AI VIET NAM
@aivietnam.edu.vn

Module 10 – Project

PEFT for Multiple-choice QA

[Code - Data]

[Github]

Nguyen Quoc Thai
MSc in Computer Science

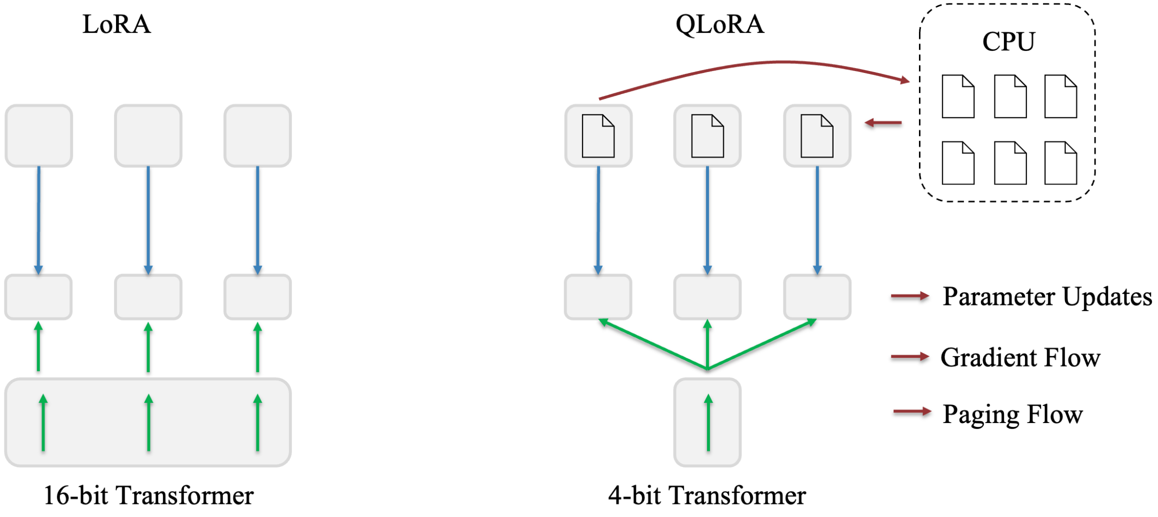
Objectives

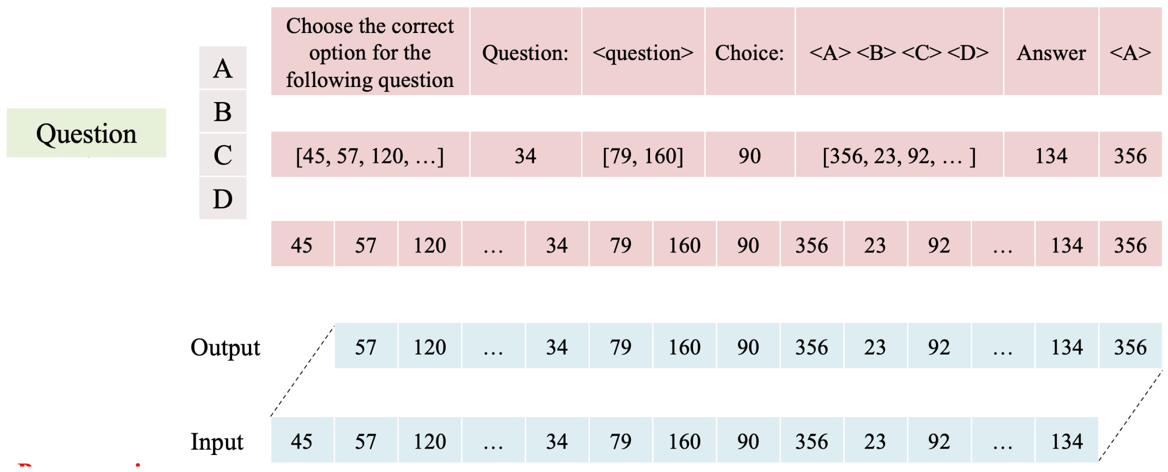
PEFT

- Fine-tuning LLMs
- Adapter / Prefix / Prompt Tuning
- LoRA, QLoRA

Multiple-choice QA

- Multiple-choice Question Answering
- MedMCQA Dataset
- Fine-tuning LLaMA-3.2-1B





