NLU: Lecture 18
Parameter-efficient Finetuning

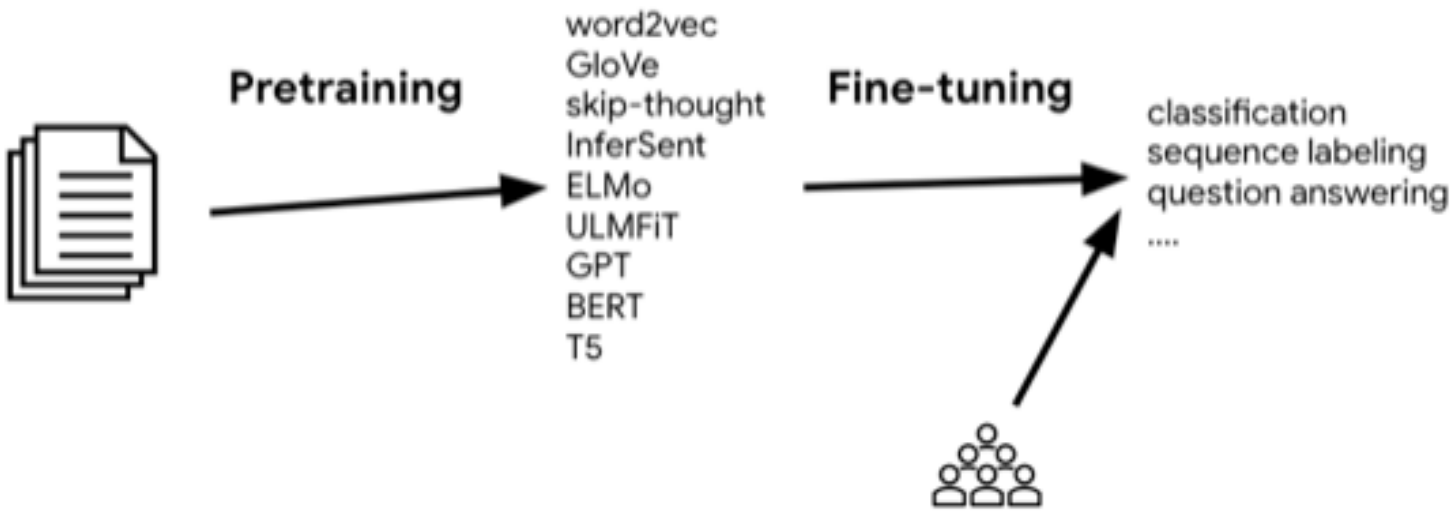
Shay Cohen
(based on slides from Pasquale Minervini)

February 28, 2025

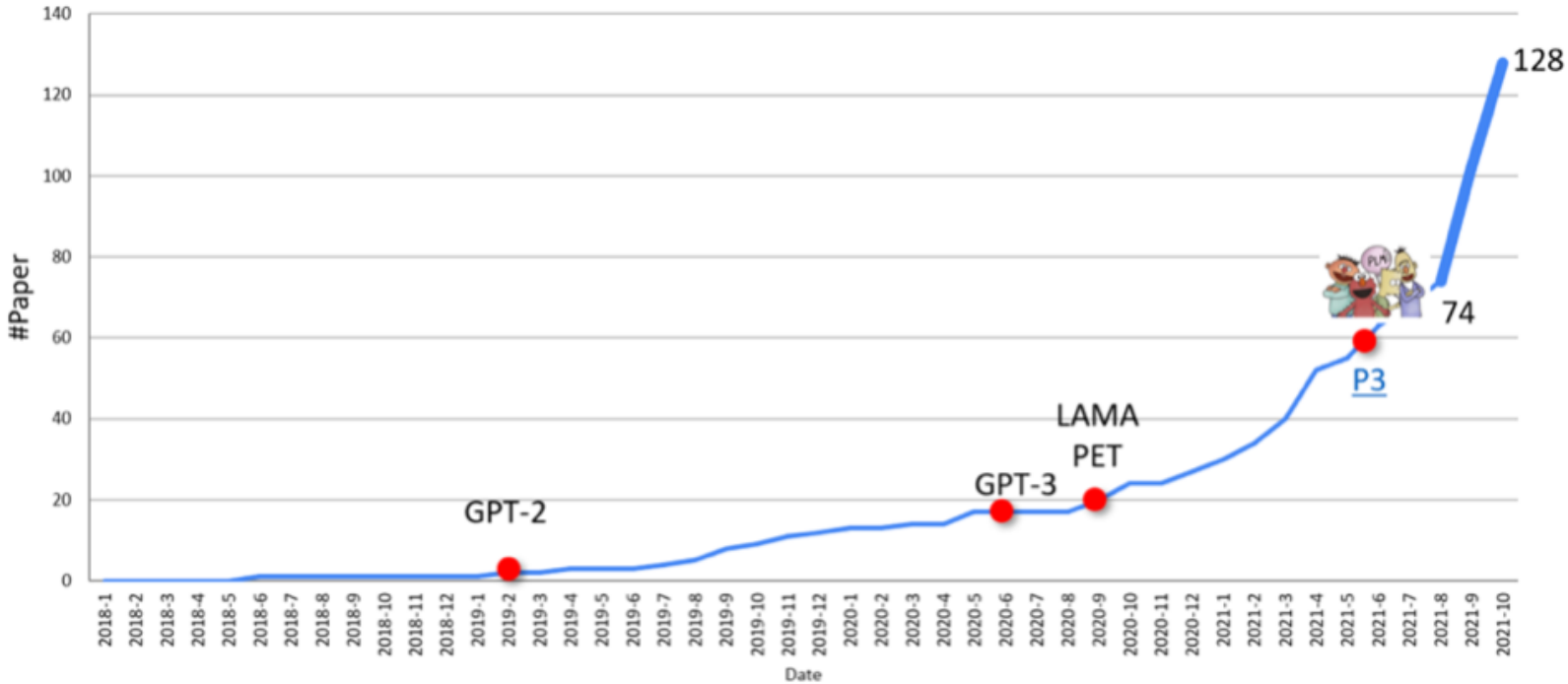
School of informatics

Finetuning

- With increasing model size, fine-tuning becomes increasingly expensive
- The standard transfer learning formula breaks down



