Efficient Fine-Tuning

EECE695D: Efficient ML Systems

Spring 2025

Recap

- **Last three classes.**
  - Efficient Training
    - Transferring knowledge from another model
  - Editing
    - Pinpoint knowledge injection to a model with minimal ops
- **Today.** Efficient Fine-Tuning
  - Update less parameters than full model

Basic idea

Motivation

- Often, we are not happy with large pre-trained models (e.g., LLMs)
  - Specialize for certain downstream tasks
  - Correct errors / outdated info / harmful behavior
  - Personalize for individuals

Photo

SAM output

Ground Truth

Photo Credit: Cpt. Yuji Byun

Motivation

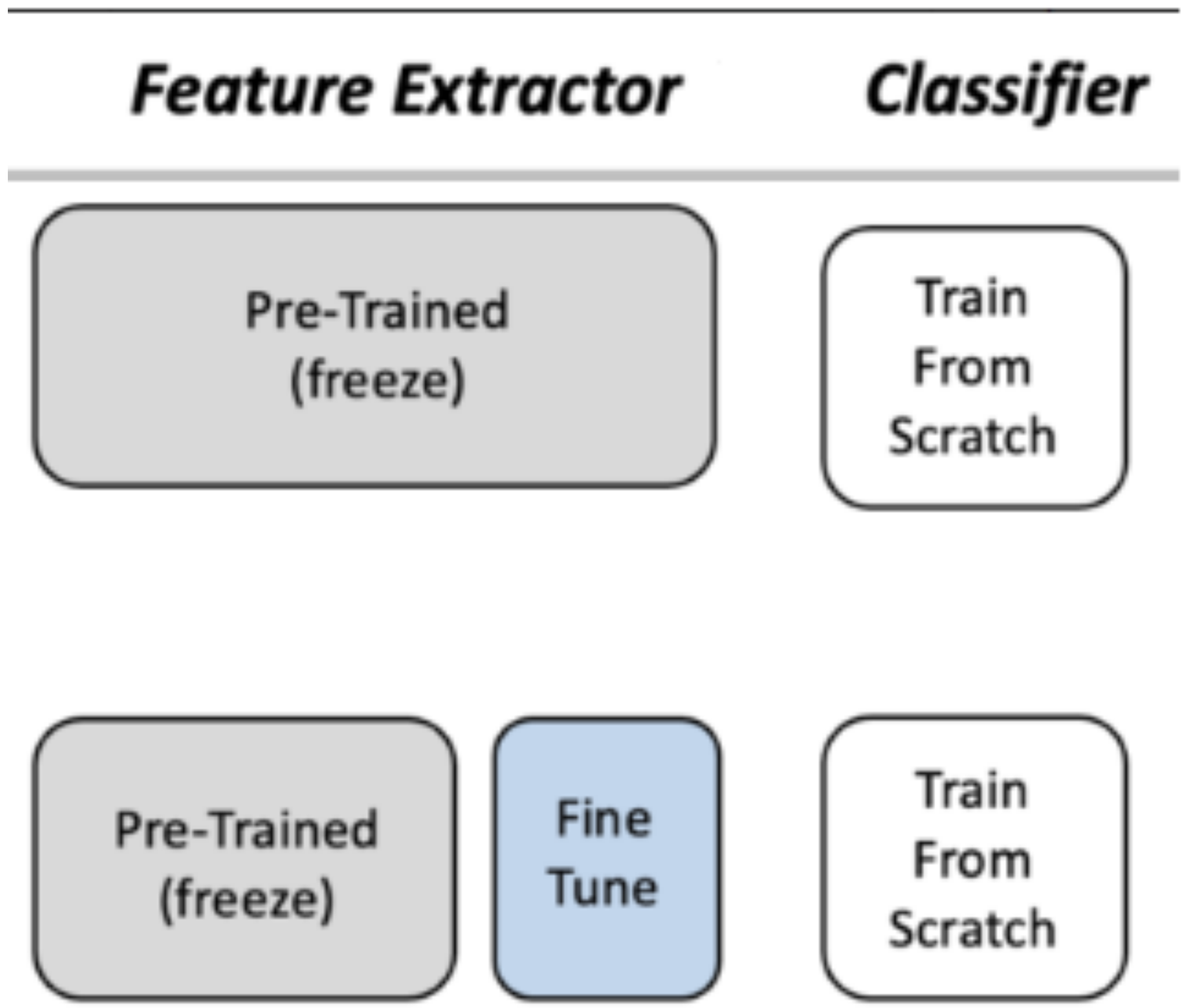
- Thus, we often want to train further using additional data—i.e., fine-tune

- **Problem.**
  - Memory. Too many trainable parameters
  - Quality. Forgets what model knows
  - Storage. Need to store the delta

+ Need to do it many times (personalization, recent info, …)

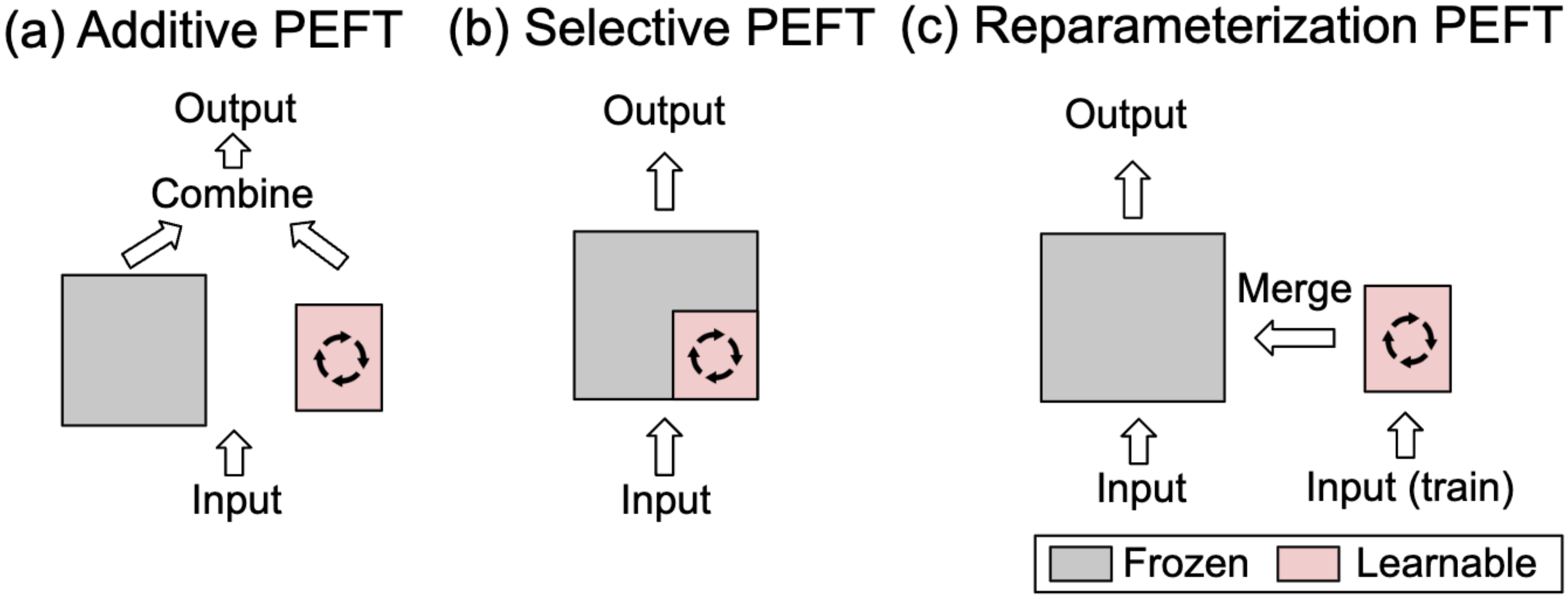
Idea

- Reduce the number of trainable parameters
  - Less memory
  - Less forgetting
  - Less storage
- **Classic example.**
  - Fine-tune later layers
  - Early layers frozen
    - Intuition. Early layers extract elementary features



Approaches

- Roughly three categories:



Han et al., "Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey" TMLR 2024

Additive PEFT

Additive PEFT

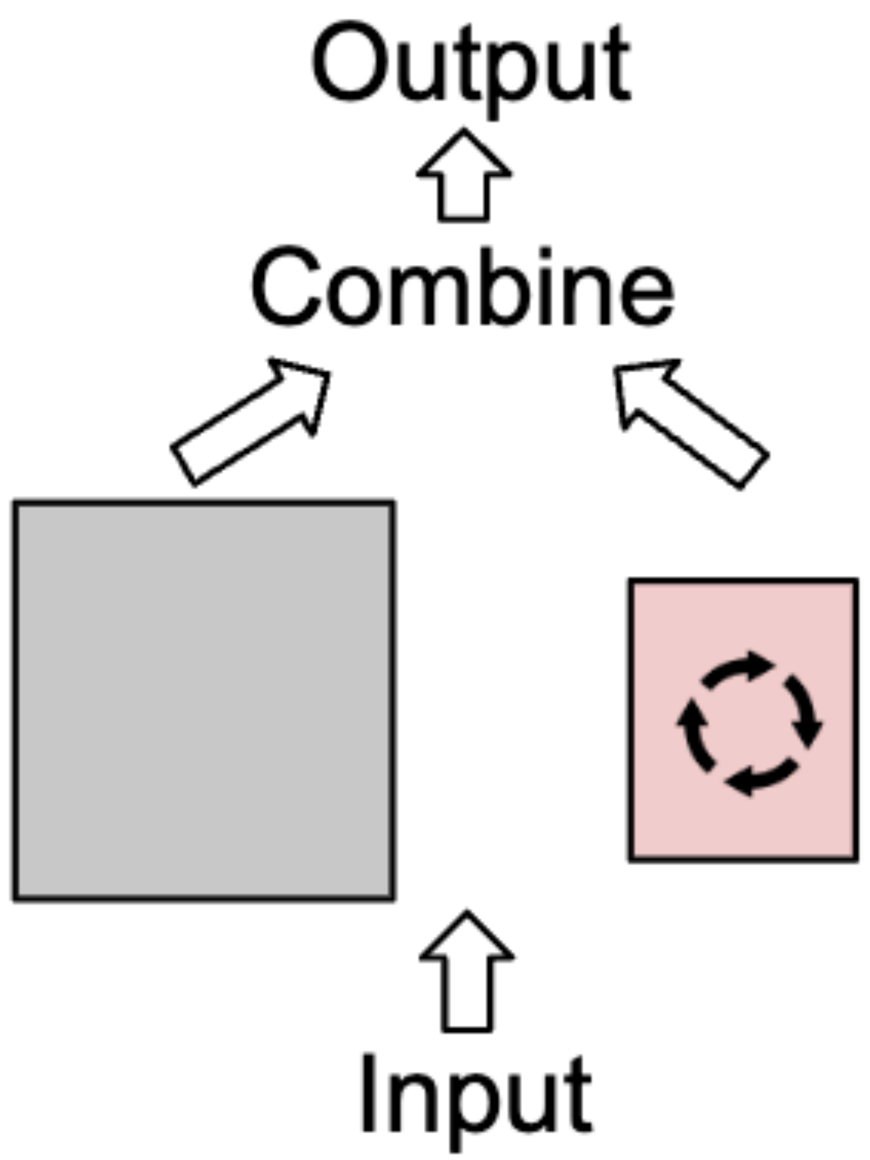
- **Idea.** Add some extra parameters, and use them during inference

