LLM Acceleration

2023.4.6

Outline

- Challenges
- Inference
- Fine-tuning

Challenges

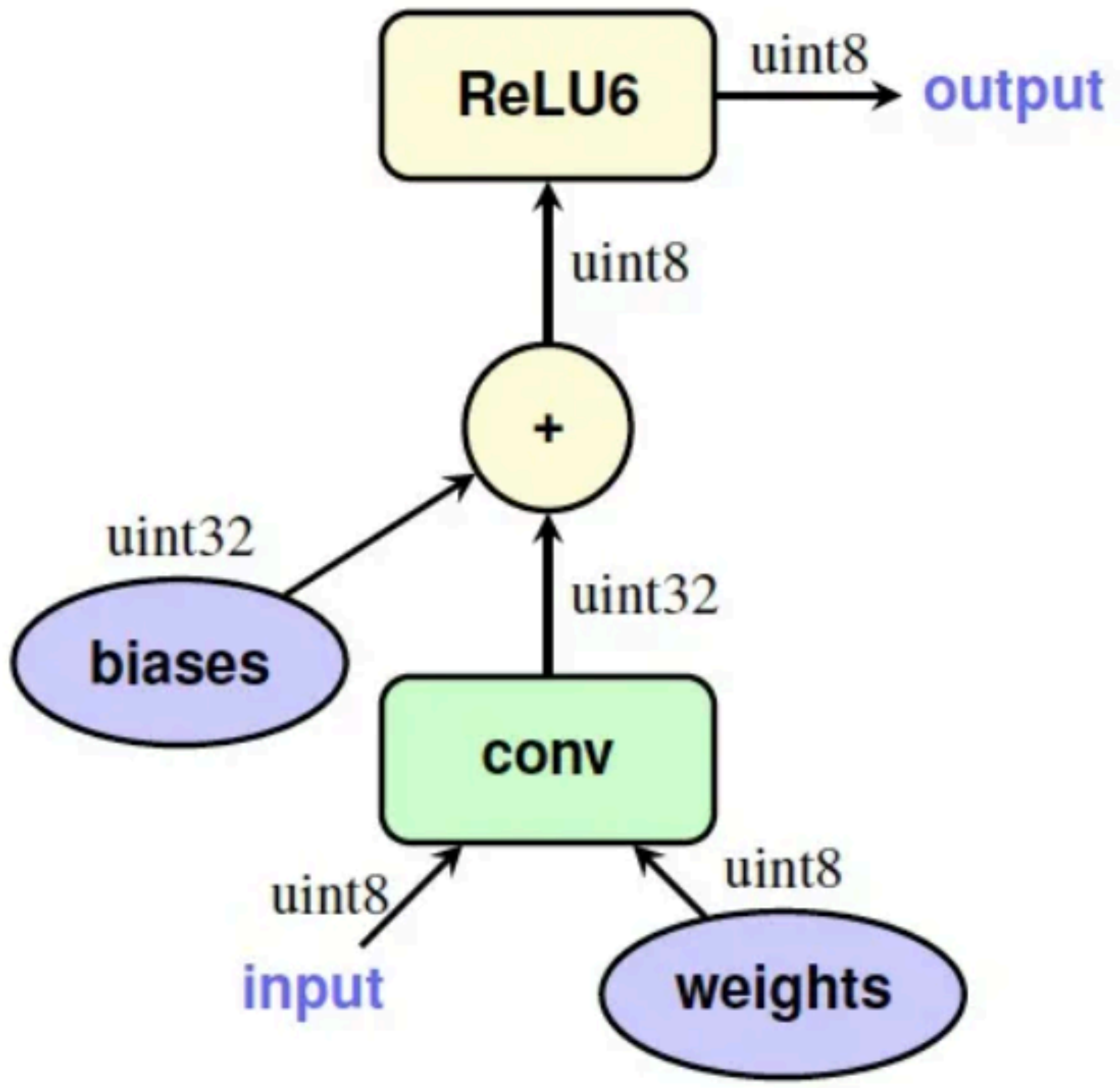
- Large memory footprint
    - Load model weights to GPU
    - KV Cache
- Long inference Latency

Inference Acceleration

- Quantization
- Pruning
- Off-loading
- Distillation

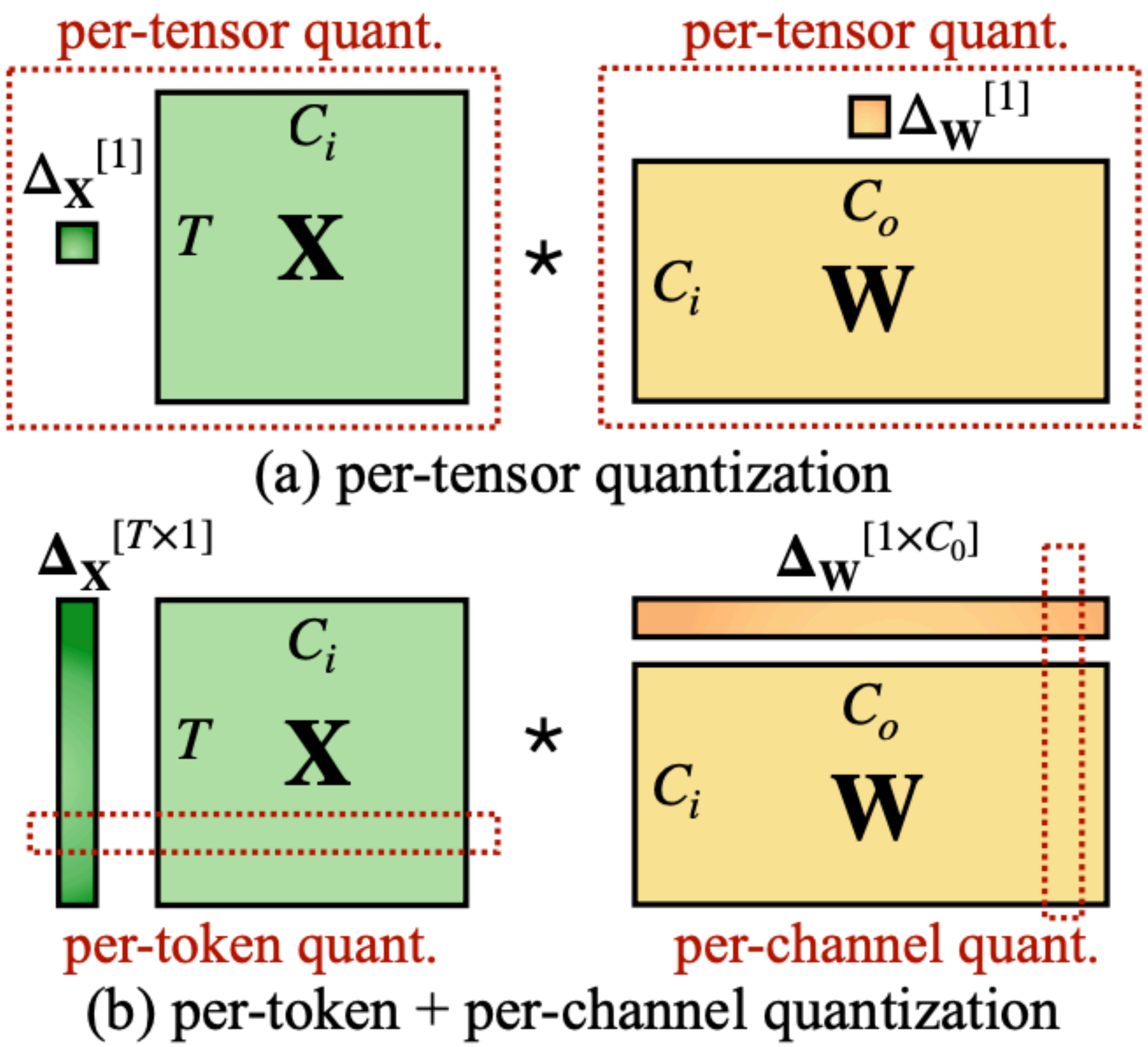
Quantization

- Converting the weights and activations from fp32 to low bit width
- Post-Training Quantization (PTQ)
- Quantization-Aware Training (QAT)



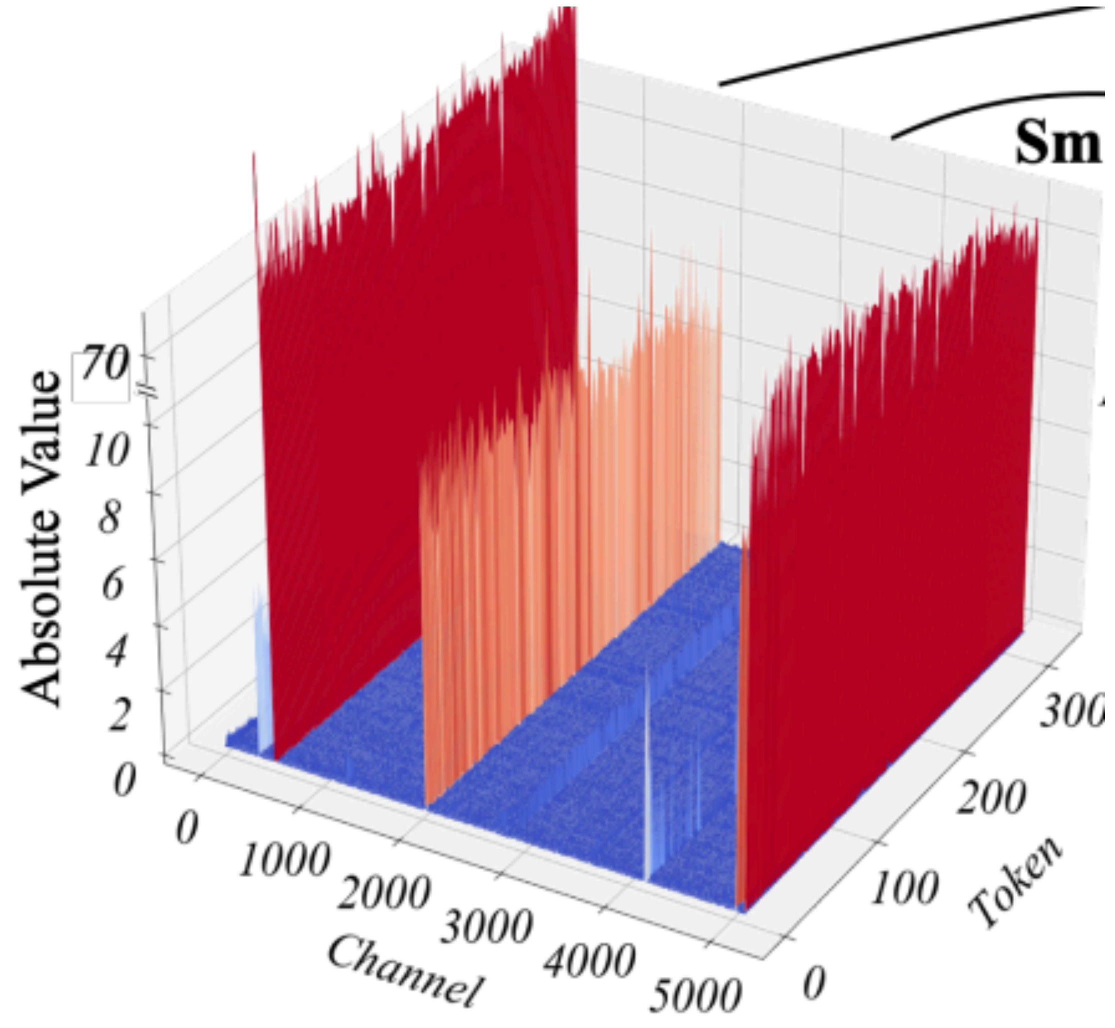
Quantization

$$\bar{\mathbf{X}}^{\mathrm{INT8}} = \lceil \frac{\mathbf{X}^{\mathrm{FP16}}}{\Delta} \rfloor, \quad \Delta = \frac{\max(|\mathbf{X}|)}{2^{N-1}-1},$$



