

CS639 Deep Learning for NLP

Parameter-Efficient Fine-tuning (PEFT)

Junjie Hu



<https://junjiehu.github.io/cs639-spring26/>

Outline

- What is PEFT and why do we care about it?
- Classes of PEFT methods
- Adapters & (IA)³
- Prefix-tuning & Prompt-tuning
- LoRA & Q-LoRA

Background: Open-source LLMs

- Language models are becoming larger over time, so it's computationally expensive to fine-tune these open-source LLMs

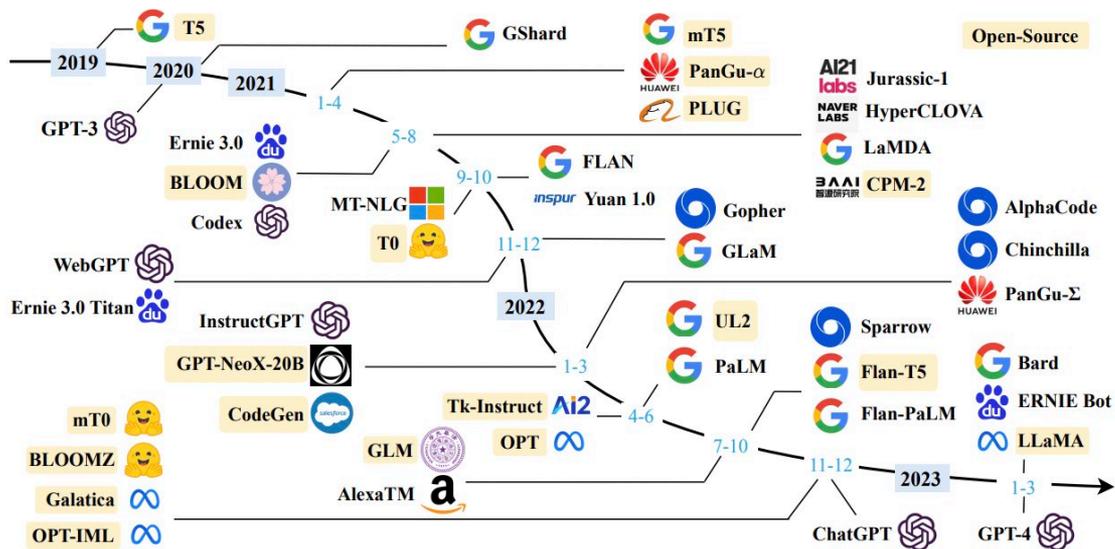
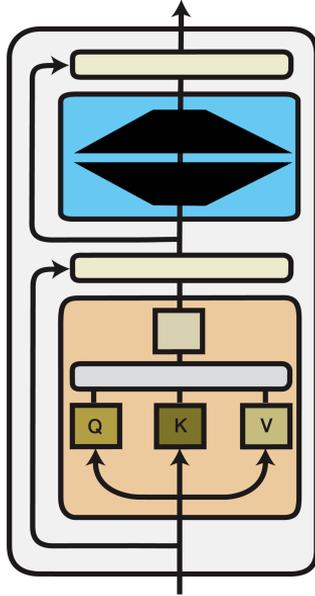
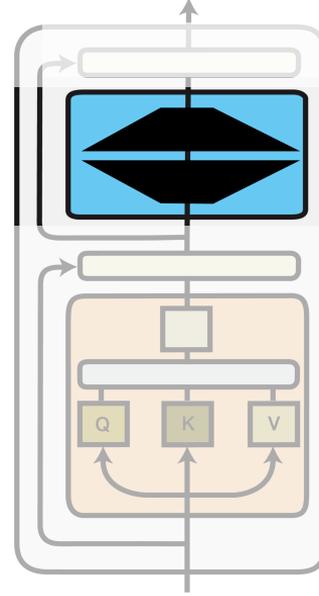


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

From Fine-tuning to Parameter-efficient Fine-tuning



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



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

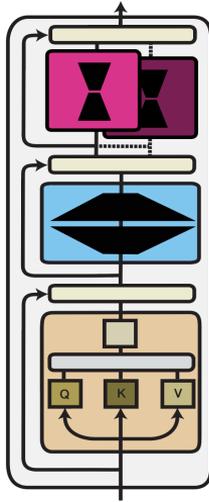
Motivation: Why PEFT?