  - Model dim. Adding model parameters

    $$f(x;\theta) \quad \Rightarrow \quad f(x;(\theta,\phi))$$

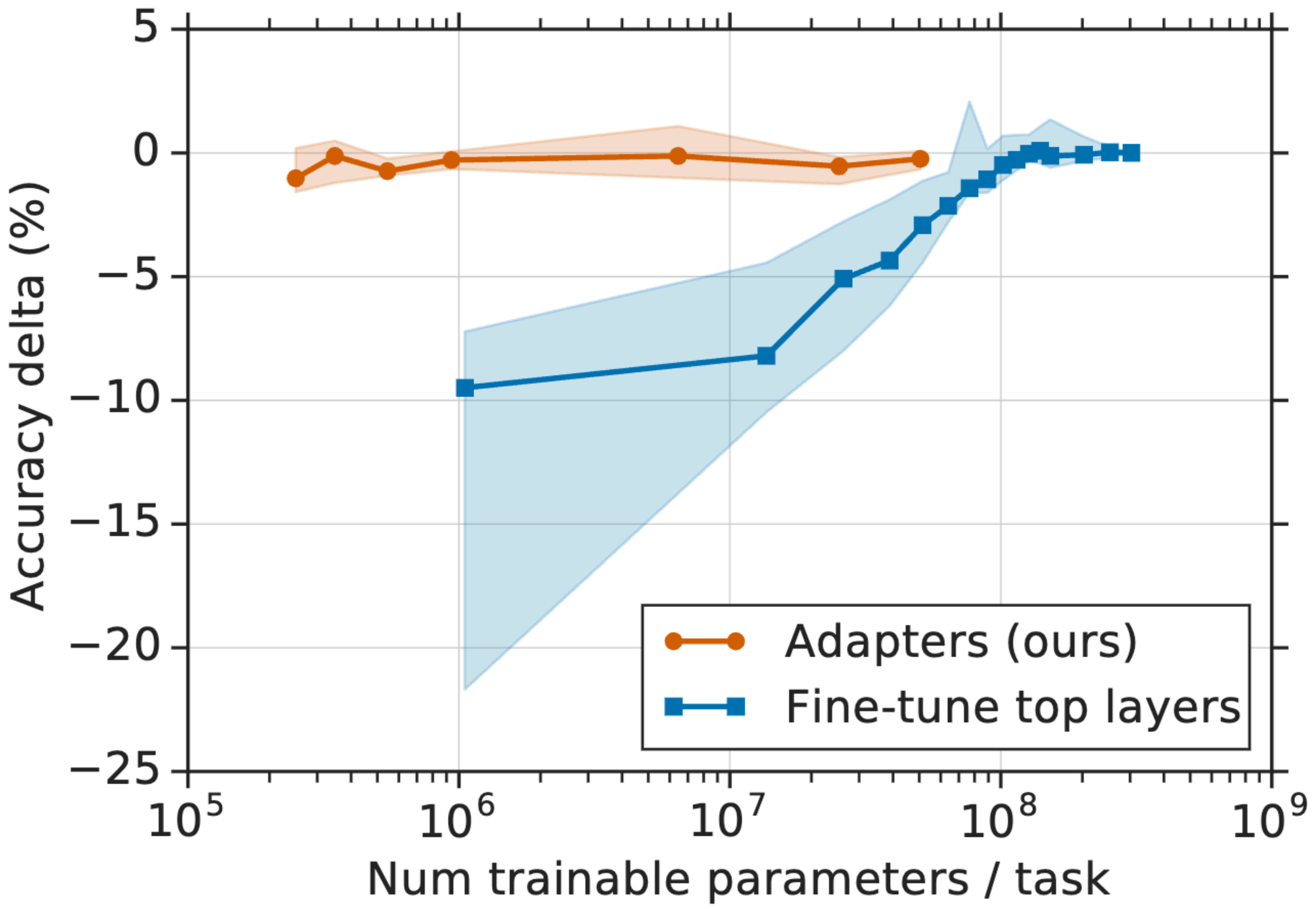
  - Data dim. Adding prompt

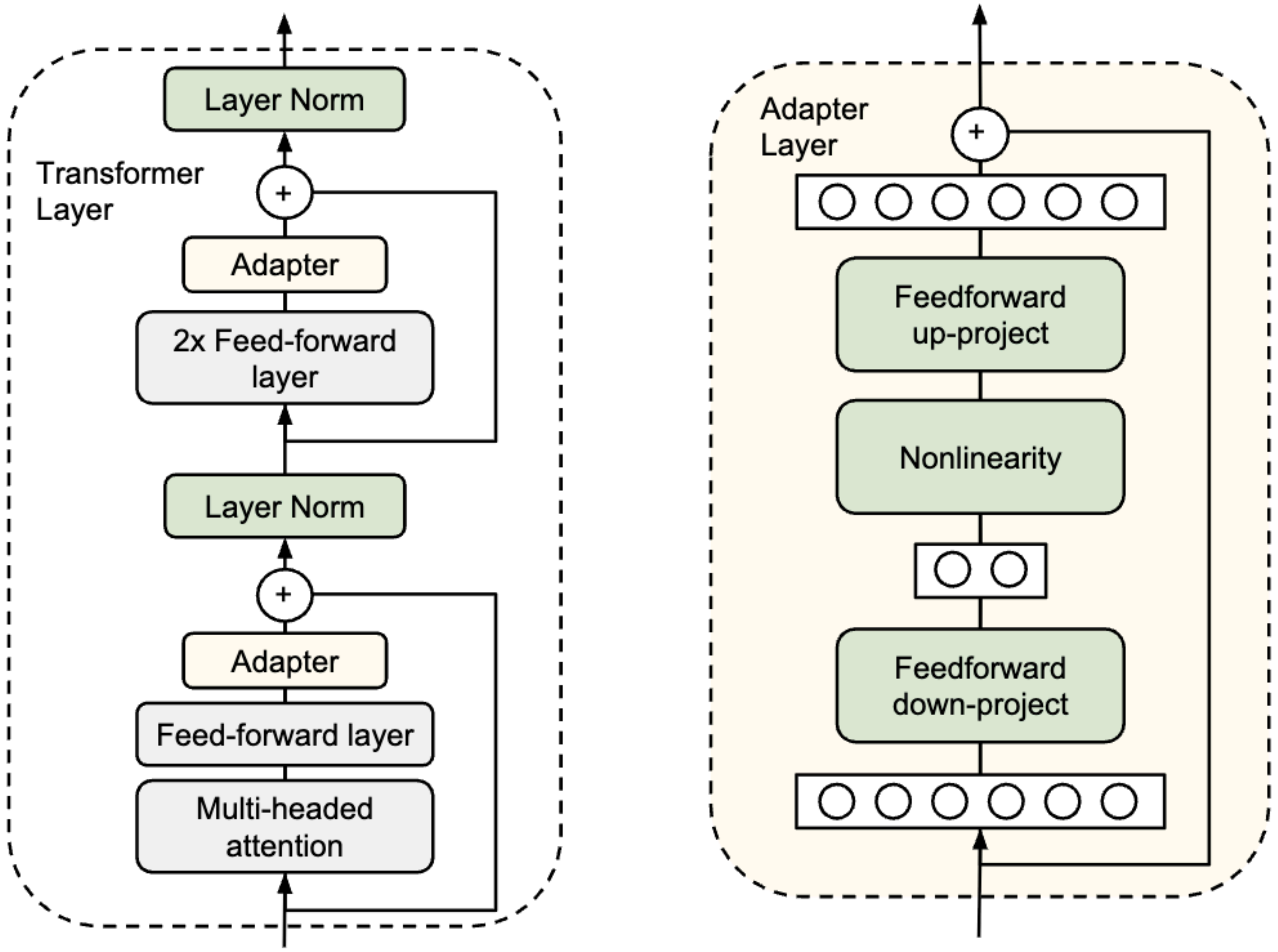
    $$f(x;\theta) \quad \Rightarrow \quad f(p \oplus x;\theta)$$



Adding parameters

- **Example.** Adapter (Houlsby et al., 2019)
- Adds small hourglass-like MLP after each layer
  - Very small init. w/ skip connection
    - Begin from "no adapter"





Houlsby et al., "Parameter-Efficient Transfer Learning for NLP" ICML 2019

Adding parameters

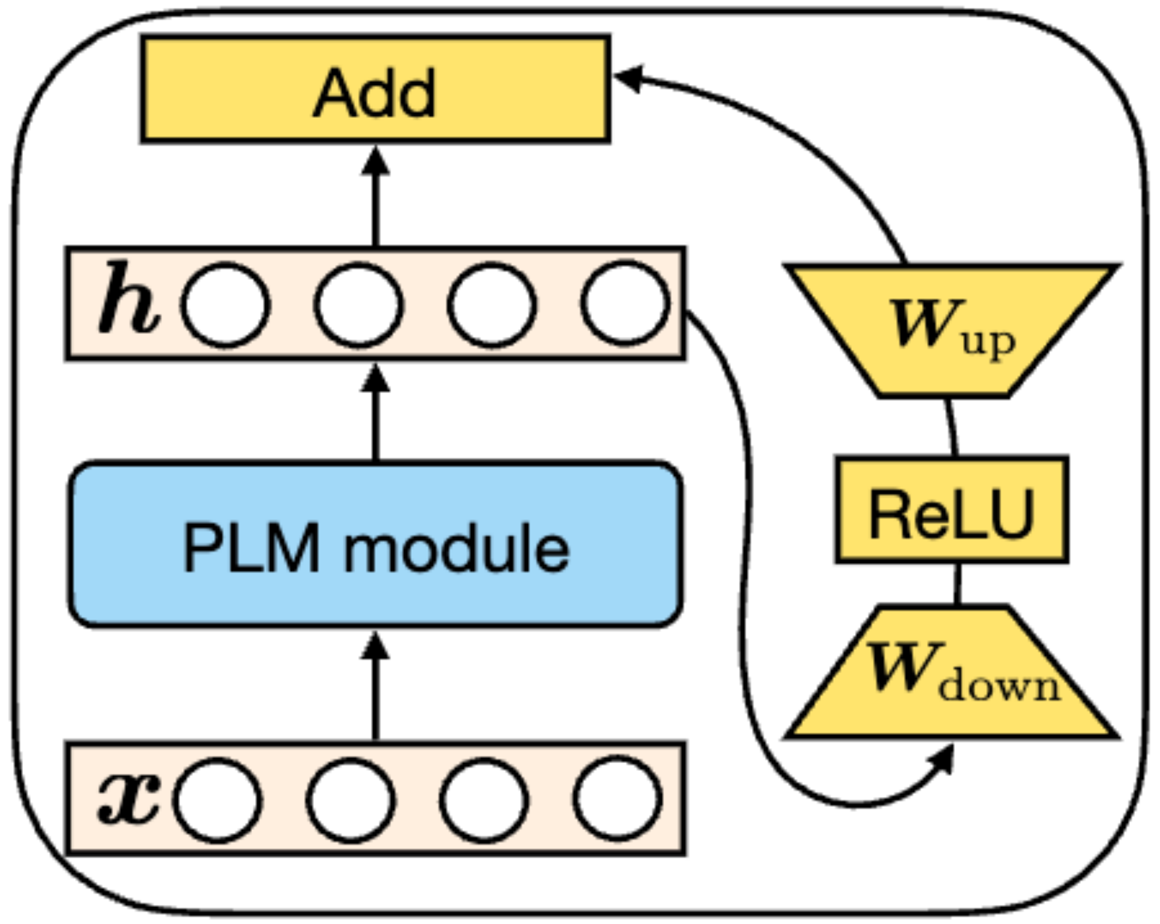
- **Drawback.** Slower inference
    - Added computation
    - Serial structure

Batch Size	32	16	1
Sequence Length	512	256	128
$\|\Theta\|$	0.5M	11M	11M
Adapter^{L}	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
Adapter^{H}	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

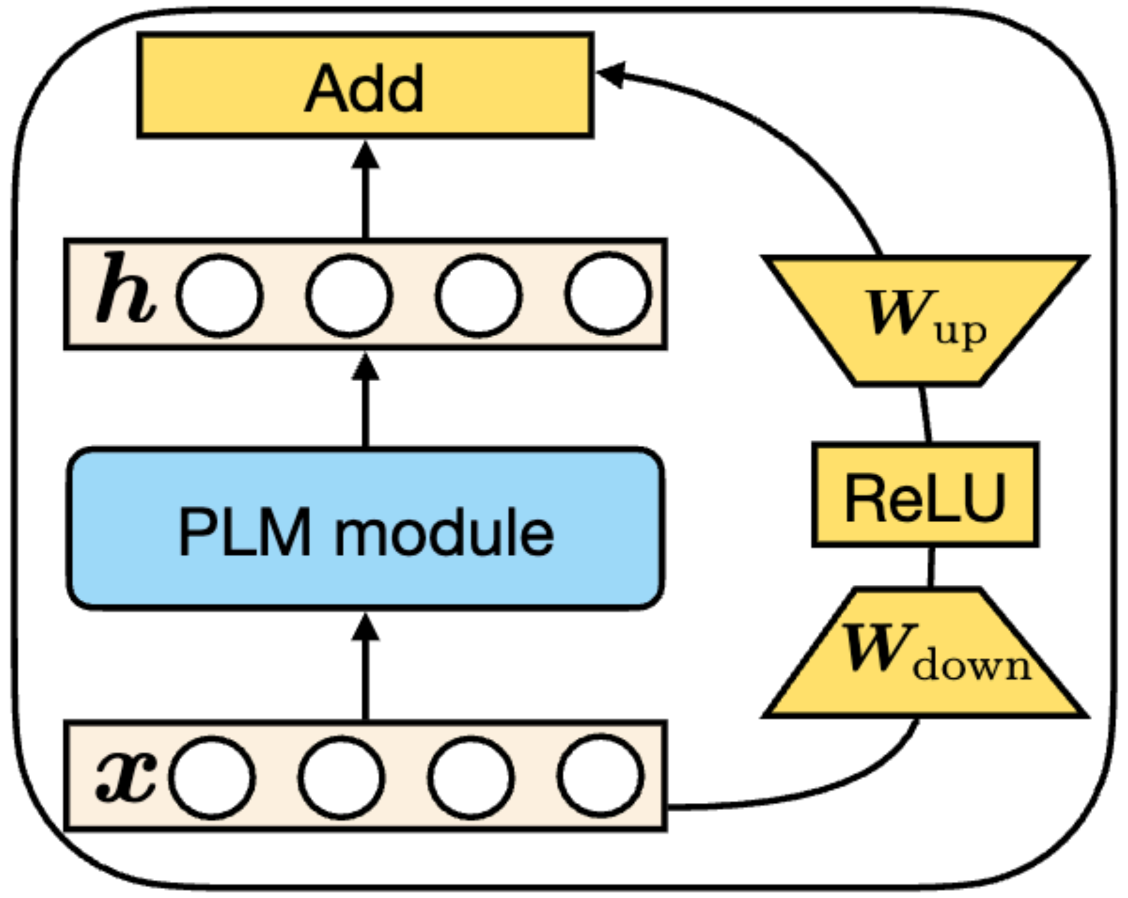
Inference latency of a single forward pass in GPT-2 medium averaged over 100 trials. Results are based on NVIDIA Quadro RTX8000

Adding parameters

- **Remedy.**
    - Parallelization
    - LoRA (later today)



Serial



Parallel

He et al., "Towards a unified view of parameter-efficient transfer learning" ICLR 2022

Adding prompts

- **Motivation.** Prepending additional examples make LLMs work better
    - Do we really need them to be “examples”?
    - Can we optimize them explicitly?