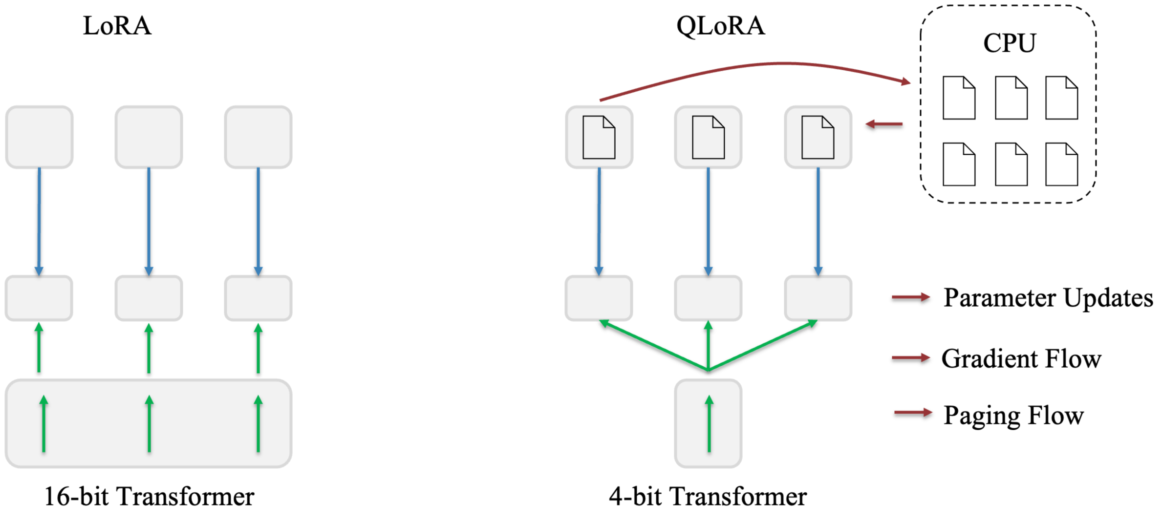
Outline

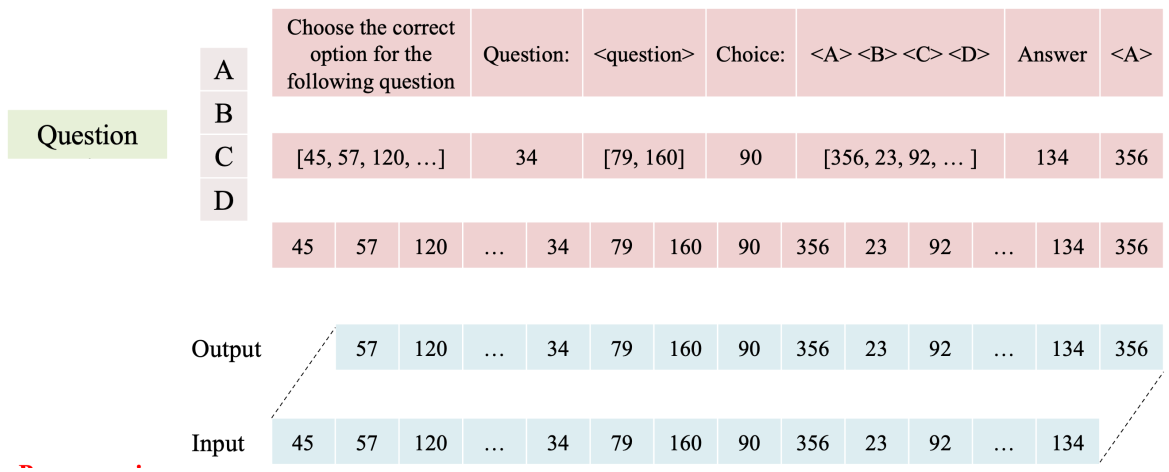
SECTION 1

Fine-tuning LLMs

SECTION 2

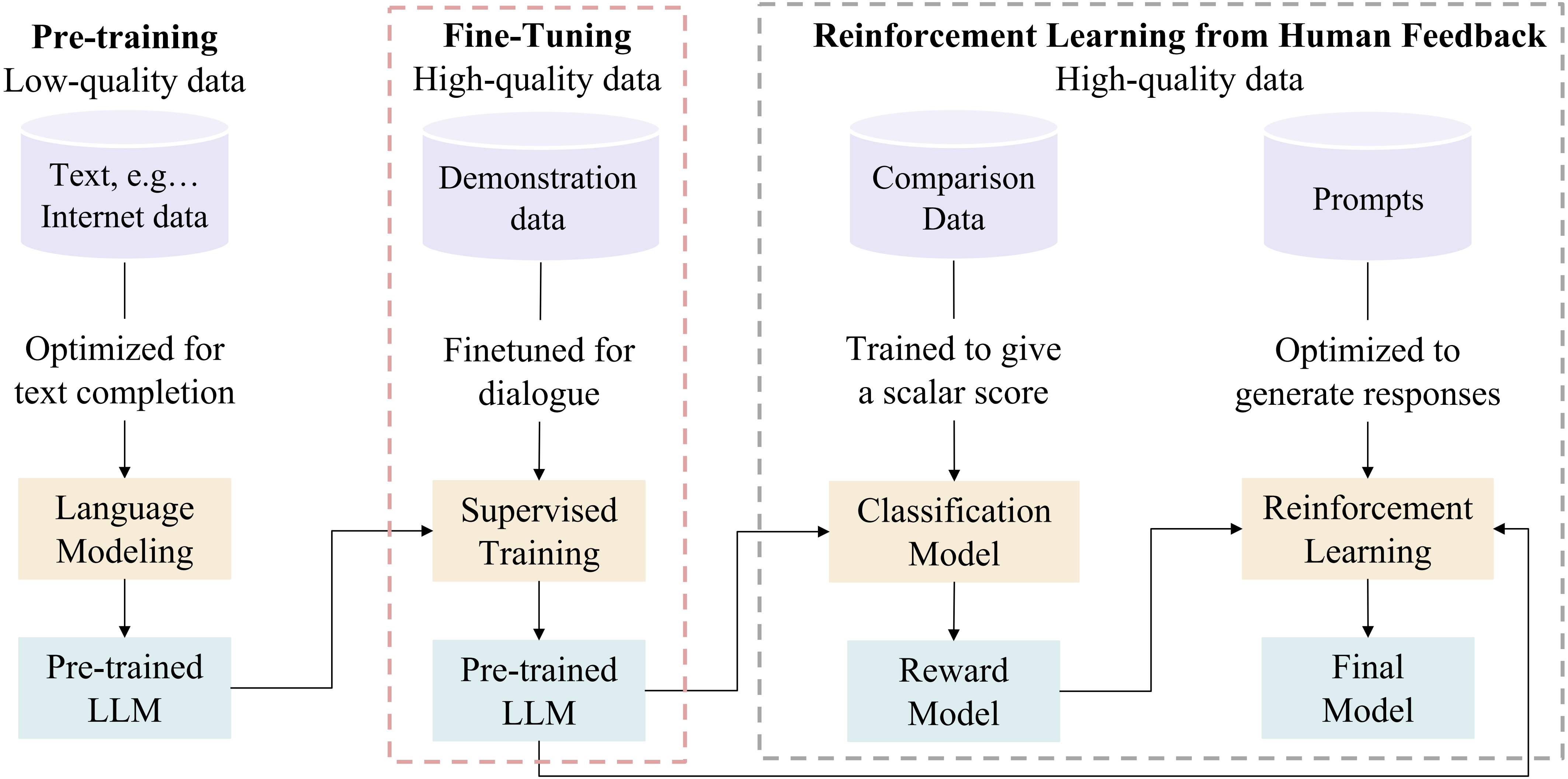
Multiple-choice QA





Training LLMs

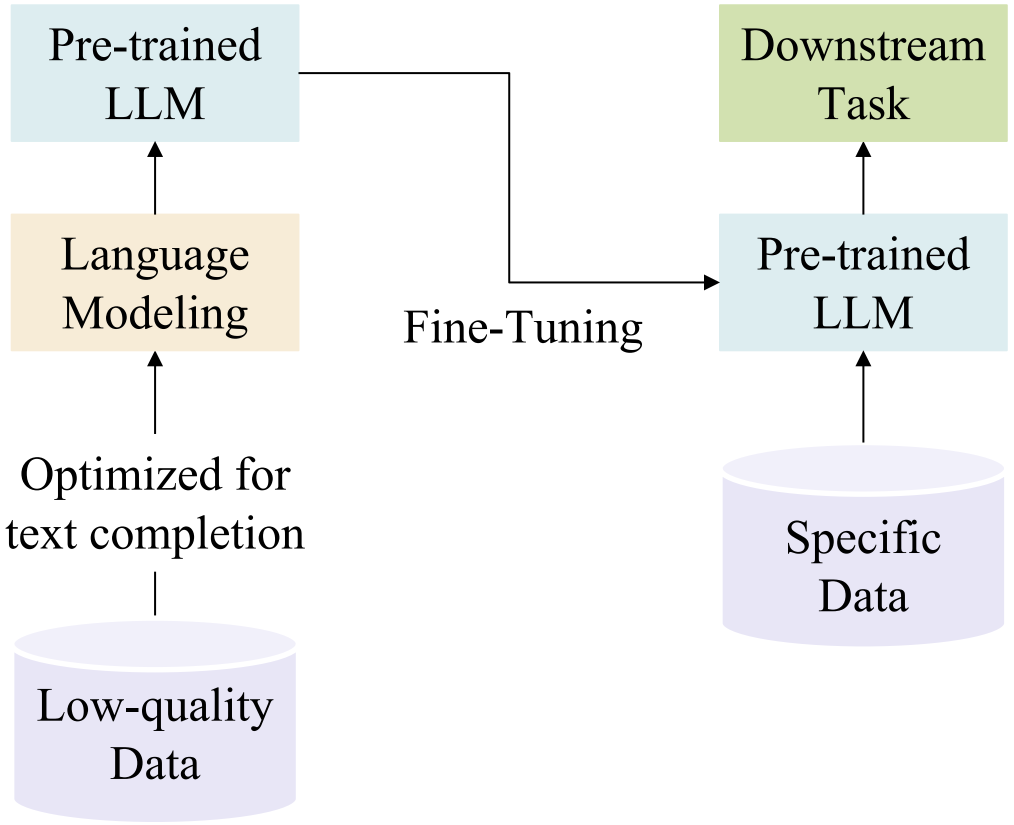
GPT-3.5 Training

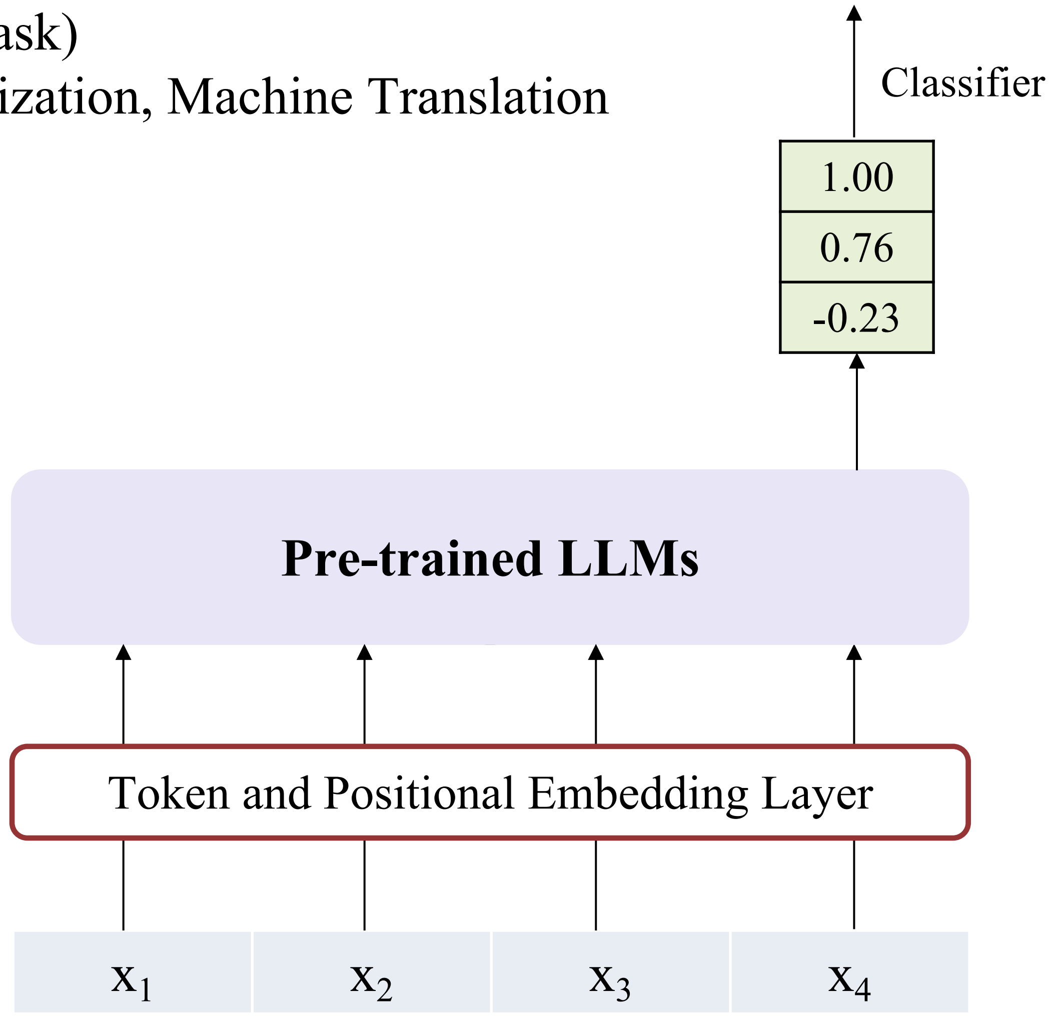


Fine-tuning LLMs

Fine-tuning LMs

- Embedded models into an application (Downstream task)
- Downstream tasks: Text Classification, Text Summarization, Machine Translation



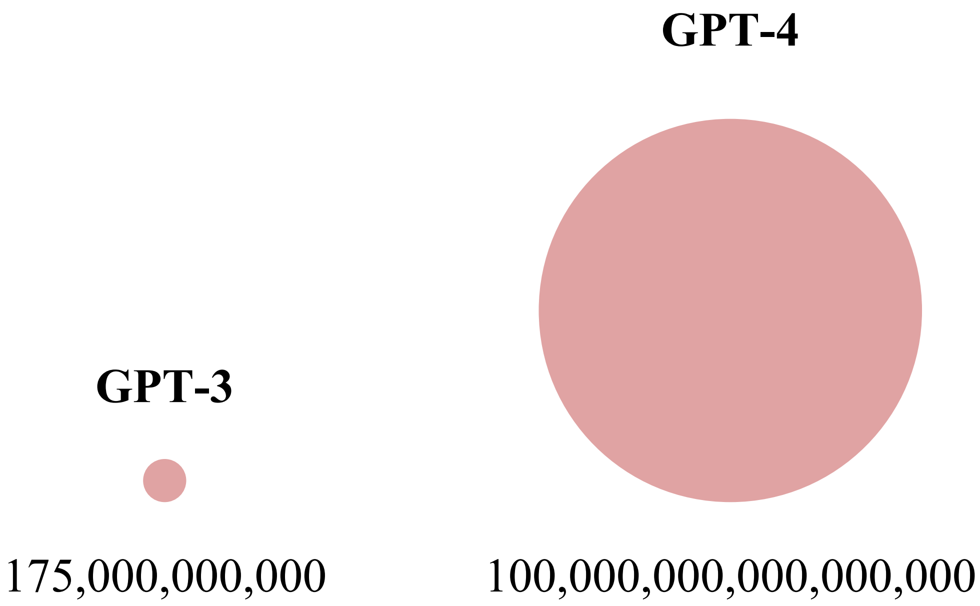


AI VIET NAM
@aivietnam.edu.vn

Fine-tuning LLMs

Model size are still growing?

- Model size scales almost two orders of magnitude quicker than single-GPU memory
- VRAM need for training / fine-tuning (full weights) GPT-3 175 B: 800– 1400 GB



Model	Parameter
GPT-4	1-1.8 T
Grok-1.5	1 T
LLaMA-4	2 T
Claude 3	200 B
Gemini 1.5	1 T
PaLM 2-Large	340 B

Fine-tuning LLMs

Parameter-Efficient Fine-Tuning (PEFT)

- Standard fine-tuning: make a new copy of the model weights for each task
- PEFT: perform fine-tuning of fewer parameters, but achieve performance on a downstream task that is comparable to fine-tuning of all parameters

Subset Fine-tuning

- Layer 1 (Frozen)
- Layer 2 (Frozen)
- Layer 3 (Fine-tuned)
- Layer 4 (Fine-tuned)

Only top K layers are fine-tuned

Adapters

- Layer 1 (Frozen)
- Adapter (Trainable)
- Layer 2 (Frozen)
- Adapter (Trainable)
- Layer 3 (Frozen)

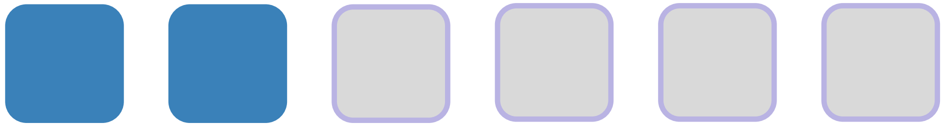
Small trainable modules between frozen layers

Fine-tuning LLMs

Parameter-Efficient Fine-Tuning (PEFT)

- Standard fine-tuning: make a new copy of the model weights for each task
- PEFT: perform fine-tuning of fewer parameters, but achieve performance on a downstream task that is comparable to fine-tuning of all parameters

Prefix Fine-tuning

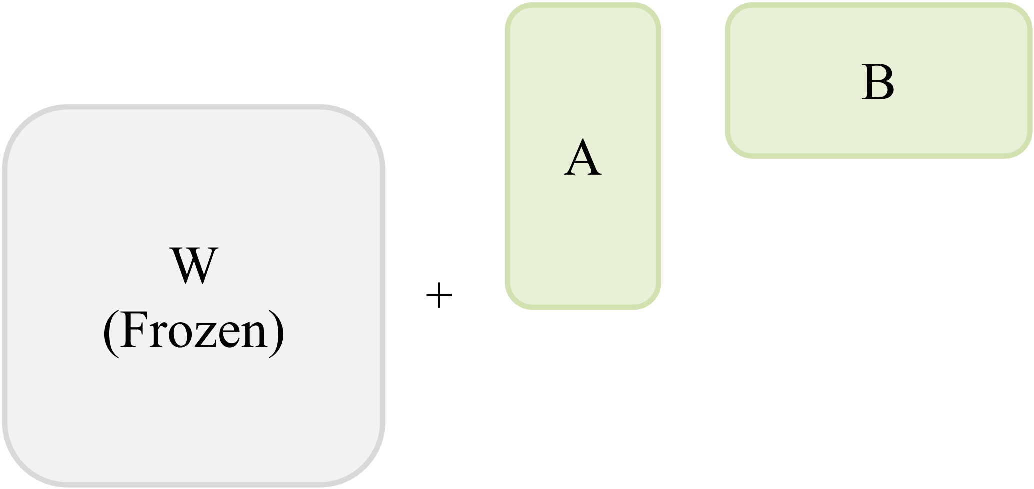


Trainable Prefix

Input Tokens (Frozen Model)

Tune virtual prefix token’s keys/values

LoRA

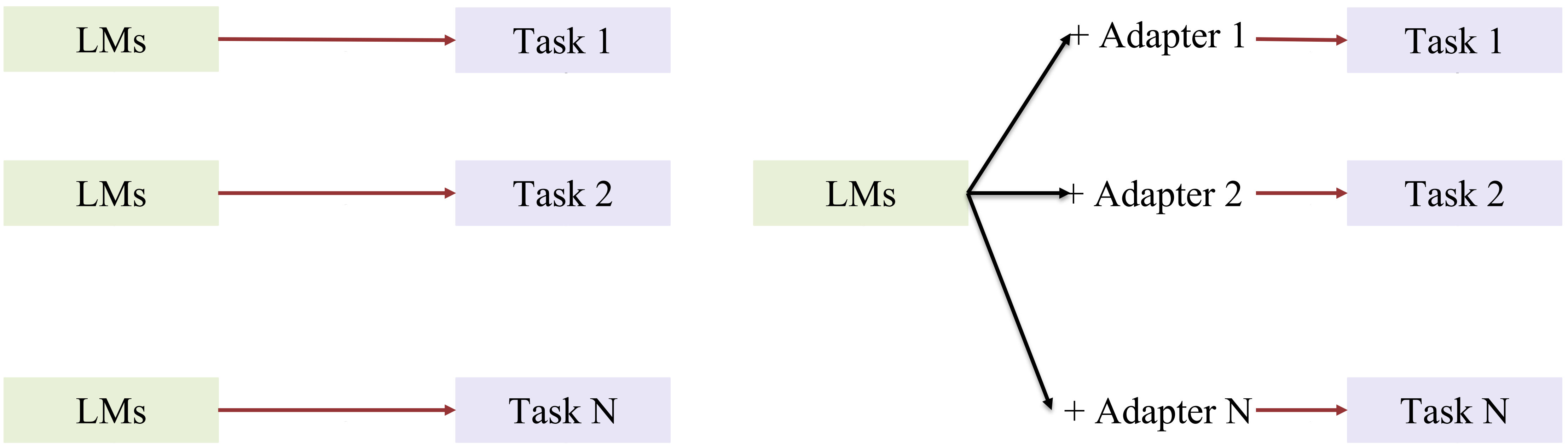


Low-rank weight W’ = A x B

Fine-tuning LLMs

Adapters

- Add a layer to adapt for downstream tasks
- An adapter layer is simply a feed-forward neural network with one hidden layer, and a residual connection

