

Parameter Efficient Fine-Tuning (PEFT)

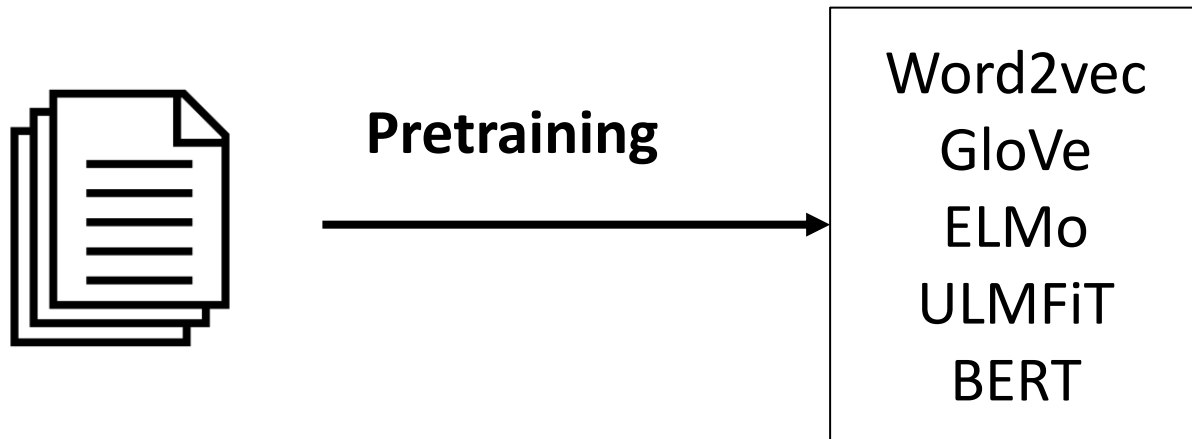
Large Language Models: Introduction and Recent Advances

ELL881 · AIL821



Dinesh Raghu
Senior Researcher, IBM Research

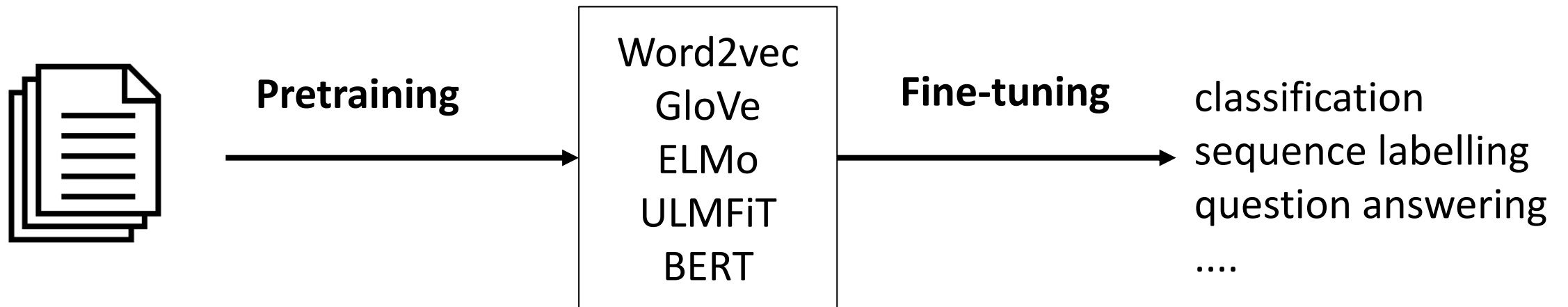
Transfer Learning Before the LLM Era



Adapted from [NAACL 2019 Transfer learning tutorial](#)



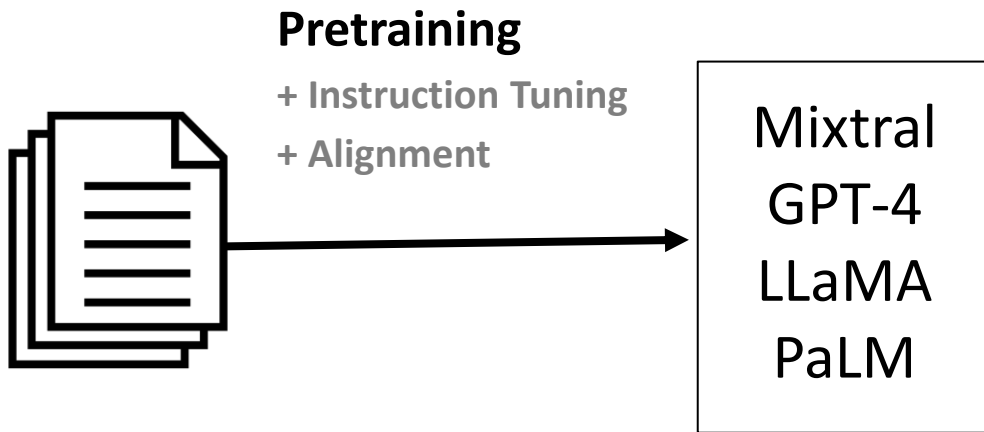
Transfer Learning Before the LLM Era



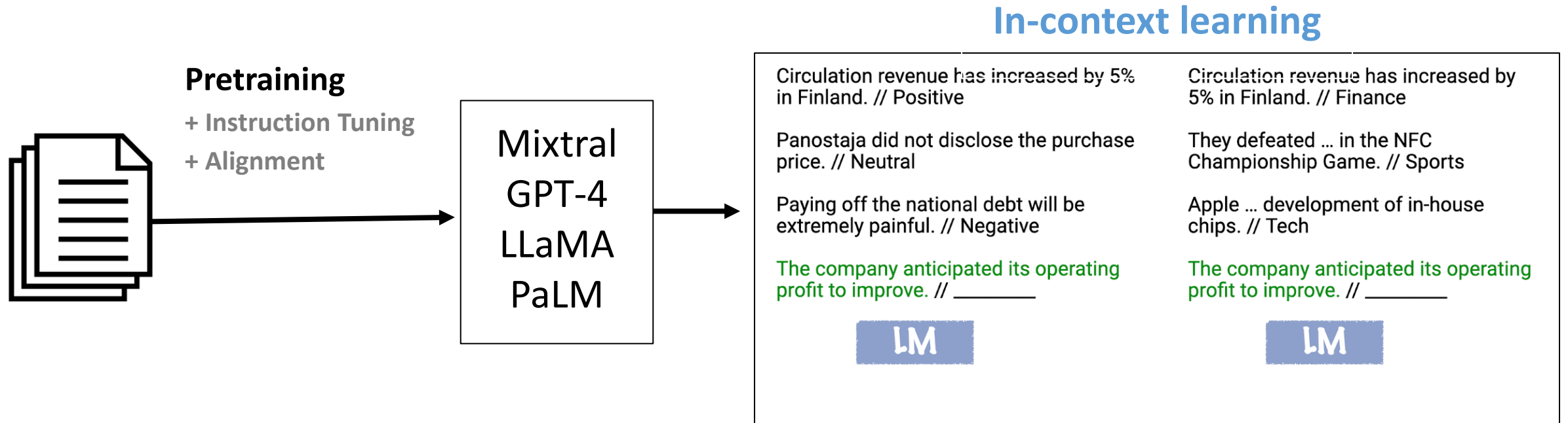
Adapted from [NAACL 2019 Transfer learning tutorial](#)



Transfer Learning in the LLM Era



Transfer Learning in the LLM Era



- In-context learning has mostly replaced **fine-tuning** in large models
- In-context learning is very useful if we don't have direct access to the model, for instance, if we are using the model through an API.



Downsides of In-context Learning



Downsides of In-context Learning

1. **Poor performance:** Prompting generally performs worse than fine-tuning [[Brown et al., 2020](#)].



Downsides of In-context Learning

1. **Poor performance:** Prompting generally performs worse than fine-tuning [[Brown et al., 2020](#)].
2. **Sensitivity** to the wording of the prompt [[Webson & Pavlick, 2022](#)], order of examples [[Zhao et al., 2021](#); [Lu et al., 2022](#)], etc.



Downsides of In-context Learning

1. **Poor performance:** Prompting generally performs worse than fine-tuning [[Brown et al., 2020](#)].
2. **Sensitivity** to the wording of the prompt [[Webson & Pavlick, 2022](#)], order of examples [[Zhao et al., 2021](#); [Lu et al., 2022](#)], etc.
3. **Lack of clarity** regarding what the model learns from the prompt. Even random labels work [[Min et al., 2022](#)]!