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:     ← task description
cheese => ..................     ← prompt
```

One-shot

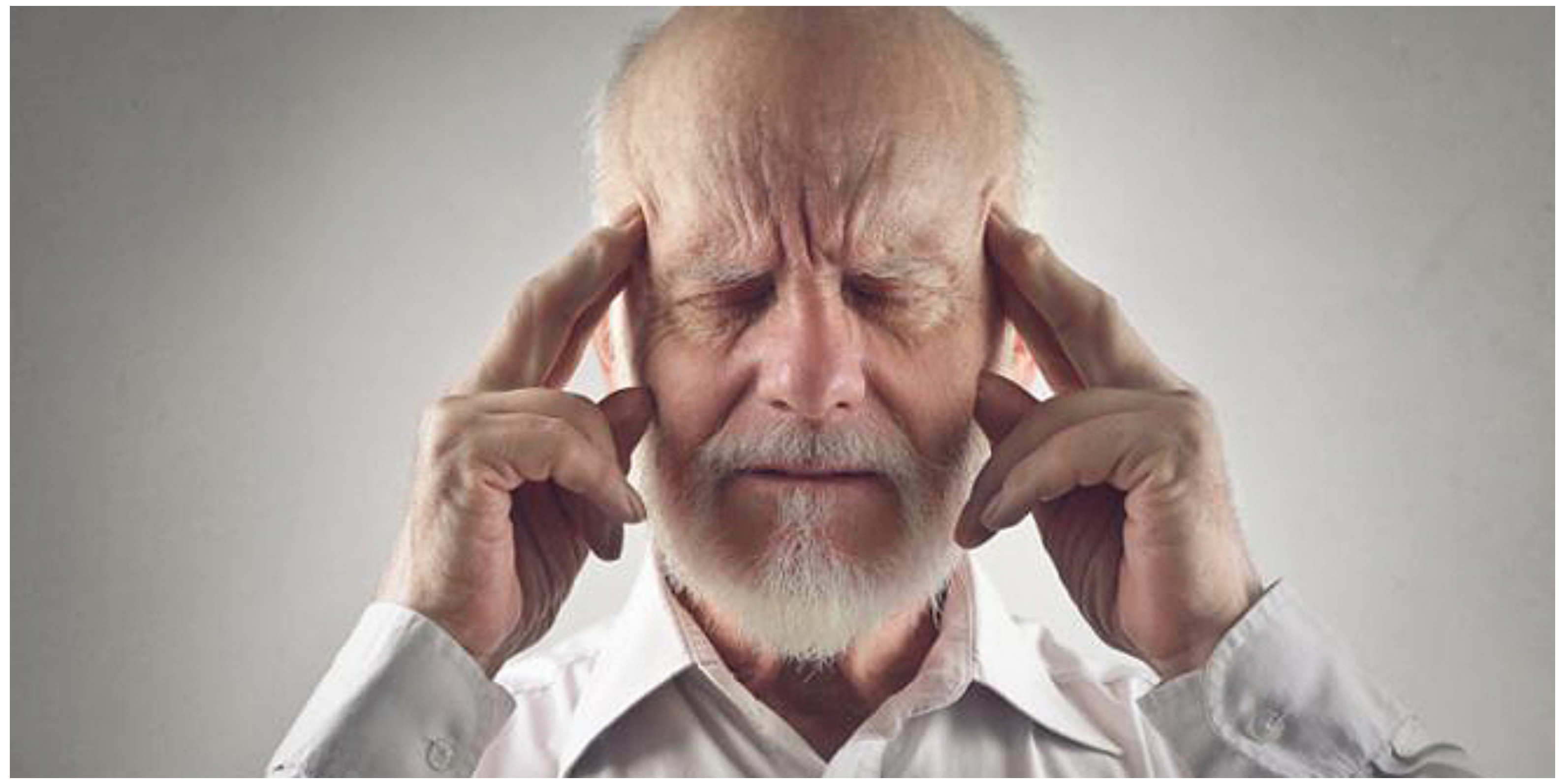
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:     ← task description
sea otter => loutre de mer       ← example
cheese => ..................     ← prompt
```

Brown et al., “Language Models are Few-Shot Learners,” arXiv 2020

Adding prompts

- **Option 1.** Human thinks hard, and write them manually

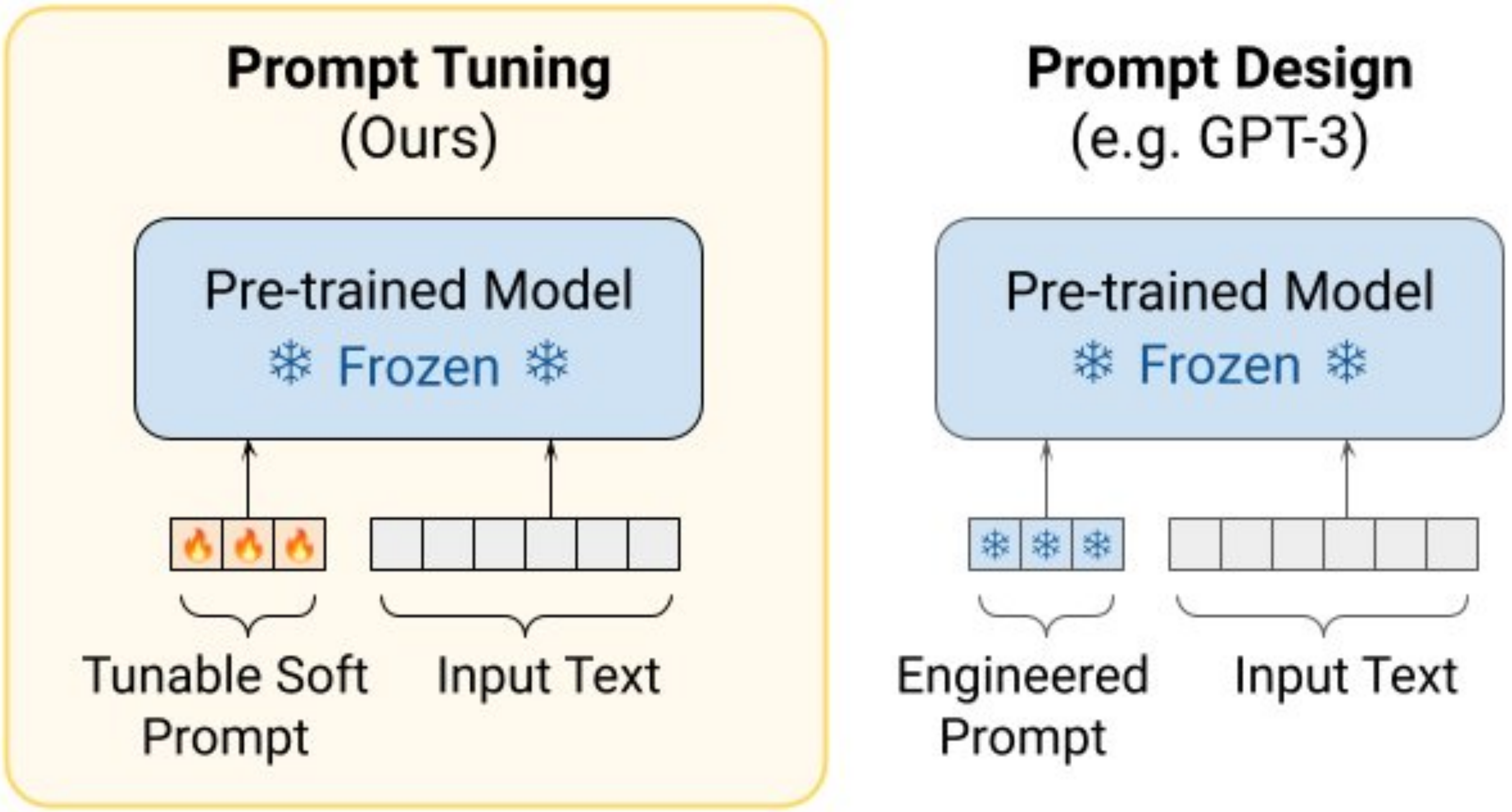


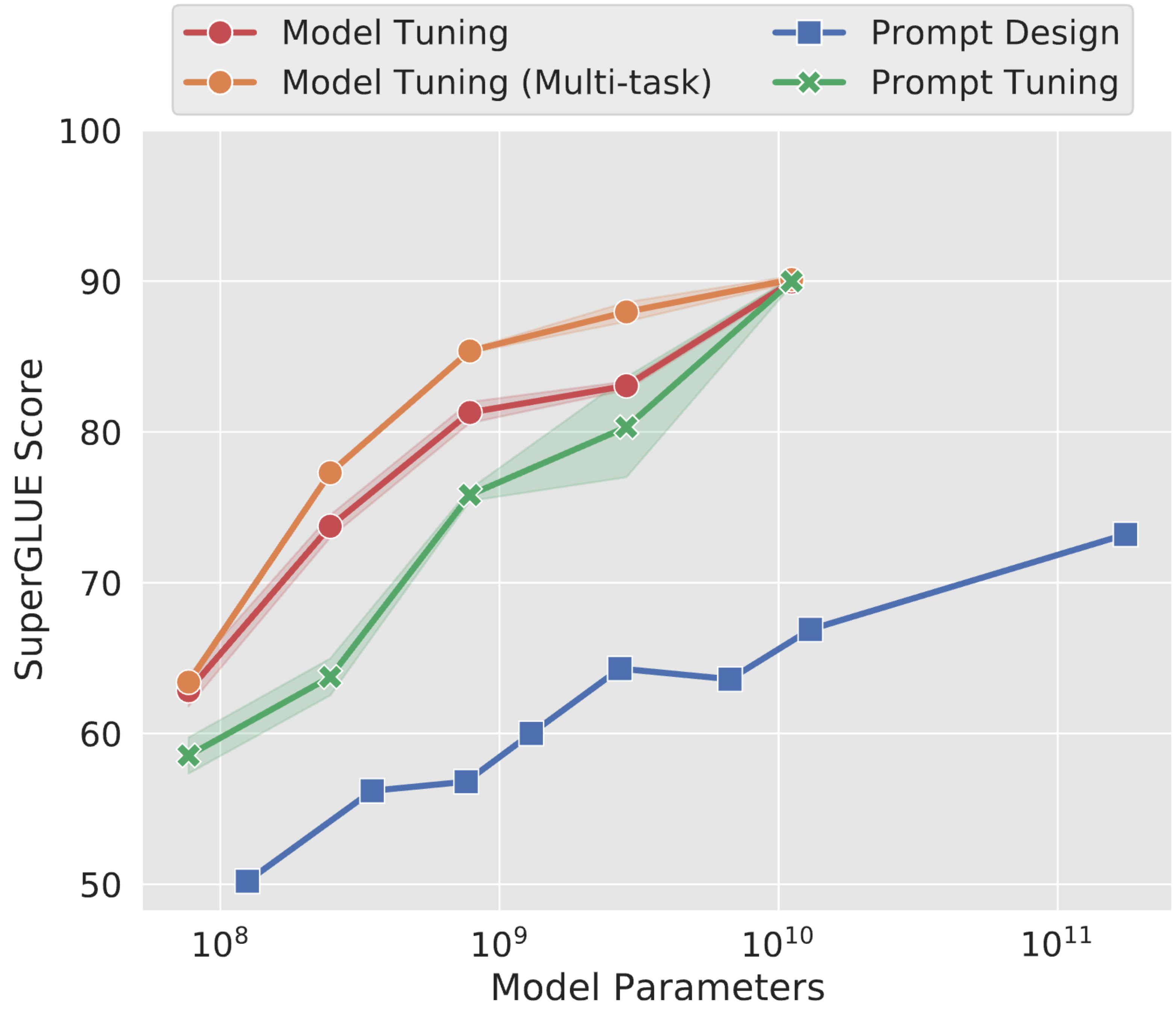
- **Option 2.** Automated search, in the discrete word space
    - Reinforcement learning
    - Gradient-based search (e.g., Autoprompt)

