15-442/15-642: Machine Learning Systems

LLM Finetuning Techniques

Tianqi Chen and Zhihao Jia

Carnegie Mellon University

3/18/24

LLMs Need Finetuning

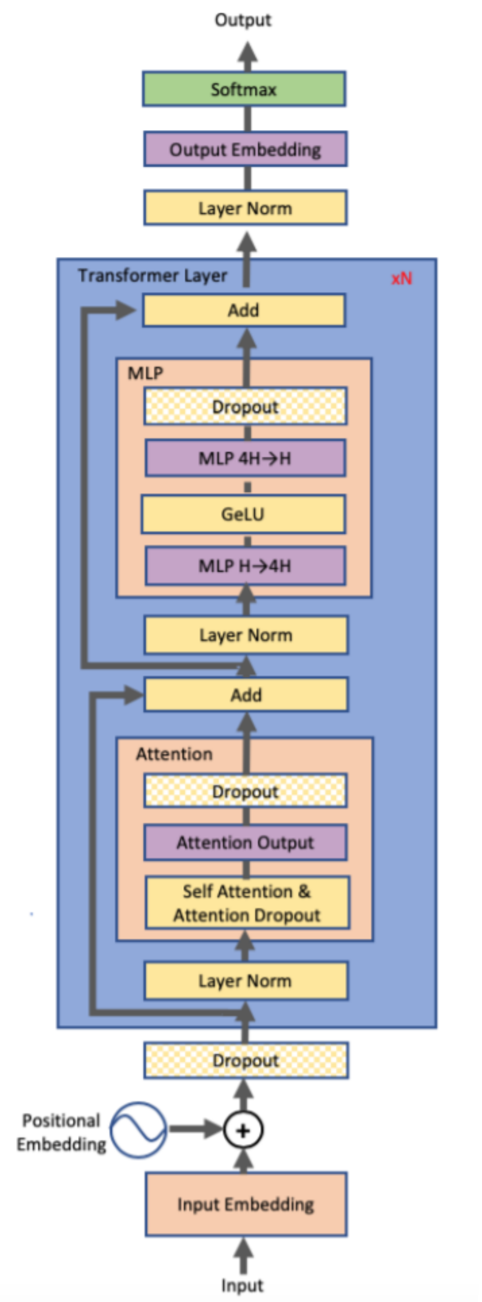
- Finetuning: start from a pretrained base model and finetune model parameters for downstream tasks

Task	Dataset	GPT-3 Few-shot	GPT-3 Finetuned
Q&A	SQuAD V2 (F1)	69.8%	88.4%
Textual Entailment	RTE (Acc)	69%	85.4%
NL2QL	WikiSQL (Acc)	20%	73%
	Spider (Acc)	18%	62%

Finetuning LLMs is Extremely Expensive

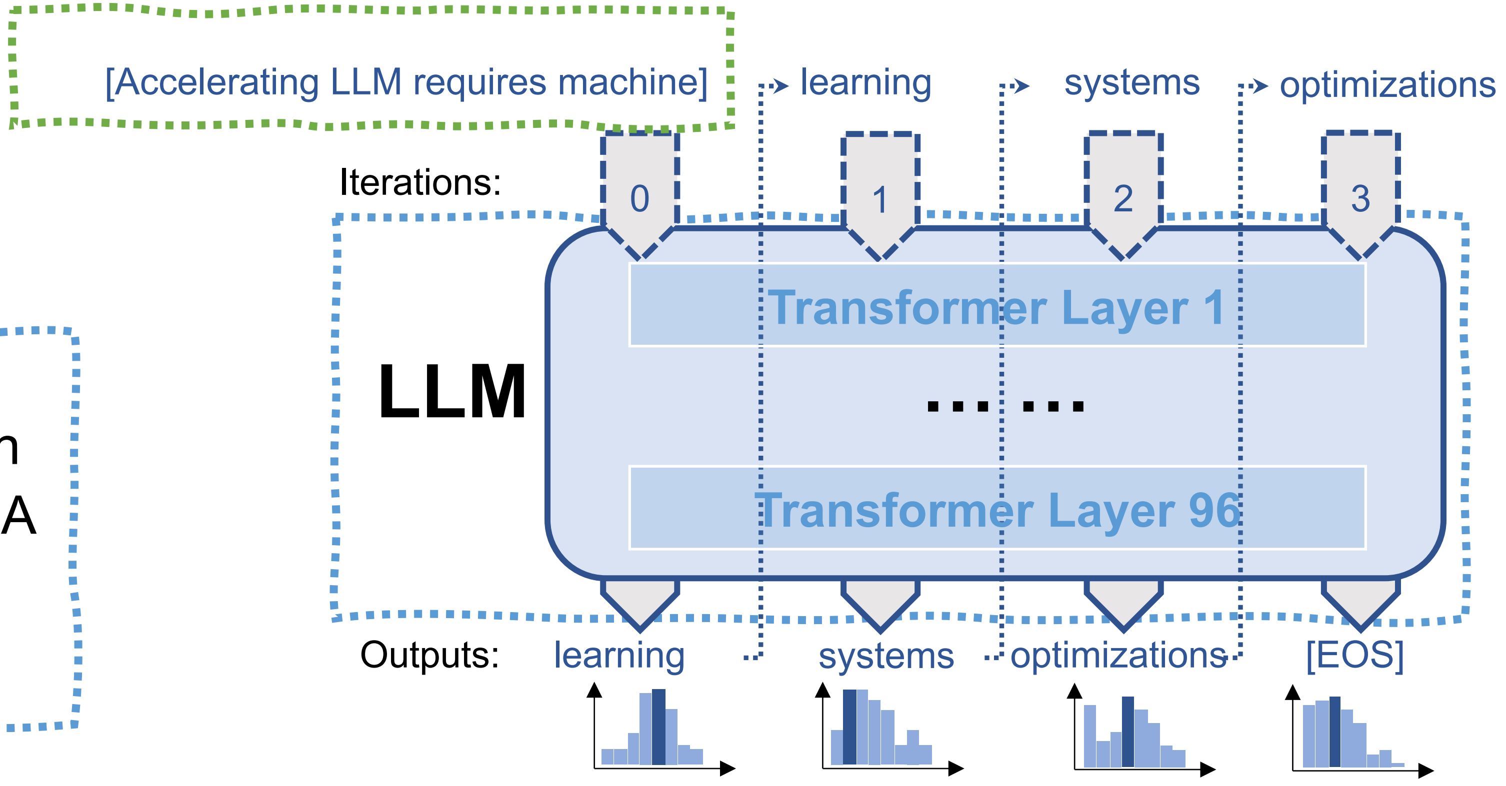
Require same resources as training from scratch:

- **Eighty** A100-40GB GPUs to finetune 175B GPT-3
- **One TB** of data per checkpoint
- **Ten** A100-40GB GPUs to serve a finetuned model



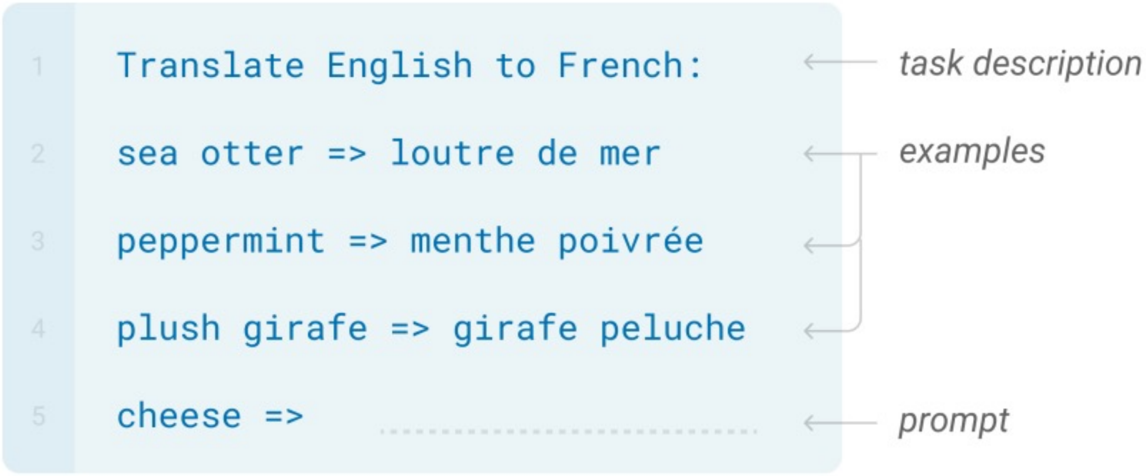
Outline: Efficient LLM Finetuning Techniques

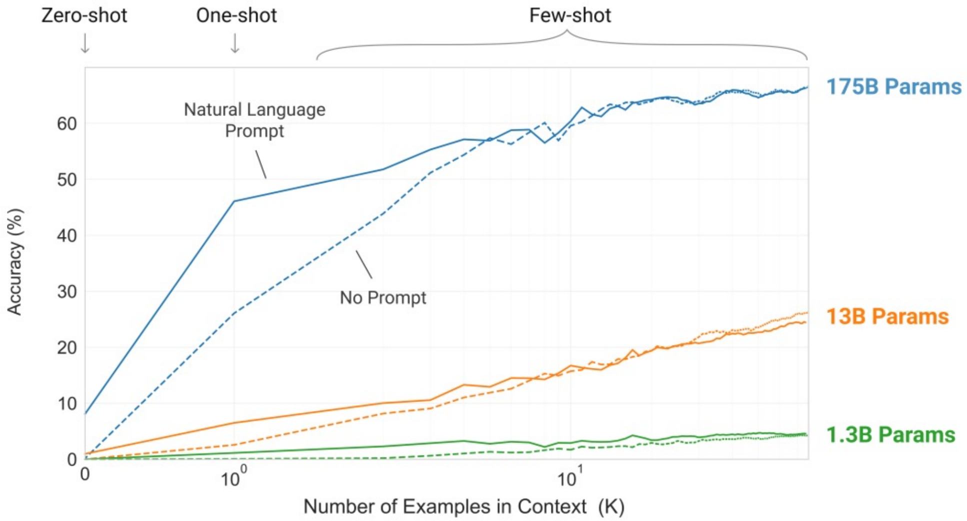
- Tuning prompts
  - Prompt engineering
  - Prefix tuning
- Tuning adapters
  - LoRA: Low-Rank Adaptation
  - QLoRA: quantization + LoRA
  - Side tuning
  - Serving adapters