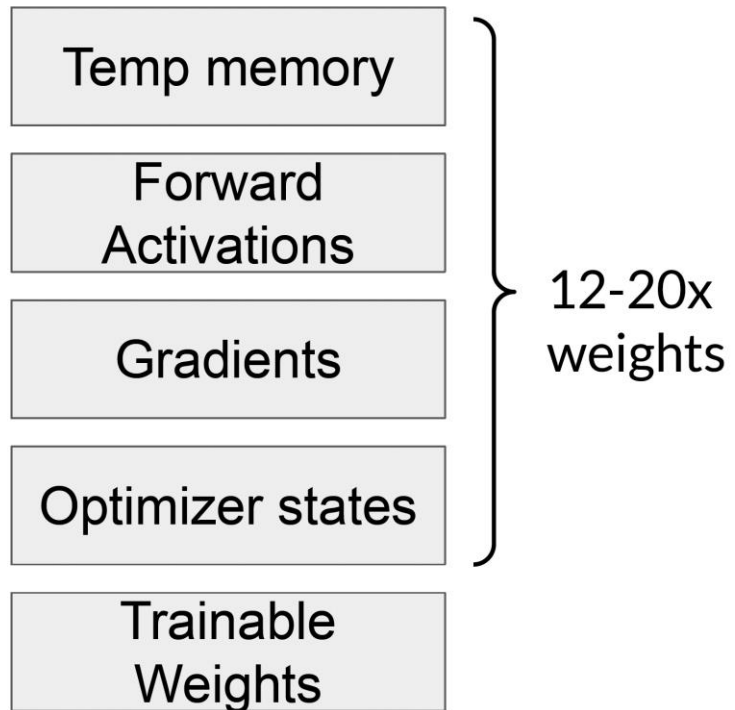


Downsides of In-context Learning

1. **Poor performance:** Prompting generally performs worse than fine-tuning [[Brown et al., 2020](#)].
2. **Sensitivity** to the wording of the prompt [[Webson & Pavlick, 2022](#)], order of examples [[Zhao et al., 2021](#); [Lu et al., 2022](#)], etc.
3. **Lack of clarity** regarding what the model learns from the prompt. Even random labels work [[Min et al., 2022](#)]!
4. **Inefficiency:** The prompt needs to be processed *every time* the model makes a prediction.



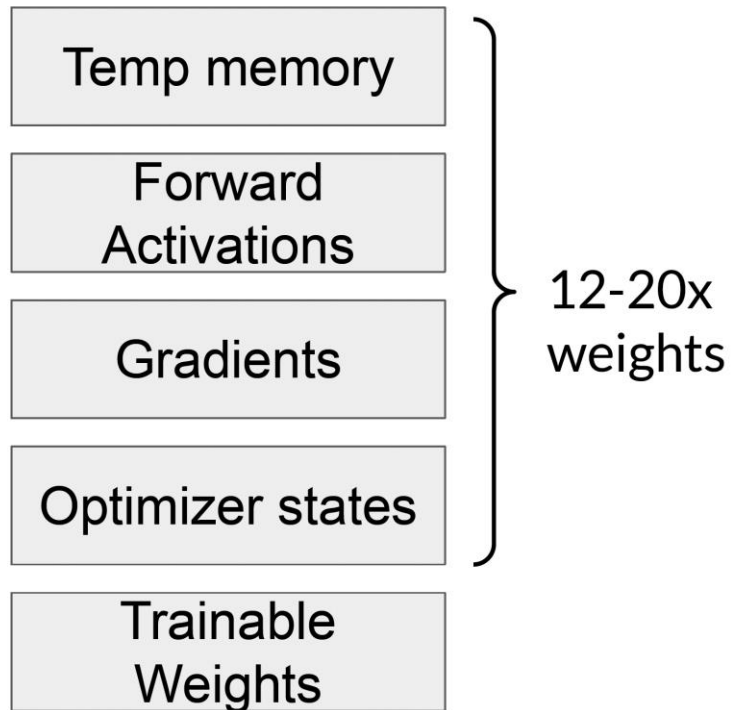
Why is Full Fine-tuning in LLM Challenging?



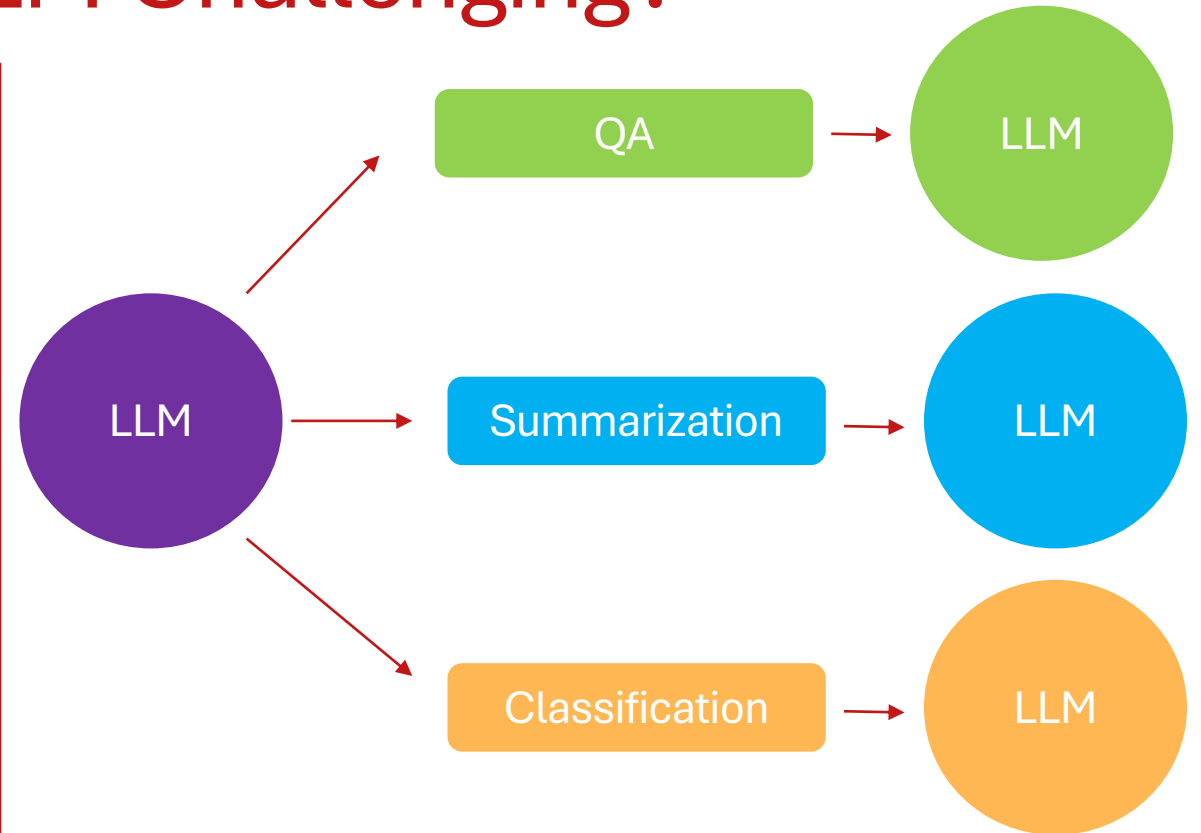
1. Hardware Requirements



Why is Full Fine-tuning in LLM Challenging?



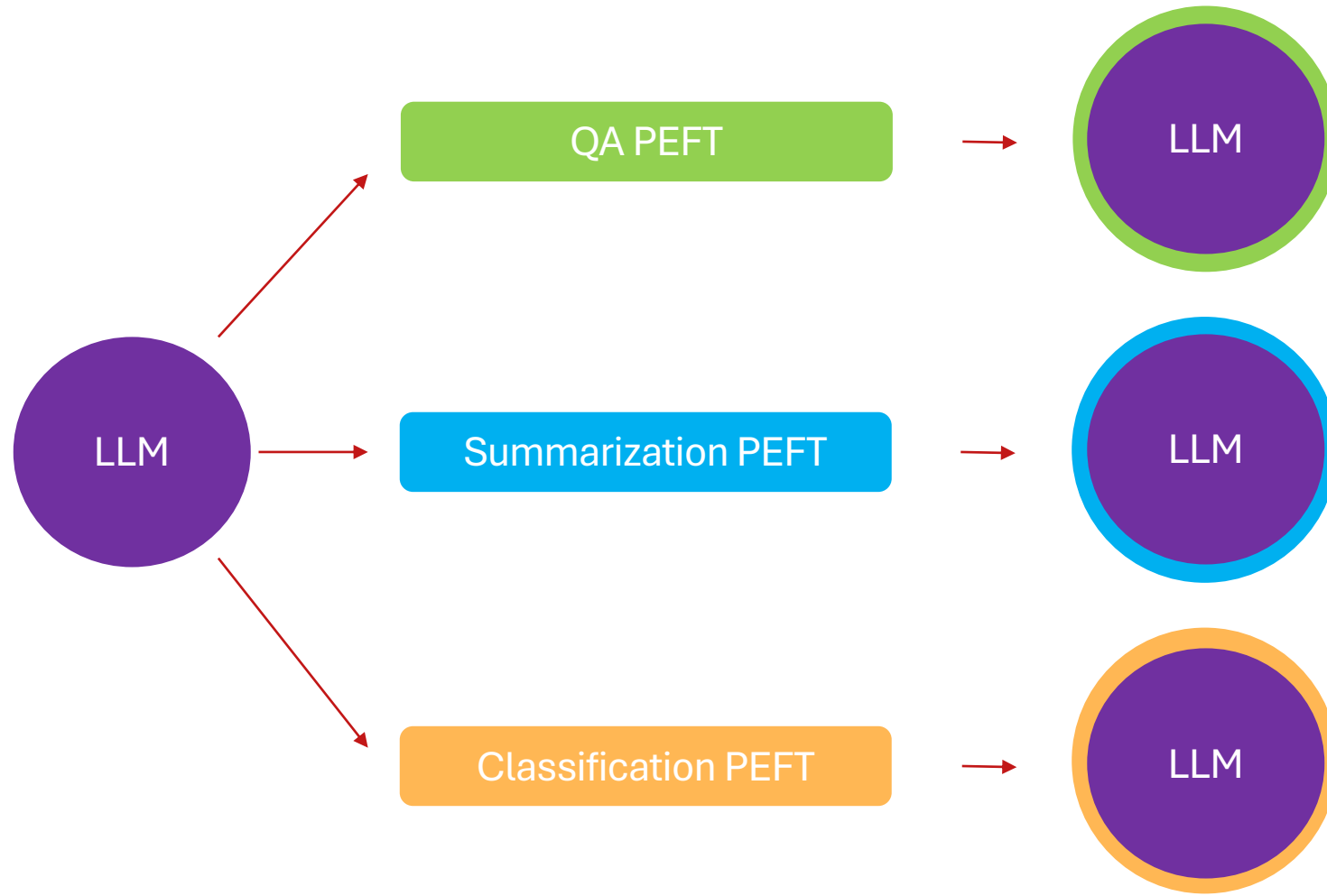
1. Hardware Requirements



2. Storage



Parameter Efficient Fine Tuning (PEFT)



