Carnegie Mellon University

Parameter Efficient Tuning

11-667: LARGE LANGUAGE MODELS: METHODS AND APPLICATIONS

Announcements

- If you still don’t have a team for the project, come see me after class.
- For issues with HW1 Question 3.1, see my post recent note on Piazza.
- Do you want feedback on your project idea? Fill out the form to ask for feedback by EOD day.
- Chenyan’s office hours today are canceled.

You want to use an LM for some task for which you've collected a dataset of examples.

What do you do?

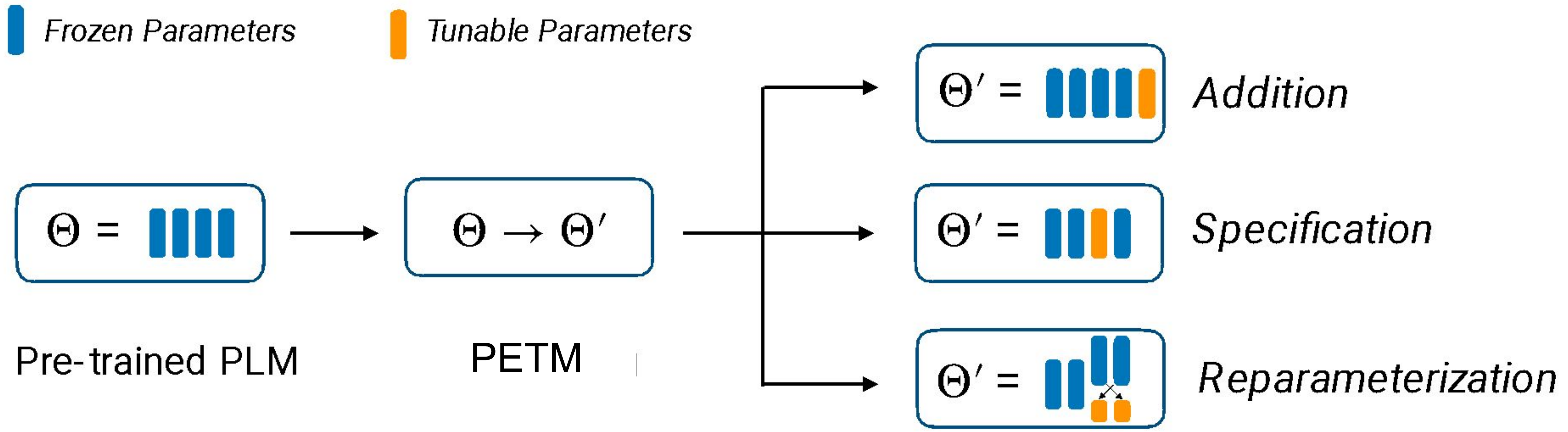
1. Prompt engineering
   1. Doesn't always work
   2. Tedious to find a good prompt
2. Finetune the full LM on your data
   1. Expensive
   2. Overfitting / catastrophic forgetting on small datasets
   3. Need to store one full set of model weights per task.
3. Parameter-efficient tuning

What is parameter efficient tuning?

Rather than finetuning the entire model, we finetune only small amounts of weights.

In this lecture, we'll break PETM techniques into roughly three categories.

1. **Addition:** What if we introduce additional trainable parameters to the neural network and just train those?
2. **Specification:** What if we pick a small subset of the parameters of the neural network and just tune those?
3. **Reparameterization:** What if we re-parameterize the model into something that is more efficient to train?



What is parameter efficient tuning?

Ideas we will cover

- AutoPrompt
- Prompt tuning
- LoRa
- $(IA)^3$

What if we introduce additional trainable parameters to the neural network and just train those?

Methods we'll discuss: prompt tuning, prefix tuning, adapters, and compacters

Intuition for Prompt Tuning

Prompt engineering sucks.

If we have a bunch of examples of the task, why can't we train a neural network to produce a good prompt for the task.

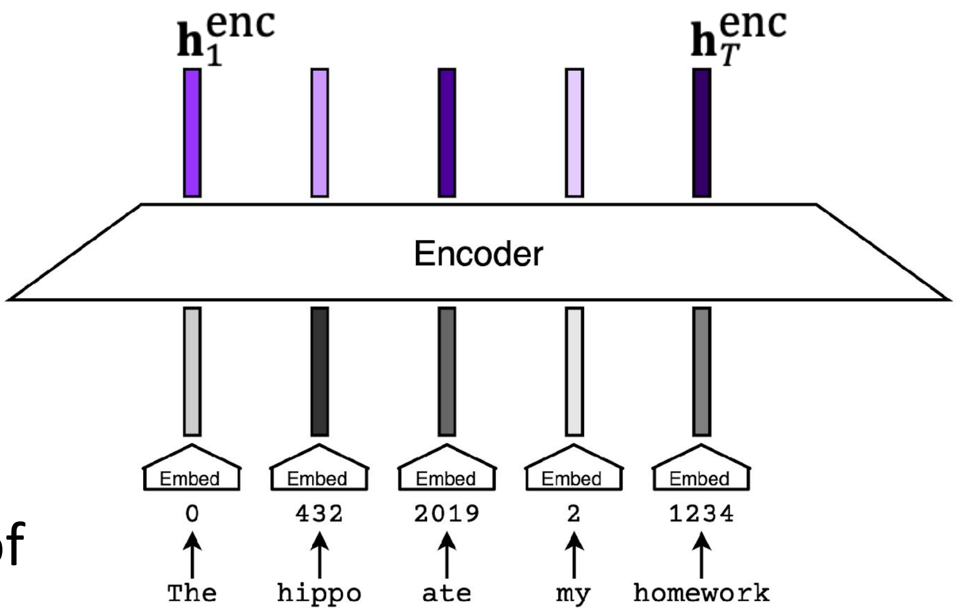
"The Power of Scale for Parameter-Efficient Prompt Tuning." Lester et al. 2021.

Prompt Tuning Method

What we want: a NN that is trained on examples of our task and produces a sequence of tokens we can prepend to our LLM query, causing the LLM to do the task in question.

In practice, optimizing over discrete tokens is hard.

What we do instead: a NN that outputs a sequence of *embeddings* we can prepend to our query to the LLM, causing the LLM to do the task.



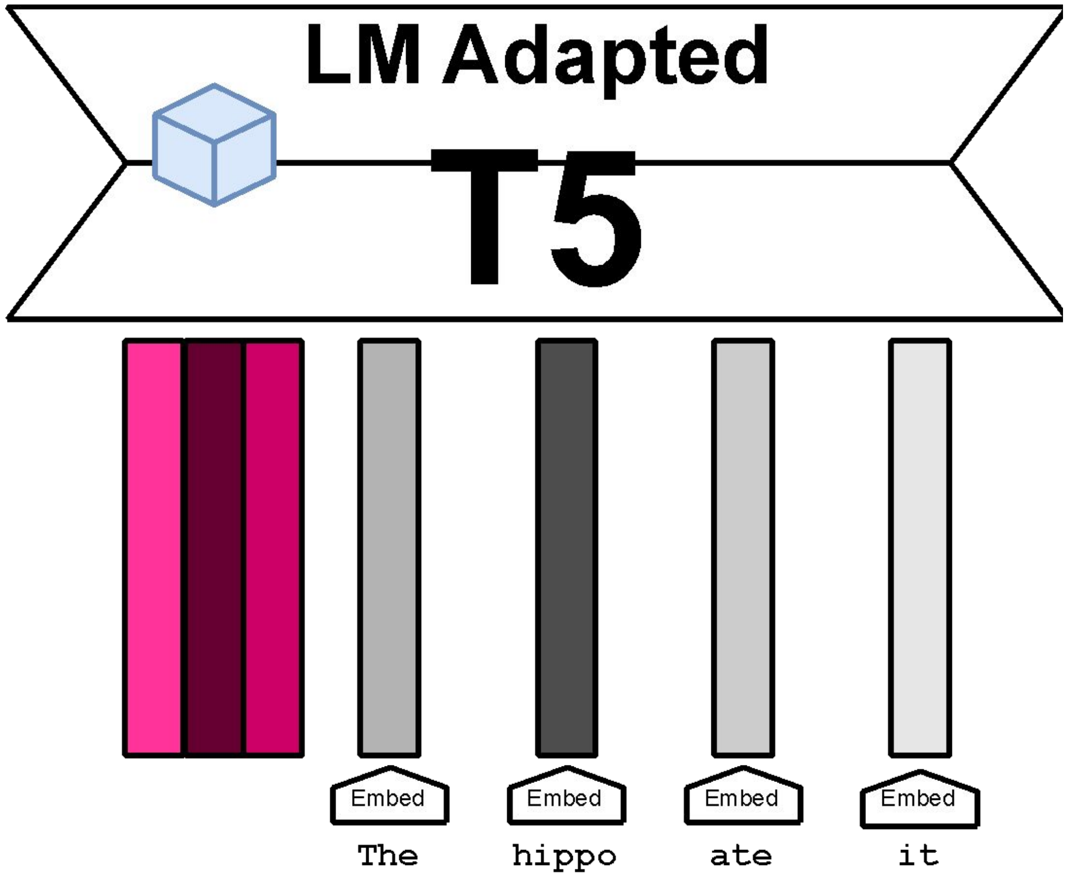
"The Power of Scale for Parameter-Efficient Prompt Tuning." Lester et al. 2021.

