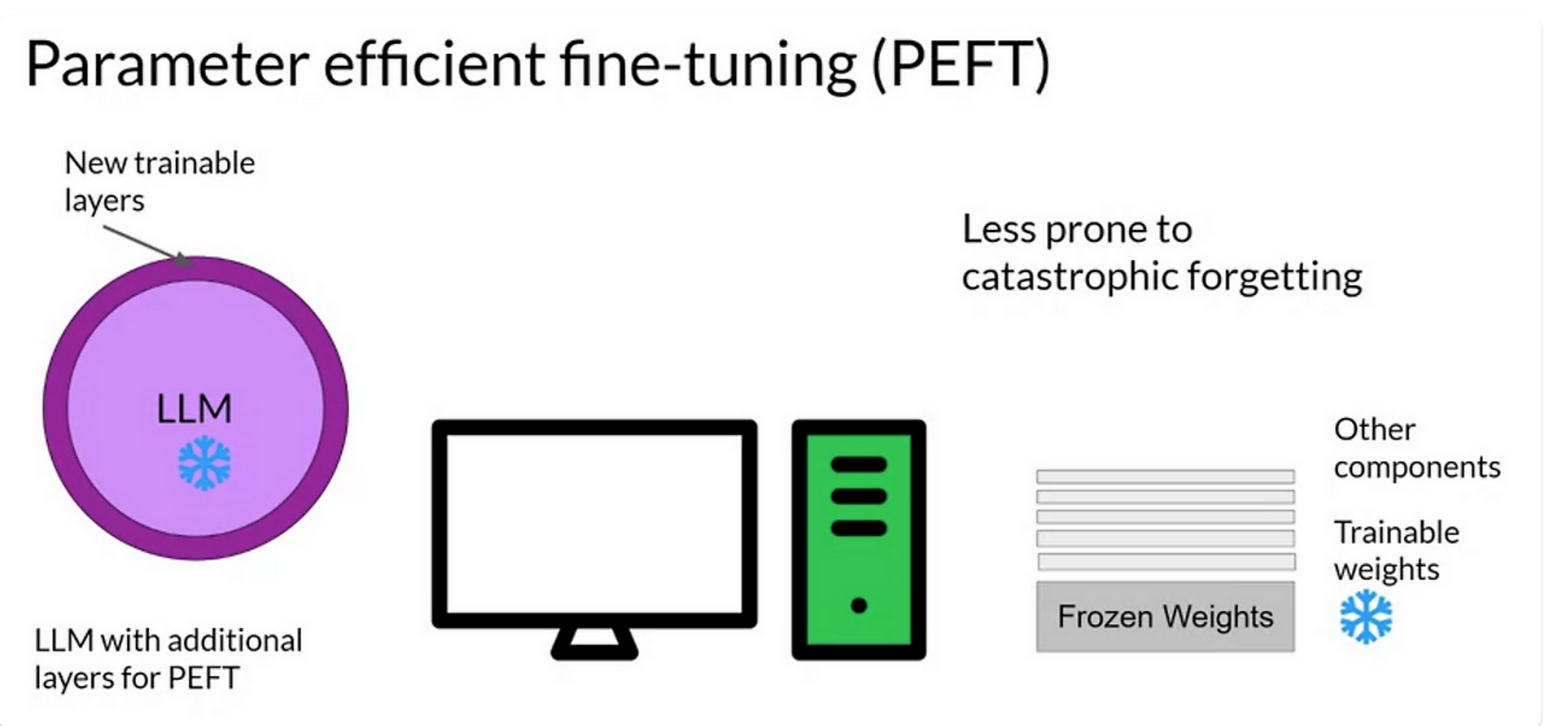
Ref: https://data-newbie.tistory.com/982#Parameter%C2%A0Efficient%C2%A0Fine-tuning(PEFT)%C2%A0techniques-1-5
https://velog.io/@tmdtng21/PEFT-Methods
https://blog-ko.superb-ai.com/lightweighting-llm-with-peft/
https://www.catalyzex.com/paper/low-rank-adaptation-method-for-wav2vec2-based?ref=blog-ko.superb-ai.com
https://data-newbie.tistory.com/982#Parameter%C2%A0Efficient%C2%A0Fine-tuning(PEFT)%C2%A0techniques-1-5
https://blog.naver.com/se2n/223375998670

PEFT (Parameter-Efficient Fine-Tuning) is a technique used to fine-tune large pre-trained language models for specific tasks. It achieves effective learning and performance improvement without significantly increasing the number of model parameters. The core idea of PEFT is **to optimize the performance of existing models by adjusting a very small number of parameters,** rather than altering the entire model architecture or introducing a large number of additional parameters. This approach significantly enhances the efficiency and practicality of the model.