PEFT Advantages

- Reduced computational costs
 - requires fewer GPUs and GPU time
- Lower hardware requirements
 - works with smaller GPUs & less memory
- Better modelling performance
 - reduces overfitting by preventing catastrophic forgetting
- Less storage
 - majority of weights can be shared across different tasks

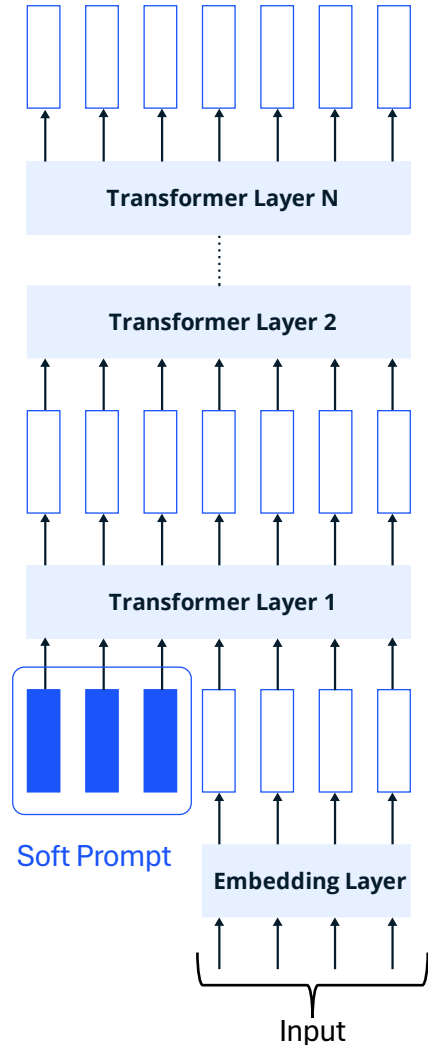


PEFT Techniques

- (Soft) Prompt Tuning
- Prefix Tuning
- Adapters
- Low Rank Adaptation



(Soft) Prompt Tuning ([Lester et al. 2021](#))



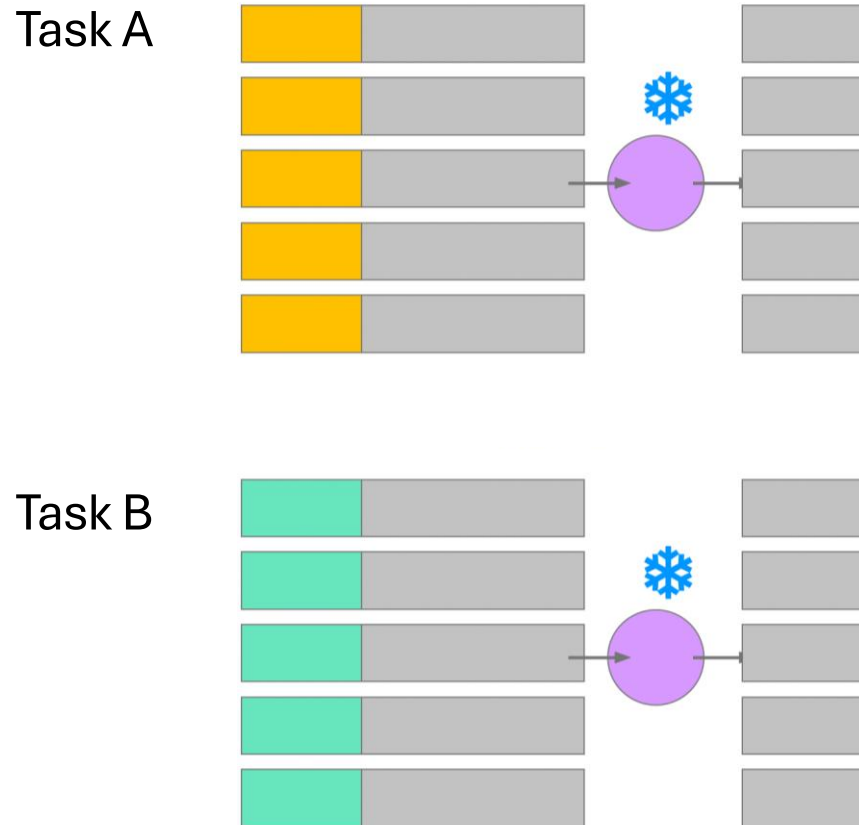
- prepends a trainable tensor to the model's input embeddings, creating a **soft prompt**
- for a specific task, only a small task-specific soft prompt needs to be stored
- soft prompt tuning is significantly more parameter-efficient than full-finetuning

Image Credits: leewayhertz.com



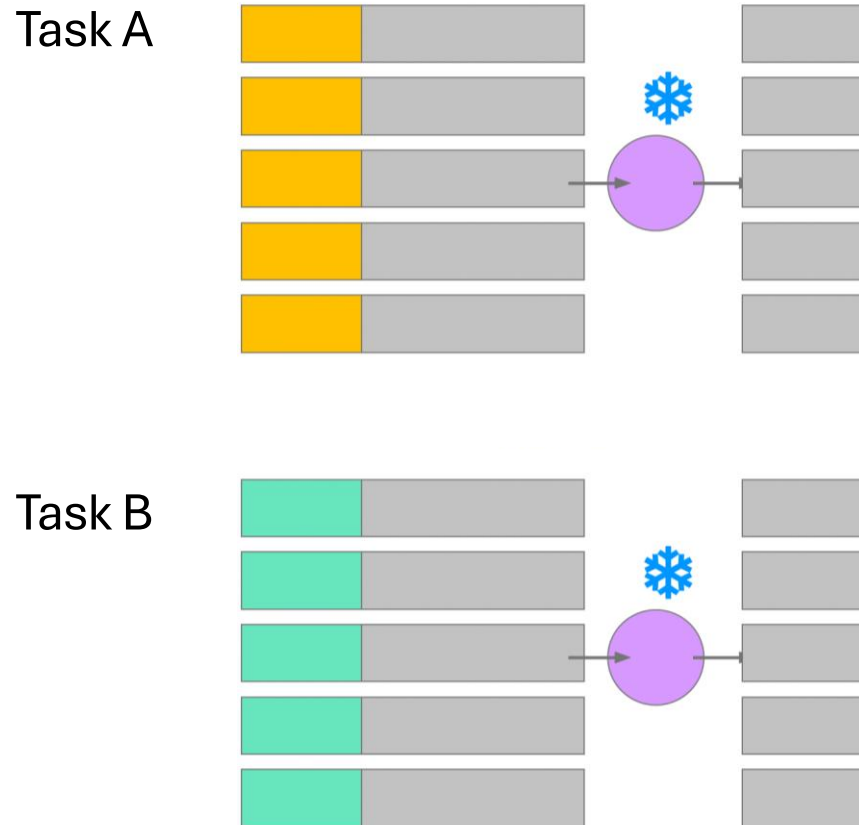
(Soft) Prompt Tuning: Multi-Task Serving

