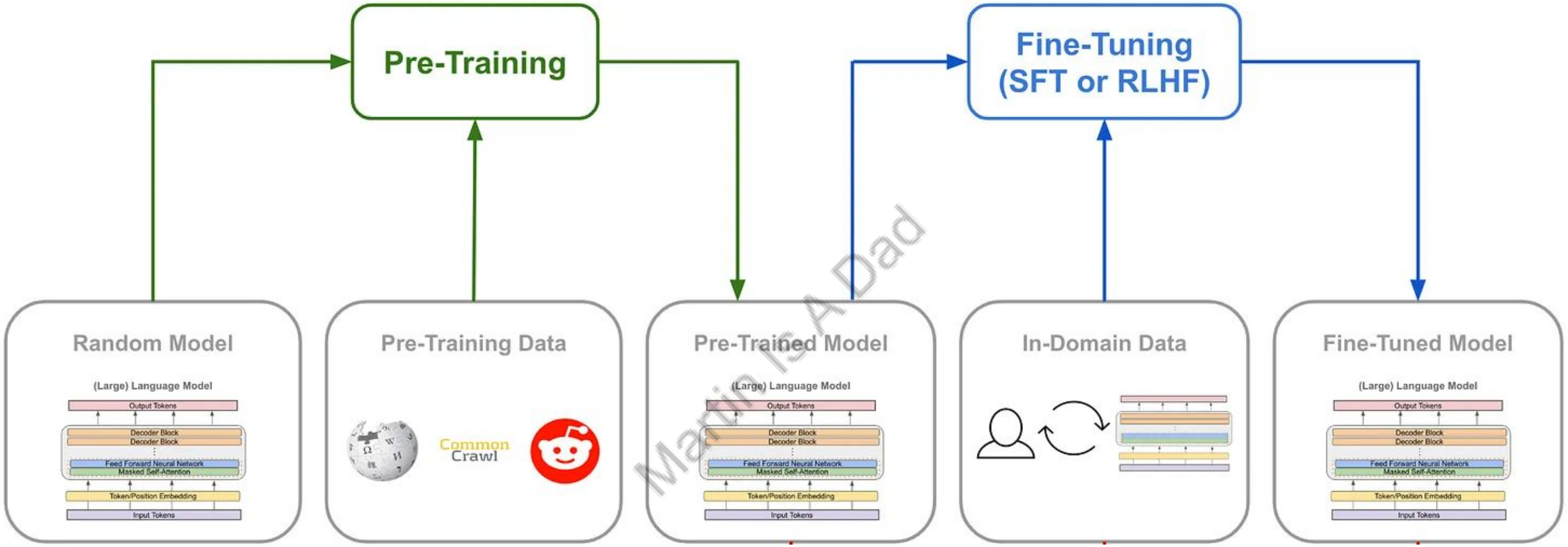
LLM Training Flow



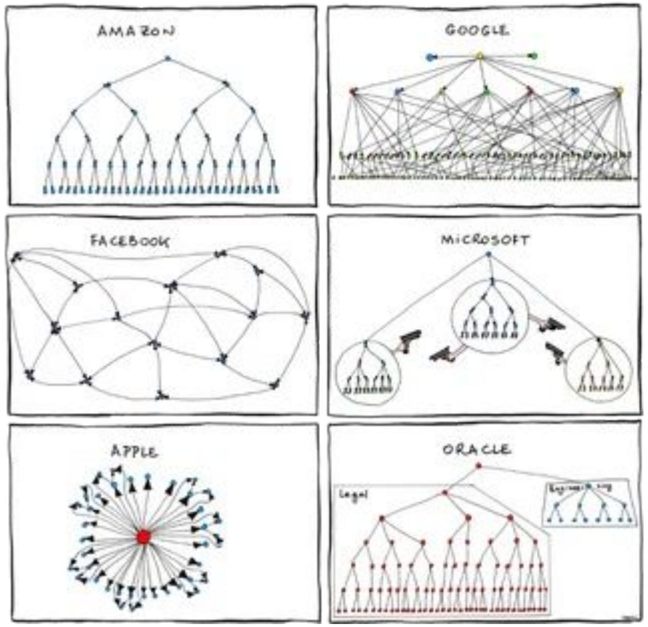
- The trend (industry / research) is focusing more and more in Post-Training (Fine-Tuning) stage from Pre-Training. More details refer to my LLM Training video.
- Modern models are large, with large numbers of parameters
  - GPT-4 is reported to have approximately 1.8 trillion parameters
  - DeepSeek-V3 has 671 billion parameters, with 37 billion activated (MoE) during inference
- Updating model weights requires computation, storing model weights requires storage

Million Dollar Questions

- Primary purpose of pre-training LLMs is to enable vast and general understanding of language and the world from massive amounts of unlabeled data (text, image, code etc).
    - In this phase it make sense to touch all parameters in the model, since we are trying to get a generalized foundation model from scratch (0 or random weights).