LLMs: In-Context Learning

LLMs can understand task description from a few input-output examples

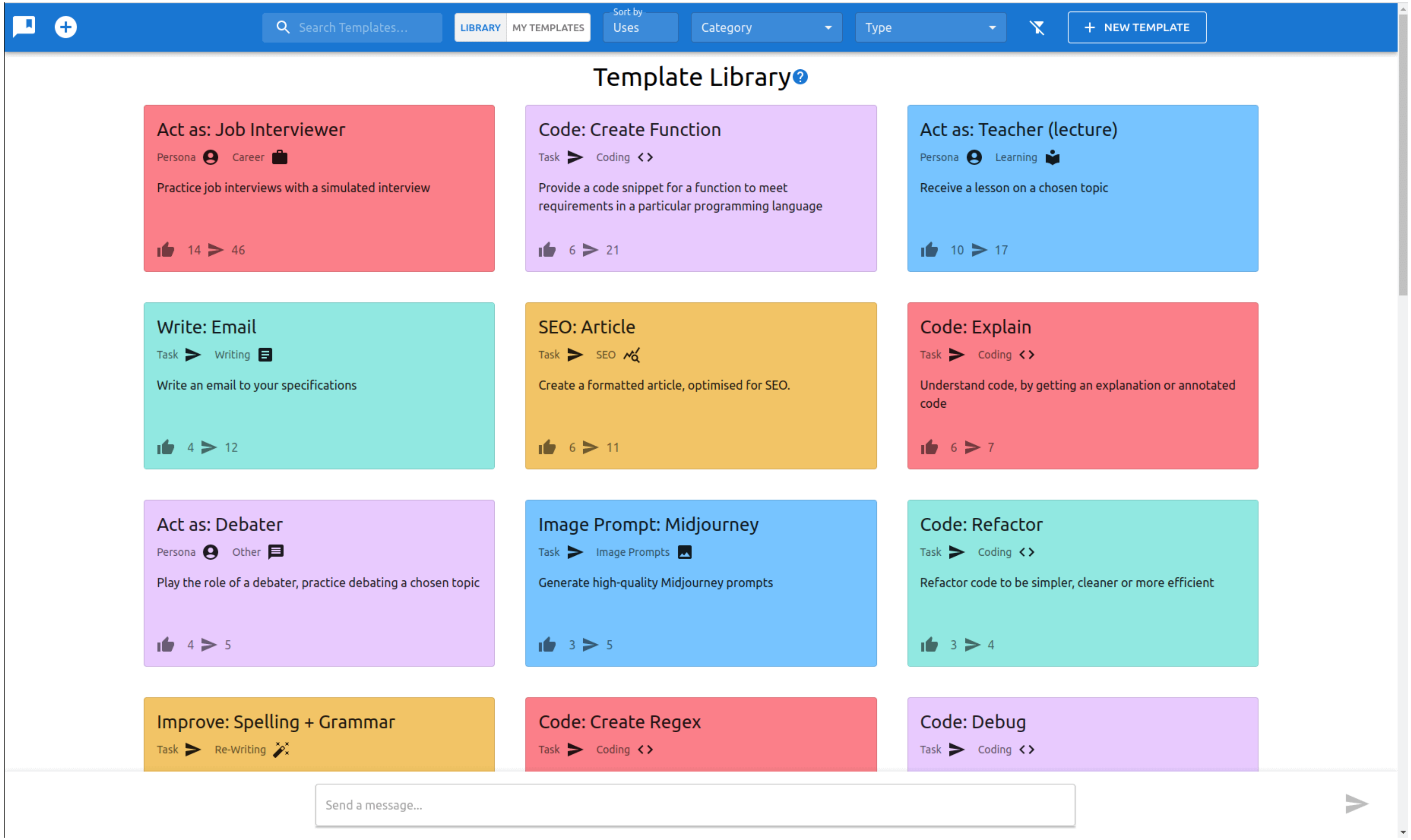




GPT-3 makes increasingly efficient use of in-context information.

In-context learning does not require task-specific training.

Prompt Engineering:
Manually Design Text Prompts for Customized Tasks



Prompt Engineering Techniques

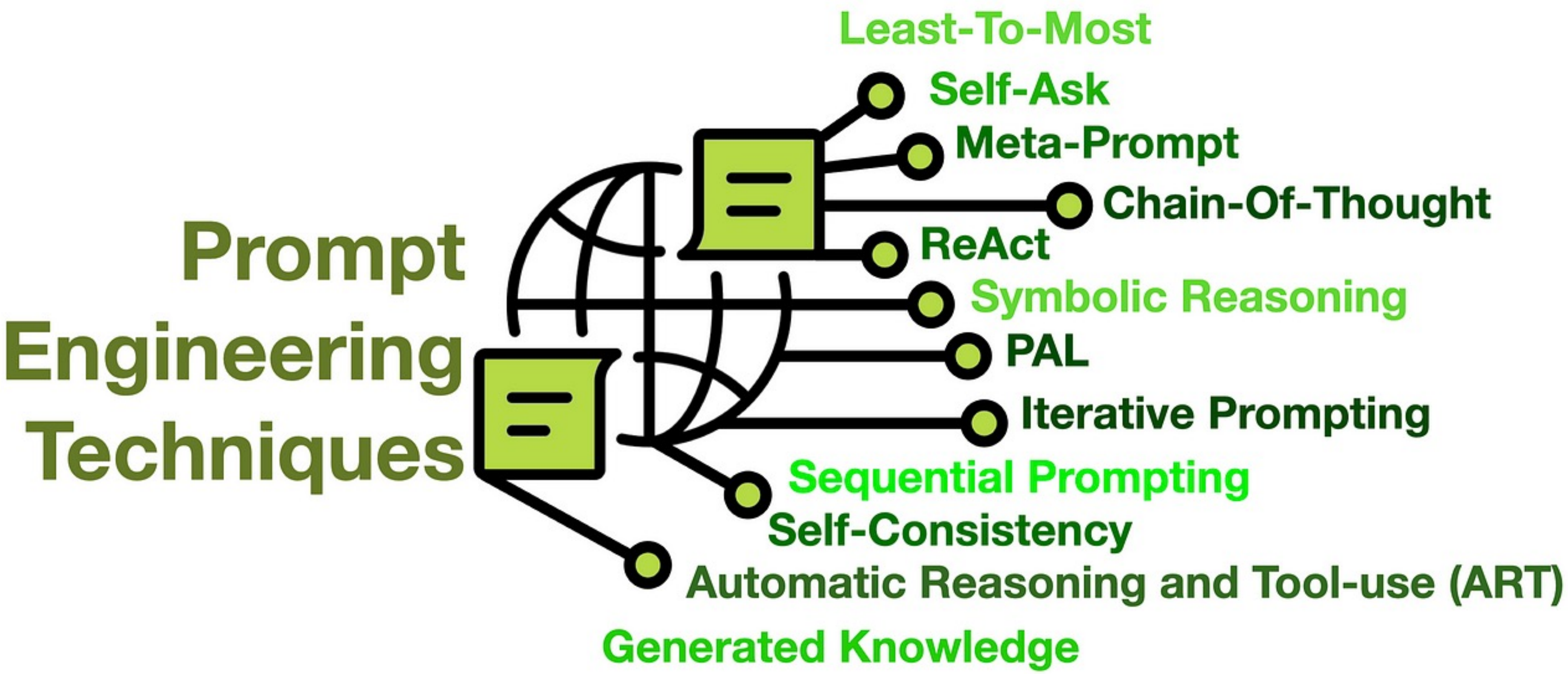
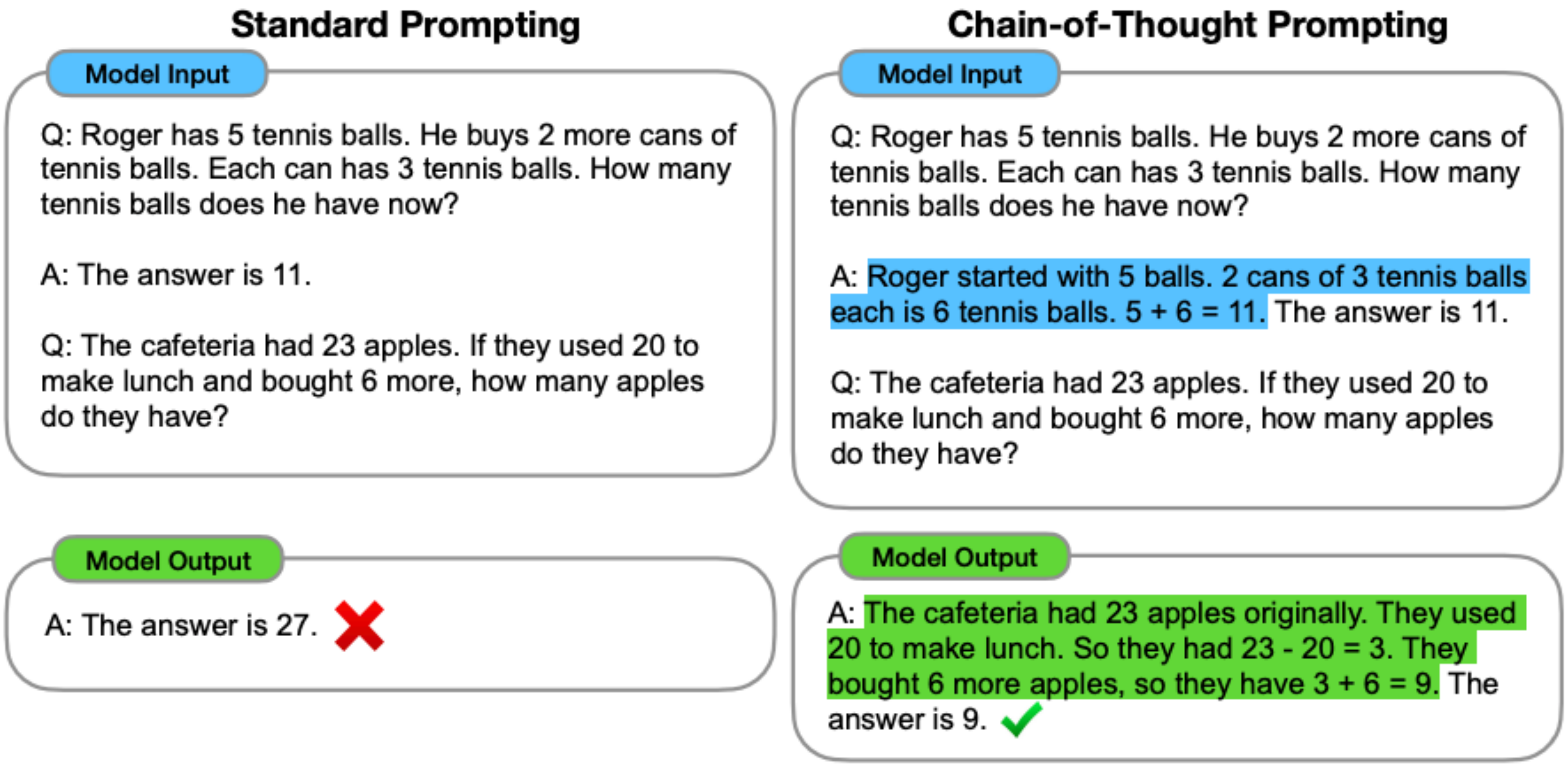