Training

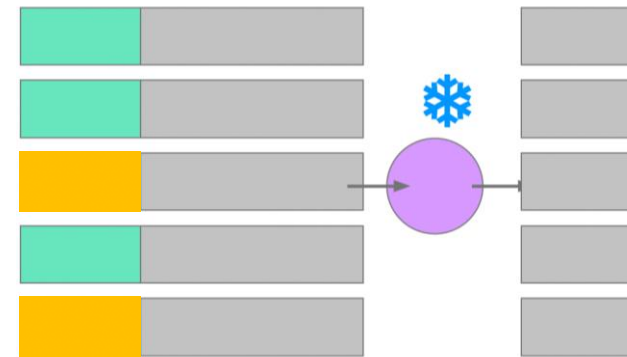


(Soft) Prompt Tuning: Multi-Task Serving

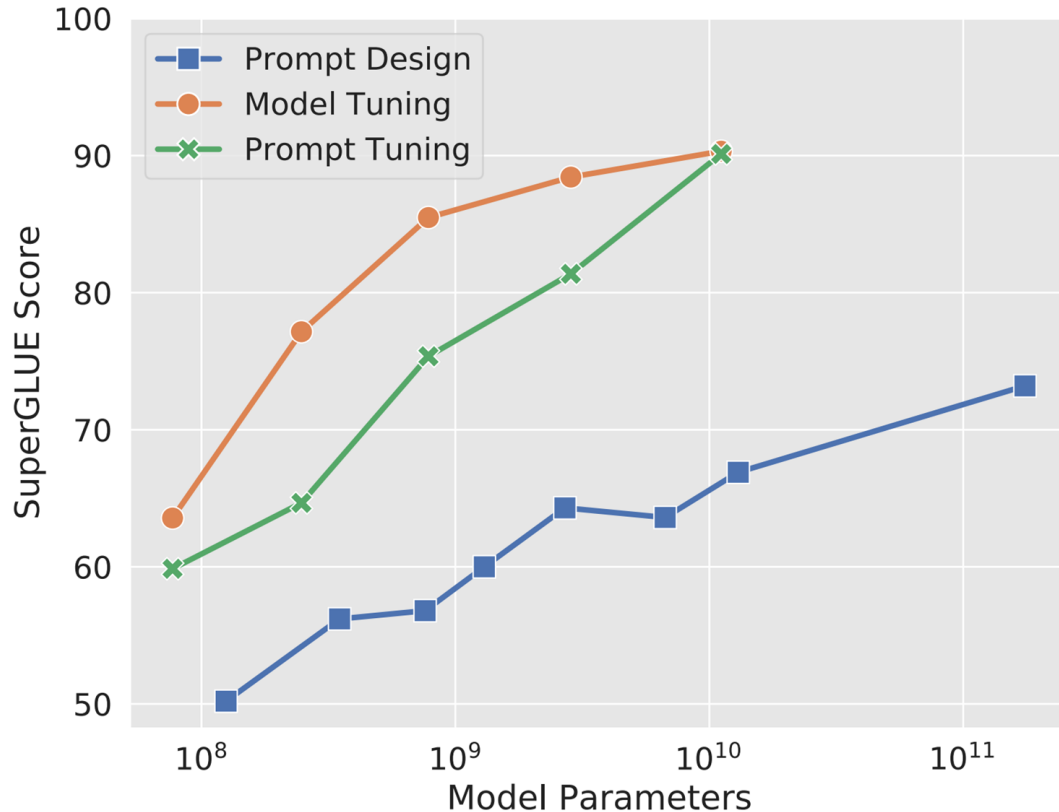
Training



Inference



(Soft) Prompt Tuning

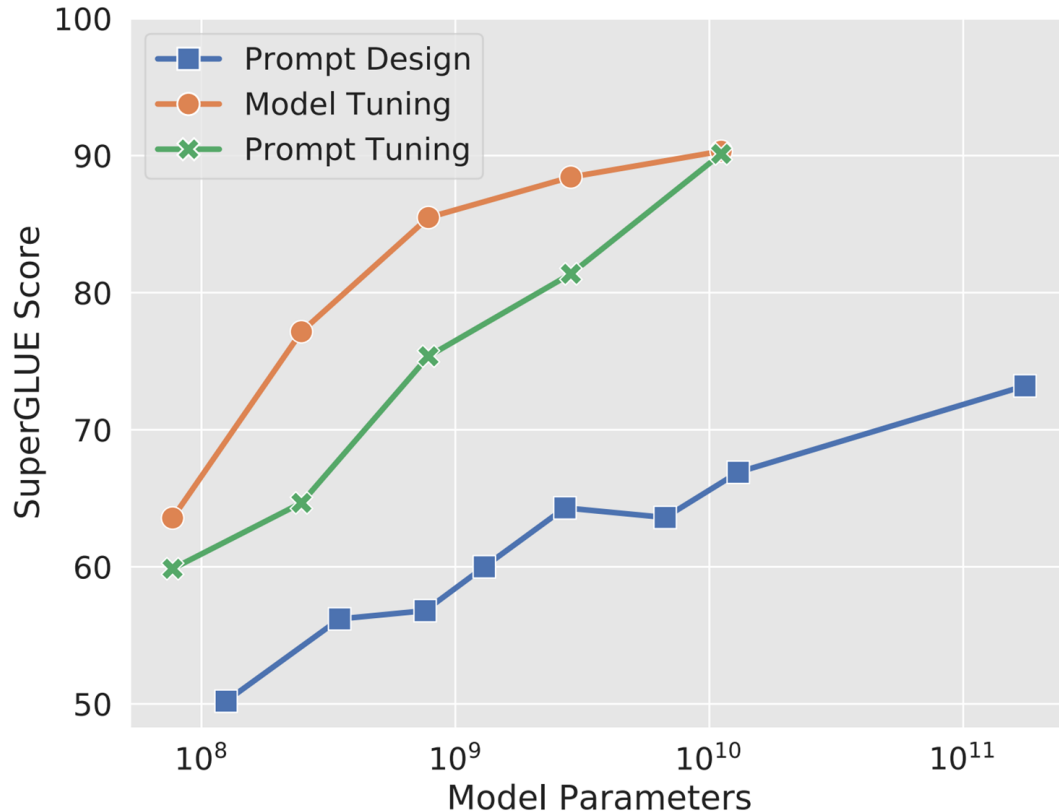


- Prompt tuning performs poorly at smaller model sizes and on harder tasks
[\[Mahabadi et al., 2021; Liu et al., 2022\]](#)

Prompt tuning vs standard full fine-tuning across T5 models of different sizes [\[Lester et al., 2021\]](#)



(Soft) Prompt Tuning



- Prompt tuning performs poorly at smaller model sizes and on harder tasks
[\[Mahabadi et al., 2021; Liu et al., 2022\]](#)
- increasing prompt length improves the performance and increasing beyond 20 tokens only yields marginal gains

Prompt tuning vs standard full fine-tuning across T5 models of different sizes [\[Lester et al., 2021\]](#)



(Soft) Prompt Tuning

| Dataset | Domain | Model | Prompt | Δ |
|------------|--------|-----------------------|-----------------------|----------|
| SQuAD | Wiki | 94.9 \pm 0.2 | 94.8 \pm 0.1 | -0.1 |
| TextbookQA | Book | 54.3 \pm 3.7 | 66.8 \pm 2.9 | +12.5 |
| BioASQ | Bio | 77.9 \pm 0.4 | 79.1 \pm 0.3 | +1.2 |
| RACE | Exam | 59.8 \pm 0.6 | 60.7 \pm 0.5 | +0.9 |
| RE | Wiki | 88.4 \pm 0.1 | 88.8 \pm 0.2 | +0.4 |
| DuoRC | Movie | 68.9 \pm 0.7 | 67.7 \pm 1.1 | -1.2 |
| DROP | Wiki | 68.9 \pm 1.7 | 67.1 \pm 1.9 | -1.8 |

F1 mean and stddev for models trained on SQuAD and evaluated on out-of-domain datasets from the MRQA 2019 shared task [\[Houlsby et al., 2019\]](#)

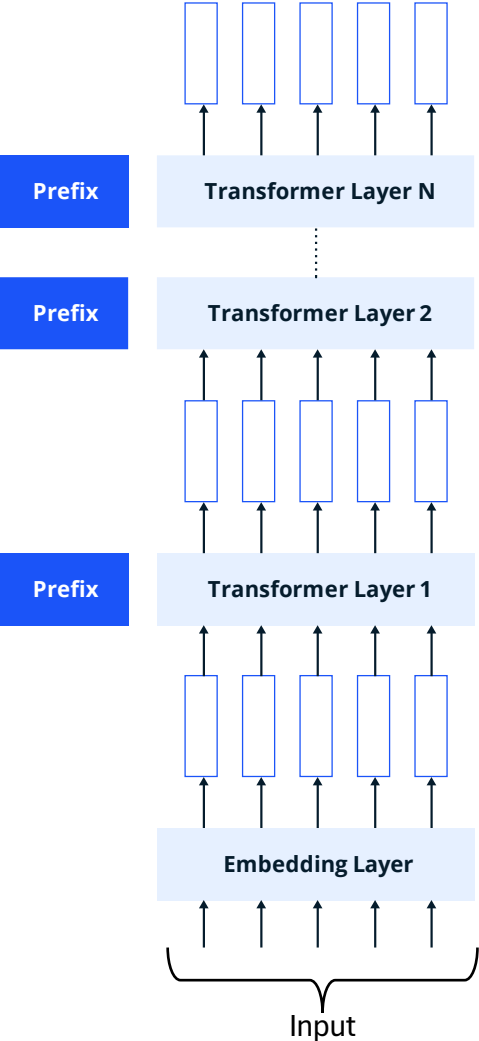


PEFT Techniques

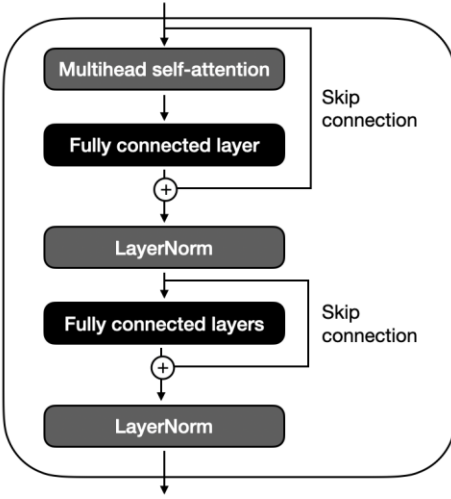
- (Soft) Prompt Tuning
- **Prefix Tuning**
- Adapters
- Low Rank Adaptation