- Primary purpose of post-training LLM is to refine and align the pre-trained model's capabilities and behavior
    - ***Do we still need to change all parameters, with full rank/dimension?***

- What if we need to make task specific modifications to the foundation model, do we need to do a full retrain everytime? ***Is this scalable?*** For example,
    - Verily needs model with extensive medical data
    - Youtube Kids team needs additional safety checks



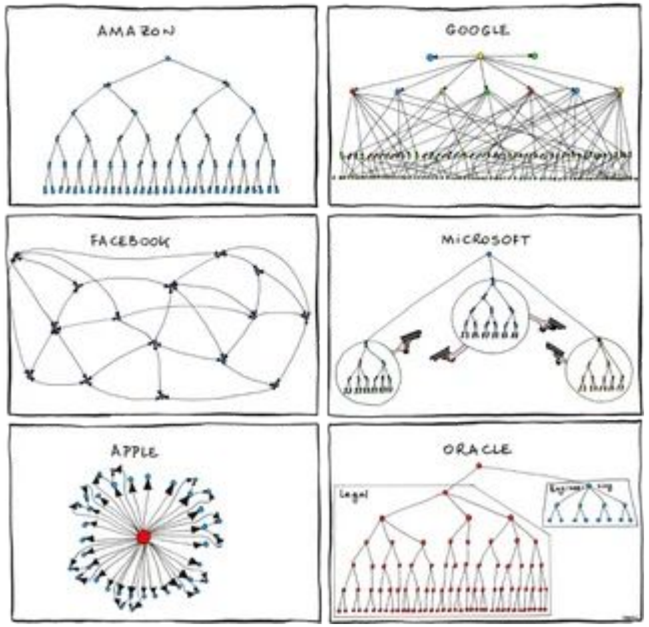
Million Dollar Questions

- Primary purpose of pre-training LLMs is to enable vast and general understanding of language and the world from massive amounts of unlabeled data (text, image, code etc).
  - In this phase it make sense to touch all parameters in the model, since we are trying to get a generalized foundation model from scratch.

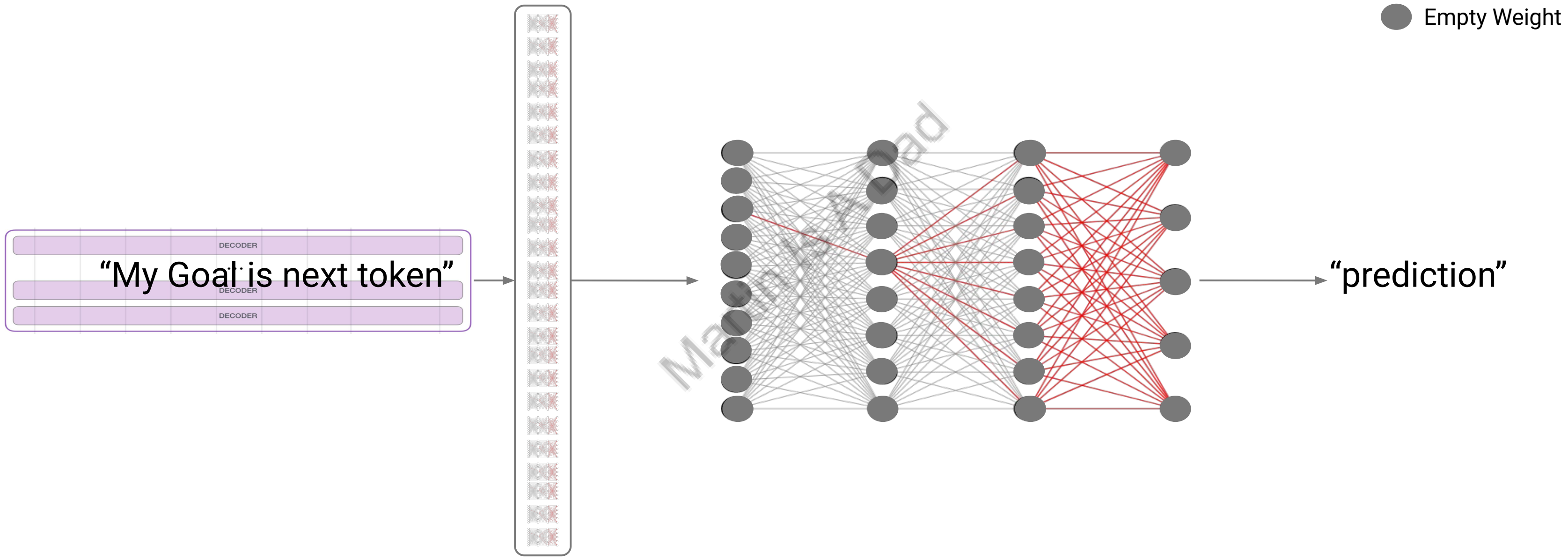
- Primary purpose of post-training LLM is to refine and align the pre-trained model's capabilities and behavior
  - ***Do we still need to change all parameters, with full rank/dimension?***

- What if we need to make task specific modifications to the foundation model, do we need to do a full retrain everytime? ***Is this scalable?*** For example,
  - Verily needs model with extensive medical data
  - Youtube Kids team needs additional safety checks

No and **No.** What can we do instead?