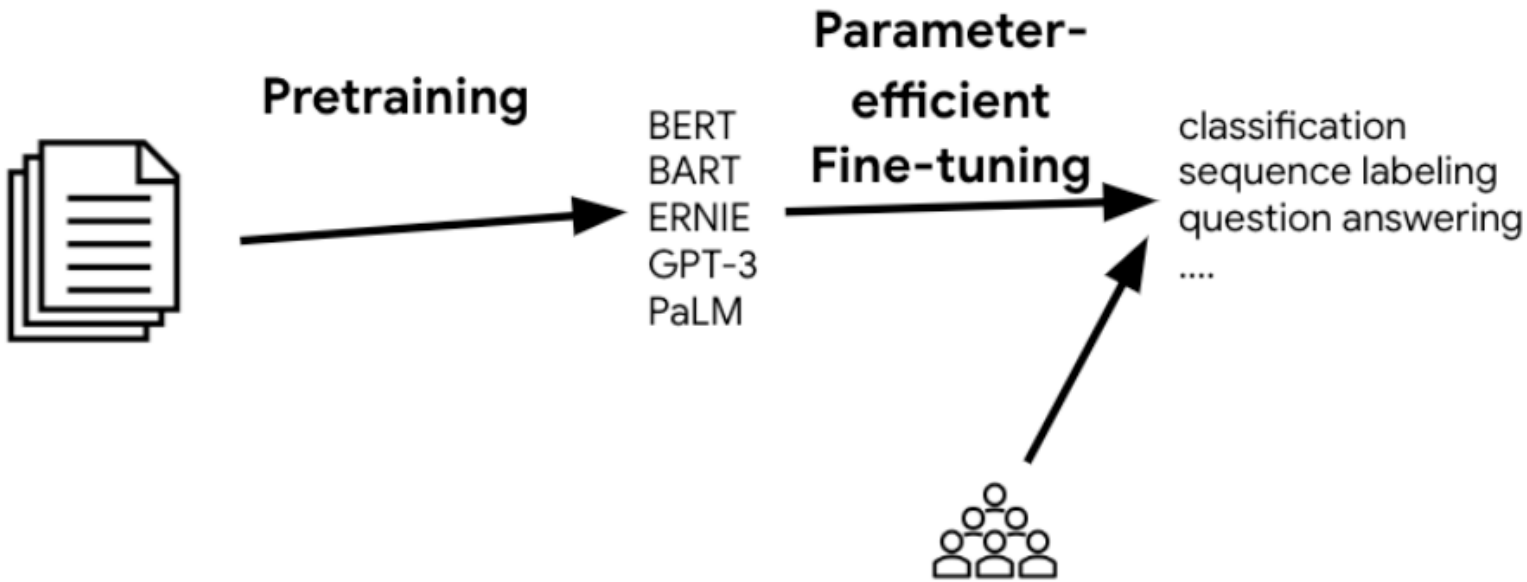
In-context learning



Disadvantages of ICL

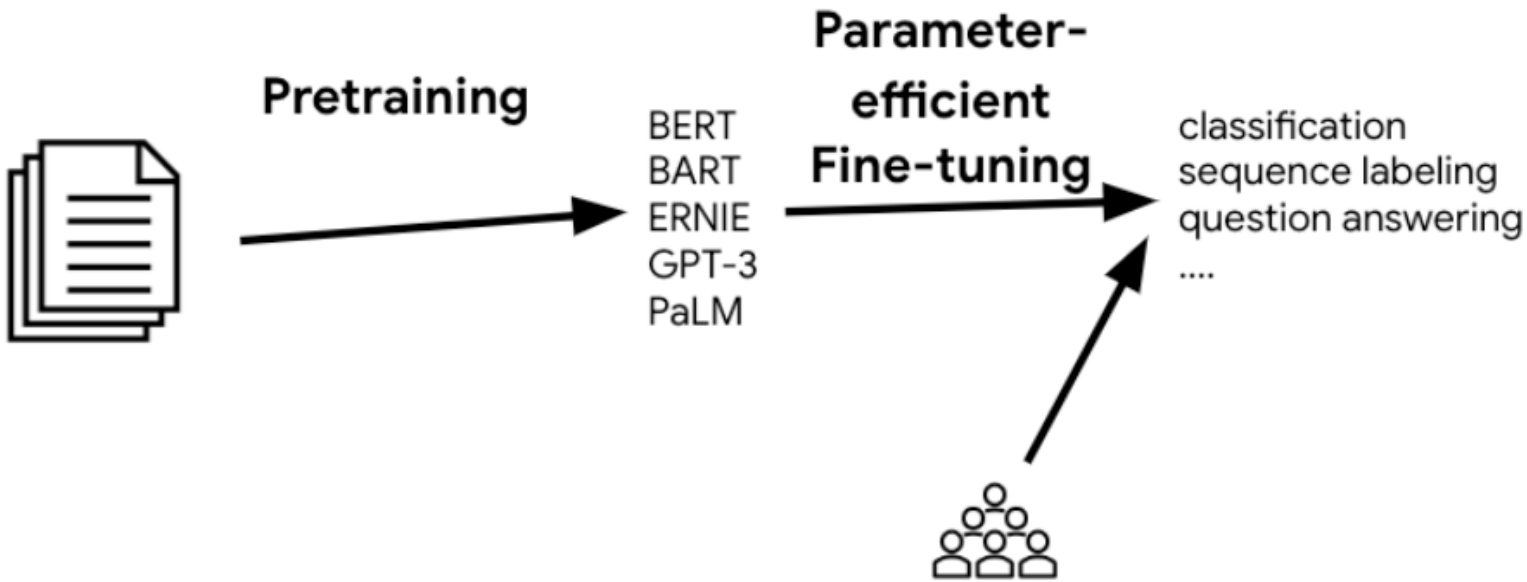
- **Inefficiency**: the prompt needs to be processed every time the model makes a prediction
- **Performance**: prompting generally performs worse than fine-tuning (Brown et al., 2020)
- **Sensitivity** to the wording of the prompt (Webson and Pavlick, 2022), order of examples (Zhao et al., 2021; Lu et al., 2022)
- **Lack of clarity** regarding what the model learns from the prompt — even random label can provide non-trivial results (Min et al., 2022)!

