Long Exposure: Accelerating Parameter-Efficient Fine-Tuning for LLMs under Shadowy Sparsity

Tuowei Wang*, Kun Li*, Zixu Hao, Donglin Bai,
Ju Ren, Yaoxue Zhang, Ting Cao, Mao Yang

Tsinghua University Microsoft Research

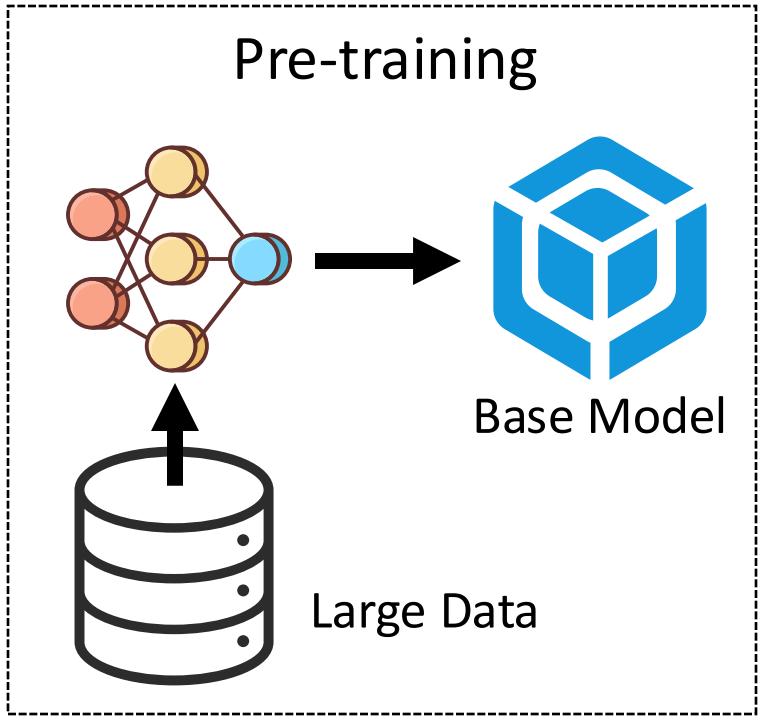
* Contributed Equally

Background: Fine-tuning is essential for LLMs

3-step LLM Paradigm:

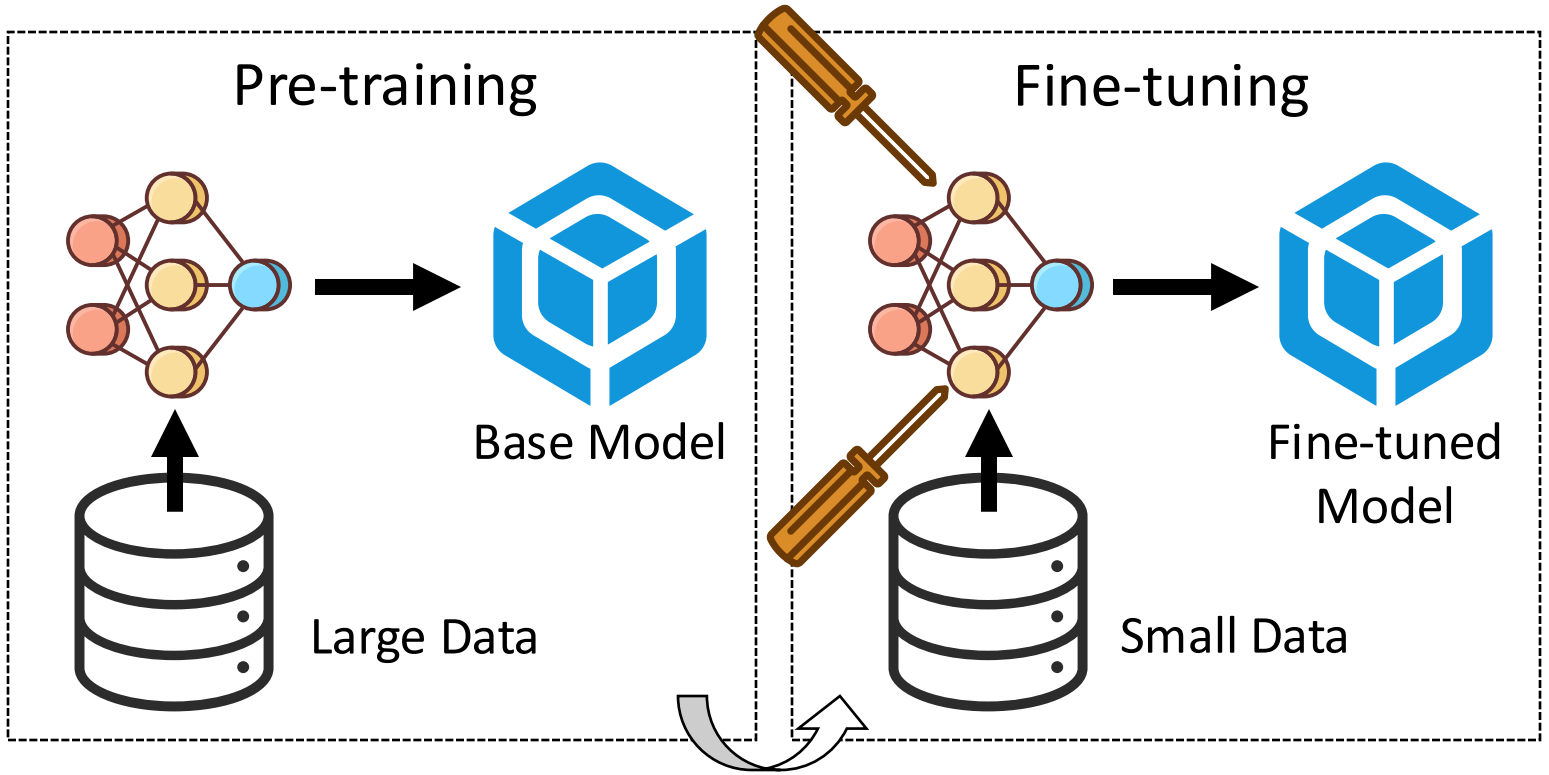
Background: Fine-tuning is essential for LLMs

3-step LLM Paradigm:



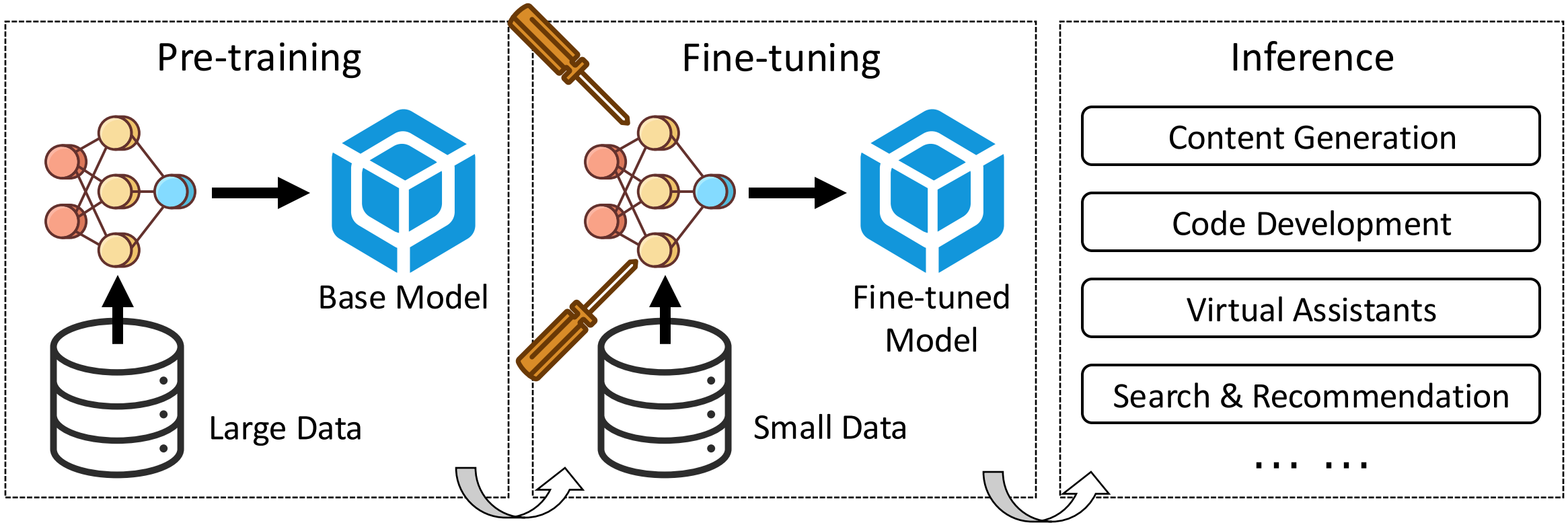
Background: Fine-tuning is essential for LLMs

3-step LLM Paradigm:



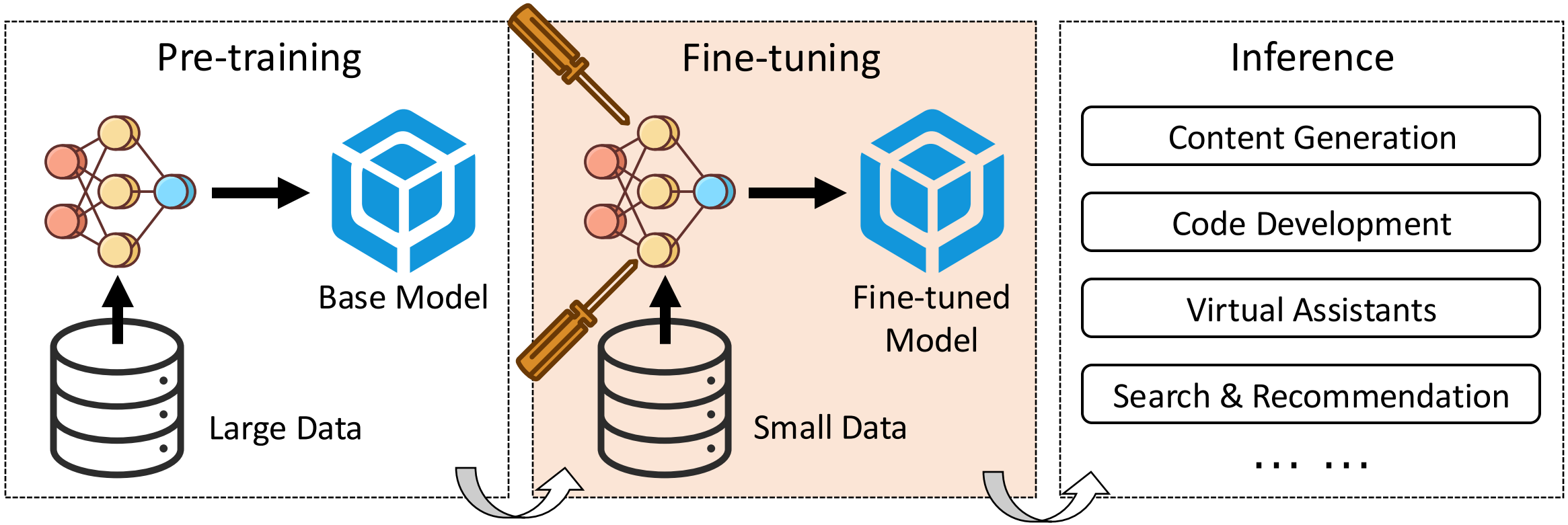
Background: Fine-tuning is essential for LLMs

3-step LLM Paradigm:



Background: Fine-tuning is essential for LLMs

3-step LLM Paradigm:



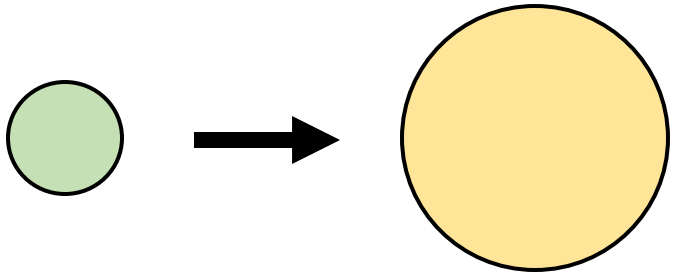
Boost LLM performance to specific tasks.

Background: Fine-tuning efficiency is essential for LLMs

Background: Fine-tuning efficiency is essential for LLMs

LLMs has experienced a trend of explosive increase in size.

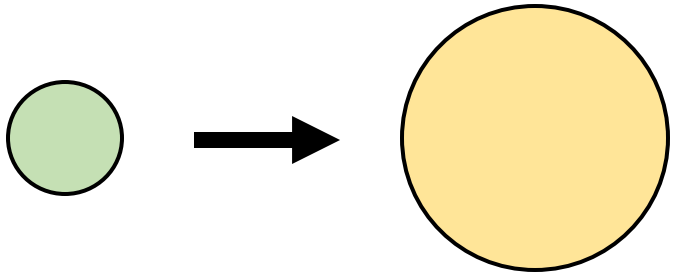
- BERT 0.34B, GPT-2 1.5B, Claude2 130B, GPT-3 175B.