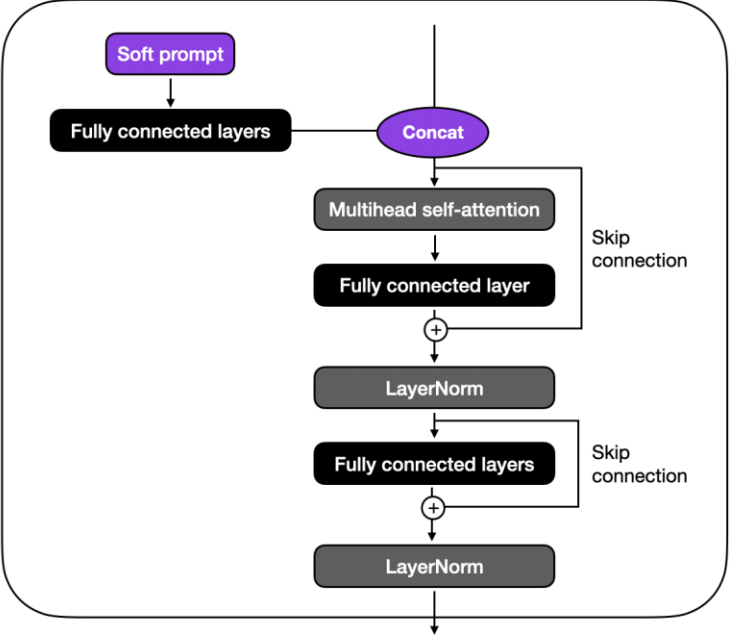
Prefix Tuning (Li & Liang 2021)



REGULAR TRANSFORMER BLOCK



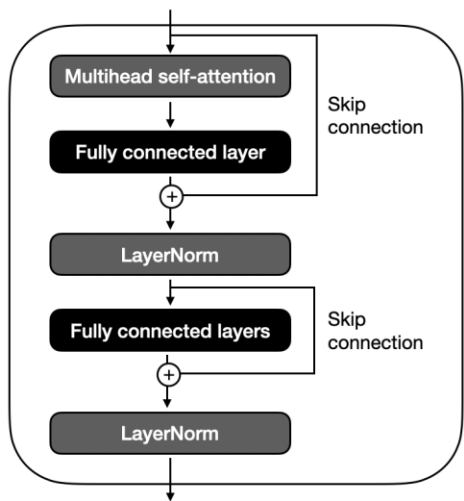
TRANSFORMER BLOCK WITH PREFIX



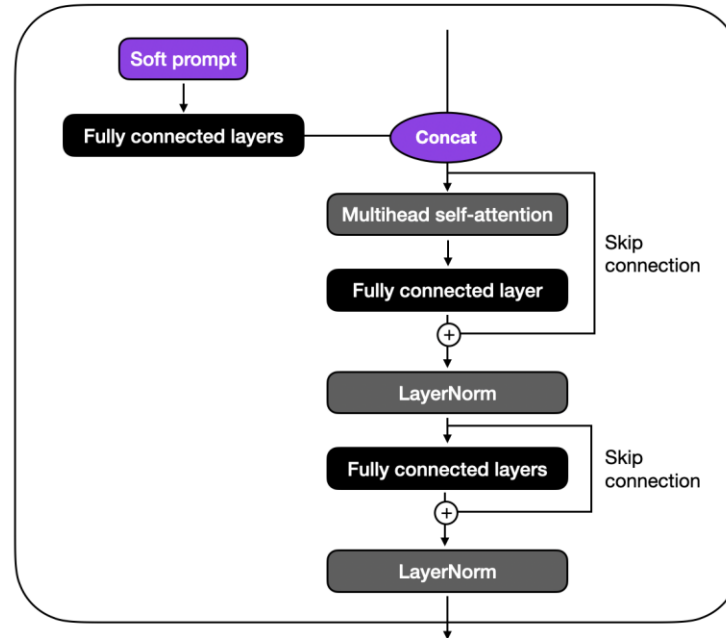
Prefix Tuning ([Li & Liang 2021](#))

Let P denote the prefix sequence and $|P|$ denote the length of the prefix sequence

REGULAR TRANSFORMER BLOCK



TRANSFORMER BLOCK WITH PREFIX



Prefix Tuning ([Li & Liang 2021](#))

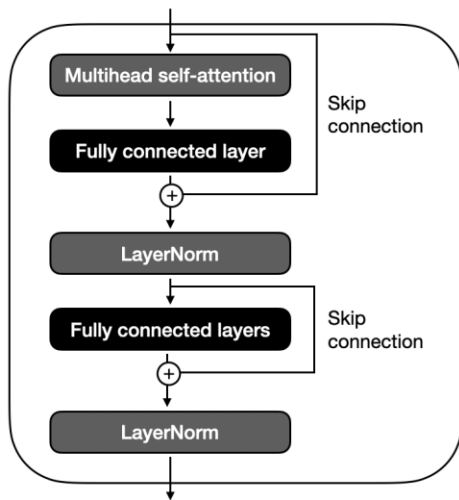
Let P denote the prefix sequence and $|P|$ denote the length of the prefix sequence

Let f_θ denote the prefix token p_i to hidden state h_i mapping

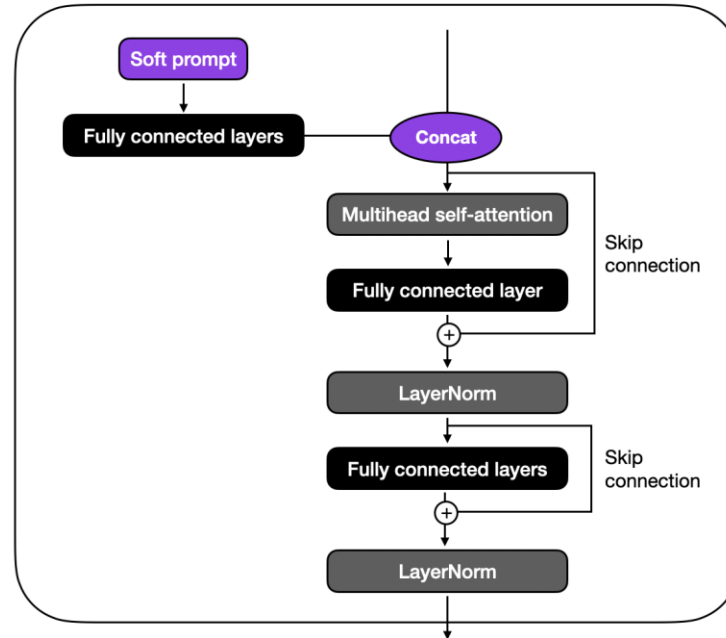
$$h_i = f_\theta(p_i)$$

f_θ dimensions are $|P| \times \text{dimension}(h_i)$

REGULAR TRANSFORMER BLOCK

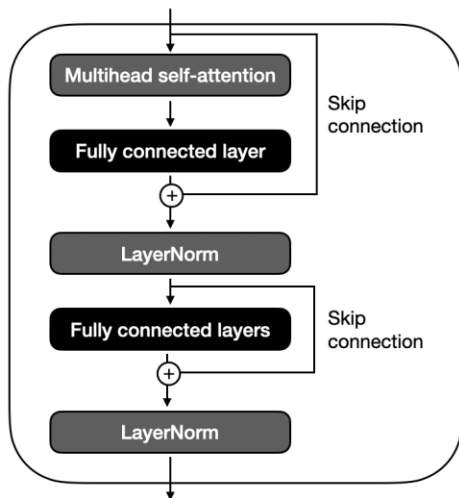


TRANSFORMER BLOCK WITH PREFIX

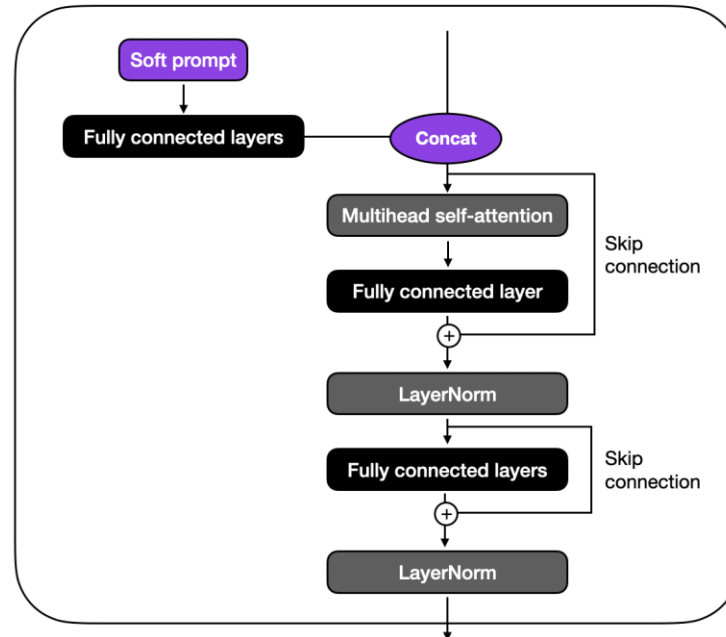


Prefix Tuning (Li & Liang 2021)

REGULAR TRANSFORMER BLOCK



TRANSFORMER BLOCK WITH PREFIX



Let P denote the prefix sequence and $|P|$ denote the length of the prefix sequence

Let f_θ denote the prefix token p_i to hidden state h_i mapping

$$h_i = f_\theta(p_i)$$

f_θ dimensions are $|P| \times \text{dimension}(h_i)$

Unstable Optimization Fix:

$$f_\theta(p_i) = MLP_\theta(f'_\theta(p_i))$$

- f'_θ is smaller than f_θ
- MLP_θ is a large FFN



Prefix Tuning ([Li & Liang 2021](#))

