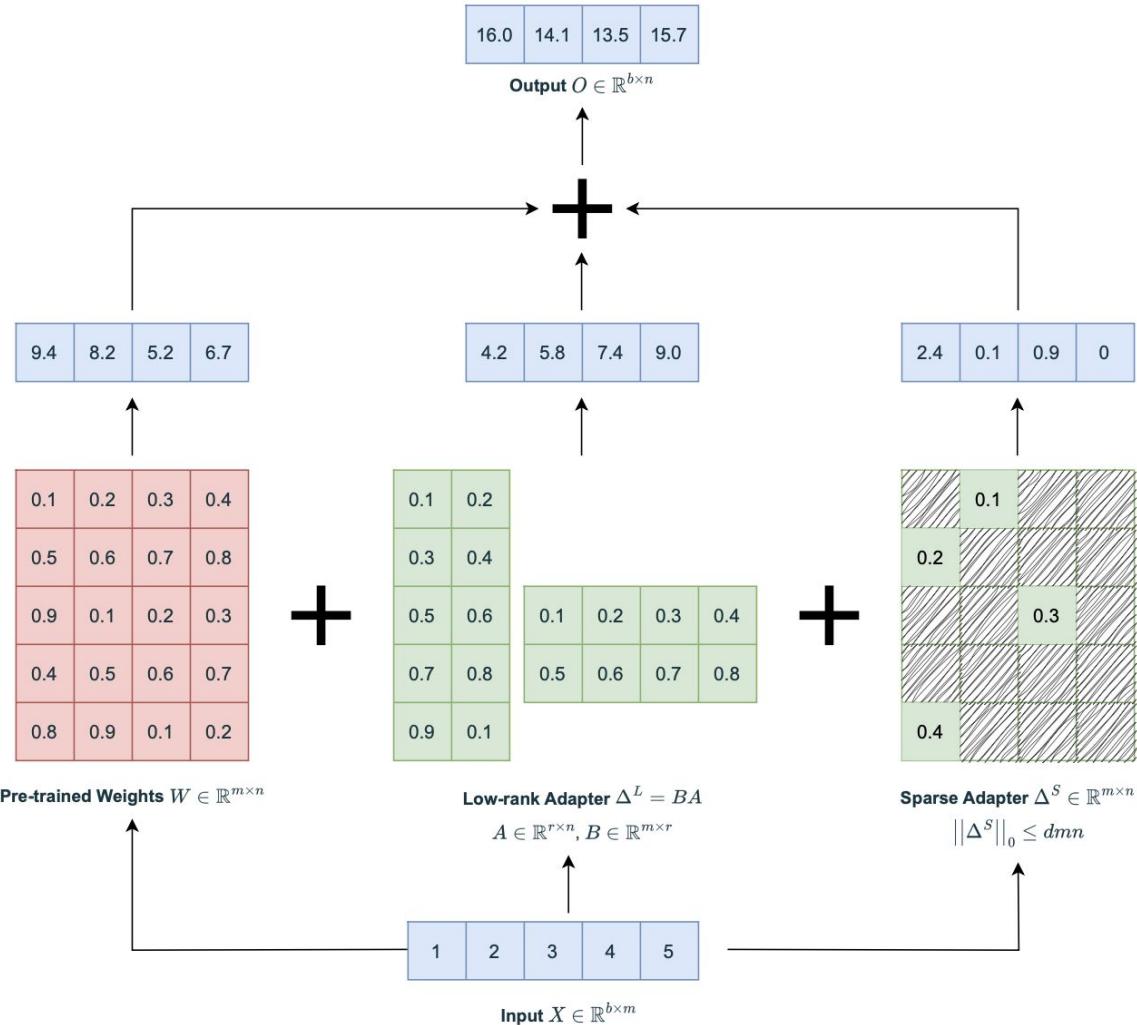
**Проблема:** LoRA показывает качество хуже, чем fine-tuning.

**Причина:** бывает так, что нужно радикально изменить несколько параметров в слое. Они выбиваются из общей картины (выбросы), и LoRA из-за низкого ранга плохо справляется с задачей.

**Решение:** авторы предлагают отдельно обучать “выбросы” с помощью разреженной матрицы.

# RoSA

- Обучаются два адаптера:  
**низкоранговый и  
разреженный**



- Маска для разреженного адаптера выбирается на градиентах функции потерь – это параметры, с самыми большими по величине производными (для оценки используется небольшая часть датасета).