Visualize Pre-Training

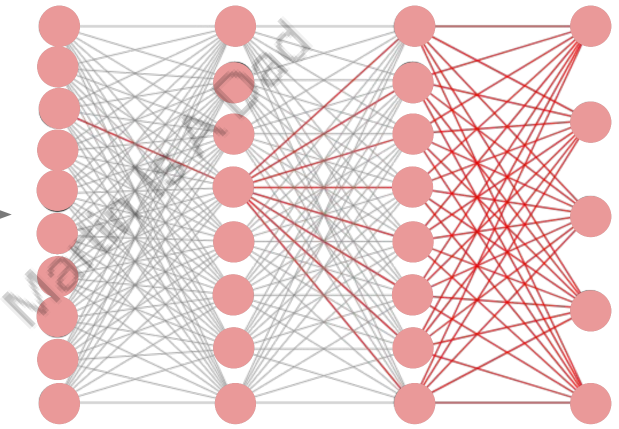


Filling **all the weights** of a model

Visualize Full Fine Tuning

● Tweakable Weight

“Answer yes or no. 2+2=5.
Answer:”



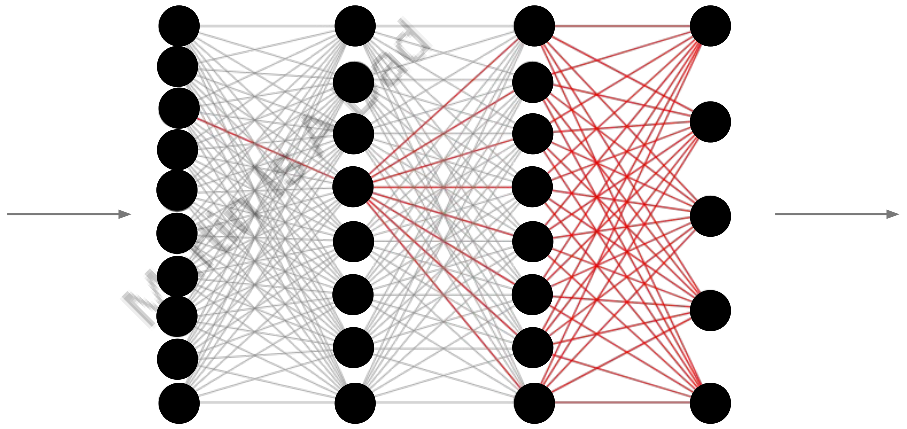
“No”

Tweaking **all the weights** of a model. More details please take a look my video on LLM Training.

Visualize Prompt Engineering

● Frozen Weight

“Image you are a chess player and playing white, what’s your first move?”



“Pawn e2 to e4”

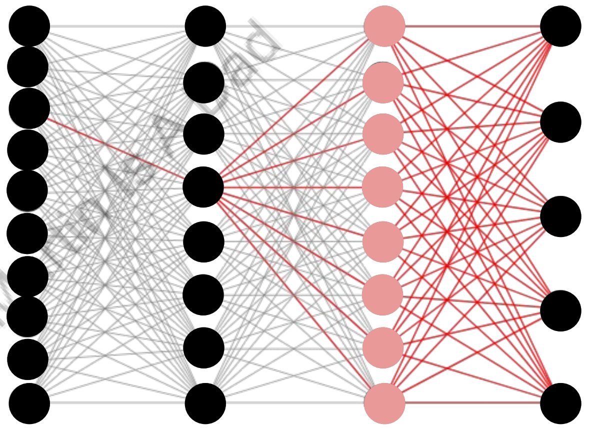
Using **existing** weights. More details take a look at my video on prompt engineering.

What If We Can Do This?

● Frozen Weight

🔴 Tweakable Weight

“Can you diagnose pneumonia with this image?” →



→ “Yes”

Tweaking **some of the weights** of a model.

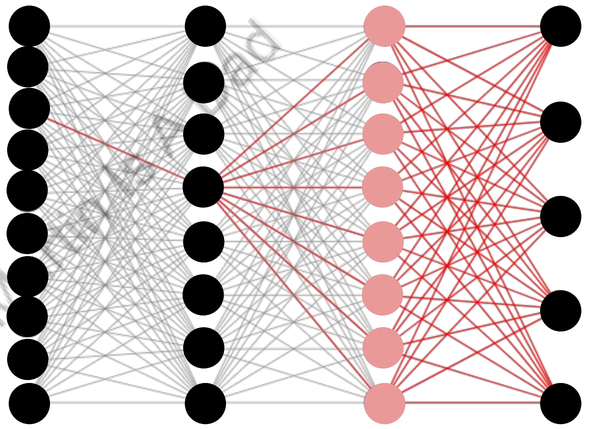
What If We Can Do This?

PEFT = Parameter-efficient fine-tuning

● Frozen Weight

● Tweakable Weight

"Can you diagnose pneumonia with this image?" →



→ "Yes"

Tweaking **some of the weights** of a model.