Challenges of Quantization on LLM

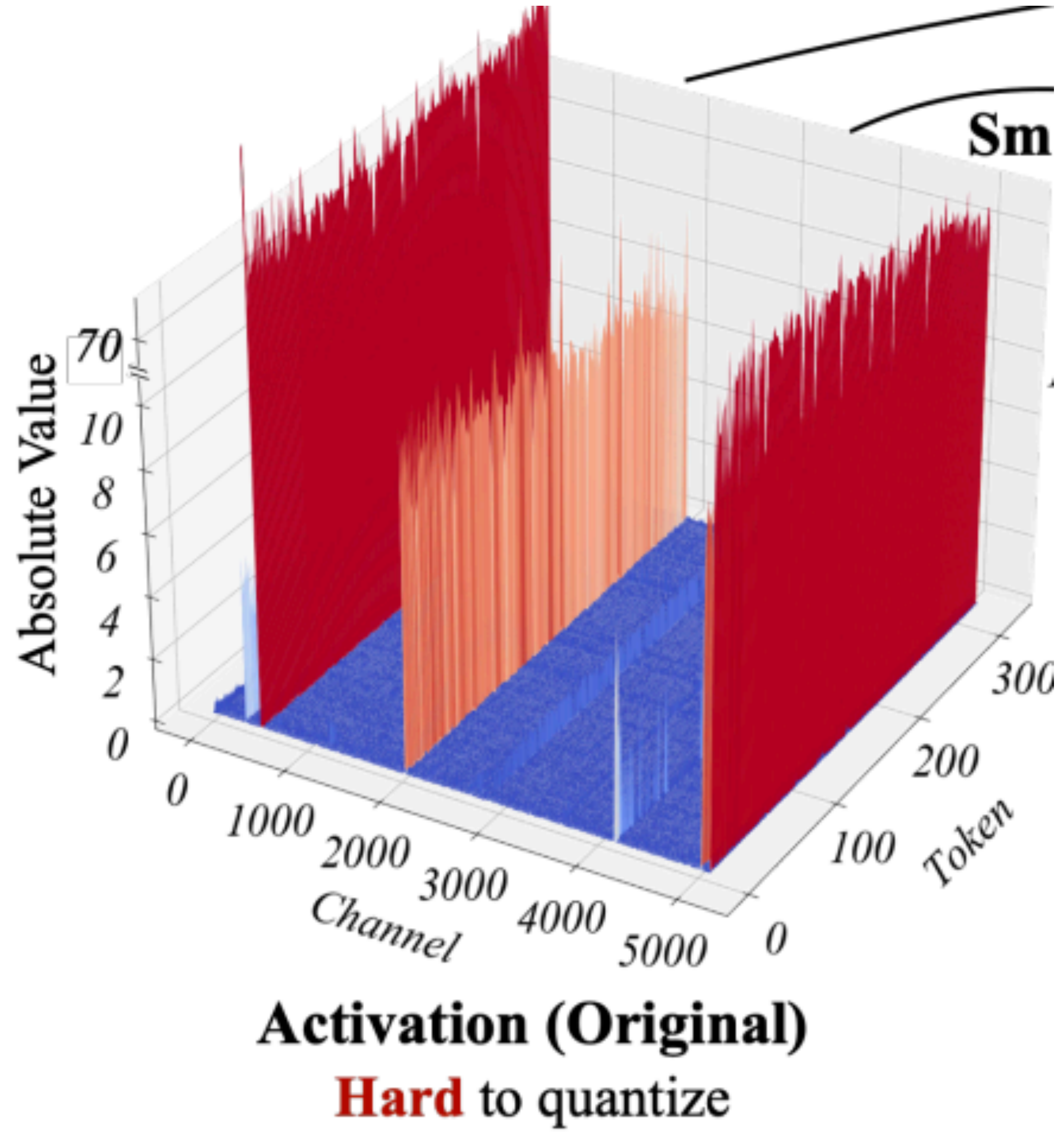
- The large number of parameters: we prefer PTQ on LLM
- Large outliers: significant degradation of quantization resolution



Configuration	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE
FP32	57.27	93.12	88.36	89.09	89.72	84.91	91.58	70.40	**83.06**
W8A8	54.74	92.55	88.53	81.02	83.81	50.31	52.32	64.98	**71.03**
W32A8	56.70	92.43	86.98	82.87	84.70	52.80	52.44	53.07	**70.25**
W8A32	58.63	92.55	88.74	89.05	89.72	84.58	91.43	71.12	**83.23**

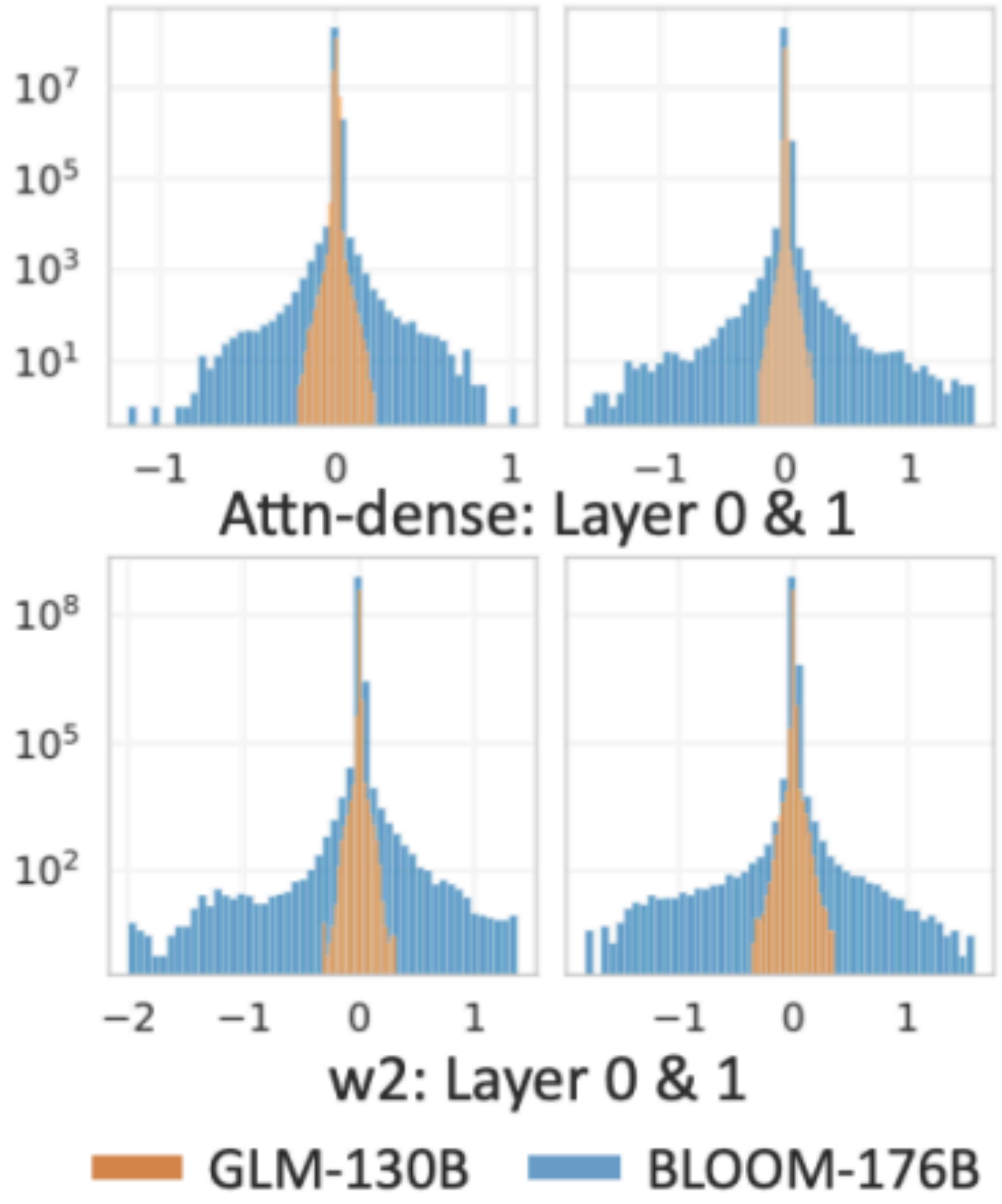
Features of Open Source GPT-like LLM

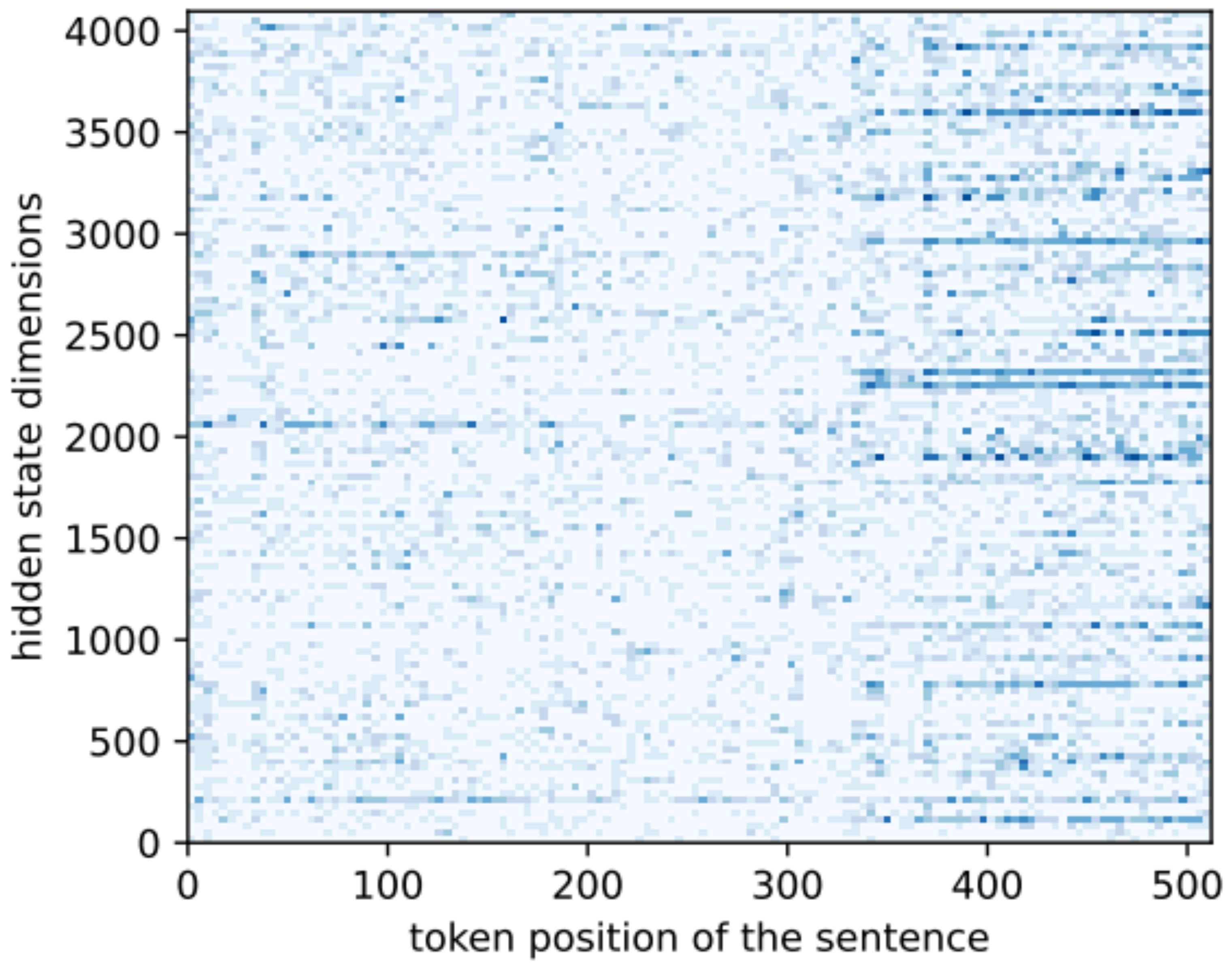
- OPT-175B
  with significant outliers, inadequate training, significant redundancy
- BLOOM-176B
  less outliers, similar performance with OPT-175B