Parameter-efficient Finetuning



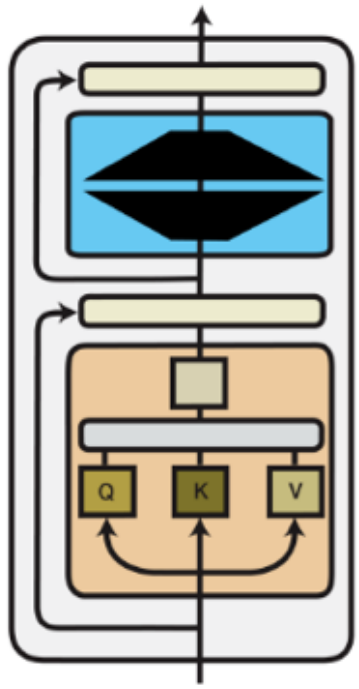
Food for thought: Why is the memory constraint more critical to training rather than inference?

Parameter-efficient Finetuning

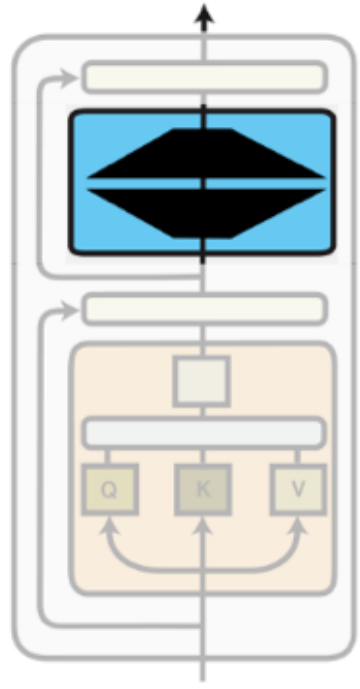


Food for thought: Why is the memory constraint more critical to training rather than inference? More memory needed, for example, gradients

Parameter-efficient Finetuning



Full Fine-tuning
Update **all model parameters**



Parameter-efficient Fine-tuning
Update a **small subset** of model parameters

PEFT

Parameter-Efficient Fine-Tuning (PEFT) is not really a new idea!

- Updating the last layer of the model was common in computer vision (Donahue et al., 2014).
- In NLP, people experimented with static (frozen) and non-static (trainable) (Kim, 2014)
- ELMo did not fine-tune word embeddings (Peters et al., 2018)

PEFT

Parameter-Efficient Fine-Tuning (PEFT) is not really a new idea!

- Updating the last layer of the model was common in computer vision (Donahue et al., 2014).
- In NLP, people experimented with static (frozen) and non-static (trainable) (Kim, 2014)
- ELMo did not fine-tune word embeddings (Peters et al., 2018)
- In practice, fine-tuning everything seems to work better - why go back o fine-tuning only some parameters?
- Fine-tuning everything is impractical with large models
- LLMs nowadays are massively over-parameterised — PEFT matches full fine-tuning in downstream accuracy

Food for thought: How would you reduce the number of parameters?

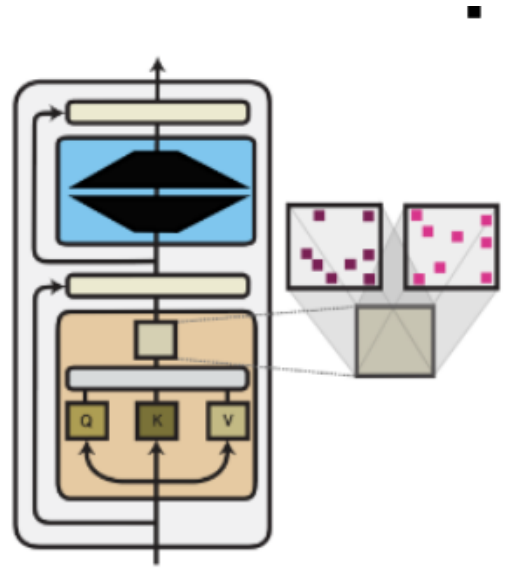
Three Kinds of PEFT

Let $f : X \to Y$ be a neural network, which can be decomposed into a composition of functions $f_{\theta_1} \odot \ldots \odot f_{\theta_n}$, where each function has parameters θ_i with $i \in \{1, \ldots, n\}$.
A module with parameters ϕ can modify a function f_{θ_i} as follows:

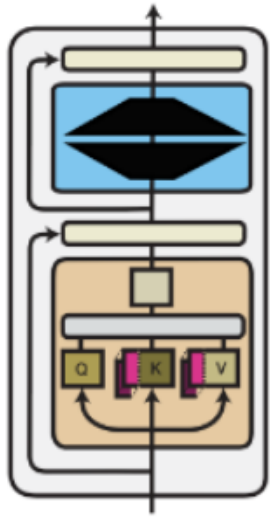
- Parameter composition: $g_i(x) = f_{\theta_i \oplus \phi}(x)$ (interpolation)
- Input composition: $g_i(x) = f_{\theta_i}([x, \phi])$ (concatenation)
- Function composition: $g_i(x) = f_{\theta_i} \odot f_{\phi}(x)$