Main types of PEFT

Additive Methods (Adding New Parameters):

- **LoRA (Low-Rank Adaptation):** One of the most widely used PEFT techniques. Instead of directly fine-tuning the pre-trained model's weight matrix W , it approximates the update ΔW by decomposing it into the product of two low-rank matrices, A and B

That is, ΔW=BA. During training, only **A** and B are learned, while W is kept fixed. For inference, W+BA can be used, effectively integrating the learned adaptation into the original model.

Advantages: Learns a very small number of parameters (≈0.01% of total parameters), high performance, almost no inference overhead, applicable to various model architectures.

Disadvantages: May be difficult to apply to some specific architectures.

- **Adapter Tuning:** Inserts small neural network modules (adapters) into specific layers of the model (e.g., between the feed-forward networks of Transformer blocks). Only these adapter modules are trained, and the parameters of the original model remain fixed.
- **Advantages:** Modular structure, multiple adapters can be chained to a single model.
- **Disadvantages:** May slightly increase inference latency (due to passing through additional layers).

Partial Methods (Selecting and Adjusting Subsets of Existing Parameters):

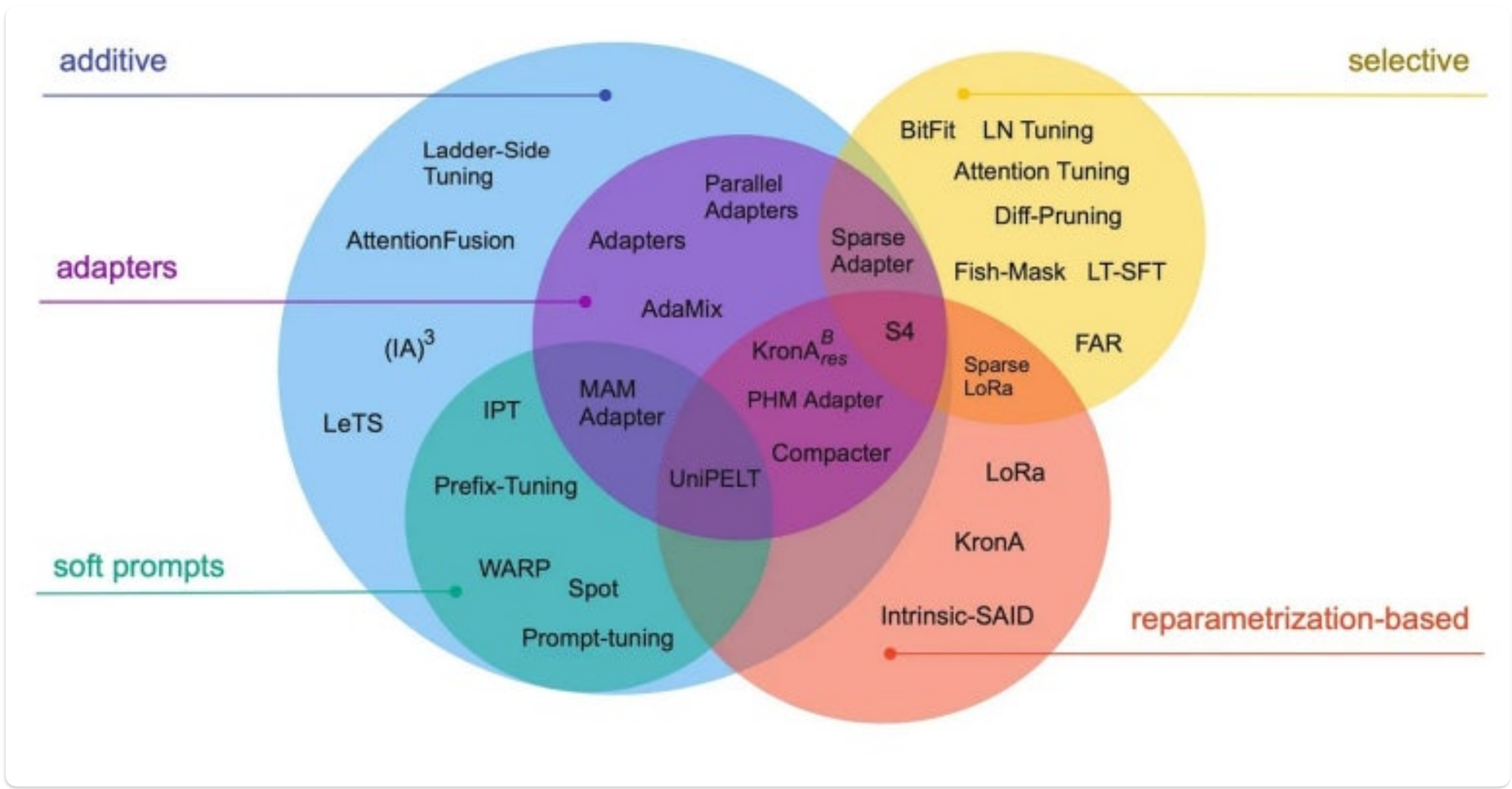
- **Prefix-Tuning:** Appends learnable "prefix" vectors to the input sequence of the model. Only these prefix vectors are trained, and the rest of the model's parameters are frozen. The prefix acts as a "soft prompt" to manipulate the model's behavior.