Background: Fine-tuning efficiency is essential for LLMs

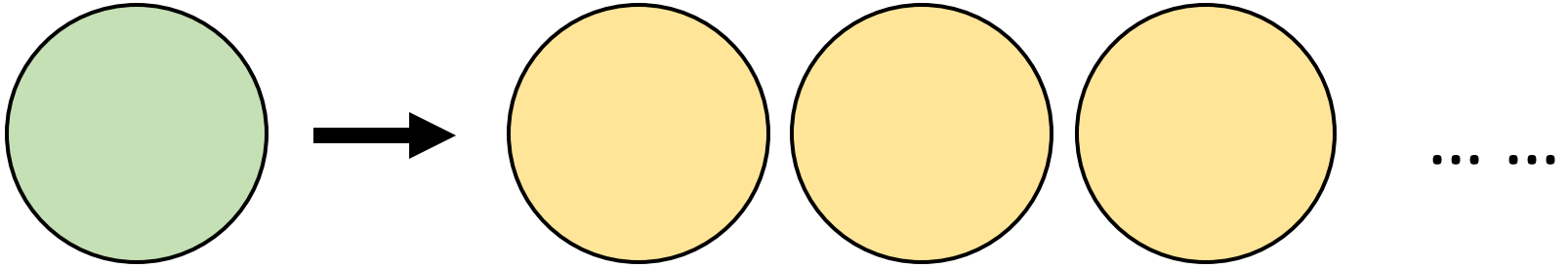
LLMs has experienced a trend of explosive increase in size.

- BERT 0.34B, GPT-2 1.5B, Claude2 130B, GPT-3 175B.



Pre-trained LLMs are periodic updated, typically every few months.

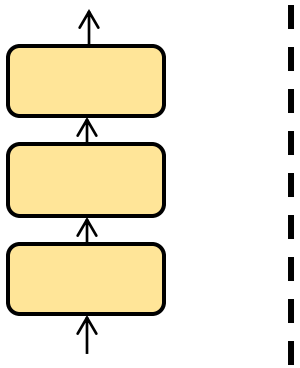
- Each update triggers hundreds of fine-tuning processes.



Background: Parameter-efficient Fine-tuning Techniques

Background: Parameter-efficient Fine-tuning Techniques

Full Fine-tuning involves updating all model parameters, which is costly.



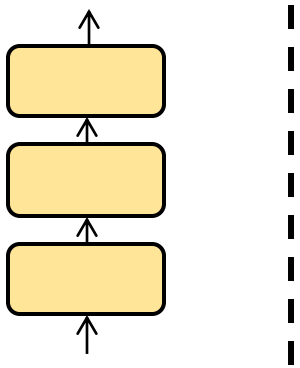
Full Fine-tuning

Background: Parameter-efficient Fine-tuning Techniques

Full Fine-tuning involves updating all model parameters, which is costly.

Parameter-efficient Fine-tuning (PEFT):

Selects or injects a minimal number of parameters for adaption.



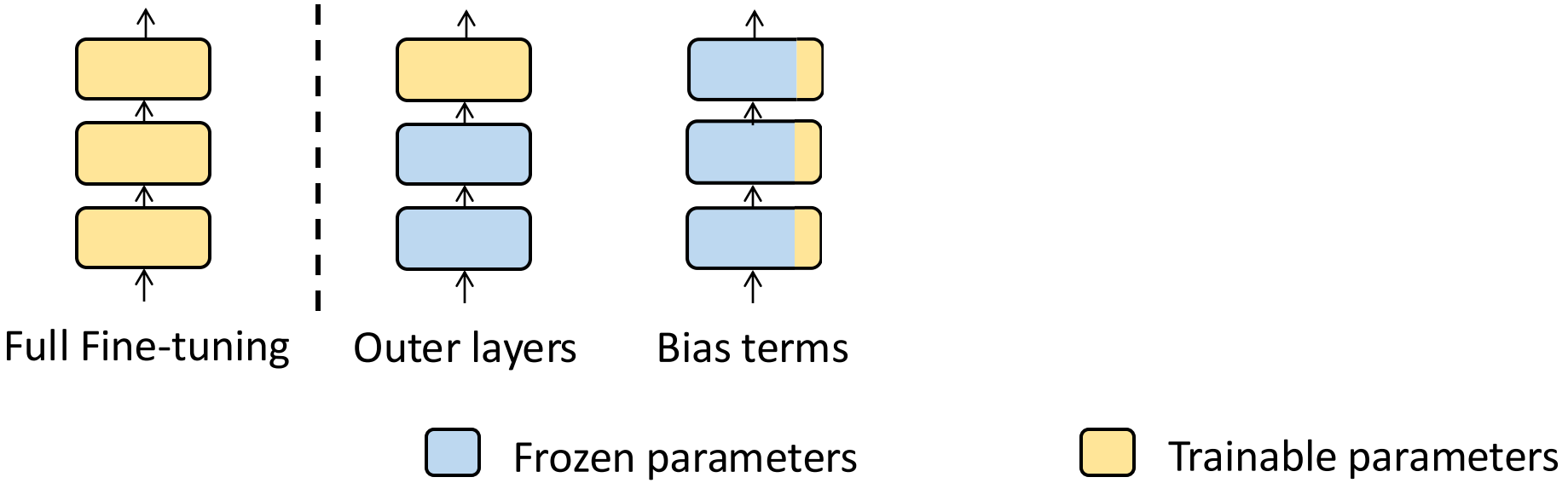
Full Fine-tuning

Background: Parameter-efficient Fine-tuning Techniques

Full Fine-tuning involves updating all model parameters, which is costly.

Parameter-efficient Fine-tuning (PEFT):

Selects or injects a minimal number of parameters for adaption.

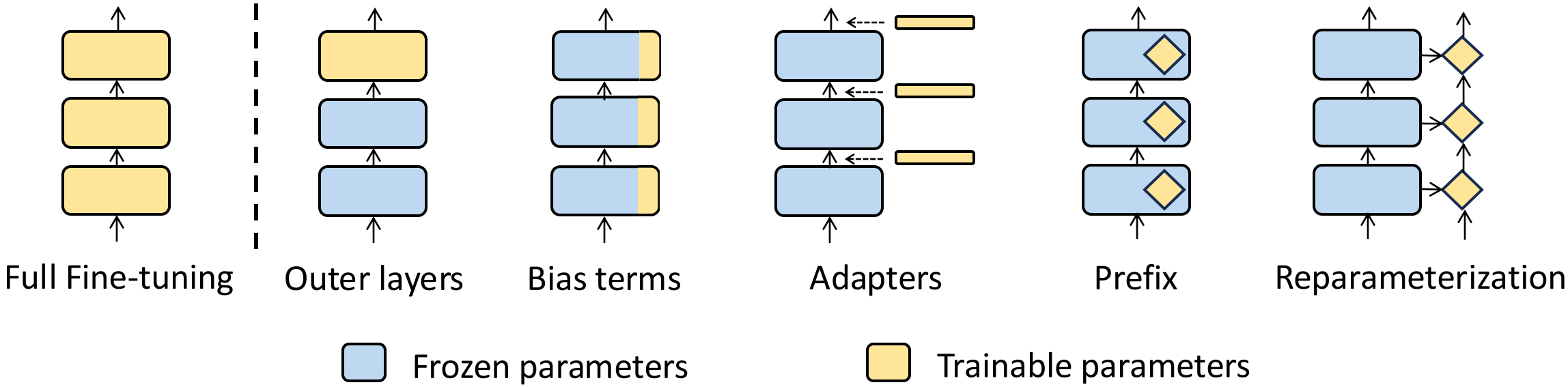


Background: Parameter-efficient Fine-tuning Techniques

Full Fine-tuning involves updating all model parameters, which is costly.

Parameter-efficient Fine-tuning (PEFT):

Selects or injects a minimal number of parameters for adaption.

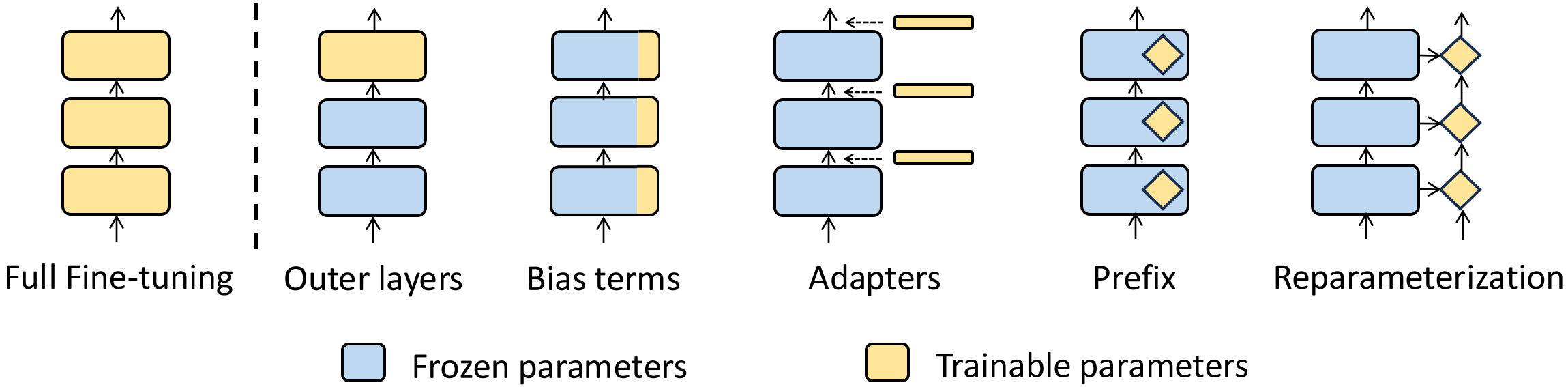


Background: Parameter-efficient Fine-tuning Techniques

Full Fine-tuning involves updating all model parameters, which is costly.

Parameter-efficient Fine-tuning (PEFT):

Selects or injects a minimal number of parameters for adaption.



Less than 1% model parameters are trainable.

Observation: Parameter-saving isn’t proportionally time-saving

PEFT techniques fall short of achieving an expected decrease in wall-clock time.

Observation: Parameter-saving isn't proportionally time-saving

PEFT techniques fall short of achieving an expected decrease in wall-clock time.

OPT-1.3B fine-tuning time breakdown. (ms/batch)

Phase	Forward	Backward	Optim. Step	Total
Full Param.	112.8(27.7%)	223.7(54.9%)	70.6(17.3%)	407.2
LoRA	135.3(40.4%)	196.3(58.7%)	2.0(0.6%)	334.6
Adapter	123.6(42.2%)	168.4(57.5%)	0.7(0.3%)	292.9
Bitfit	117.6(40.5%)	172.4(59.4%)	0.2(0.07%)	290.3
P-Tuning	137.5(40.1%)	193.9(56.6%)	11.1(3.2%)	342.6

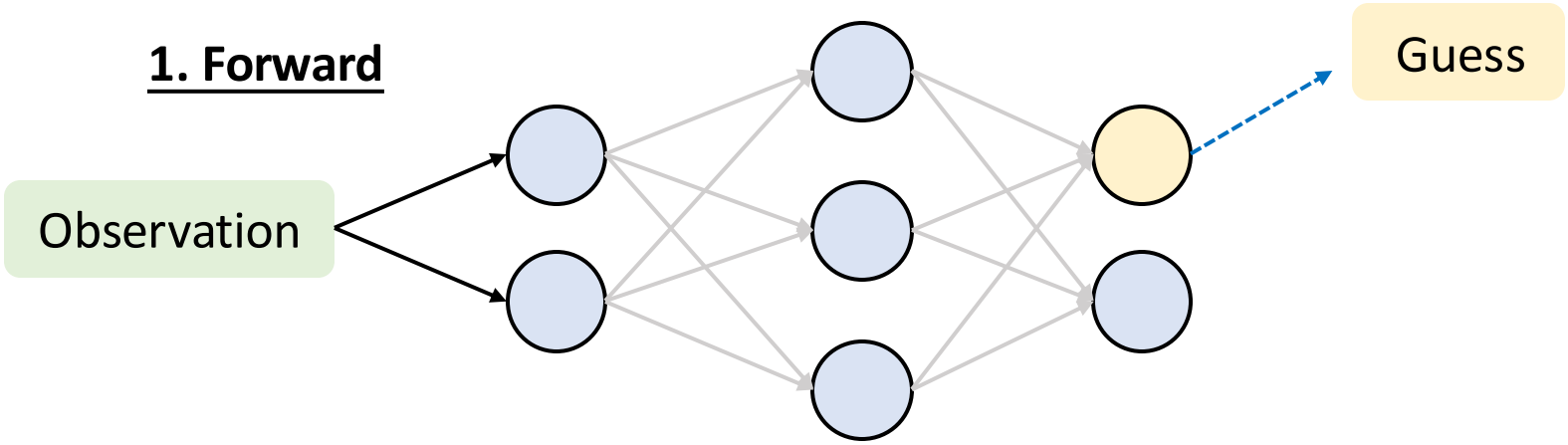
More than 70% wall-clock time is remained.

Insight #1: Forward and backward are the performance bottlenecks

Insight #1: Forward and backward are the performance bottlenecks

Fine-tuning, as well as training, consists of 3 phrases:

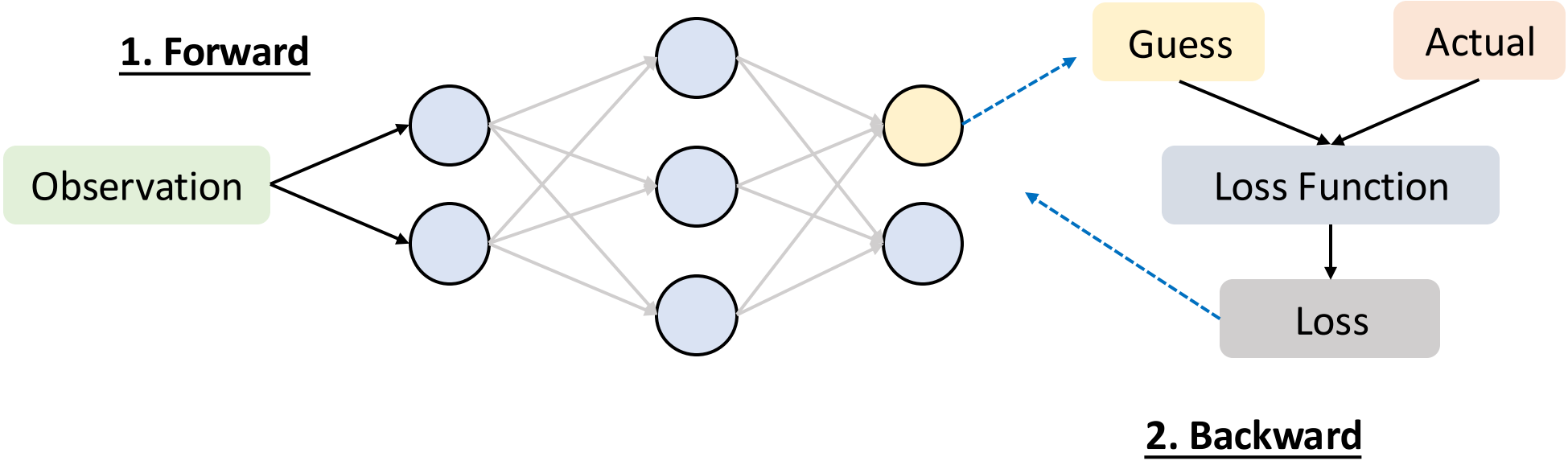
- **Forward:** calculates the loss.