Typically, only module parameters ϕ are updated while θ is fixed

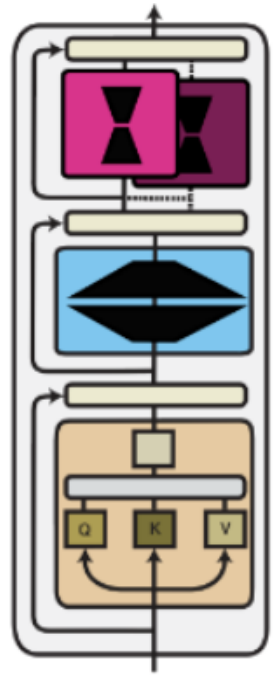
Three Kinds of PEFT



Parameter Composition



Input Composition



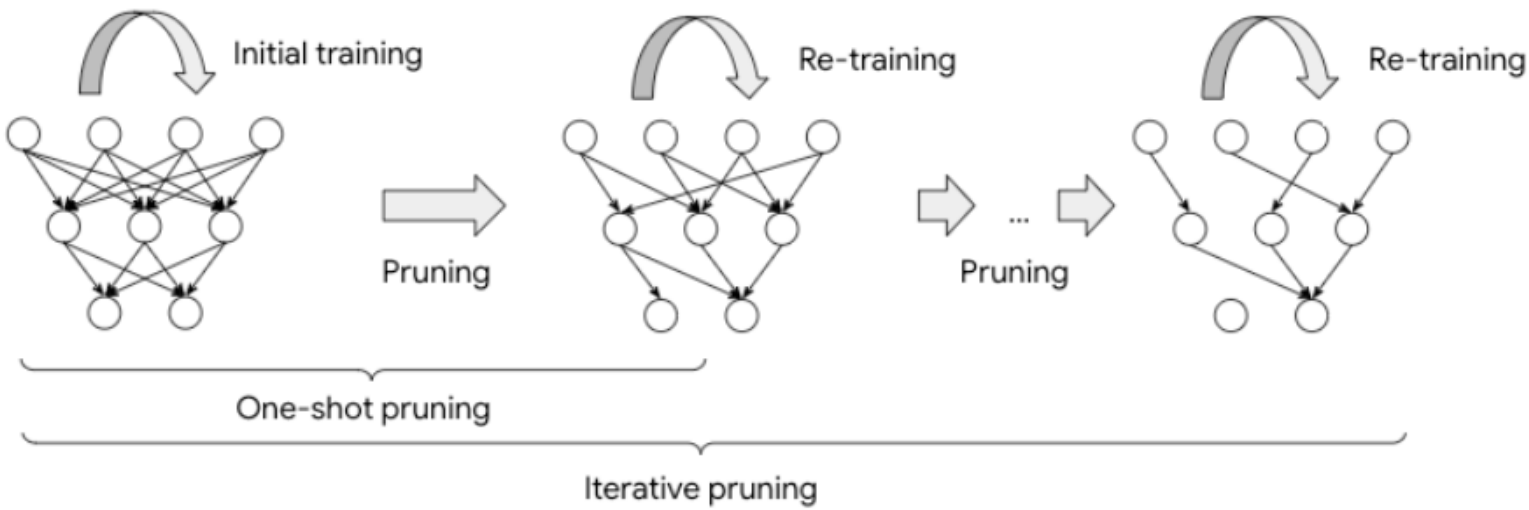
Function Composition

Parameter composition

- **Sparse Subnetworks**, where module parameters ϕ are enforced to be sparse
- **Structured Composition**, where we impose a structure on the weights θ_i that we select - e.g., we update the weights belonging to a pre-defined group
- **Low-Rank Composition**, where the module parameters ϕ lie in a low-dimensional space

Parameter composition: Sparse networks

A common inductive bias on module parameters ϕ is **sparsity**: when we do $\odot$ (element-wise product), we mask part of the neural network f. Most common sparsity method: **pruning** - e.g., see (Han et al., 2017)



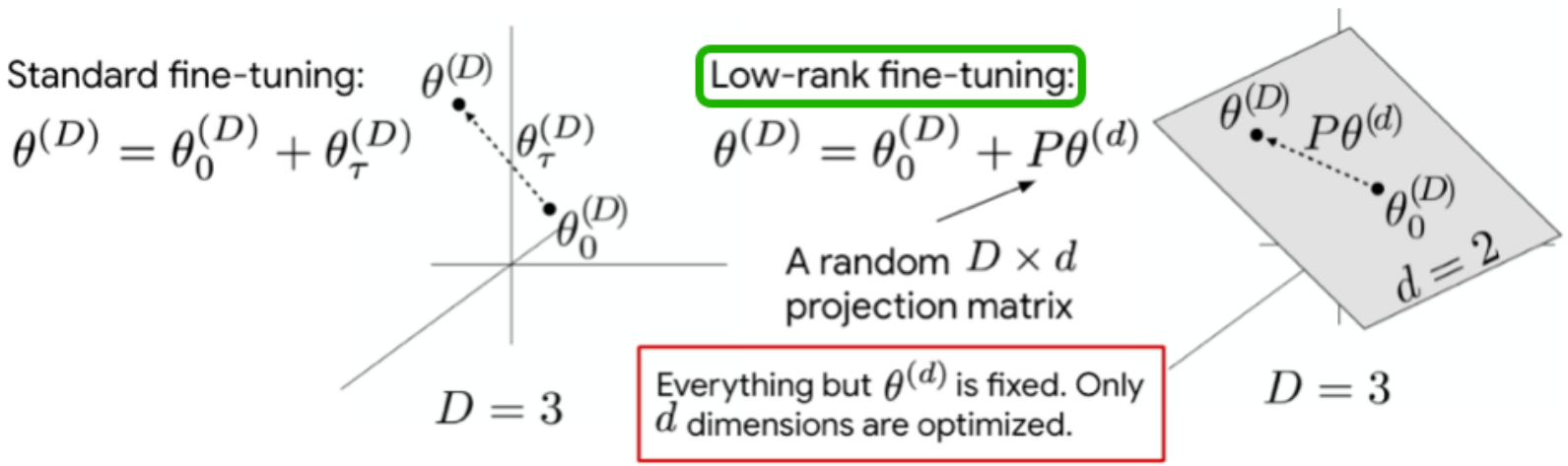
Parameter composition: imposing structure

We can impose a structure on the weights that we select: we only modify the weights that are associated in a pre-defined group G, for example, a layer, a group of layers, or more fine-grained components.

Example: only update bias vectors — BitFit (Ben-Zaken et al., 2022)

Parameter composition: low-rank composition

Another useful inductive bias: module parameters ϕ should lie in a low-dimensional space. Li et al., 2018 show that models can be optimised in a low-dimensional, randomly oriented subspace rather than the full parameter space.
Low-rank finetuning takes the form $g = f_{\theta+P\phi}$ where $P \in \mathbb{R}^{D\times d}$ - with a dense matrix of shape $D \times d$, this scales as $O(D \times d)$ in time and storage. D is the dimension of the model parameter, and d is a reduced size.