Figure credits: Cobus Greyling

Example: Chain-of-Thought Prompting

- Break a large task into sub-tasks and chain them together



Limitations of Prompt Engineering

- Hard to design prompts: require extensive effort to create a good prompt
- Non-differentiable: cannot directly finetune on a given dataset

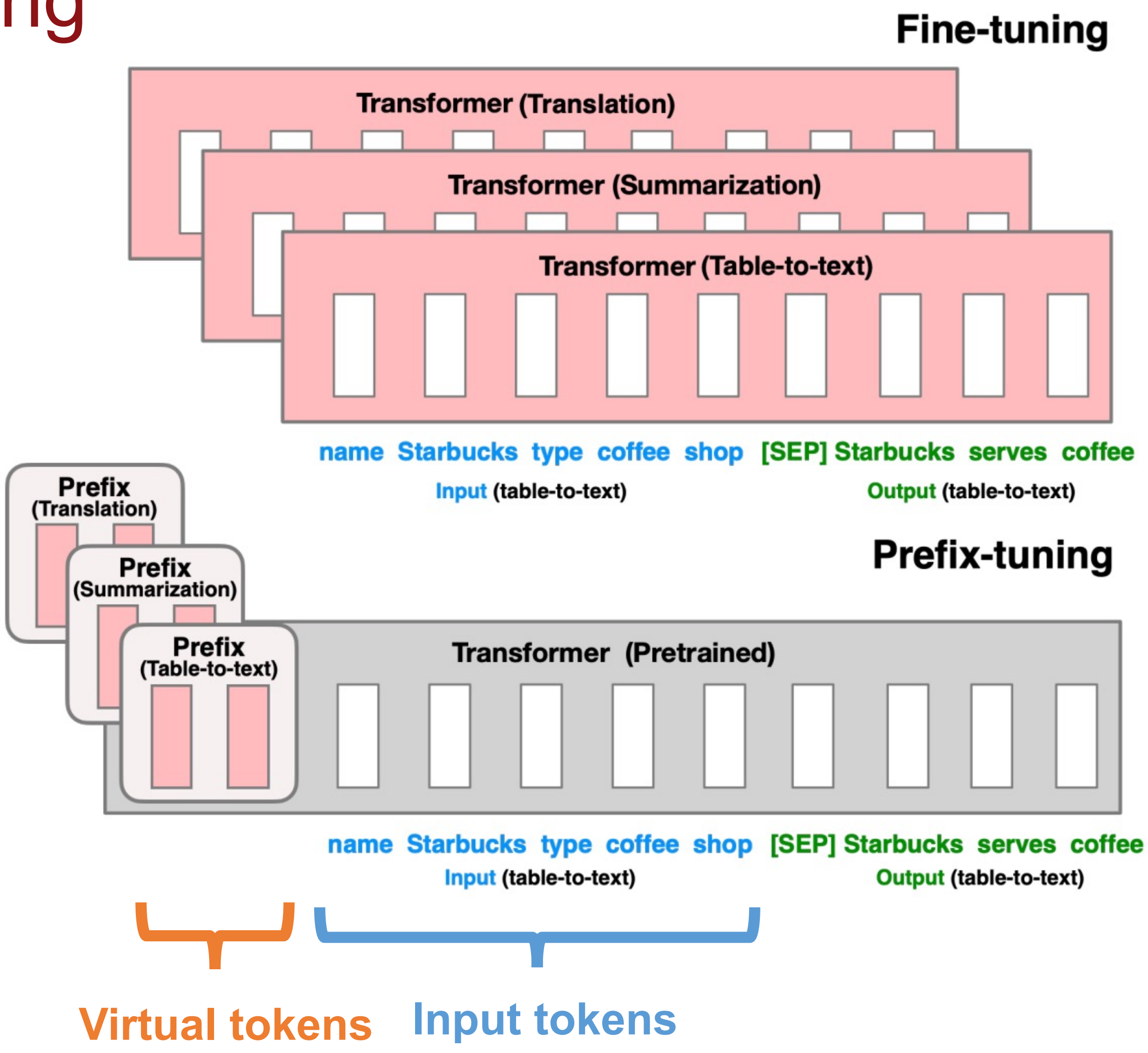
Can we make prompt trainable/differentiable?

Prompt Tuning / Prefix Tuning

Prepend a sequence of virtual tokens to the input

LLM attends to the prefix as if it were a sequence of tokens

Freeze the LLM's parameters and finetune prefix parameters



* Prefix-Tuning: Optimizing Continuous Prompts for Generation

Prompt Engineering v.s. Prompt Tuning

Prompt Engineering

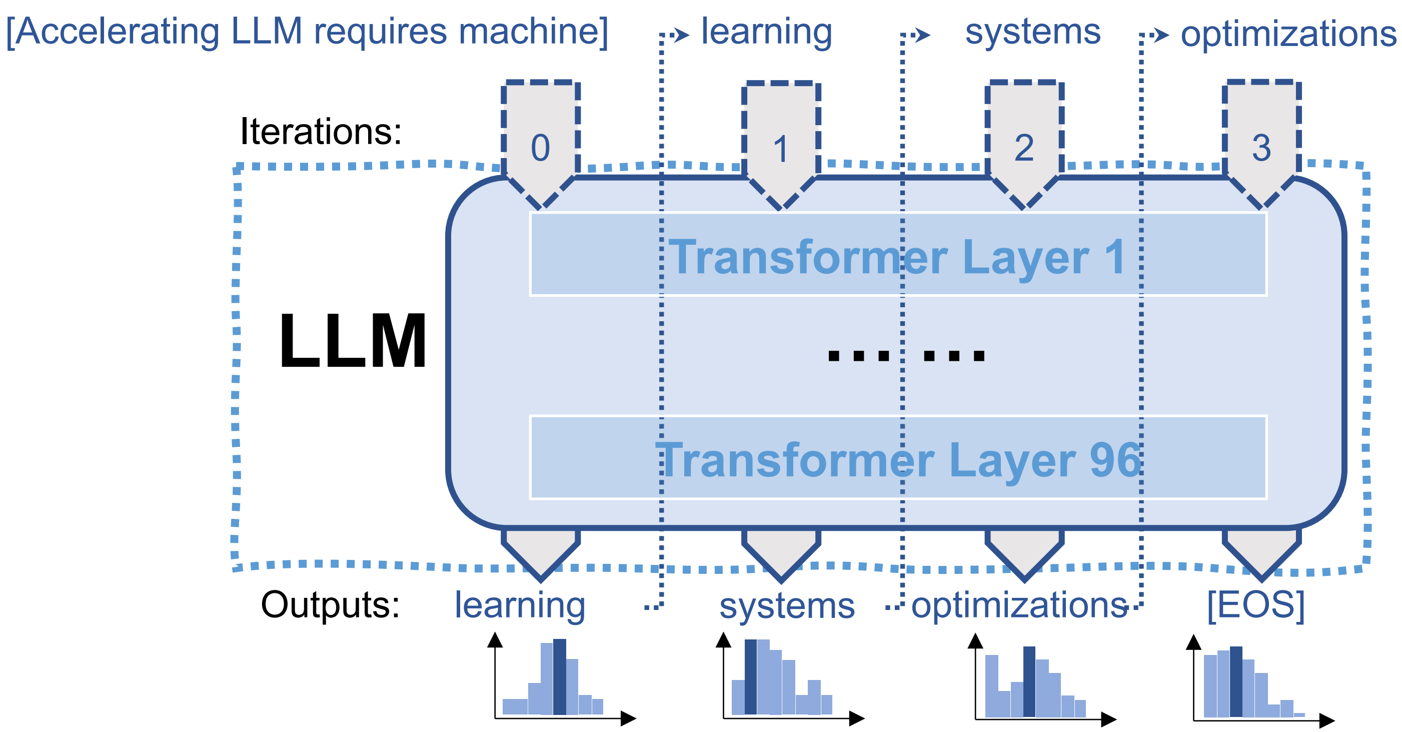
- Use a sequence of regular tokens as text prompt
- Leverage LLM’s in-context learning

Prompt Tuning

- Use a sequence of virtual tokens, each with trainable parameters
- Freeze LLM’s parameters and finetune new parameters through backpropagation