Insight #1: Forward and backward are the performance bottlenecks

Fine-tuning, as well as training, consists of 3 phrases:

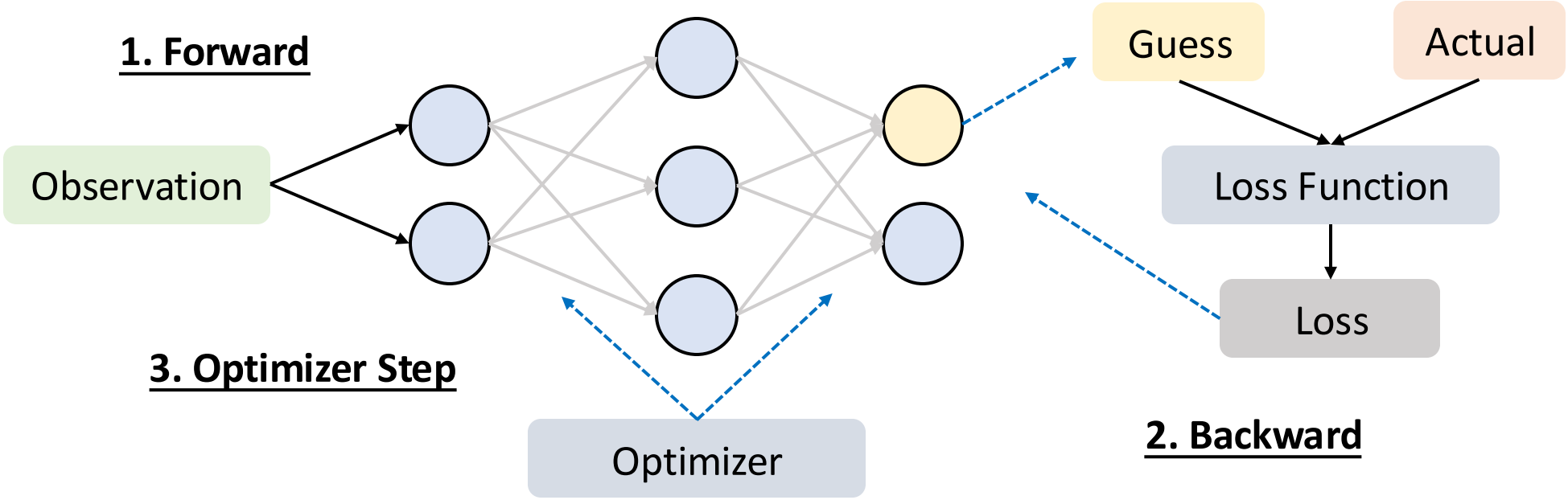
- **Forward:** calculates the loss.
- **Backward:** calculates the gradient.



Insight #1: Forward and backward are the performance bottlenecks

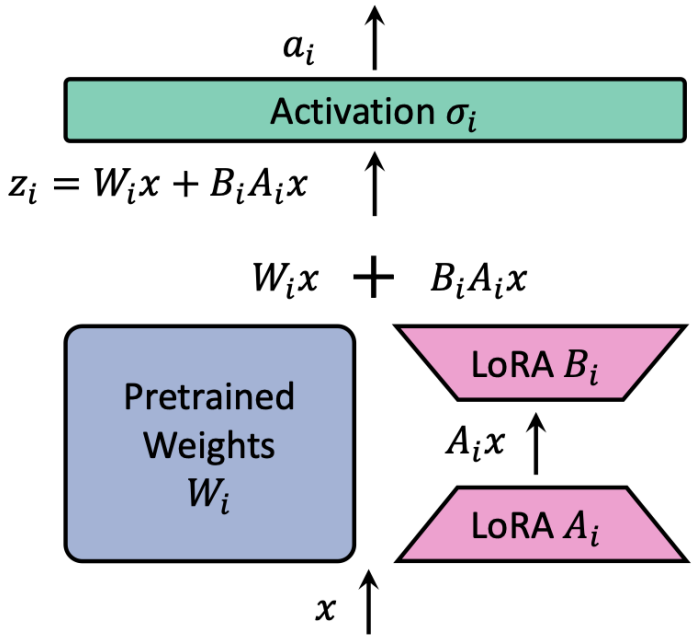
Fine-tuning, as well as training, consists of 3 phrases:

- **Forward:** calculates the loss.
- **Backward:** calculates the gradient.
- **Optimizer step:** updates the trainable parameters.



Insight #1: Forward and backward are the performance bottlenecks

Applying LoRA to fine-tune an MLP block

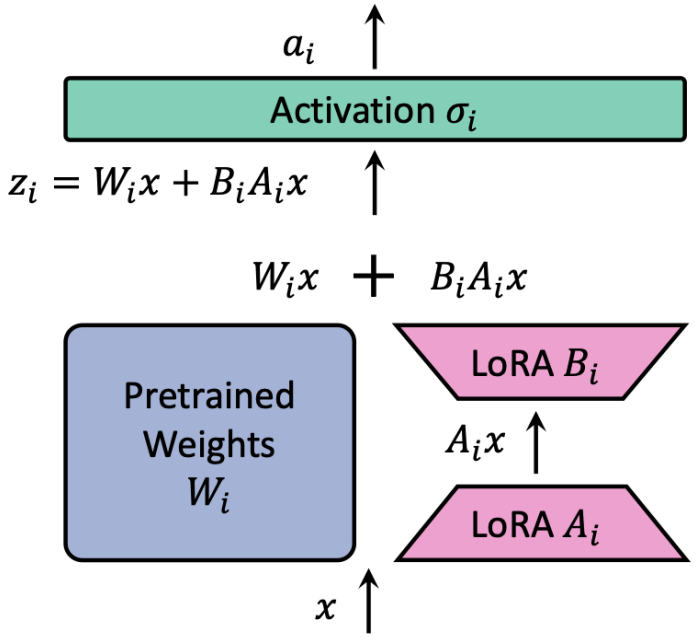


Insight #1: Forward and backward are the performance bottlenecks

Forward

- Vanilla: $z_i = \sigma_i(W_i x)$
- LoRA: $z_i = \sigma_i(W_i x + B_i A_i x)$

Applying LoRA to fine-tune an MLP block



Insight #1: Forward and backward are the performance bottlenecks

Forward

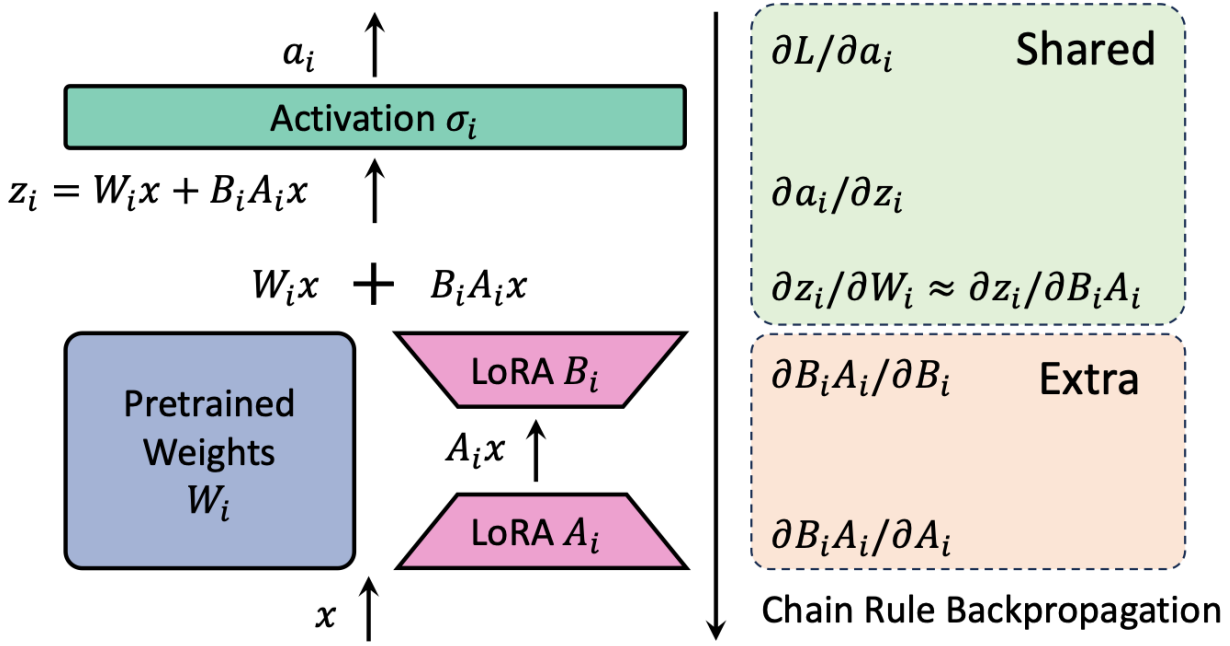
- Vanilla: $z_i = \sigma_i(W_i x)$
- LoRA: $z_i = \sigma_i(W_i x + B_i A_i x)$

Backward

- Vanilla: $\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial W_i}$
- LoRA: $\frac{\partial L}{\partial A_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial B_i A_i}\frac{\partial B_i A_i}{\partial A_i}$

$$\frac{\partial L}{\partial B_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial B_i A_i}\frac{\partial B_i A_i}{\partial B_i}$$

Applying LoRA to fine-tune an MLP block



Insight: Forward and backward are the performance bottlenecks

Forward

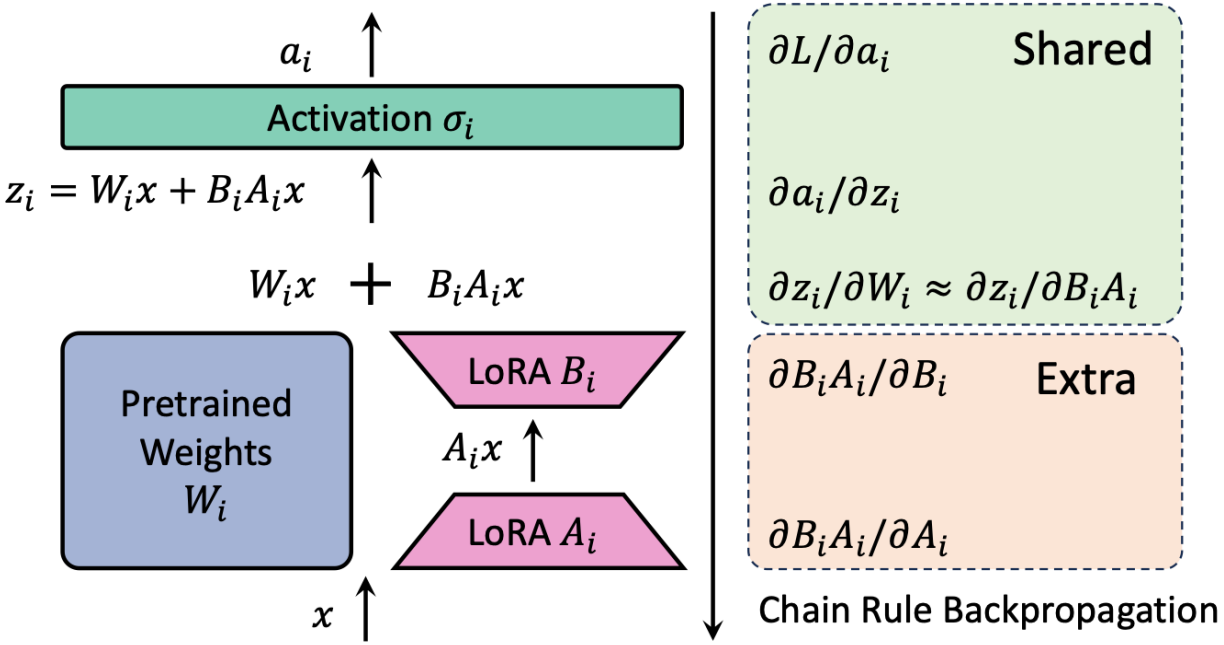
- Vanilla: $z_i = \sigma_i(W_i x)$
- LoRA: $z_i = \sigma_i(W_i x + B_i A_i x)$

Backward

- Vanilla: $\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial W_i}$
- LoRA: $\frac{\partial L}{\partial A_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial B_i A_i}\frac{\partial B_i A_i}{\partial A_i}$

$$\frac{\partial L}{\partial B_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial B_i A_i}\frac{\partial B_i A_i}{\partial B_i}$$

Applying LoRA to fine-tune an MLP block



Optimizer step

- Reduced due to fewer trainable parameters.
- Savings vary with the choice of optimizer.