- **PEFT:** Fine-tune [a small amount of model parameters](#) (instead of the entire model) on a small dataset of downstream tasks. Other parameters are frozen.
- **Question: What are benefits?**
 - Reduce the computational and storage costs
 - Mitigate catastrophic forgetting — forgetting often occur when the model changes a lot after fine-tuning. PEFT can be regarded as a regularization on the difference between the two checkpoints before and after PEFT.
 - Easy to update models to new data and facts
 - Better performance in low-data regimes
 - Comparable performance to full fine-tuning

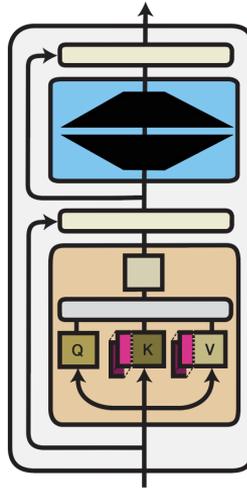
Comparison Between PEFT and Fine-tuning

	PEFT	Full Fine-tuning
Learnable parameters	A small subset	Entire model
Training Performance	Close to fine-tuning	Closed to fine-tuning
Training Data	Small	Large
Training Time	Faster	Longer training time
Overfitting / forgetting	Less prone to overfitting	More prone to overfitting

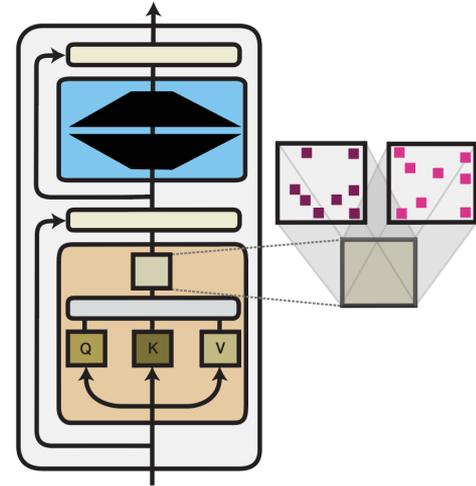
Three Computation Functions



Function Composition



Input Composition



Parameter Composition

Three Computation Functions

Let a neural network $f_{\theta} : \mathcal{X} \rightarrow \mathcal{Y}$ be decomposed into a composition of functions:

$f_{\theta_1} \odot f_{\theta_2} \odot \cdots \odot f_{\theta_l}$. Each has parameters $\theta_i, i = 1, \dots, l$

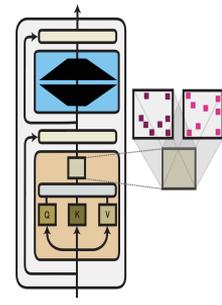
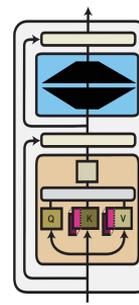
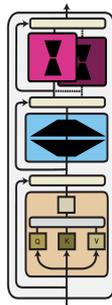
A module with parameters ϕ can modify the i -th subfunction as follows:

1. Function composition: $f'_i(\mathbf{x}) = f_{\theta_i} \odot f_{\phi}(\mathbf{x})$ Function composition
2. Input composition: $f'_i(\mathbf{x}) = f_{\theta_i}([\mathbf{x}, \phi])$ Concatenation
3. Parameter composition: $f'_i(\mathbf{x}) = f_{\theta_i \oplus \phi}(\mathbf{x})$ Interpolation, e.g., element-wise addition

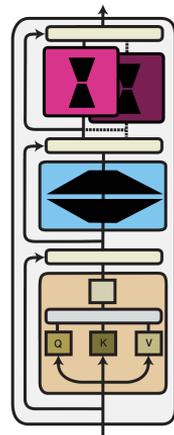
In practice, typically only the module parameters ϕ are updated while θ is fixed.

Three Computation Functions

	Function Composition	Input Composition	Parameter Composition
Example Methods	Adapters, (IA) ³	Prompt Tuning, Prefix Tuning	LoRA, QLoRA, Pruning
Impact on Model Size	Additional modules in layers	Context window of model is increased	No increase in model size
Performance	Matches or outperforms fine-tuning	Good with large models	Good

