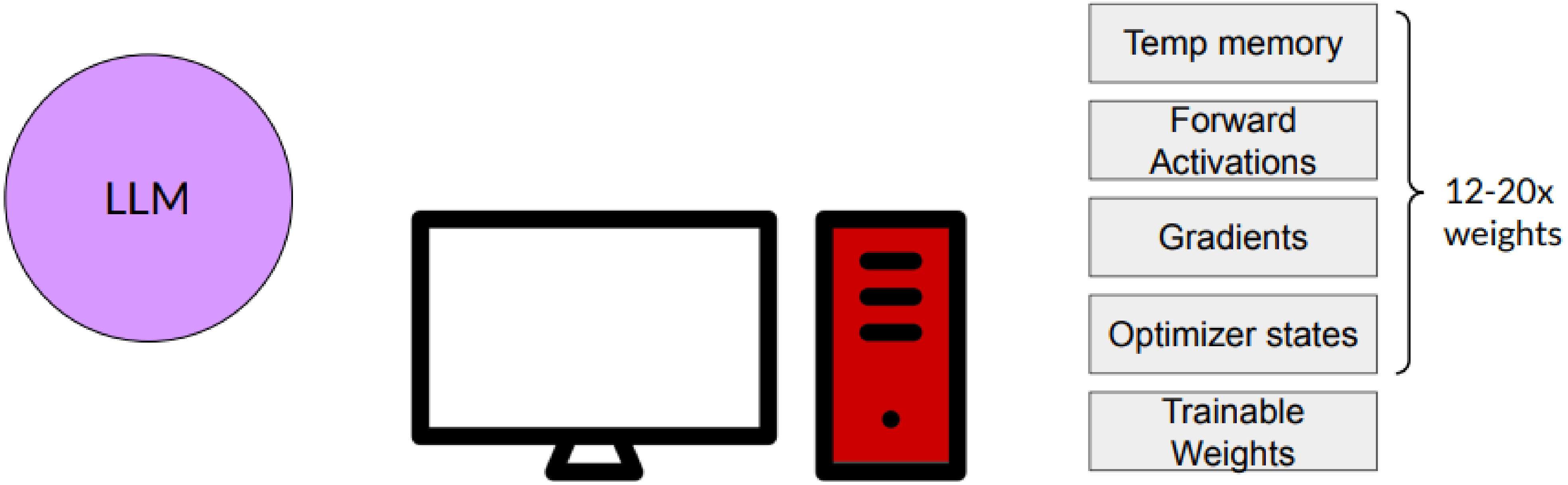
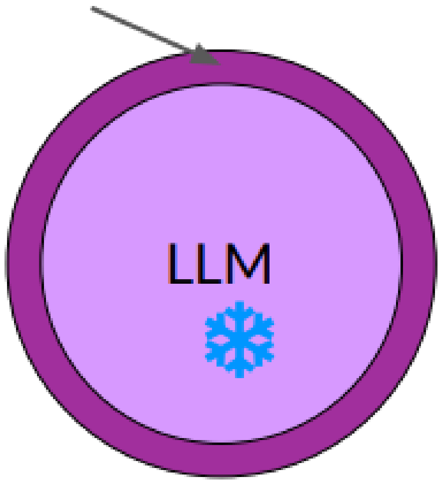
Parameter-efficient Fine-tuning (PEFT)

Full fine-tuning of large LLMs is challenging

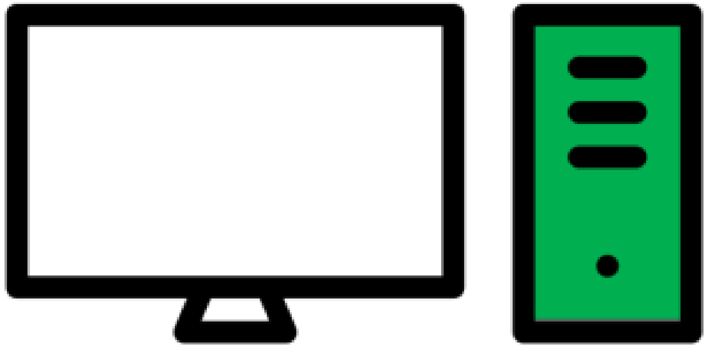


Parameter efficient fine-tuning (PEFT)