Insight: Forward and backward are the performance bottlenecks

Forward

- Vanilla: $z_i = \sigma_i(W_i x)$
- LoRA: $z_i = \sigma_i(W_i x + B_i A_i x)$

Backward

- Vanilla: $\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial W_i}$
- LoRA: $\frac{\partial L}{\partial A_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial B_i A_i}\frac{\partial B_i A_i}{\partial A_i}$

$$\frac{\partial L}{\partial B_i} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial B_i A_i}\frac{\partial B_i A_i}{\partial B_i}$$

Optimizer step

- Reduced due to fewer trainable parameters.
- Savings vary with the choice of optimizer.

OPT-1.3B fine-tuning time breakdown. (ms/batch)

Phase	Forward	Backward	Optim. Step	Total
Full Param.	112.8(27.7%)	223.7(54.9%)	70.6(17.3%)	407.2
LoRA	135.3(40.4%)	196.3(58.7%)	2.0(0.6%)	334.6
Adapter	123.6(42.2%)	168.4(57.5%)	0.7(0.3%)	292.9
Bitfit	117.6(40.5%)	172.4(59.4%)	0.2(0.07%)	290.3
P-Tuning	137.5(40.1%)	193.9(56.6%)	11.1(3.2%)	342.6

Forward and Backward are the bottlenecks.

Insight: PEFT and inference share high similarity

In both PEFT and inference, most model parameters remain frozen.

Insight: PEFT and inference share high similarity

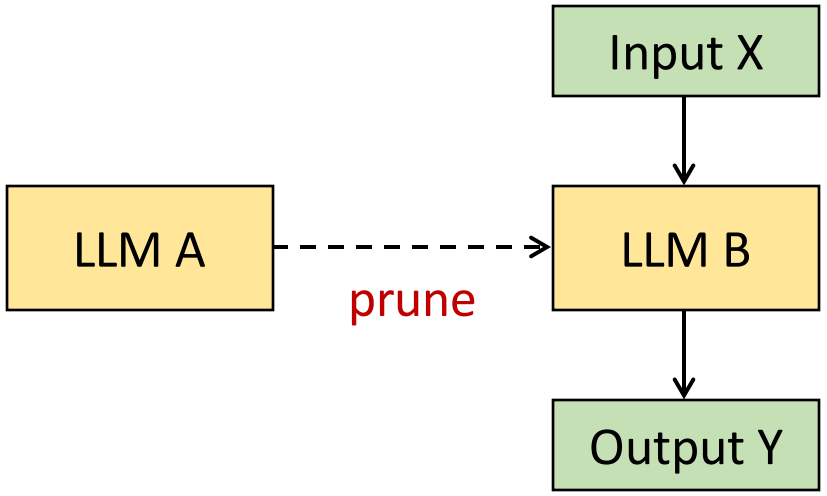
In both PEFT and inference, most model parameters remain frozen.

Sparsity has been widely used for model inference acceleration.

Insight: PEFT and inference share high similarity

In both PEFT and inference, most model parameters remain frozen.

Sparsity has been widely used for model inference acceleration.

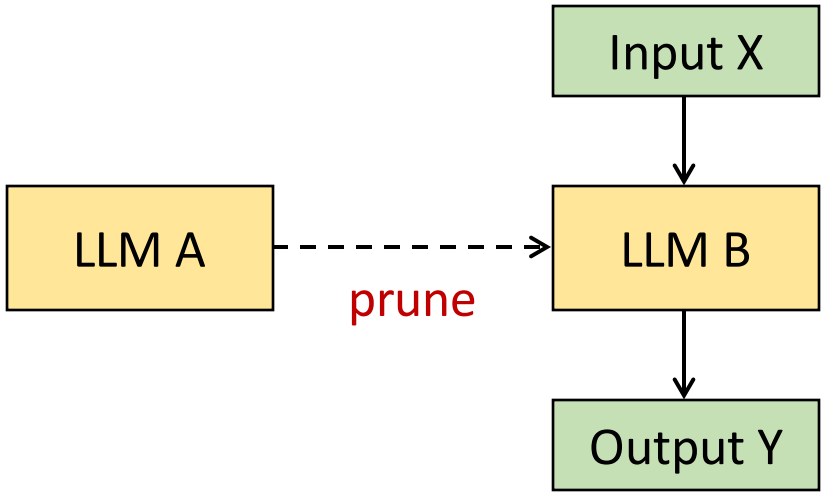


Static Sparsity

Insight: PEFT and inference share high similarity

In both PEFT and inference, most model parameters remain frozen.

Sparsity has been widely used for model inference acceleration.



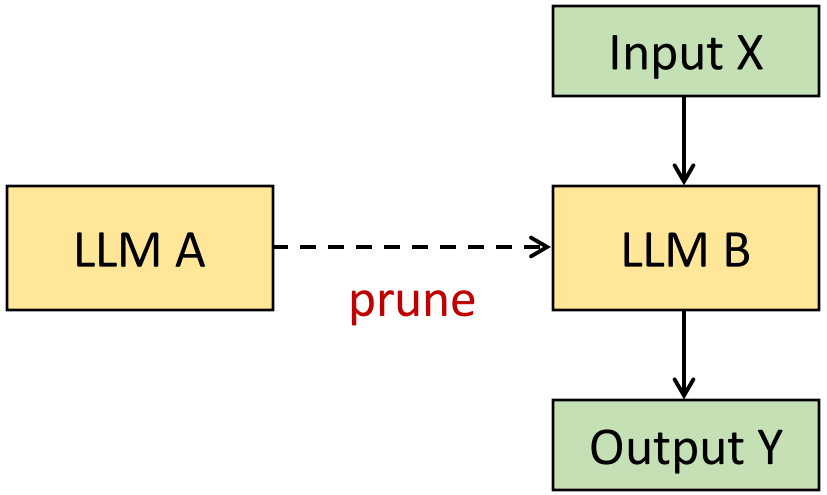
Static Sparsity

☹ Costly Retraining.

Insight: PEFT and inference share high similarity

In both PEFT and inference, most model parameters remain frozen.

Sparsity has been widely used for model inference acceleration.



Static Sparsity

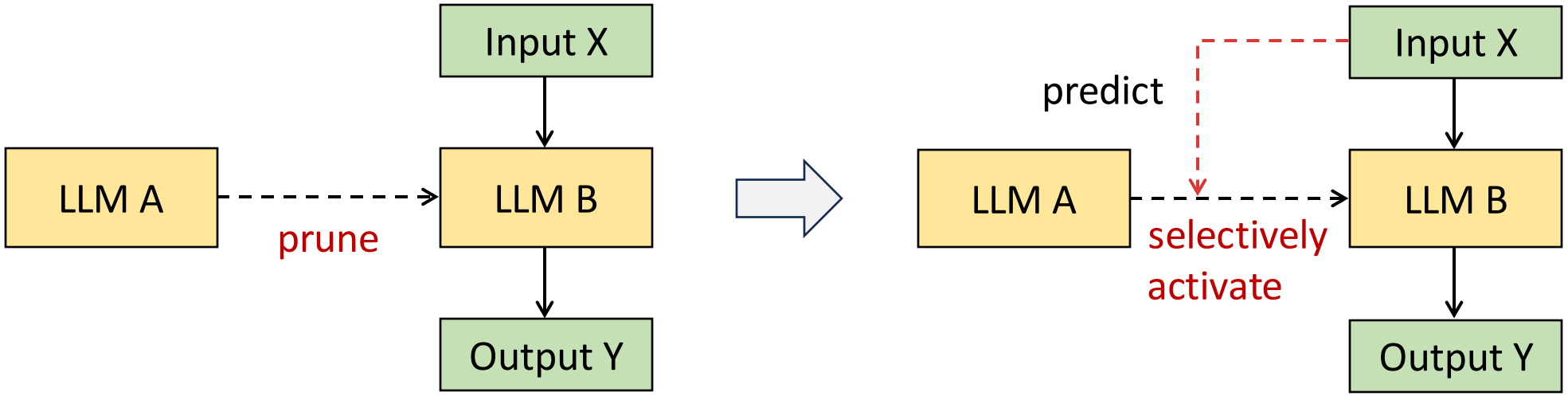
☹ Costly Retraining.

☹ Generalization Loss.

Insight: PEFT and inference share high similarity

In both PEFT and inference, most model parameters remain frozen.

Sparsity has been widely used for model inference acceleration.



Static Sparsity

Activation Sparsity

Insight: PEFT and inference share high similarity

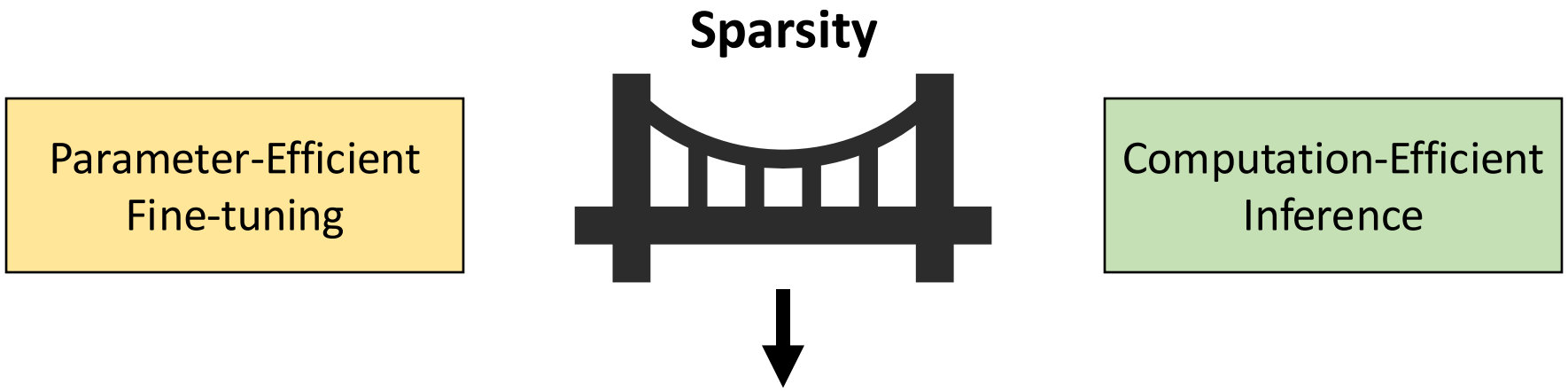
Why not build a bridge between PEFT and inference by capturing LLM sparsity?

- Backbone models are frozen.
- Activation sparsity also exists.
- Fine-tuning can be even more robust.

Insight: PEFT and inference share high similarity

Why not build a bridge between PEFT and inference by capturing LLM sparsity?

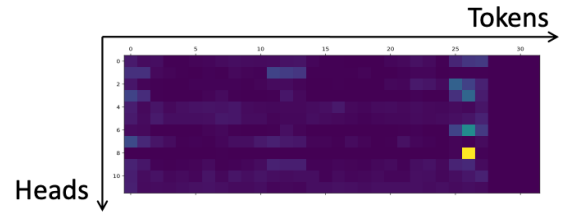
- Backbone models are frozen.
- Activation sparsity also exists.
- Fine-tuning can be even more robust.



Challenge: Sparsity is superimposed of token batches during fine-tuning

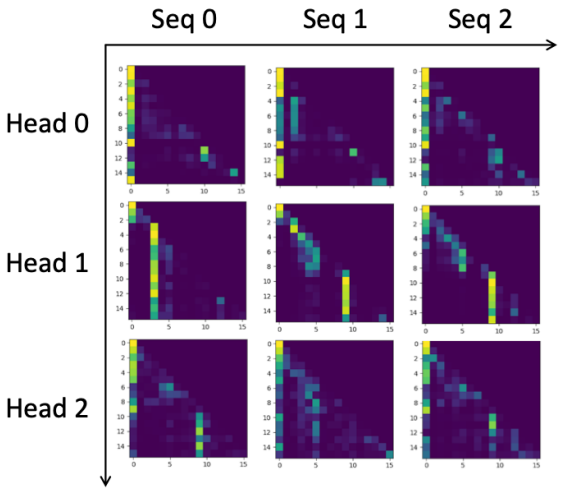
In inference, the model input is a single token.

In fine-tuning, the model input is a sequence of tokens.

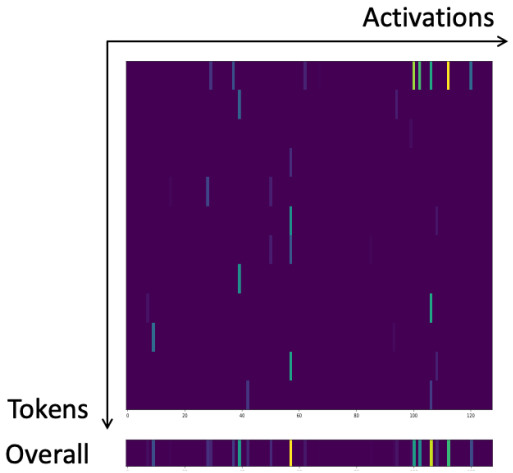


(a) Multi-head Attention Inference

(c) MLP Block Inference



(b) Multi-head Attention Fine-tuning



(d) MLP Block Fine-tuning

The overall sparsity pattern is the overlapped sparsity of different tokens – **Shadowy Sparsity**.

Challenge: Sparsity is superimposed of token batches during fine-tuning

In inference, the model input is a single token.