Features of Open Source GPT-like LLM

- GLM-130B
  Weight: INT4
  Activations: 30% are outliers





Features of Open Source GPT-like LLM

- LLaMA-65B / Alpaca-65B
  well-trained, little outliers in both weights and activations

```
layers.10.attention.wq torch.S
tensor([[ 7.8828, -0.0096],
        [ 1.4023,  0.1812],
        [ 0.5474,  0.1475],
        [ 0.0201,  0.1840],
        [-0.2106,  0.2035],
        [-0.0312,  0.1938],
        [ 0.0599,  0.0648],
        [ 0.2041,  0.0465],
        [ 0.1179,  0.1434],
```

```
model.decoder.layers.21.self_attn.k_
tensor([[     61.5625,       0.0128],
        [     44.0000,       0.6533],
        [     24.0625,       0.1434],
        [     22.3750,       1.4014],
        [     33.4375,       0.1888],
        [     36.7188,      -0.8003],
        [     38.3125,       0.4158],
        [     42.5000,       0.2886],
        [     29.8438,      -0.3481],
```

Types of Quantization

- Quantize both weight and activation
  - Advantages: accelerate inference and reduce memory cost
  - Disadvantages: activation is difficult to maintain precision in low bit width
  - Related works:
    - LLM.int8(): 99.9% W8A8 LLM loseless
    - Outlier Suppression: W6A6 BERT loseless
    - SmoothQuant: W8A8 LLM loseless

Types of Quantization

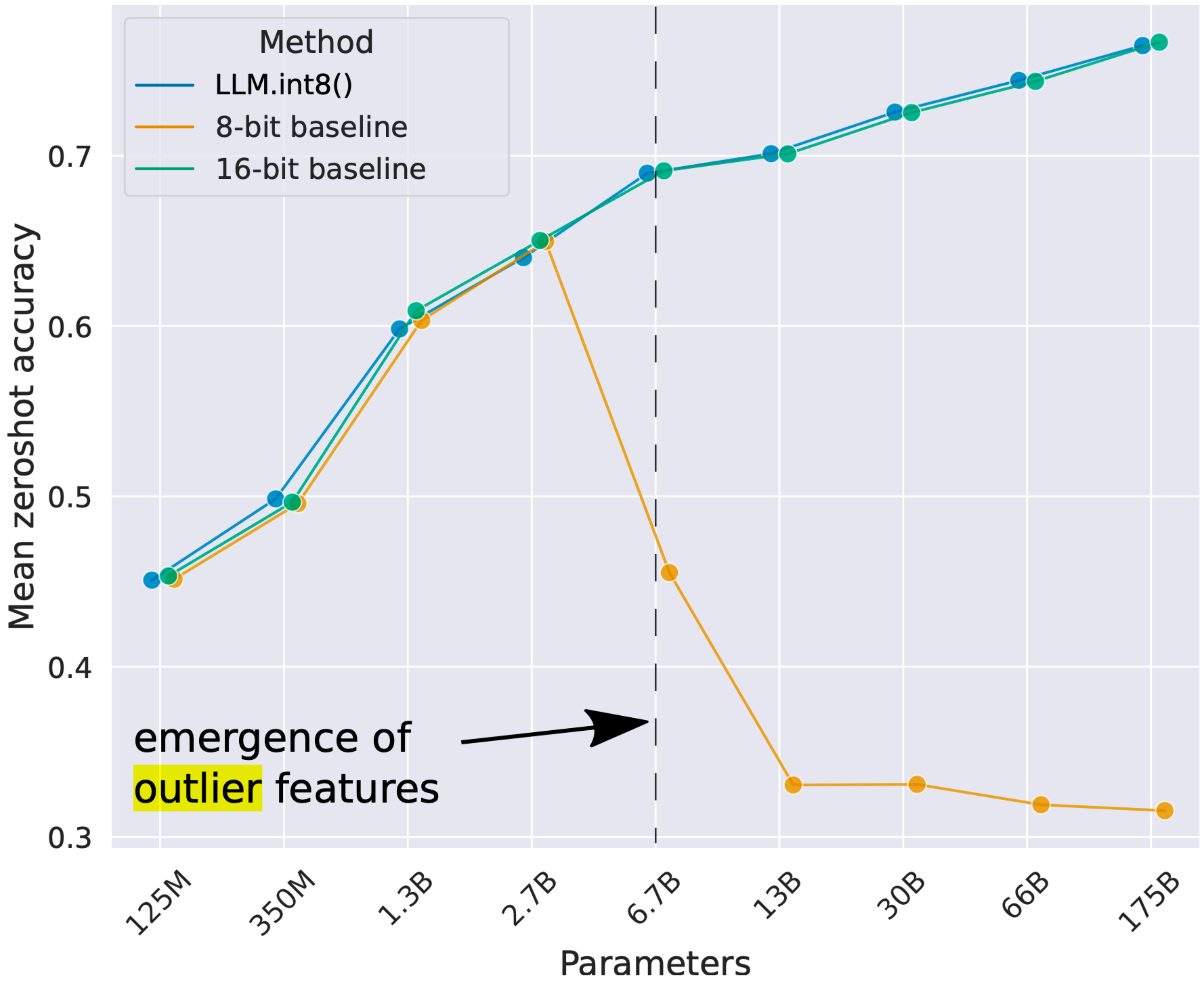
- Weight-only quantization
  - Advantages: Lower the memory requirements (People currently prefer)
  - Disadvantages: Fairly slow inference speed
  - Related works:
    - OBQ
    - GPTQ (OPT-175B 3bits loseless)
    - llama.cpp (LLaMA-65B on Mbp, LLaMA-7B on Raspberry Pi 4G)
  - Is it possible to make hardware support mixed precision multiplication?

LLM.int8() (NIPS 2022)

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

- 1. As the model size grows to billions of parameters, outliers start to emerge in all transformer layers, causing failure of simple low-bit quantization.

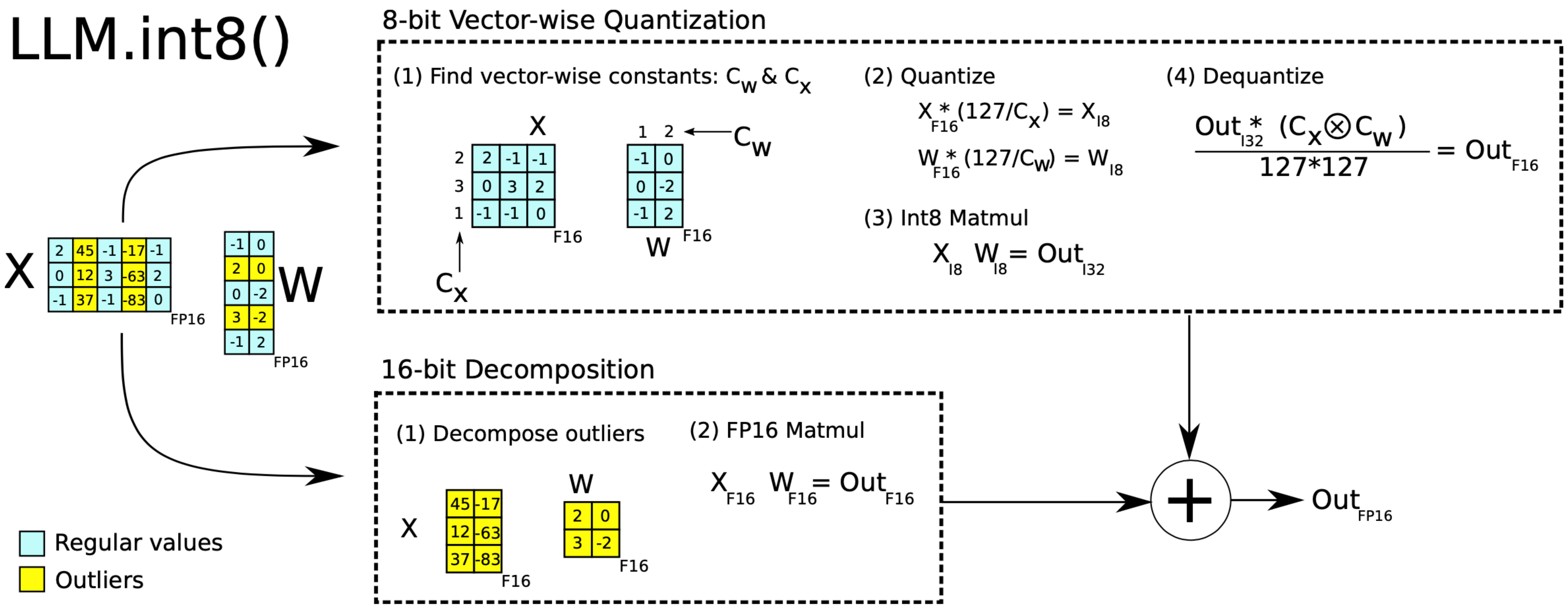
- 2. Outliers persist in fixed channels



LLM.int8() (NIPS 2022)

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

- Isolate the outlier feature dimensions into a 16-bit matrix multiplication while still more than 99.9% of values are multiplied in 8-bit



LLM.int8() (NIPS 2022)

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

- Perform inference in INT8 LLMs with up to 175B parameters without any performance degradation
- Only the 13B and 175B models have speedups

SmoothQuant (ICML 2023)