Prompt Tuning Method

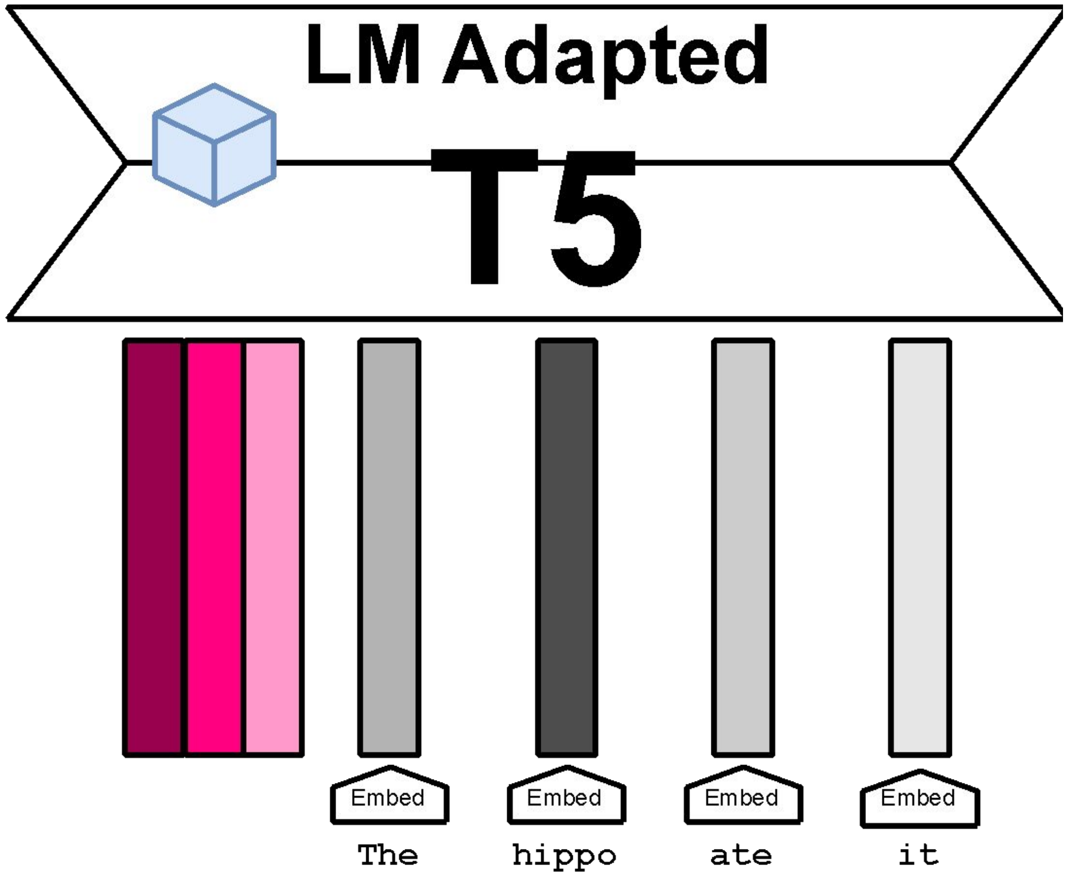
1. Finetune T5 to act a bit more like a traditional language model.
    1. This only needs to be done once, and empirically makes prompt tuning working better.
    2. This is probably because the span-corruption objective T5 was originally trained with isn't amenable to prompting.
2. Freeze the weights of T5. Set the first *k* input embeddings to be learnable.
    1. *k* is a hyperparameter up to the choice of the implementer.
3. Initialize the *k* learnable embeddings. Some options include:
    1. Random initialization
    2. Initialize to values drawn from the vocabulary embedding matrix
4. Train on your task specific data,

"The Power of Scale for Parameter-Efficient Prompt Tuning." Lester et al. 2021.