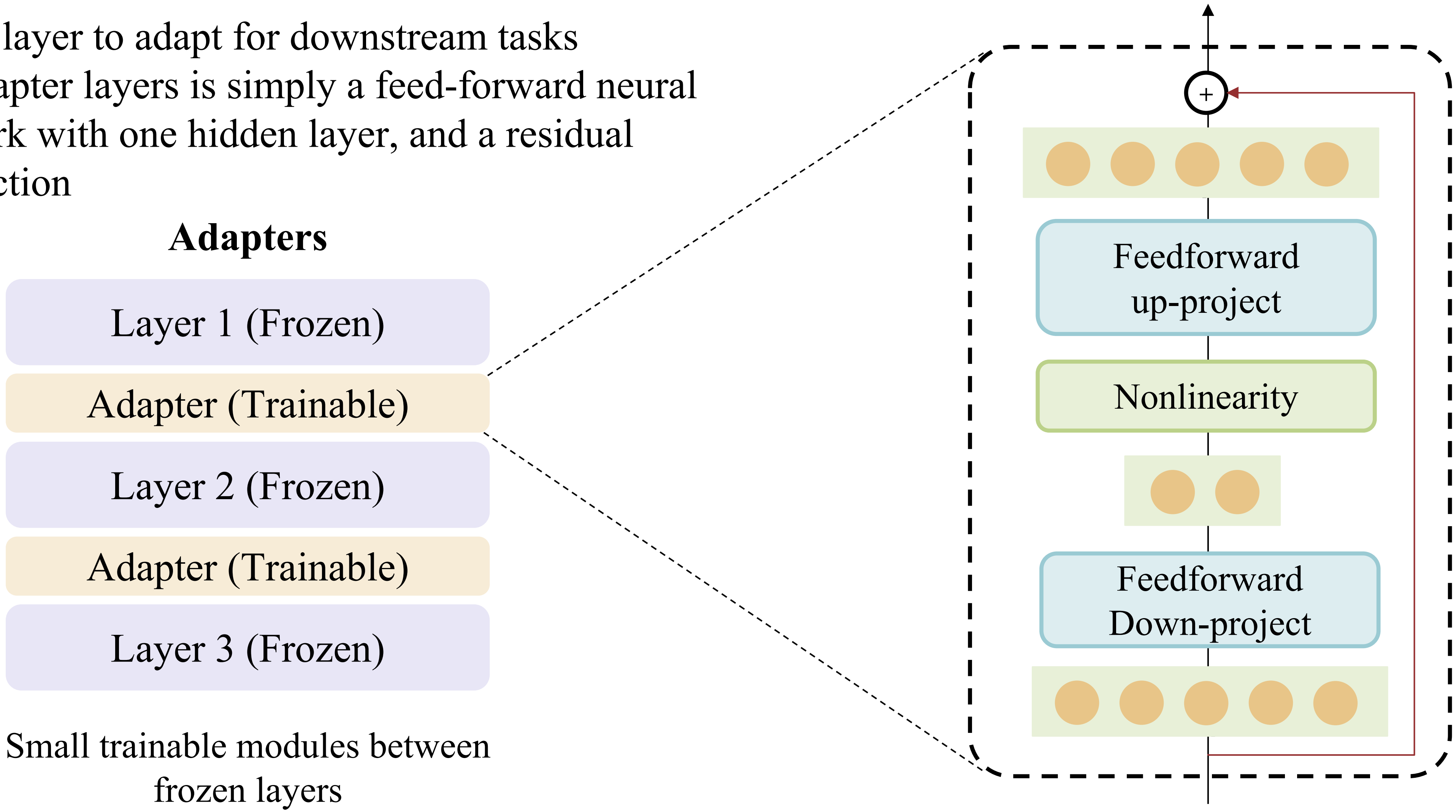


Fine-tuning LLMs

Adapters

- Add a layer to adapt for downstream tasks
- An adapter layers is simply a feed-forward neural network with one hidden layer, and a residual connection

Adapters

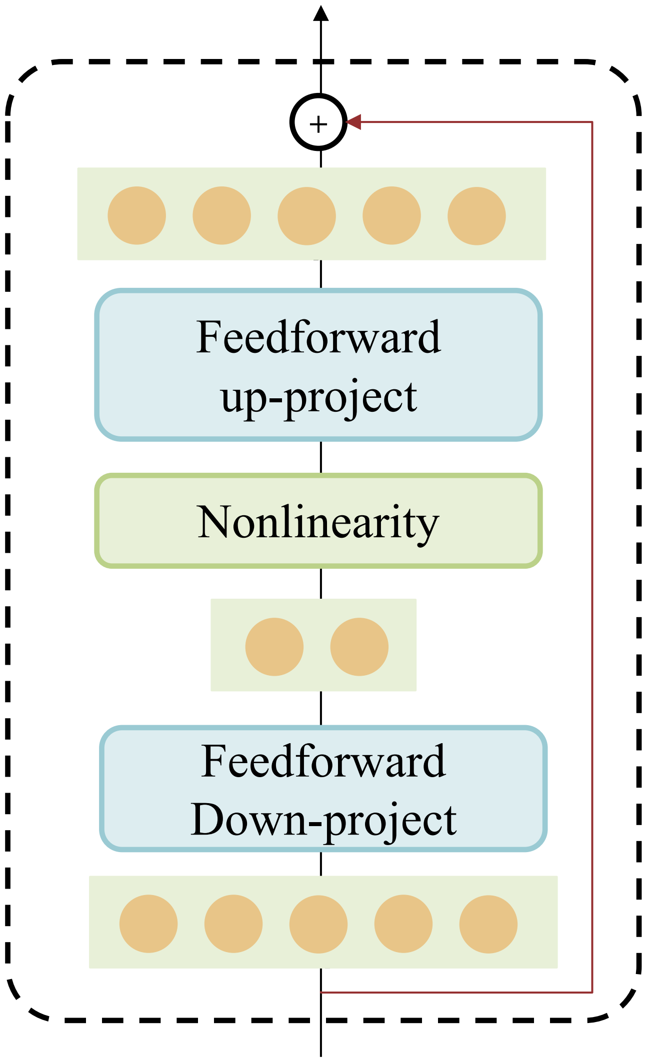


Small trainable modules between frozen layers

Fine-tuning LLMs

Adapters

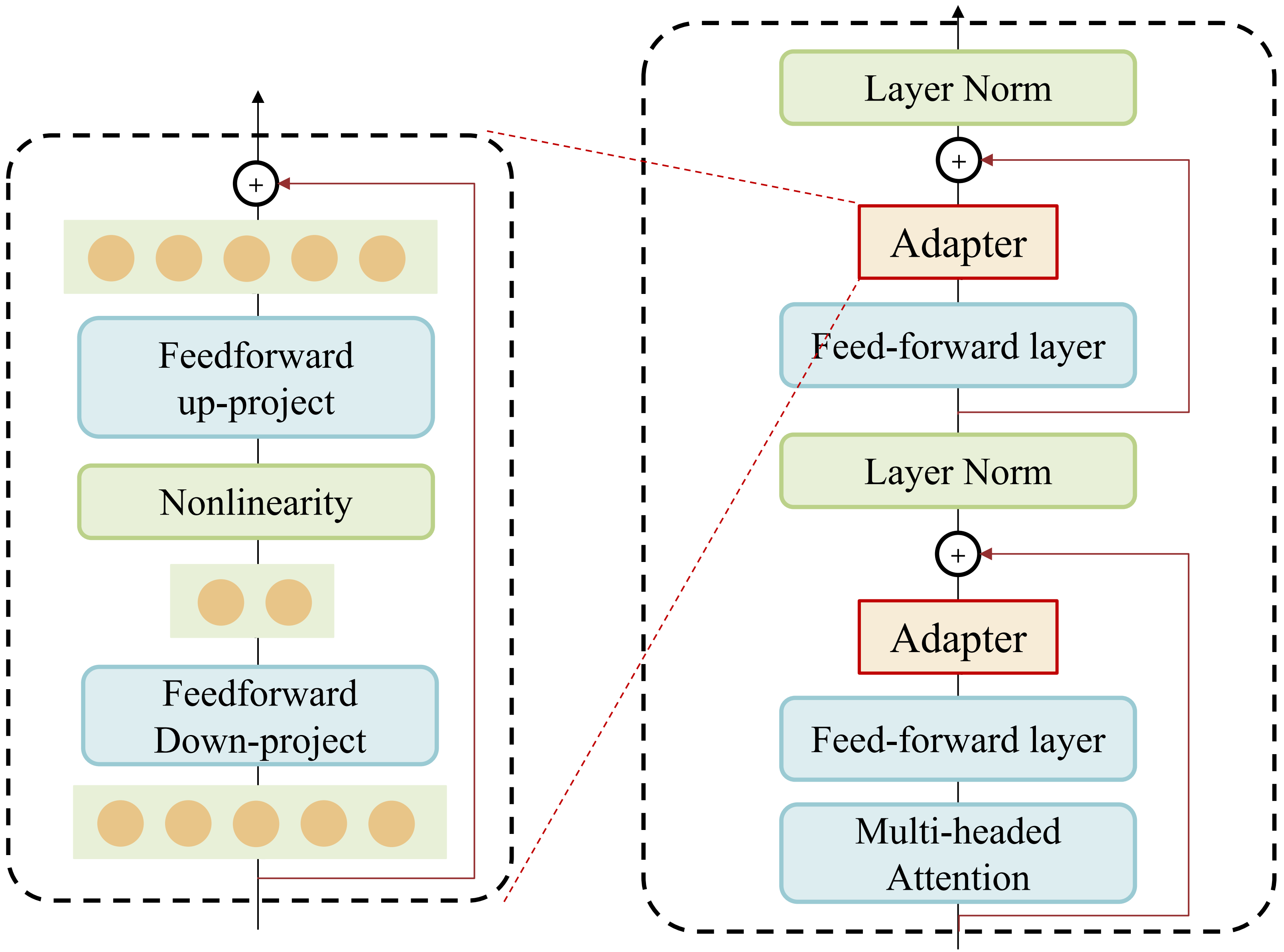
```python
import torch.nn as nn

class NoAdapter(nn.Module):
    def __init__(self, hidden_size=32):
        super().__init__()
        self.linear = nn.Linear(hidden_size, hidden_size)

    def forward(self, x):
        return self.linear(x)

class Adapter(nn.Module):
    def __init__(self, hidden_size=32, bottleneck_size=8):
        super().__init__()
        self.down_proj = nn.Linear(hidden_size, bottleneck_size)
        self.activation = nn.ReLU()
        self.up_proj = nn.Linear(bottleneck_size, hidden_size)

    def forward(self, x):
        return x + self.up_proj(self.activation(self.down_proj(x)))
```



Fine-tuning LLMs

Adapters

- Add adapter layers in the transformer layers of a large model
- During fine-tuning, fix the original model parameters and only tune the adapter layers.



Fine-tuning LLMs

Adapters

- 2 – 3 % of parameters needed

```
from adapters import AutoAdapterModel
from transformers import AutoModelForSequenceClassification

# Full weights
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased", num_labels=2
)
print(sum(p.numel() for p in model.parameters() if p.requires_grad))

# Adapter
model = AutoAdapterModel.from_pretrained("bert-base-uncased")
model.add_adapter("imdb_adapter")
model.train_adapter("imdb_adapter")
model.add_classification_head("imdb_adapter", num_labels=2)
print(sum(p.numel() for p in model.parameters() if p.requires_grad))
```

109 M

GLEU (Accuracy)
80.4

2 M

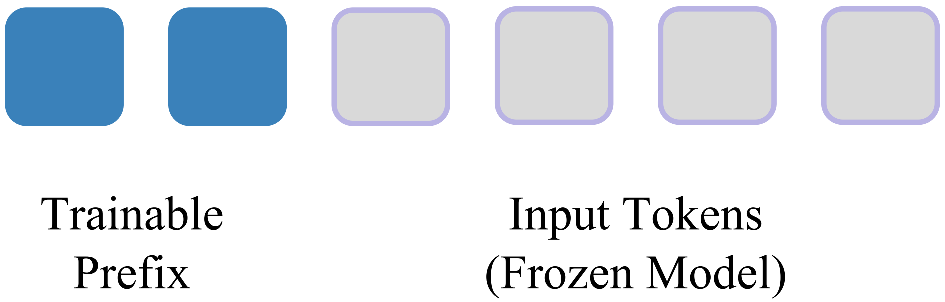
GLEU (Accuracy)
79.6 – 80.0

Fine-tuning LLMs

Prefix-Tuning

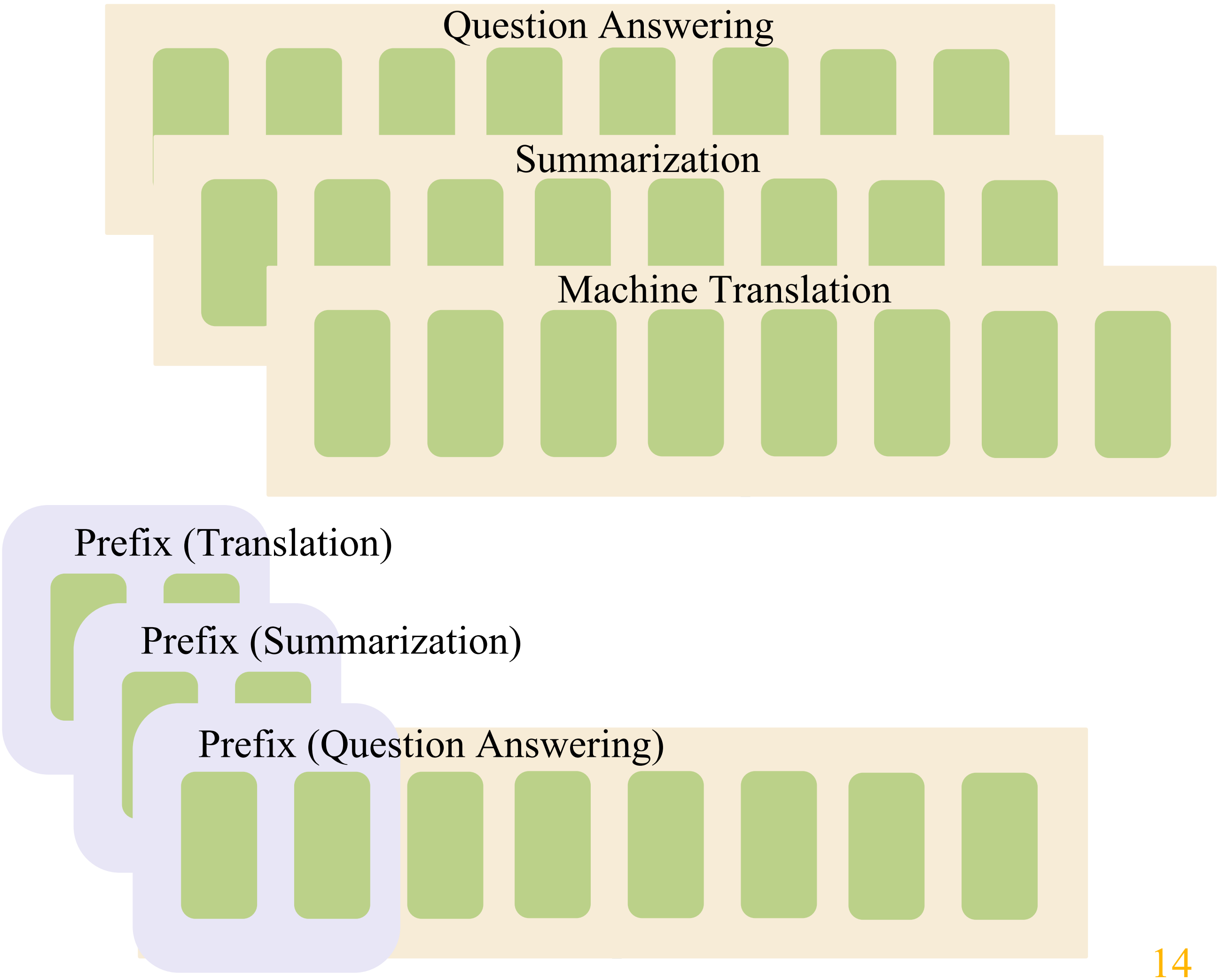
- Optimizing continuous prompts for generation
- Learn an optimal prefix for each task
- Only 1% of parameters need to be tuned