Shin et al., “AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts,” EMNLP 2020

Adding prompts

- **Option 3.** Continuous optimization (Prompt tuning)
  - Train embedding vectors with SGD
    - Lose interpretability, but good performance

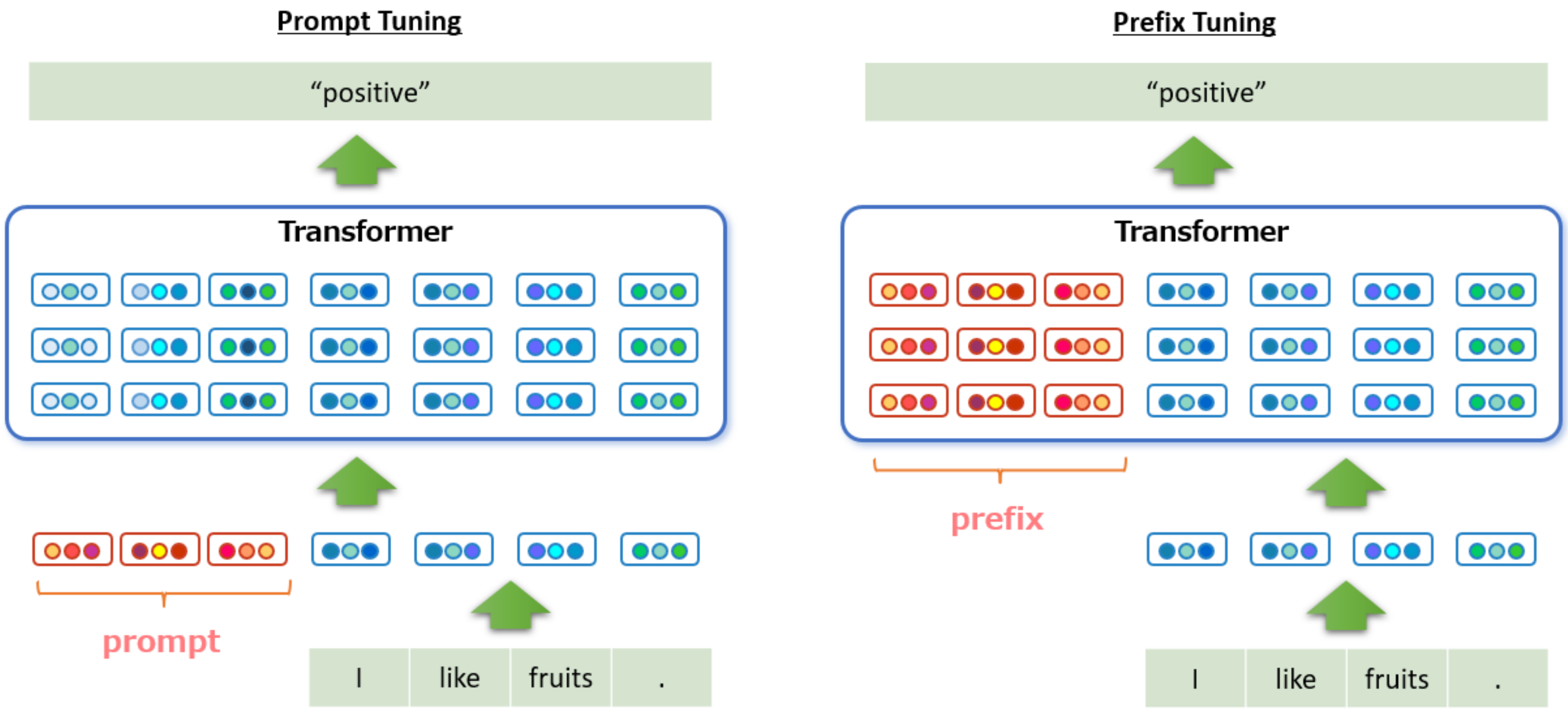




Lester et al., "The Power of Scale for Parameter-Efficient Prompt Tuning," EMNLP 2021

Adding prompts

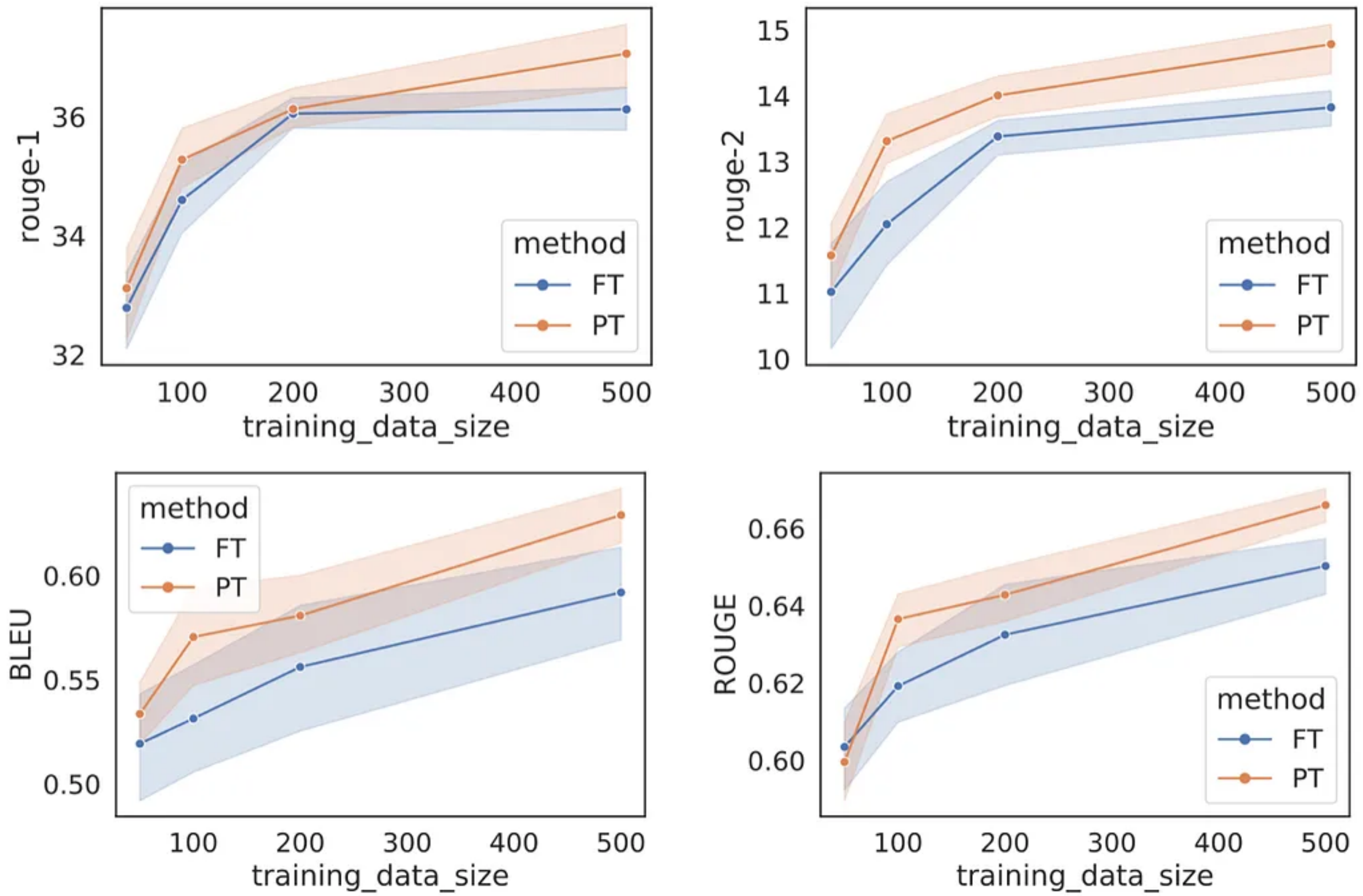
- **Prefix Tuning.** Continuous optimization for the intermediate features
  - i.e., modifies key-value cache, not the input
  - more storage, less computation, same memory



Li and Liang, "Prefix-Tuning: Optimizing Continuous Prompts for Generation," ACL 2021
https://www.ogis-ri.co.jp/otc/hiroba/technical/similar-document-search/part28.html

Adding prompts

- Prefix tuning matches / outperforms full fine-tuning
    - Especially good in the low-data scenarios

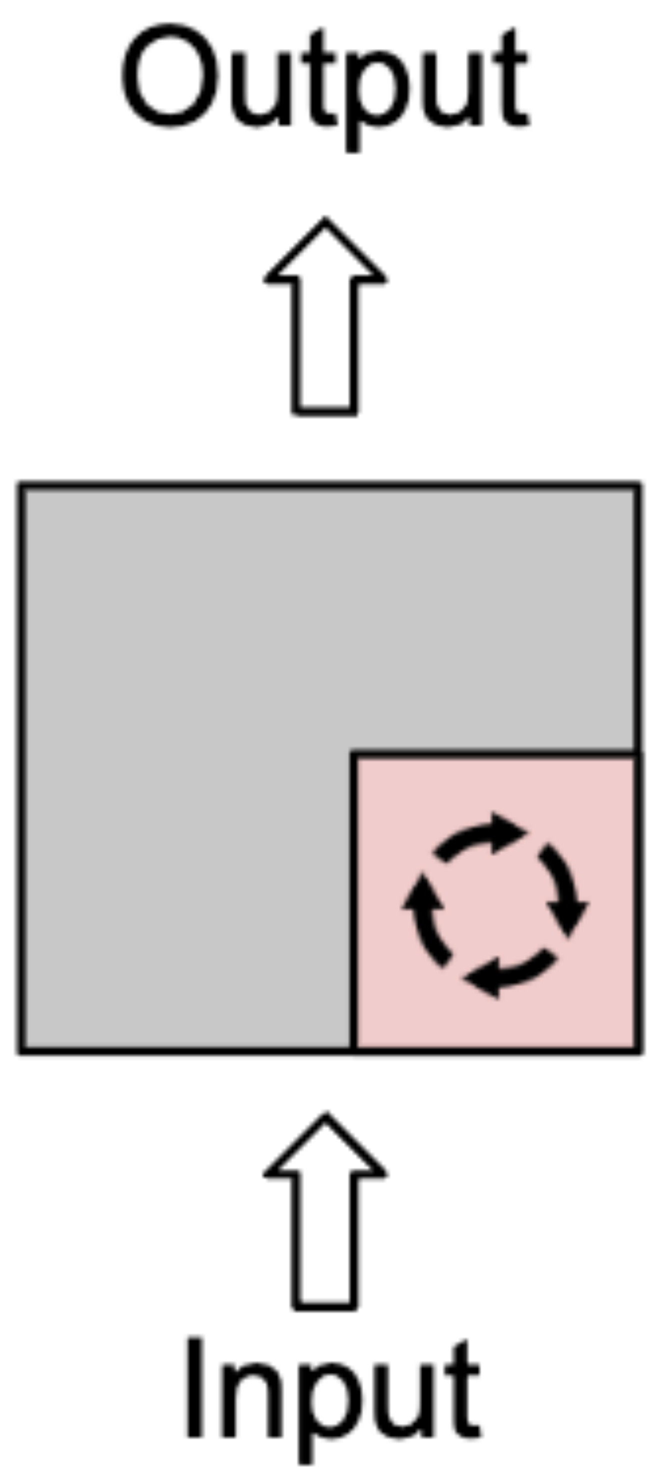


https://magazine.sebastianraschka.com/

Selective PEFT

Selective PEFT

- **Idea.** Fine-tune only a fraction of the parameters
  - Naturally, involves the notion of sparsity:
    - Unstructured
    - Structured