Prompt Tuning Method

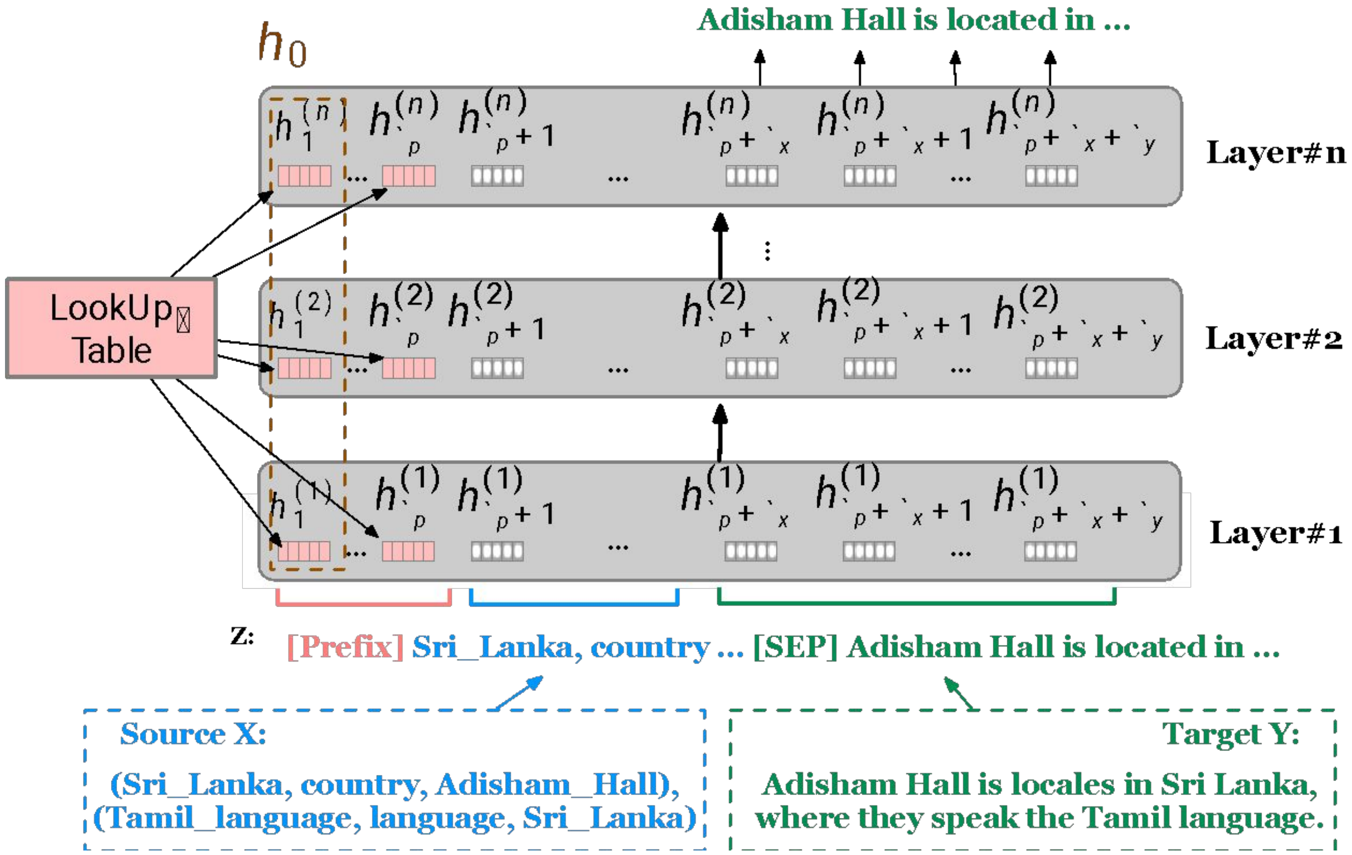


Prompt Tuning Method



Prefix Tuning

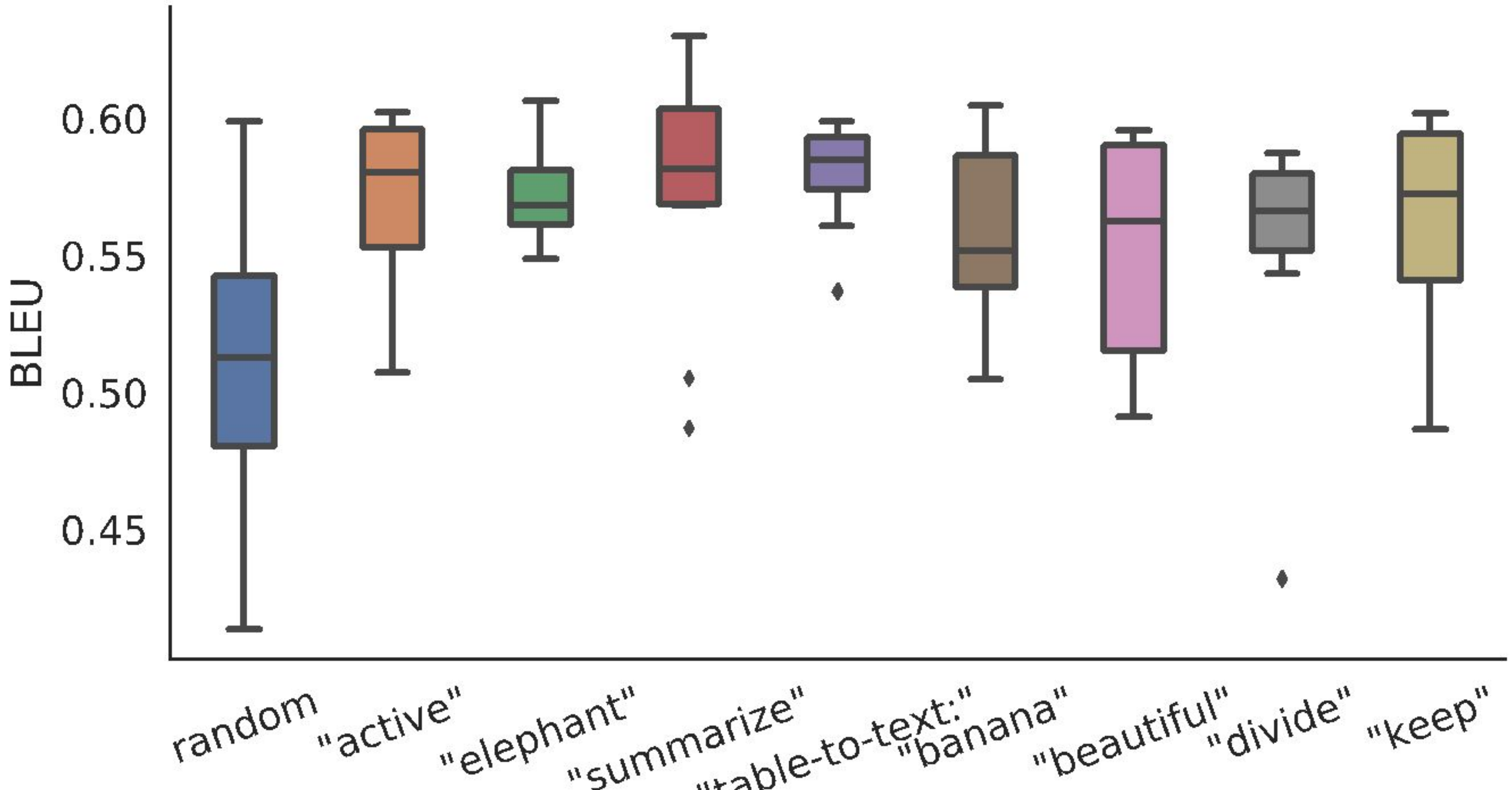
Same idea as prompt tuning, except that the learned prefix is appended not just to the input embeddings, but rather at each layer of the Transformer.



"Prefix-Tuning: Optimizing Continuous Prompts for Generation." Li and Liang. 2021.

How to initialize the prefix?

Initializing to real embeddings seems to work better than random initialization.



"Prefix-Tuning: Optimizing Continuous Prompts for Generation." Li and Liang. 2021.

Advantages of Prefix/Prompt Tuning

- The learned embeddings tend to be relatively small, just a few megabytes or less.
  - It is cheap to keep around one set of embeddings per task.
- The pre-trained LLM can be loaded into memory (such as on a server), and at inference time, the appropriate task-specific embeddings can be passed in.
  - Example use case: User customization

Pitfalls of Prefix and Prompt Tuning

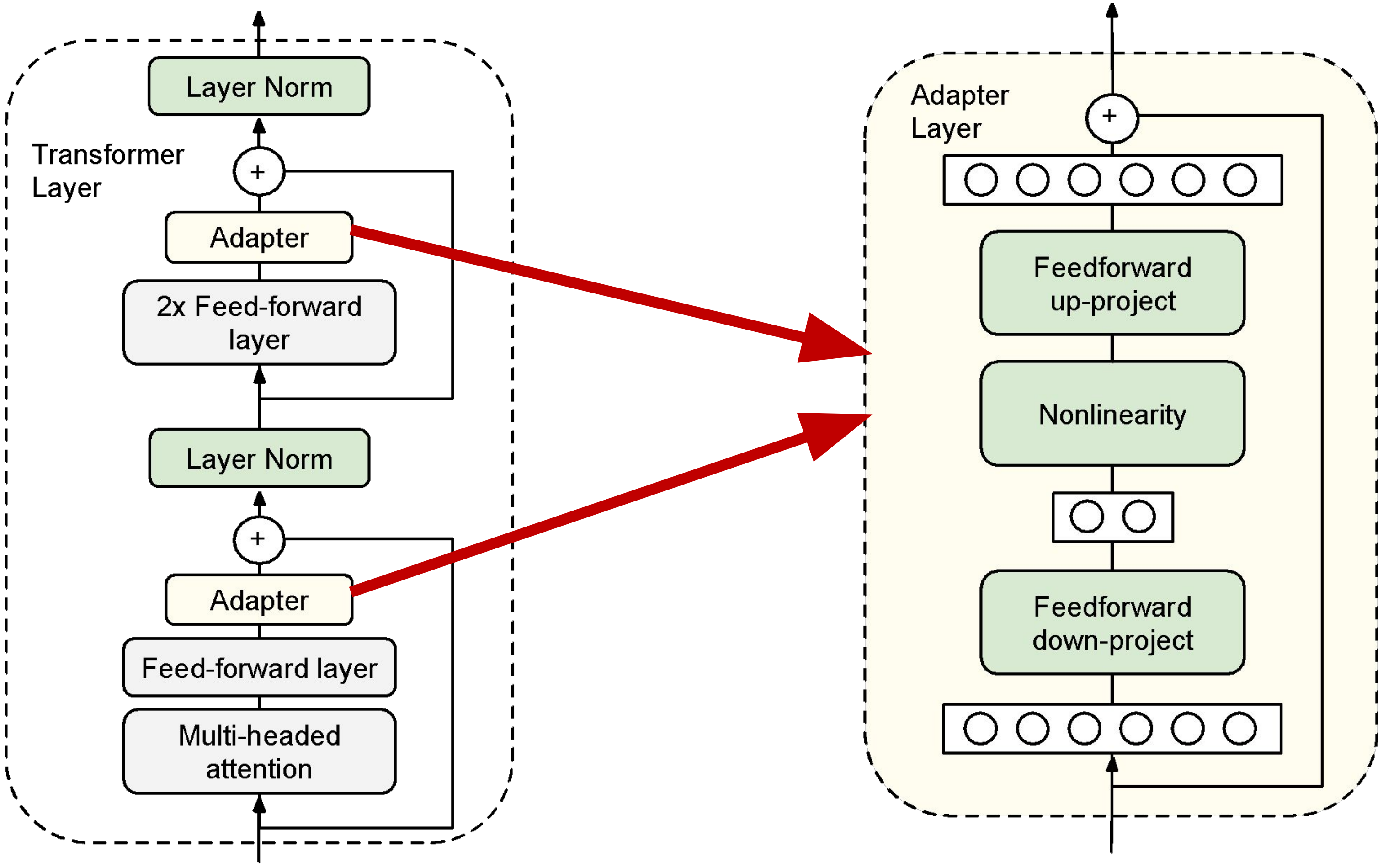
- In practice, these methods tend to converge significantly slower than full parameter fine-tuning.
- Unclear what the best prefix length is for any particular task.
  - Every sequence position you “spend” on the prefix is one less you have for your actual task.
- Learned embeddings are not very interpretable.

What are adapters?

- **Adapters** are new modules are added between layers of a pre-trained network.
- The original model weights are fixed; just the adapter modules are tuned.
- The adapters are initialized such that the output of the adapter-inserted module resembles the original model.

"Parameter-Efficient Transfer Learning for NLP." Houslby et al. 2019.

What are adapters?