- **Advantages:** Learns a very small number of parameters, proposed earlier than LoRA.
- **Disadvantages:** Generally performs slightly worse than LoRA, or may incur a slight overhead during inference (processing additional tokens).
-
- **Prompt-Tuning:** Similar to Prefix-Tuning, it adds learnable "virtual tokens" to the input sequence. Only the embeddings of these virtual tokens are trained, and the model's parameters are frozen.

- **Advantages:** Learns the fewest parameters, allows for different prompts per task.
- **Disadvantages:** Tends to perform better with larger models; effectiveness might be limited with smaller models.

Reparameterization Methods (Reparameterizing Existing Parameters):

- **BitFit:** Fine-tunes only specific parameters of the model, primarily bias parameters. All weight matrices of the model are fixed, and only the bias vectors are trained.
- **Advantages:** Learns a very small number of parameters, simple to implement.
- **Disadvantages:** May yield lower performance than other PEFT techniques (due to limited flexibility).
-
2



Advantages of PEFT

- **Reduced Computational Resources:** Since only a small number of parameters need to be trained instead of the entire model, it significantly reduces GPU memory, CPU usage, and power consumption.
- **Storage Efficiency:** Only the fine-tuned PEFT modules (e.g., LoRA adapters) for each task need to be stored, which is far more efficient than storing full copies of the entire model. This is especially crucial when applying LLMs to hundreds or thousands of downstream tasks.
- **Faster Training Speed:** The fine-tuning process is much quicker because fewer parameters need to be learned.
- **Maintained or Improved Performance:** Surprisingly, PEFT techniques often achieve performance comparable to, or even better than, full fine-tuning. This can be attributed to a reduction in overfitting and an improvement in the model's generalization capabilities.
- **Easier Deployment and Management:** Only one base model needs to be maintained, and various PEFT modules can be loaded as needed. This is useful for A/B testing or model serving.
- **Environmentally Friendly:** Reduced energy consumption contributes to lowering the carbon footprint.

Limitations and Considerations of PEFT

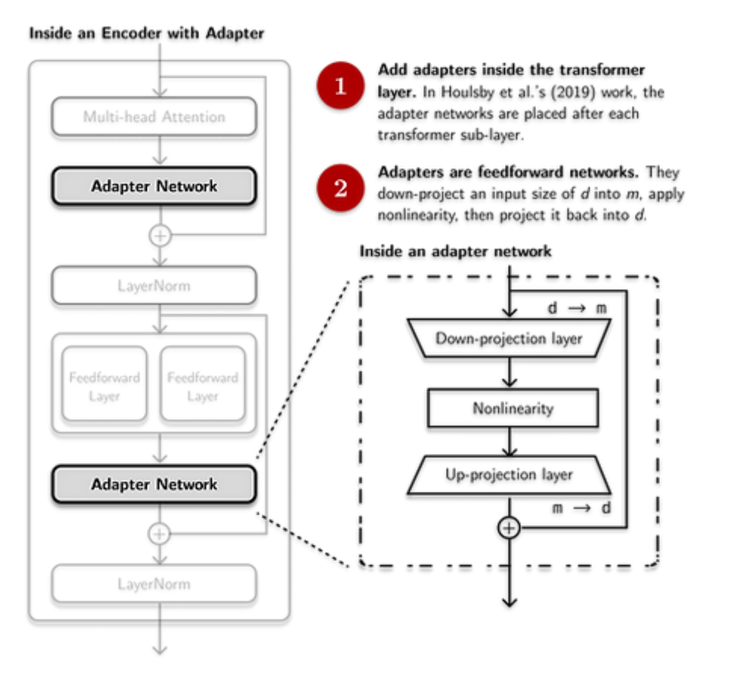
- **Optimal Technique Selection:** It's difficult to definitively say that one PEFT technique is best for all tasks and models. The most suitable technique should be chosen based on the specific characteristics of the task, the size of the model, and available resources.
- **Hyperparameter Tuning:** PEFT techniques themselves have additional hyperparameters, such as the rank (r in LoRA), adapter size, and prefix length. Properly tuning these is crucial.
- **Performance Ceiling:** For certain complex tasks, full fine-tuning might still yield better performance. While PEFT is highly effective in most cases, it may not be the optimal solution in every scenario.
- **Integration Complexity:** There might be some complexity when using multiple PEFT modules simultaneously or integrating them with specific frameworks.