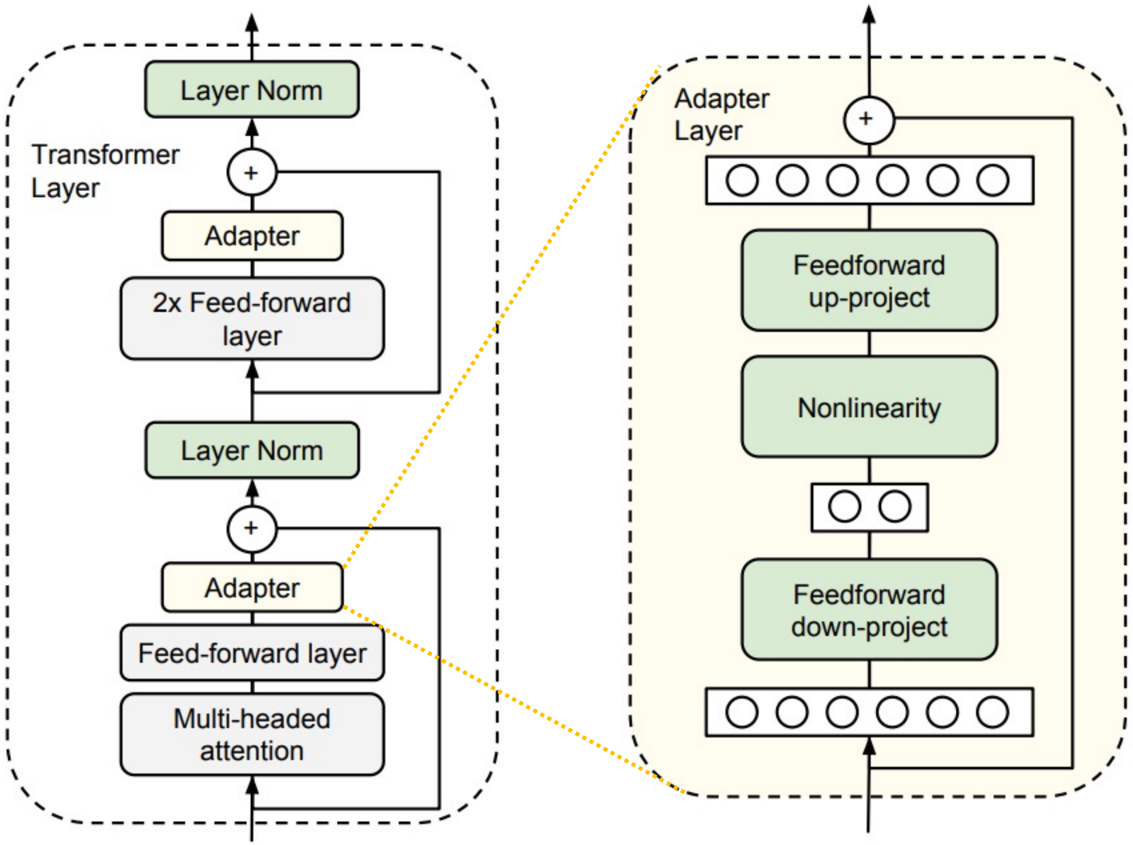
Outline: Efficient LLM Finetuning Techniques

- Tuning prompts
  - Prompt engineering
  - Prompt/prefix tuning
- **Tuning adapters**
  - LoRA: Low-Rank Adaptation
  - QLoRA: quantization + LoRA
  - Side tuning
  - Serving adapters



Adapter Tuning

- Add new adapter modules (with a few parameters) to an LLM
- During finetuning, freeze the original network and only finetune adapters



Adapter modules:

- Small number of parameters -> low memory/compute overheads
- Near-identity initialization -> stable training of the adapted model

* Parameter-Efficient Transfer Learning for NLP. 2019

Adapter Tuning

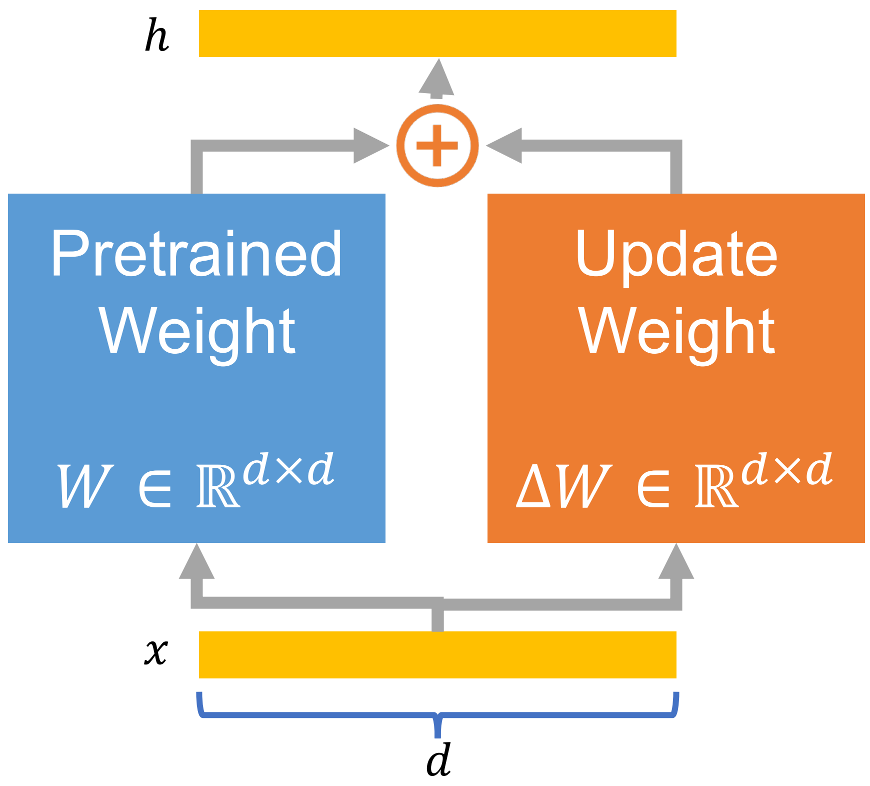
- Comparable performance as full training
- Much fewer trainable parameters

	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	MNLI_m	MNLI_mm	QNLI	RTE	Total
BERT_LARGE	$9.0\times$	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	70.1	80.4
Adapters (8-256)	$1.3\times$	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	71.5	80.0
Adapters (64)	$1.2\times$	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	68.8	79.6

* Parameter-Efficient Transfer Learning for NLP. 2019

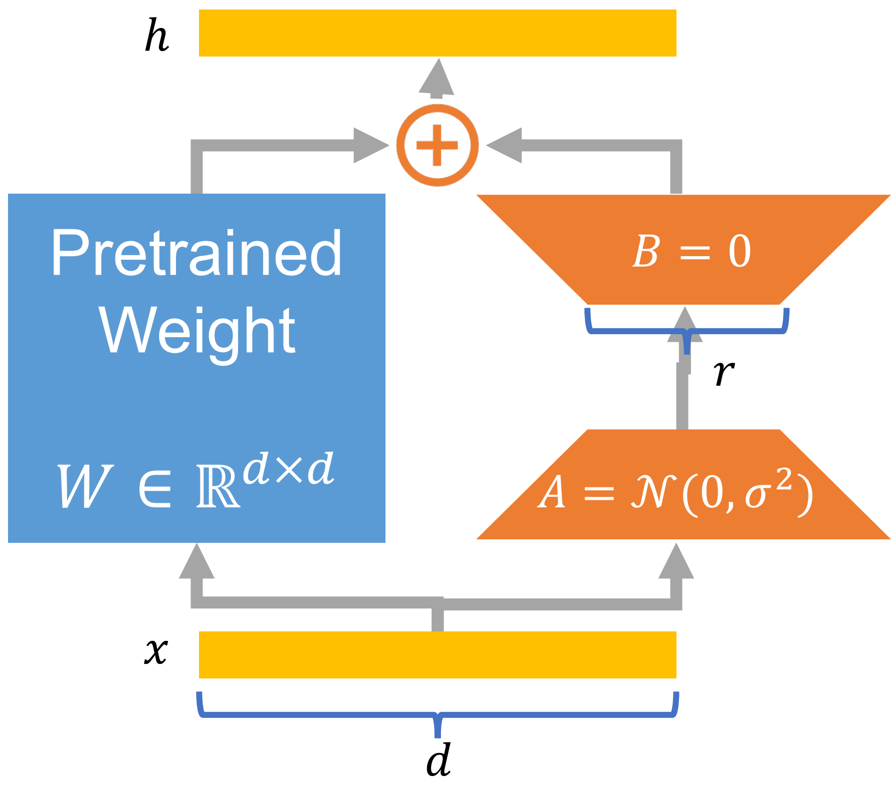
Low-Rank Adaptation (LoRA)