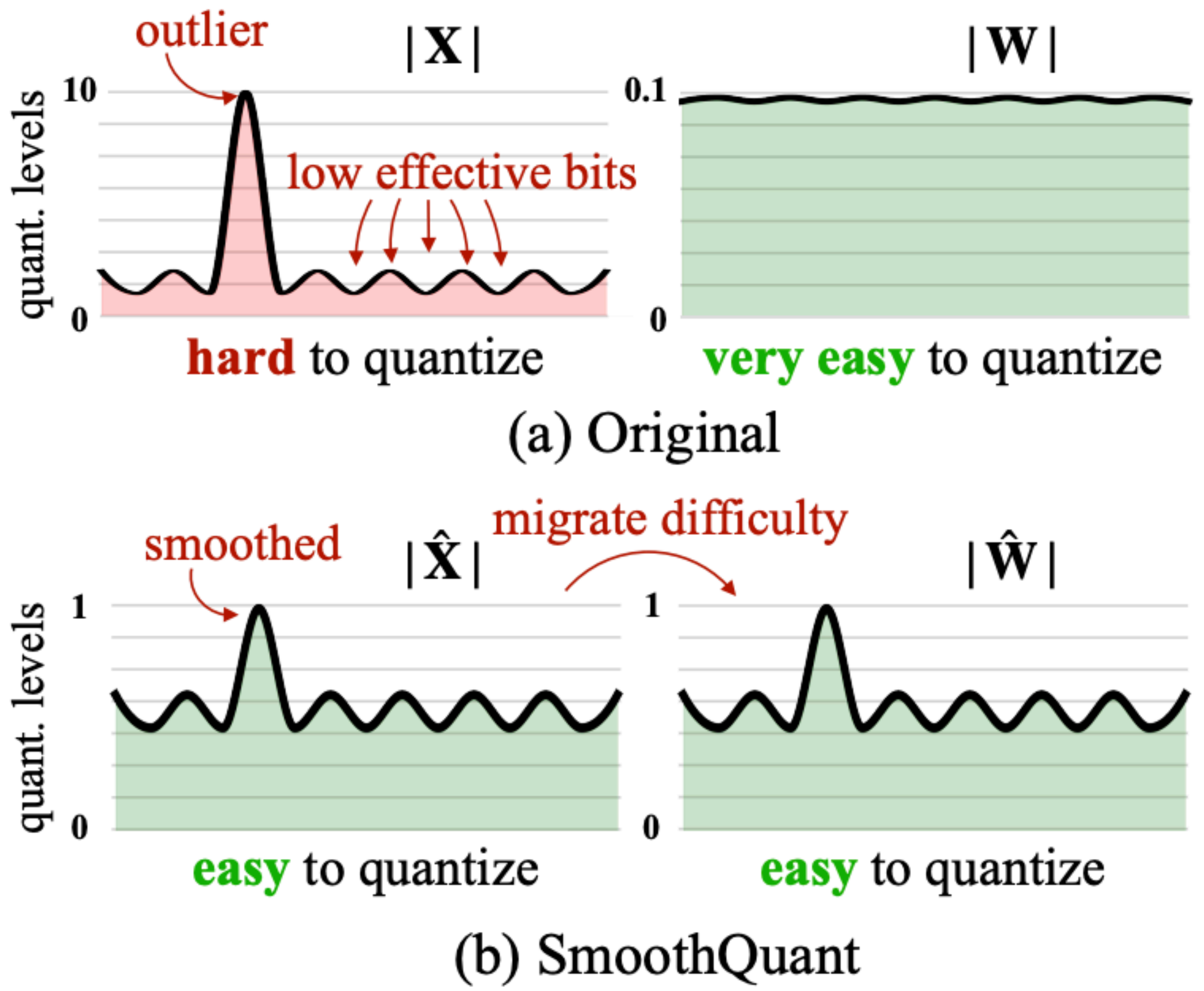
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

	LLM (100B+) Accuracy	Hardware Efficiency
ZeroQuant	✗	✔
Outlier Suppression	✗	✔
`LLM.int8()`	✔	✗
SmoothQuant	✔	✔

SmoothQuant (ICML 2023)

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

- 1. Activations are harder to quantize than weights
- 2. Outliers make activation quantization difficult
- 3. Outliers persist in fixed channels.
- Offline migrates the quantization difficulty from activations to weights



(a) Original

(b) SmoothQuant

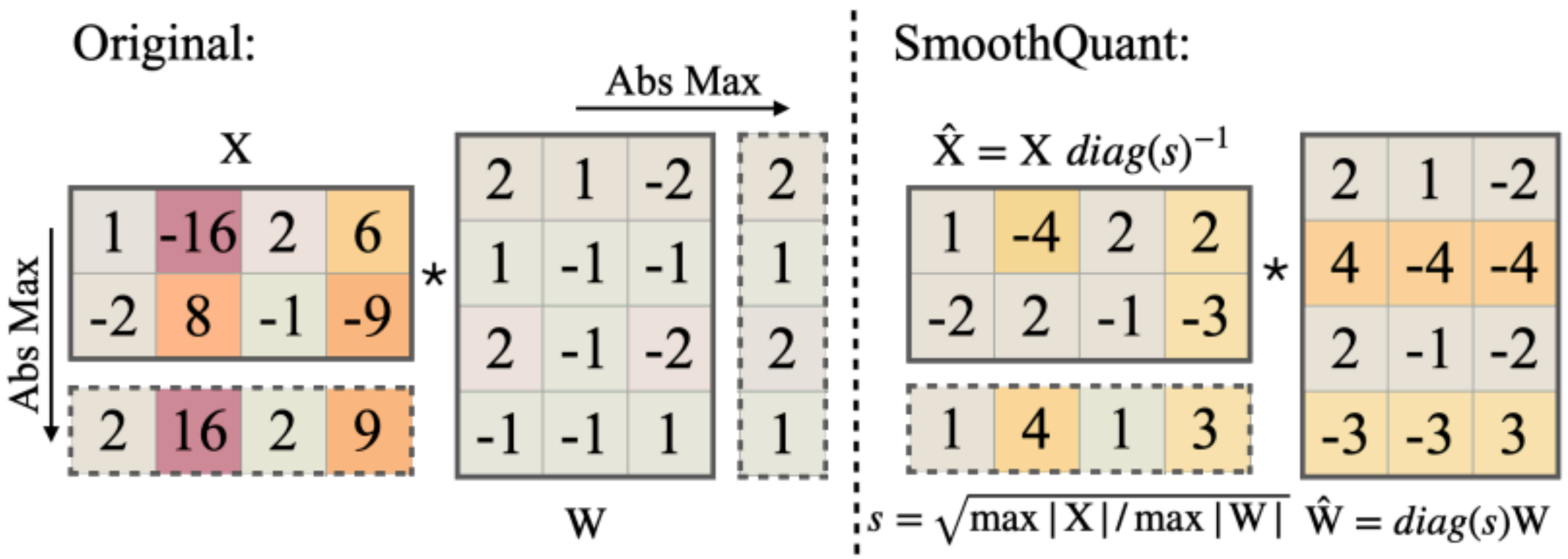
SmoothQuant (ICML 2023)

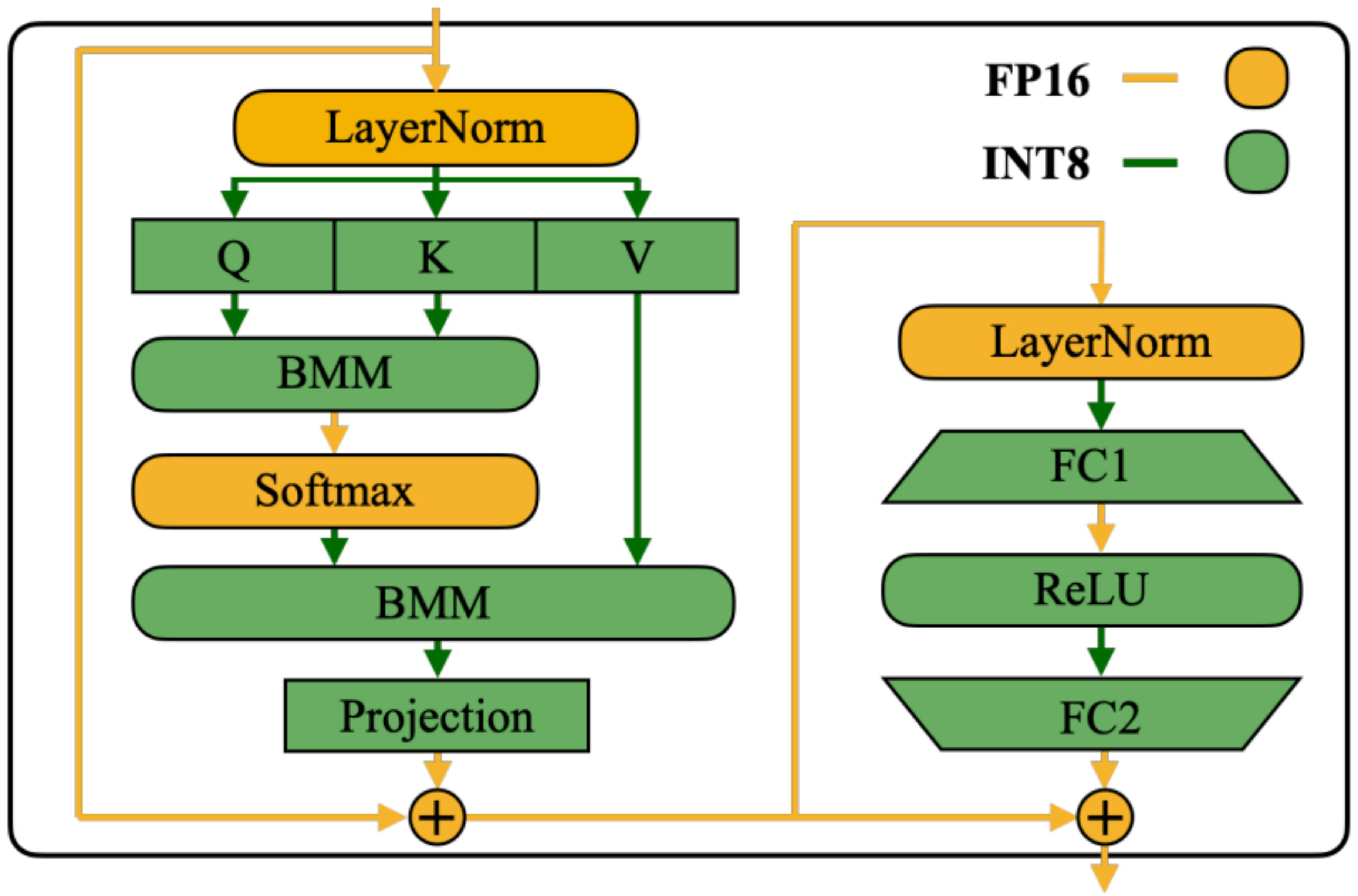
SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

- “smooth” the input activation by dividing it by a per-channel smoothing factor

$$\mathbf{Y} = (\mathbf{X}\mathrm{diag}(\mathbf{s})^{-1}) \cdot (\mathrm{diag}(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha}$$