---

**Algorithm 1** Mask Generation

---

**Require:**  $\mathcal{W}, \bar{w} \leftarrow$  the fully connected weights and the rest of the LLM parameters, respectively

**Require:**  $\mathcal{D}_M \leftarrow$  the mask generation dataset, typically a small subset of the actual dataset

**Require:**  $\mathcal{L}(\cdot) \leftarrow$  the loss function

**Require:**  $d \leftarrow$  mask density

**Require:**  $\alpha \leftarrow$  gradient accumulation exponent

$\mathcal{G} \leftarrow \{\mathbf{0}, \mathbf{0}, \dots, \mathbf{0}\}$

[iterate through samples of  $\mathcal{D}_M$ ]

**for**  $s \in \mathcal{D}_M$  **do**

[calculate the gradients for this sample]

$\mathcal{G}^s, \bar{g}^s \leftarrow \nabla \mathcal{L}(s; \mathcal{W}, \bar{w})$

[accumulate the gradients]

$\mathcal{G} \leftarrow \mathcal{G} + (\mathcal{G}^s)^\alpha$

**end for**

**for**  $G_i \in \mathcal{G}$  **do**

[top-k elements of the accumulated grads]

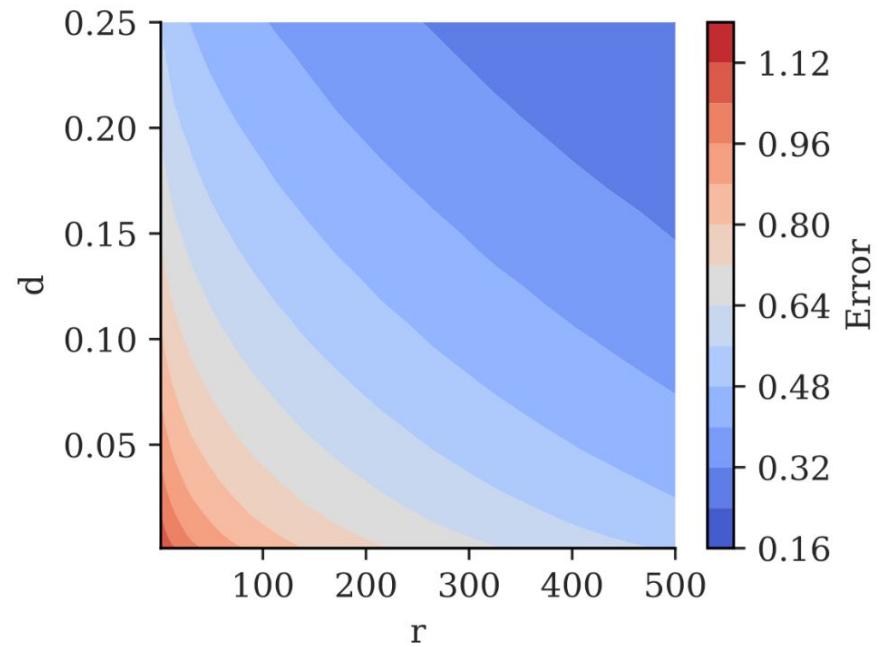
$\mathcal{M}_i \leftarrow TopK\text{-Mask}(G_i, d \times numel(G_i))$

**end for**

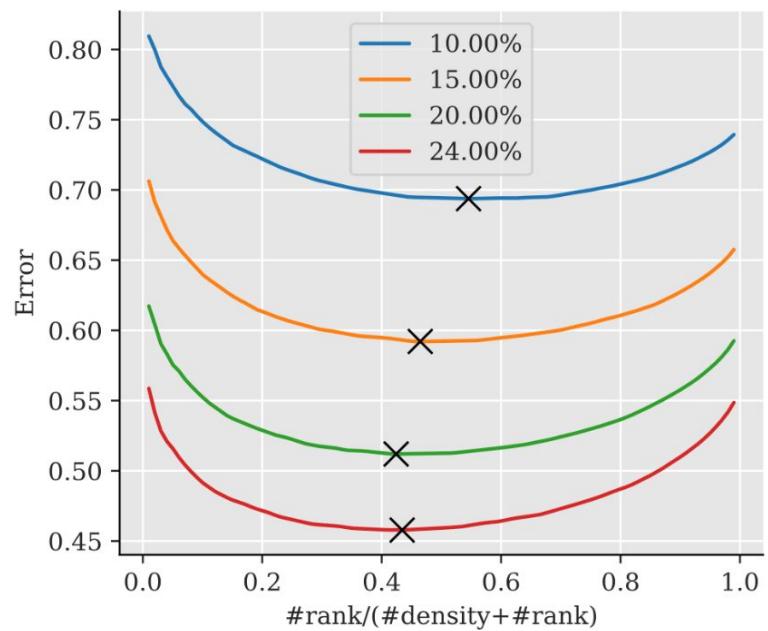
**return**  $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k\}$