- Freeze pretrained model weights and inject trainable rank decomposition matrices into each layer



$$h = Wx + \Delta Wx$$

Full Finetuning

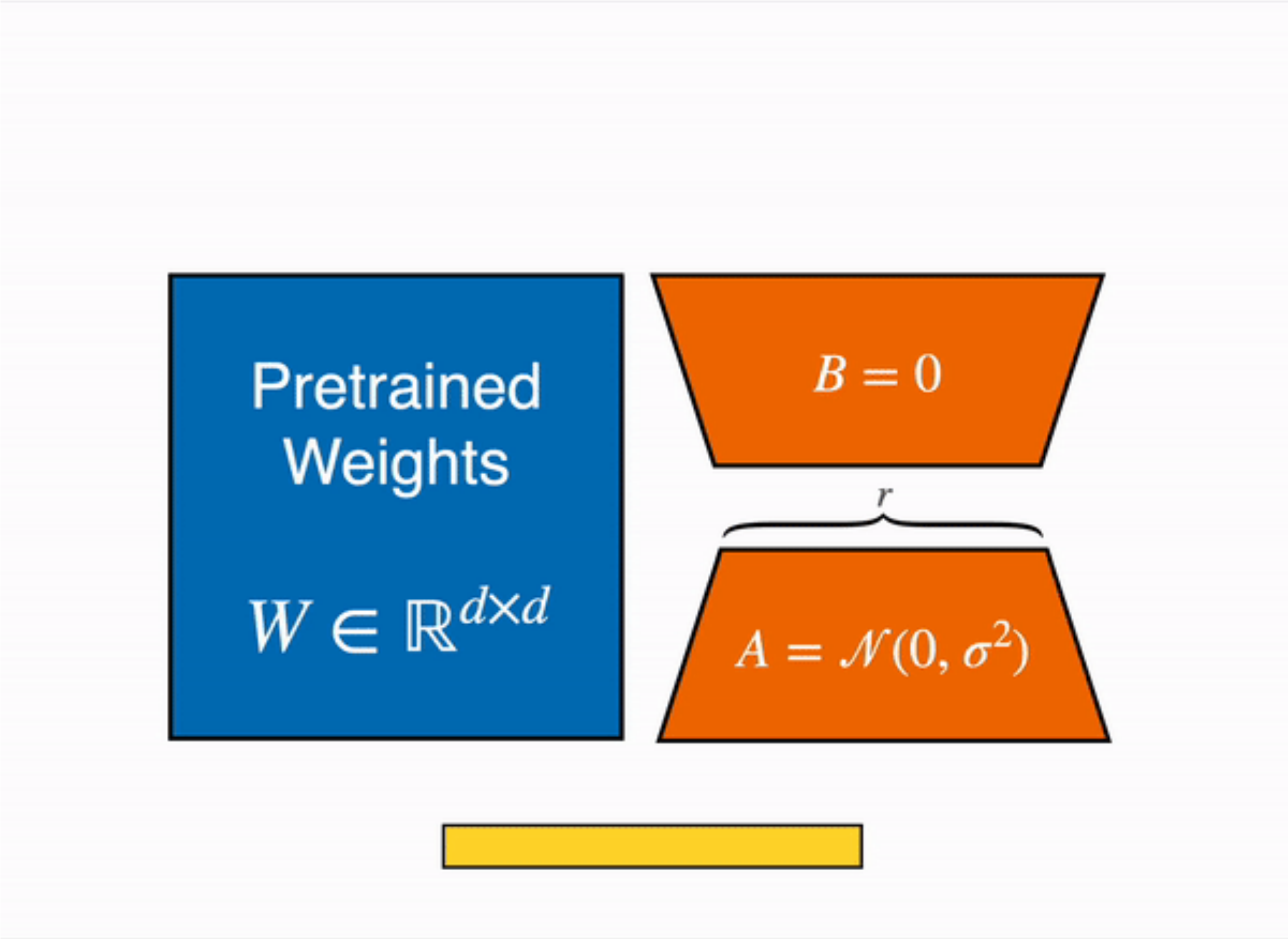


$$h = Wx + BAx$$

LoRA

* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

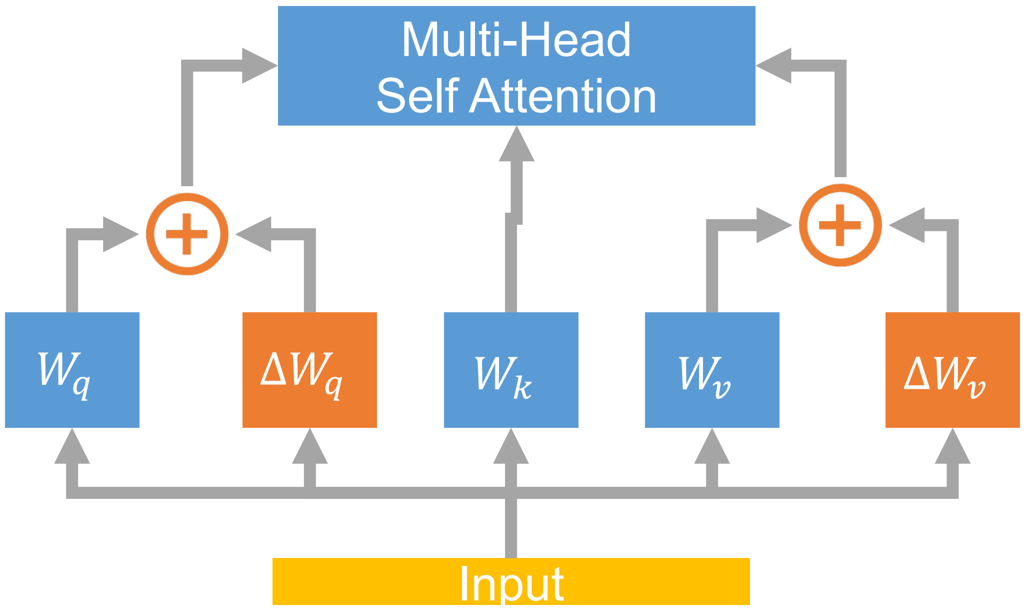
Low-Rank Adaptation (LoRA)



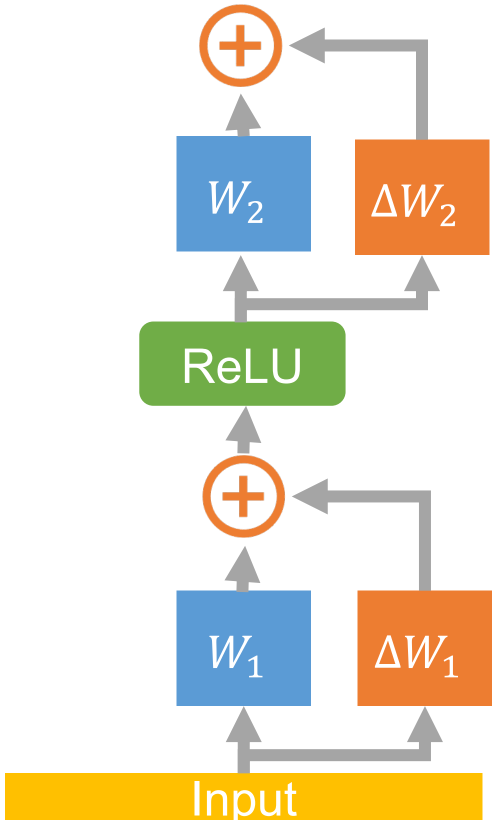
* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

Low-Rank Adaptation (LoRA)