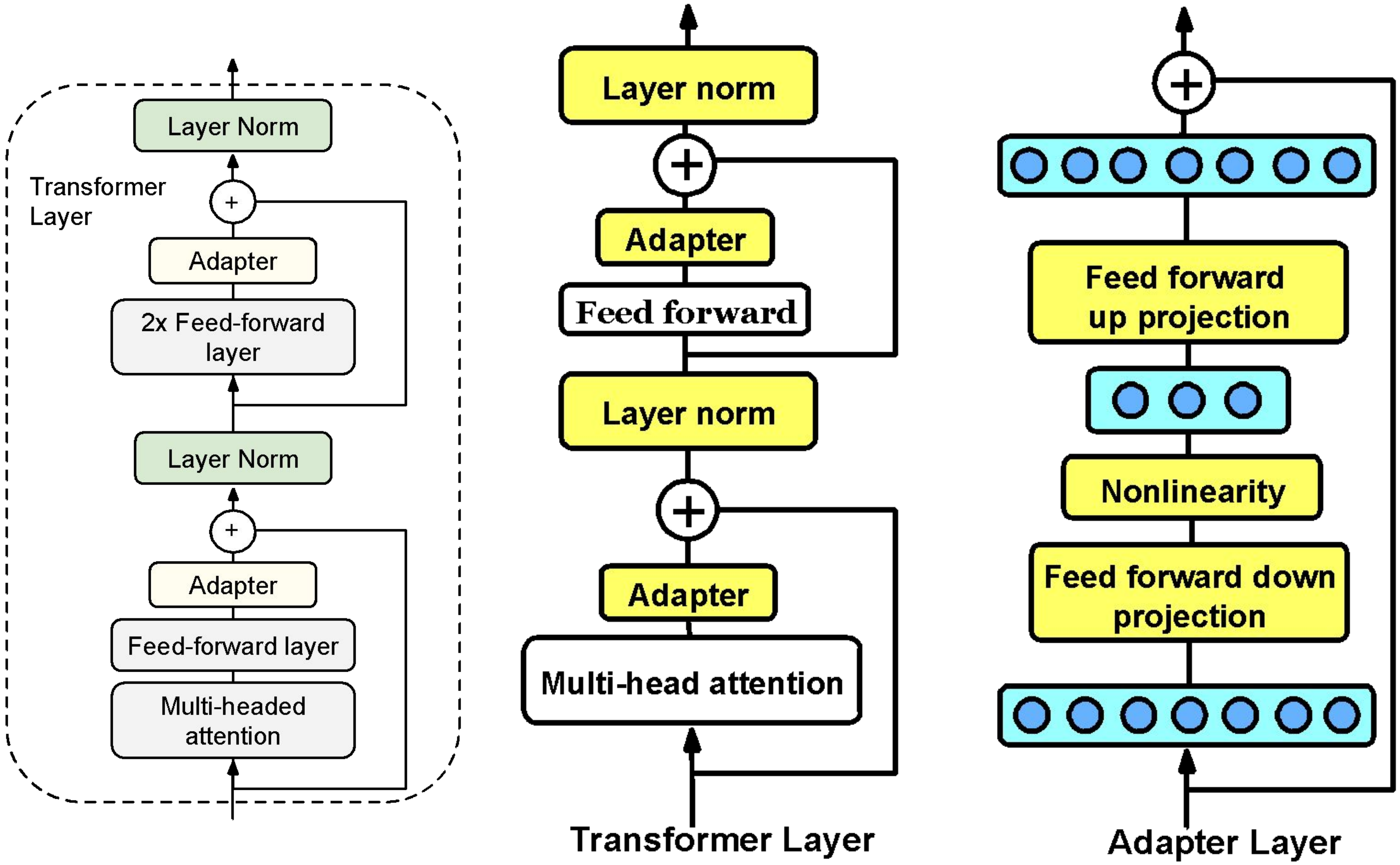
"Parameter-Efficient Transfer Learning for NLP." Houslby et al. 2019.

What are compacters?

- **Compacters** are an extension of adapters which aim to make the technique even more efficient.
- Adapters are standard fully connected layers.
  - Linear project to lower dimension followed by nonlinearity followed by projection back up to original dimension.
  - $y = \boldsymbol{W_2}\ \mathrm{GELU}(\boldsymbol{W_1 x} + \boldsymbol{b_1}) + \boldsymbol{b_2}$
- The compacter replaces the fully connected layer with a. parameterized hypercomplex multiplication layer.
  - Each $\boldsymbol{W}$ is learned as a sum of *n* Kronecker products
  - *n* is a user-specified hyperparameter.
- Compacters reduce the number of parameters in the adapter layer to 1/*n* without harming the performance.

"COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers." Mahabadi et al. 2021.

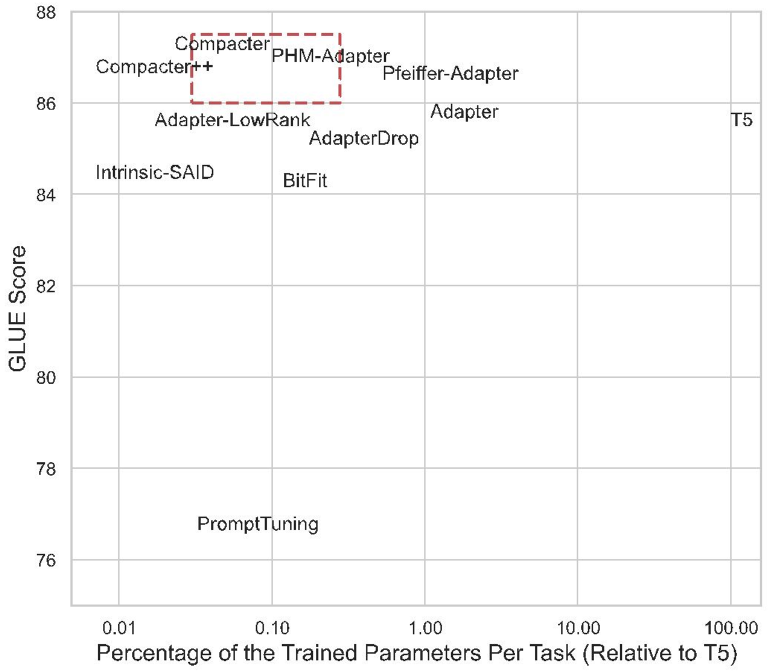
What are compacters?



“COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers.” Mahabadi et al. 2021.

There have been many other extensions to adapters which we won't discuss in this class.

Name & Refs	Method	#Params
SEQUENTIAL ADAPTER Houlsby et al. (2019)	$\text{LayerNorm}(\mathbf{X}+\mathbf{H}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{H}(\mathbf{X})))$	$L \times 2 \times (2d_h d_m)$
COMPACTER Mahabadi et al. (2021a)	$\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{F}(\mathbf{X})))$	$L \times 2 \times (2(d_h + d_m))$
ADAPTERDROP Rücklé et al. (2021)	$\text{ADT}(\mathbf{X}) = \mathbf{X} + \sigma(\mathbf{X}\mathbf{W}_{d_h \times d_m})\mathbf{W}_{d_m \times d_h}, \quad \sigma = \text{activation}$	$(L-n) \times 2 \times (2d_h d_m)$
PARALLEL ADAPTER He et al. (2022)	$\text{LayerNorm}(\mathbf{X}+\mathbf{H}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{X})+\mathbf{H}(\mathbf{X}))$ $\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{X})+\mathbf{F}(\mathbf{X}))$ $\text{ADT}(\mathbf{X}) = \sigma(\mathbf{X}\mathbf{W}_{d_h \times d_m})\mathbf{W}_{d_m \times d_h}, \quad \sigma = \text{activation}$	$L \times 2 \times (2d_h d_m)$
ADAPTERBIAS	$\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\text{ADT}(\mathbf{X})+\mathbf{F}(\mathbf{X}))$ $\text{ADT}(X) = \mathbf{X}\mathbf{W}_{d_h \times 1}\mathbf{W}_{1 \times d_h}$	$L \times 2 \times d_h$



"COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers." Mahabadi et al. 2021.

Advantages of Adapter-Based Methods

- Have been shown to be quite effective in multi-task settings.
  - There are methods for training task-specific adapters and then combining the to leverage the cross-task knowledge (see AdapterFusion).
- Tend to be faster to tune than full model finetuning.
- Possibly more robust to adversarial perturbations of the tuning data than full model finetuning.
  - (see robust transfer learning paper)

Pitfalls or Adapter Methods

- Adding in new layers means making inference slower.
- It also makes the model bigger (possibly harder to fit on available GPUs).
- Adapter layers need to be processed sequentially at inference time, which can break model parallelism.

What if we pick a small subset of the parameters of the neural network and just tune those?

Methods we'll discuss: layer freezing, BitFit, diff pruning

Layer Freezing

- Research has shown that earlier layers of the Transformer tend to capture linguistic phenomena and basic language understand; later layers are where the task-specific learning happens.
- This means we should be able to learn new tasks by freezing the earlier layers and just tuning the later ones.

BitFit: Bias-terms Fine-tuning

- Only tune the bias terms and final classification layer (if doing classification)
- Recall the equations for multi-head attention

$$
\begin{aligned}
\mathbf{Q}^{m,\ell}(\mathbf{x}) &= \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell} \\
\mathbf{K}^{m,\ell}(\mathbf{x}) &= \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell} \\
\mathbf{V}^{m,\ell}(\mathbf{x}) &= \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}
\end{aligned}
$$

  - ℓ is the layer index
  - m is the attention head index
  - Only the bias terms (shown in red) are updated.

“BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models.” Zaken et al. 2022.