New trainable layers



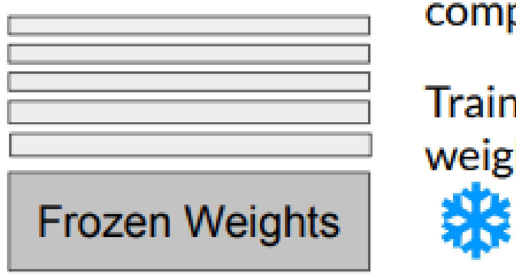
LLM with additional layers for PEFT



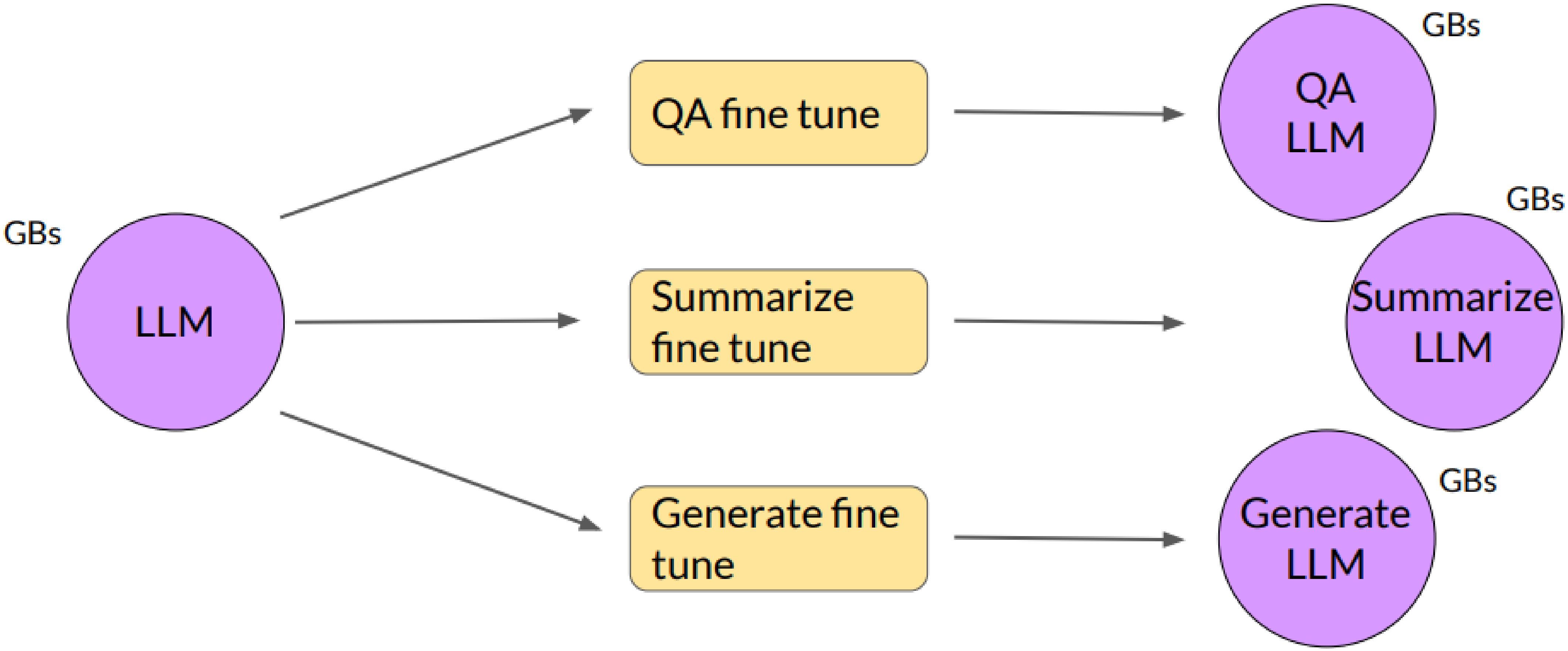
Less prone to catastrophic forgetting