- Apply LoRA to Attention

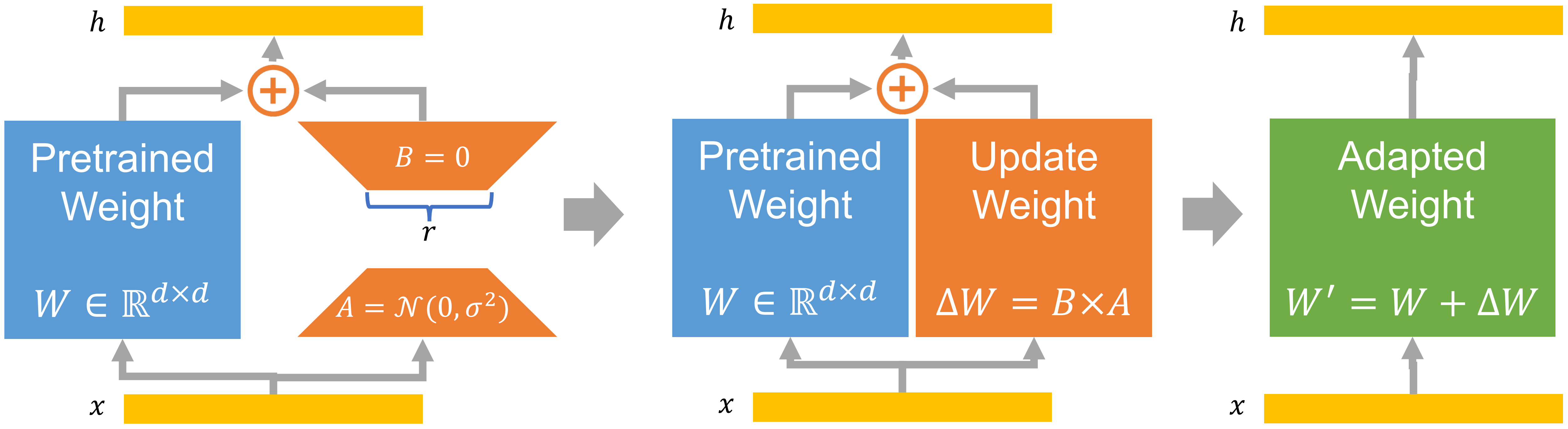


- Apply LoRA to MLP layer



* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

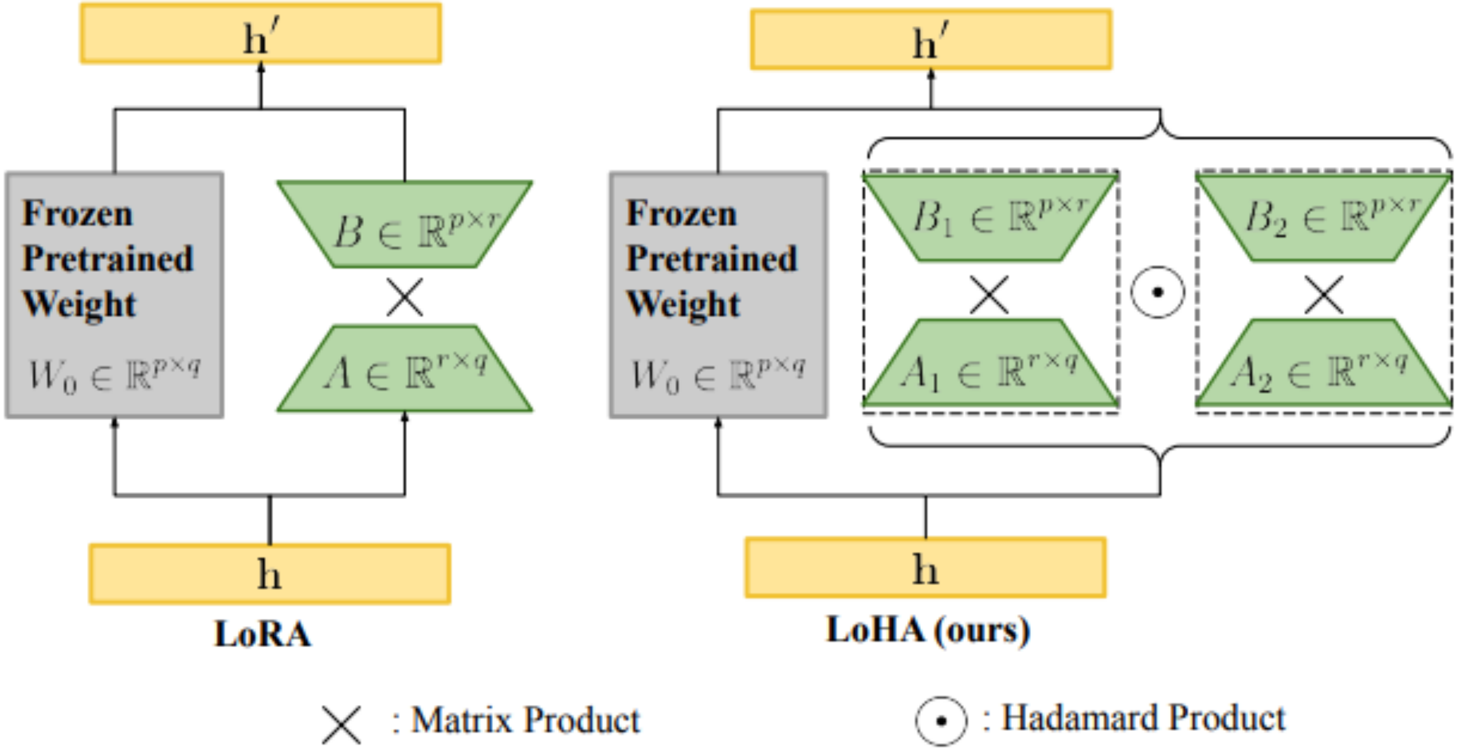
LoRA Does Not Increase Inference Latency



* LORA: Low-Rank Adaption of Large Language Models. Hu et al. 2021

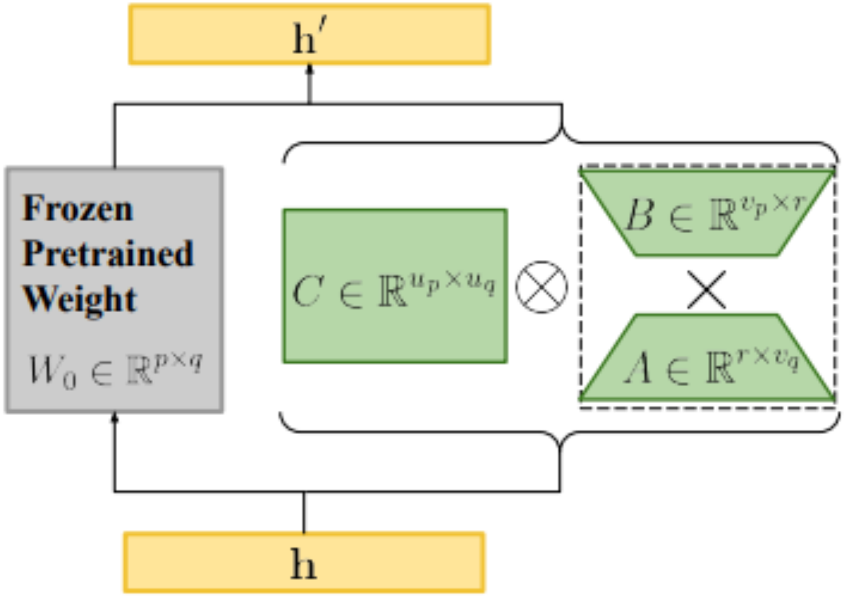
LoRA Variant 1 (LoHa): Low-Rank Hadamard Product

- Use Hadamard (element-wise) product
- ΔW can have the same number of trainable parameters but a higher rank and expressivity



LoRA Variant 2 (LoKr): Low-Rank Kronecker Product

- Replace matrix product with Kronecker product
- Preserve the rank of the original weight matrix through Kronecker product



LoKr (ours)

$\otimes$: Kronecker Product

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix},$$

Outline: LLM Finetuning Methods

- Tuning prompts
  - Prompt engineering
  - Prefix tuning
- Tuning adapters
  - LoRA: Low-Rank Adaptation
  - **QLoRA: quantization + LoRA**
  - Side tuning
  - Serving adapters

Quantization

Quantization: converting a data type into fewer bits

Example: FP32 tensor -> Int8 tensor with range [−128, 127]:

Quantization: $$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}),$$

Dequantization: $$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}}$$

Issue: when there are large magnitude values, quantization bins are not well utilized

Block-wise Quantization

Chunk input tensor into blocks that are independently quantized

Each block has its own quantization constant c_i saved in FP32

- Assume a block size of B, quantization overhead: $\frac{32}{B}$ bits per parameter
- When B=64, block-wise quantization introduces an overhead of 0.5 bits per parameter

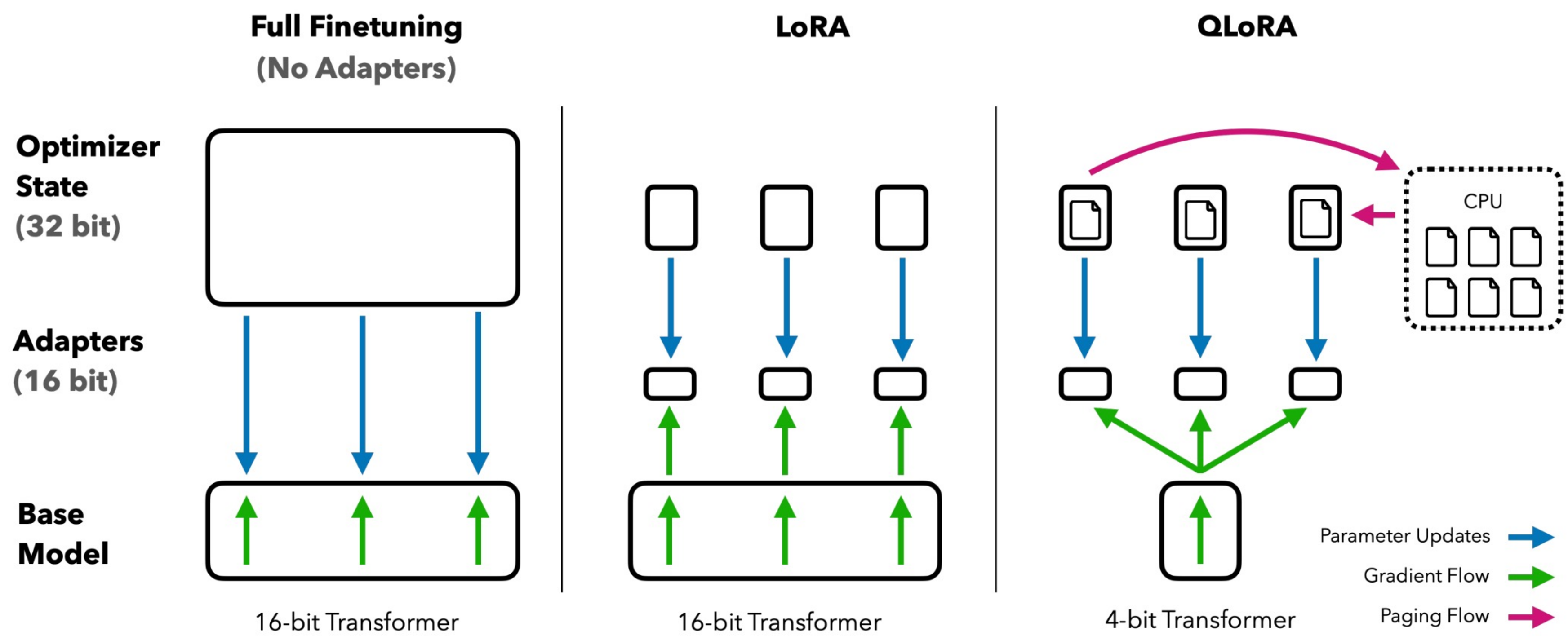
Block-wise Double Quantization

Quantization constants $C_2^{FP32}[i]$ of the first quantization as inputs to a second quantization

$$C_2^{INT8}[i] = round(\frac{127}{absmax\_k(C_2^{FP32}[k])} C_2^{FP32}[i])$$

- Use 8-bit integers with a block size of 256 for the second quantization
- Reduce the block-wise quantization overhead from 32/64=0.5 bits to 8/64+32/(64*256)=0.127 bits

QLoRA: Quantized LoRA



- QLoRA for a single linear layer

$$\underbrace{\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}}_{\text{16-bit activations}} \text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \underbrace{\mathbf{W}^{\text{NF4}}}_{\text{4-bit frozen weights}}) + \mathbf{X}^{\text{BF16}} \underbrace{\mathbf{L}_1^{\text{BF16}} \mathbf{L}_2^{\text{BF16}}}_{\text{16-bit LoRA weights}}$$