Intuition for DiffPruning

- In prior methods we discussed, the choice of what parameters to freeze and what parameters to tune was done manually.
- Why not learn this instead?
- Main idea:
  - For each parameter, finetune a learnable "delta" which gets added to the original parameter value.
  - Use an L_0-norm penalty to encourage sparsity in the deltas.

"Parameter-Efficient Transfer Learning with Diff Pruning." Guo et al. 2021.

What if we re-parameterize the model into something that is more efficient to train?

Methods we'll discuss: LoRa, $(IA)^3$

Intuition for Re-Parameterizing the Model

Finetuning has a low **intrinsic dimension**, that is, the minimum number of parameters needed to be modified to reach satisfactory performance is not very large.

This means, we can reparameterize a subset of the original model parameters with low-dimensional proxy parameters, and just optimize the proxy.

What do we mean by **intrinsic dimension**?

- An objective function’s intrinsic dimension measures the minimum number of parameters needed to reach a satisfactory solution to the objective.
- Can also be thought of as the lowest dimensional subspace in which one can optimize the original objective function to within a certain level of approximation error.

What do we mean by **intrinsic dimension**?

- Suppose we have model parameters θ^D
  - D is the number of parameters.
- Instead of optimizing θ^D, we could instead optimize a smaller set of parameters θ^d where $d \ll D$.
- This is done through clever factorization:
  - $\theta^D = \theta_0^D + P(\theta^d)$ where $P\colon \mathbb{R}^d \to \mathbb{R}^D$
  - *P* is typically a linear projection: $\theta^D = \theta_0^D + \theta^d M$
- If you are interested in this, there are a lot more details in the paper.

"Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning." Aghajanyan et al. 2021.

LoRA: **L**ow **R**ank **A**daption

- Intuition: It's not just the model weights that are low rank, *updates* to the model weights are also low-rank.
- LoRA freezes the pretrained model weights and injects trainable rank decomposition matrices into each layer.
- Like DiffPruning, we are learning a delta to apply to each weight. In the case of LoRA, this delta has been re-paramaterized to be lower dimension than the original model parameters.
- In practice, LoRA only adapts the attention weights and keeps the rest of the Transformer as-is.

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

$(IA)^3$: **I**nfused **A**dapter by **I**nhibiting and **A**mplifying **I**nner **A**ctivations

- Intended to be an improved over LoRA
- Three goals:
  - must add or update as few parameters as possible to avoid incurring storage and memory costs
  - should achieve strong accuracy after training on only a few examples of a new tasks
  - must allow for mixed-task batches
- Main idea:
  - Rescale inner activations with lower-dimensional learned vectors, which are injected into the attention and feedforward modules
- Main differences from LoRA:
  - LoRA learns low-rank **updates** to the attention weights
  - $(IA)^3$ learns injectable vectors.

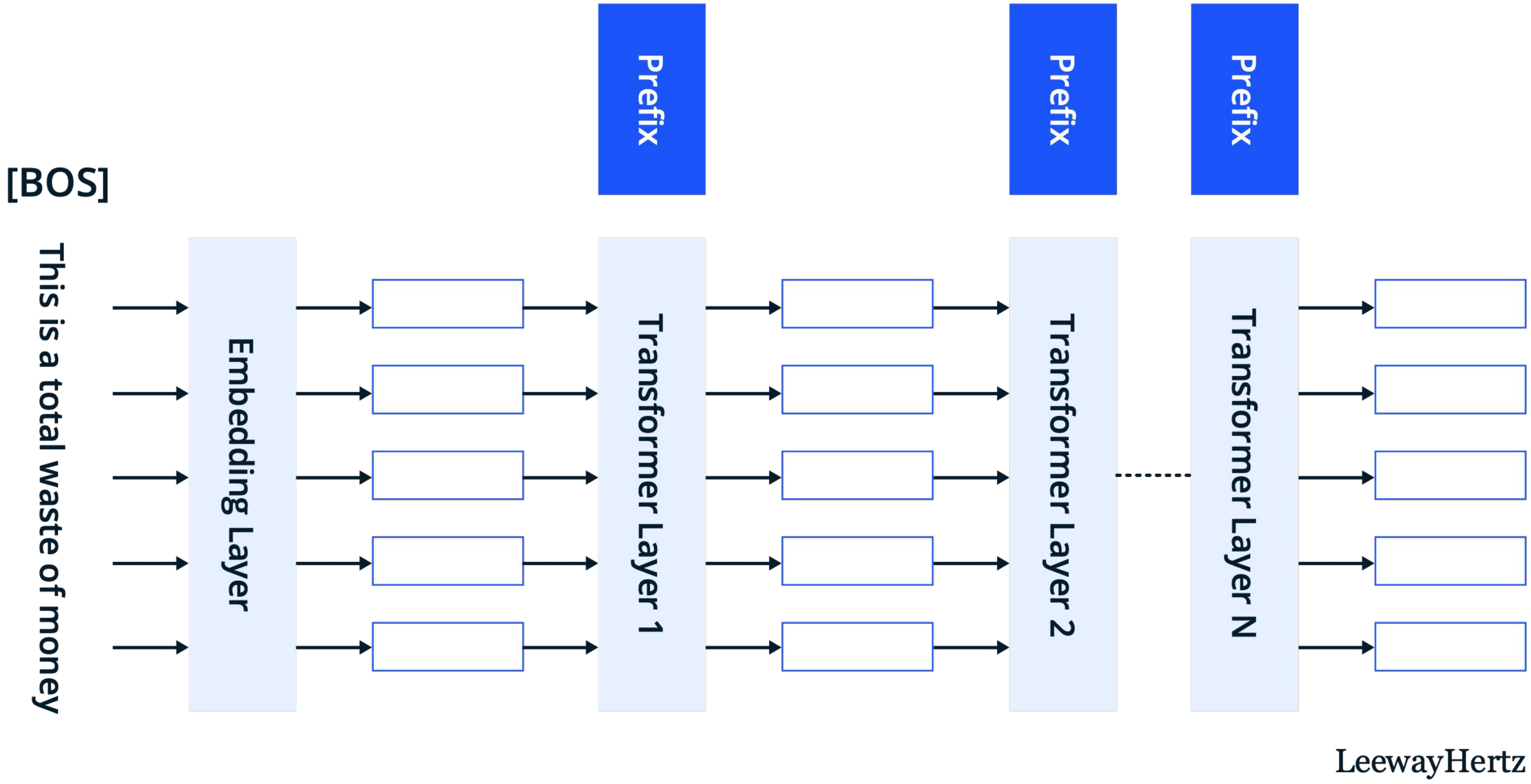
“Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning.” Liu at al. 2022.

Advantages of Re-Paramaterization Methods

- Training tends to be more memory-efficient, since we only need to calculate gradients and maintain optimizer state for a small number of parameters.
- These methods are faster to tune than standard full model finetuning.
- It is straight-forward to swap between tasks by swapping in and our just the tuned weights.

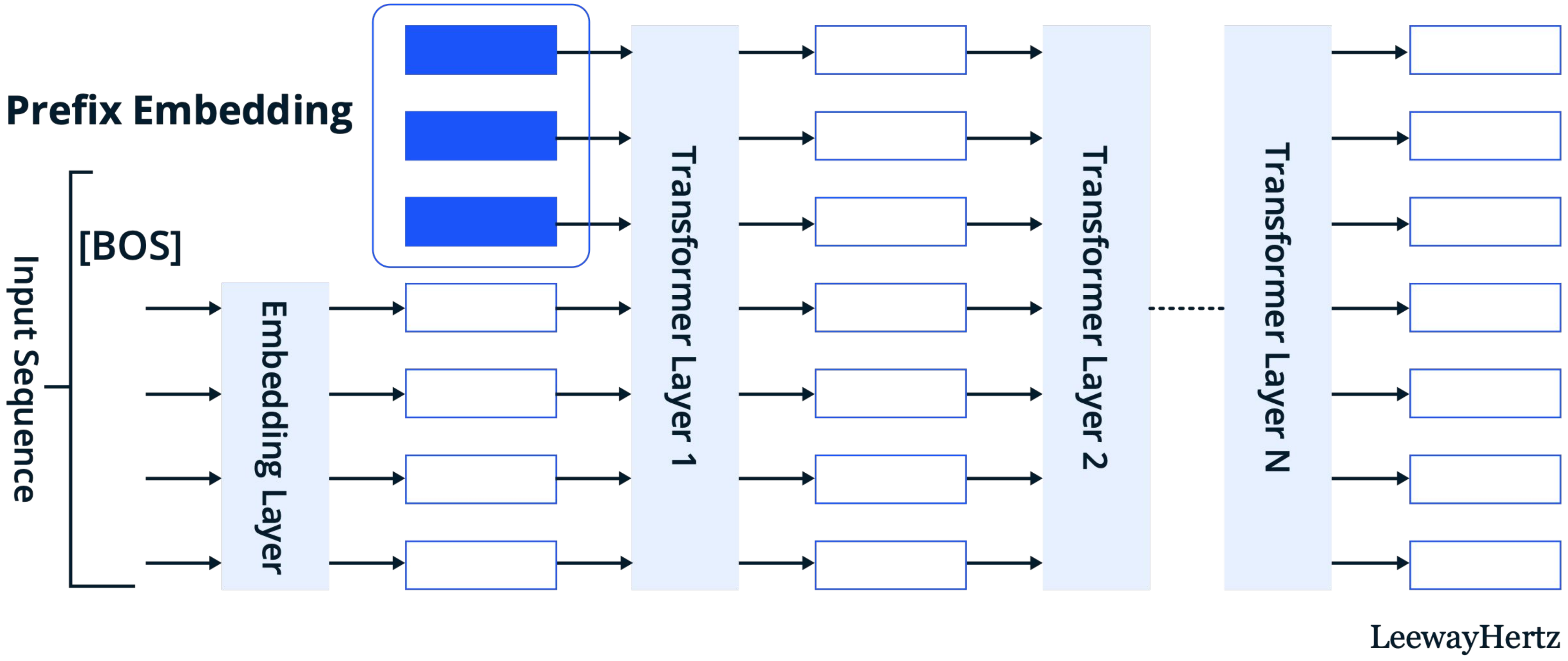
Summary

Prefix Tuning



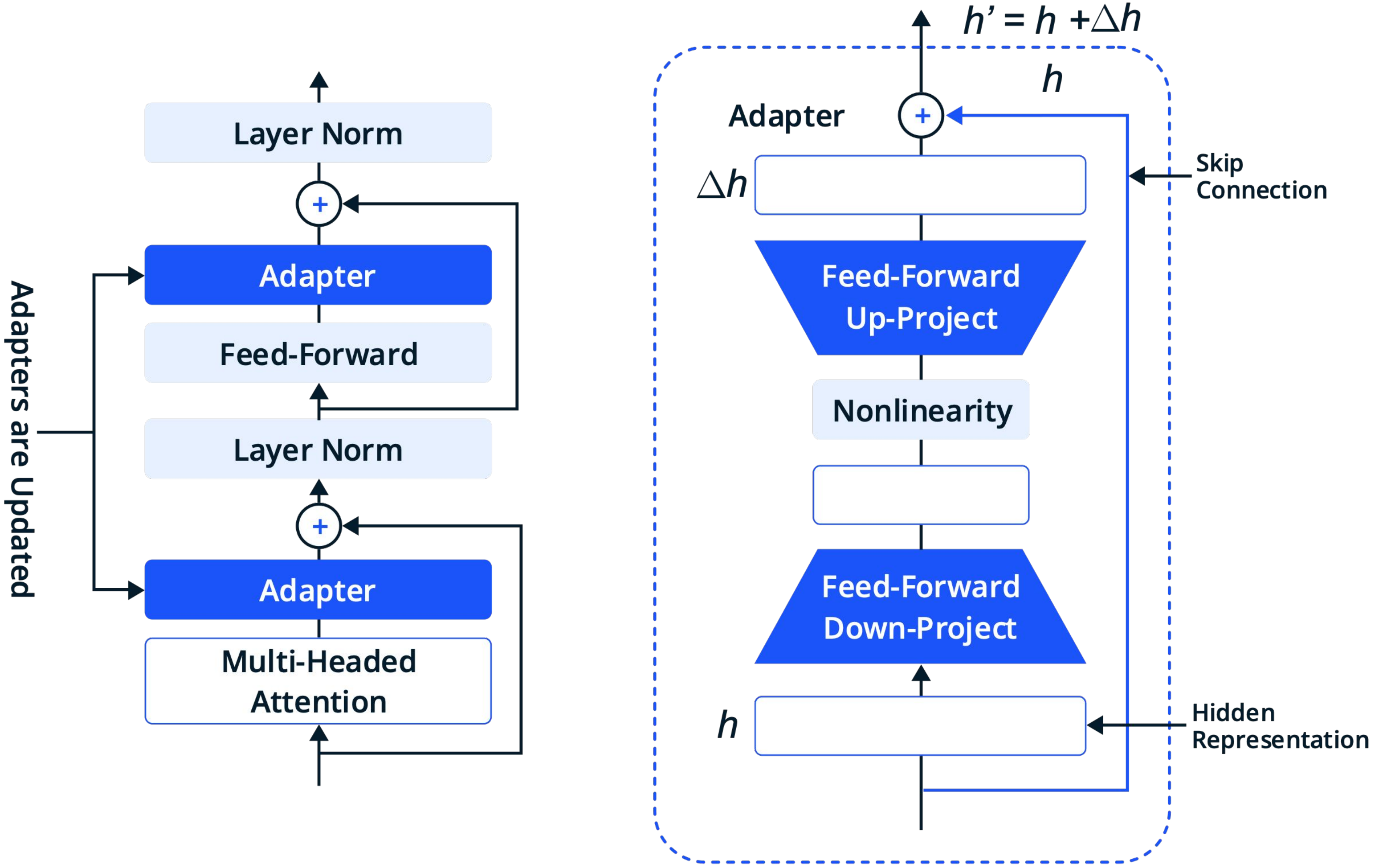
https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Prompt Tuning



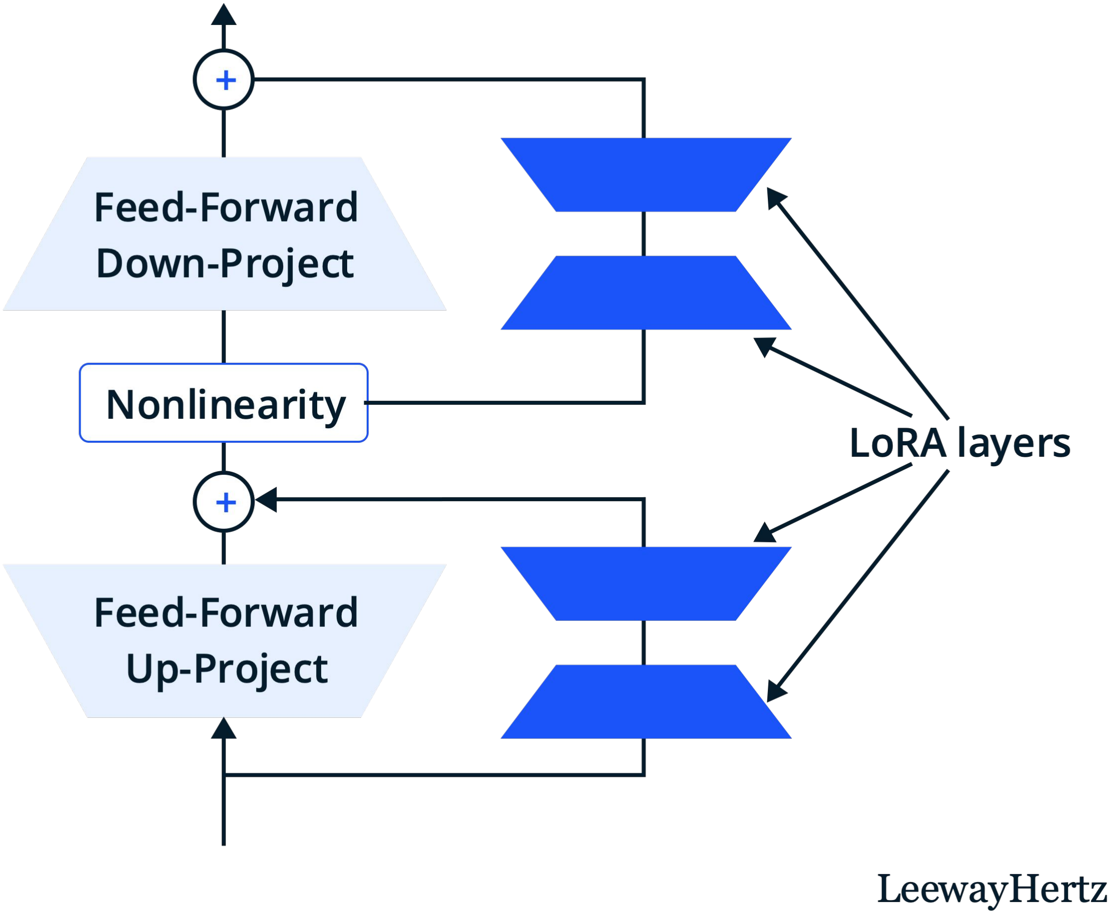
https://www.leewayhertz.com/parameter-efficient-fine-tuning/

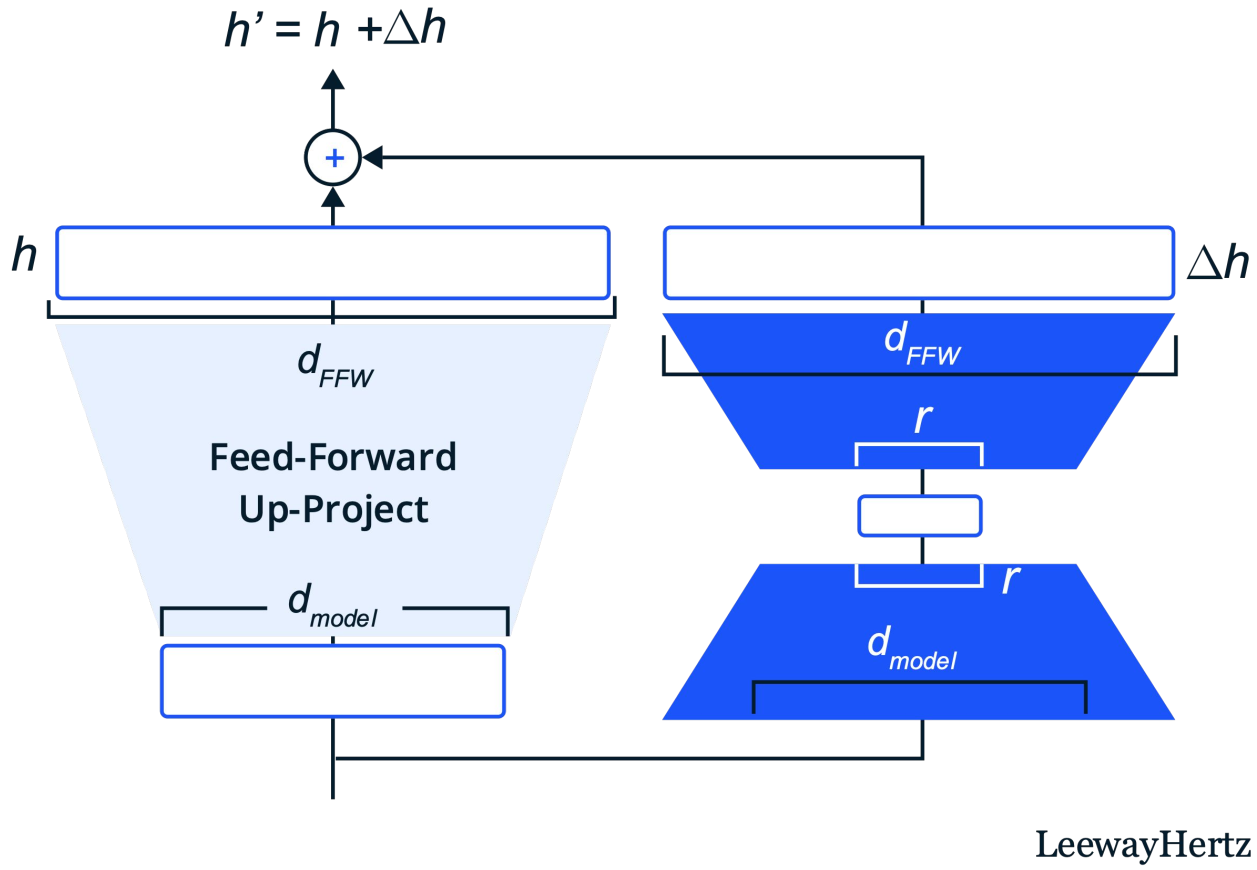
Adapters: adding in new trainable layers



https://www.leewayhertz.com/parameter-efficient-fine-tuning/

LoRA: injecting trainable rank decomposition matrices





https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Training Subset of Existing Parameters

- Manually chose what to tune
  - Just tune the last few layers
  - Just tune the bias terms (BitFit)
- Learn which parameters need to be tuned (DiffPruning)

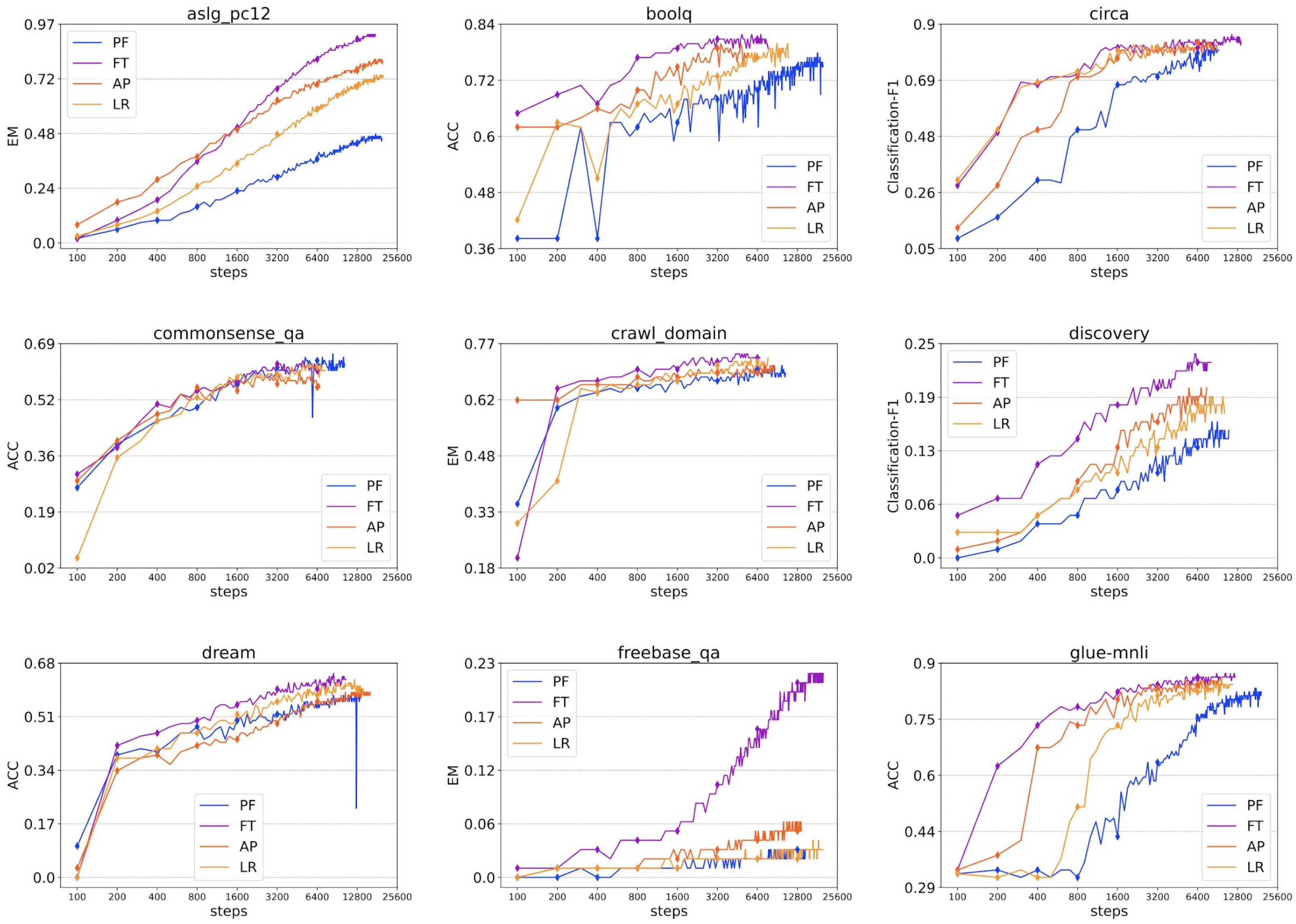
Summary

Name & Refs	Method	#Params
SEQUENTIAL ADAPTER Houlsby et al. (2019)	$\text{LayerNorm}(\mathbf{X}+\mathbf{H}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{H}(\mathbf{X})))$	$L \times 2 \times (2d_h d_m)$
COMPACTER Mahabadi et al. (2021a)	$\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{F}(\mathbf{X})))$	$L \times 2 \times (2(d_h + d_m))$
ADAPTERDROP Rücklé et al. (2021)	$\text{ADT}(\mathbf{X}) = \mathbf{X} \ + \ \sigma(\mathbf{X}\mathbf{W}_{d_h \times d_m})\mathbf{W}_{d_m \times d_h}, \quad \sigma = \text{activation}$	$(L-n) \times 2 \times (2d_h d_m)$
PARALLEL ADAPTER He et al. (2022)	$\text{LayerNorm}(\mathbf{X}+\mathbf{H}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{X})+\mathbf{H}(\mathbf{X}))$ $\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\mathbf{X}+\text{ADT}(\mathbf{X})+\mathbf{F}(\mathbf{X}))$ $\text{ADT}(\mathbf{X}) = \sigma(\mathbf{X}\mathbf{W}_{d_h \times d_m})\mathbf{W}_{d_m \times d_h}, \quad \sigma = \text{activation}$	$L \times 2 \times (2d_h d_m)$
ADAPTERBIAS	$\text{LayerNorm}(\mathbf{X}+\mathbf{F}(\mathbf{X})) \rightarrow \text{LayerNorm}(\text{ADT}(\mathbf{X})+\mathbf{F}(\mathbf{X}))$ $\text{ADT}(X) = \mathbf{X}\mathbf{W}_{d_h \times 1}\mathbf{W}_{1 \times d_h}$	$L \times 2 \times d_h$
PREFIX-TUNING Li & Liang (2021)	$\mathbf{H}_i = \text{ATT}(\mathbf{X}\mathbf{W}_q^{(i)}, [\text{MLP}_k^{(i)}(\mathbf{P}'_k) : \mathbf{X}\mathbf{W}_k^{(i)}], [\text{MLP}_v^{(i)}(\mathbf{P}'_v) : \mathbf{X}\mathbf{W}_v^{(i)}])$ $\text{MLP}^{(i)}(\mathbf{X}) = \sigma(\mathbf{X}\mathbf{W}_{d_m \times d_m})\mathbf{W}^{(i)}_{d_m \times d_h}$ $P' = \mathbf{W}_{n \times d_m}$	$n \times d_m + d_m^2$ $+ L \times 2 \times d_h d_m$
LORA Hu et al. (2021a)	$\mathbf{H}_i = \text{ATT}(\mathbf{X}\mathbf{W}_q^{(i)}, \text{ADT}_k(\mathbf{X}) + \mathbf{X}\mathbf{W}_k^{(i)}, \text{ADT}_v(\mathbf{X}) + \mathbf{X}\mathbf{W}_v^{(i)})$ $\text{ADT}(\mathbf{X}) = \mathbf{X}\mathbf{W}_{d_h \times d_m}\mathbf{W}_{d_m \times d_h}$	$L \times 2 \times (2d_h d_m)$
BITFIT Zaken et al. (2021)	$f(\mathbf{X}) \rightarrow f(\mathbf{X}) + \mathbf{B}, \quad \text{for all function } f$	$L \times (7 \times d_h + d_m)$

"Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models." Ding et al. 2023.

Results

If you have the resources, full fine-tuning tends to work the best.



PF: prefix tuning

FT: full fine-tuning

AP: adapter

LR: LoRA

This survey overall found:

Full fine-tuning >

LoRA >

Adapters >

Prefix Tuning >

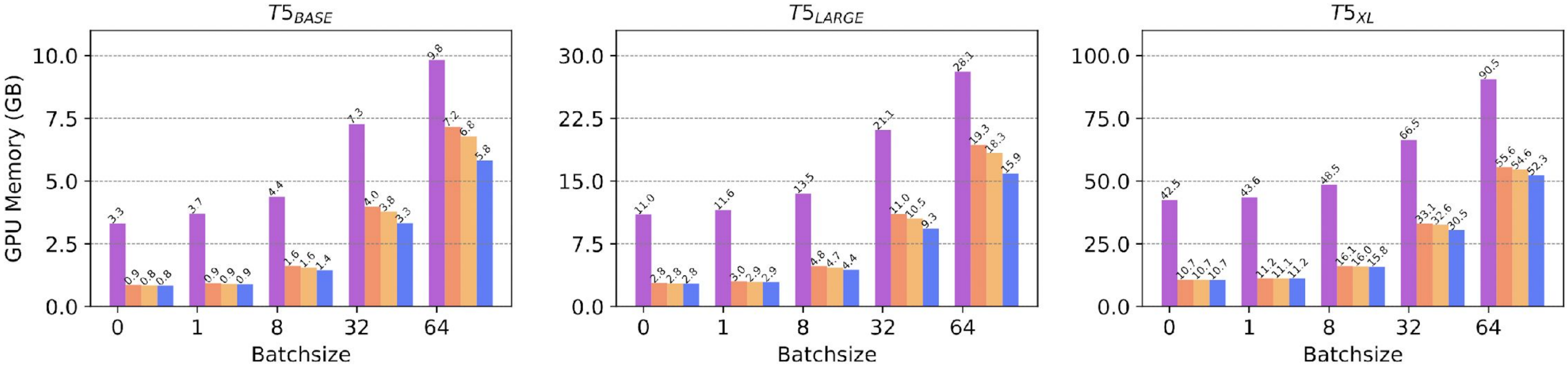
Prompt Tuning

In terms of performance.

Plots for many more tasks can be found in the paper.

“Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models.” Ding et al. 2023.

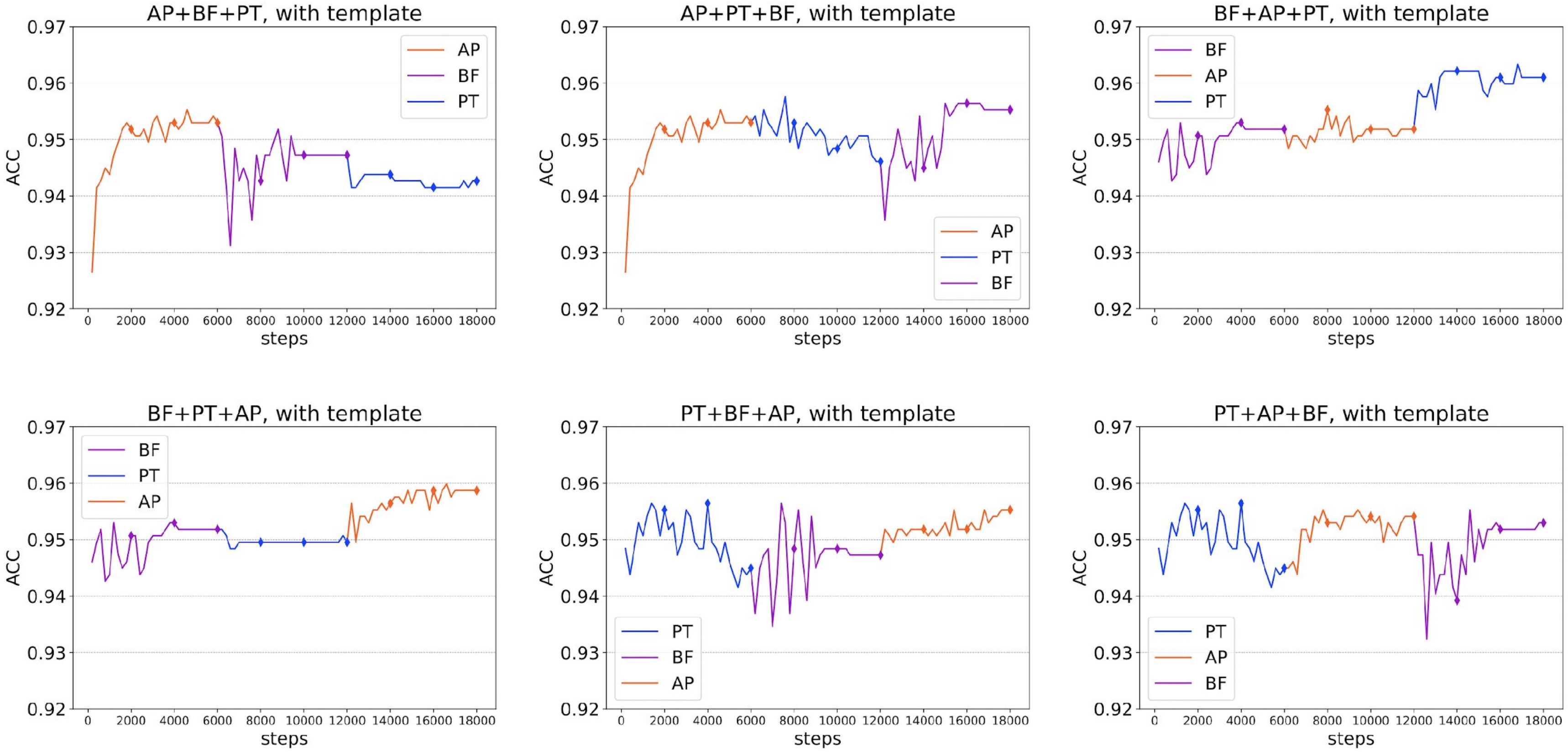
What does memory usage look like?



FT: full fine-tuning
AP: adapter
LR: LoRA
BF: BitFit
Prompt tuning and prefix tuning not included because they use the same amount of memory as full fine-tuning

"Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models." Ding et al. 2023.

Can the methods be combined?



FT: BitFit

AP: adapter

PT: prompt tuning

Results on SST-2 sentiment classification

“Delta Tuning: A Comprehensive Study of Parameter Efficient Methods for Pre-trained Language Models.” Ding et al. 2023.

Options Available to You

How can you use parameter-efficient tuning?

- OpenAI finetuning API
  - It is extremely likely they are using a version of one of the methods described.
  - Unfortunately, we can only rely on speculation.
  - Models available: gpt-3.5-turbo-0613, babbage-002, and davinci-002
- HuggingFace PETM Library
  - LoRA, prefix tuning, prompt tuning, and $(IA)^3$ all implemented.
  - Several different models available to be adapted.

Quiz Question

Suppose you want your LM to be able to perform two different tasks.

You could use prompt tuning to tune a separate prompt for each task.
Or you could tune a single prompt for both tasks simultaneously.

Under what circumstances might one of these approaches work better than the other?