Other components

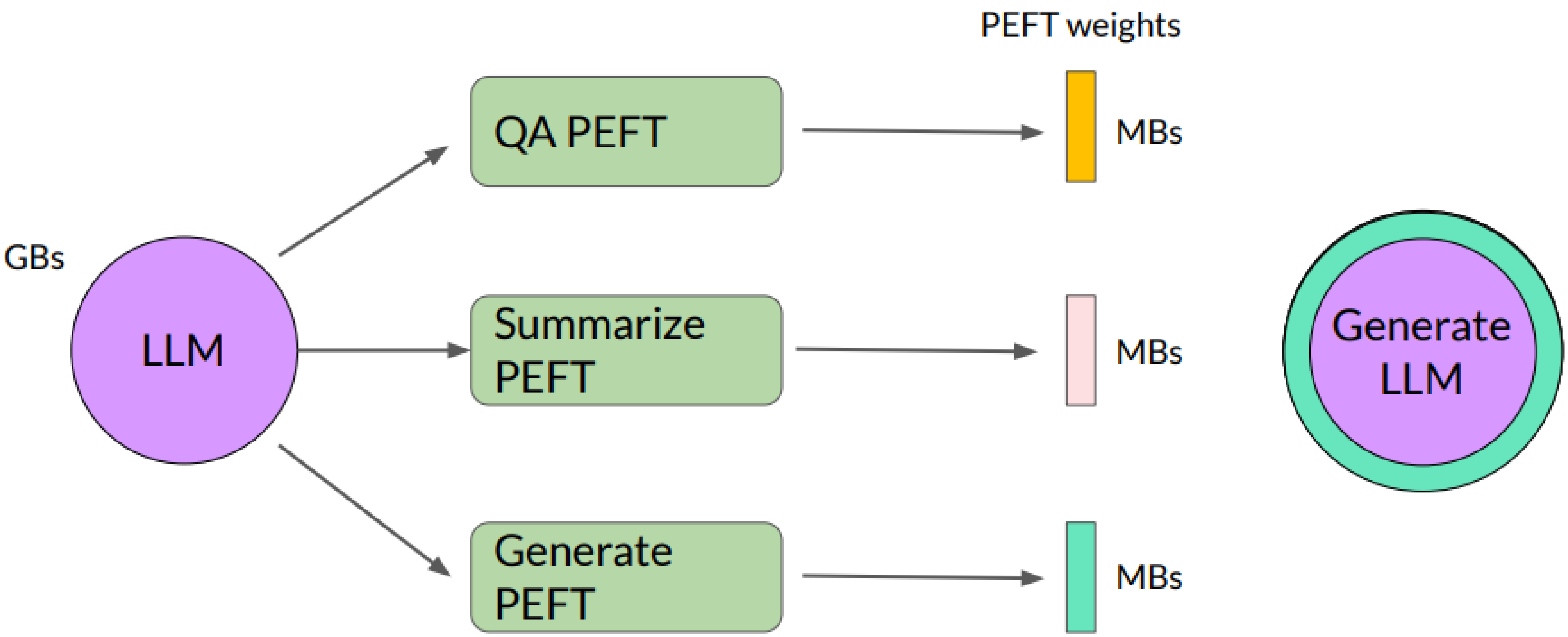
Trainable weights



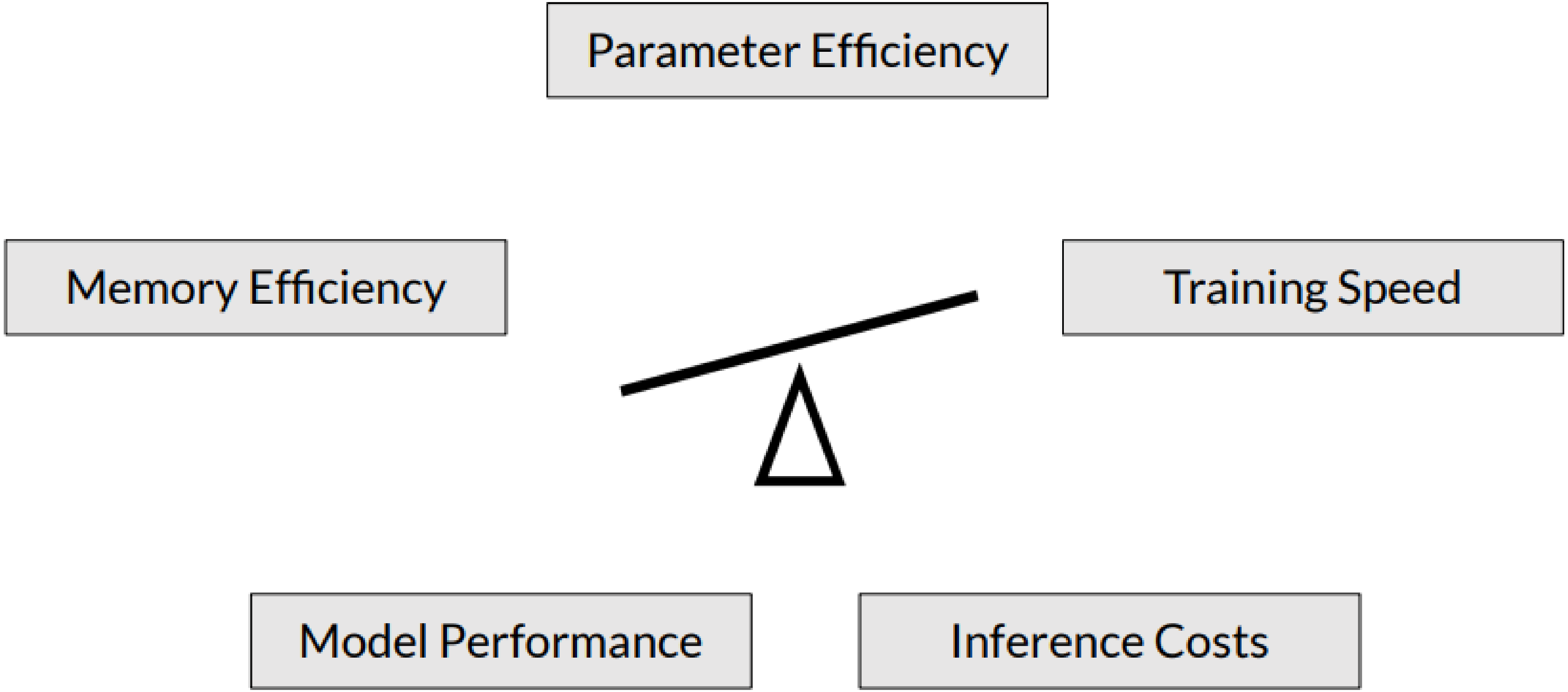
Full fine-tuning creates full copy of original LLM per task