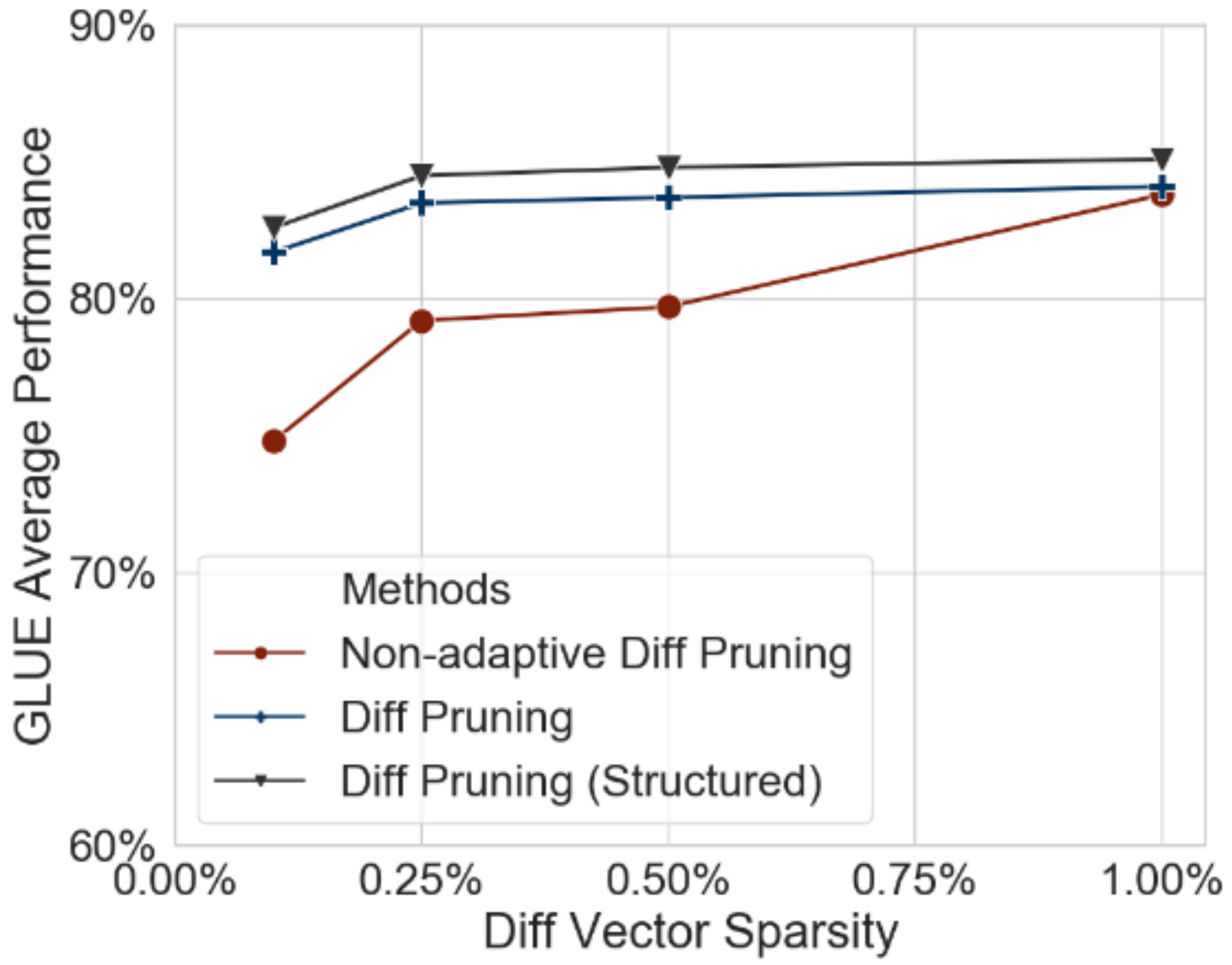


Unstructured sparse PEFT

- **Example.** Diff pruning (2020)
  - Train a sparse update vector with ℓ_0-norm penalty

$$\min_{\delta} \left(\mathbb{E}[\ell(f(x;\theta+\delta),y)] \quad + \quad \lambda \cdot \|\delta\|_0 \right)$$

  - Uses a stochastic gating function, similar to usual sparse training

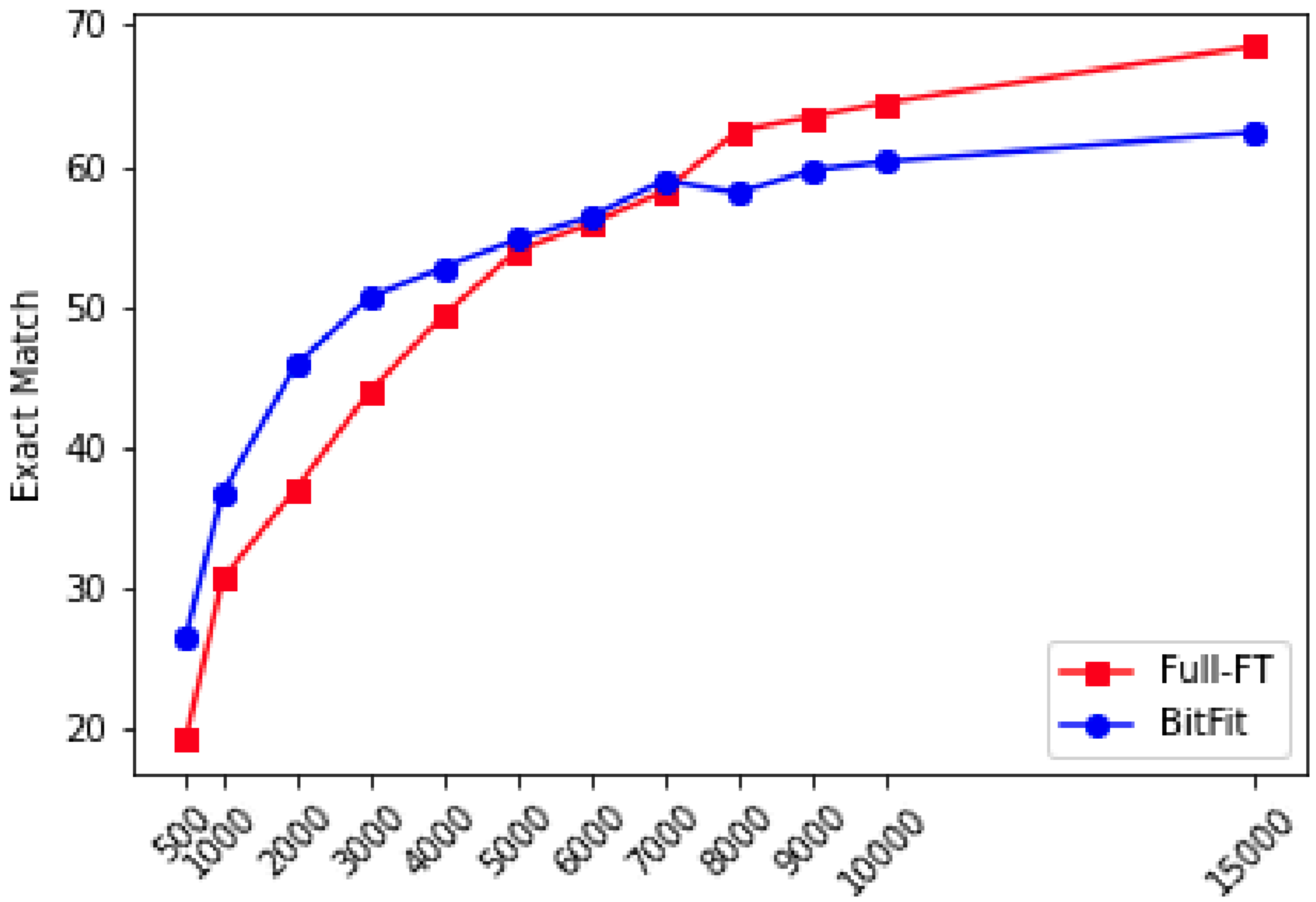


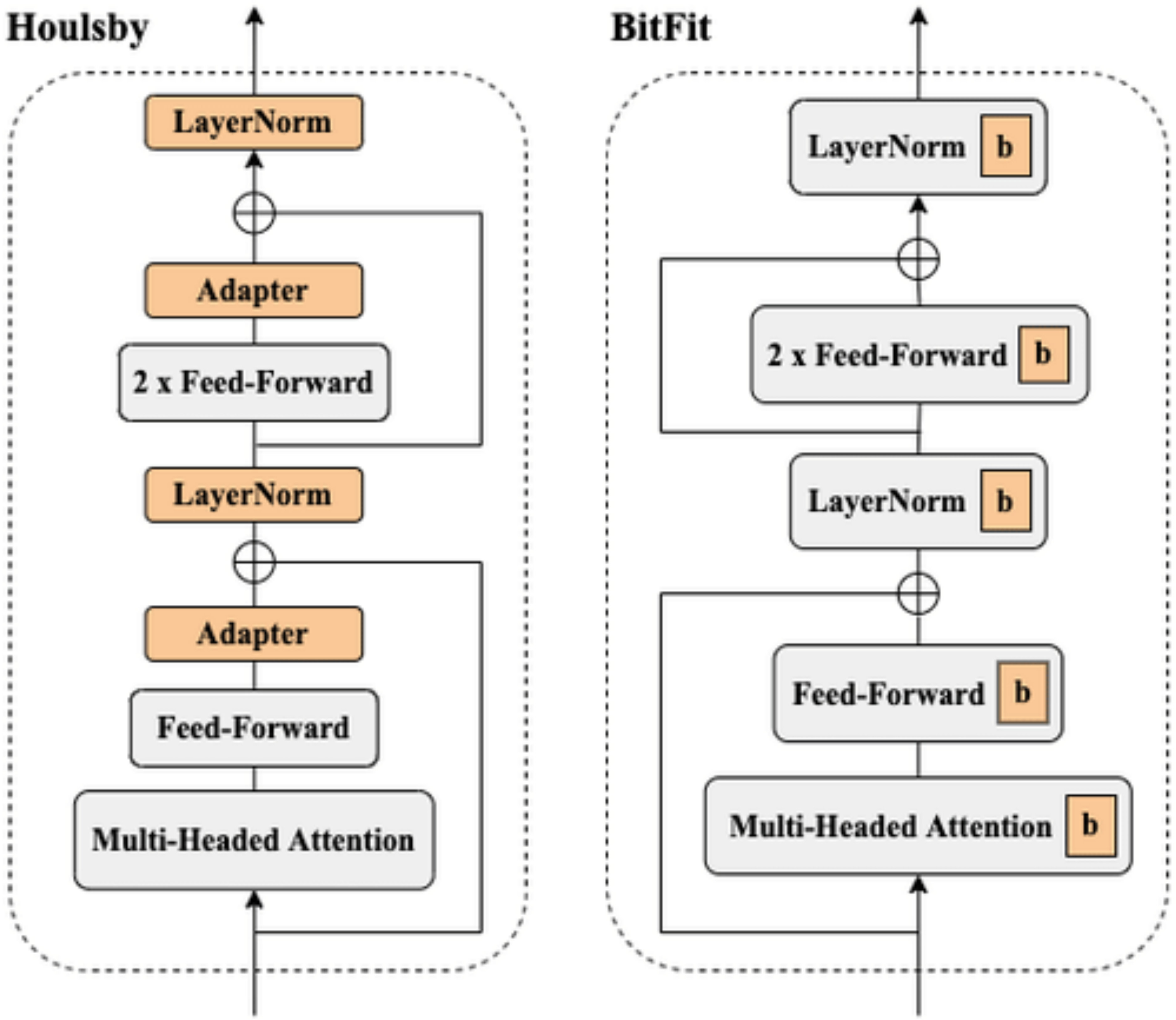
	SQuAD	
	New Params	F_1
Houlsby et al. (2019)		
Full finetuning	100%	90.7
Adapters	2.0%	90.4
This work		
Full finetuning	100%	90.8
Diff pruning	1.0%	92.1
Diff pruning (struct.)	0.5%	91.1
Diff pruning (struct.)	1.0%	93.2