PEFT Techniques

- **Adapters**
- **LoRA (Low-Rank Adaptation)**
- **QLoRA (Quantized Low-Rank Adaptation)**
- **Prefix tuning**
- **Prompt tuning**
- **P-tuning**
- **IA3**

PEFT Methods	Description	When to Use	Computational Overhead	Memory Efficiency	Versatility across Tasks	Performance Impact
Prompt Tuning	Modifies LLM's hidden states with trainable parameters in response to task-specific prompts.	Large pre-trained LLM. Adaptation to multiple tasks.	Low	Moderate	High	Depends on prompt quality
Prefix Tuning	Adds a trainable prefix to modify LLM's learned representation.	Task-specific adaptation. Limited resources.	Low	Moderate	Moderate	Can vary, but usually positive with proper tuning
Adapters	Inserts neural modules between LLM layers; only adapter weights are updated during fine-tuning.	Multiple tasks on one LLM. Flexibility required.	Moderate	Good (only adapters are fine-tuned)	High (can be added for multiple tasks)	Typically positive if adapters are well-tuned
LoRA	Introduces a low-rank matrix into the attention mechanism to learn task-specific patterns.	Tasks with specialized attention requirements. Limited resources.	Low-Moderate	Good	Moderate	Generally positive with good training
QLoRA	Builds on LoRA with quantization for enhanced memory efficiency.	Strict memory constraints. Emphasis on performance & efficiency.	Low	Excellent	High	Comparable or better than full fine-tuning
QALoRA	Enhances LoRA with quantization-aware techniques for fine-tuning low-bit diffusion models.	Optimizing efficiency in low-bit settings. Resource-constrained environments.	Low	Excellent	Moderate	Enhanced efficiency and effectiveness in specific settings
ReLoRA	Iteratively applies low-rank updates for efficient training of high-rank networks.	Large-scale models requiring high-rank training with reduced resources.	Moderate	Good	Moderate	Achieves high-rank training efficiency and performance

Adapters



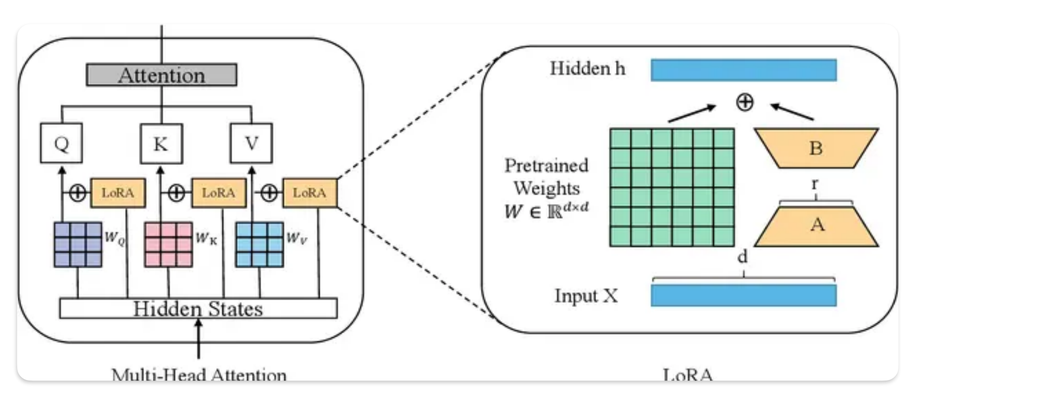
- **"Adapters are special sub-modules added to pre-trained language models that modify hidden representations during fine-tuning."**
- This is the central idea of Adapter Tuning. The adapter's role is to maintain the knowledge embedded in the original model while making subtle adjustments to suit a specific task. "Modifying hidden representations" means adapting the model's internal feature extraction or information flow to be more suitable for the task.

- **"Positioned after specific layers in the Transformer architecture, adapters allow parameters to be updated while keeping the rest of the model fixed."**
- This is also precise. Adapters are typically inserted within Transformer blocks, for example, between the Multi-Head Attention and Feed-Forward Network layers. By doing so, the vast parameters of the original model remain frozen, and only the adapter's parameters are trained, significantly saving computational resources and storage space.

Key points across the entire PEFT learning methodology include:

- The goal of **"efficient fine-tuning with fewer parameters."**
- Emphasis on **"preserving knowledge"** from the pre-trained model (by freezing existing parameters).
- Achieving **"adaptability"** to specific tasks through either new, small sets of parameters or partial adjustments of existing ones.

LoRA(Low Rank Adoption)



This significantly reduces the number of trainable parameters for downstream tasks, often by more than 10,000 times, and cuts GPU memory requirements by 3 times. Despite this reduction, LoRA maintains or even surpasses the quality of full fine-tuning models across tasks, lowering hardware barriers and ensuring efficient task switching without additional inference latency.
LoRA represents a smart balance, efficiently adapting large pre-trained models to specific tasks or datasets while preserving their core strengths. It is a technology that redefines efficiency in the world of massive language models.

it largely keeps most parameter weights as they are while only fine-tuning a small portion, and that this approach helps save training costs and computational resources while improving performance on specific tasks.

How Lora Works

- **Freeze Original Weights:** All weights W of the pre-trained model are **frozen** during training.
- **Add LoRA Modules:** LoRA adds small LoRA modules in parallel to specific weight matrices within the Transformer blocks (primarily query (WQ), key (WK)), value (WV), and output (WO) weights).
- **Train LoRA Modules:** Each LoRA module consists of two linear layers (matrices A and B).
- Given an input X, the original path computes $x\mathbf{W}$.
- The LoRA path first computes $x\mathbf{A}$ and then transforms that result into $x\mathbf{AB}$. (Typically, **A** is initialized with a random Gaussian distribution, and **B** is initialized to zero, so $\Delta\Delta\mathbf{W}$ is zero at the start of training.)
- The final output is $x\mathbf{W}+x\mathbf{BA}$.

- **What is Learned:** During the training process, only the parameters of these small **A** and **B** matrices are updated. The parameters of the original model **W** remain unchanged.

4. Why LoRA Works Well (Intuitive Explanation)

- Large pre-trained models already possess vast general knowledge.
- Fine-tuning for a specific task is less about fundamentally changing the model's "knowledge" and more about "adjusting" or "adapting" it in a particular direction.
- LoRA's key insight is that this "adjustment" can be sufficiently represented by a low-rank change in the overall weight space. It's like making subtle modifications to specific details rather than drastically altering the entire painting.

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Lora is effective with Large Language Models

QLoRA(Quantized Low Rank Adoption)

Quantization is also a PEFT methodology, referring to the process of reducing the number of bits by converting the parameters of a language model from floating-point to integer format. For example, it's a method of reducing model size by decreasing the number of bits, such as converting 32-bit floating-point parameters to 8-bit integers.

QLoRA (Quantized Low-Rank Adaptation) is an extension of the Parameter-Efficient Finetuning (PEFT) approach for large pre-trained language models like BERT, building upon the Adapter concept.
In QLoRA, instead of adding new task-specific layers while keeping the pre-trained model fixed, existing higher layers are adapted.
These layers are made more efficient by quantizing their weight matrices and decomposing them into low-rank approximations.

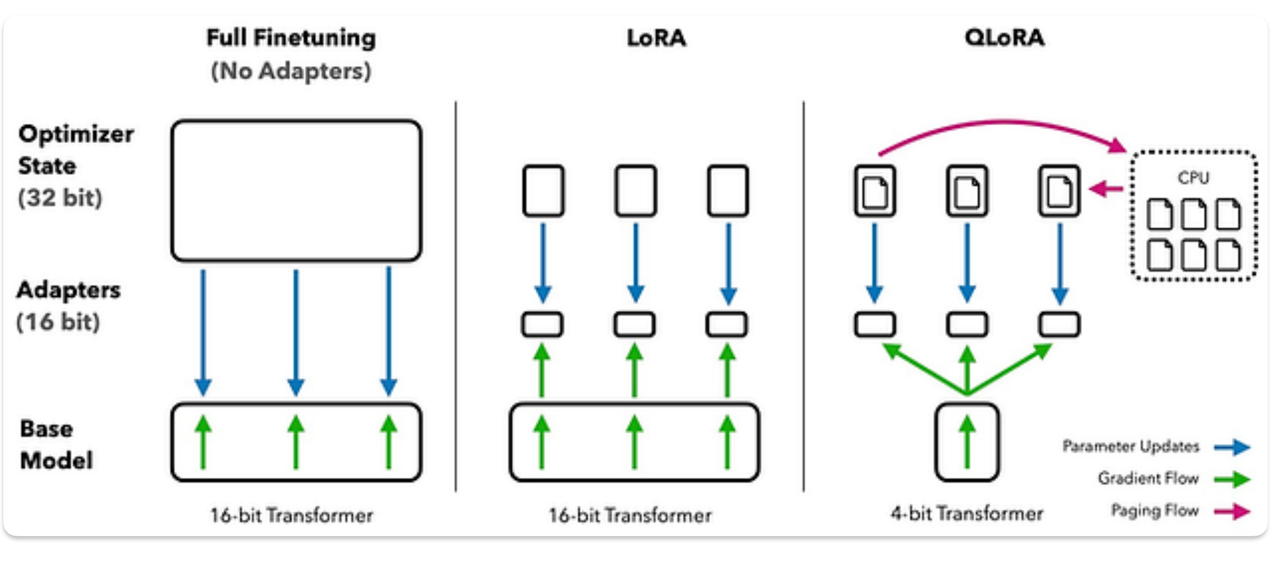
with QLoRA, you efficiently modify existing pieces. And these small modifications improve the performance of large models while reducing memory and computational costs. Ultimately, we perform new tasks well without forgetting what was previously learned.

. Double Quantization

- QLoRA uses a technique called "Double Quantization" to further reduce memory usage.
- This involves quantizing the quantization constants themselves. In the typical quantization process, "quantization constants" are generated to quantize the weights, and these constants can also occupy significant memory. Double Quantization further reduces memory usage by re-quantizing these quantization constants to 8 bits. This is conceptually similar to compressing small files with ZIP.

Performance

- The QLoRA paper demonstrated that fine-tuning a 65B model with QLoRA achieved nearly identical performance to 16-bit full fine-tuning, while using significantly less memory. This indicates that QLoRA is not just a memory-saving technique but also competitive in terms of actual performance.



Prefix-Tuning

- **"Prefix-tuning is a simple method for training large language models for tasks like writing."**
- Here, "writing" can encompass various text generation-related tasks (e.g., summarization, translation, dialogue generation).
- "Simple method" implies that instead of the complex and resource-intensive process of fine-tuning the entire model, only a small number of parameters are trained.
-
- **"Instead of adjusting all parts of the model, which can be costly, prefix-tuning focuses on a small, task-specific part called the prefix."**
- This is the fundamental philosophy of PEFT. The parameters of the pre-trained model are fixed, and only a small, task-specific part is learned.
- Here, "prefix" refers to a sequence of learnable continuous vectors added to the input sequence of the model.
-
- **"This prefix helps guide the model to write in a specific way for a particular task."**
- Similar to how prompt engineering involves a human writing "instructions" to guide the model's behavior, Prefix-tuning uses a "soft prompt" learned by the model itself to manipulate the model's internal state and induce the desired output. In essence, it indirectly instructs the model to "generate this kind of text."
-

To add more features of Prefix-tuning:

- **Continuous Prompts:** The prefix learned in Prefix-tuning is not actual word tokens but a continuous sequence of vectors directly learned in the model's embedding space. This is one of the main differences from Prompt-tuning.
- **Influence on Attention Mechanism:** These prefix vectors participate in the model's attention mechanism, influencing what information the model focuses on and how it understands context when generating text.
- **Few Trainable Parameters:** While not as few as LoRA or Adapters, it still trains an extremely small number of parameters compared to the entire model.
- **Inference Overhead:** Since the prefix is treated like part of the actual input sequence, the inference sequence length becomes slightly longer, potentially causing a very minimal additional latency.

Prefix-tuning was proposed as an important technique in the early stages of PEFT research and subsequently influenced the development of Prompt-tuning, LoRA, and other methods.