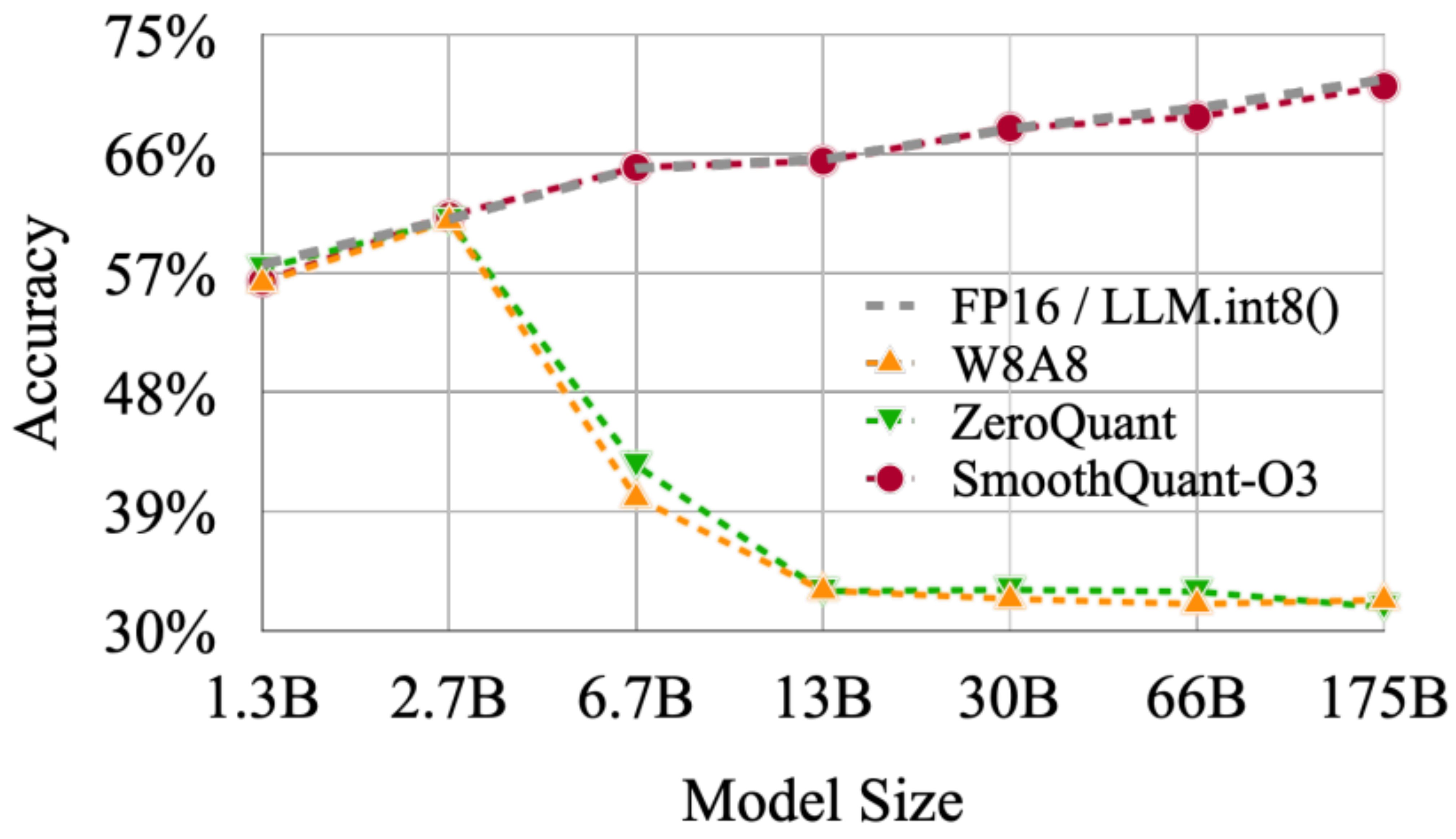




SmoothQuant (ICML 2023)

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

Method	OPT-175B	BLOOM-176B	GLM-130B*
FP16	71.6%	68.2%	73.8%
W8A8	32.3%	64.2%	26.9%
ZeroQuant	31.7%	67.4%	26.7%
`LLM.int8()`	71.4%	68.0%	73.8%
Outlier Suppression	31.7%	54.1%	63.5%
SmoothQuant-O1	**71.2%**	68.3%	**73.7%**
SmoothQuant-O2	71.1%	**68.4%**	72.5%
SmoothQuant-O3	71.1%	67.4%	72.8%



SmoothQuant (ICML 2023)

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

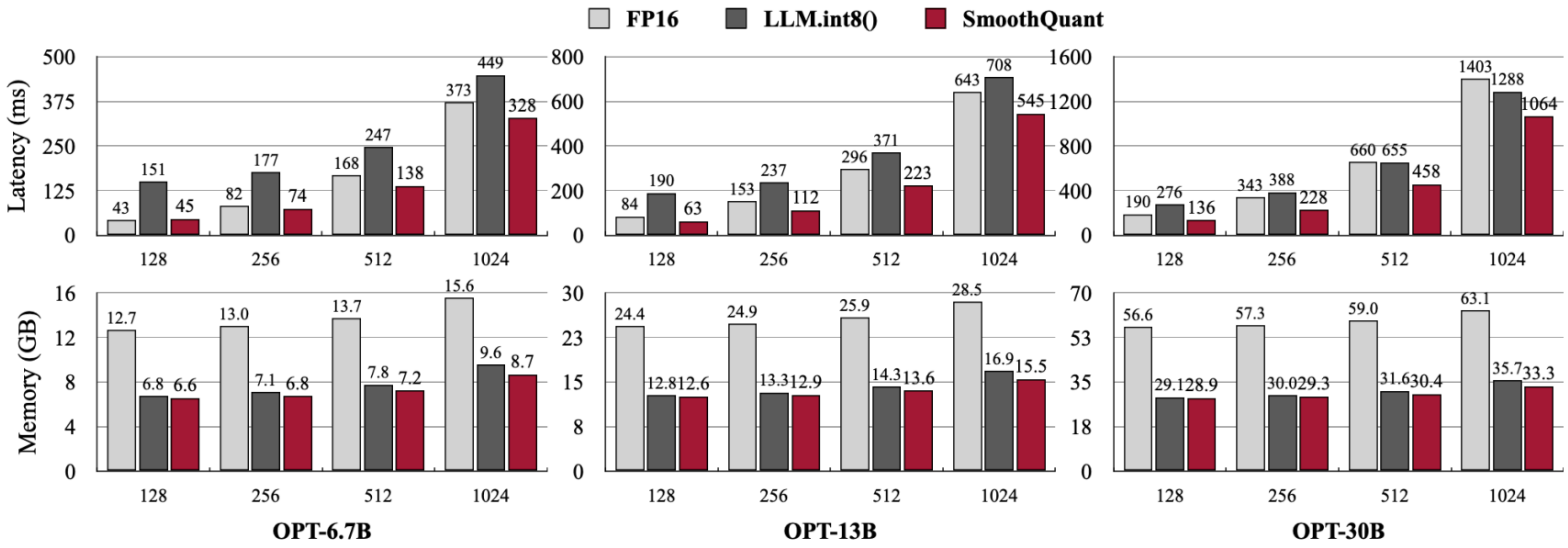
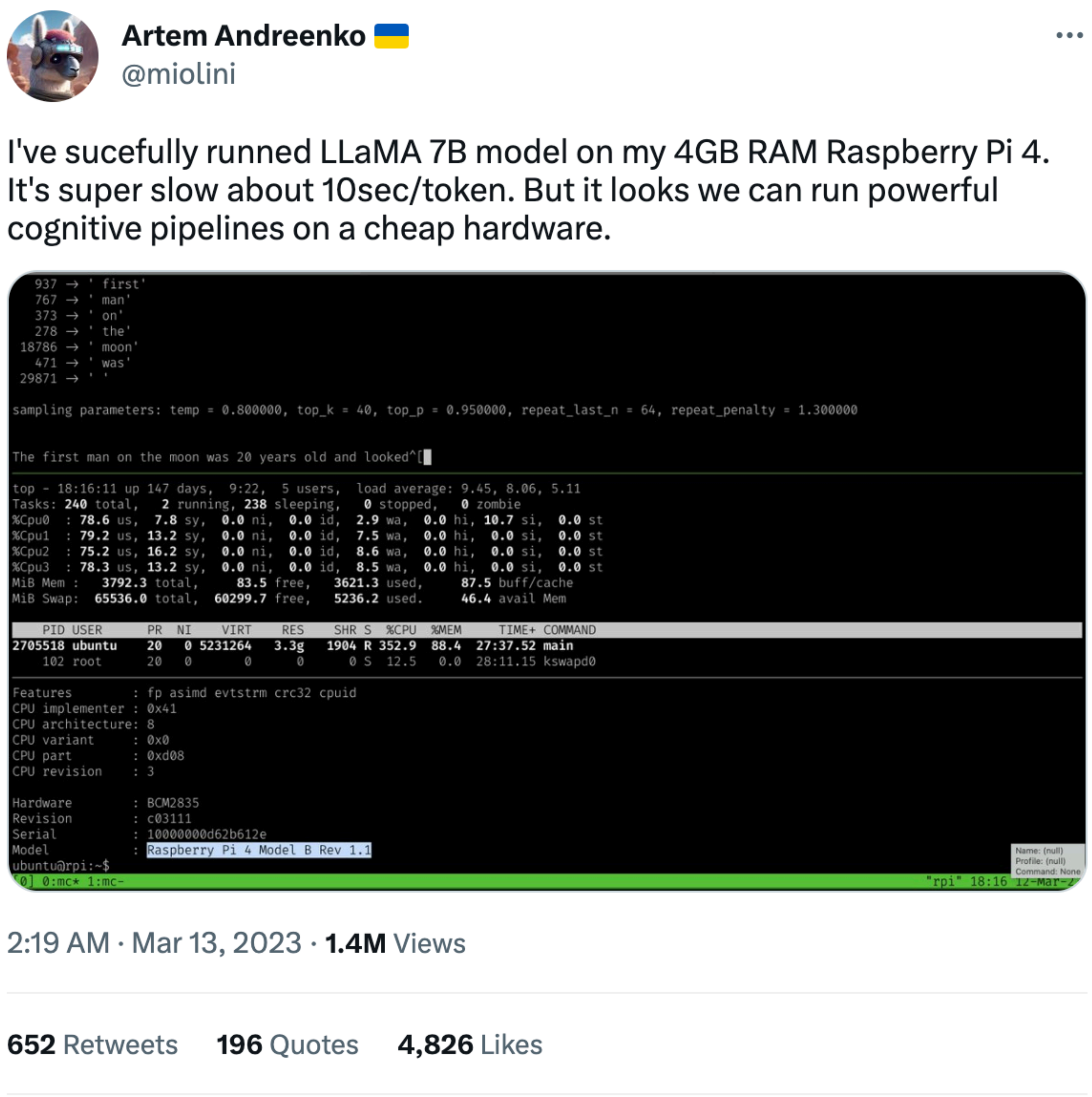


Figure 7: The PyTorch implementation of SmoothQuant-O3 achieves up to **1.51**× speedup and **1.96**× memory saving for OPT models on a single NVIDIA A100-80GB GPU, while `LLM.int8()` slows down the inference in most cases.

llama.cpp (15.2k stars on Github)

- 4-bit quantization / INT4-FP16 mixed precision
- Run on CPU
  Run LLaMA-13B on 64GB M2 MacBook Pro
  Run LLaMA-7B on 4GB RAM Raspberry Pi 4
- 10 sec/token

model	original size	quantized size (4-bit)
7B	13 GB	3.9 GB
13B	24 GB	7.8 GB
30B	60 GB	19.5 GB
65B	120 GB	38.5 GB



llama.cpp (15.2k stars on Github)

- Plain C/C++ implementation without dependencies
- 4-bit quantization for 99% normal values
- FP16 for 1% outliers
- Use CSC / CSR to mark the positions of outliers
- Very slow but simple and work method

OBQ (NIPS 2022)

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning

- Find a matrix of quantized weights W^ which minimizes the squared error

$$\text{argmin}_{\widehat{\mathbf{W}}_\ell} \quad ||\mathbf{W}_\ell \mathbf{X}_\ell - \widehat{\mathbf{W}}_\ell \mathbf{X}_\ell||_2^2$$

- Taylor approximation provides explicit formulas for the optimal single weight to remove, as well as the optimal update of the remaining weights which would compensate for the removal.

$$w_q = \text{argmin}_{w_q} \frac{(\text{quant}(w_q) - w_q)^2}{[\mathbf{H}_F^{-1}]_{qq}}, \quad \boldsymbol{\delta}_F = -\frac{w_q - \text{quant}(w_q)}{[\mathbf{H}_F^{-1}]_{qq}} \cdot (\mathbf{H}_F^{-1})_{:,q}.$$

OBQ (NIPS 2022)

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning

- OBQ handles each row independently in parallel, quantizing one weight at a time while always updating all not-yet-quantized weights, in order to compensate for the error incurred by quantizing a single weight

$$w_q = \text{argmin}_{w_q} \frac{(\text{quant}(w_q) - w_q)^2}{[\mathbf{H}_F^{-1}]_{qq}}, \quad \boldsymbol{\delta}_F = -\frac{w_q - \text{quant}(w_q)}{[\mathbf{H}_F^{-1}]_{qq}} \cdot (\mathbf{H}_F^{-1})_{:,q}.$$

- Gaussian elimination

$$\mathbf{H}_{-q}^{-1} = \left(\mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}_{:,q}^{-1} \mathbf{H}_{q,:}^{-1}\right)_{-p}.$$

OBQ (NIPS 2022)

Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning

Algorithm 1 Prune $k \leq d_{\text{col}}$ weights from row $\mathbf{w}$ with inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top)^{-1}$ according to OBS in $O(k \cdot d_{\text{col}}^2)$ time.