---

# RoSA



(a)



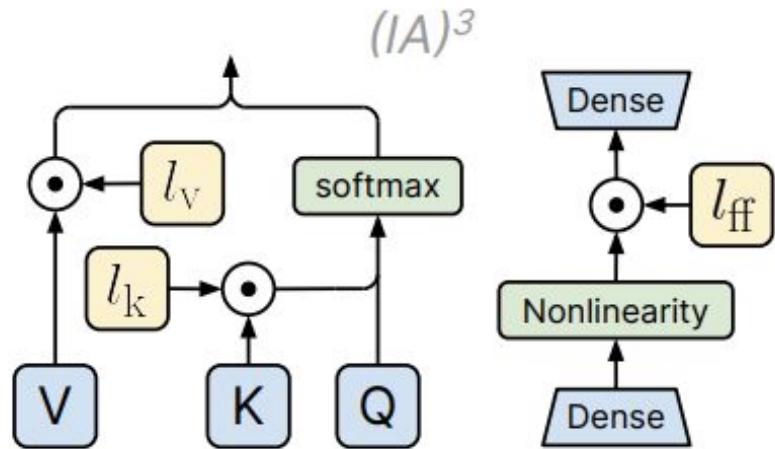
(b)

# RoSA

	#Params	Memory	GSM8k		ViGGO		SQL
	1 Epoch	Extended	1 Epoch	Extended	1 Epoch		
FFT	6.7 B	> 60 GB	<b>32.3</b>	<b>38.8</b>	<b>82.1</b>	<b>95.0</b>	<b>89.0</b>
LoRA $r = 16$	41.1 M	20.6 GB	28.4	<b>37.8</b>	90.5	95.8	88.7
RoSA $r = 12, d = 0.15\%$	41.0 M	20.3 GB	<b>31.2</b>	36.0	<b>95.0</b>	96.5	88.3
RoSA $r = 8, d = 0.3\%$	40.8 M	20.3 GB	29.2	37.5	94.5	<b>97.1</b>	77.6
RoSA $r = 4, d = 0.45\%$	40.6 M	20.3 GB	30.6	35.5	93.4	96.6	<b>89.7</b>
SpA $d = 0.6\%$	40.4 M	20.3 GB	26.2	29.5	72.6	89.8	83.2
LoRA $r = 32$	82.3 M	20.9 GB	29.6	36.2	87.0	96.8	<b>89.1</b>
RoSA $r = 24, d = 0.3\%$	81.9 M	20.6 GB	30.5	37.8	94.4	95.8	88.9
RoSA $r = 16, d = 0.6\%$	81.6 M	20.7 GB	<b>32.2</b>	<b>38.6</b>	<b>95.2</b>	<b>97.1</b>	88.3
RoSA $r = 8, d = 0.9\%$	81.2 M	20.7 GB	30.3	37.2	94.5	96.9	88.9
SpA $d = 1.2\%$	80.9 M	20.7 GB	21.9	29.9	45.8	95.7	74.2
LoRA $r = 64$	164.5 M	21.7 GB	27.4	35.5	76.9	95.0	88.7
RoSA $r = 48, d = 0.6\%$	163.8 M	21.3 GB	30.5	38.2	93.0	96.6	88.1
RoSA $r = 32, d = 1.2\%$	163.1 M	21.4 GB	32.2	36.2	93.4	<b>97.3</b>	<b>89.2</b>
RoSA $r = 16, d = 1.8\%$	162.4 M	21.5 GB	<b>32.8</b>	<b>38.4</b>	<b>95.1</b>	96.5	84.6
SpA $d = 2.4\%$	161.7 M	21.8 GB	29.6	37.2	92.3	95.7	87.8