Prefix Fine-tuning



Tune virtual prefix token’s keys/values

Prefix (Translation)

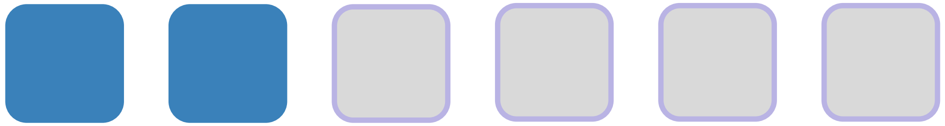


Fine-tuning LLMs

Prefix-Tuning

- As the tunable prefix-length increases, performance increase, with diminishing returns
- Optimal length for table to text is 10 tokens, summarization is 200 tokens,…

Prefix Fine-tuning

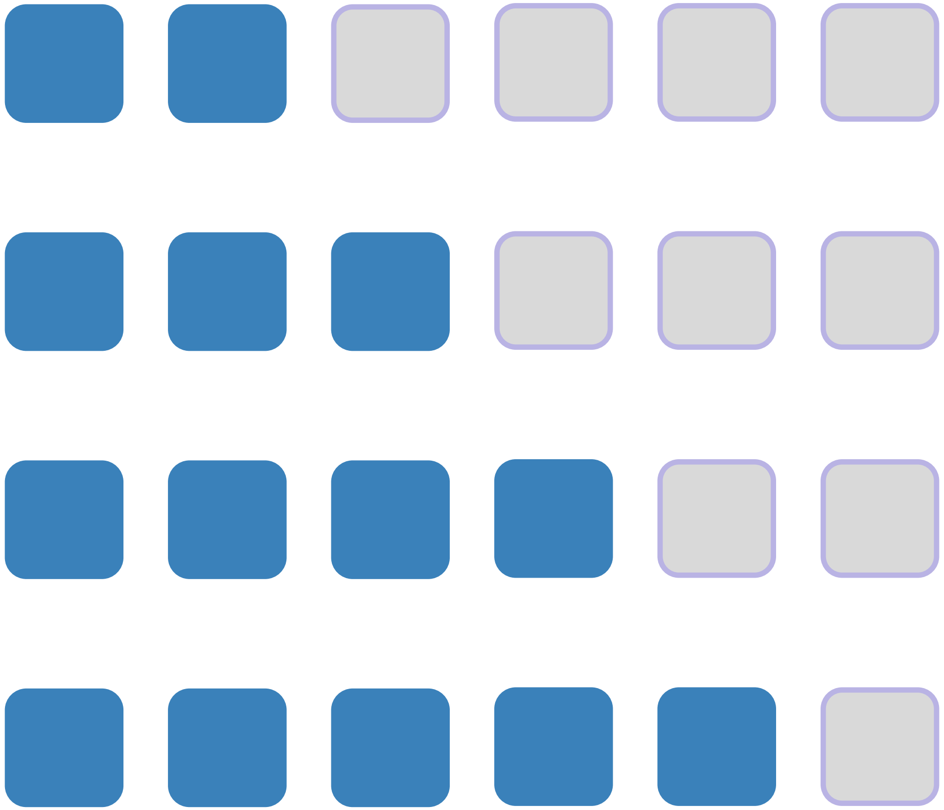


Trainable Prefix

Input Tokens (Frozen Model)

Tune virtual prefix token’s keys/values

Prefix (Translation)



Fine-tuning LLMs

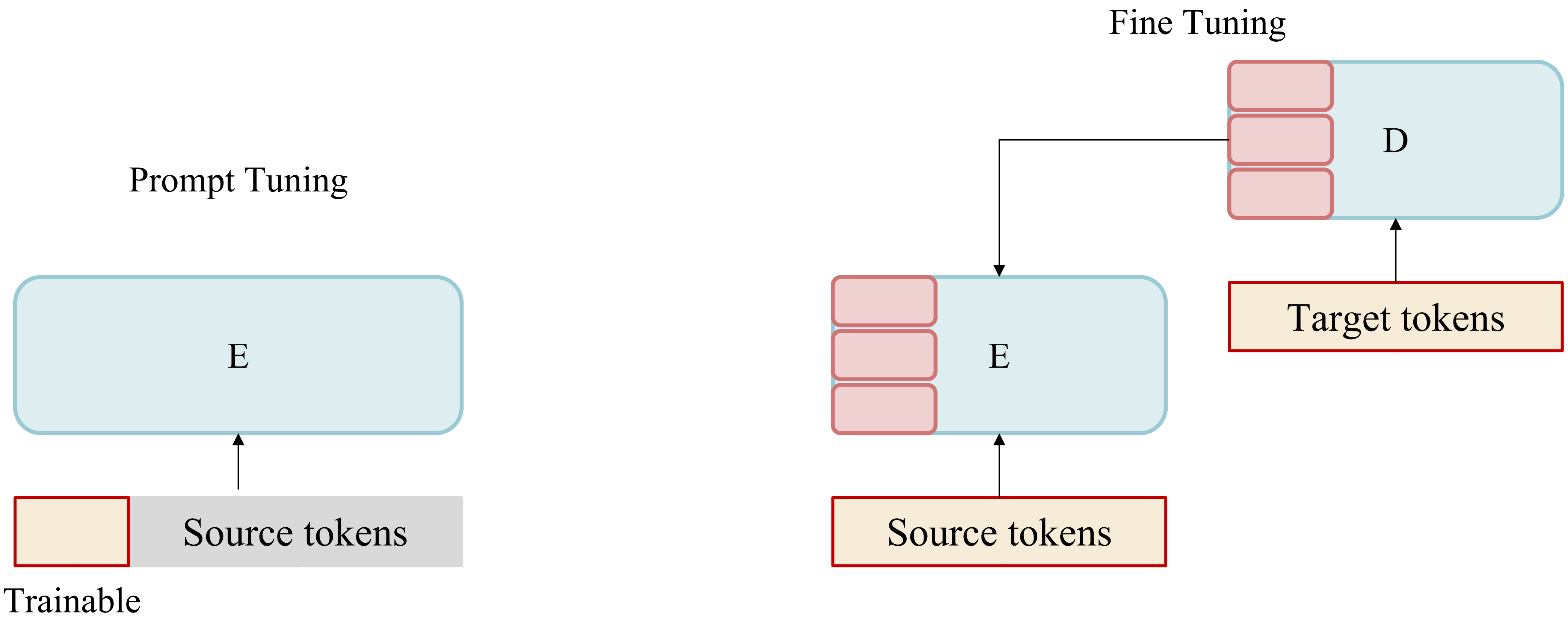
Prefix-Tuning

```python
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from peft import get_peft_model, PrefixTuningConfig, TaskType

model_name = "bert-base-uncased"
# Load model & tokenizer
model = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=2
)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Prefix Tuning config
peft_config = PrefixTuningConfig(
    task_type=TaskType.SEQ_CLS,
    num_virtual_tokens=10,
    encoder_hidden_size=768,  # for bert-base
    prefix_projection=True    # optional: adds an MLP to project prefix tokens
)

# Wrap model with PEFT
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()
```

Fine-tuning LLMs

Prompt-Tuning

- Prefix-tuning learn a sequence of prefixes (are prepended at every transformer layer)
- Prompt-tuning uses a single prompt representation is prepended to the embedded input



Fine-tuning LLMs

Prompt-Tuning

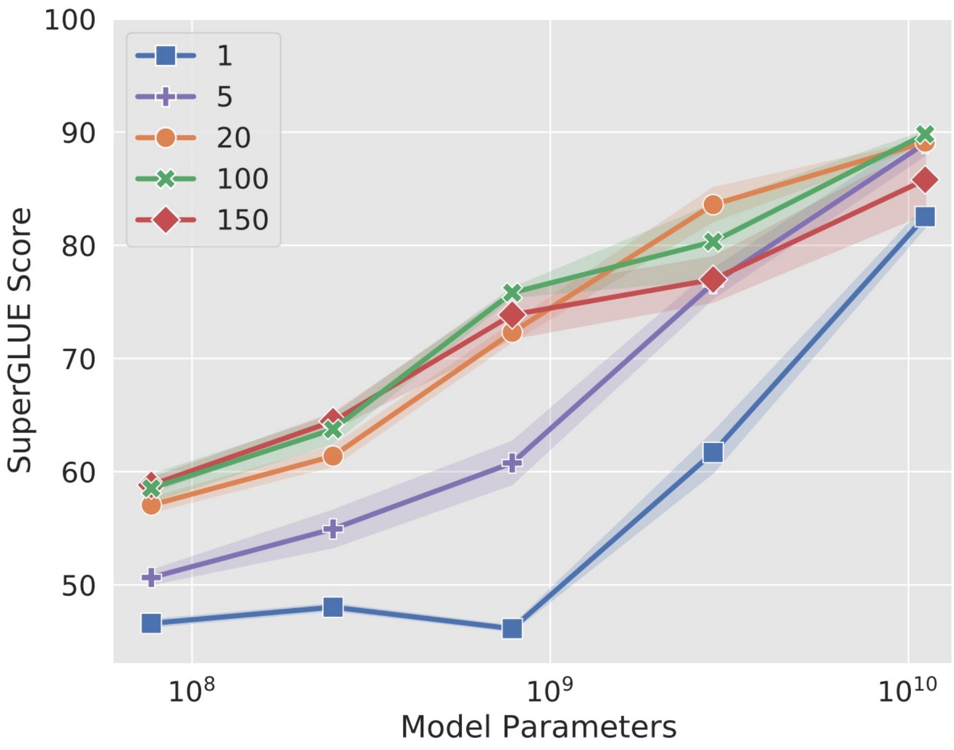
- Trainable params: 18,436
- All params: 109,500,676
- Trainable%: 0.016

```python
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=2
)

# Prompt Tuning config
peft_config = PromptTuningConfig(
    task_type=TaskType.SEQ_CLS,
    num_virtual_tokens=20,
    tokenizer_name_or_path=model_name,
)

model = get_peft_model(model, peft_config)
model.print_trainable_parameters()
```

Fine-tuning LLMs

Prompt-Tuning

- Experiments (Evaluating on SuperBLEU)

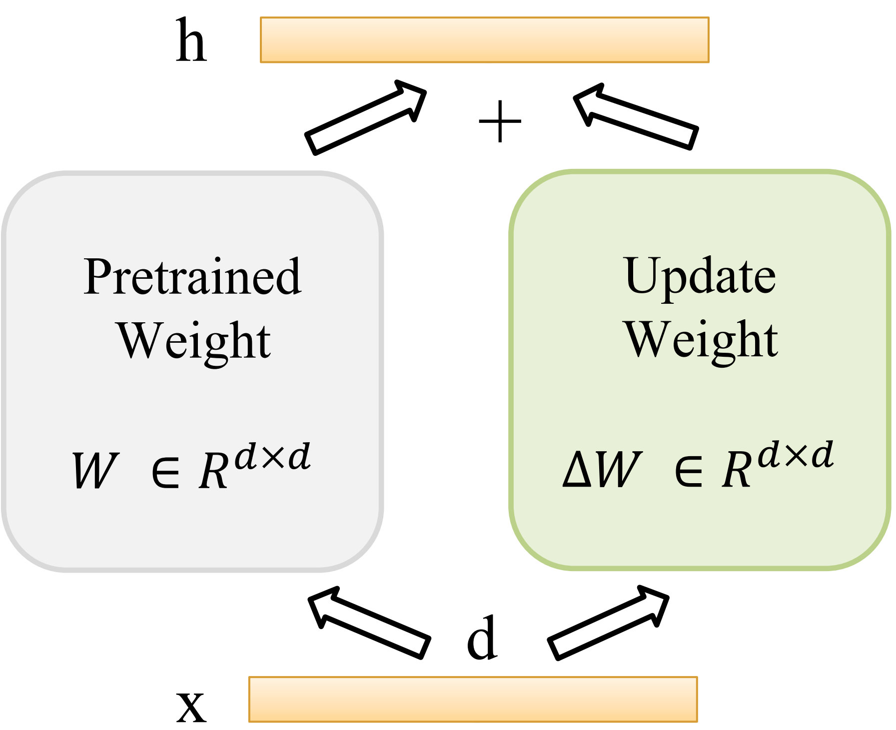


The Power of Scale for Parameter-Efficient Prompt Tuning

Fine-tuning LLMs

Low-rank Adaptation (LoRA)

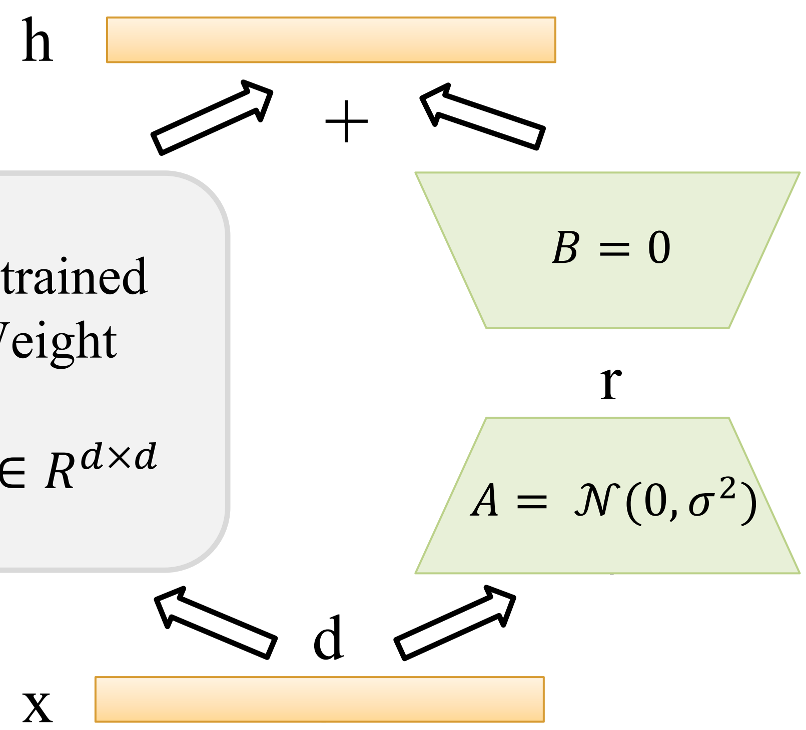
- By freezing the pre-trained model weights and injecting trainable rank decomposition matrices into each layer of the Transformer architecture.