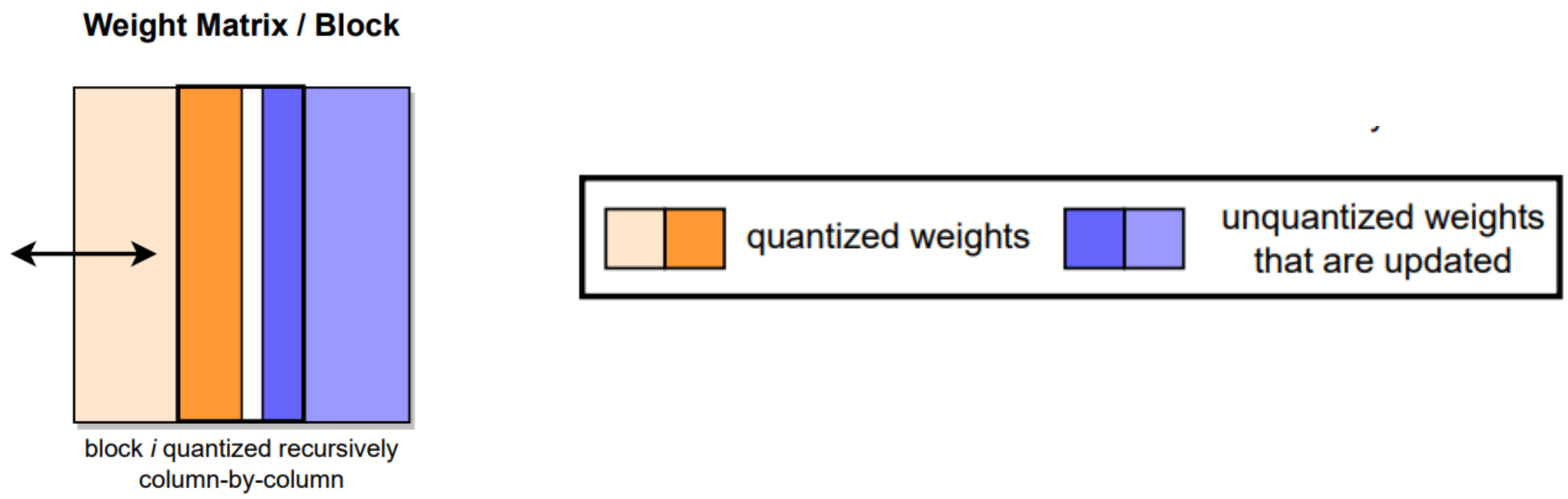
$M = \{1, \ldots, d_{\text{col}}\}$
for $i = 1, \ldots, k$ **do**
  $p \leftarrow \text{argmin}_{p \in M} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p^2$
  $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,p} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p$
  $\mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:}$
  $M \leftarrow M - \{p\}$
end for

- Total time complexity: O(d_row * d_col^3)

GPTQ (ICLR 2023)

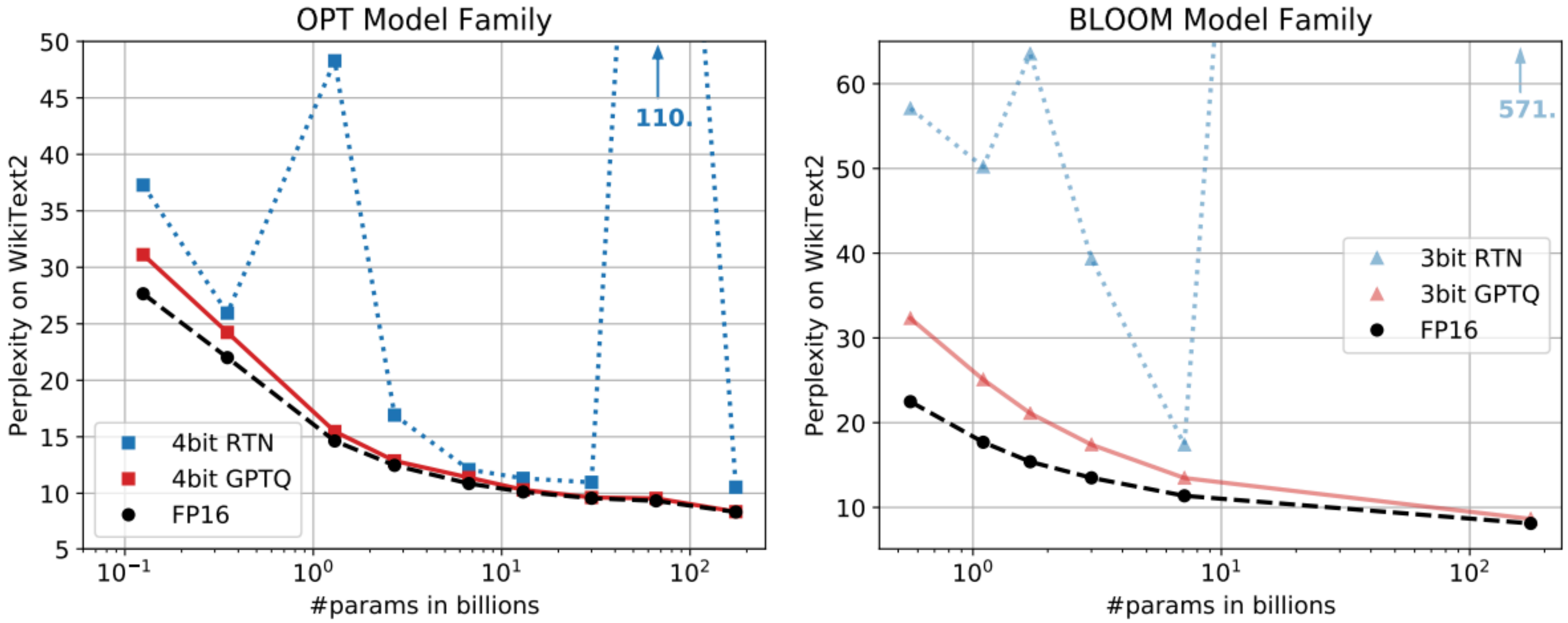
GPTQ: ACCURATE POST-TRAINING QUANTIZATION FOR GENERATIVE PRE-TRAINED

- Any fixed quantization order may perform well, especially on large models
- So we can quantize the weights of all rows in the same order
- Reduce the time complexity
  from O(d_row * d_col^3) to O(max{d_row * d_col^2, d_row^3})



GPTQ (ICLR 2023)

GPTQ: ACCURATE POST-TRAINING QUANTIZATION FOR GENERATIVE PRE-TRAINED



OPT	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
full	16	27.65	22.00	14.63	12.47	10.86	10.13	9.56	9.34	8.34
RTN	4	37.28	25.94	48.17	16.92	12.10	11.32	10.98	110	10.54
GPTQ	4	**31.12**	**24.24**	**15.47**	**12.87**	**11.39**	**10.31**	**9.63**	**9.55**	**8.37**
RTN	3	1.3e3	64.57	1.3e4	1.6e4	5.8e3	3.4e3	1.6e3	6.1e3	7.3e3
GPTQ	3	**53.85**	**33.79**	**20.97**	**16.88**	**14.86**	**11.61**	**10.27**	**14.16**	**8.68**

Table 3: OPT perplexity results on WikiText2.

Model	FP16	g128	g64	g32	3-bit
OPT-175B	8.34	9.58	9.18	8.94	8.68
BLOOM	8.11	9.55	9.17	8.83	8.64

Table 7: 2-bit GPTQ quantization results with varying group-sizes; perplexity on WikiText2.

GPTQ (ICLR 2023)

GPTQ: ACCURATE POST-TRAINING QUANTIZATION FOR GENERATIVE PRE-TRAINED

- GPT for LLaMA: https://github.com/qwopqwop200/GPTQ-for-LLaMa
- GPTQ does not improve the quantized performance on LLaMA (sometimes even worse)
- Trick: quantizing columns in order of decreasing activation size

LLaMA-65B	Bits	group-size	memory(MiB)	Wikitext2
FP16	16	-	OOM	3.53
RTN	4	-	-	3.92
GPTQ	4	-	OOM	3.84
GPTQ	4	128	OOM	3.65
RTN	3	-	-	10.59
GPTQ	3	-	OOM	5.04
GPTQ	3	128	OOM	4.17

Inference Acceleration

- Quantization
- Pruning
- Off-loading
- Distillation

SparseGPT

SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot

- Find a sparsity mask M which minimizes the squared error

$$\text{argmin}_{\text{mask } \mathbf{M}_\ell, \widehat{\mathbf{W}}_\ell} ||\mathbf{W}_\ell \mathbf{X}_\ell - (\mathbf{M}_\ell \odot \widehat{\mathbf{W}}_\ell)\mathbf{X}_\ell||_2^2.$$

- The optimal values of all weights in the mask can be calculated exactly by solving the sparse reconstruction problem corresponding to each matrix row

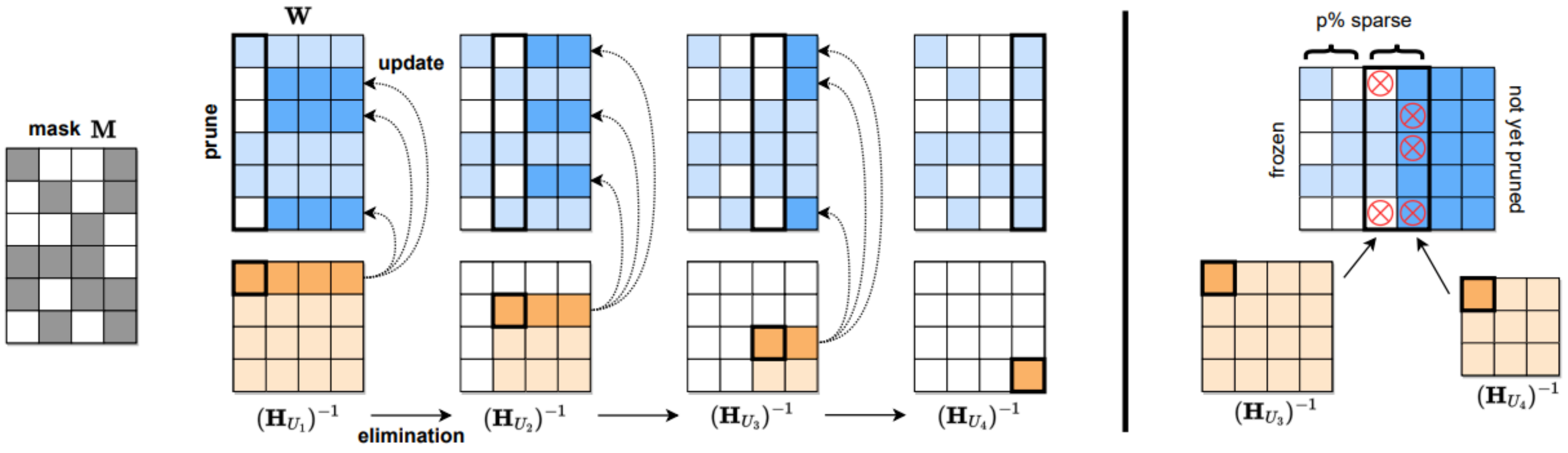
$$w_p = \text{argmin}_{w_p} \frac{w_p^2}{[\mathbf{H}^{-1}]_{pp}}, \quad \boldsymbol{\delta_p} = -\frac{w_p}{[\mathbf{H}^{-1}]_{pp}} \cdot \mathbf{H}^{-1}_{:,p},$$