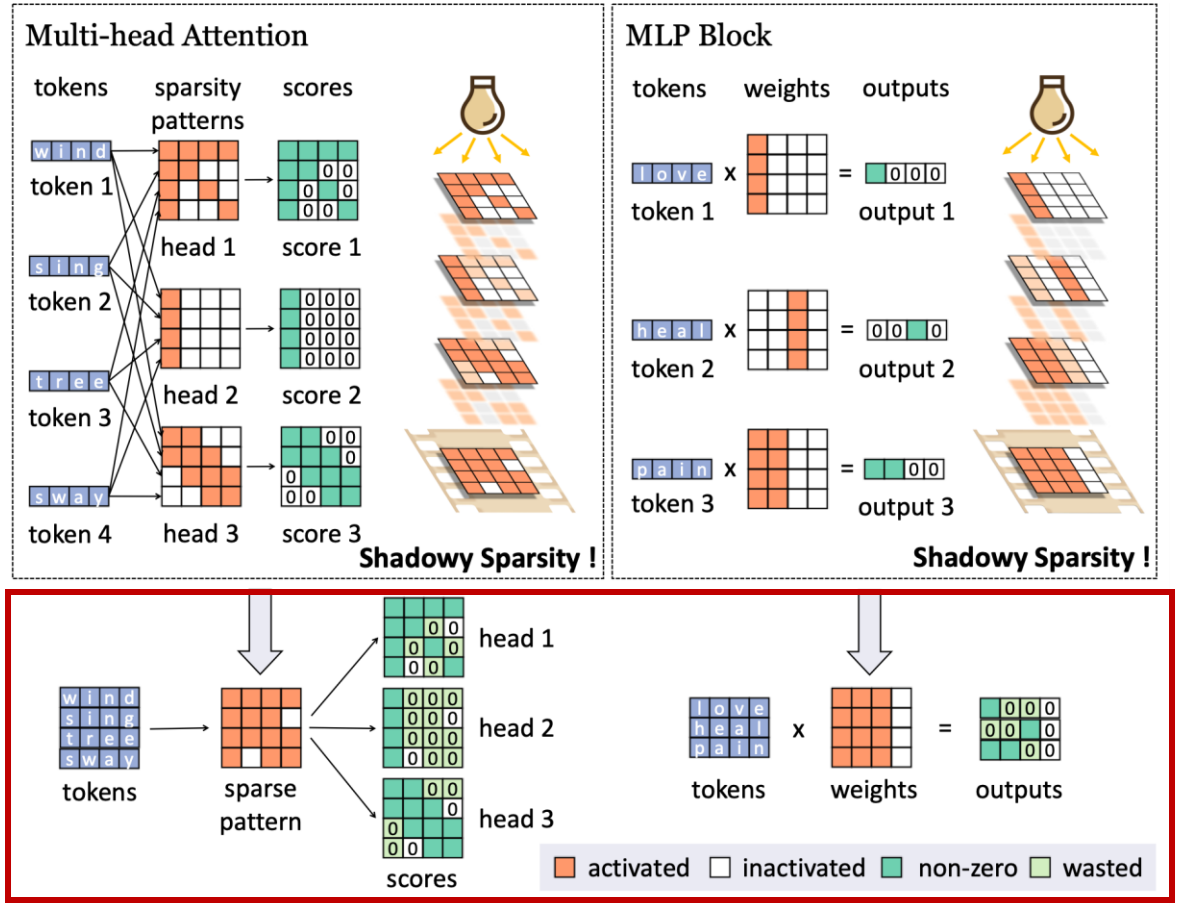
In fine-tuning, the model input is a sequence of tokens.

1. How can we capture sparse patterns effectively?
2. How can we predict sparse patterns seamlessly?
3. How can we perform on sparse patterns efficiently?

Challenge #1: How can we capture sparse patterns effectively?

The dense units for one token coincide with the sparse units for another.

Although each token may exhibit high sparsity, the overall sparsity is limited.



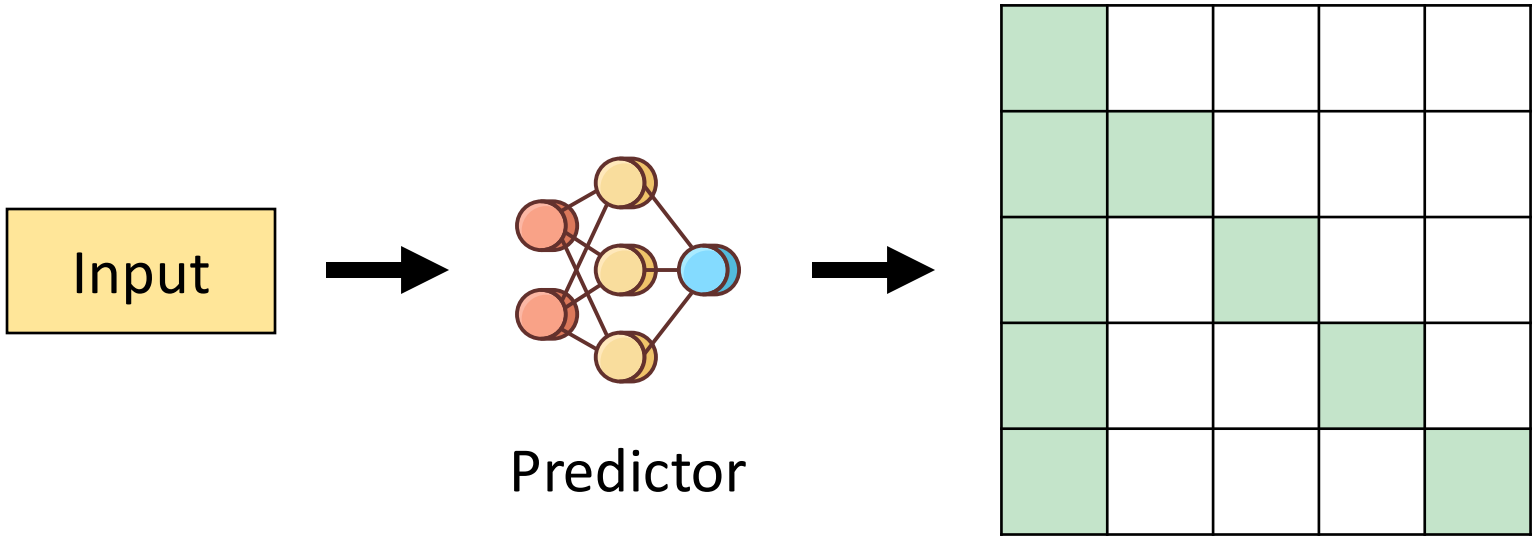
Computation Waste

Challenge #2: How can we predict sparse patterns seamlessly?

Obtain gain from activation sparsity requires prediction.

Neural-networks-based prediction has shown promise in LLM inference.

This prediction could become costly in fine-tuning due to large input size.

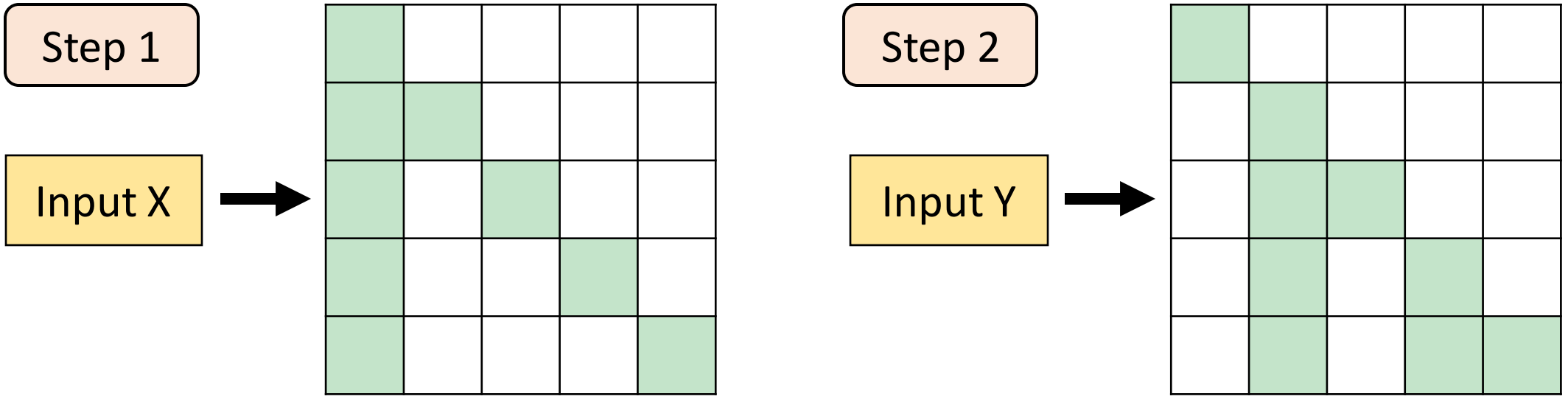


Challenge #3: How can we perform on sparsity patterns efficiently?

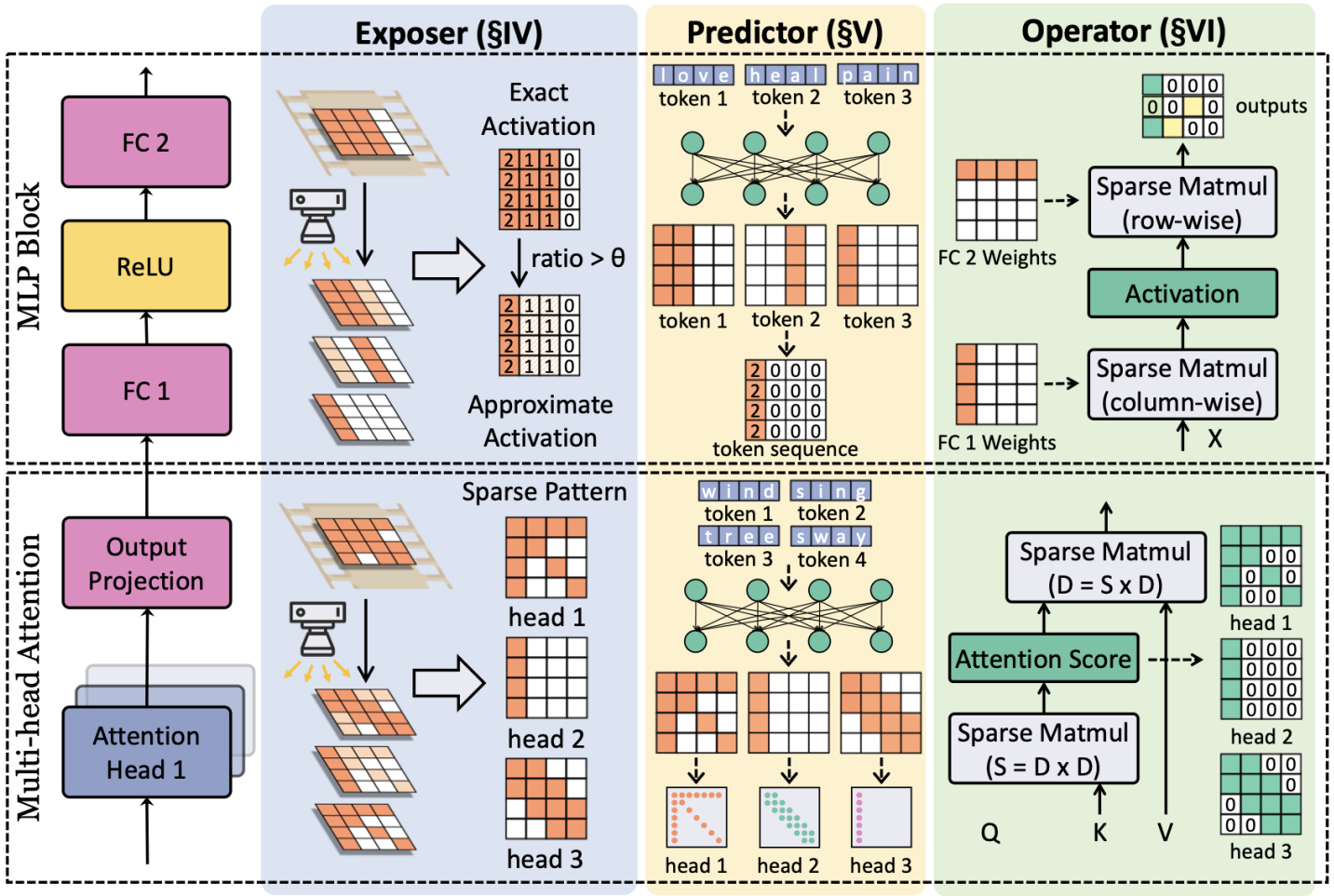
Sparse operations are challenging due to scattered memory accesses.

Besides, the computation patterns vary with different inputs at runtime.

Existing tools are limited by static sparsity or introduce conversion overhead.