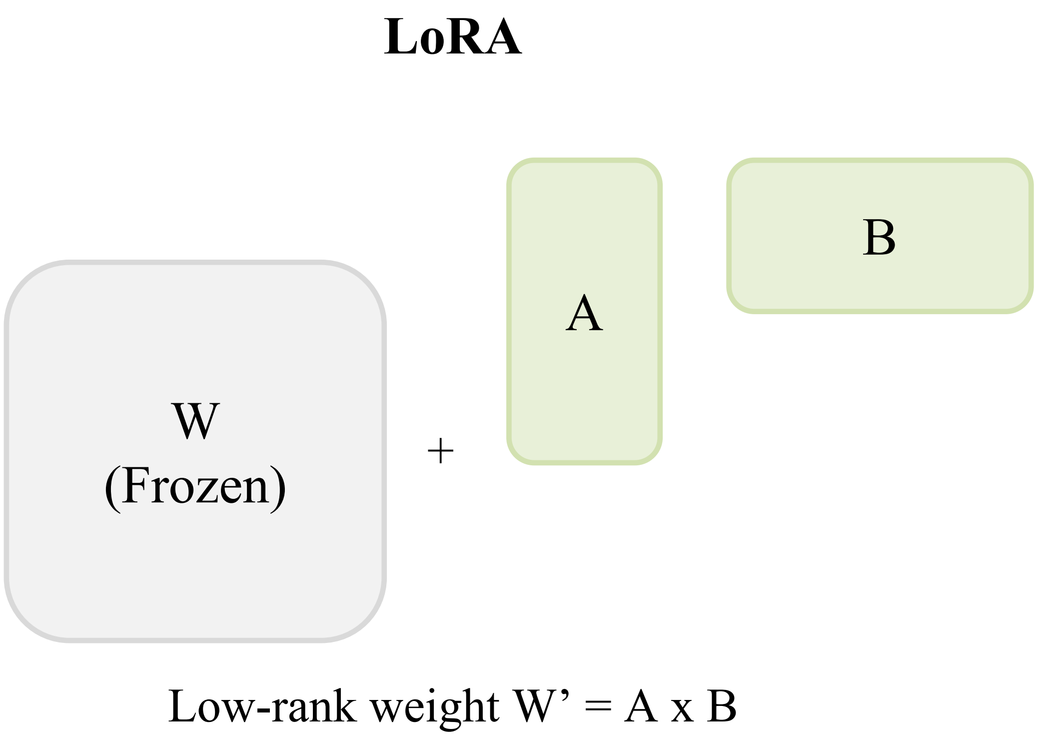


Fine-tuning LLMs

Low-rank Adaptation (LoRA)

- By freezing the pre-trained model weights and injecting trainable rank decomposition matrices into each layer of the Transformer architecture.

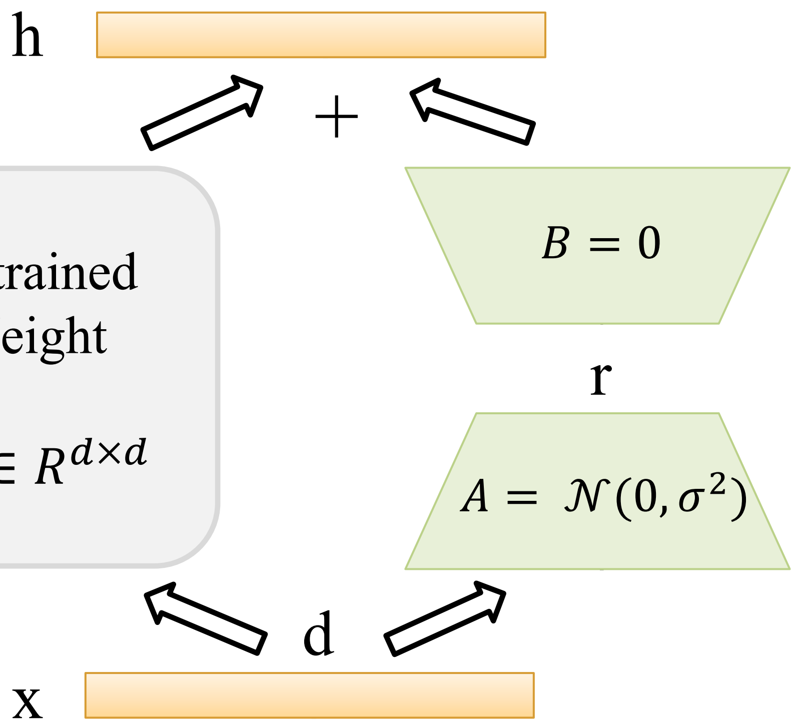




Fine-tuning LLMs

Low-rank Adaptation (LoRA)

- LoRA reimagines fine tuning not as learning better parameters, but as adjustments required to the existing parameters to make them better
- Pre-trained language models have a low “intrinsic dimension”



$$\overbrace{\mathrm{W_{ft}}}^{\text{Finetuned Weights}} = \underbrace{\mathrm{W_{pt}}}_{\text{Pretrained Weights}} + \overbrace{\Delta\mathrm{W}}^{\text{Weight Update}}$$

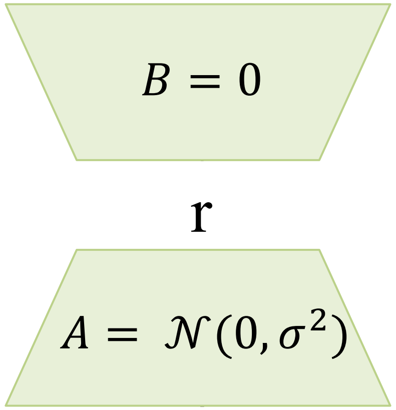
$$W_0 x + \Delta W x = W_0 x + BAx$$

Fine-tuning LLMs

LoRA Explained

- Saving memory with LoRA: The full 5x5 matrix above has 25 values in it, whereas if count the values in the decomposed matrices, there just 10

$$W_0x + \Delta W x = W_0 x + BAx$$



$$\begin{bmatrix} 1 \\ 2 \\ -1 \\ 3 \\ 1 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & -1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 3 & -1 & 2 & 0 \\ 4 & 6 & -2 & 4 & 0 \\ -2 & -3 & 1 & -2 & 0 \\ 6 & 9 & -3 & 6 & 0 \\ 2 & 3 & -1 & 2 & 0 \end{bmatrix}$$

Fine-tuning LLMs

How Does LoRA Work?

- First, freeze the model parameters (Using these parameters to make inferences, not update them)
- Create two matrices, calculate the change matrix (delta W)

Model Parameters

1	0	2	3	-1
2	0	2	2	1
1	1	2	3	3
2	0	1	1	1
1	2	3	1	1

Fine tune matrices A & B

2	3	-1	2	0

1
2
-1
3
1

=

Change Matrix

2	3	-1	2	0
4	6	-2	4	0
-2	-3	1	-2	0
6	9	-3	6	0
2	3	-1	2	0

Fine-tuning LLMs

How Does LoRA Work?

- Pass through the frozen weights and the change matrix
- Calculate the loss and update matrices A and B

Output → Loss Function

+

Model Parameters

1	0	2	3	-1
2	0	2	2	1
1	1	2	3	3
2	0	1	1	1
1	2	3	1	1

Change Matrix

2	3	-1	2	0
4	6	-2	4	0
-2	-3	1	-2	0
6	9	-3	6	0
2	3	-1	2	0

1
1
-1
2
1

2	1	-1	2	0

Update matrices A & B

Input

Fine-tuning LLMs

How Does LoRA Work?

- At inference time, add the change matrix to the frozen weights and pass the input.

$$h = W_0 x + \frac{\alpha}{r} \Delta W x = W_0 x + \frac{\alpha}{r} BAx$$

Output

+

Model Parameters

1	0	2	3	-1
2	0	2	2	1
1	1	2	3	3
2	0	1	1	1
1	2	3	1	1

Change Matrix

2	1	-1	2	0
2	1	-1	2	0
-2	-1	1	-2	0
4	2	-2	4	0
2	1	-1	2	0

2	1	-1	2	0

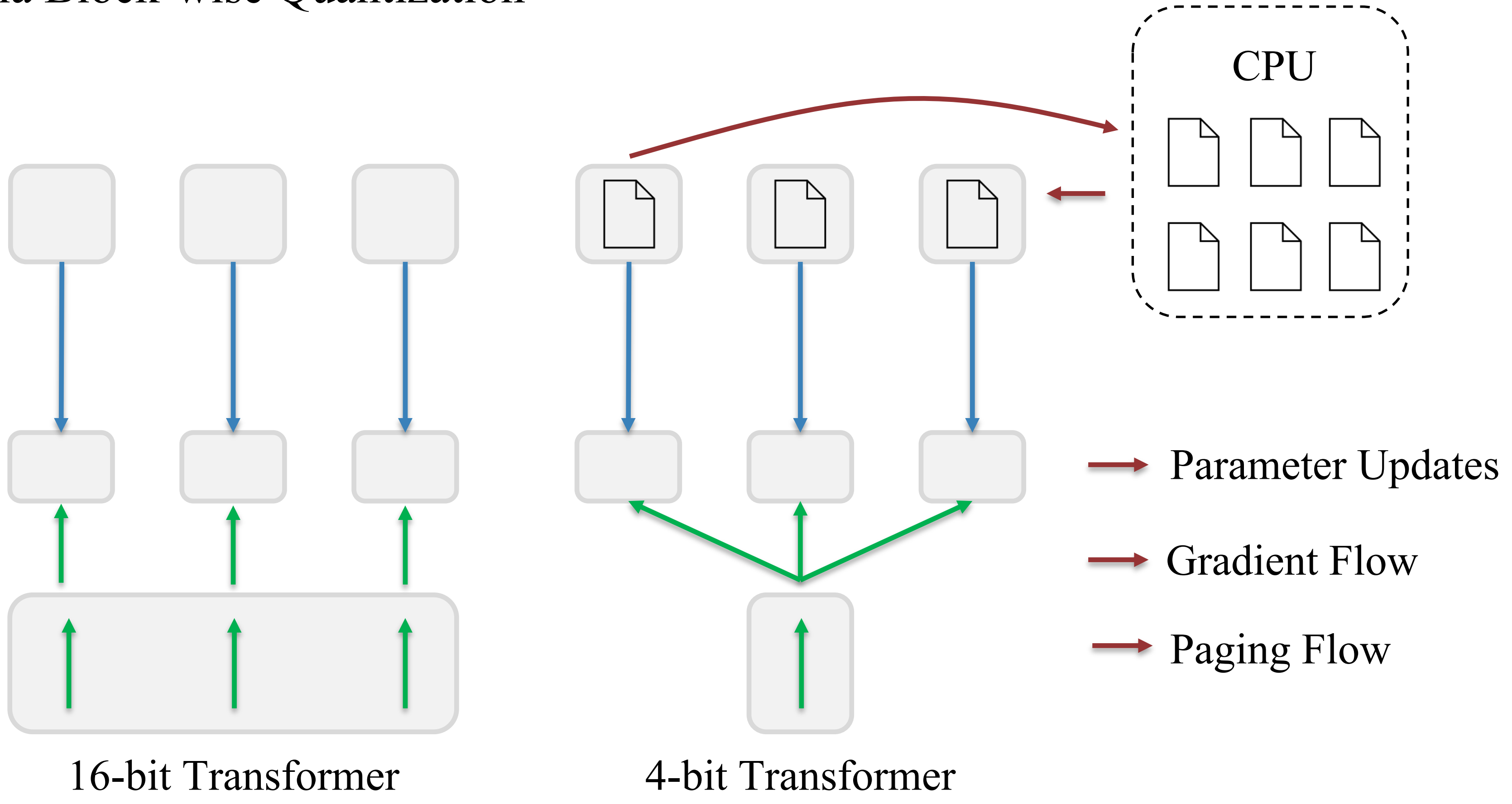
1
1
-1
2
1

Input

Fine-tuning LLMs

QLoRA: Efficient Finetuning of Quantized LLMs

- QLoRA: save memory without sacrificing performance
- 4-bit NormalFloat (NF4) via Block-wise Quantization
- Double Quantization
- Paged Optimizers
- Combined with LoRA