PEFT Family Graph

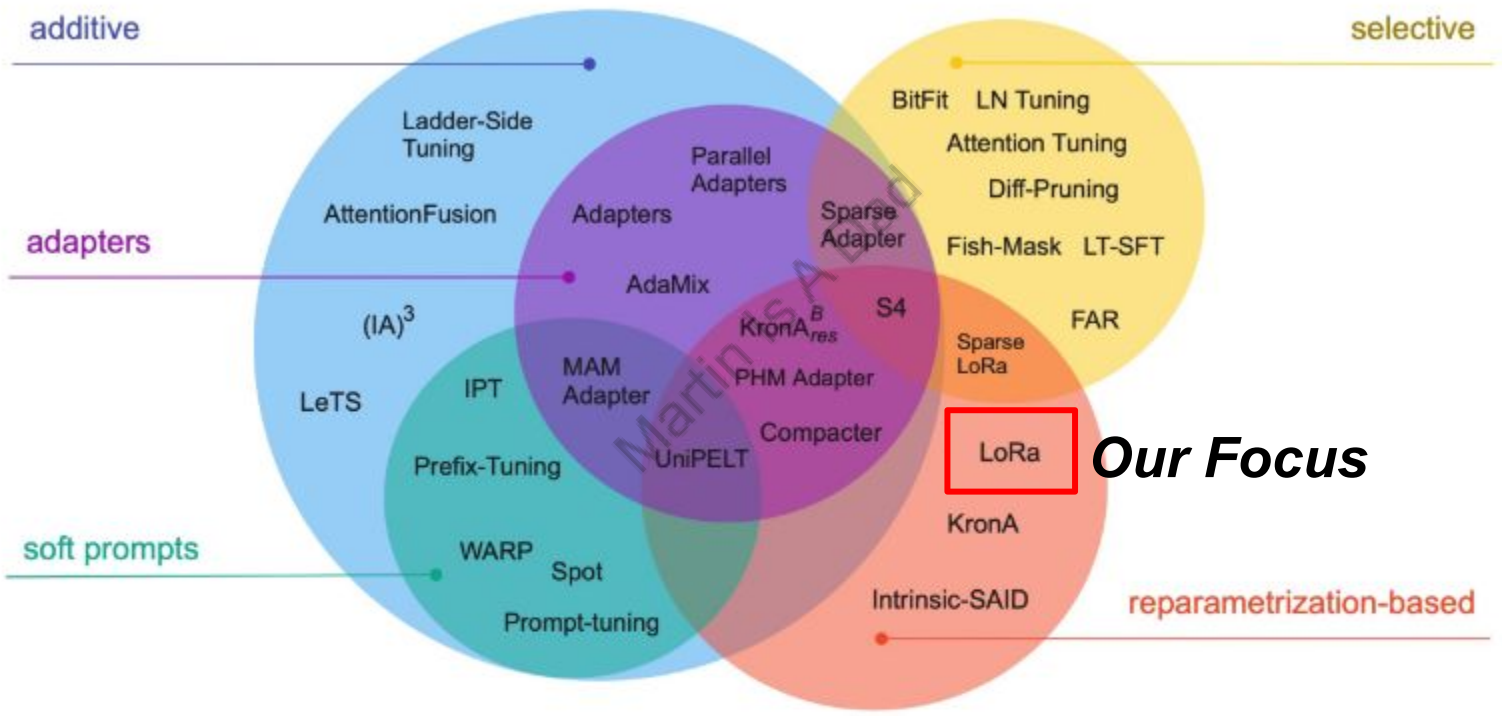
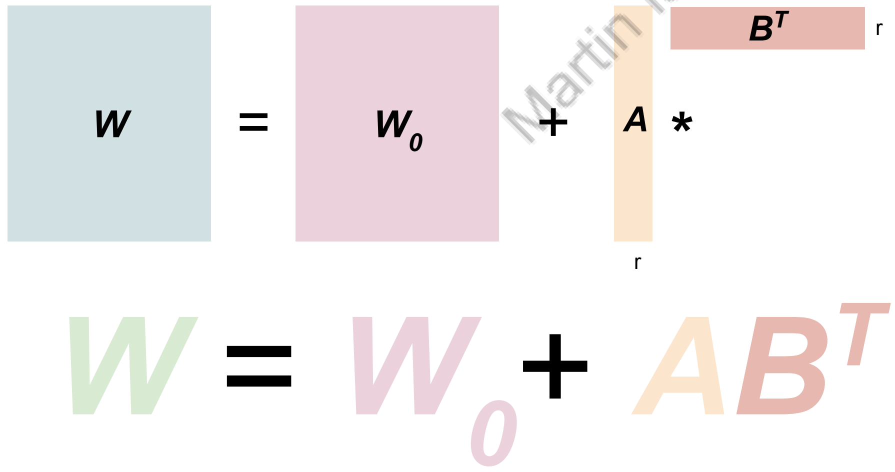


Image from [2303.15647] Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning

LoRA (Low-Rank Adaptation)

- Low-Rank Adaptation (LoRA) proxy model ($W \in R^{d \times k}$) updates ($\Delta W \in R^{d \times k}$) in the form of two low ranked matrices, $A \in R^{d \times r}$ $B \in R^{k \times r}$, where r << min(d, k). $\boldsymbol{\Delta W \approx A B^T}$
  - Significantly reduce fine-tuning time
  - Significantly reduce checkpoint size
- Intuition: not all weights are equally important, some are a lot more important, say attentions