QLoRA Achieves On-Par Performance as Full Finetuning

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

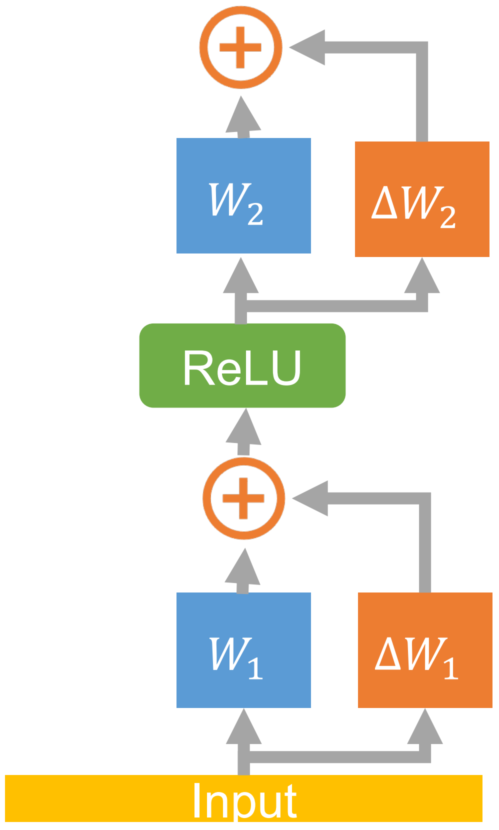
Outline: LLM Finetuning Methods

- Tuning prompts
  - Prompt engineering
  - Prefix tuning

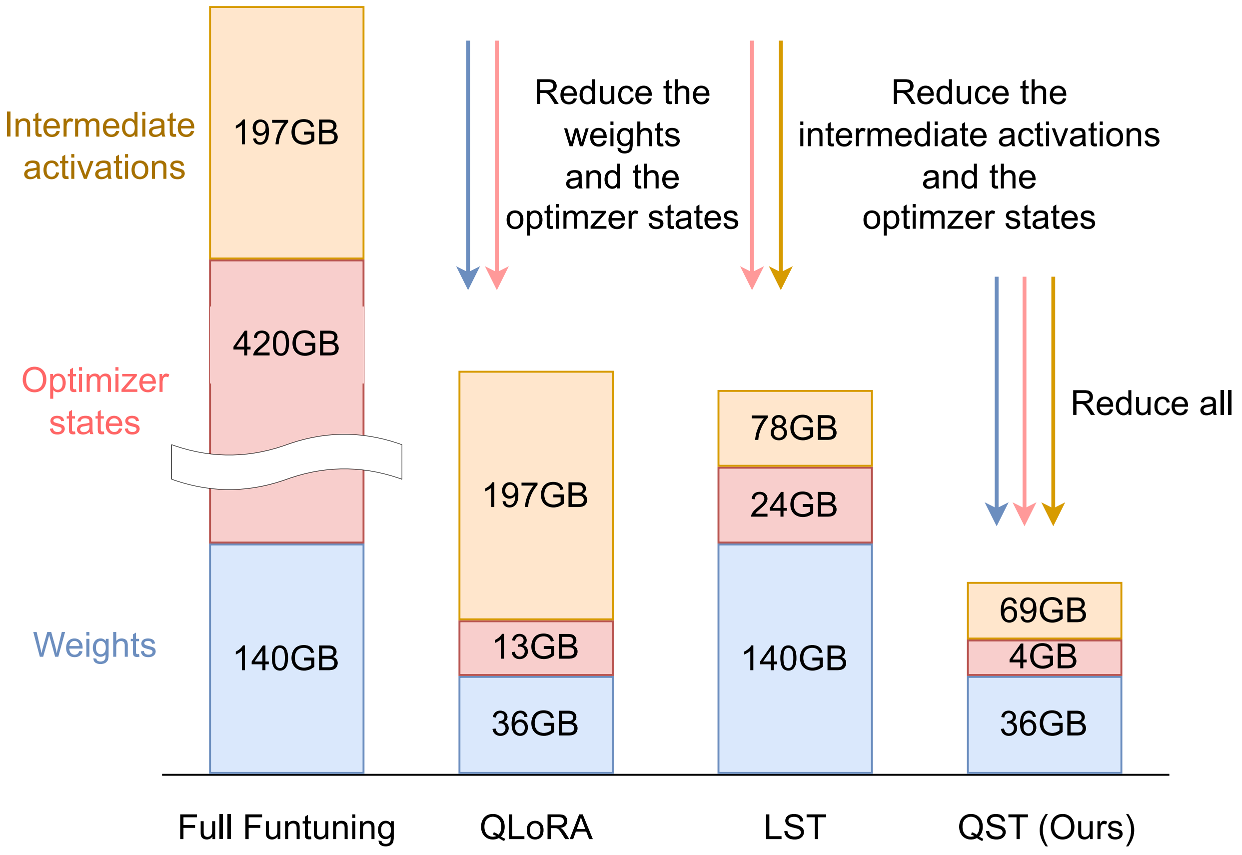
- Tuning adapters
  - LoRA: Low-Rank Adaptation
  - QLoRA: quantization + LoRA
  - **Side tuning**
  - Serving adapters

An Issue with LoRA (and Other Adapter Tuning Methods)

- Adapter networks reduce trainable weights and optimizer states
- **But require saving intermediate activations for back propagation**



LoRA for MLP layers

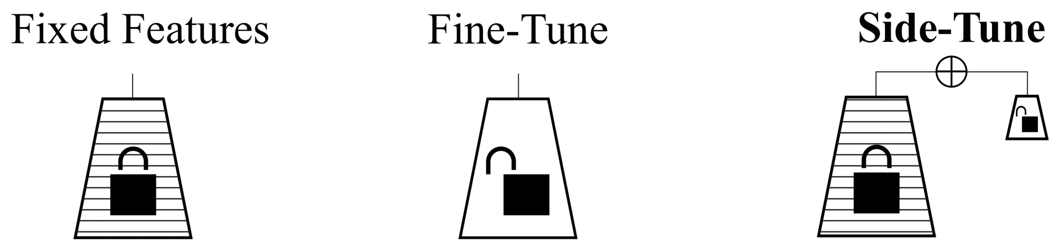


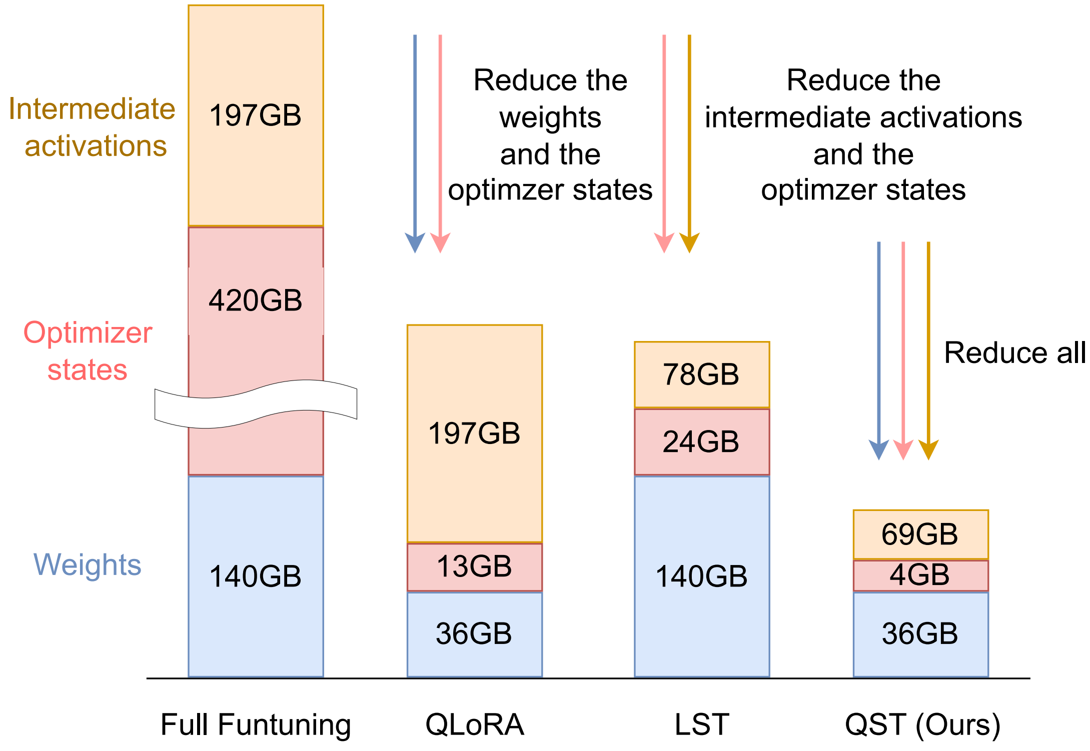
Side Tuning

Use side networks (a smaller version of the base model) as adapters

☺ **Avoid backpropagating the base model**

- No need to keep intermediate activations



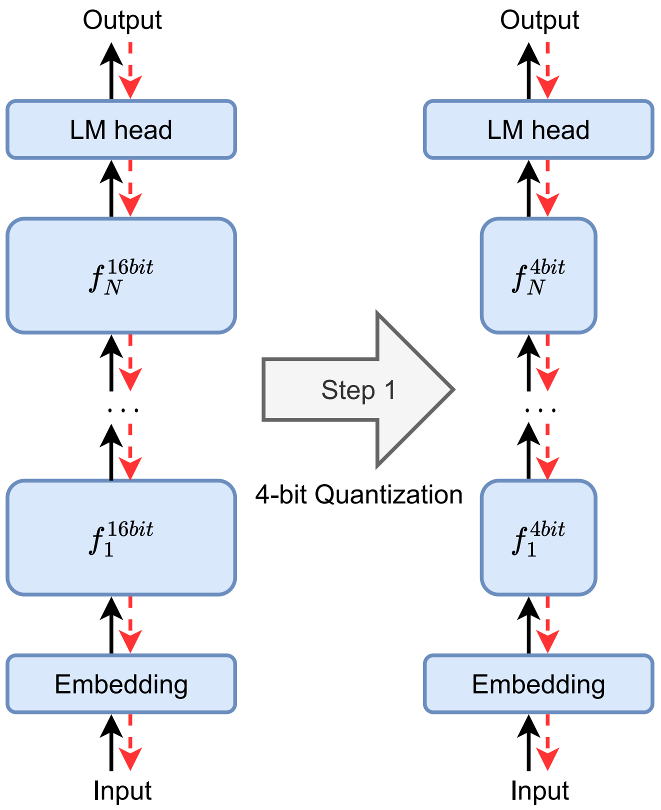


* Quantized Side Tuning: Fast and Memory-Efficient Tuning of Quantized Large Language Models.