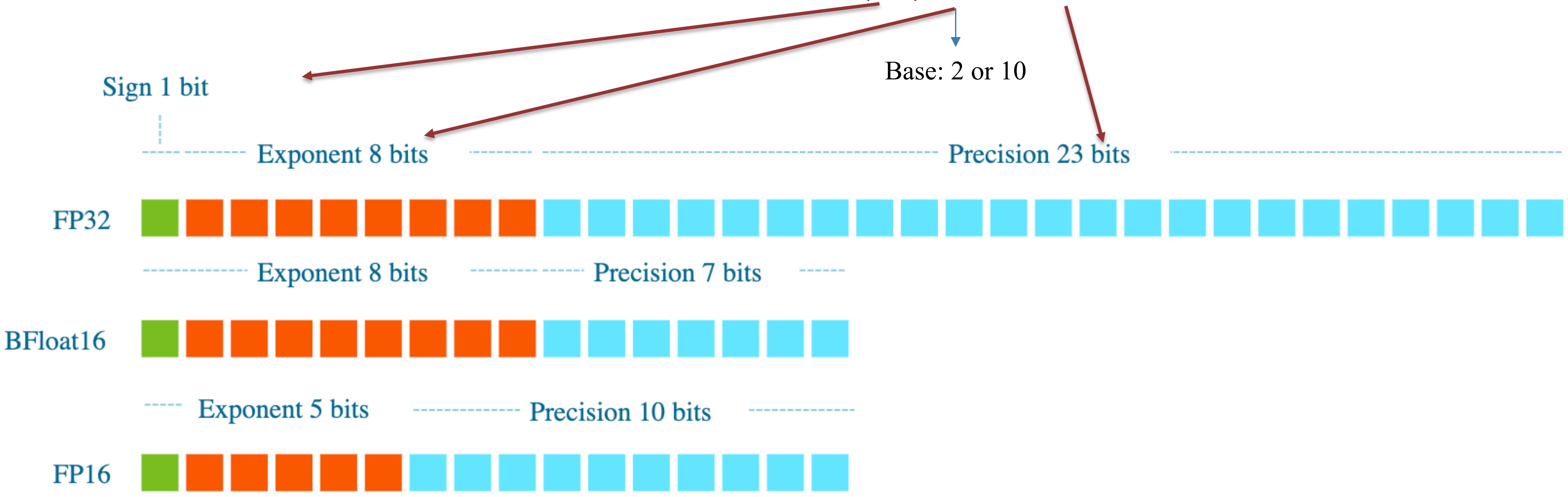


Fine-tuning LLMs

Data Type

- FP32: Single Precision Floating Point
- FP16: Half Precision Floating Point
- BFLOAT16: Brain Floating Point

$$+0.15625 = (+1) \times 2^{-3} \times 1.25$$



Fine-tuning LLMs

Quantization

- FP32: Quantization: mapping input values from a large set (often a continuous set) to outputs values in a (countable) smaller set.
- Quantize from dtype FP32 to target dtype INT8.
- INT8: [-127, 127]

$$X^{Int8} = \mathrm{round}\left(\frac{127}{absmax(X^{FP32})} X^{FP32}\right) = \mathrm{round}(c^{FP32} X^{FP32})$$

c: constant

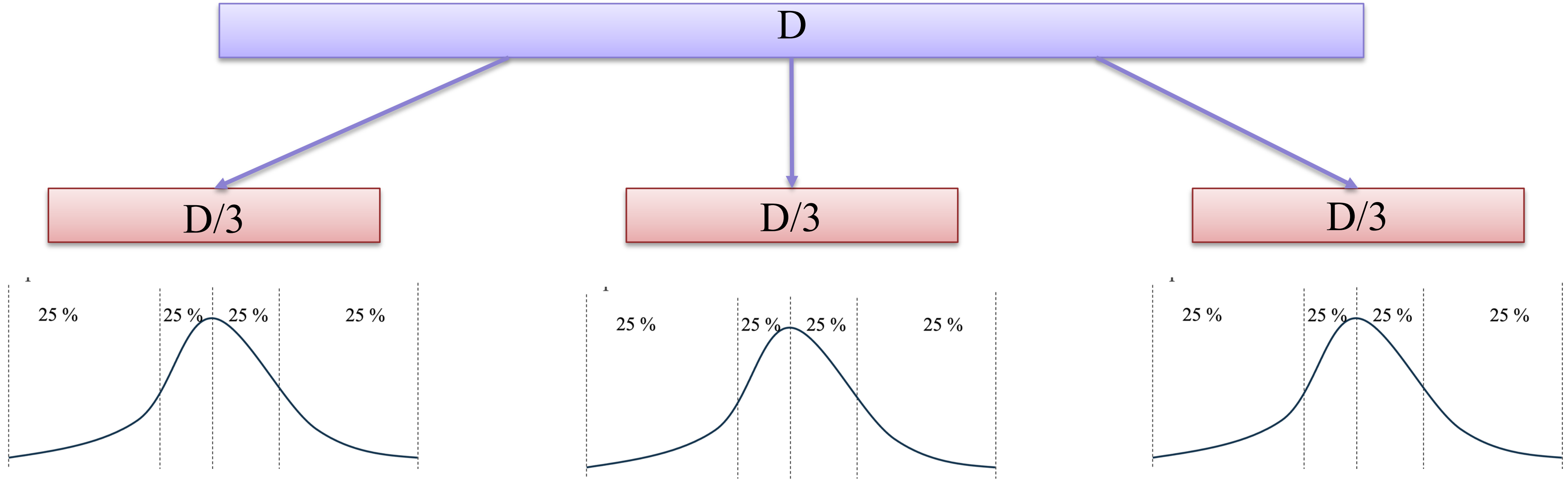
$$\text{FP32} \quad \boxed{0.1 \mid 0.2 \mid 0.4} \longrightarrow c = \frac{127}{0.4} = 317.5 \longrightarrow \boxed{32 \mid 64 \mid 127} \quad \text{INT8}$$

$$\mathrm{dequant}(c^{FP32} X^{FP32}) = \frac{X^{Int8}}{c^{FP32}} = X^{FP32}$$

Fine-tuning LLMs

QLoRA: 4-bit NormalFloat (NF4)

- Find quantiles in each chunks
- Use fixed distribution zero-mean normal distribution with standard deviation σ



=> The main limitation of quantile quantization: process of quantile estimation very expensive

Fine-tuning LLMs

QLoRA: 4-bit NormalFloat (NF4)

- Step 1: Estimate the 2k + 1 quantiles of a theoretical N(0, 1) distribution to obtain a k-bit quantile quantization data type for normal distributions

$$q_i = \frac{1}{2}\left(Q_X\left(\frac{i}{2^k+1}\right) + Q_X\left(\frac{i}{2^k+1}\right)\right)$$

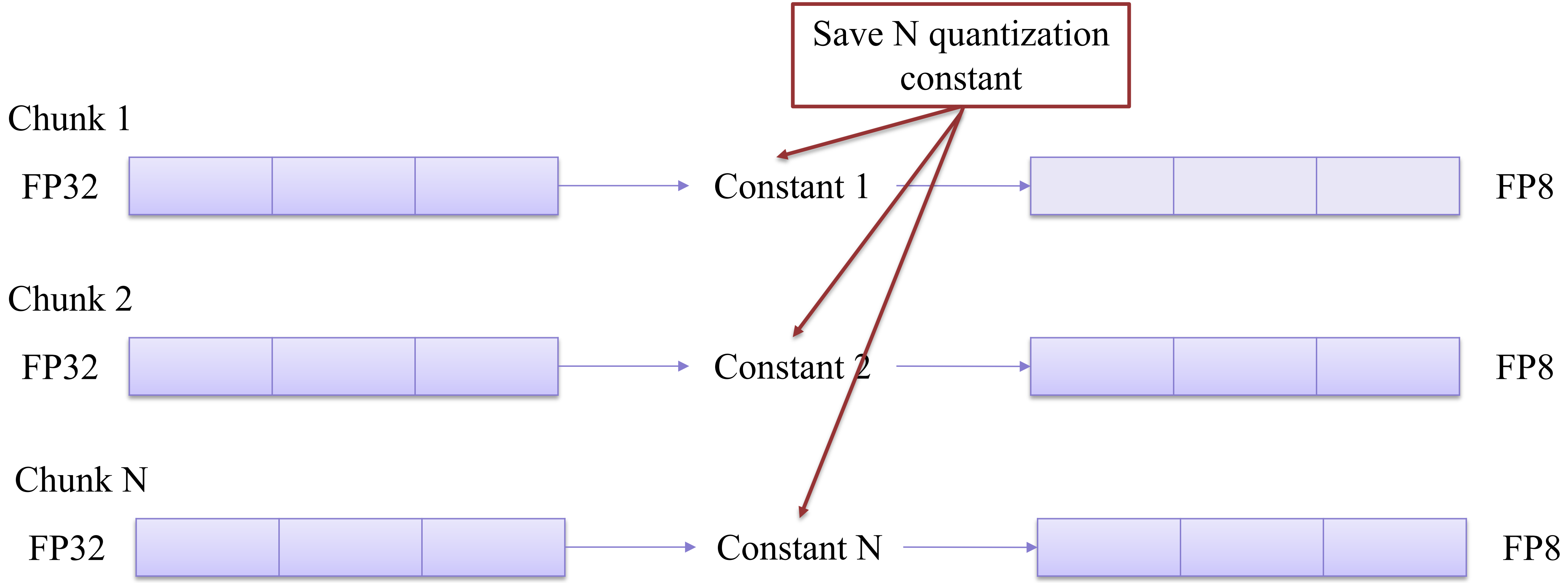
Q_X the quantile function of the standard normal distribution N(0,1)

- Step 2: Take this data type and normalize its values into the [−1, 1] range

- Step 3: Quantize an input weight tensor by normalizing into the [−1, 1] range through absolute maximum rescaling

Fine-tuning LLMs

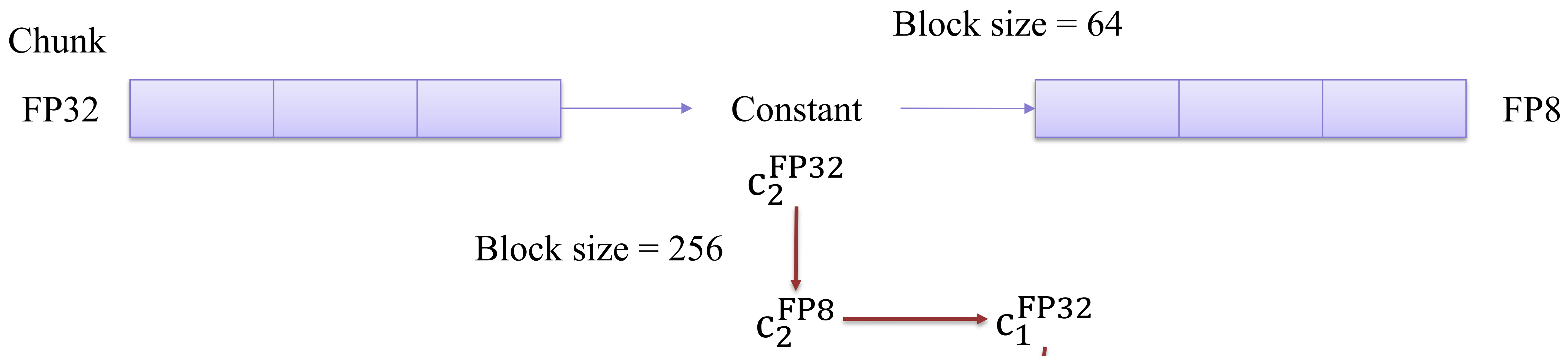
QLoRA: Double Quantization



Fine-tuning LLMs

QLoRA: Double Quantization

- The process of quantizing the quantization constants for additional memory savings

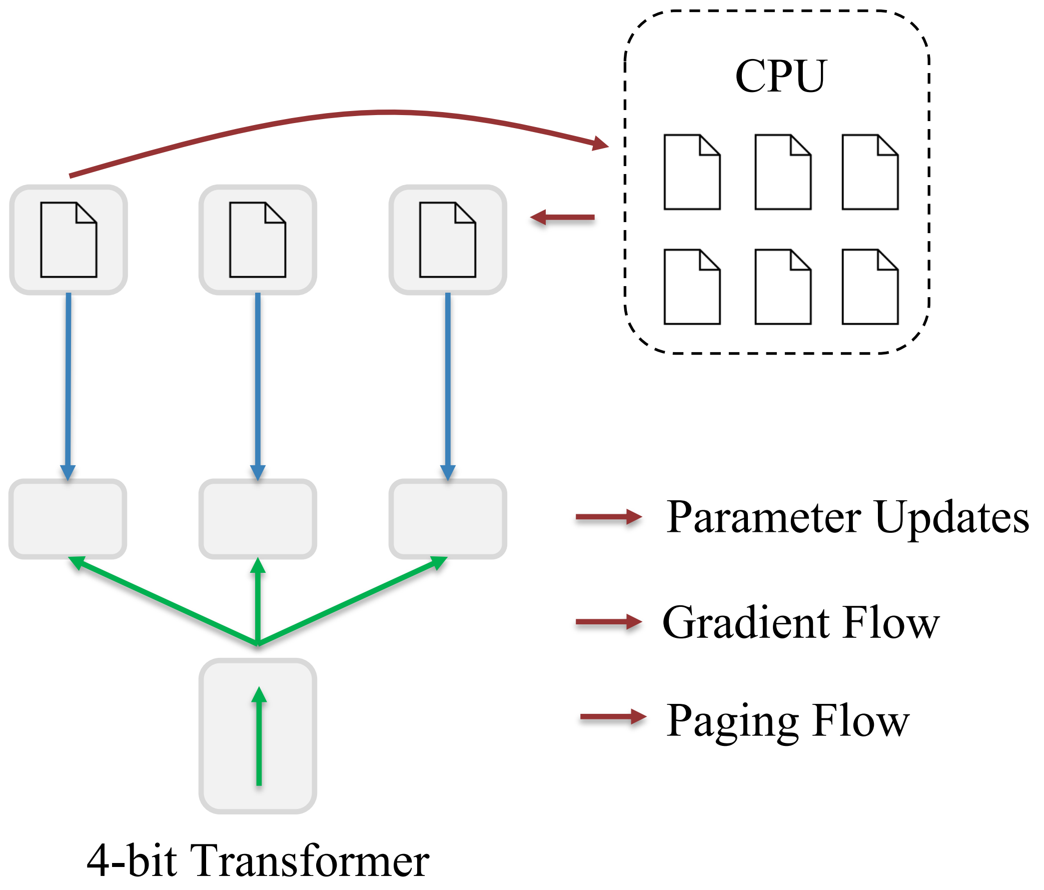


$$\text{dequant}\left(c^{FP32}X^{FP32}\right) = \frac{X^{FP8}}{c^{FP32}} = X^{FP32}$$