Experimental Setup:

- GPT-2 for table-to-text generation
- BART for summarization

Results:

- by learning only 0.1% of the parameters, prefix-tuning obtains comparable performance to full fine tuning
- extrapolates better to examples with topics unseen during training

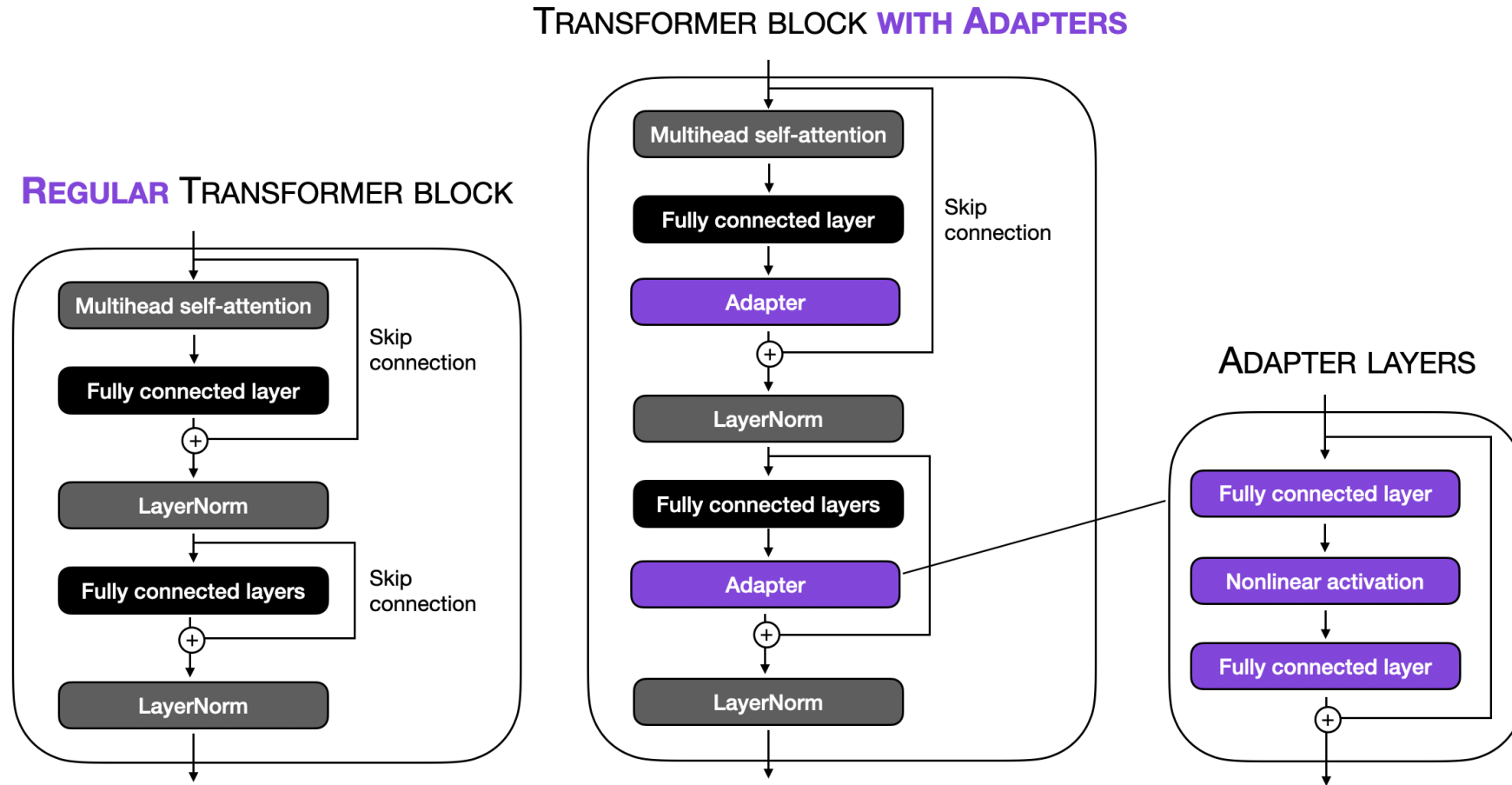


PEFT Techniques

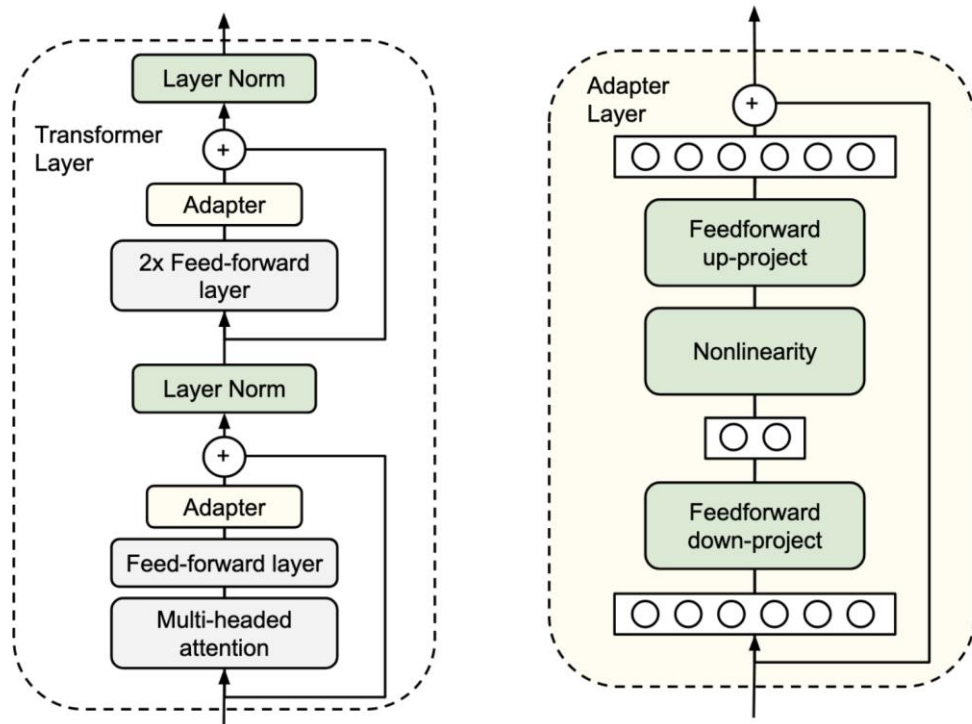
- (Soft) Prompt Tuning
- Prefix Tuning
- **Adapters**
- Low Rank Adaptation



Adapters (Houlsby et al 2019)



Adapters



Architecture of adapter module and its integration with the transformer [\[Houlsby et al., 2019\]](#)

Bottleneck Structure

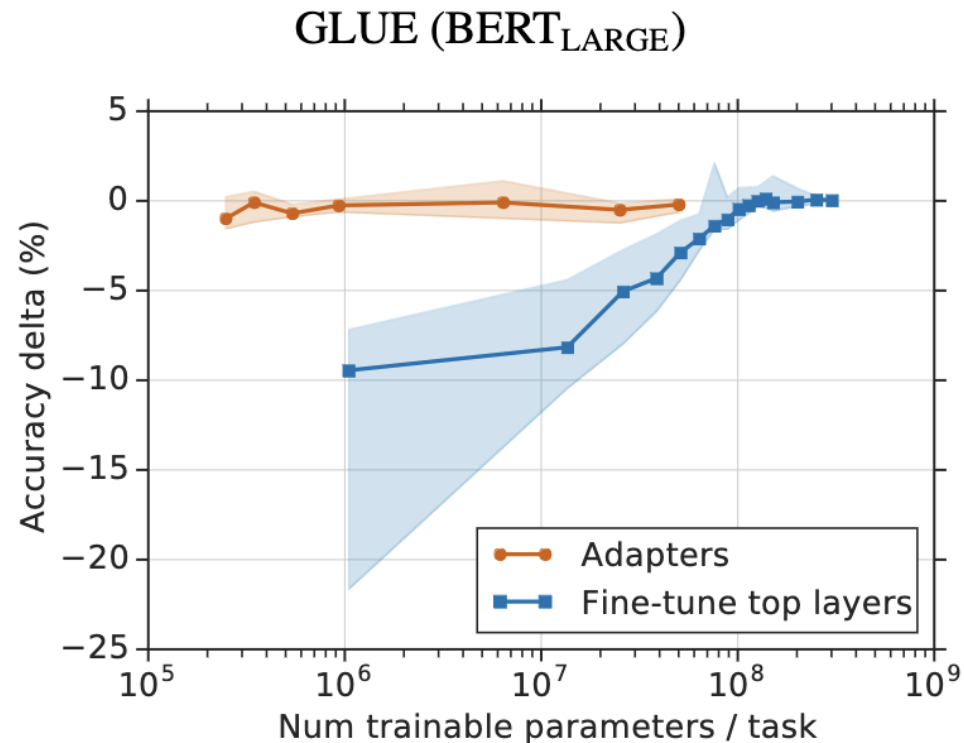
- significantly reduces the number of parameters
- reduces d -dimensional features into a smaller m -dimensional vector
- example: $d=1024$ and $m=24$
 - (1024×1024) requires 1,048,576 parameters
 - $2 * (1024 * 24)$ requires 49,152 parameters
- m determines the number of optimizable parameters and hence poses a parameter vs performance trade-off.