Guo et al., "Parameter-efficient transfer learning with diff pruning" ACL 2021

Structured sparse PEFT

- **Example.** BitFit (2022)
  - Trains only the bias terms, not weights



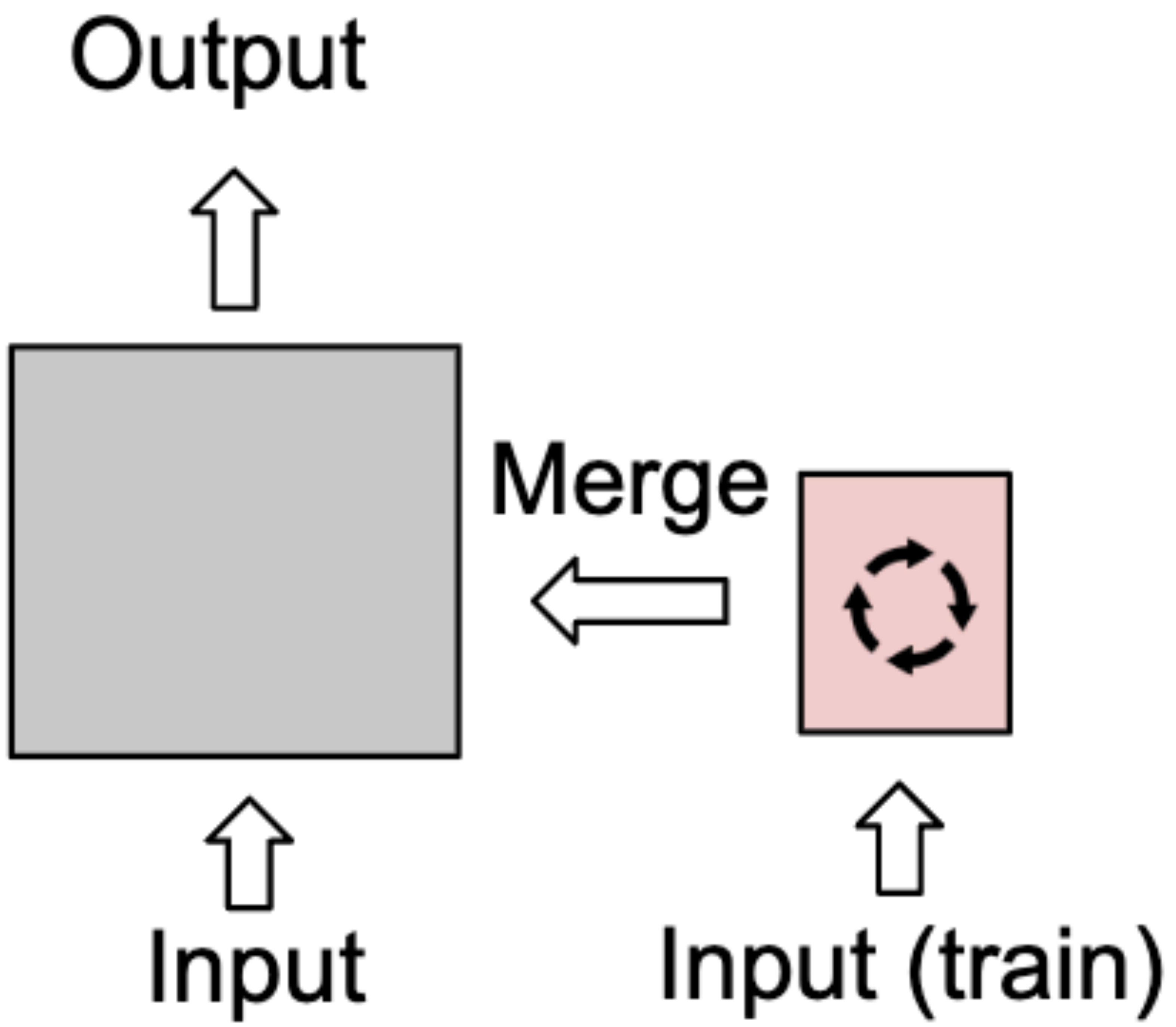


Ben-Zaken et al., "BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models" ACL 2022

Reparameterization PEFT

Reparameterization PEFT

- **Idea.** Similar to additive, but the additional parameters can be merged
  - Zero increase in the inference cost!

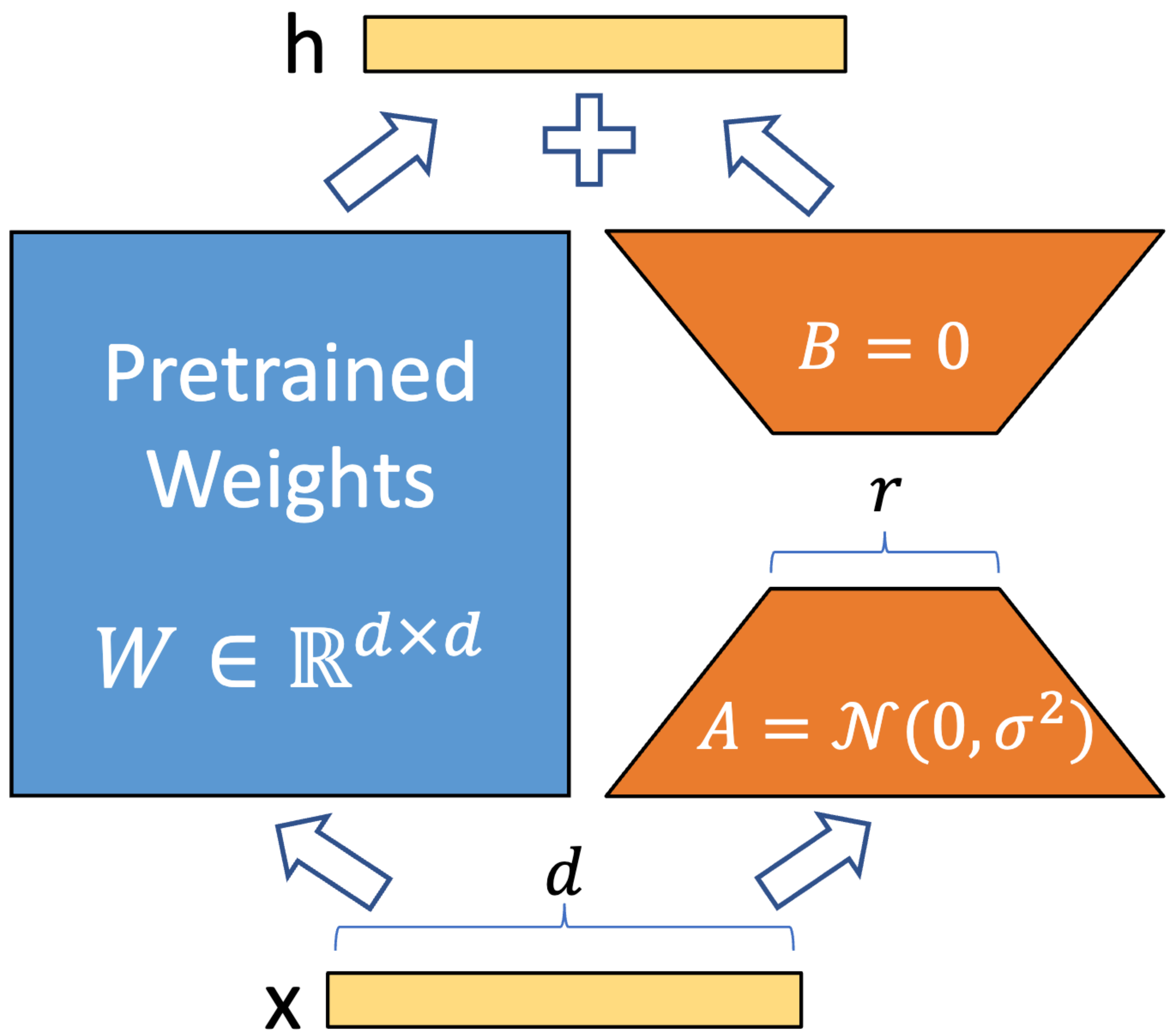


LoRA

- **Idea.** Add low-rank updates to the model

$$f(x; W) \quad \Rightarrow \quad f(x; W + BA)$$

  - Here, $B \in \mathbb{R}^{m \times r}, A \in \mathbb{R}^{r \times n}$ with small rank r
  - Very few parameters; $mn \to r(m+n)$
  - B is initialized as $\mathbf{0}$
    - Initial model is same as "no LoRA"



Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models" ICLR 2022

LoRA

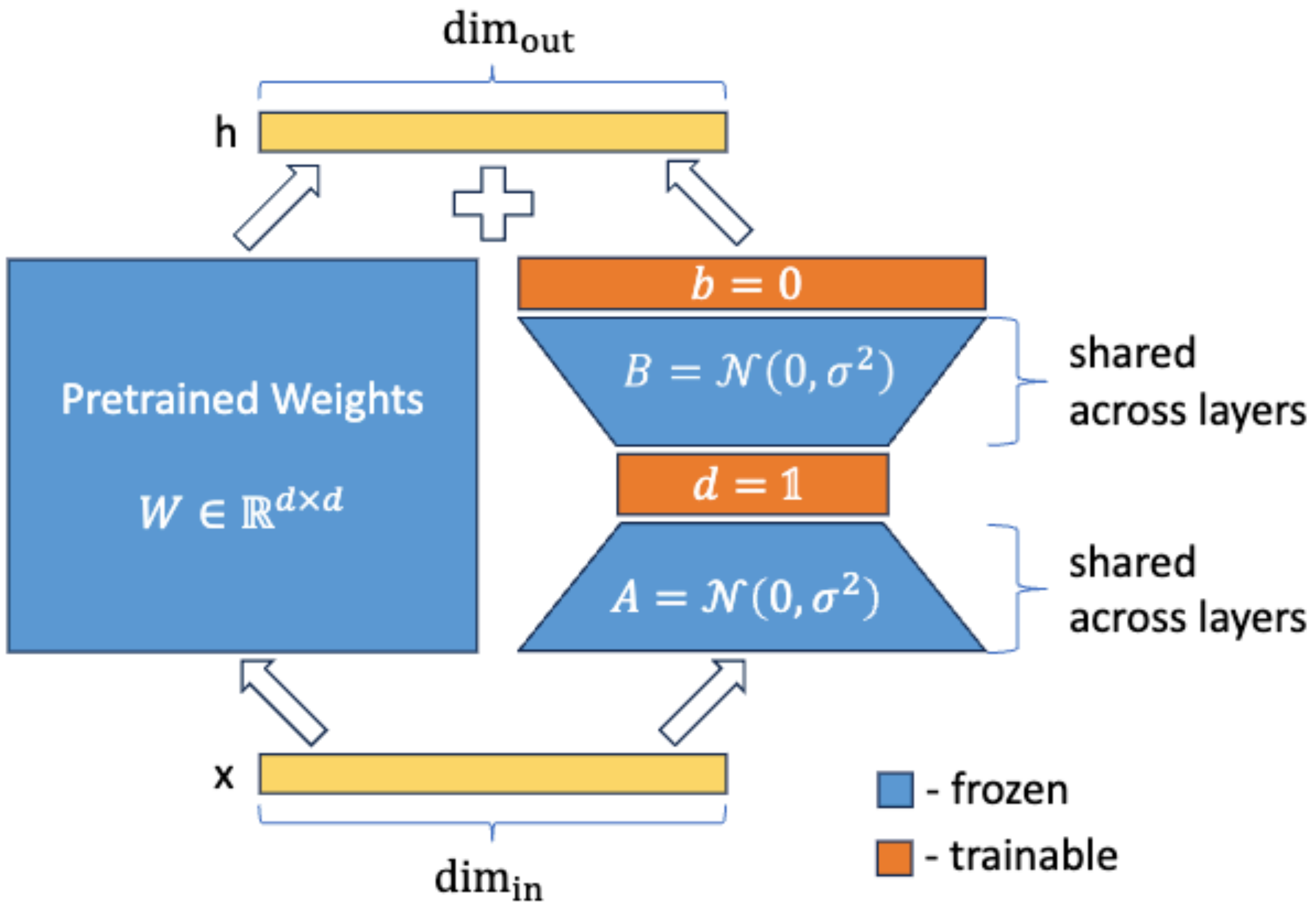
- LoRA matches or outperforms full fine-tuning
  - with very small rank, usually (e.g., 8)
  - applied only to self-attention layer

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum 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 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{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

Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models" ICLR 2022

Variants

- VeRA reduces the number of per-task parameters using random features

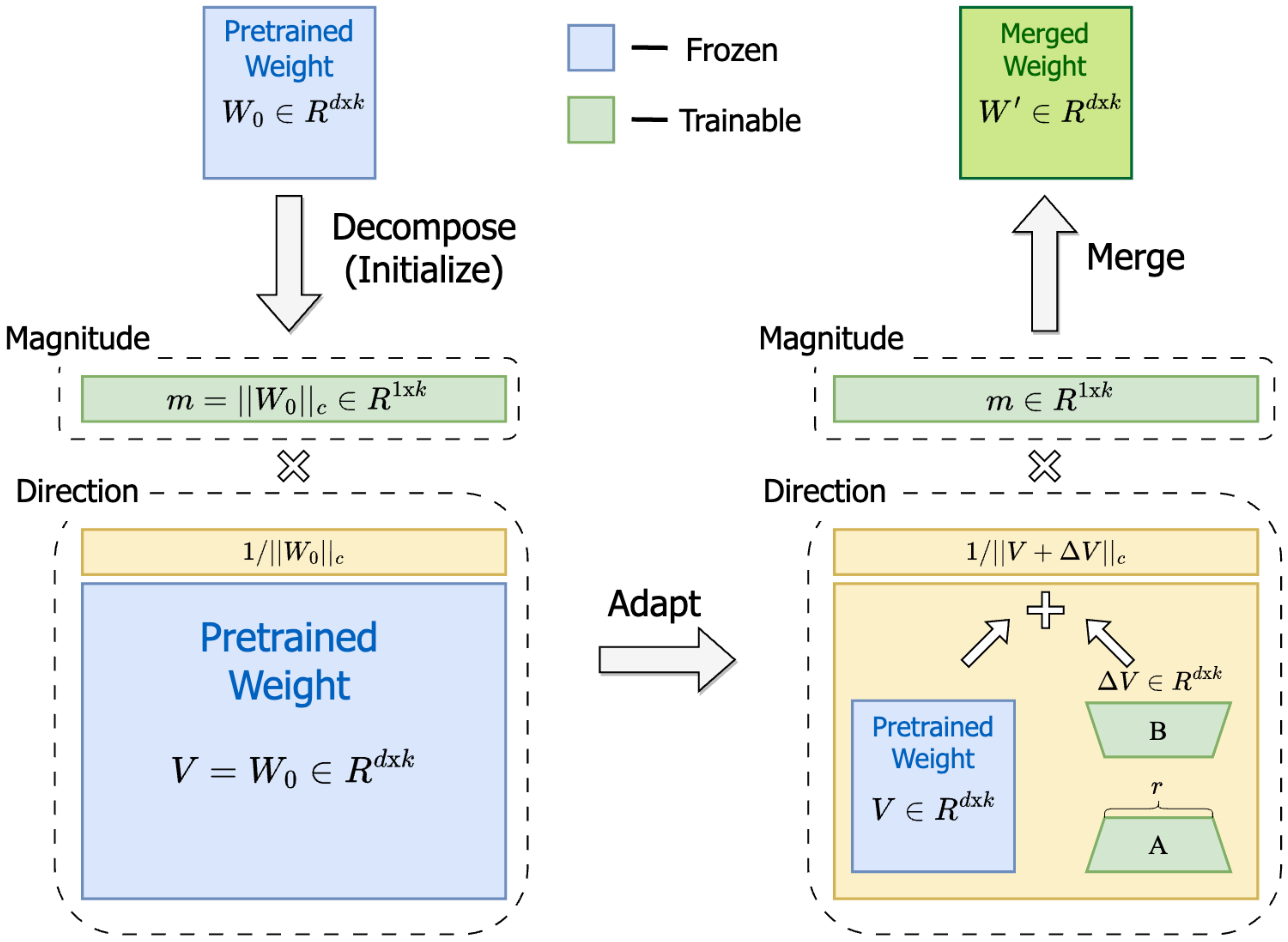


	Method	# Trainable Parameters	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
BASE	FT	125M	94.8	90.2	63.6	92.8	78.7	91.2	85.2
	BitFit	0.1M	93.7	**92.7**	62.0	91.8	81.5	90.8	85.4
	Adpt^{D}	0.3M	$94.2_{\pm 0.1}$	$88.5_{\pm 1.1}$	$60.8_{\pm 0.4}$	$93.1_{\pm 0.1}$	$71.5_{\pm 2.7}$	$89.7_{\pm 0.3}$	83.0
	Adpt^{D}	0.9M	$94.7_{\pm 0.3}$	$88.4_{\pm 0.1}$	$62.6_{\pm 0.9}$	$93.0_{\pm 0.2}$	$75.9_{\pm 2.2}$	$90.3_{\pm 0.1}$	84.2
	LoRA	0.3M	$\mathbf{95.1}_{\pm 0.2}$	$89.7_{\pm 0.7}$	$63.4_{\pm 1.2}$	$\mathbf{93.3}_{\pm 0.3}$	$\mathbf{86.6}_{\pm 0.7}$	$\mathbf{91.5}_{\pm 0.2}$	**86.6**
	VeRA	**0.043M**	$94.6_{\pm 0.1}$	$89.5_{\pm 0.5}$	$\mathbf{65.6}_{\pm 0.8}$	$91.8_{\pm 0.2}$	$78.7_{\pm 0.7}$	$90.7_{\pm 0.2}$	85.2
LARGE	Adpt^{P}	3M	$96.1_{\pm 0.3}$	$90.2_{\pm 0.7}$	$\mathbf{68.3}_{\pm 1.0}$	$\mathbf{94.8}_{\pm 0.2}$	$83.8_{\pm 2.9}$	$92.1_{\pm 0.7}$	87.6
	Adpt^{P}	0.8M	$\mathbf{96.6}_{\pm 0.2}$	$89.7_{\pm 1.2}$	$67.8_{\pm 2.5}$	$\mathbf{94.8}_{\pm 0.3}$	$80.1_{\pm 2.9}$	$91.9_{\pm 0.4}$	86.8
	Adpt^{H}	6M	$96.2_{\pm 0.3}$	$88.7_{\pm 2.9}$	$66.5_{\pm 4.4}$	$94.7_{\pm 0.2}$	$83.4_{\pm 1.1}$	$91.0_{\pm 1.7}$	86.8
	Adpt^{H}	0.8M	$96.3_{\pm 0.5}$	$87.7_{\pm 1.7}$	$66.3_{\pm 2.0}$	$94.7_{\pm 0.2}$	$72.9_{\pm 2.9}$	$91.5_{\pm 0.5}$	84.9
	LoRA-FA	3.7M	96.0	90.0	68.0	94.4	86.1	92.0	87.7
	LoRA	0.8M	$96.2_{\pm 0.5}$	$90.2_{\pm 1.0}$	$68.2_{\pm 1.9}$	$\mathbf{94.8}_{\pm 0.3}$	$85.2_{\pm 1.1}$	$\mathbf{92.3}_{\pm 0.5}$	**87.8**
	VeRA	**0.061M**	$96.1_{\pm 0.1}$	$\mathbf{90.9}_{\pm 0.7}$	$68.0_{\pm 0.8}$	$94.4_{\pm 0.2}$	$\mathbf{85.9}_{\pm 0.7}$	$91.7_{\pm 0.8}$	**87.8**

Kopiczko et al., "VeRA: Vector-based Random Matrix Adaptation," ICLR 2024

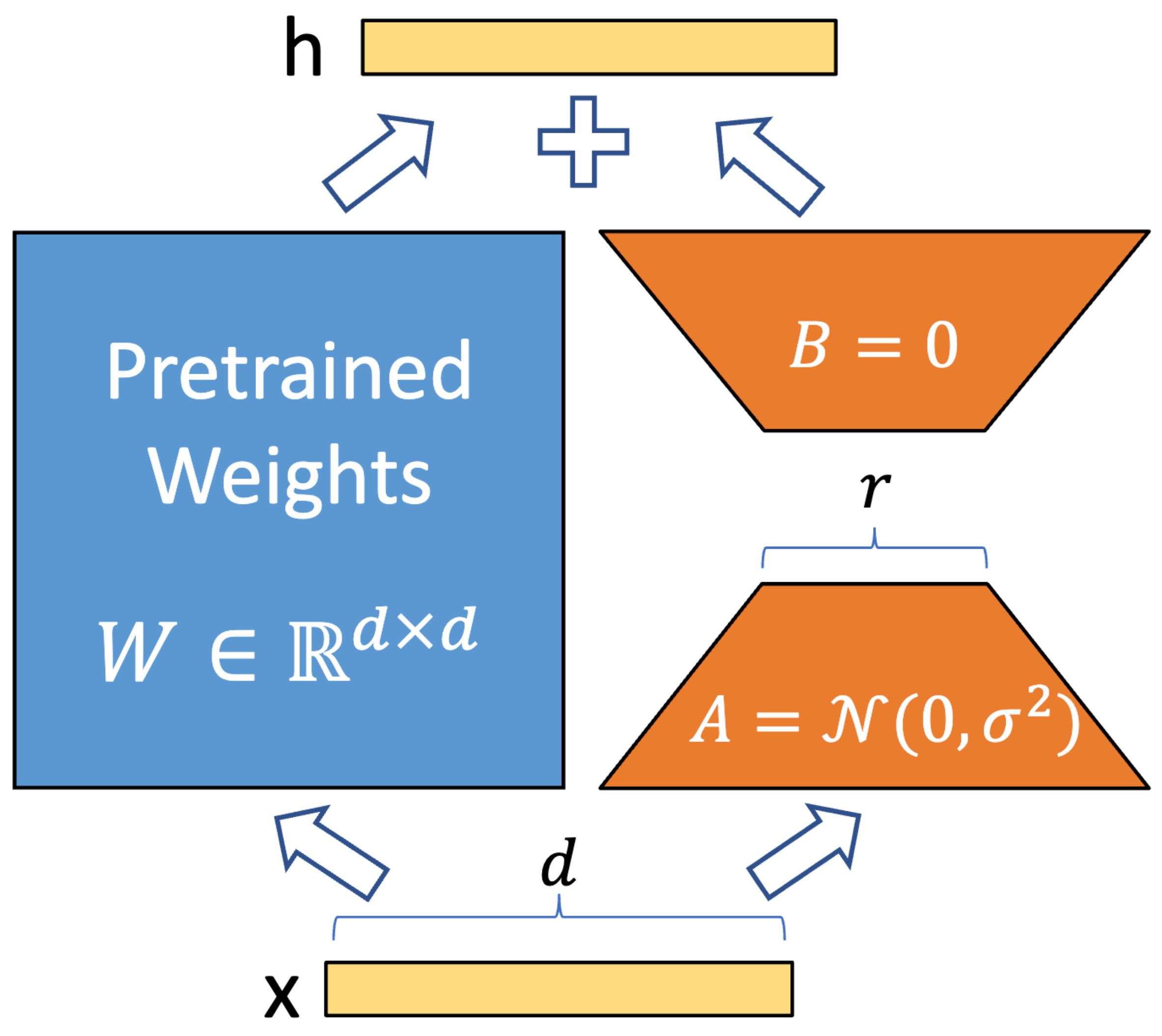
Variants

- DoRA additionally separates out magnitude vector (related: training only scale&shift works well for CNNs; Frankle et al., 2021)
  - Much lower rank needed
- Intuition. Plays a similar role as "weight/batch normalization," which makes the loss Hessian closer to the identity matrix
  - Thus enhances training



Frankle et al., "Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs," ICLR 2021
Liu et al., "DoRA: Weight-Decomposed Low-Rank Adaptation" ICML 2024

Further ideas



- Initialization matters in LoRA
  - Is $\mathbf{0}$ initialization optimal?
  - Suppose that $f(x) = w_2 w_1 x,$ and $\ell(y, f(x)) = (y - f(x))^2/2$
    - $\Delta w_2 = (f(x) - y) w_1 x$
    - $\Delta w_1 = (f(x) - y) w_2 x$ $\Leftarrow$ 0 (very slow training)
    - $\Delta(w_2 w_1) \approx (f(x) - y)(w_1^2 + w_2^2)x$
      (thus, rescaling $(w_1, w_2) \to (cw_1, w_2/c)$ changes)

Further readings

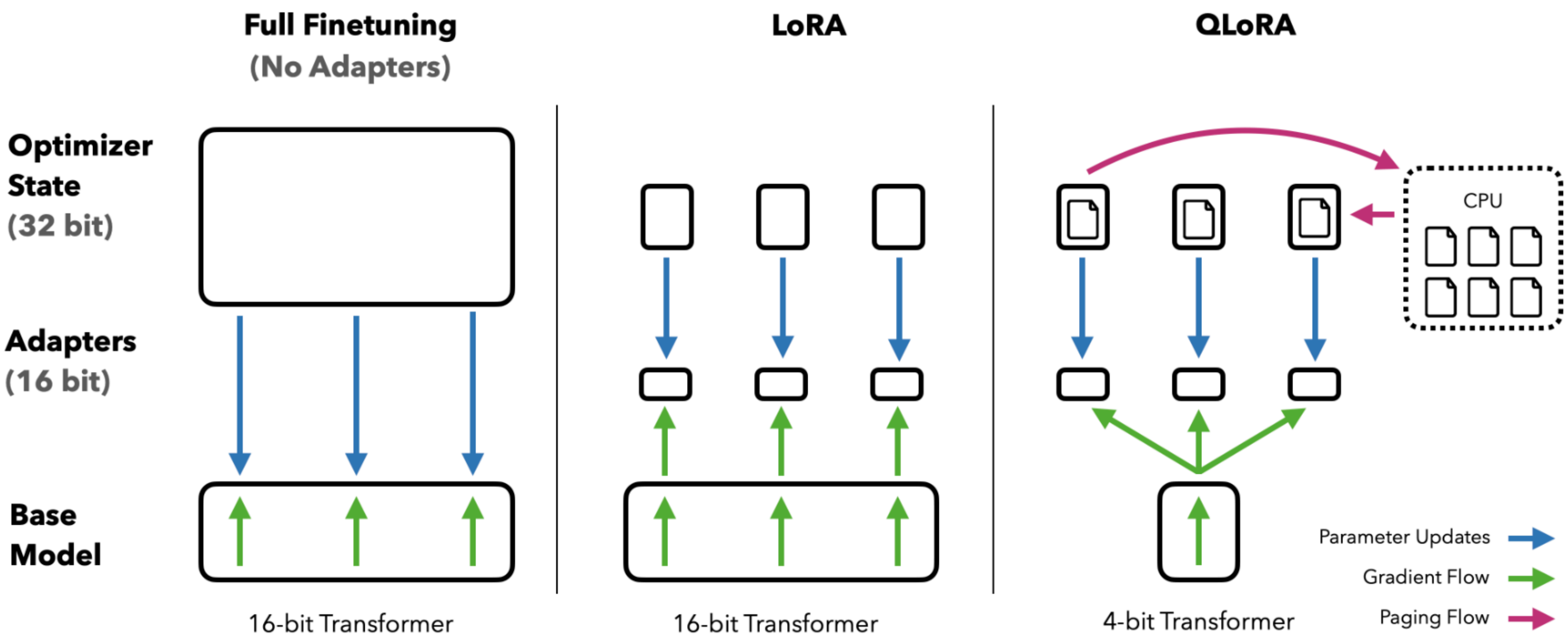
- The Impact of initialization on LoRA Finetuning Dynamics
  - https://arxiv.org/abs/2406.08447
- PiSSA
  - https://arxiv.org/abs/2404.02948
- MiLoRA
  - https://arxiv.org/abs/2406.09044