Fine-tuning LLMs

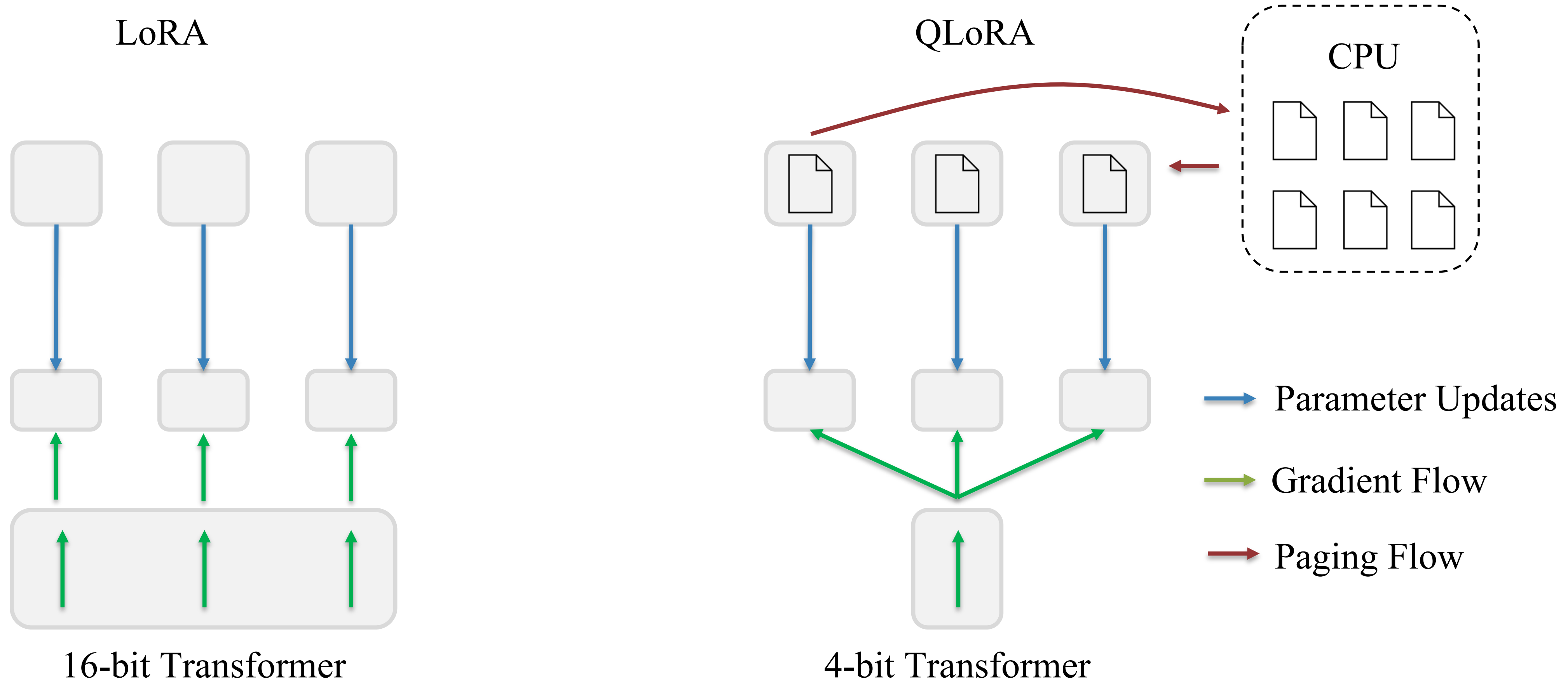
QLoRA: Paged Optimizers

- Page Optimizers to manage memory spikes
- Allocate paged memory for the optimizer states which are then automatically evicted to CPU RAM when the GPU runs out-of-memory and paged back into GPU memory when the memory is needed in the optimizer update step



Fine-tuning LLMs

QLoRA: Efficient Finetuning of Quantized LLMs



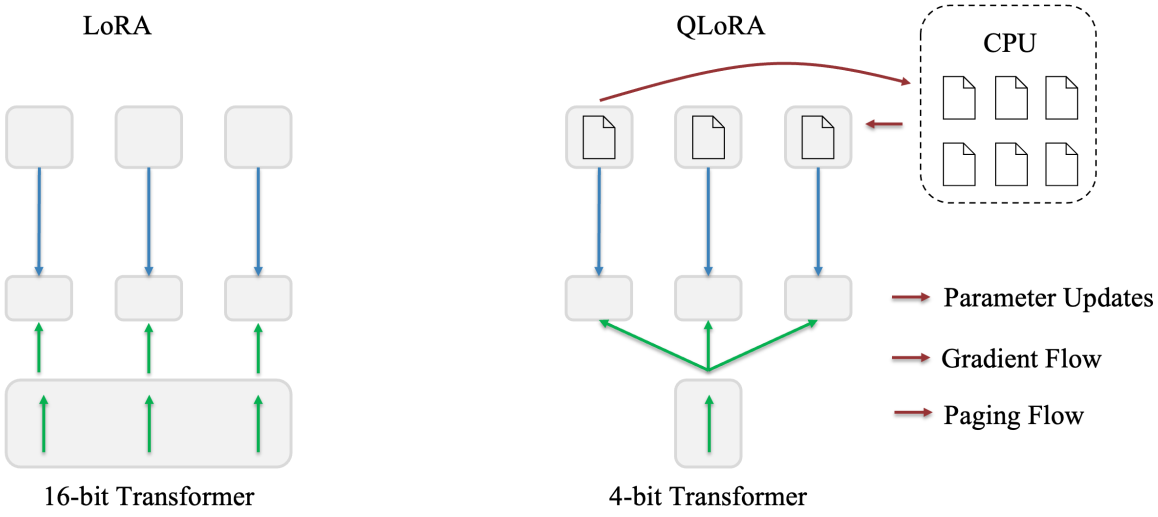
Outline

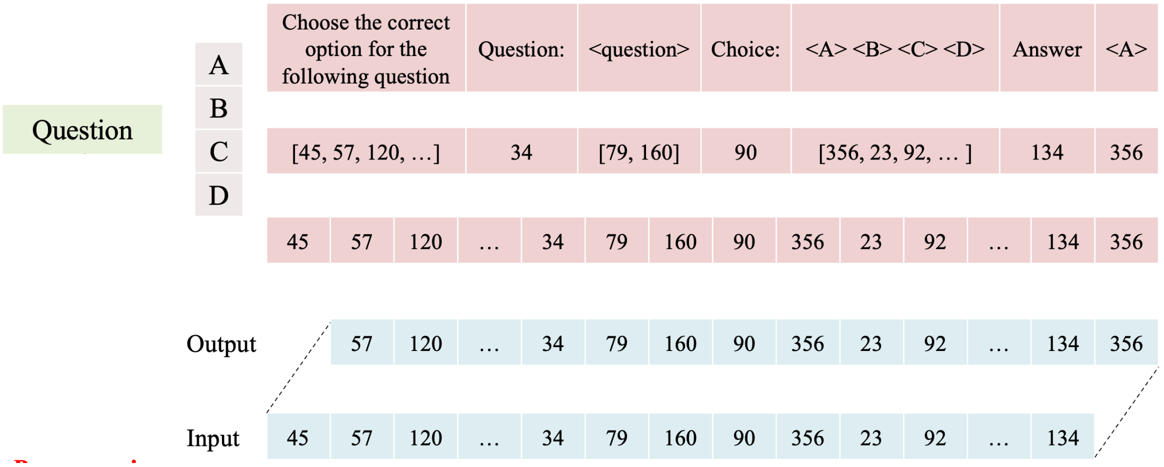
SECTION 1

Fine-tuning LLMs

SECTION 2

Multiple-choice QA





Multiple-choice QA

Multiple-choice QA Dataset

Question

A 40-year-old man has megaloblastic anemia and early signs of neurological abnormality. The drug most probably required is

A	Folic acid
B	Iron sulphate
C	Erythropoietin
D	Vitamin B12

Retrieval Module

Context / Explanation

Deficienty of vitamin B12 results in megaloblastic anemia and demyelination. It can cause subacute combined degeneration of the spinal cord and peripheral neuritis.

Multiple-choice QA

Classification Model

Question

A
B
C
D

Input

CLS	<Question>	<SEP>	<A>	<SEP>	<B>	…	<D>

Vector

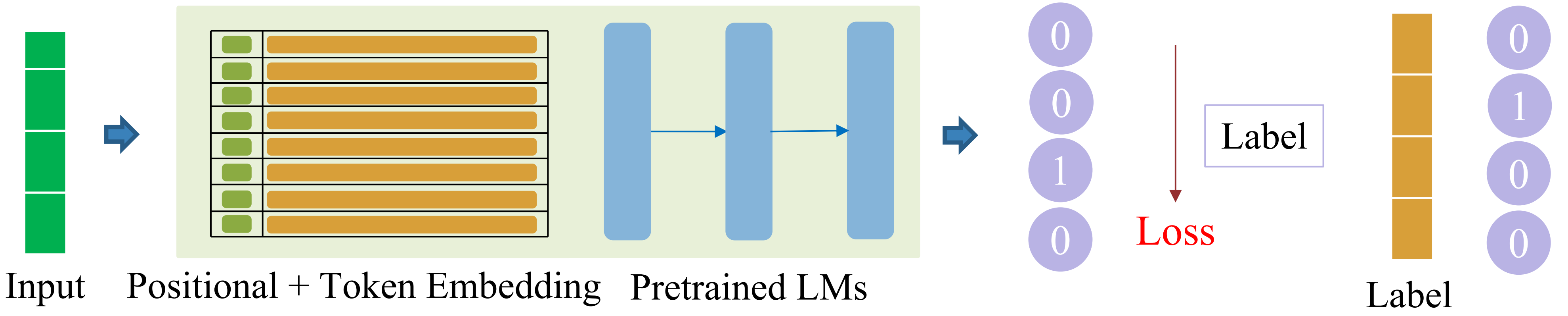
32	…	23	…	23	…	…	8

Classifier

0	A
1	**B**
0	C
0	D

Preprocessing

Training



Multiple-choice QA

Generation Model

Question

A
B
C
D

Input

<Question>	Choices:	<A>	<B>	<C>	<D>	Answer:

Vector

[25, 67,…]	54	…	…	…	…	145

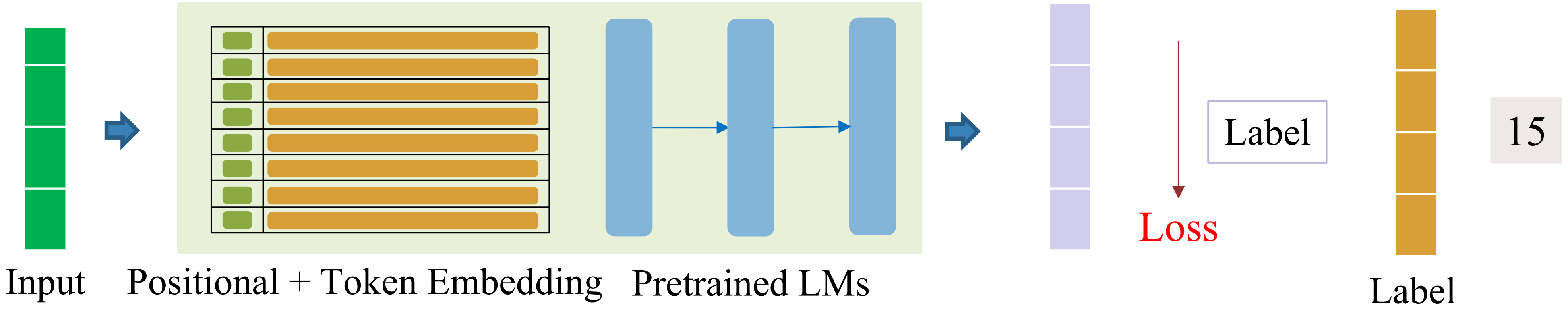
Label

C

15

Preprocessing

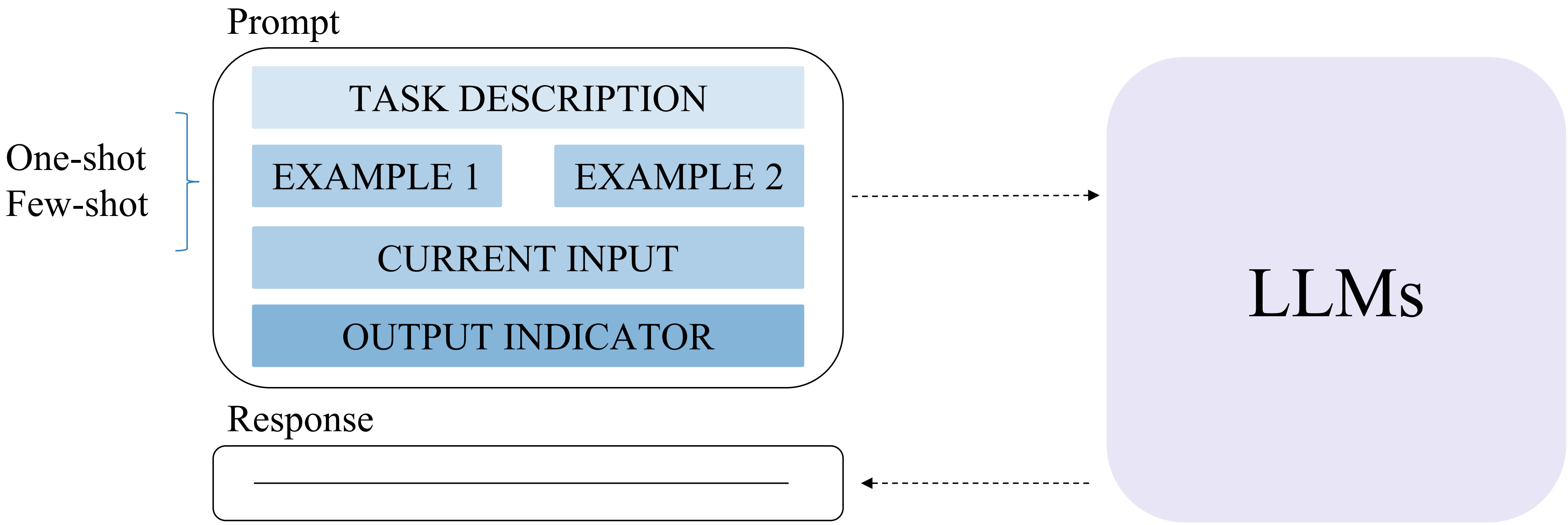
Training



Multiple-choice QA

Prompting Model (LLMs)