Inference Overhead

- Additional adapter in each transformer layer increases the inference latency



Adapters



- comparable to a fully finetuned BERT model while only requiring the training of 3.6% of the parameters
- when the adapter method is used to tune 3% of the model parameters, the method ties with prefix tuning of 0.1% of the model parameters

Accuracy versus the number of trained parameters, aggregated across tasks. The lines and shaded areas indicate the 20th, 50th, and 80th percentiles across tasks. [\[Houlsby et al., 2021\]](#)



PEFT Techniques

- (Soft) Prompt Tuning
- Prefix Tuning
- Adapters
- Low Rank Adaptation



Low Rank Composition

- [Li et al. \[2018\]](#) show that models can be optimized in a low-dimensional, randomly oriented subspace rather than the full parameter space

Standard fine-tuning:

$$\theta^{(D)} = \theta_0^{(D)} + \theta_{\tau}^{(D)}$$

Low-rank fine-tuning:

$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)}$$

A random $D \times d$
projection matrix



Intrinsic Dimensionality (ID)

- [Li et al. \[2018\]](#) refer to the minimum d where a model achieves within 90% of the full-parameter model performance, d_{90} as the intrinsic dimensionality of a task
- [Aghajanyan et al. \[2021\]](#) investigate the intrinsic dimensionality of different NLP tasks and pre-trained models
 - the method of finding the intrinsic dimension proposed by Li et al. (2018) is unaware of the layer-wise structure of the function parameterized by θ
 - Would require about 1TB of memory to store the projection matrix for even BERT based models.



Structure-Aware Intrinsic Dimension (SAID)

- [Aghajanyan et al. \[2021\]](#) also propose a structure-aware version
- Allocate one scalar λ_i per layer to learn layer-wise scaling:

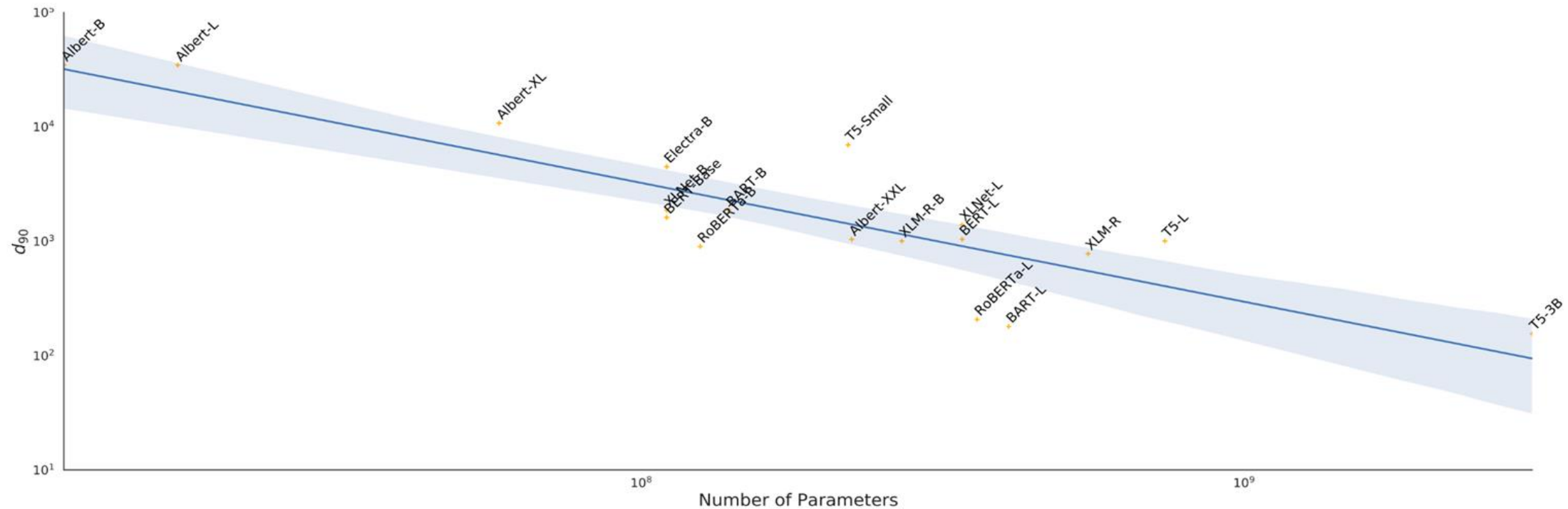
$$\theta_i^D = \theta_{0,i}^D + \lambda_i P(\theta^{d-m})_i$$

where m is the number of layers in the network

- However, storing the random matrices still requires a lot of extra space and is slow to train [\[Mahabadi et al., 2021\]](#)



Structure-Aware Intrinsic Dimension (SAID)



d_{90} on the MRPC dataset for models of different sizes



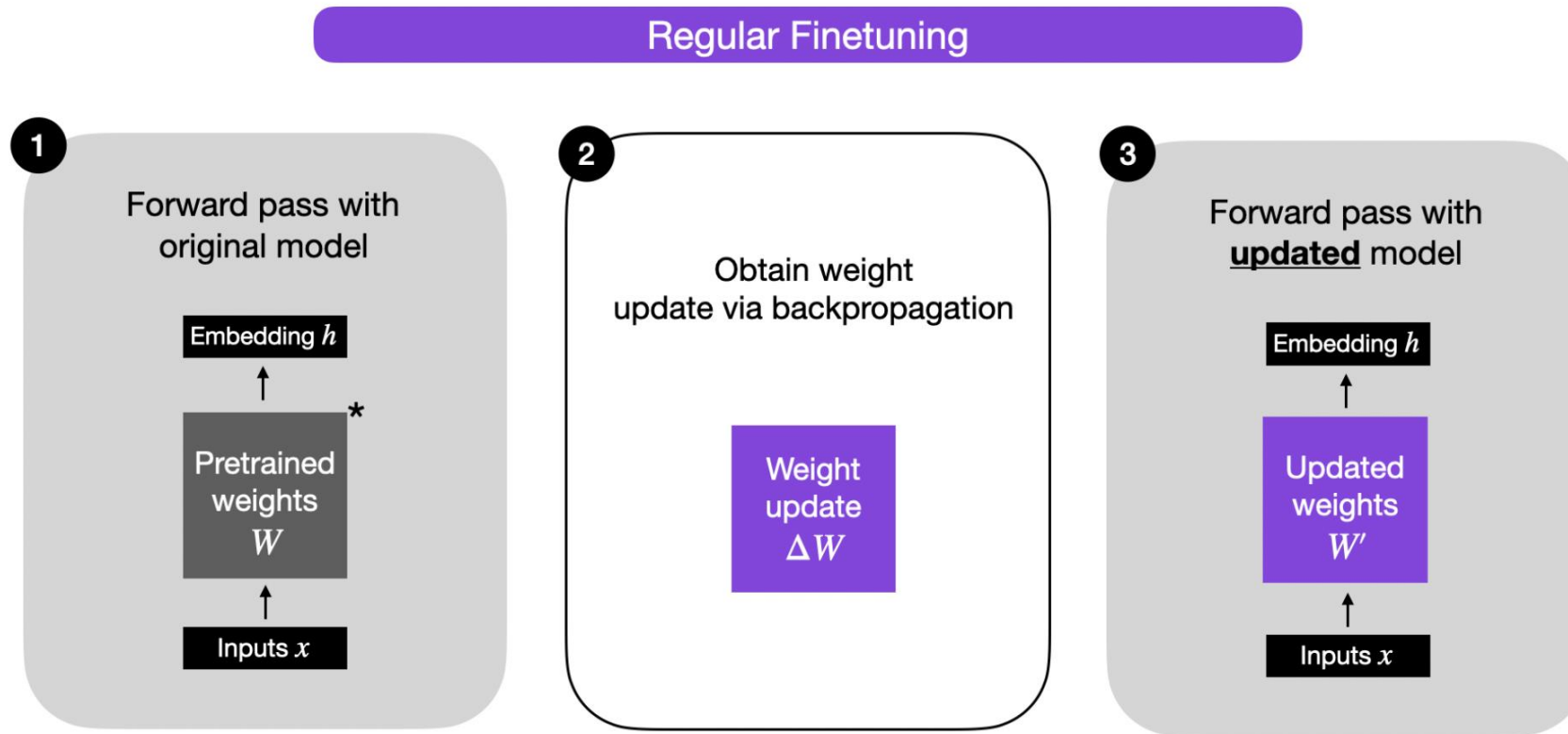
Structure-Aware Intrinsic Dimension (SAID)

| Model | SAID | | ID | |
|---------------|------------|------------|------|------------|
| | MRPC | QQP | MRPC | QQP |
| BERT-Base | 1608 | 8030 | 1861 | 9295 |
| BERT-Large | 1037 | 1200 | 2493 | 1389 |
| RoBERTa-Base | 896 | 896 | 1000 | 1389 |
| RoBERTa-Large | 207 | 774 | 322 | 774 |

Estimated d_{90} intrinsic dimension for a set of sentence prediction tasks and common pre-trained models.