Overview: Long Exposure



I: Shadowy-sparsity Exposer

- capture sparse patterns

II: Sequence-oriented Predictor

- predict sparse patterns

III: Dynamic-aware Operator

- exploit sparse patterns

Component #1: Shadow-sparsity Exposer

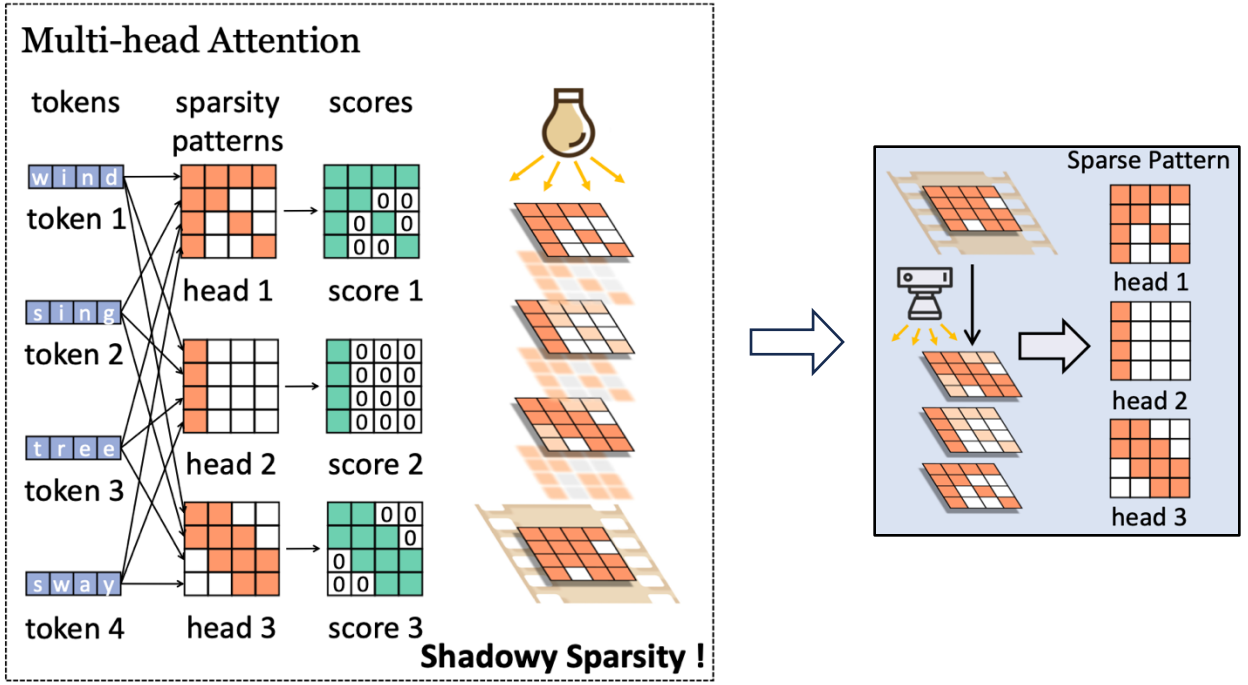
Core idea: Explore the intricate details of individual sparse patterns.

Component #1: Shadow-sparsity Exposer

Core idea: Explore the intricate details of individual sparse patterns.

MHA: Head-specific sparse mask

- Emphasize the strong interactions among different tokens.
- Consider the unique data distribution of each head.

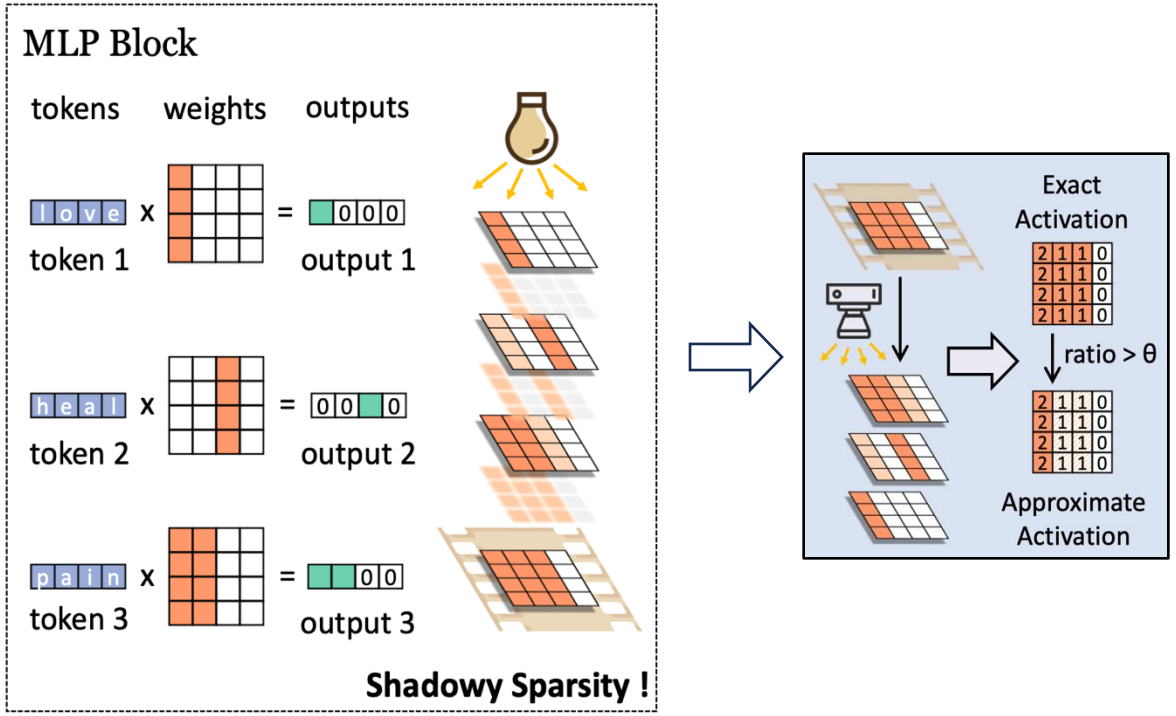


Component #1: Shadow-sparsity Exposer

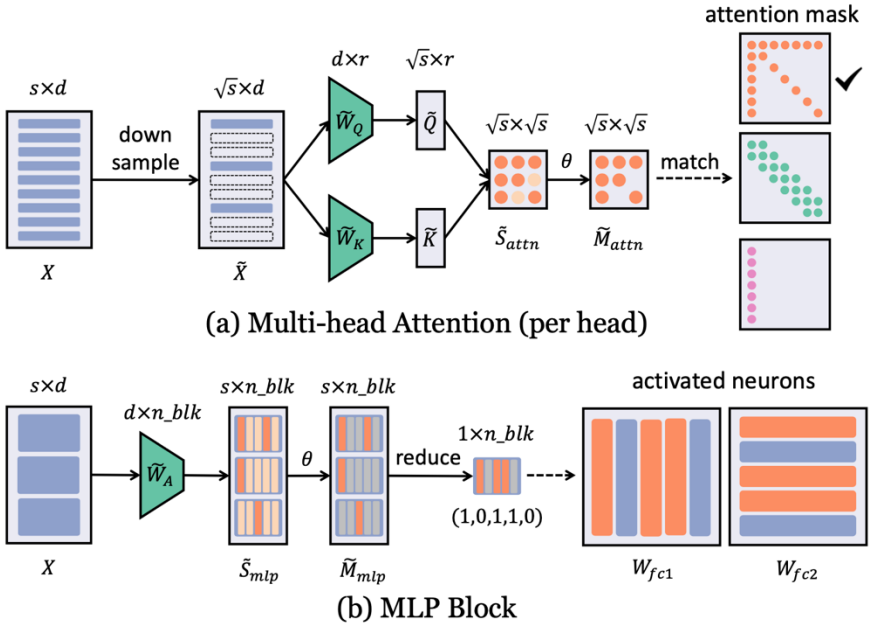
Core idea: Explore the intricate details of individual sparse patterns.

MLP: Threshold-based filter

- Emphasize both the activated counts and activated values of each neurons.
- Employ a block-wise manner to align with the hardware characteristics.



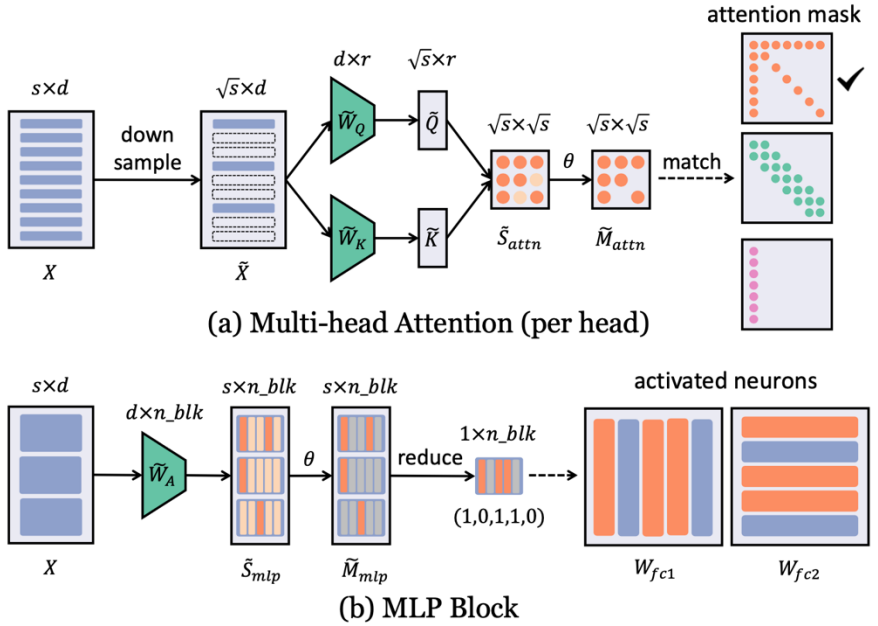
Component #2: Sequence-oriented Predictor



Component #2: Sequence-oriented Predictor

Core idea I: Effectiveness

- Prioritize *recall* over *precision*.
- Add noise for avoiding overfitting.



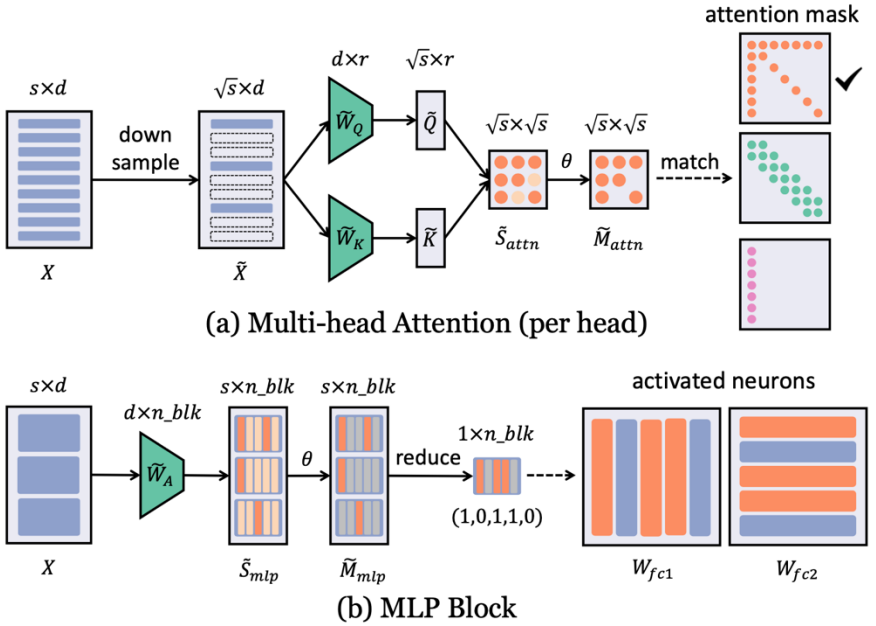
Component #2: Sequence-oriented Predictor

Core idea I: Effectiveness

- Prioritize *recall* over *precision*.
- Add noise for avoiding overfitting.

Core idea II: Efficiency

- Make predictions at the token level and then reduce the results.
- Simplify the objective: from values to pattern.



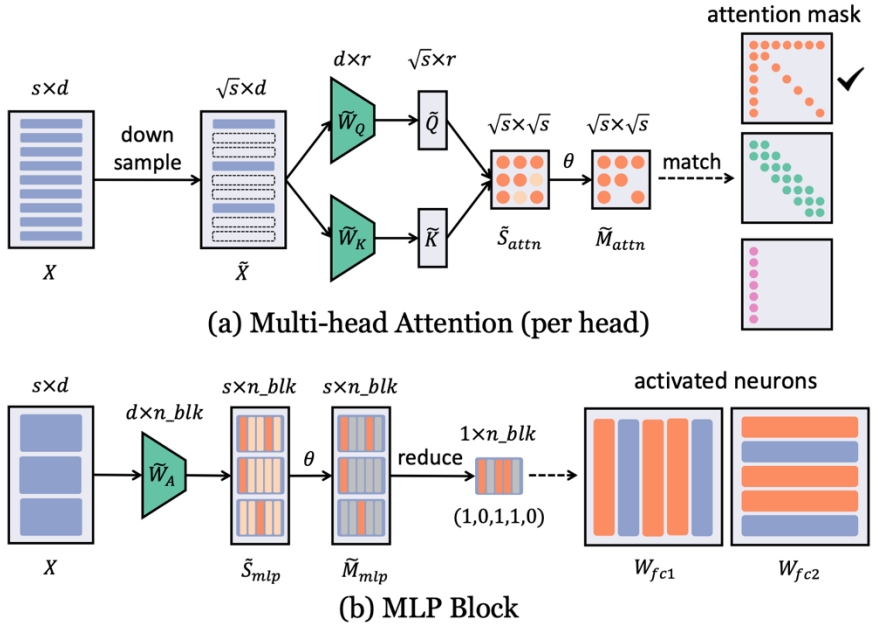
Component #2: Sequence-oriented Predictor

Core idea I: Effectiveness

- Prioritize *recall* over *precision*.
- Add noise for avoiding overfitting.

Core idea II: Efficiency

- Make predictions at the token level and then reduce the results.
- Simplify the objective: from values to pattern.



Savings: $O(s^2)$ to $O(s)$

Overhead: $\text{Cost}_{attn} = \text{Cost}_Q + \text{Cost}_K + \text{Cost}_{QK}$

$= \sqrt{s}dr + \sqrt{s}dr + sr$

Savings: depends on sparsity ratio, around 0.8-0.9

Overhead: $\text{Cost}_{mlp} = \text{Cost}_A + \text{Cost}_{AND} = sdr + s$

Component #3: Dynamic-aware Operator

Core idea: Simple and efficient; focused on the task at hand.

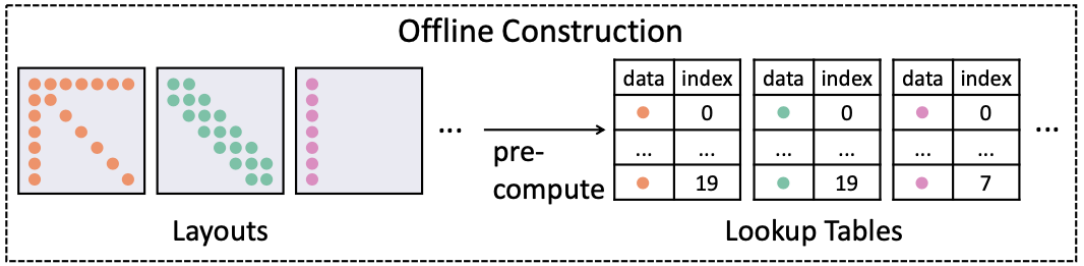
Component #3: Dynamic-aware Operator

Core idea: Simple and efficient; focused on the task at hand.