- In-context Learning



Multiple-choice QA

Prompting Model (LLMs)

Question

A
B
C
D

Choose the correct option for the following question	Question:	<question>	Choice:	<A> <B> <C> <D>	Answer	<A>
[45, 57, 120, …]	34	[79, 160]	90	[356, 23, 92, …]	134	356

45	57	120	…	34	79	160	90	356	23	92	…	134	356

Output		57	120	…	34	79	160	90	356	23	92	…	134	356
Input	45	57	120	…	34	79	160	90	356	23	92	…	134	

Preprocessing

Multiple-choice QA

Prompting Model (LLMs)

Choose the correct option for the following question	Question:	<question>	Choice:	<A> <B> <C> <D>	Answer	<A>

Question

A
B
C
D

```
### Question:
{}

### Choice:
{}

### Answer:
"""

id2label = {
    0: 'A',
    1: 'B',
    2: 'C',
    3: 'D'
}
```

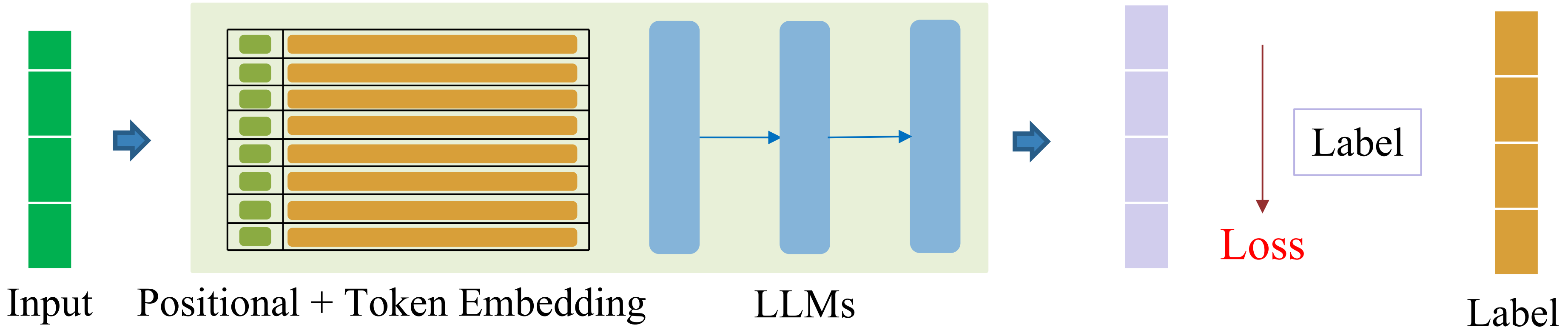
```
from datasets import load_dataset

ds = load_dataset("openlifescienceai/medmcqa")
del ds["test"]
```

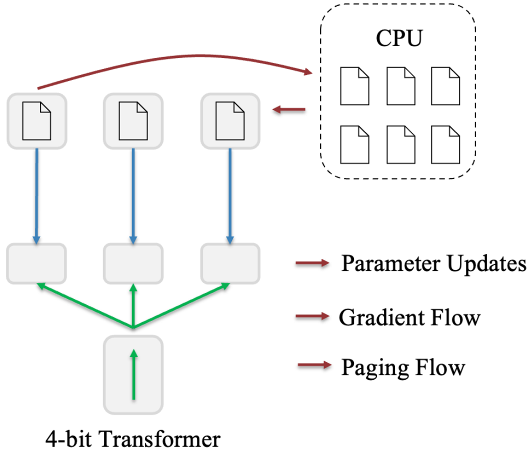
```
data_prompt = """Choose the correct option for the following question.
```

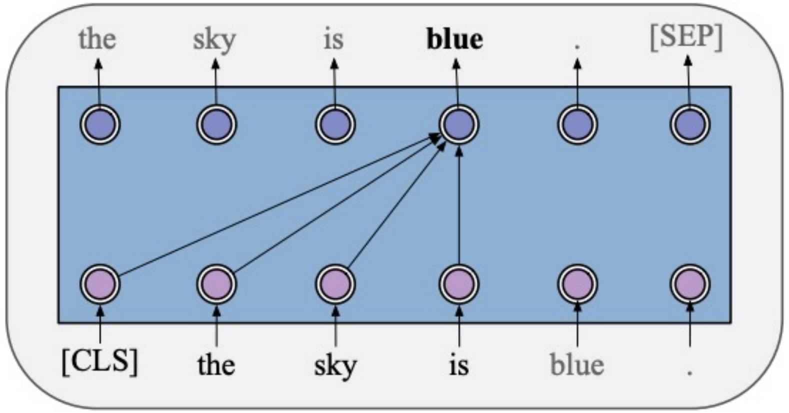
Multiple-choice QA

Prompting Model (LLMs)

Training

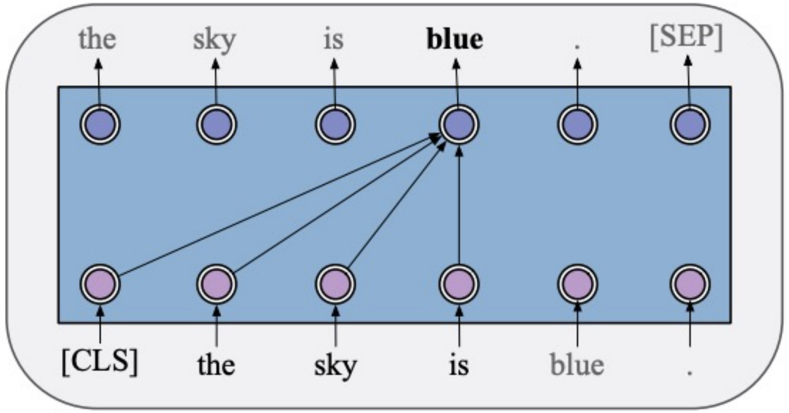


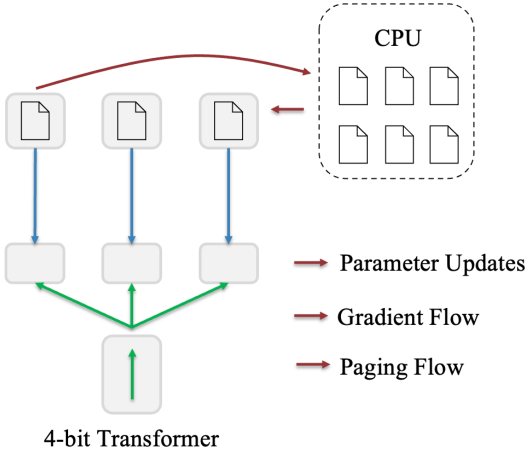




Multiple-choice QA

Prompting Model (LLMs)






```python
from unsloth import FastLanguageModel

max_seq_length = 2048
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/Llama-3.2-1B-bnb-4bit",
    max_seq_length=max_seq_length,
    load_in_4bit=True,
    dtype=None,
)

model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    lora_alpha=16,
    lora_dropout=0,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "up_proj",
        "down_proj", "o_proj", "gate_proj"],
    use_rslora=True,
    use_gradient_checkpointing="unsloth",
    random_state = 42,
    loftq_config = None,
)
print(model.print_trainable_parameters())
```

AI VIET NAM
@aivietnam.edu.vn

Multiple-choice QA

Prompting Model (LLMs)

```
trainer=SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    args=args,
    train_dataset=process_ds["train"],
    eval_dataset=process_ds["validation"],
    dataset_text_field="text",
)
```

```
args = TrainingArguments(
        output_dir="med-mcqa-llama-3.2-1B-4bit-lora",
        logging_dir="logs",
        learning_rate=3e-4,
        lr_scheduler_type="linear",
        per_device_train_batch_size=64,
        gradient_accumulation_steps=16,
        num_train_epochs=2,
        eval_strategy="steps",
        save_strategy="steps",
        logging_strategy="steps",
        eval_steps=50,
        save_steps=50,
        logging_steps=50,
        save_total_limit=1,
        load_best_model_at_end=True,
        fp16=not is_bfloat16_supported(),
        bf16=is_bfloat16_supported(),
        optim="adamw_8bit",
        weight_decay=0.01,
        warmup_steps=10,
        seed=0,
    ),
```

AI VIET NAM
@aivietnam.edu.vn

Multiple-choice QA

Deployment

- Github - Demo

Multiple-choice Question Answering

Model: LLaMA-3.2-1B. Dataset: MedMCQA

Prompt:

"Choose the correct option for the following question. ### Question:Which of the following is not true f

"Choose the correct option for the following question.

```
### Question:
```

Which of the following is not true for myelinated nerve fibers:

```
### Choice:

A. Impulse through myelinated fibers is slower than non-myelinated fibers.
B. Membrane currents are generated at nodes of Ranvier.
C. Saltatory conduction of impulses is seen.
D. Local anesthesia is effective only when the nerve is not covered by myelin shea


### Answer:
```

D Local anesthesia is effective only when the nerve is not covered by myelin sheath

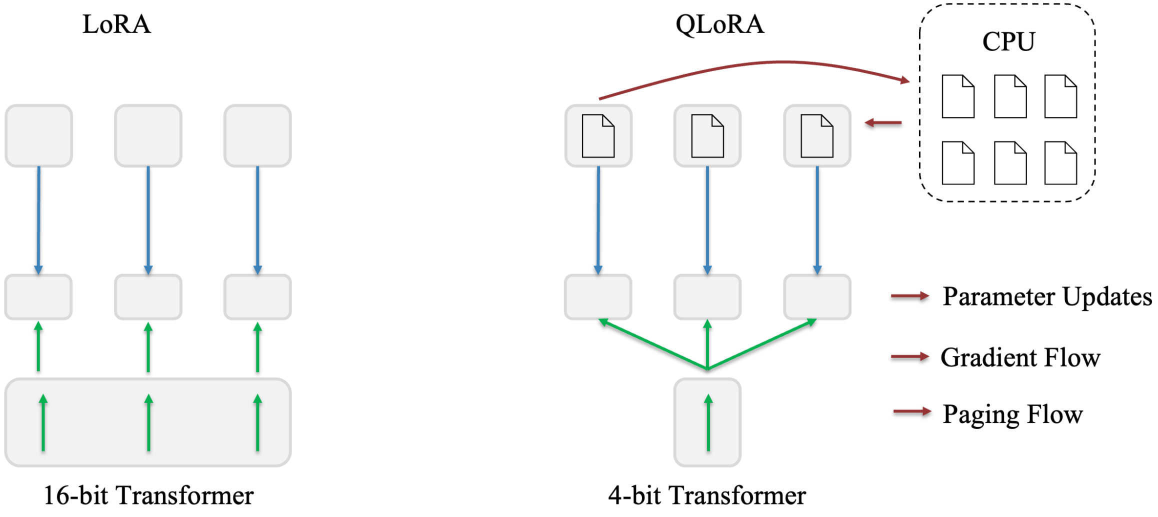
Objectives

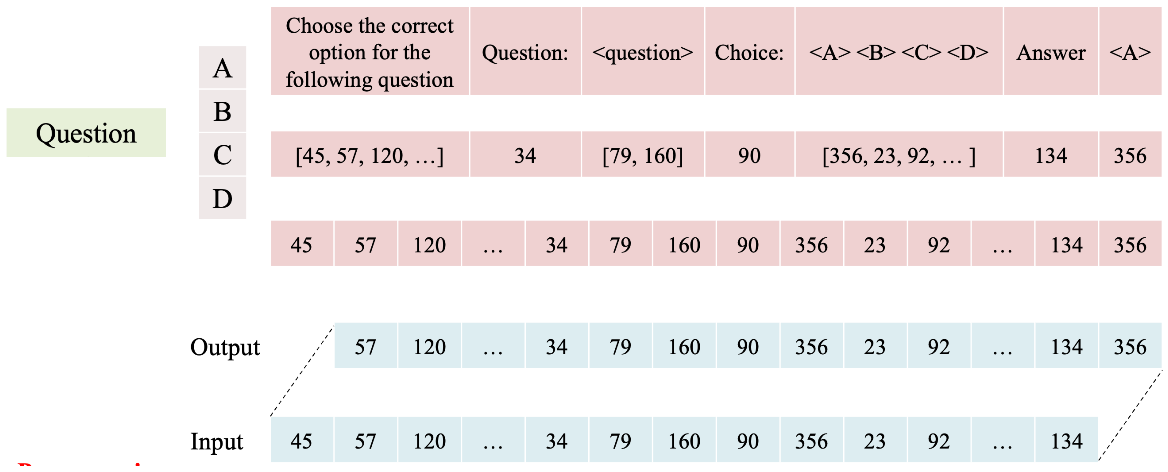
PEFT

- Fine-tuning LLMs
- Adapter / Prefix / Prompt Tuning
- LoRA, QLoRA

Multiple-choice QA

- Multiple-choice Question Answering
- MedMCQA Dataset
- Fine-tuning LLaMA-3.2-1B





Thanks!

Any questions?