PEFT fine-tuning saves space and is flexible



PEFT Trade-offs



PEFT methods

Selective

Select subset of initial LLM parameters to fine-tune

Reparameterization

Reparameterize model weights using a low-rank representation

LoRA

Additive

Add trainable layers or parameters to model

Adapters

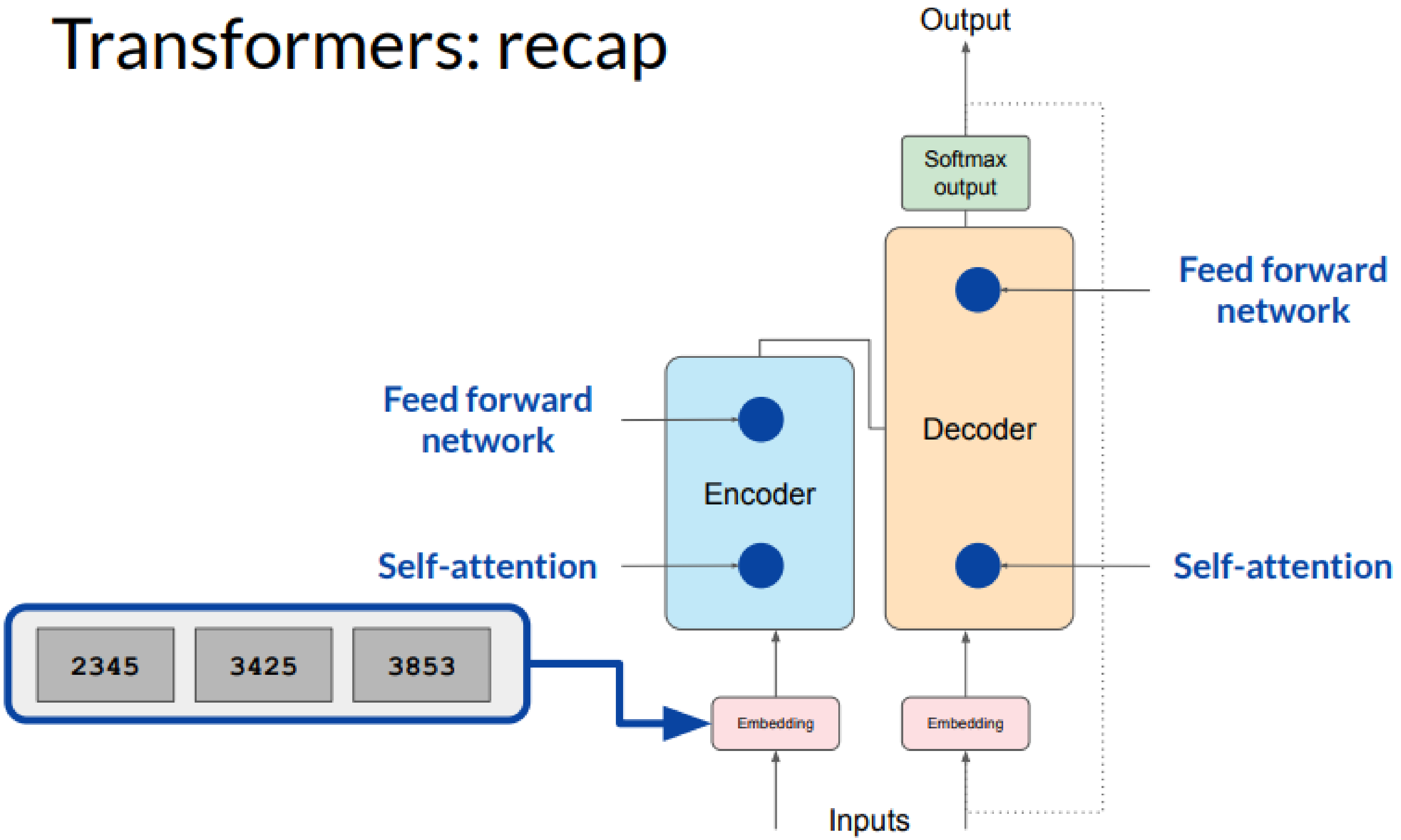
Soft Prompts

Prompt Tuning

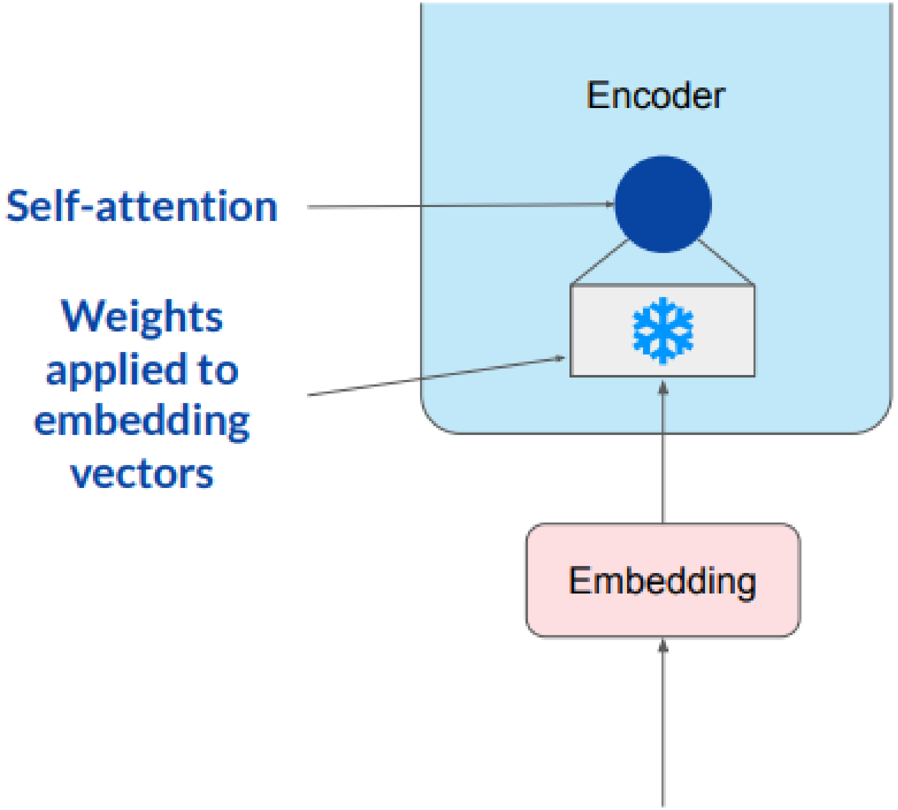
Source: Lialin et al. 2023, "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning",

Low-Rank Adaptation of Large Language Models (LoRA)

Transformers: recap

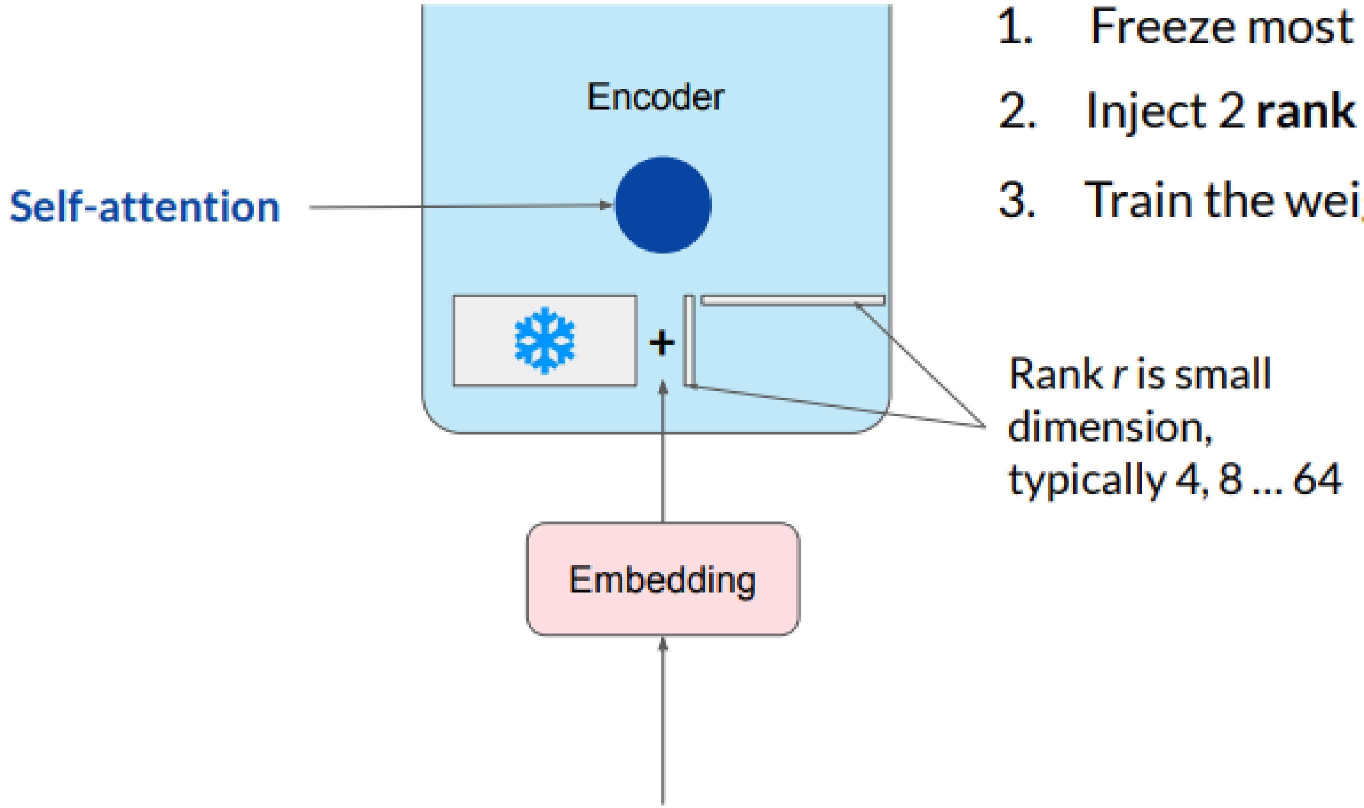


LoRA: Low Rank Adaption of LLMs



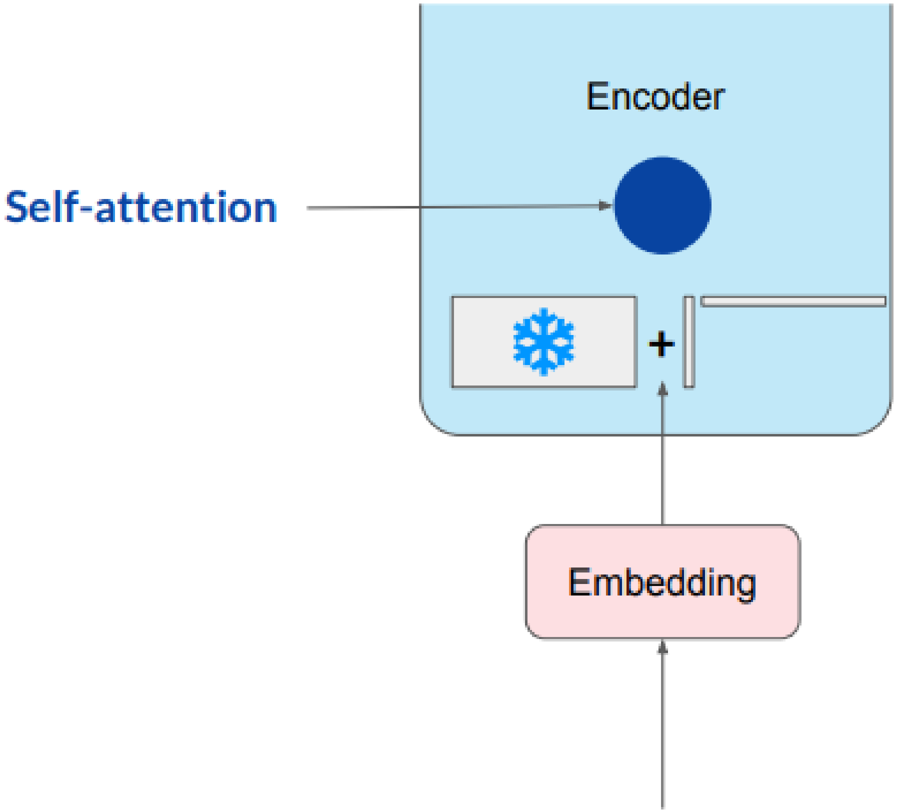
1. Freeze most of the original LLM weights.

LoRA: Low Rank Adaption of LLMs



1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

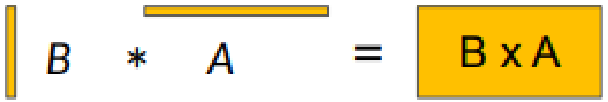
LoRA: Low Rank Adaption of LLMs



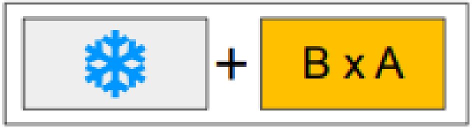
1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

Steps to update model for inference

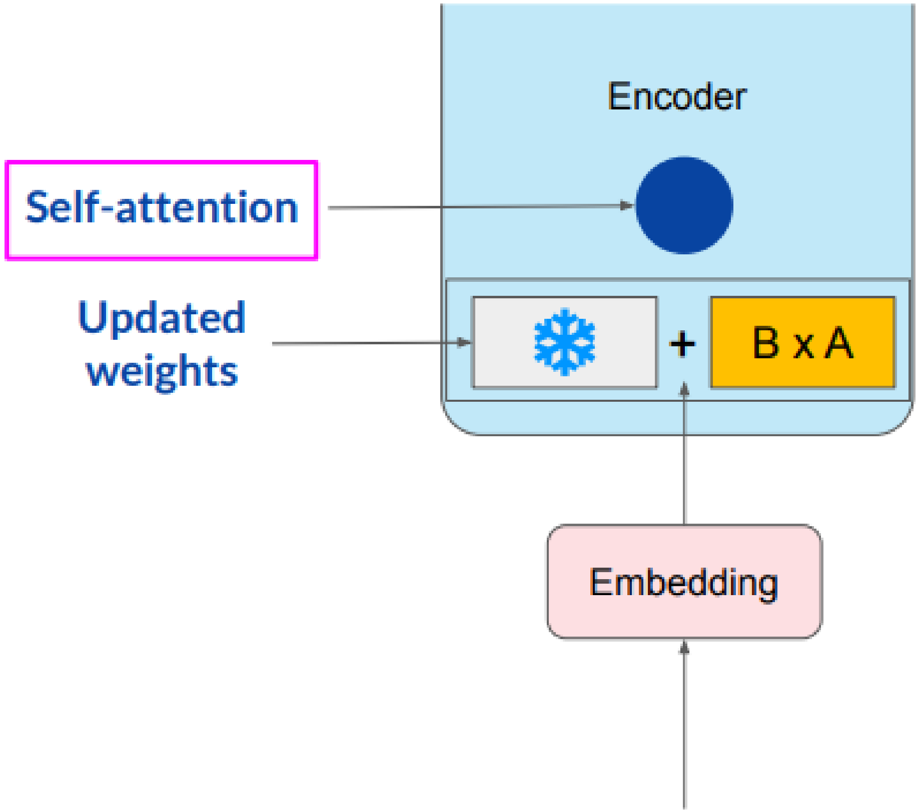
1. Matrix multiply the low rank matrices



2. Add to original weights



LoRA: Low Rank Adaption of LLMs



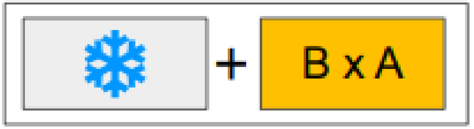
1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

Steps to update model for inference:

1. Matrix multiply the low rank matrices

$$B * A = B \times A$$

2. Add to original weights



Concrete example using base Transformer as reference

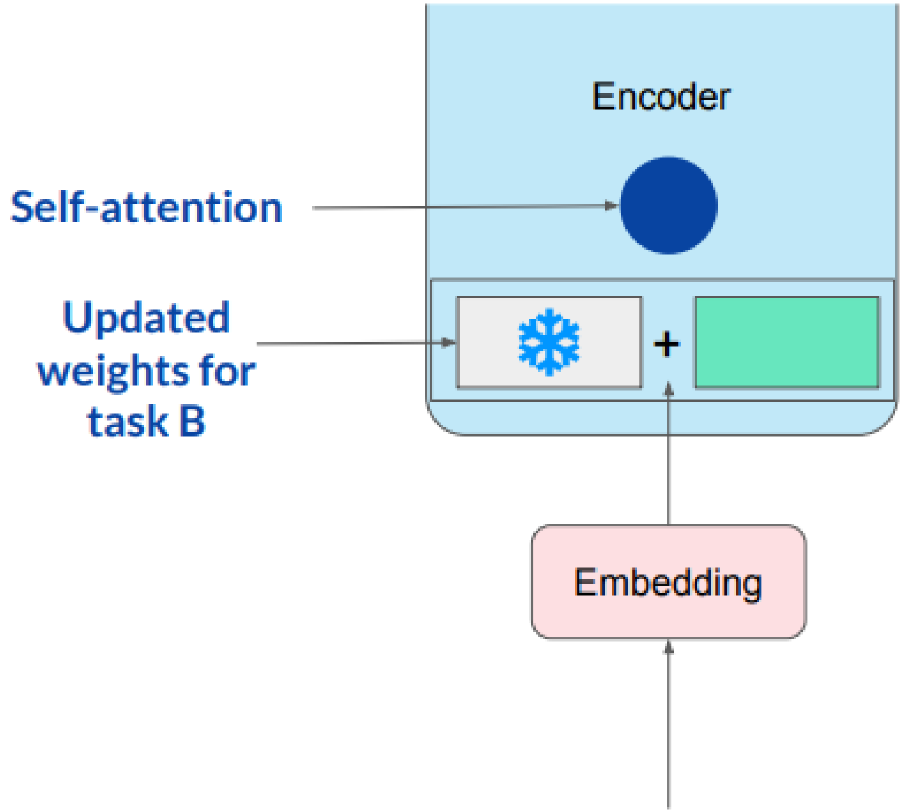
Use the base Transformer model presented by Vaswani et al. 2017:

- Transformer weights have dimensions $d \times k = 512 \times 64$
- So 512 x 64 = 32,768 trainable parameters

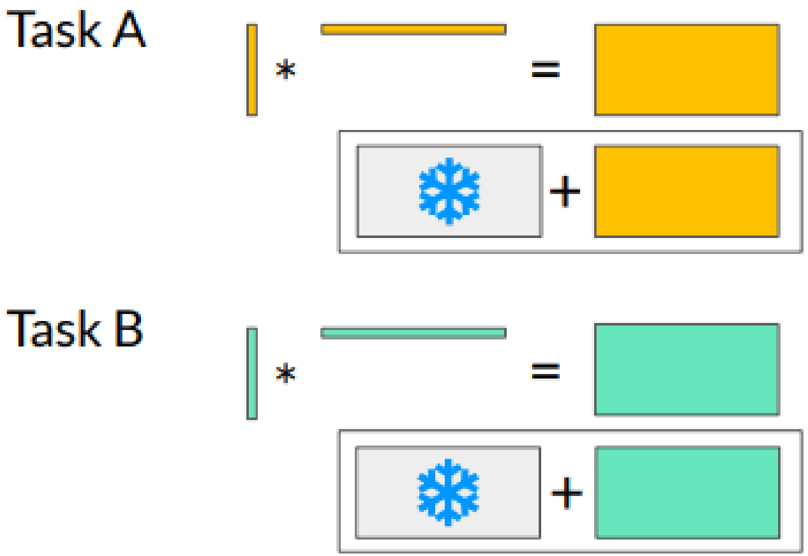
In LoRA with rank $r = 8$:

- A has dimensions $r \times k = 8 \times 64 = 512$ parameters
- B has dimension $d \times r = 512 \times 8 = 4{,}096$ trainable parameters
- **86% reduction in parameters to train!**

LoRA: Low Rank Adaption of LLMs



1. Train different rank decomposition matrices for different tasks
2. Update weights before inference



Sample ROUGE metrics for full vs. LoRA fine-tuning

Base model ROUGE

Full fine-tune ROUGE

FLAN-T5

Dialog summarization

```
flan_t5_base
{'rouge1': 0.2334,
 'rouge2': 0.0760,
 'rougeL': 0.2014,
 'rougeLsum': 0.2015}
```

Baseline score

Sample ROUGE metrics for full vs. LoRA fine-tuning

FLAN-T5

Dialog summarization

Base model ROUGE

```
flan_t5_base
{'rouge1': 0.2334,
 'rouge2': 0.0760,
 'rougeL': 0.2014,
 'rougeLsum': 0.2015}
```

Baseline score

+80.63%

Full fine-tune ROUGE

```
flan_t5_base_instruct_full
{'rouge1': 0.4216,
 'rouge2': 0.1804,
 'rougeL': 0.3384,
 'rougeLsum': 0.3384}
```

-3.20%

LoRA fine tune ROUGE

```
flan_t5_base_instruct_lora
{'rouge1': 0.4081,
 'rouge2': 0.1633,
 'rougeL': 0.3251,
 'rougeLsum': 0.3249}
```

Choosing the LoRA rank

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

- Effectiveness of higher rank appears to plateau
- Relationship between rank and dataset size needs more empirical data

Source: Hu et al. 2021, "LoRA: Low-Rank Adaptation of Large Language Models"

QLoRA: Quantized LoRA

- Introduces 4-bit NormalFloat (nf4) data type for 4-bit quantization
- Supports double-quantization to reduce memory ~0.4 bits per parameter (~3 GB for a 65B model)
- Unified GPU-CPU memory management reduces GPU memory usage
- LoRA adapters at every layer - not just attention layers
- Minimizes accuracy trade-off

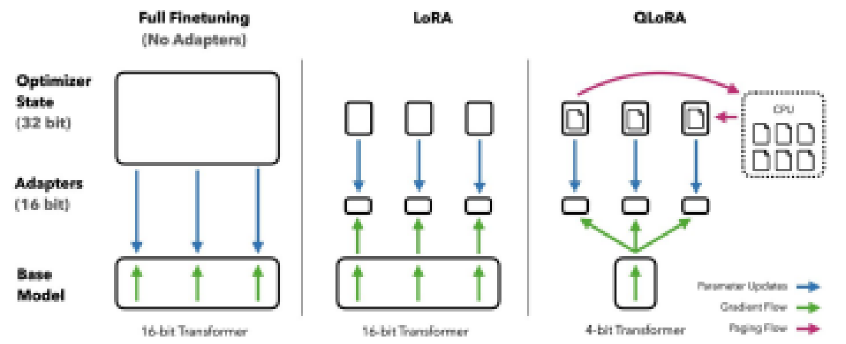


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

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

PEFT methods summary

Selective

Select subset of initial LLM parameters to fine-tune

Reparameterization

Reparameterize model weights using a low-rank representation

LoRA

Additive

Add trainable layers or parameters to model

Adapters

Soft Prompts

Prompt Tuning