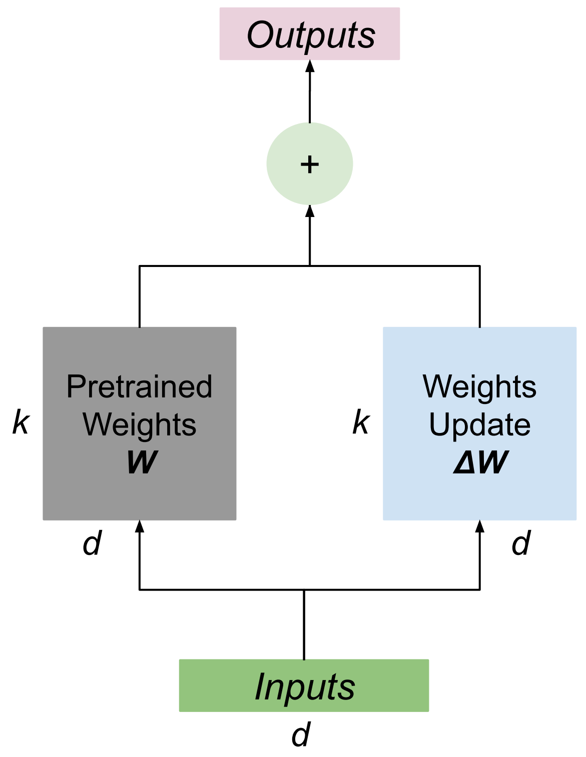


$$W = W_0 + AB^T$$

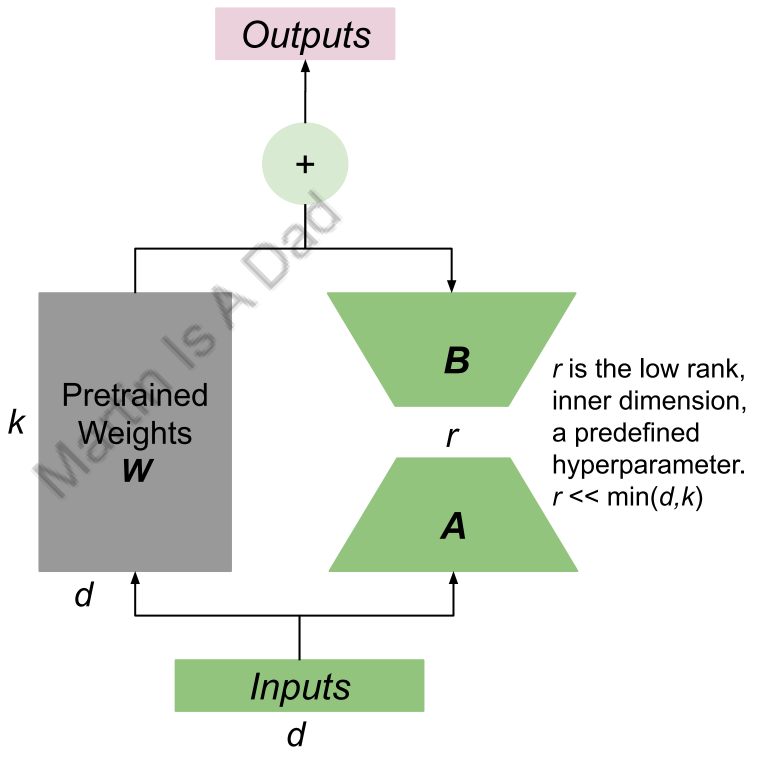
r is a hyperparameter

- Small value will shorten training time and storage
- If value too small might cause information loss and hurt model quality
- Empirically, *r* can range from 8 to 256

LoRA (Low-Rank Adaptation)



Full Fine Tuning Weight Updates



r is the low rank, inner dimension, a predefined hyperparameter. *r* << min(*d*,*k*)

LoRa Weight Updates

Example:

d = 100 k
k = 200 k
r = 16

Without LoRA:
100 k X 200 k = 20 **billion**

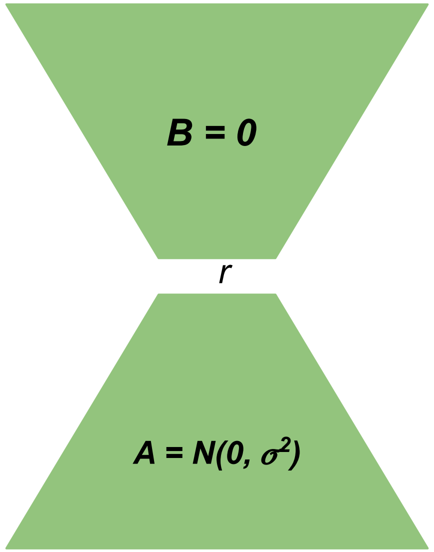
With LoRA:
100 k X 16 + 200 k X 16 = 4.8 **million**

4167X less parameters with LoRA!