Quantized Side Tuning in Two Steps

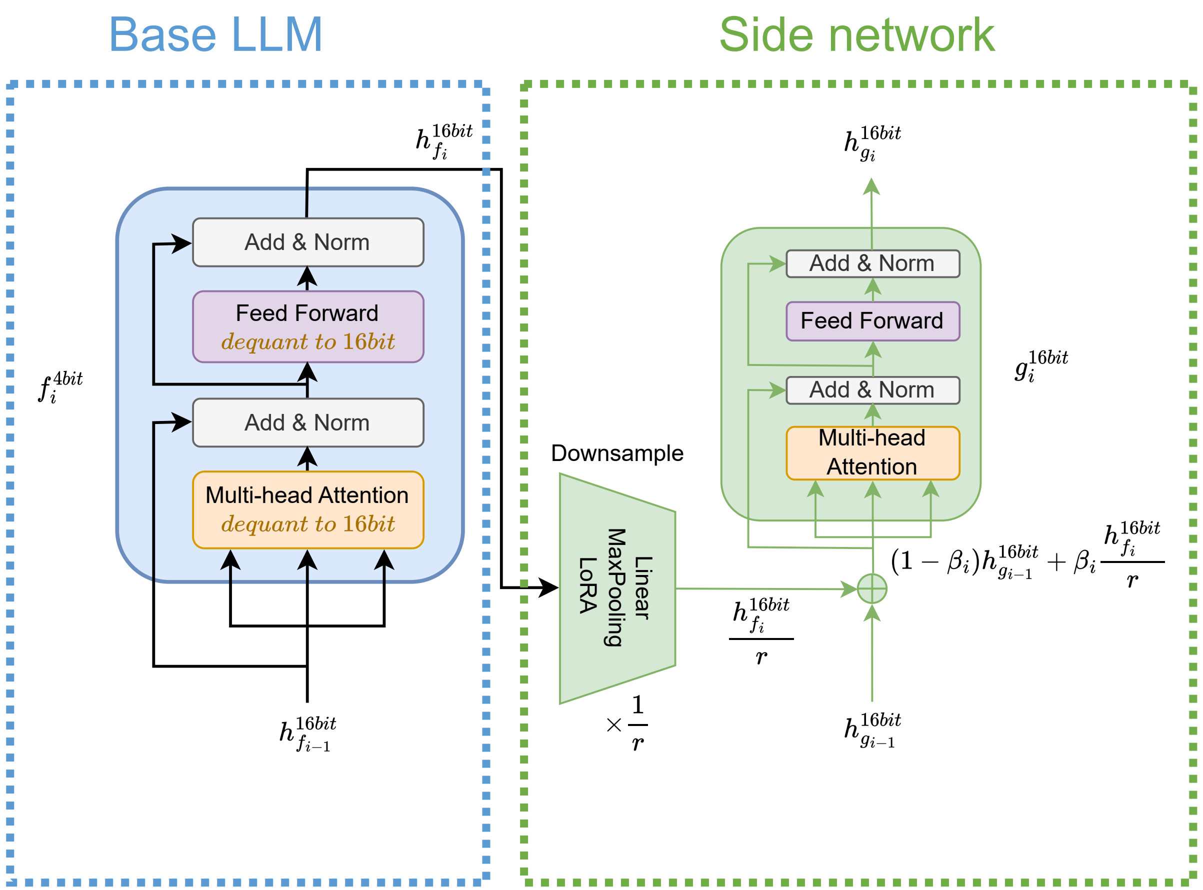
1. 4-bit block-wise, double quantization

$$
\begin{aligned}
X^{4bit} &= \text{round}\left(\frac{M_{4bit}}{\text{Absmax}(X^{16bit})} X^{16bit}\right) \quad (1)\\
&= \text{round}\left(c^{16bit} \cdot X^{16bit}\right), \quad (2)
\end{aligned}
$$



Quantized Side Tuning in Two Steps

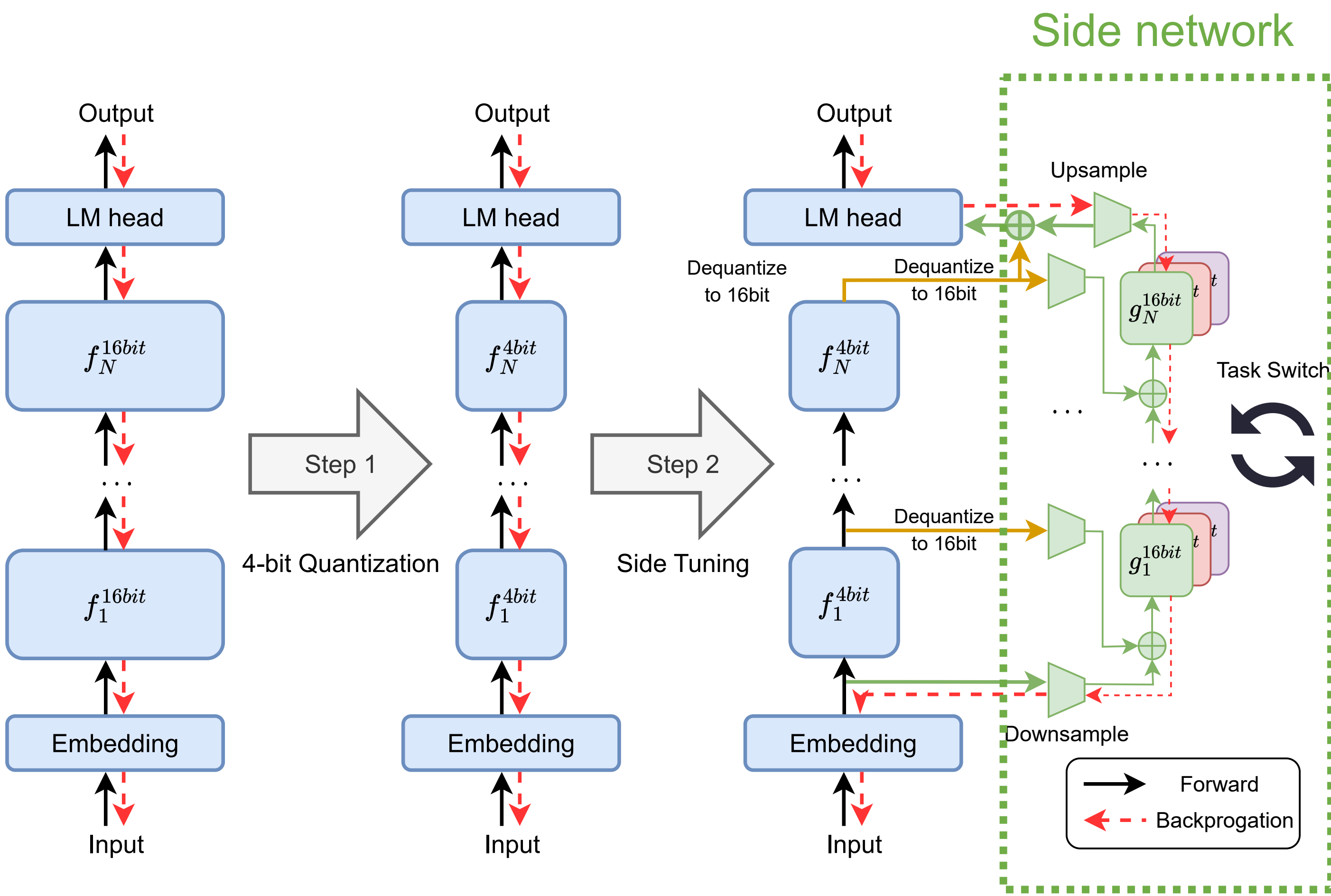
1. 4-bit block-wise, double quantization
2. Introduce a side network separated from the base LLM



Quantized Side Tuning in Two Steps

1. 4-bit block-wise, double quantization
2. Introduce a side network separated from the base LLM
   - Avoids backpropagation through the base LLM

Key insight: information flows from base to side but not the other way



Quantized Side Tuning

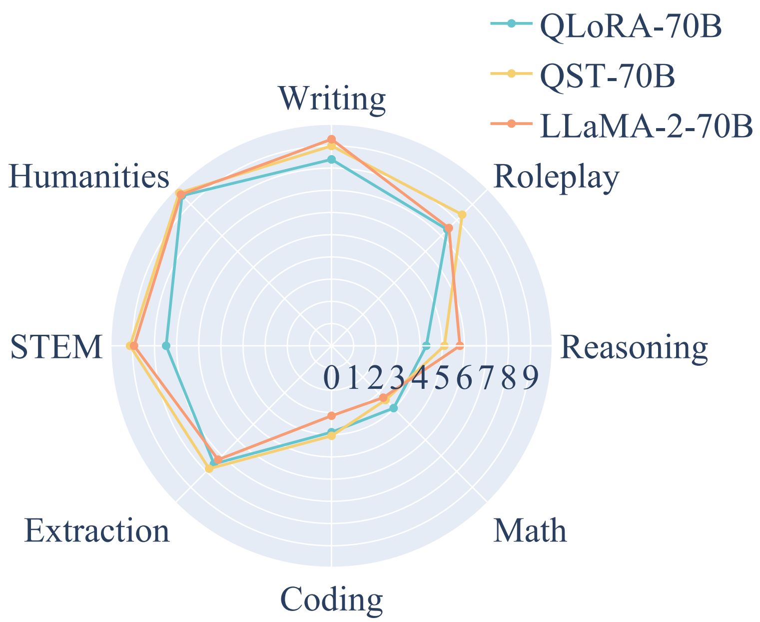
- Same accuracy as QLoRA while requiring less memory

Method	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B	OPT-66B	LLaMA-2-7B	LLaMA-2-13B	LLaMA-2-70B	Avg.
QLoRA	25.0/6.3	25.2/10.1	25.6/15.5	26.5/25.4	27.7/46.8	36.4/87.5	45.9/15.6	54.7/25.4	64.1/95.5	36.8/36.5
QST	24.3/3.2	25.5/4.8	26.2/7.2	26.8/12.6	27.3/25.7	36.0/52.3	45.1/7.3	56.8/12.6	63.9/56.0	36.9/20.2

Table 2: Experiment results (accuracy/memory) on MMLU 5-shot.

On-par chatbot performance as others

(Use GPT-4 as a judge)



Recap: Efficient LLM Finetuning Methods

- Tuning prompts
  - Prompt engineering
  - Prefix tuning

- Tuning adapters
  - LoRA: Low-Rank Adaptation
  - QLoRA: quantization + LoRA
  - Side tuning