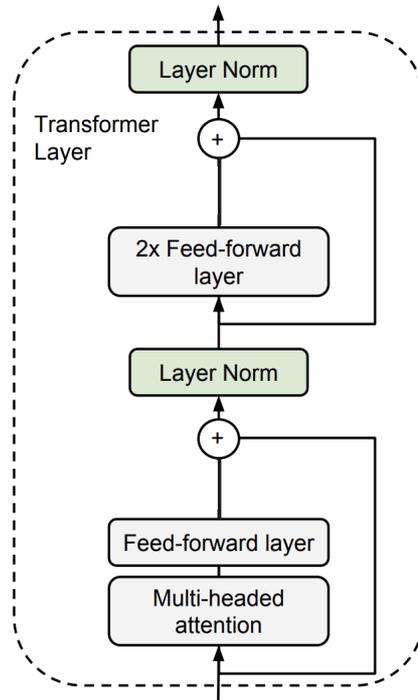


Function Composition: Adapters, $(IA)^3$



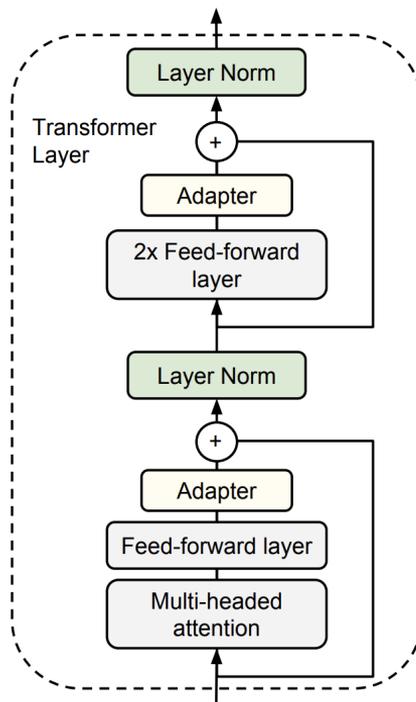
Standard Transformer Layer

- Standard Transformer Layer:
 - Attention layers
 - Feed-forward layers
 - Layer normalization
 - Residual connection