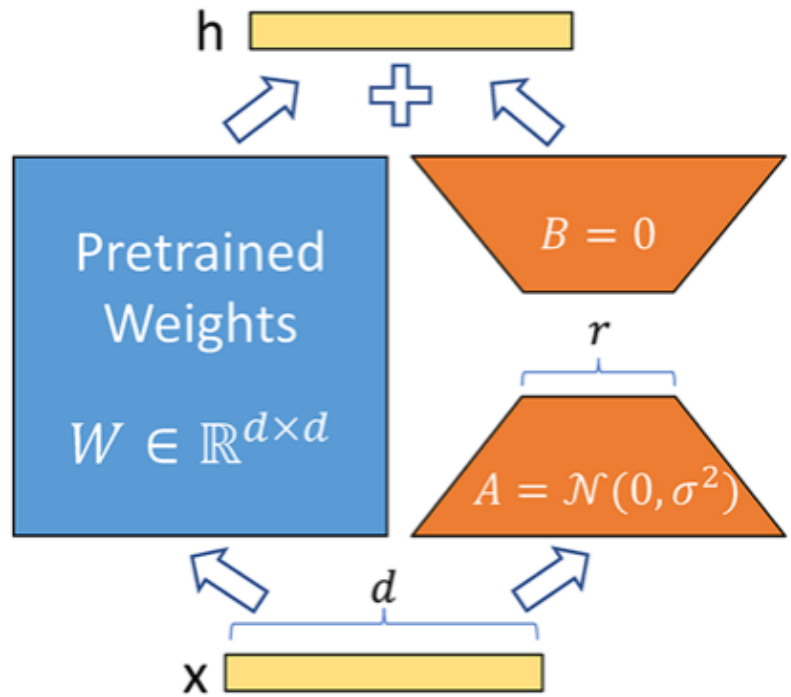
Parameter composition: LoRA

Low-Rank Adaptation — instead of learning a low-rank factorisation via a random matrix P, we can learn the projection matrix directly. LoRA (Hu et al., 2022) learns two matrices $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ that are applied to the self-attention weights:

$$h = [W_0 + \Delta W]x = [W_0 + BA]x$$

In our notation:

$$g_i = f_{\theta_i + B_i A_i}, \forall f_i \in G$$



Parameter composition: LoRA

Applying LoRA to a Transformer layer - remember how Transformers work:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_O$$

where

$$\text{head}_i = \text{Attention}(QW_{Q,i}, KW_{K,i}, VW_{V,i})$$

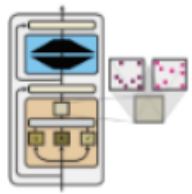
and

$$\text{Attention}(Q', K', V') = \text{softmax}\left(\frac{Q'K'^{\top}}{\sqrt{d'}}\right).$$

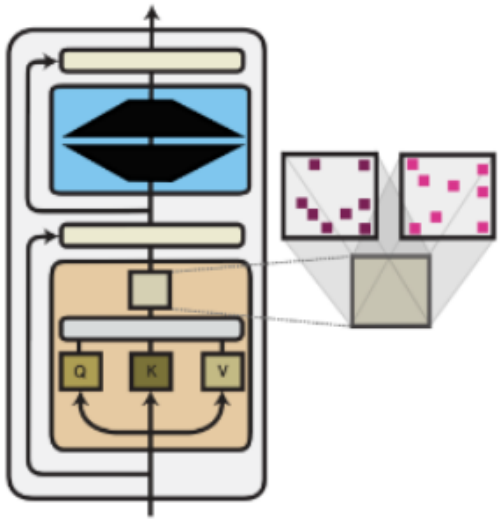
with d' being the dimension of the relevant head
We can use LoRA to adapt the weights $W_{Q,i}$, $W_{K,i}$, and/or $W_{V,i}$ — in the case of $W_{Q,i}$, the updated weights will be:

$$\begin{aligned}\widetilde{W_{Q,i}} &= W_{Q,i} + \Delta W_{Q,i} \\ &= W_{Q,i} + B_{Q,i}A_{Q,i} \text{ with } B_{Q,i} \in \mathbb{R}^{d \times r}, A_{Q,i} \in \mathbb{R}^{r \times d'}\end{aligned}$$

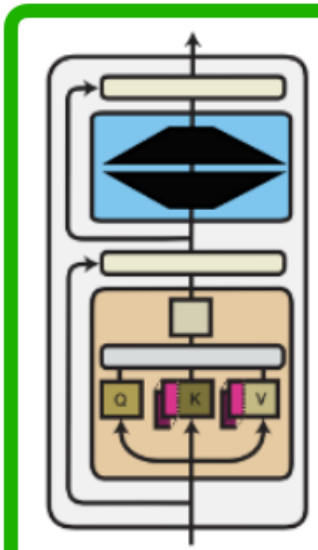
Comparison of Kinds of PEFT

		Parameter efficiency	Training efficiency	Inference efficiency	Performance
Parameter composition		Methods such as LoRA require < 3% of parameters	Pruning requires re-training iterations	Does not increase the model size	E.g., LoRA achieves strong performance
		+	-	++	+

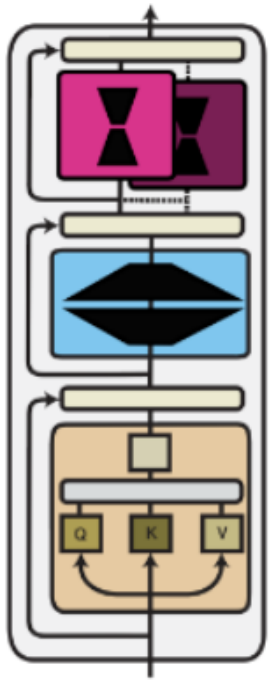
Three Kinds of PEFT



Parameter Composition



Input Composition



Function Composition

Input composition

Augment the input of the model with a learnable vector ϕ:

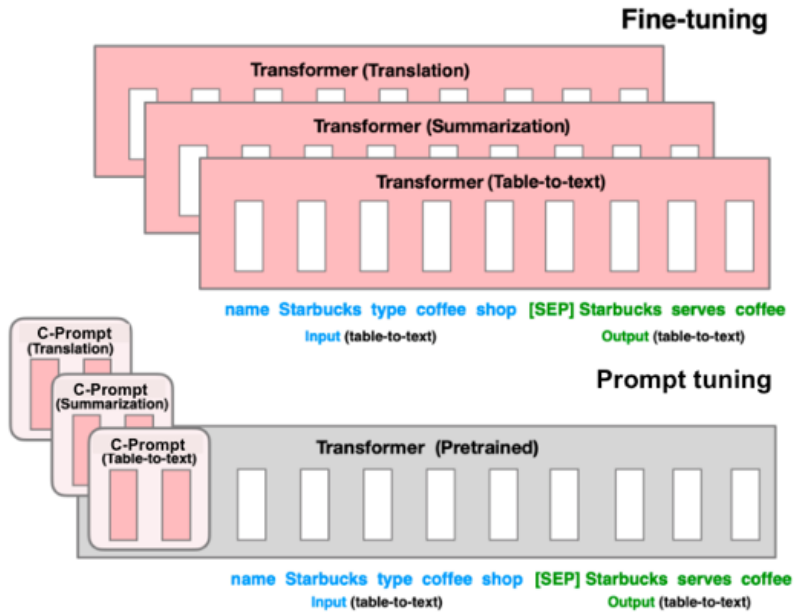
$$g_i(x) = f_{\theta_i}([\phi_i, x])$$

Input Composition and Prompting — standard prompting can be seen as finding a discrete text prompt that, when embedded using the model's embedding layer, yields ϕ_i

However, models tend to be sensitive to the choice of the prompt (Webson and Pavlick, 2022) and the order of examples (Zhao et al., 2021; Lu et al., 2022)

Prompt tuning

Idea — we can directly learn a continuous prompt which is prepended to the input (Liu et al., 2021; Hambardzumyan et al., 2021; Lester et al., 2021)
Here the module parameters is typically a matrix consisting of a sequence of continuous prompt embeddings

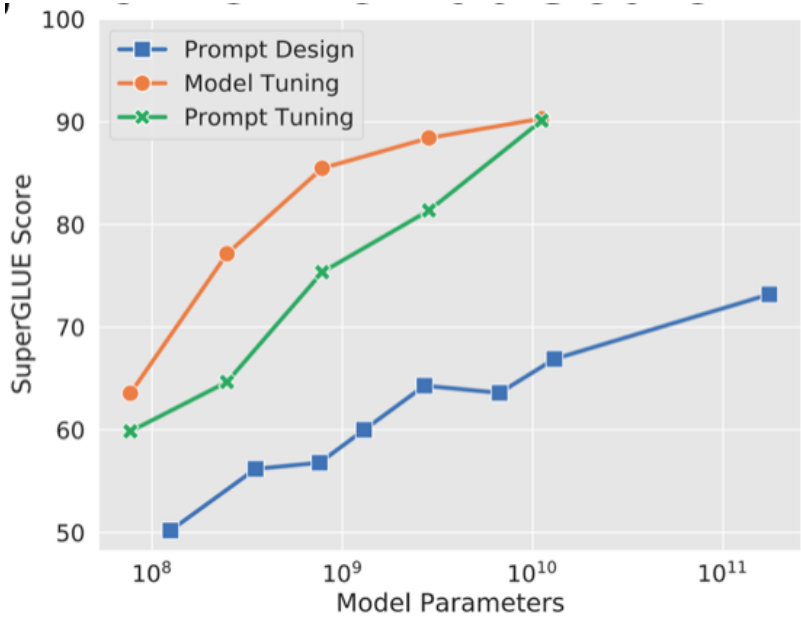


(Li and Liang, 2021)

Prompt tuning works well at scale

Only using trainable parameters at the input layer limits its capacity for adaptation

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

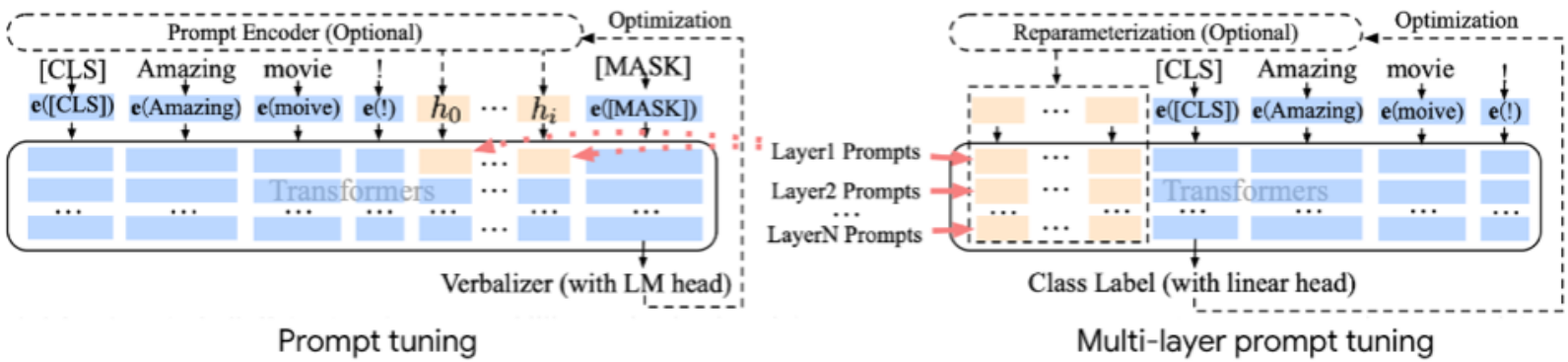


(Li and Liang, 2021)

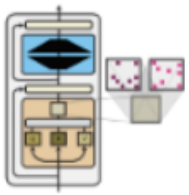
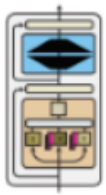
Multi-layer prompt tuning

Instead of learning the module parameters ϕ_i only at the input layer, we can learn them at every layer of the model (Li and Jiang, 2021; Liu et al., 2022)

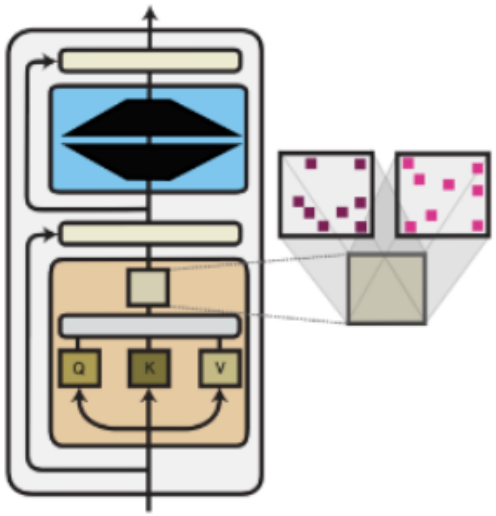
In practice, continuous prompts ϕ_i are concatenated with the keys and values in the self-attention layer (Li and Jiang, 2021)



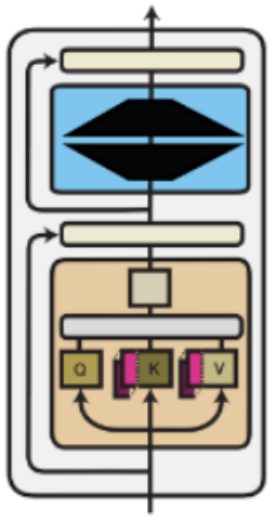
Comparison of Kinds of PEFT

		Parameter efficiency	Training efficiency	Inference efficiency	Performance
Parameter composition		+	-	++	+
Input composition		Only add a small number of parameters	Extend the model's context window		Requires large models to perform well
		++	--	--	-

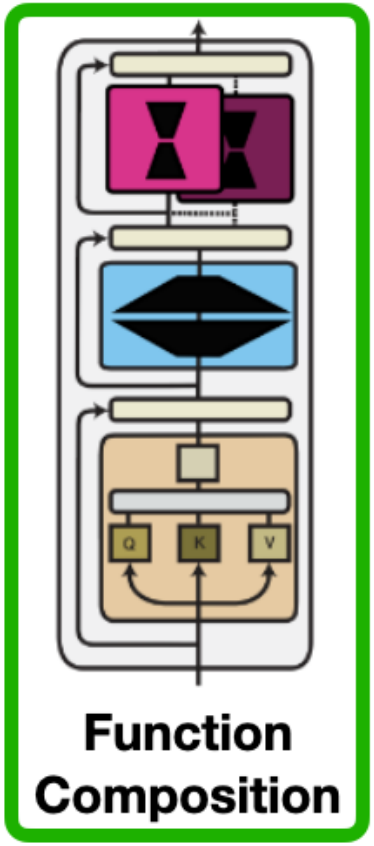
Three Kinds of PEFT



Parameter Composition



Input Composition



Function Composition

Function composition

Function composition augments a model's functions with new task-specific functions:

$$g_i(x) = f_{\theta_i} \odot f_{\phi}(x)$$

Commonly used in multi-task learning, where we have multiple task-specific models composed together — e.g., see the surveys in (Ruder, 2017; Crawshaw, 2020)

However, here we focus on functions that can be added to pre-trained models like LLMs

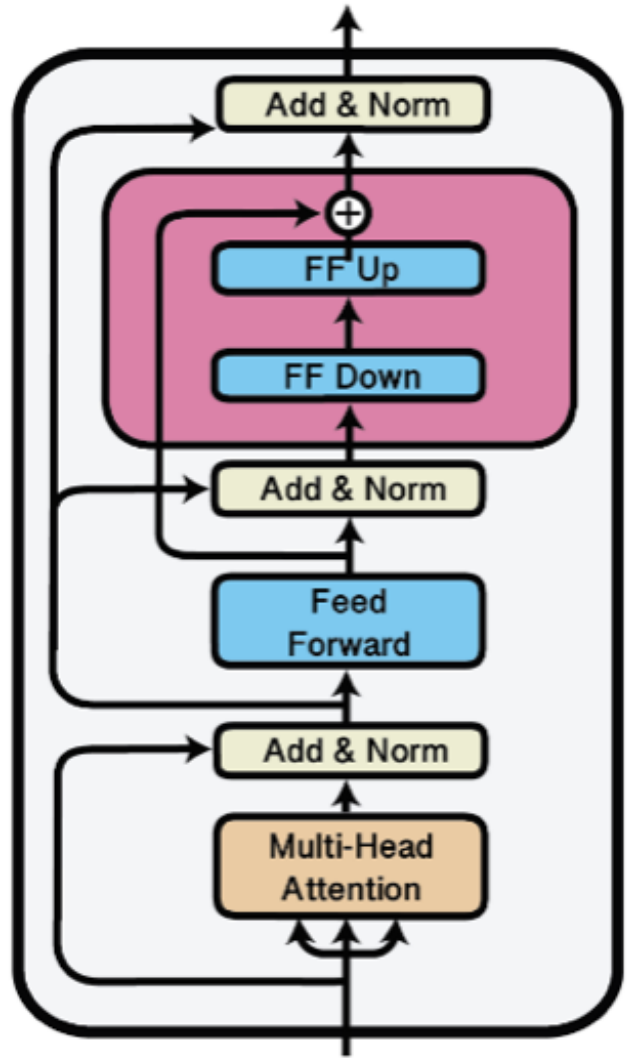
Adapters

The main purpose of functions f_{ϕ_i} added to a pre-trained model is to adapt it to a new task — these functions are also known as **adapters**

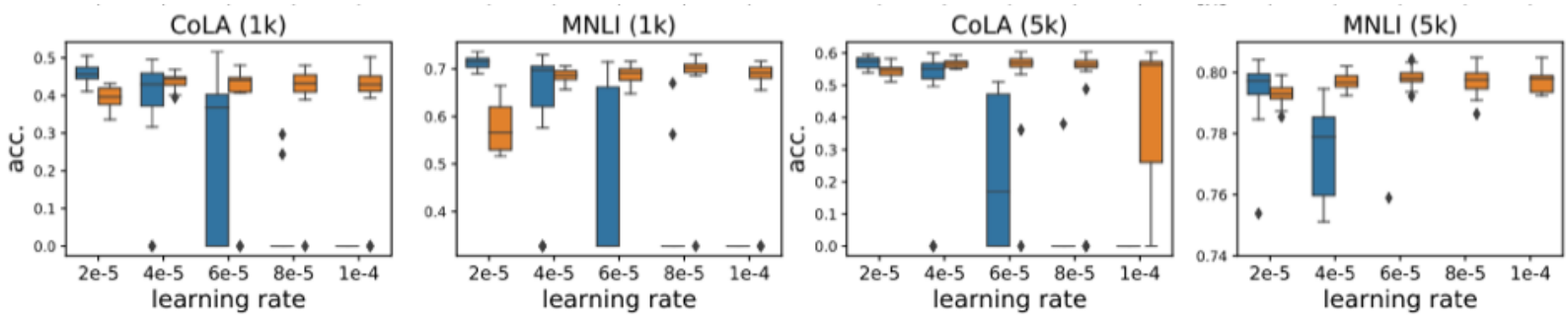
In NLP, an adapter in a Transformer layer typically consists of a feed-forward down-projection $W_D \in \mathbb{R}^{k \times d}$, a feed-forward up-projection $W_U \in \mathbb{R}^{d \times k}$, and an activation function σ (Houlsby et al., 2019)

$$f_{\phi_i}(x) = W_D \left[\sigma \left(W_U x\right)\right]$$

Adapter usually placed after multi-head attention and/or after the feed-forward layer



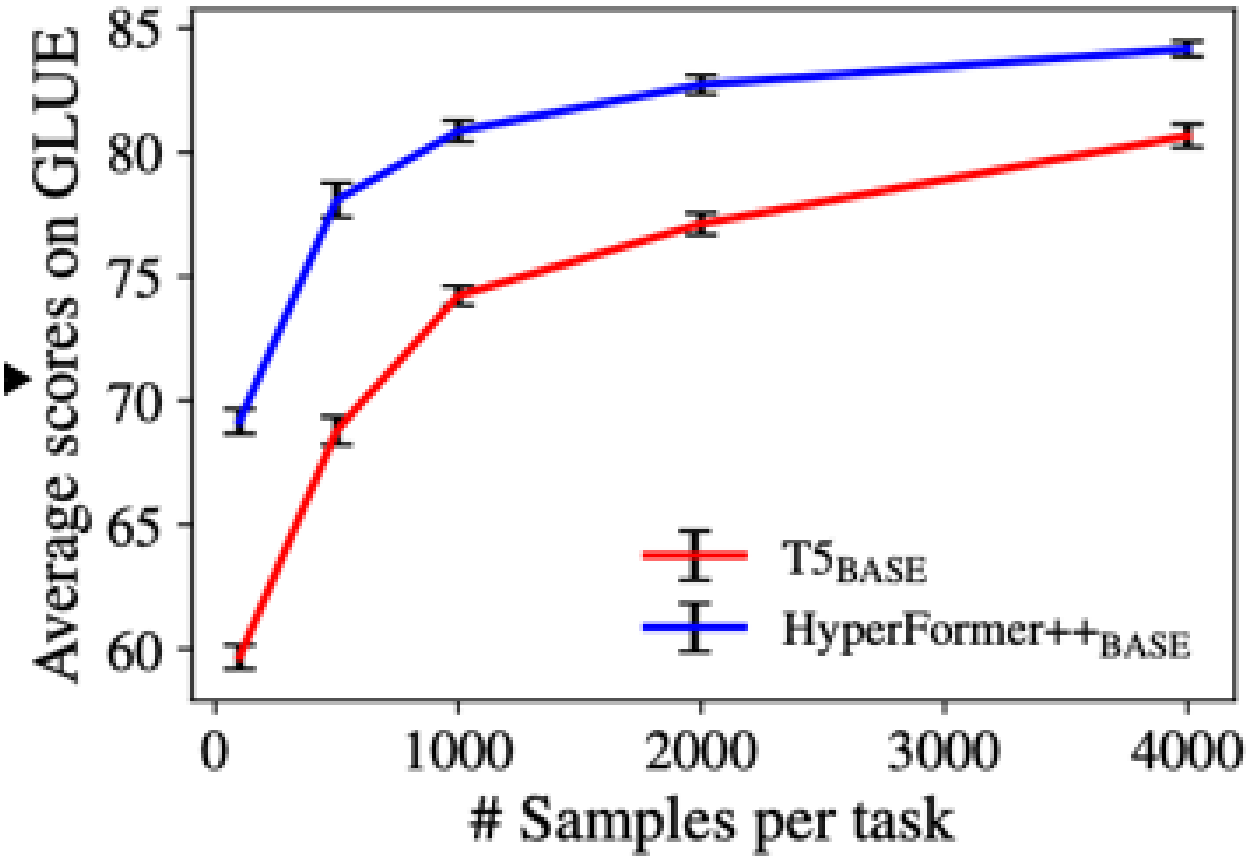
Benefits of adapters



Increased robustness (He et al., 2021; Han et al., 2021)

- (blue - finetuning; orange - adapters)
- Adapters are more stable varying over tasks and learning rates

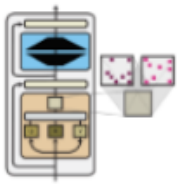
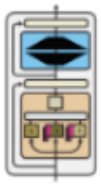
Benefits of adapters



Increased sample efficiency (Mahabadi et al., 2021)

Results on GLUE with different numbers of training examples per task

Benefits of adapters

		Parameter efficiency	Training efficiency	Inference efficiency	Performance
Parameter composition		+	-	++	+
Input composition		++	--	--	-
Function Composition		Adapters depend on the hidden size	Does not require gradients of frozen params	New functions increase # of operations	Match or outperform standard fine-tuning
		-	+	-	++

Summary

- We need PEFT because it is difficult to finetune the model in full
- We learned about three kinds of PEFT methods: parameter composition, input composition, function composition
- Each has its own advantages and disadvantages
- LoRA and adapters are probably the most commonly used PEFT methods in practice