LoRA Training Details

Initialization of ***A*** and ***B***:

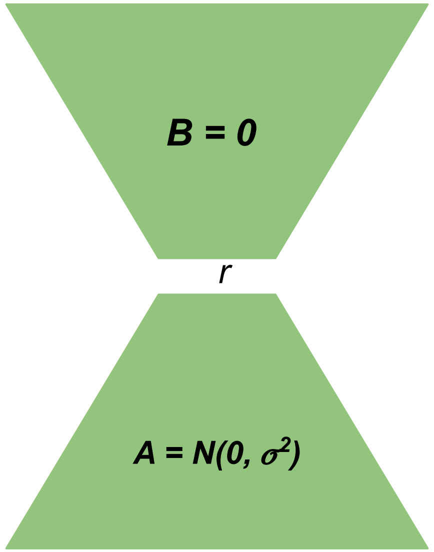
- The initialization of matrices ***A*** and ***B*** is crucial for stable and effective training.
- The most common initialization strategy is:
  - Matrix ***A***: Initialized randomly using a standard Gaussian distribution with a small standard deviation (e.g., $N(0, \sigma^2)$).
  - Matrix ***B***: Initialized with all zeros.
- Intuition
  - At the beginning of fine-tuning, $\Delta W = B \times A$ should be zero, to preserve the original pre-trained weights. The random initialization of ***A*** allows for different initial directions for adaptation during training.



LoRA Training Details

Training ***A*** and ***B***:

- Once ***A*** and ***B*** are initialized, they are trainable parameters in the model during the fine-tuning process with LoRA
- The forward pass of a layer with LoRA is modified as follows:
  - $h = \mathbf{W_0}x + \mathbf{BA}x = (\mathbf{W_0} + \mathbf{BA})x$
  - where x is the input to the layer and h is the output.
- In backpropagation, the gradients are calculated **only for the parameters in matrices *A* and *B***, while $\mathbf{W_0}$ remains frozen. These gradients are then used to update ***A*** and ***B*** using an optimization algorithm (e.g., Adam) to do gradient descent.



LoRA Pros

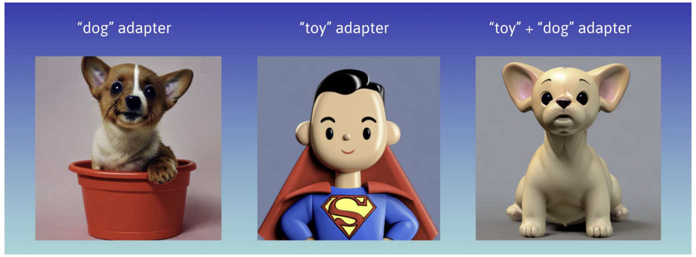
- **Significant Reduction in Trainable Parameters**
  - Lower Computational Cost: Less memory (GPU RAM) is required during training, allowing fine-tuning on less powerful hardware.
  - Faster Training Times: With fewer parameters to update, the training process converges much faster.
  - Smaller Storage Footprint: The LoRA adapters are very small compared to the full model (usually between 0.001% and 1%), making them easier to store and share.

- **Harder to overfit:** The frozen base model parameters and low-rank effectively acting as a regularizer

- **Better or comparable quality** when training data set is limited (O(K)), compared with full fine tuning

- **Task Isolation:** Multiple small, task-specific adapters can be attached to a single base LLM. These adapters can be easily loaded and swapped depending on the task, without needing to store or load multiple full fine-tuned models. This enables efficient multi-task learning or serving different applications with the same base model.

- LoRA deltas can be combined



LoRA Cons

- **Serving Latency Increase**
  - After merging with base model’s weights, inference latency should not increase
  - However, serving multiple checkpoints (base model, one or more LoRA adapters) could result into serving latency increase, depending on the infrastructure (RPC, in memory etc)

LoRA *vs* Full Fine Tuning *vs* Prompt Engineering

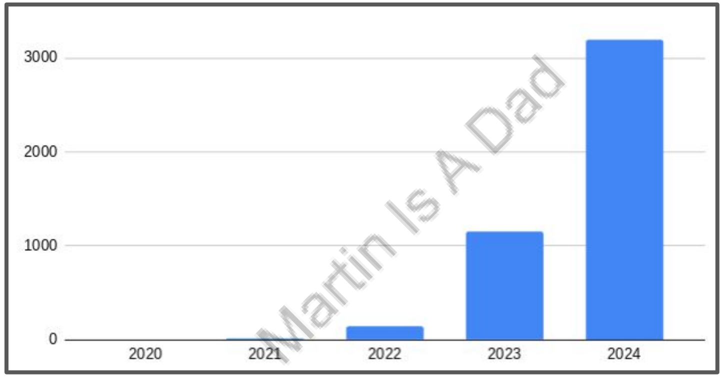
	Full Fine Tuning	LoRA	Prompt Engineering
Quality Improvement	✔✔ Usually have best quality	✔ Close quality, or better when training data is limited	✖ No model improvement, just a way to get better response
Tuning Time	✔ Long	✔ A lot shorter (hours)	✔✔ Very little time to run prompt
Tuning Cost	✔ More memory and chips than LoRA	✔ Lot lower cost than Full Fine Tuning	✔✔ No tuning cost
Training Data	✔ Large number of data (> 10k)	✔ Small number of data O(k)	✔✔ No additional data needed