MHA: Two-stage algorithm

Offline Pool Construction

- Construct an atomic sparse pattern pool.
- Pre-calculate their layout lookup tables.



Component #3: Dynamic-aware Operator

Core idea: Simple and efficient; focused on the task at hand.

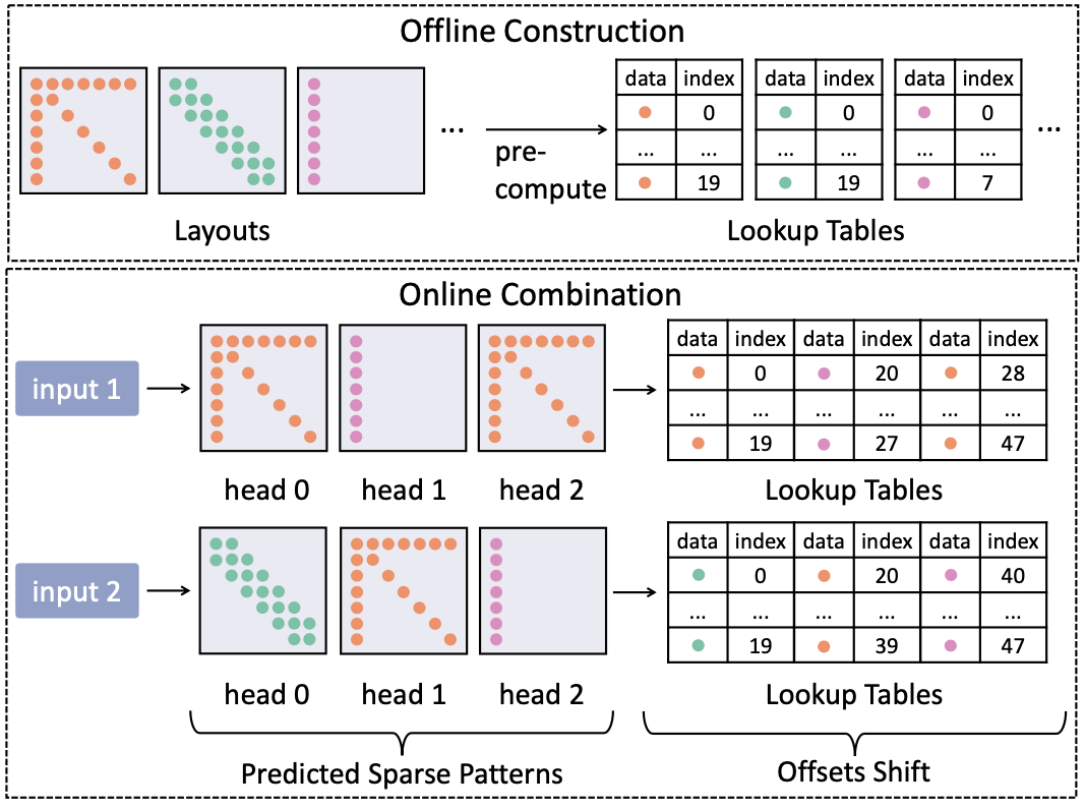
MHA: Two-stage algorithm

Offline Pool Construction

- Construct an atomic sparse pattern pool.
- Pre-calculate their layout lookup tables.

Online Pattern Combination

- Combine atomic sparse patterns at runtime.
- Flexible yet incurs minimal overhead.



Component #3: Dynamic-aware Operator

Core idea: Simple and efficient; focused on the task at hand.

MLP: Neuron (i.e., a column or row in the weight matrix) is the basic unit.

Component #3: Dynamic-aware Operator

Core idea: Simple and efficient; focused on the task at hand.

MLP: Neuron (i.e., a column or row in the weight matrix) is the basic unit.

Tiling Technique

- The block manner aligns with the tiling technique in GEMM.
- Only the activated blocks are loaded and processed.

Component #3: Dynamic-aware Operator

Core idea: Simple and efficient; focused on the task at hand.

MLP: Neuron (i.e., a column or row in the weight matrix) is the basic unit.

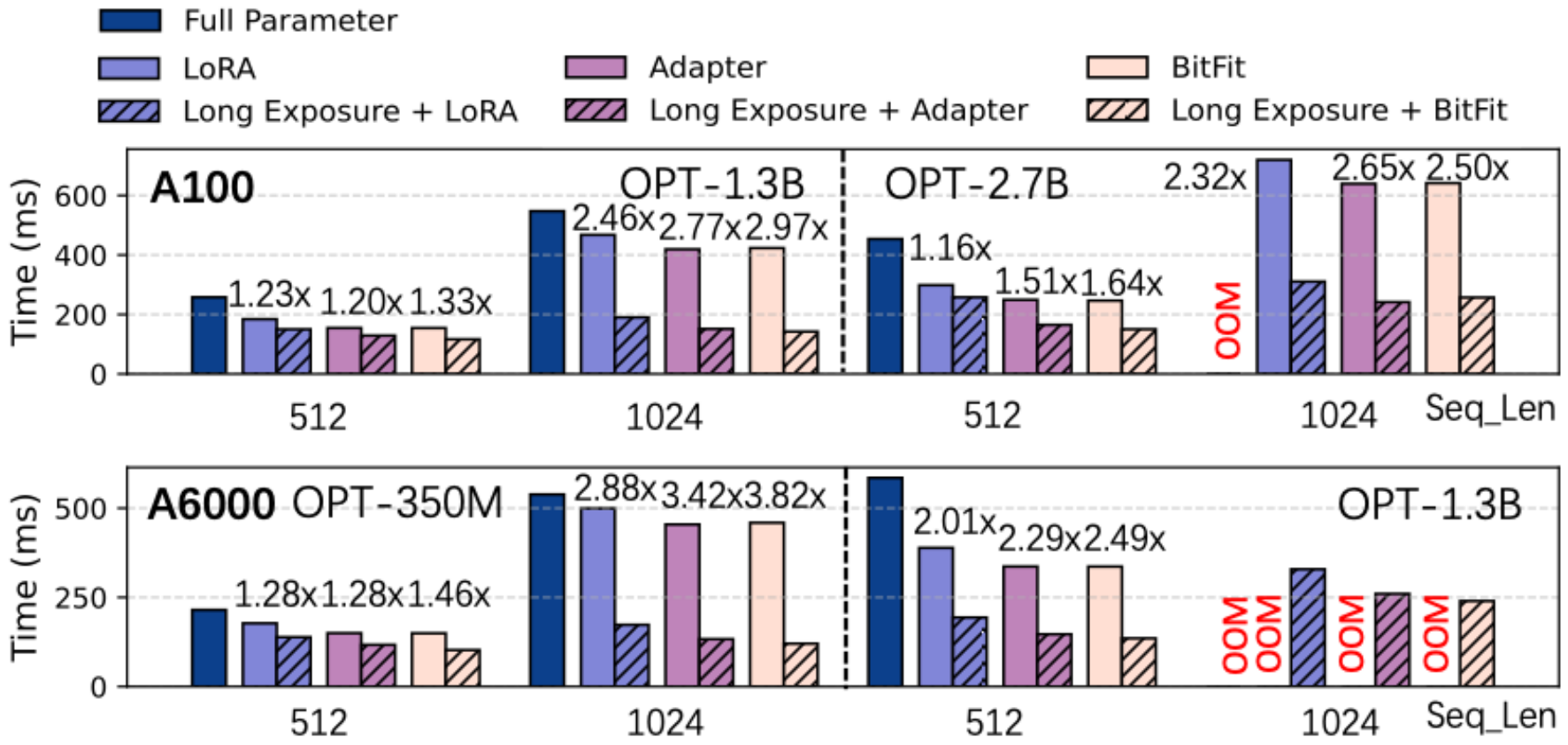
Tiling Technique

- The block manner aligns with the tiling technique in GEMM.
- Only the activated blocks are loaded and processed.

Memory Coalescing

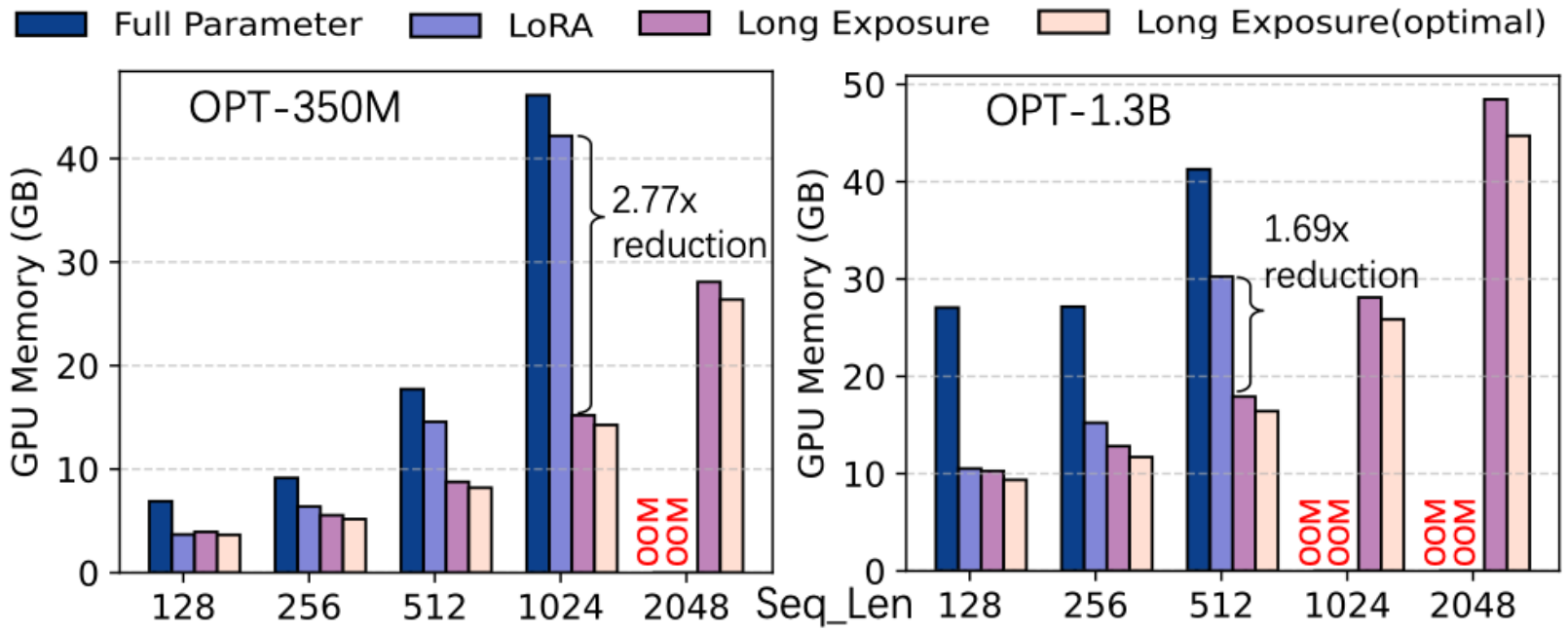
- Store the weights based on their access patterns.
- Column-major for the up projection and row-major for the down projection.

End-to-end Fine-tuning Performance (Nvidia A100 and A6000)



Long Exposure outperforms existing methods by on average 1.20-2.69x across various sequence lengths and model sizes.

End-to-end Fine-tuning Memory Footprint (Nvidia A100)



Long Exposure alters the memory complexity from $O(s^2)$ to $O(s)$, achieving 1.69x-2.77x memory reduction compared to PyTorch Implementation.

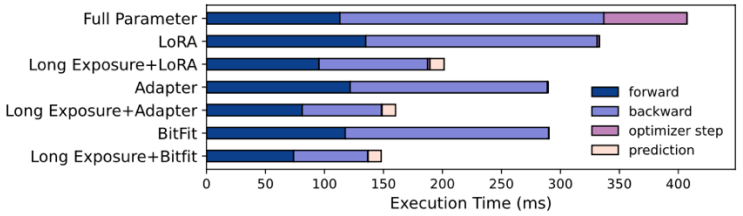
Model Accuracy on Downstream Tasks

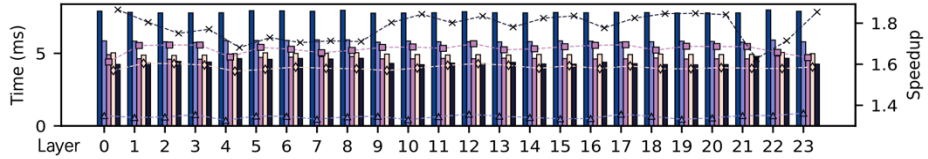
		350M-w/o	350M-w	1.3B-w/o	1.3B-w	2.7B-w/o	2.7B-w
PIQA	Acc.	65.13%	64.80%	72.25%	72.09%	74.70%	73.45%
	Stderr	1.11%	1.12%	1.05%	1.06%	1.02%	1.02%
Winog.	Acc.	53.04%	53.12%	58.88%	58.80%	62.27%	62.19%
	Stderr	1.40%	1.40%	1.38%	1.38%	1.37%	1.36%
RTE	Acc.	54.51%	55.60%	54.15%	54.51%	52.71%	53.79%
	Stderr	2.99%	3.01%	3.01%	3.01%	3.00%	2.04%
COPA	Acc.	69.00%	70.00%	81.00%	81.00%	78.00%	76.00%
	Stderr	4.61%	4.51%	4.23%	4.02%	4.29%	4.09%
Hella.	Acc.	32.26%	32.40%	42.08%	42.11%	46.76%	43.95%
	Stderr	0.47%	0.47%	0.499%	0.49%	0.50%	0.50%

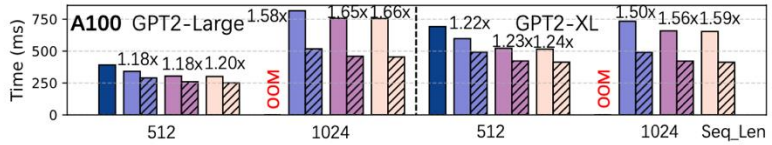
Long Exposure incurs a minimal loss in downstream task accuracy across all 5 tasks and 3 model sizes.

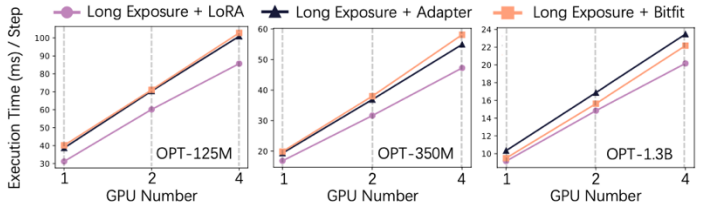
More Evaluation in Paper

1. Detailed performance breakdown.
2. Ablation studies on three key components.
3. Sensitivity analysis on model structures.
4. Scalability analysis with increasing machine sizes.



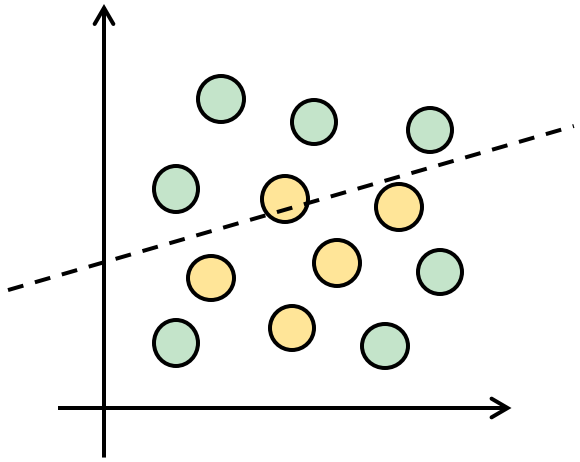






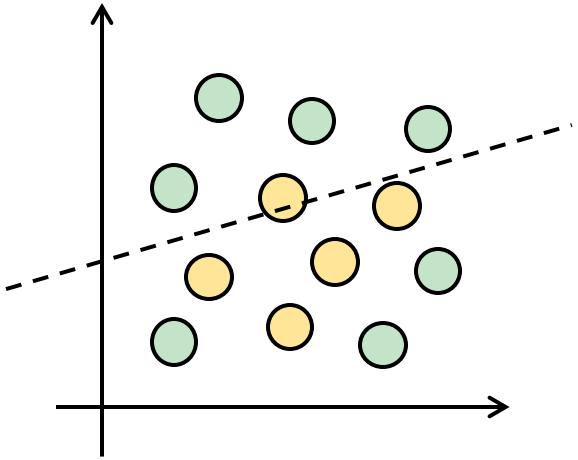
Next Step

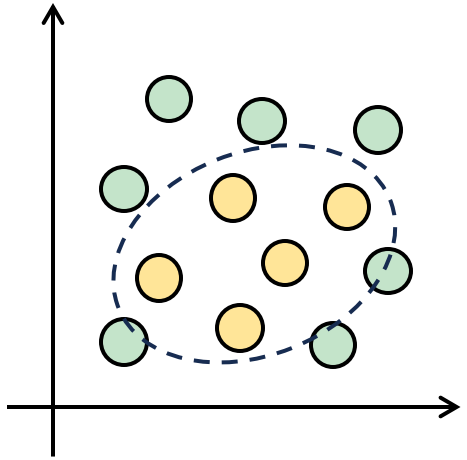
Binary classification on model parameters



Next Step

Binary classification on model parameters

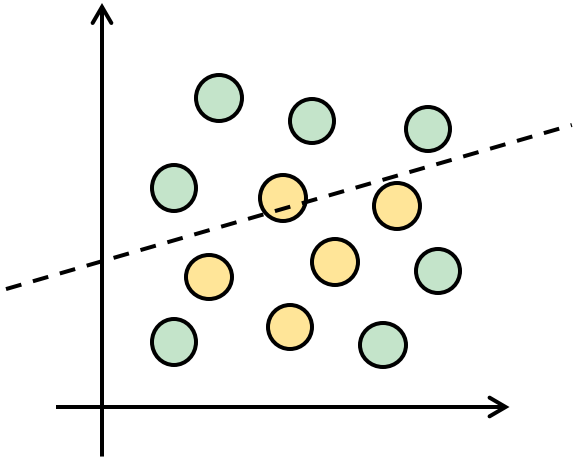


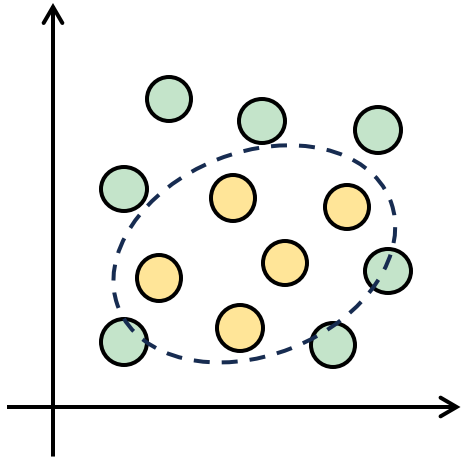


Design manually or through neural network fitting

Next Step

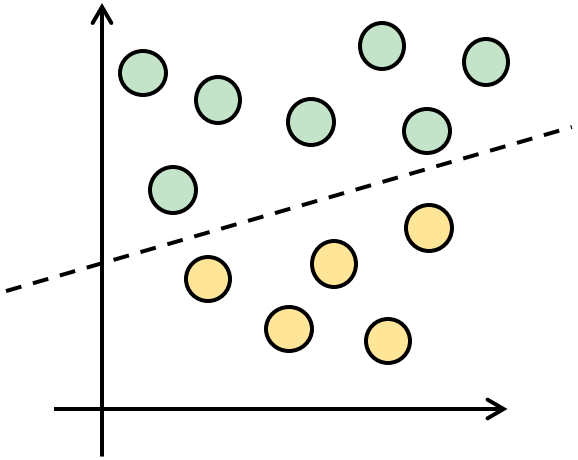
Binary classification on model parameters





Design manually or through neural network fitting

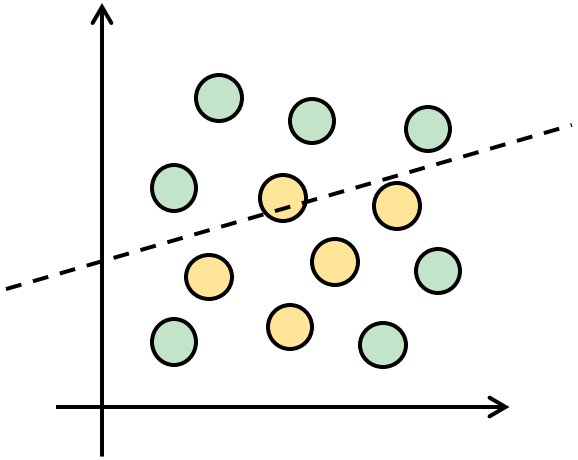
Is there a better alternative?

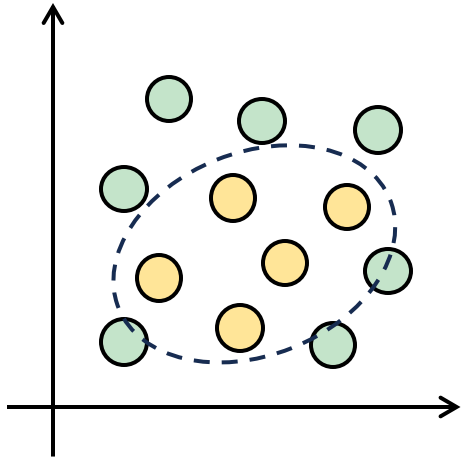


Change Sparsity Distribution

Next Step

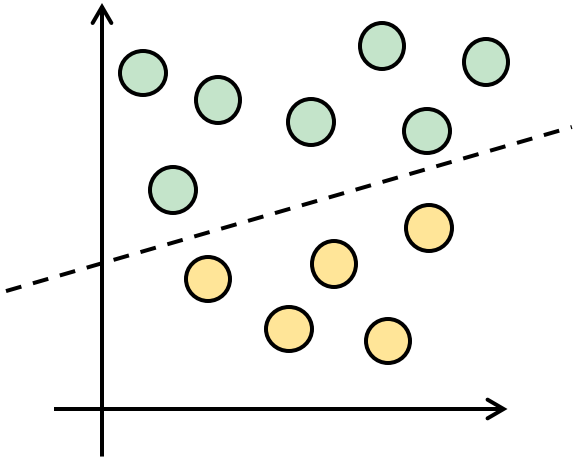
Binary classification on model parameters

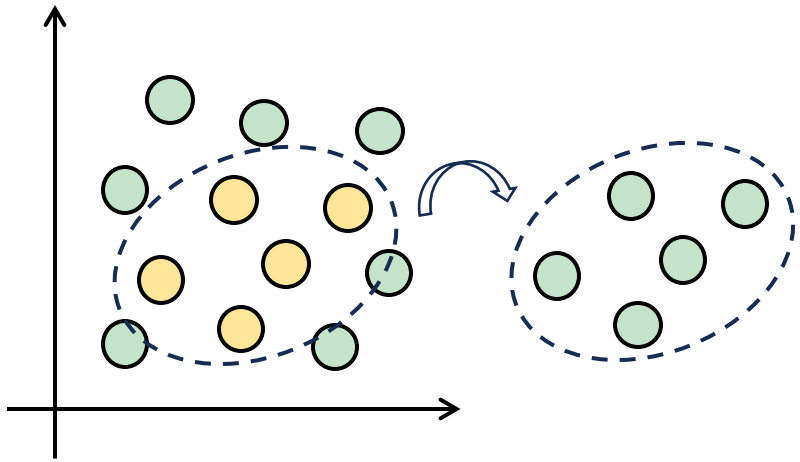




Design manually or through neural network fitting

Is there a better alternative?



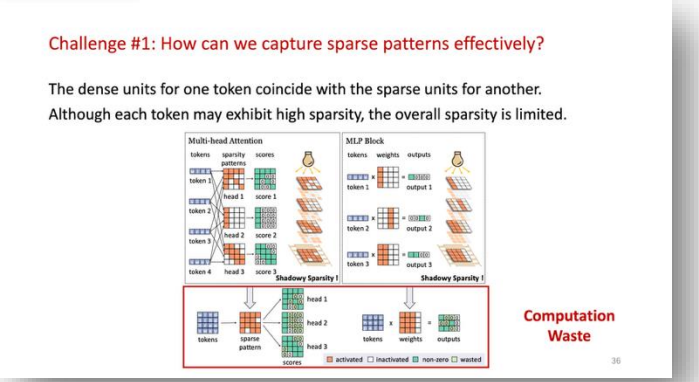


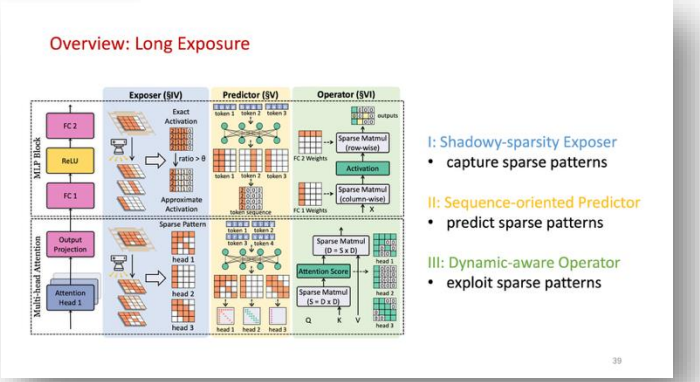
Change Sparsity Distribution

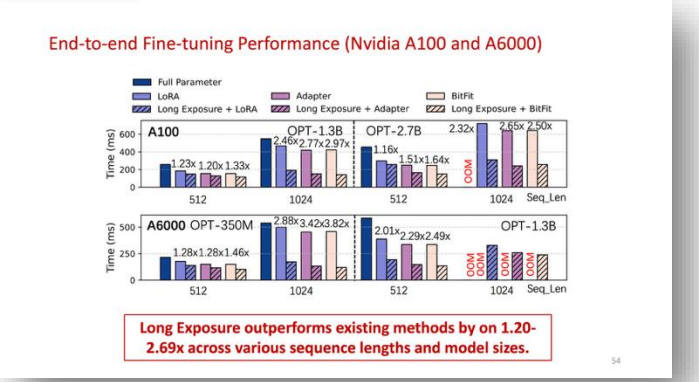
Increase Model Density

Long Exposure

- Identify and leverage the intrinsic sparsity within LLM fine-tuning, namely Shadowy Sparsity.
- Design three key components that capture, predict, and exploit sparsity patterns, respectively.
- Implement an end-to-end fine-tuning system compatible with various PEFT techniques.







Long Exposure

- Identify and leverage the intrinsic sparsity within LLM fine-tuning, namely Shadowy Sparsity.
- Design three key components that capture, predict, and exploit sparsity patterns, respectively.
- Implement an end-to-end fine-tuning system compatible with various PEFT techniques.

Thanks for your listening!

wtw23@mails.tsinghua.edu.cn

kunli@microsoft.com

renju@tsinghua.edu.cn