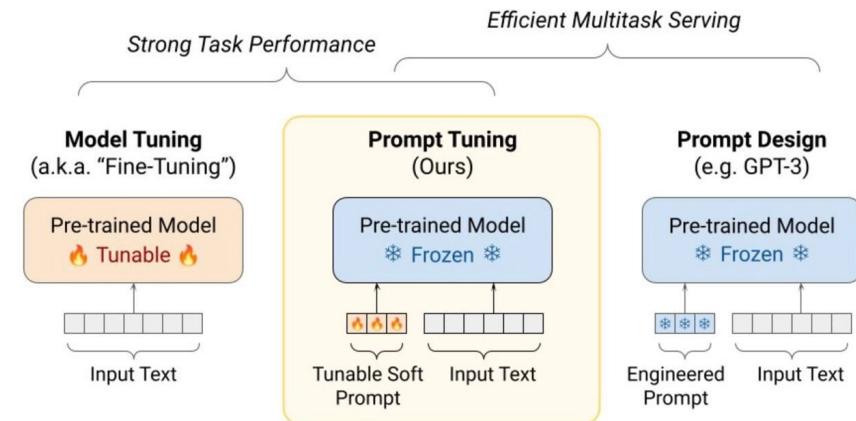
# IA3 (Infused Adapter by Inhibiting and Amplifying Inner Activations)

- 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.



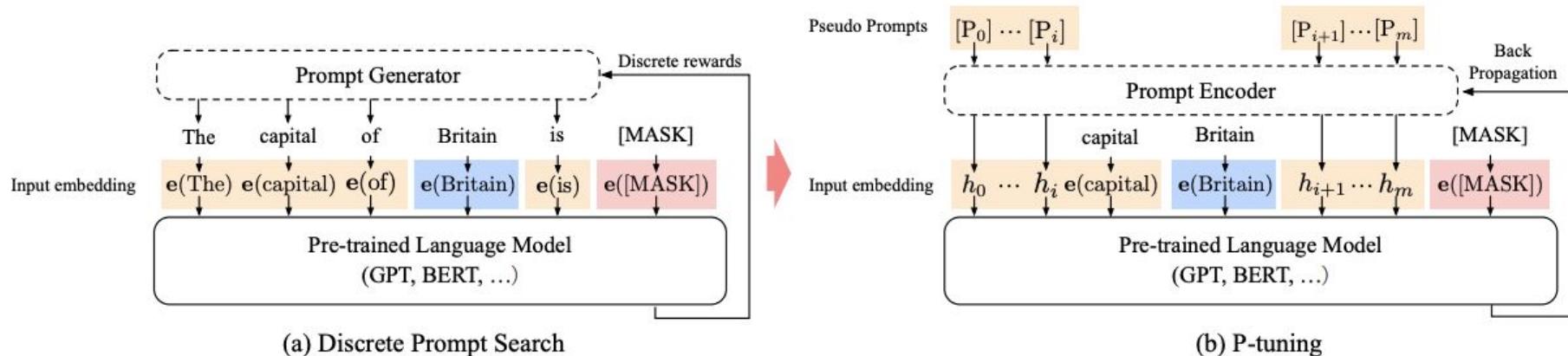
# Prompt tuning

- 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).