LoRA *vs* Full Fine Tuning *vs* Prompt Engineering

	Full Fine Tuning	LoRA	Prompt Engineering
Model Storage Cost	✓ Large storage needed to save full weights	✓ Only saving LoRA adapters' weights	✓✓ No additional storage needed
Task Isolation	✓ Separate models needed for different task	✓ Task specific LoRA adapters can be easily combined, swapped, removed	✓✓ Use different prompt for different tasks
Serving Latency	✓✓ No additional serving latency	✓ Little serving latency increase	✓✓ No additional serving latency
Serve within Mobile	✖ Too big for mobile, unless distilled	✓✓	✓✓

LoRA Adoption Across Industry

- Since the original LoRA publication in 2021, it has been adopted across essentially all customer surfaces: OSS (Open Source LLMs) fine-tuning & serving applications, Cloud tuning, on-device AI (Apple for example)



Google Scholar results for LoRA

- Not only SFT, Reinforcement Learning is adopting LoRA
    - RLHF can use LoRA for both reward modeling and policy optimization, achieving comparable performance to full fine-tuning with significantly reduced computational costs. More commonly being referred as PERL (Parameter Efficient Reinforcement Learning)
    - Different LoRA adapters can be trained for different RL tasks or environments using the same pre-trained backbone.

QLoRA (Quantized Low-Rank Adaptation)

- Quantization reduces the number of bits used to represent model weights, significantly decreasing memory usage. I have gone through this in the DeepSeek-V3 video.

- QLoRA workflow:

  - **Quantization**: Pre-trained LLM's weights are frozen after being quantized to 4-bit NF4 (4-bit NormalFloat). Model will be in a very memory-efficient state.

  - **LoRA Integration**: Low-rank adapters are added to the chosen layers of the frozen, quantized model. These adapters introduce a small number of trainable, high-precision (typically 16-bit or 32-bit) parameters.

  - **Fine-tuning**: Only the weights of the low-rank adapters are updated using backpropagation. The quantized base model remains frozen.

  - **Inference**: The low-rank updates for $\boldsymbol{A}$ and $\boldsymbol{B}$ matrix are added back to the original quantized weights $\boldsymbol{W_0}$ to create an effectively fine-tuned weight matrix. Alternatively, the original weights $\boldsymbol{W_0}$ can be dequantized back to a higher precision and then the adapter updates added.

QLoRA (Quantized Low-Rank Adaptation)

Feature	**LoRA**	**QLoRA**
Quantization	Typically uses standard precision (e.g., 16-bit) for the base model weights.	Quantizes the base model weights to a lower precision (e.g., 4-bit).
Memory Usage	Lower than full fine-tuning.	Significantly lower than LoRA, 75% smaller peak GPU memory usage.
Training Speed	Generally faster than QLoRA (~66% faster)	Slower than LoRA due to quantization/dequantization steps.
Complexity	Simpler to implement.	More complex due to quantization techniques.
Batch Size	Limited by memory constraints.	Can support much larger (10X) batch sizes, due to lower memory footprint. Thus have better training stability and faster convergence

Other PEFT Techniques

Adapter Tuning:

- **TL;DR**: Adapter tuning introduces small, new neural network modules called adapters into the existing architecture of the LLM. The weights of the original pre-trained LLM are frozen, only the parameters within these newly added adapters are trained on the task-specific data.

- **Architecture (Attached)**
  - Bottleneck architecture like AutoEncoder, aims to limit the number of trainable parameters. Refer to my AutoEncoder video for more details.
  - **Down-projection**: Reducing the high-dimensional input to a lower-dimensional space.
  - **Non-linearity**: Applying non-linear activation function (like ReLU).
  - **Up-projection**: Projecting the lower-dimensional representation back to the original higher-dimensional space.
  - **Residual Connection**: Adding the output of the adapter module to the original input of that layer, to improve gradient flow and address vanishing gradient

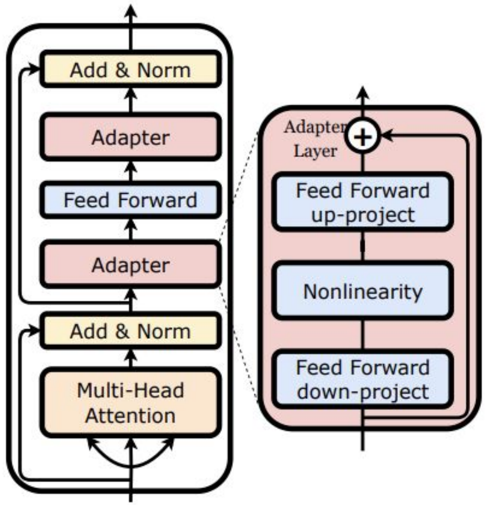


Image from [2304.01933] LLM-Adapters

Other PEFT Techniques

DoRA (Weight-Decomposed Low-Rank Adaptation):

- **Motivation:** Full fine-tuning and LoRA often exhibit different patterns of weight updates, particularly in terms of magnitude and direction. DoRA aims to bridge the gap by allowing for more nuanced updates to both aspects of the weights.