Other Ideas

QLoRA

- **Motivation.** LoRA still requires much memory for loading weight parameters of the backbone model

- **Idea.** Quantize the backbone to a smaller size

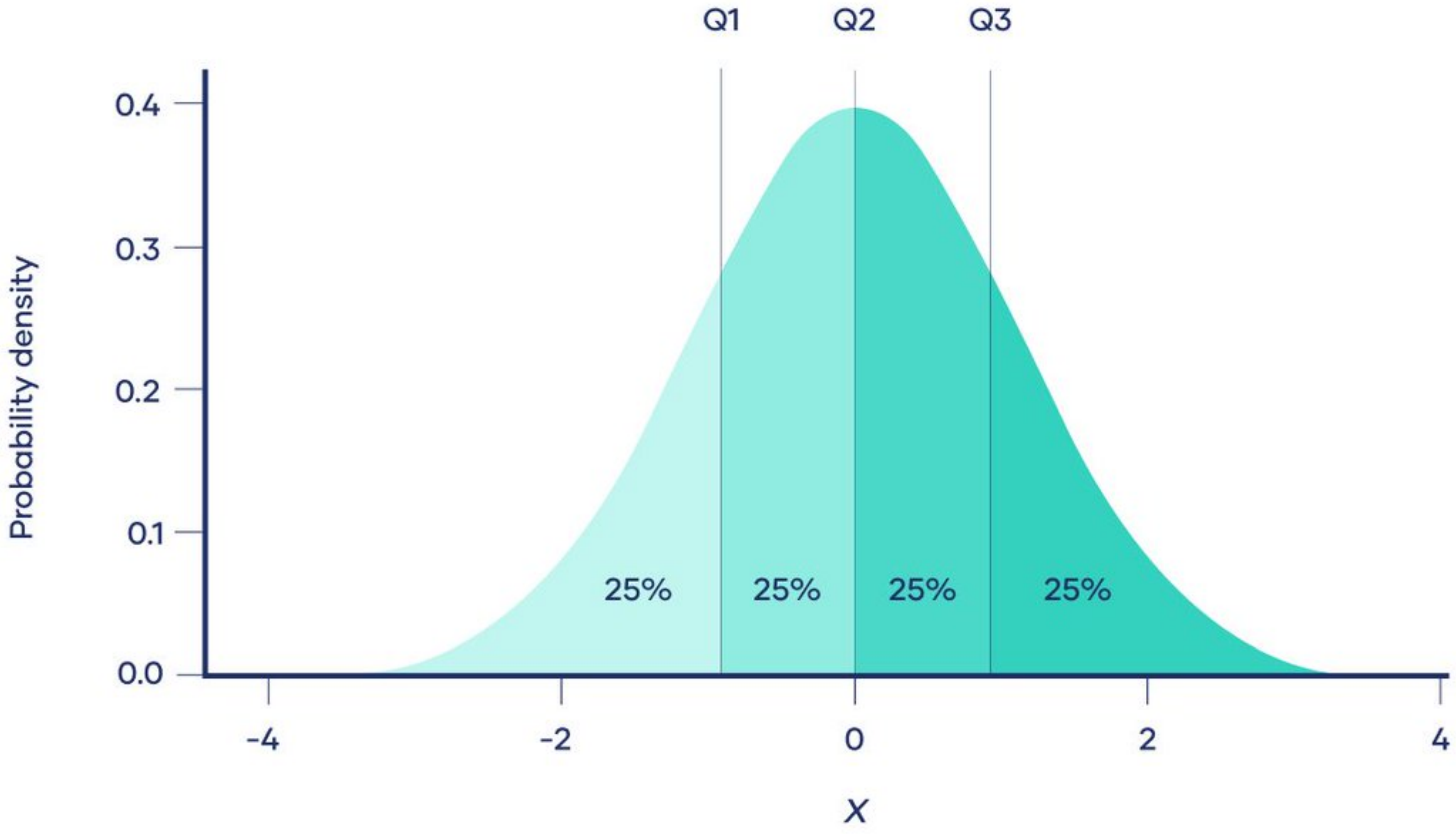


Dettmers et al., "QLoRA: Efficient Finetuning of Quantized LLMs" NeurIPS 2023

QLoRA

- QLoRA introduces several tricks
  - **NormalFloat4.** A new 4-bit format that assigns similar number of elements in each quantized bin (given that data is normally distributed)
  - **Double quantization.** Quantize the scale factors
  - **PagedOptimizer.** Fast GPU-CPU transfers of the optimizer states

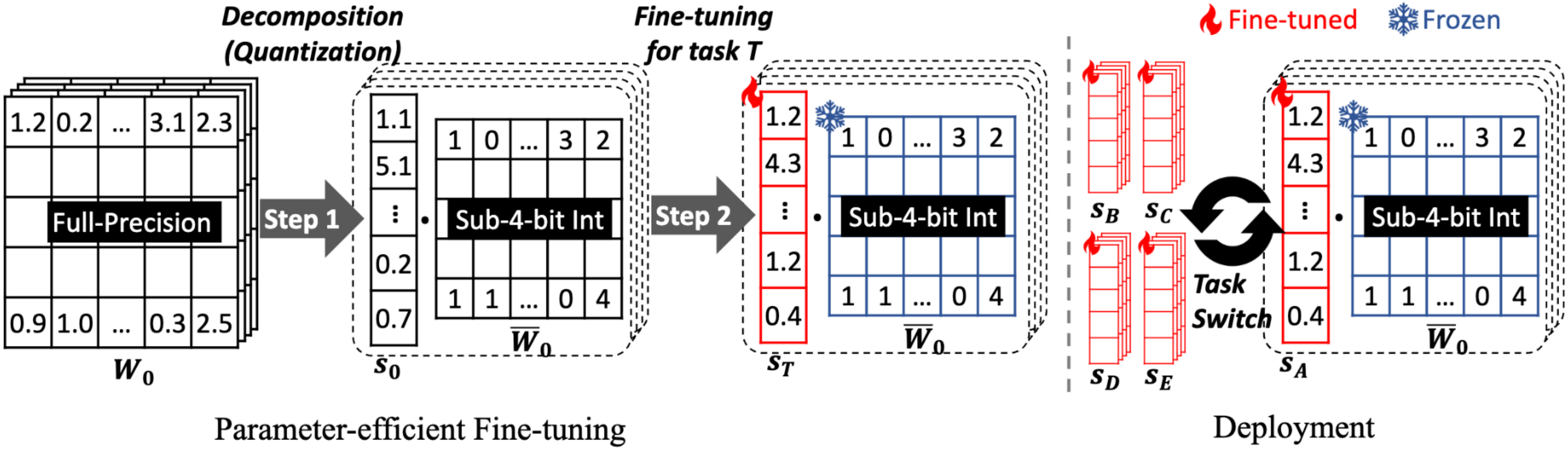
  (will be discussed later)



Dettmers et al., "QLoRA: Efficient Finetuning of Quantized LLMs" NeurIPS 2023

PEQA

- Simpler modification of PTQ that aims for acceleration
- **Idea.** Fine-tune the scaling factors of PTQ-ed model



PEQA

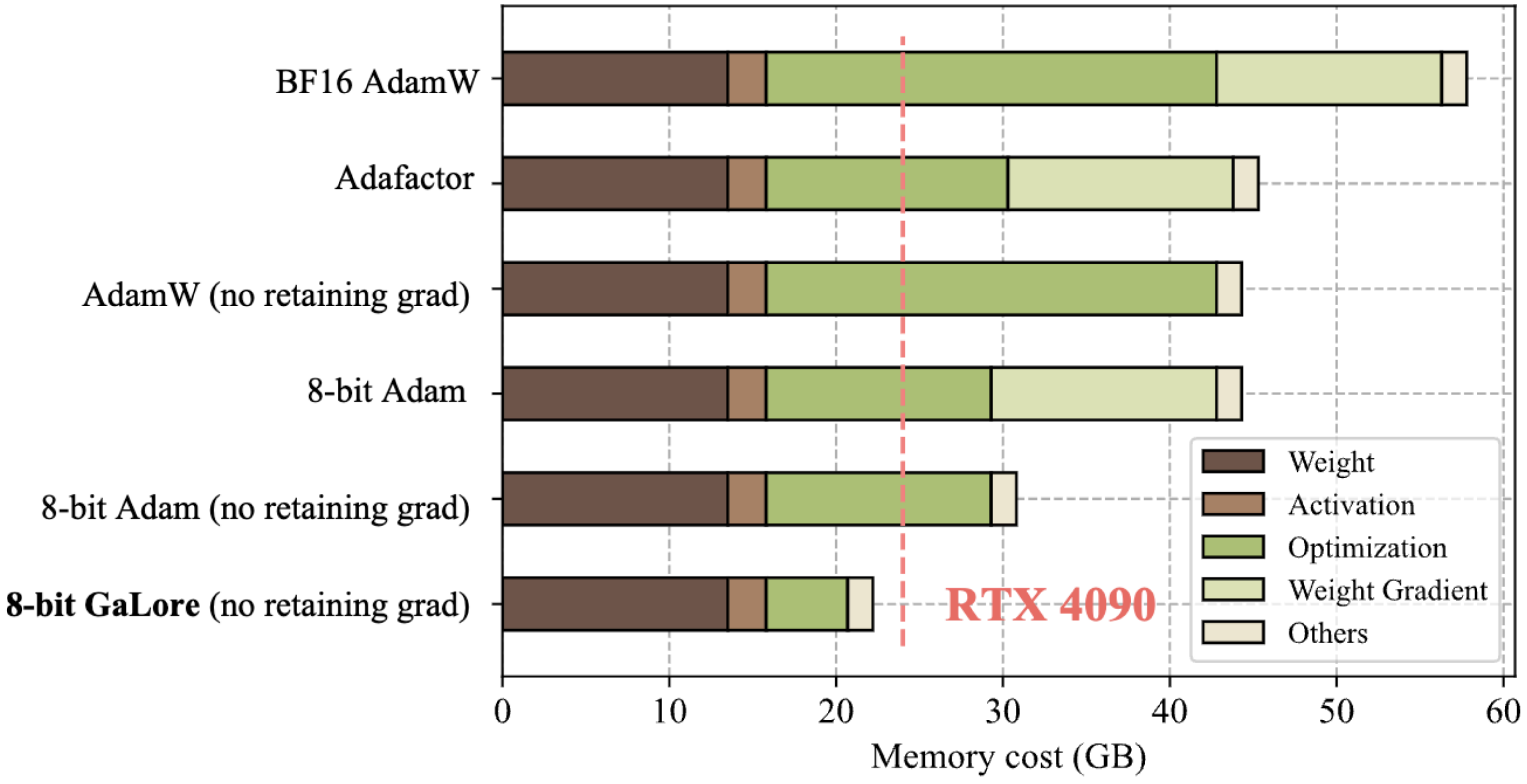
Method	DRAM (Fine-Tuning)	DRAM (Deployment)	Inference Speed	Task-Switching
Full Fine-Tuning	457GB	131GB	Slow	Slow
PEFT	131GB	131GB	Slow	Fast
PEFT+PTQ	131GB	33GB	Fast	Slow
PTQ+PEFT	33GB	33GB	Slow	Fast
PEQA (Ours)	**33GB**	**33GB**	**Fast**	**Fast**

Method	W Bits	GPT-Neo 2.7B	GPT-J 6B	LLaMA 7B	LLaMA 13B
QAT	4	11.07	8.81	5.76	5.26
LoRA + OPTQ	4	12.09	8.91	7.13	5.31
PEQA (Ours)	4	11.38	8.84	5.84	5.30
QAT	3	12.37	9.60	6.14	5.59
LoRA + OPTQ	3	21.93	11.22	19.47	7.33
PEQA (Ours)	3	12.54	9.36	6.19	5.54

Kim et al., “Memory-Efficient Fine-Tuning of Compressed Large Language Models via sub-4-bit Integer Quantization” NeurIPS 2023

GaLore

- **Motivation.** Keeping the optimizer states of Adam requires much memory
- **Idea.** Keep the weight updates full-rank, but run optimizer in projected space



Algorithm 1: GaLore, PyTorch-like

```
for weight in model.parameters():
    grad = weight.grad
    # original space -> compact space
    lor_grad = project(grad)
    # update by Adam, Adafactor, etc.
    lor_update = update(lor_grad)
    # compact space -> original space
    update = project_back(lor_update)
    weight.data += update
```

Zhao et al., "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection" ICML 2024

Further Readings

- Checkpointing for RAM savings
  - LOMO: https://arxiv.org/abs/2306.09782
- Long-Context LoRA
  - LongLoRA: https://arxiv.org/abs/2309.12307

That’s it for today 🙌