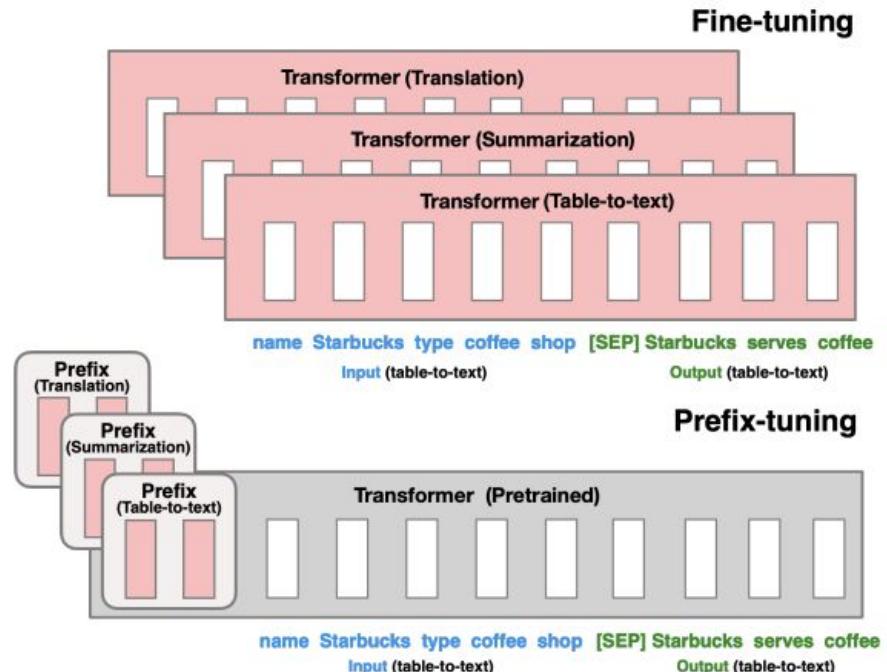
# P-tuning

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



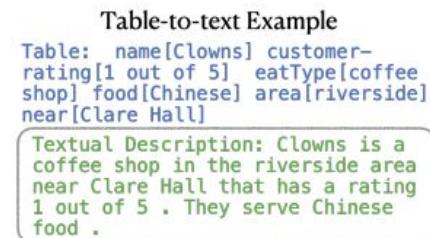
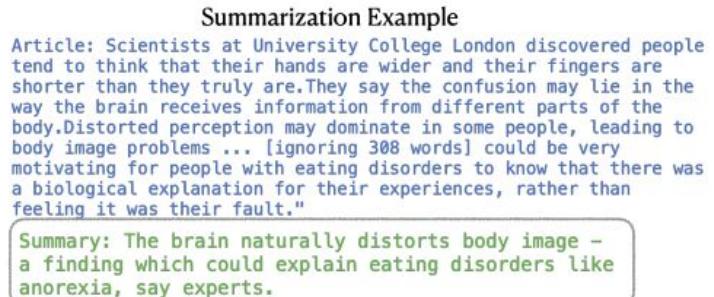
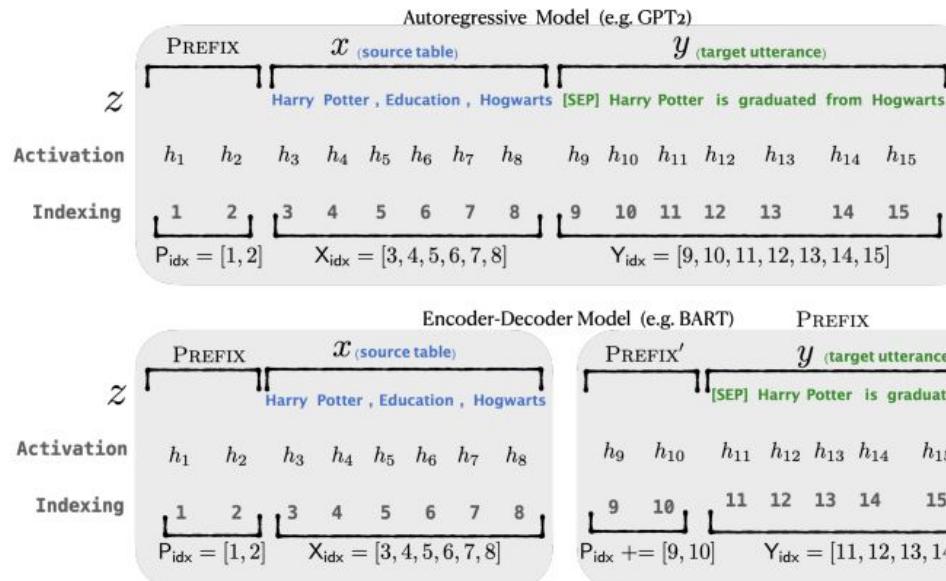
# Prefix-tuning

- 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



# Что сегодня изучили?

- Обсудили, зачем оптимизировать обучение и инференс
- Рассмотрели способы квантизации модели
- Изучили несколько методов PEFT:
  - LoRA
  - QLoRA
  - DoRA
  - RoSA
  - IA3
  - Prompt-tuning
  - Prefix-tuning
  - P-tuning



## Что еще почитать?

- [LLM.int8\(\): 8-bit Matrix Multiplication for Transformers at Scale](#)
- [Optimal Brain Damage](#)
- [Second order derivatives for network pruning: Optimal Brain Surgeon](#)
- [Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning](#)
- [GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers](#)
- [LoRA: Low-Rank Adaptation of Large Language Models](#)
- [QLoRA: Efficient Finetuning of Quantized LLMs](#)
- [DoRA: Weight-Decomposed Low-Rank Adaptation](#)
- [Sparse Low-rank Adaptation of Pre-trained Language Models](#)
- [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](#)

Спасибо  
за внимание!