Low Rank Adaptation (LoRA)

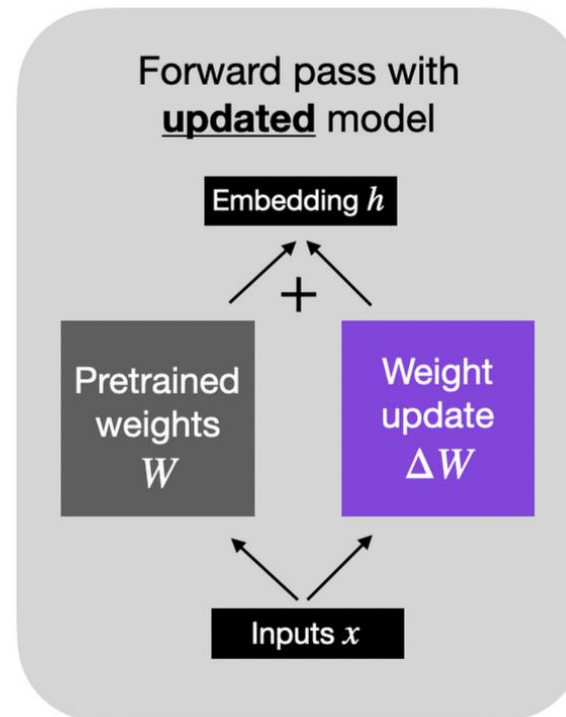


* The pretrained model could be any LLM, e.g., an encoder-style LLM (like BERT) or a generative decoder-style LLM (like GPT)



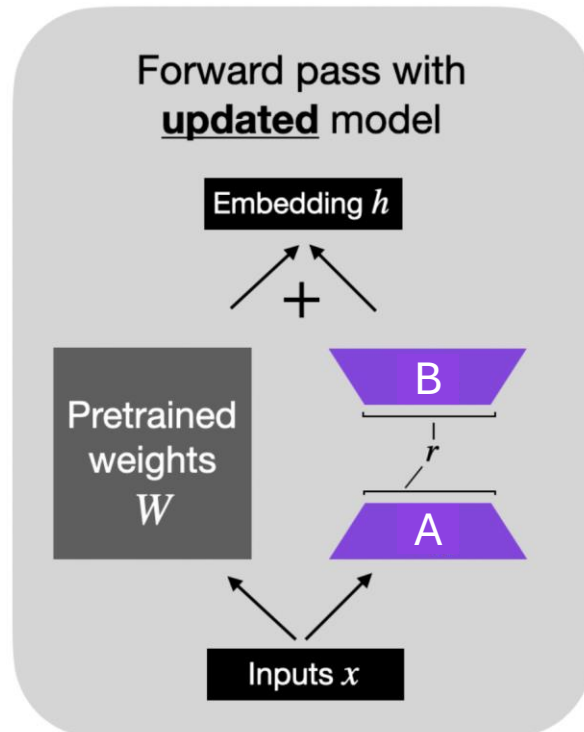
Low Rank Adaptation (LoRA)

Alternative formulation (regular finetuning)



Low Rank Adaptation (LoRA)

LoRA weights, W_A and W_B , represent ΔW



- Instead of learning a low-rank factorization via a random matrix P , we can learn the projection matrix directly - **if it is small enough**
- Better use of the network structure
- LoRA [Hu et al., 2022] learns two low-rank matrices A and B that are applied to the self-attention weights

$$h = W_0x + \Delta Wx = W_0x + BAx$$



LoRA

| Model&Method | # Trainable Parameters | WikiSQL | MNLI-m | SAMSum |
|-------------------------------|------------------------|-------------|-------------|-----------------------|
| | | Acc. (%) | Acc. (%) | R1/R2/RL |
| GPT-3 (FT) | 175,255.8M | 73.8 | 89.5 | 52.0/28.0/44.5 |
| GPT-3 (BitFit) | 14.2M | 71.3 | 91.0 | 51.3/27.4/43.5 |
| GPT-3 (PreEmbed) | 3.2M | 63.1 | 88.6 | 48.3/24.2/40.5 |
| GPT-3 (PreLayer) | 20.2M | 70.1 | 89.5 | 50.8/27.3/43.5 |
| GPT-3 (Adapter ^H) | 7.1M | 71.9 | 89.8 | 53.0/28.9/44.8 |
| GPT-3 (Adapter ^H) | 40.1M | 73.2 | 91.5 | 53.2/29.0/45.1 |
| GPT-3 (LoRA) | 4.7M | 73.4 | 91.7 | 53.8/29.8/45.9 |
| GPT-3 (LoRA) | 37.7M | 74.0 | 91.6 | 53.4/29.2/45.1 |

Performance of different adaptation methods on GPT-3 175B [\[Hu et al., 2021\]](#)



Effect of Apply LoRA to Weight Matrices in Transformers

| | # of Trainable Parameters = 18M | | | | | | |
|--------------------------|---------------------------------|------------|------------|------------|-----------------|-----------------|---------------------------|
| Weight Type Rank r | W_q 8 | W_k 8 | W_v 8 | W_o 8 | W_q, W_k 4 | W_q, W_v 4 | W_q, W_k, W_v, W_o 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 |

Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters [\[Hu et al., 2021\]](#)



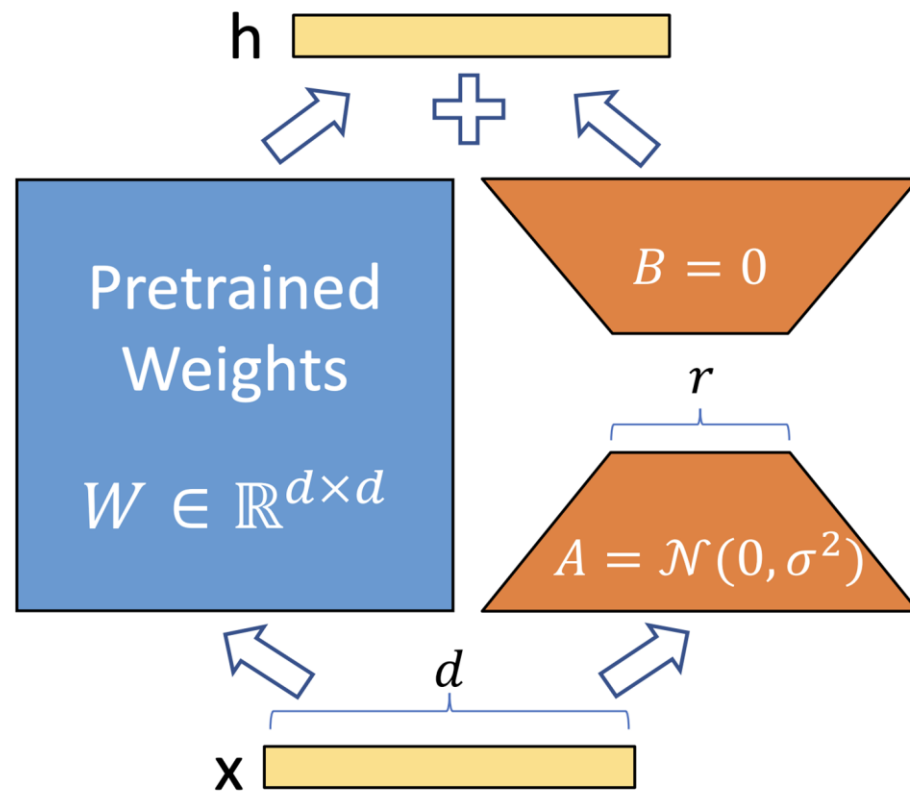
LoRA: Effect of rank on Performance

| | Weight Type | $r = 1$ | $r = 2$ | $r = 4$ | $r = 8$ | $r = 64$ |
|--------------------------|----------------------|---------|---------|---------|---------|----------|
| WikiSQL($\pm 0.5\%$) | W_q | 68.8 | 69.6 | 70.5 | 70.4 | 70.0 |
| | W_q, W_v | 73.4 | 73.3 | 73.7 | 73.8 | 73.5 |
| | W_q, W_k, W_v, W_o | 74.1 | 73.7 | 74.0 | 74.0 | 73.9 |
| MultiNLI ($\pm 0.1\%$) | W_q | 90.7 | 90.9 | 91.1 | 90.7 | 90.7 |
| | W_q, W_v | 91.3 | 91.4 | 91.3 | 91.6 | 91.4 |
| | W_q, W_k, W_v, W_o | 91.2 | 91.7 | 91.7 | 91.5 | 91.4 |

Validation accuracy on WikiSQL and MultiNLI with different rank [\[Hu et al., 2021\]](#)



LoRA Weights Initialization



- By setting B to zero, the product $\Delta W = BA$ initially equals zero. This preserves the behaviour of the original model at the start of fine-tuning
- Gaussian distribution helps ensure that the values in A are neither too large nor too biased in any direction, which could lead to disproportionate influence on the updates when B begins to change



Extensions of LoRA

- QLoRA [[Dettmers et al., 2023](#)]
 - backpropagates gradients through 4-bit quantized model for reducing memory usage
- LongLoRA [[Chen et al., 2024](#)]
 - sparse local attention to support longer context length during finetuning
- LoRA+ [[Hayou et al., 2024](#)]
 - different learning rates for the LoRA adapter matrices A and B improves finetuning speed
- DyLoRA [[Valipou et al., 2023](#)]
 - selects rank without requiring multiple runs of training



PEFT Techniques

- (Soft) Prompt Tuning
- Prefix Tuning
- Adapters
- Low Rank Adaptation