- **Essence:**
  - Weight Decomposition: The key innovation of DoRA is that it decomposes the pre-trained weight matrices into two components: magnitude and direction.
  - DoRA then fine-tunes both of these components. It specifically applies LoRA to update the directional component. This is because the directional component typically has a larger number of parameters, making low-rank adaptation efficient. The magnitude component can be updated more directly.

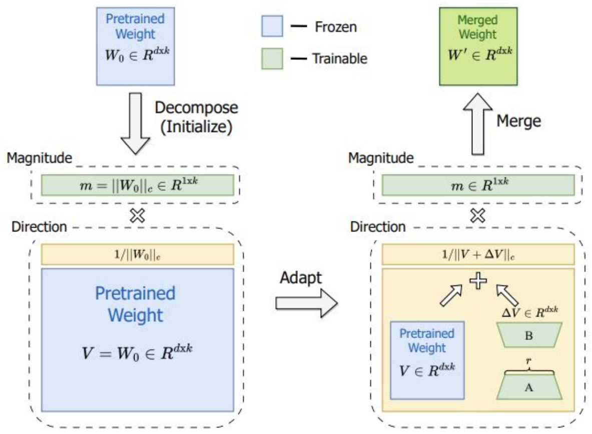


Image from [2402.09353] DoRA

Other PEFT Techniques

AND MORE!

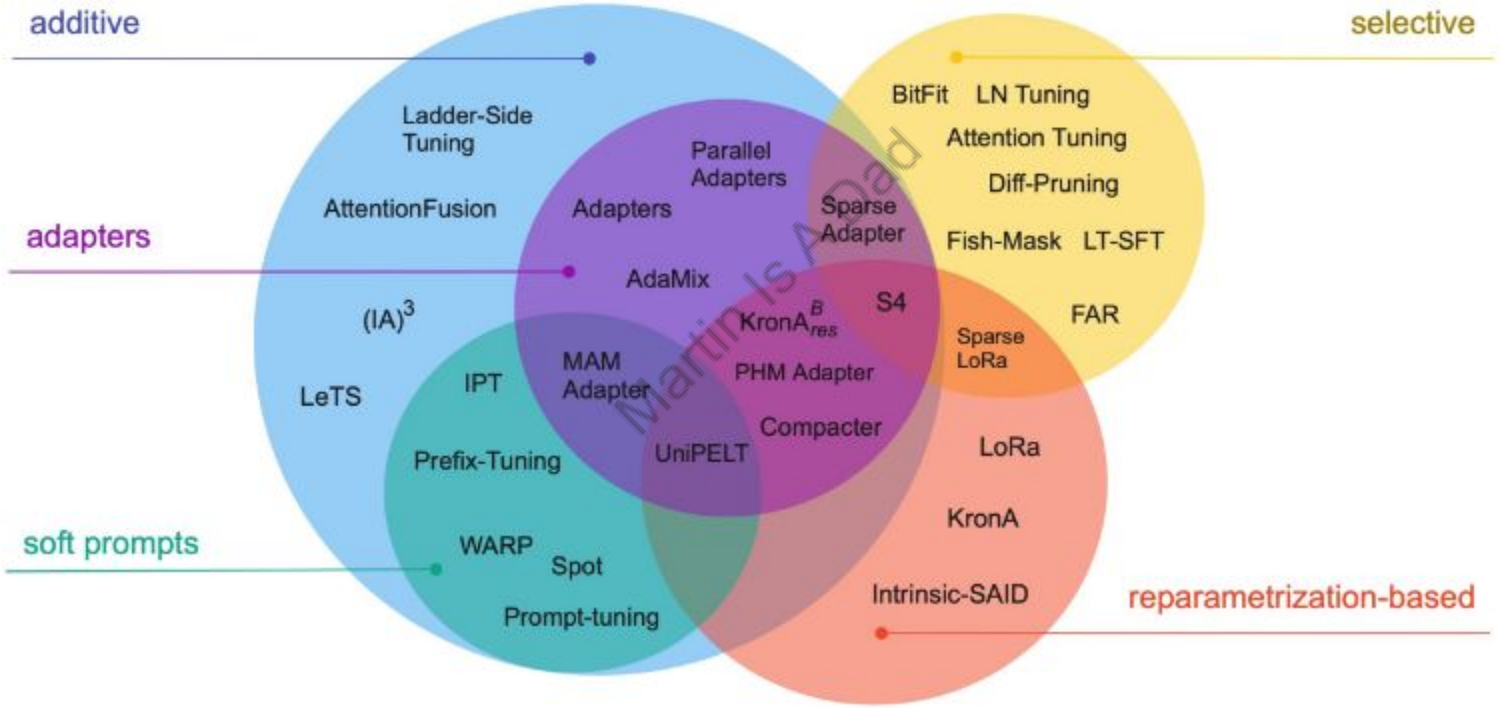


Image from [2303.15647] Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning