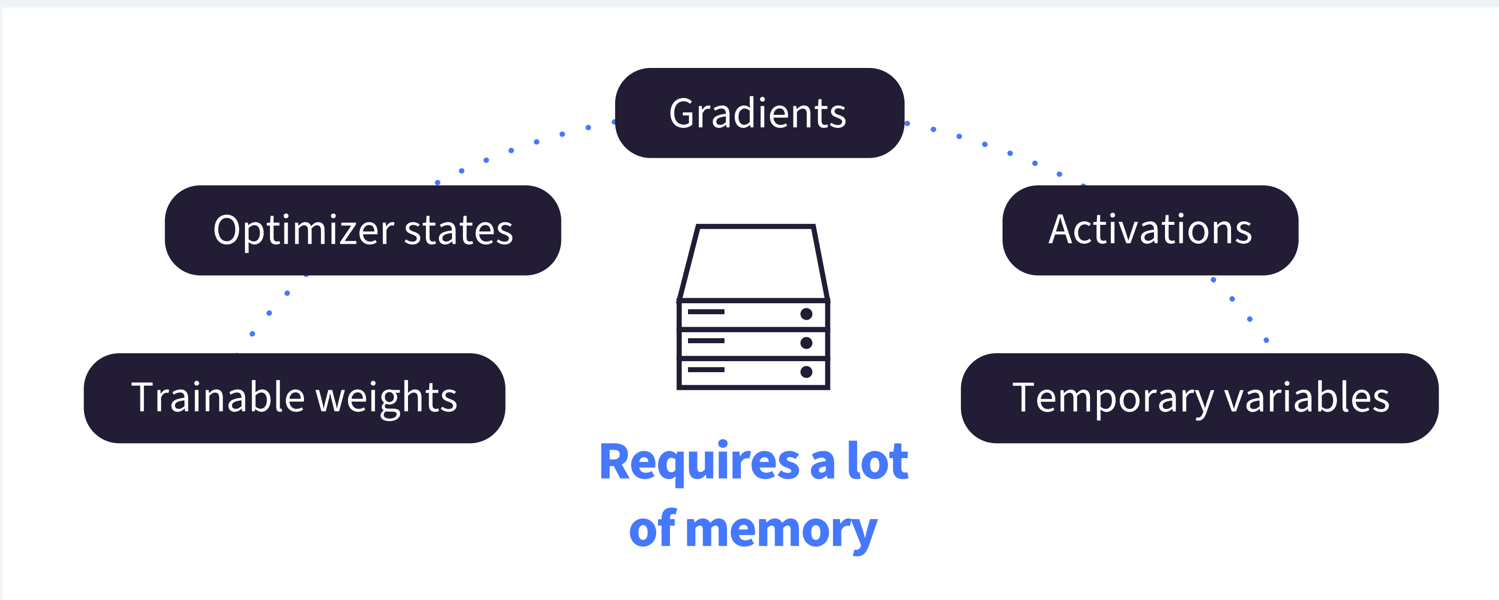
Parameter Efficient Fine-Tuning (PEFT) Methods

PEFT

Full fine-tuning of LLMs is challenging:



PEFT methods **only update a small number of model parameters.**
Examples of PEFT techniques:

- Freeze most model weights, and **fine tune only specific layer parameters.**
- Keep existing parameters untouched; **add only a few new ones or layers** for fine-tuning.

→ The trained parameters can account for only 15%-20% of the original LLM weights.

Main benefits:

- Decrease memory usage, often requiring just 1 GPU.
- Mitigate risk of catastrophic forgetting.
- Limit storage to only the new PEFT weights.

Multiple methods exist with trade-offs on parameters or memory efficiency, training speed, model quality, and inference costs.
Three PEFT methods classes from literature:

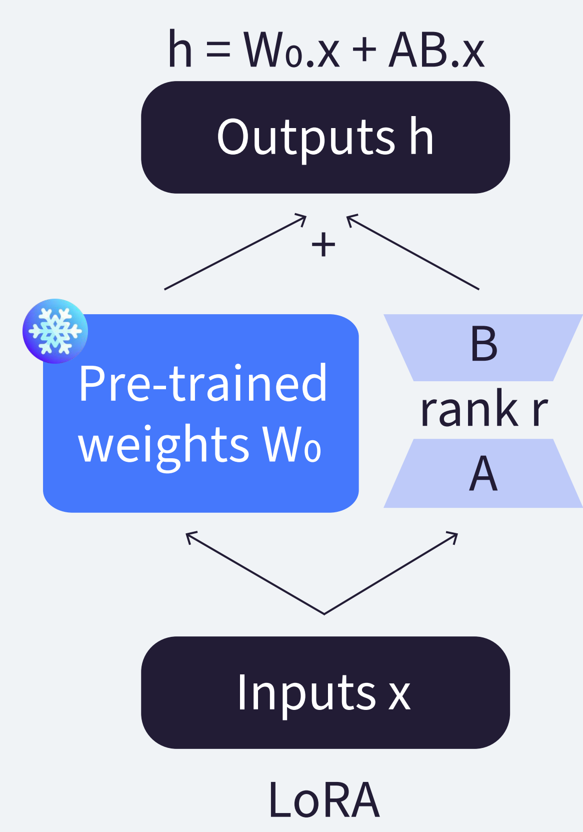
Selective
Fine-tune only specific parts of the original LLM.

Reparameterization
Use low-rank representations to reduce the number of trainable parameters.
E.g., LoRA

Additive
Augment the pre-trained model with new parameters or layers, training only the additions.
→ Adapter
→ Soft prompts

LoRA

Method to reduce the number of trainable parameters during fine-tuning **by freezing all original model parameters** and injecting a **pair of rank decomposition matrices** alongside the original weights



1 - Keep the majority of the original LLM weights frozen.
2 - Introduce a pair of **rank decomposition matrices.**
3 - Train the new matrices A and B.

Model weights update:

1 - Matrix multiplication:

B * A = BxA

2 - Add to original weights :

+ BxA

Additional notes:

- No impact on inference latency.
- Fine-tuning specifically on the **self-attention layers** using LoRA is often enough to enhance performance for a given task.
- Weights can be switched out as needed, allowing for training on **many different tasks.**

Rank Choice for LoRA Matrices:

Trade-Off: A smaller rank reduces parameters and accelerates training **but** risks lower adaptation quality due to reduced task-specific information capture.

*In literature, it appears that a **rank between 4-32** is a good trade-off.*

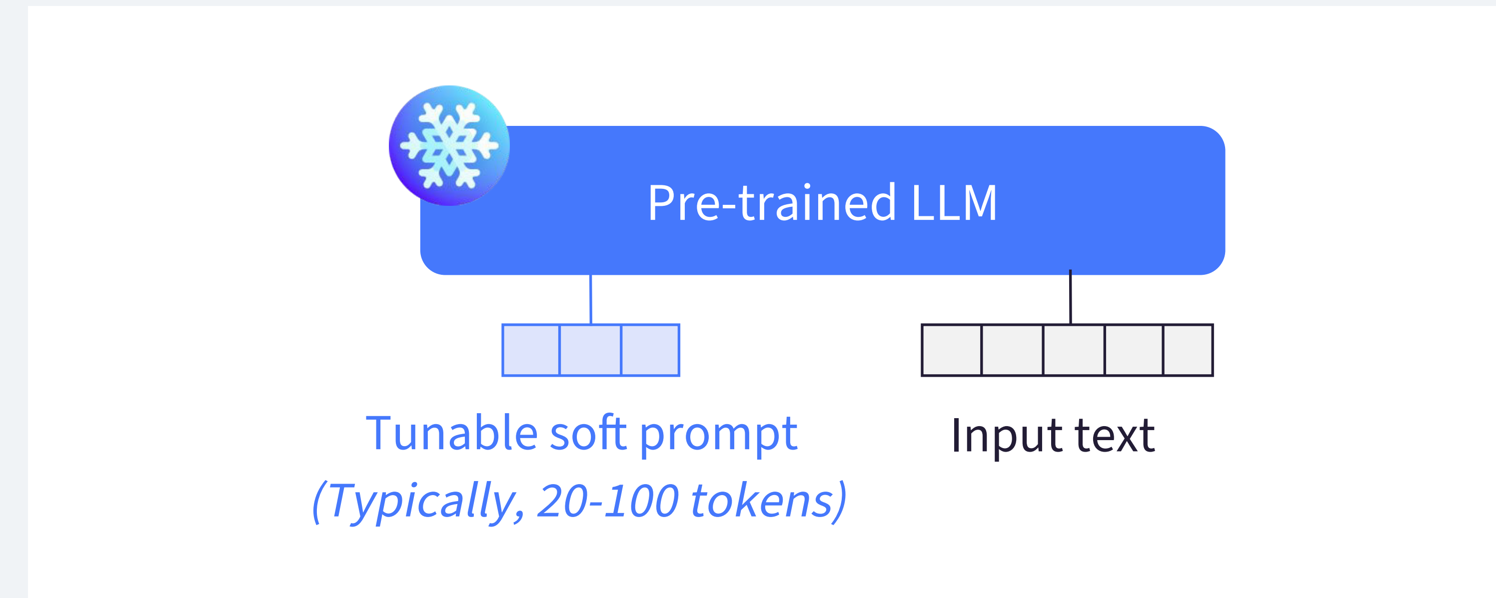
LoRA can be combined with quantization (=QLoRA).

SOFT PROMPTS

Unlike prompt engineering, whose limits are:

- The manual effort requirements
- The length of the context window

Prompt tuning: Add trainable tensors to the model input embeddings, commonly known as "soft prompts," optimized directly through gradient descent.



Soft prompt vectors:

- Equal in length to the embedding vectors of the input language tokens
- Can be seen as **virtual tokens** which can take any value within the multidimensional embedding space

In prompt tuning, LLM weights are frozen:

- Over time, the embedding vector of the soft prompt is adjusted to optimize model's completion of the prompt
- Only **few parameters are updated**
- A different set of soft prompts can be trained for each task and easily swapped out during inference (occupying very little space on disk).

From literature, it is shown that at 10B parameters, prompt tuning is as efficient as full fine-tuning.

⚠ *Interpreting virtual tokens can pose challenges (nearest neighbor tokens to the soft prompt location can be used).*