SparseGPT

SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot

Gaussian elimination:

$$(\mathbf{H}_{U_{j+1}})^{-1} = \left(\mathbf{B} - \frac{1}{[\mathbf{B}]_{11}} \cdot \mathbf{B}_{:,1}\mathbf{B}_{1,:}\right)_{2:,2:},$$



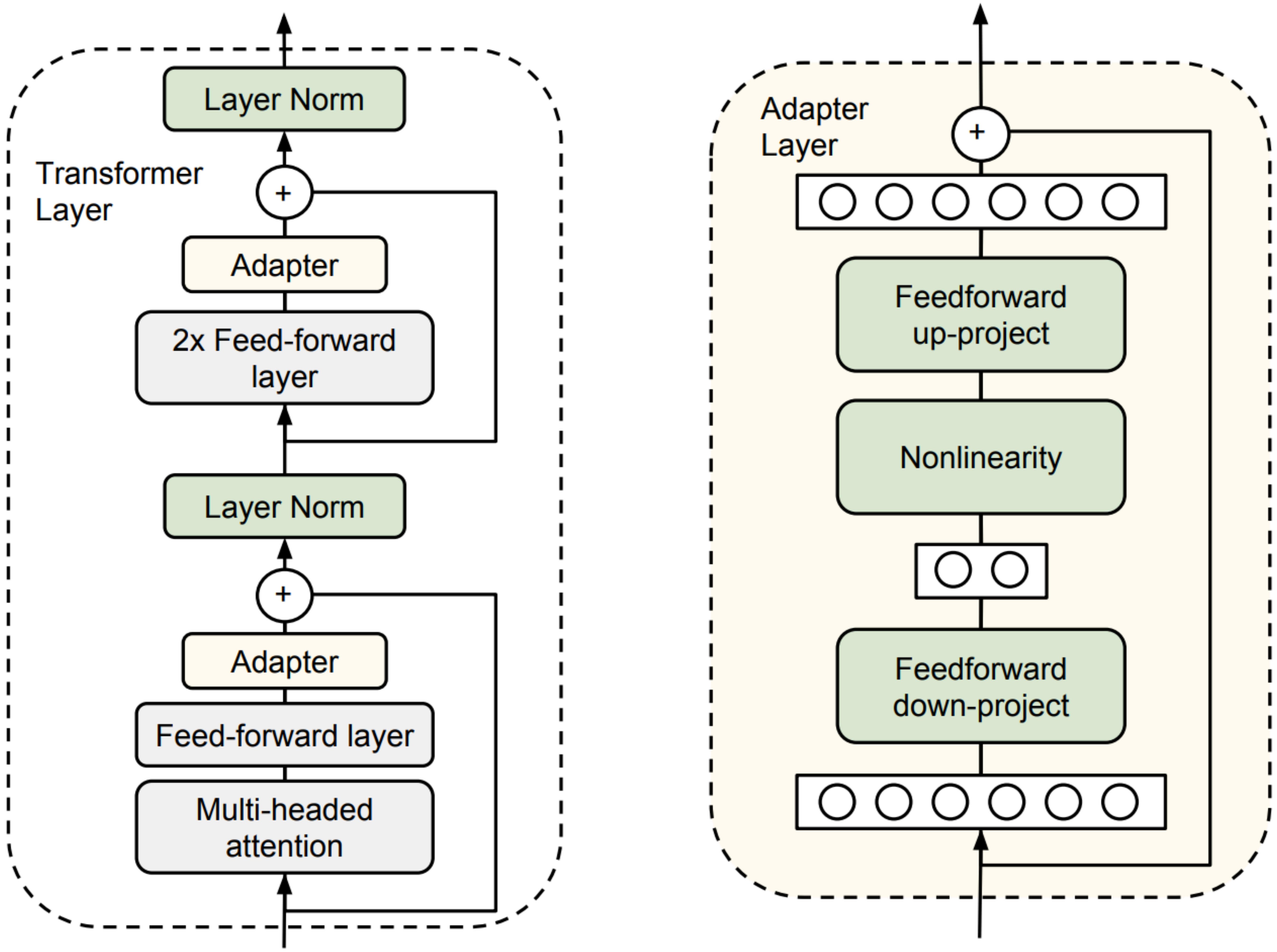
- Time Complexity: O(d_row * d_col^2 + d_row^3)

Fine-tuning

- Parameter Efficient Fine-tuning (PEFT)
    - Prompt Tuning (In-context Learning)
    - Prefix Tuning
        - P-Tuning V2 (ACL 2022)
    - Adapter
        - Adapter tuning for NLP (ICML 2019)
        - Offsite-tuning (ICML 2023 submission)
    - LoRA
        - Low Rank Adaptation for LLM

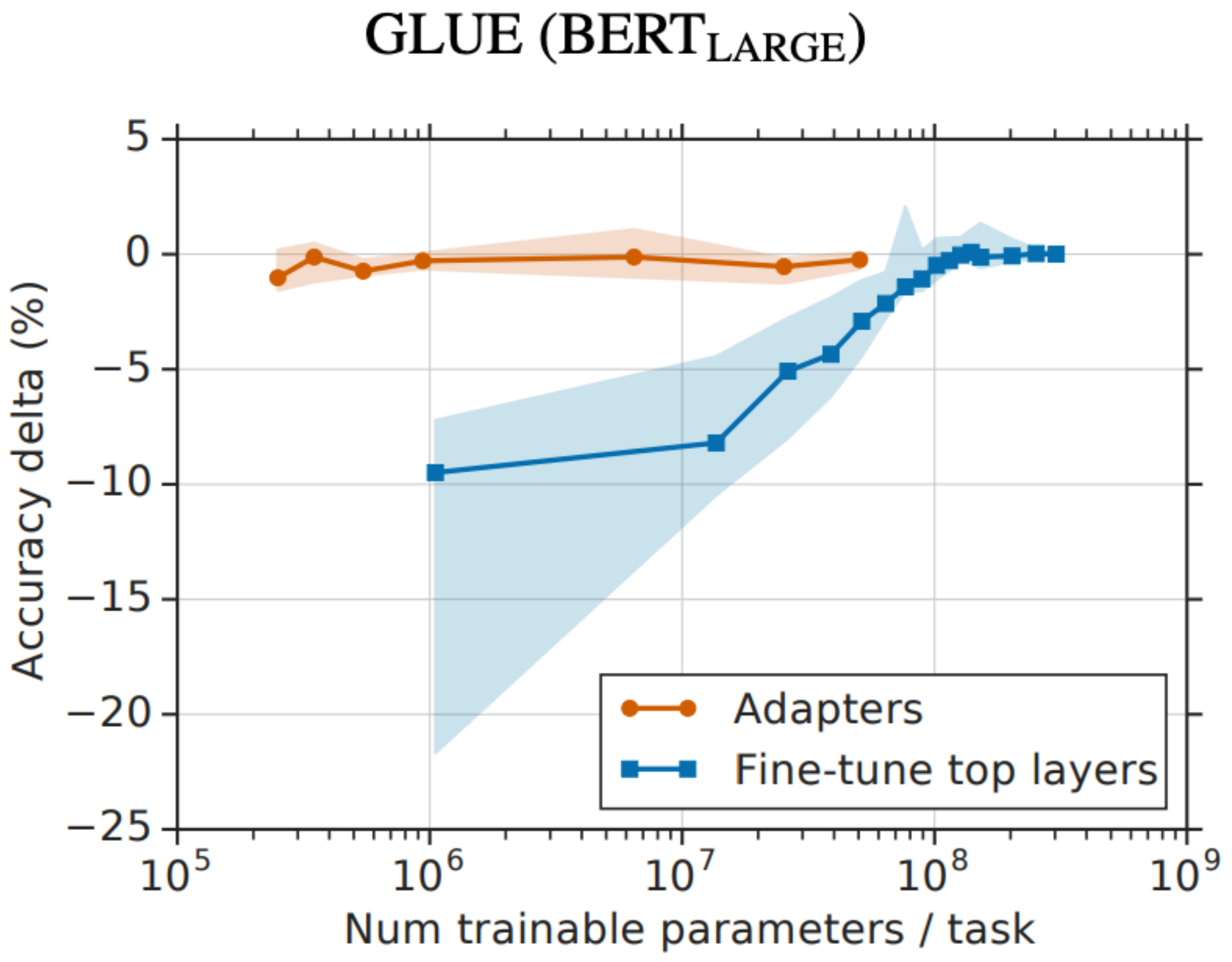
Adapter Tuning (ICML 2019)

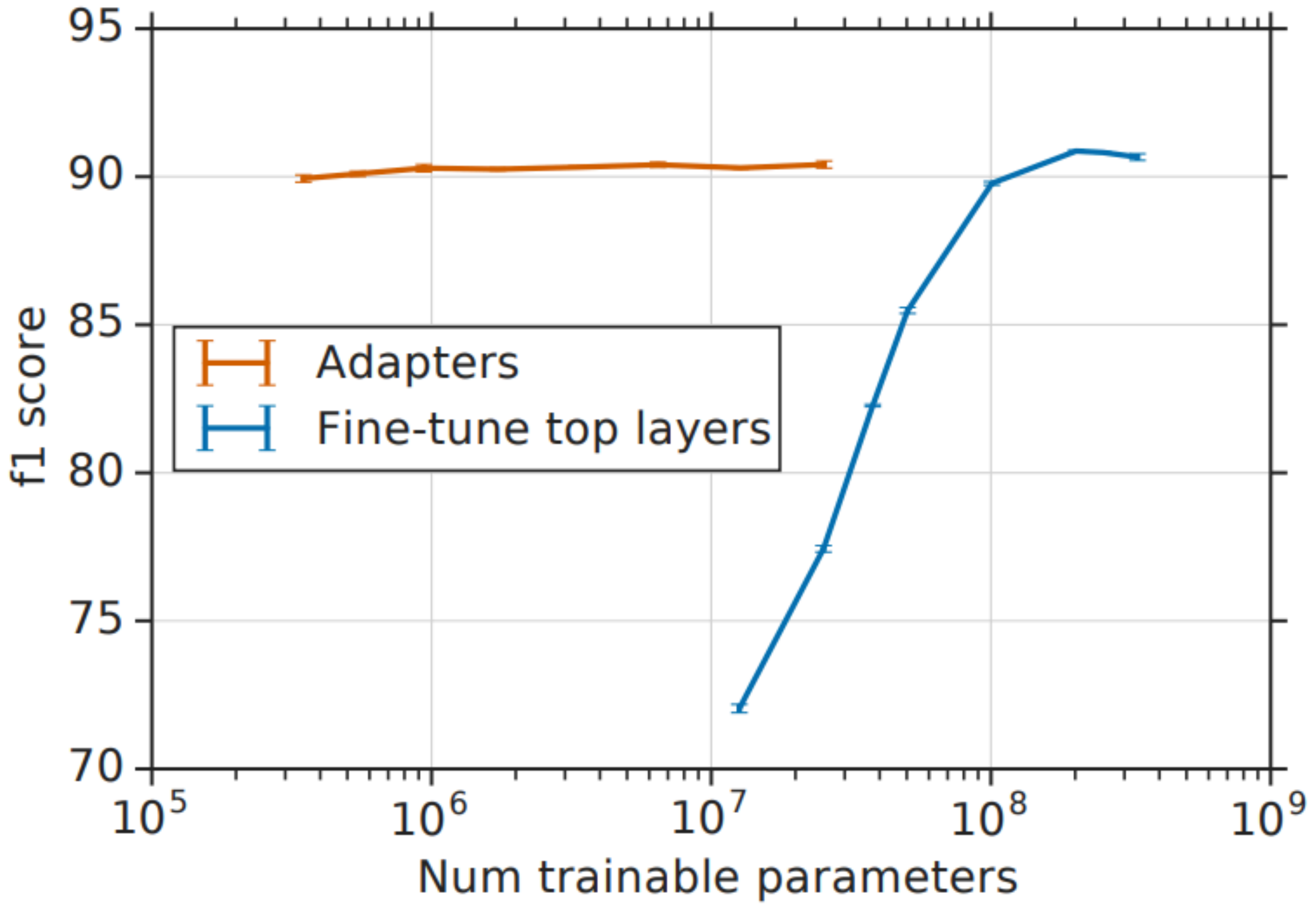
P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks



Adapter Tuning (ICML 2019)

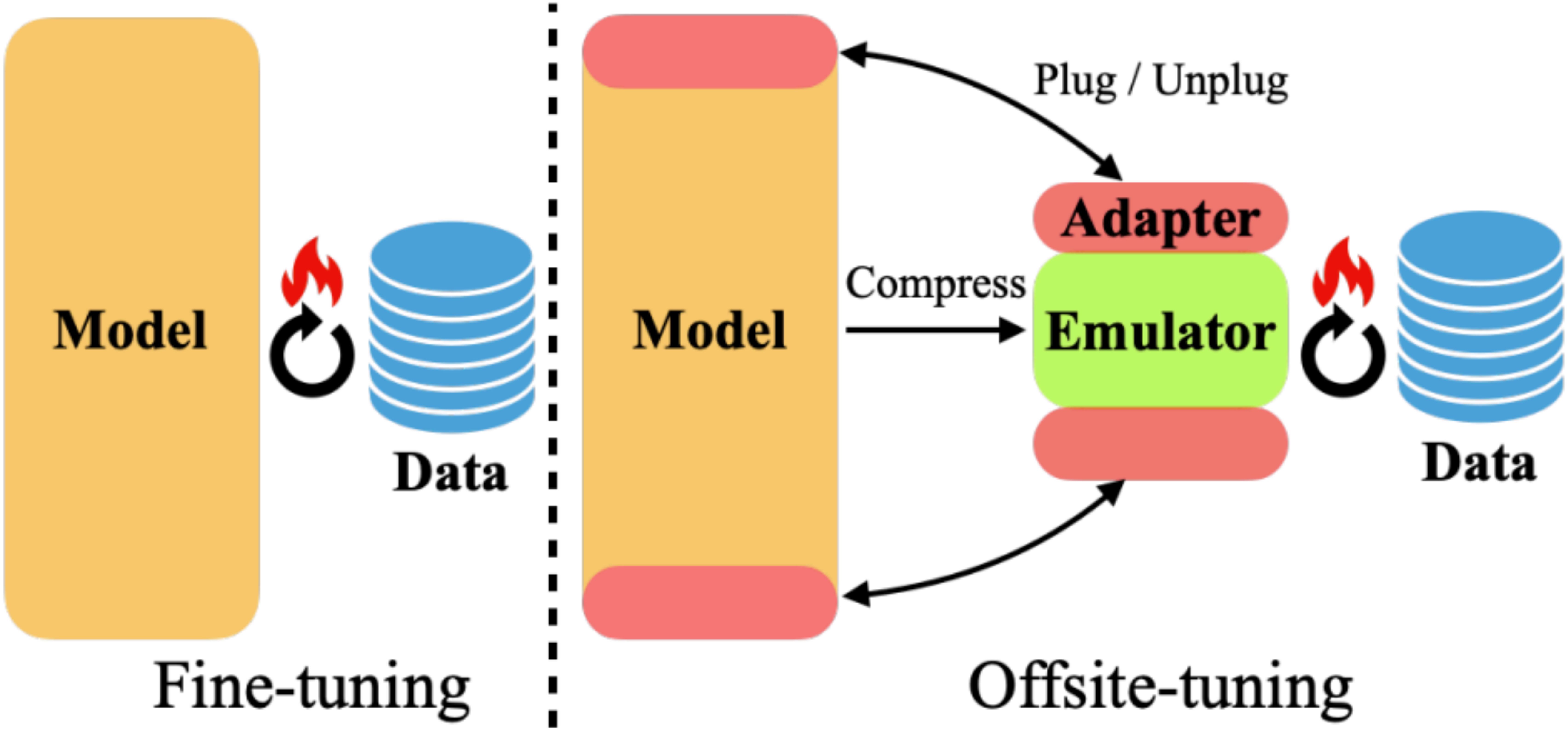
P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks





Offsite-Tuning (ICML 2023 submission)

Offsite-Tuning: Transfer Learning without Full Model



Offsite-Tuning (ICML 2023 submission)

Offsite-Tuning: Transfer Learning without Full Model

Setting	OpenBookQA	PIQA	ARC-E	ARC-C	HellaSwag	SciQ	WebQs	RACE	WikiText (↓)
					GPT2-XL (2-16-2 Distill)				
Full ZS	23.0%	70.9%	58.2%	25.1%	40.0%	83.2%	1.5%	33.0%	20.44
Emulator ZS	18.8%	67.7%	53.2%	20.8%	33.5%	77.0%	0.2%	30.0%	25.12
FT	30.0%	73.2%	62.9%	30.0%	40.7%	92.5%	26.4%	43.2%	13.58
OT Emulator	24.0%	70.3%	58.2%	23.9%	35.8%	92.7%	18.9%	39.4%	17.64
OT Plug-in	28.2%	73.6%	61.4%	28.5%	41.6%	93.2%	19.9%	39.9%	14.94
					OPT-1.3B (2-8-2 Distill)				
Full ZS	23.4%	71.6%	56.9%	23.5%	41.5%	84.4%	4.6%	34.2%	31.48
Emulator ZS	19.4%	68.7%	53.9%	21.5%	35.1%	80.9%	1.3%	33.0%	38.55
FT	31.4%	75.2%	61.3%	27.7%	42.7%	92.5%	31.2%	37.0%	12.52
OT Emulator	24.8%	71.6%	58.1%	26.1%	37.0%	92.2%	24.3%	38.6%	15.54
OT Plug-in	29.0%	74.5%	59.4%	27.8%	43.3%	92.9%	26.2%	38.9%	13.15

LoRA

LoRA: Low-Rank Adaptation of Large Language Models

$$W_0 + \Delta W = W_0 + BA, \text{ where } B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}, \text{ and the rank } r \ll \min(d,k)$$

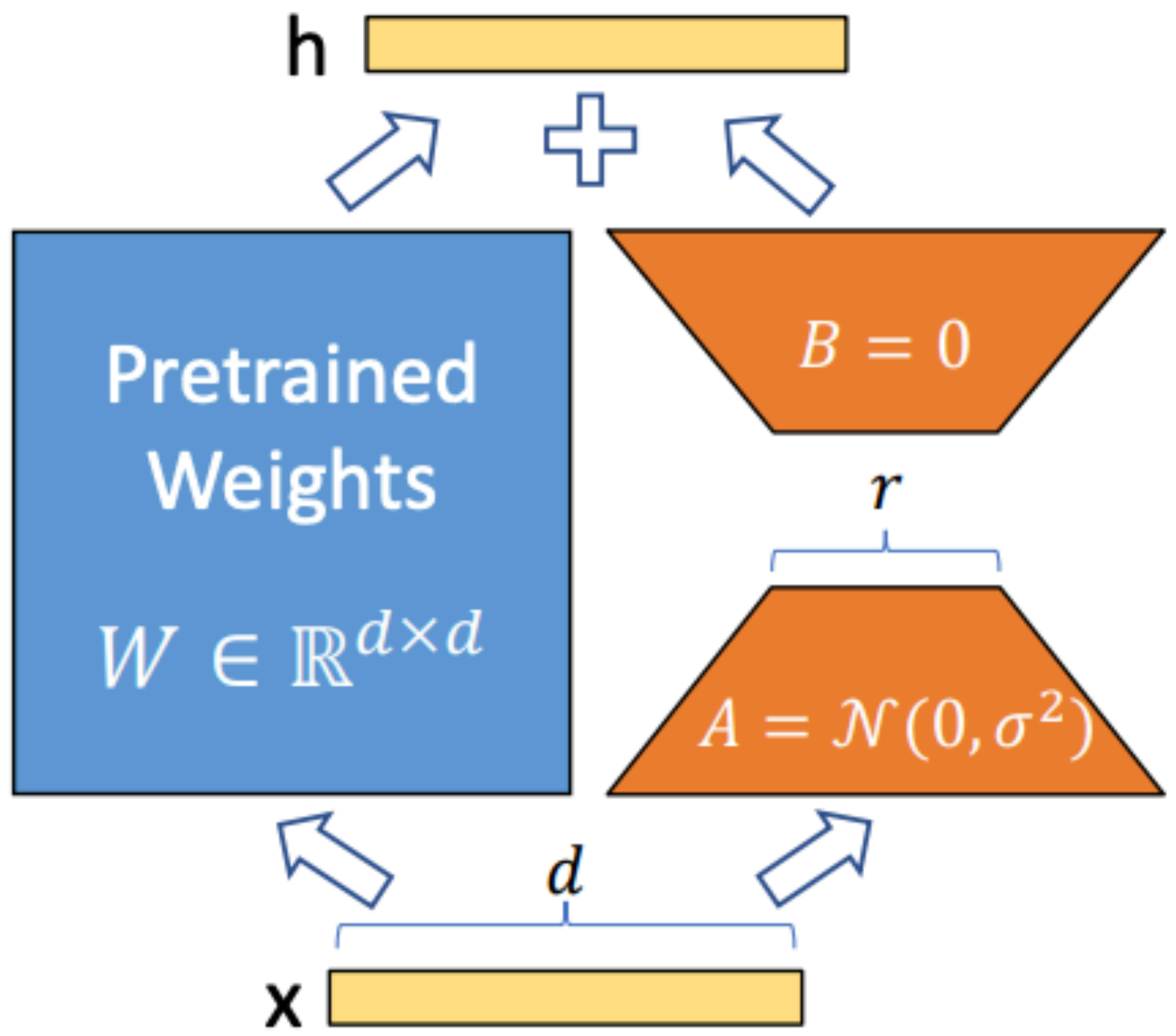


Figure 1: Our reparametrization. We only train A and B.

LoRA

LoRA: Low-Rank Adaptation of Large Language Models

Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1