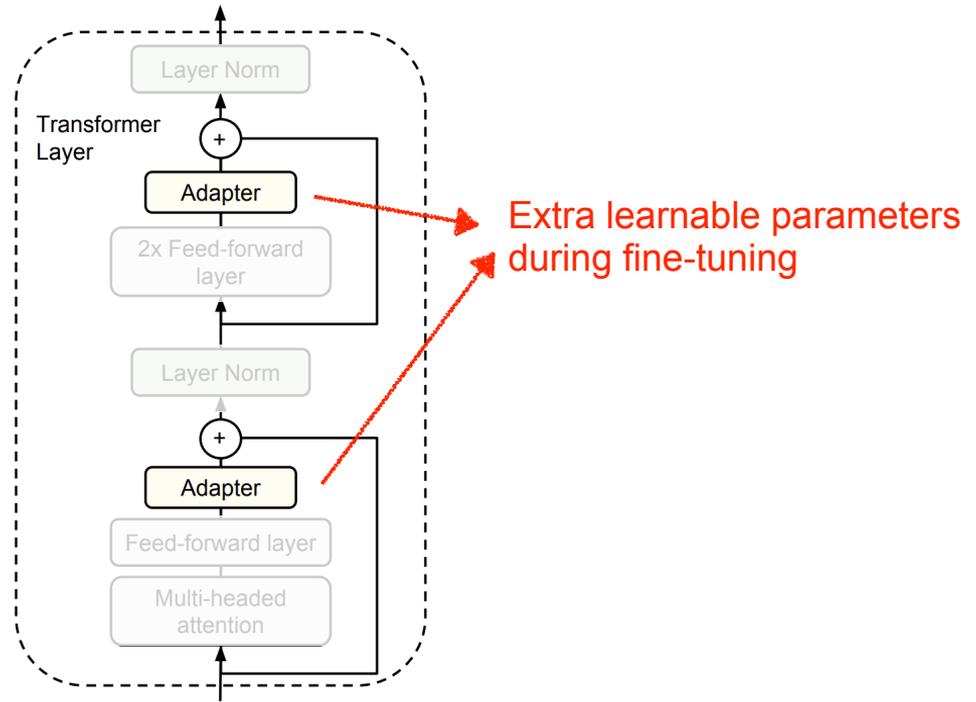
Adapter

- Add an adapter after the feed-forward layer in each Transformer layer



Adapter

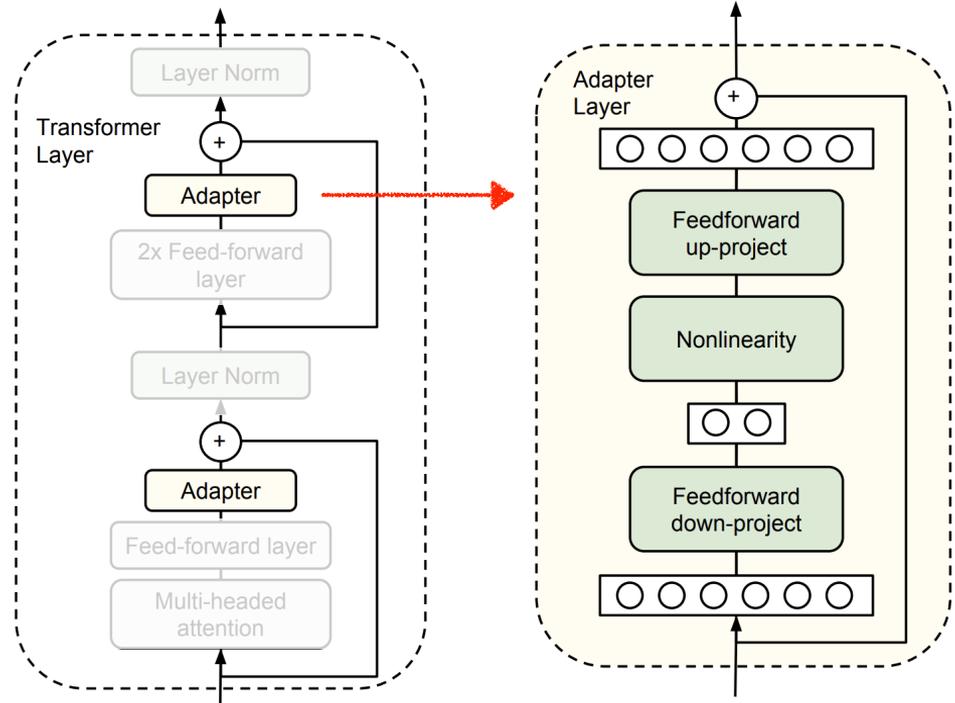
- Add an adapter after the feed-forward layer in each Transformer layer
- The adapter will be **fine-tuned** in PEFT, while the other layers are **frozen**.



Adapter

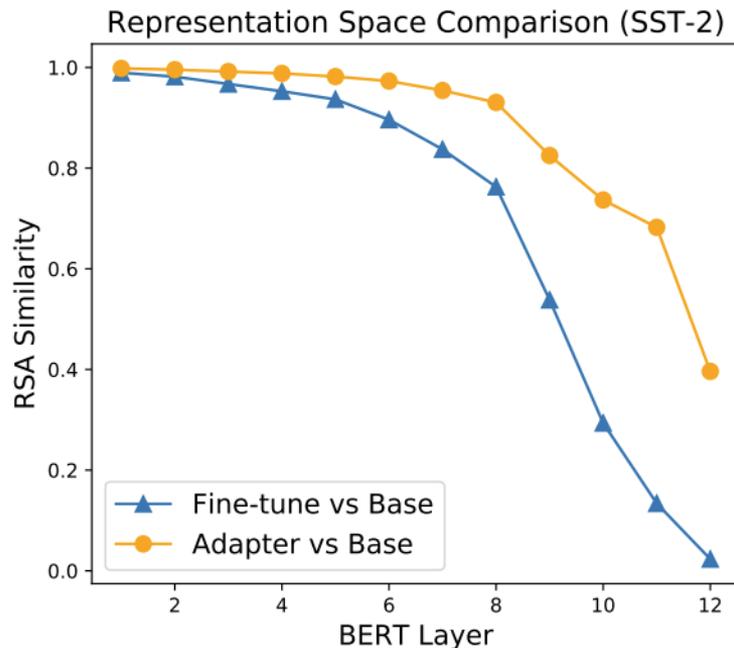
- An adapter is a small MLP
 - One Feedforward layer
 - Nonlinear activation
 - Another Feedforward layer

$$f_{\phi}(x) = W^U(\sigma(W^D x))$$



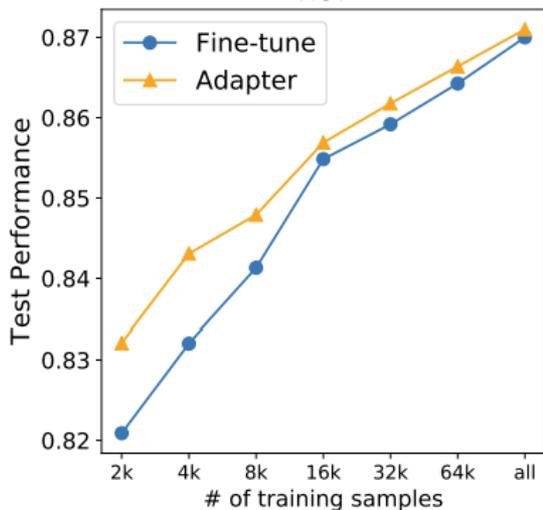
Adapters v.s. Full-parameter fine-tuning

- In practice, Adapters change the embeddings less than fine-tuning

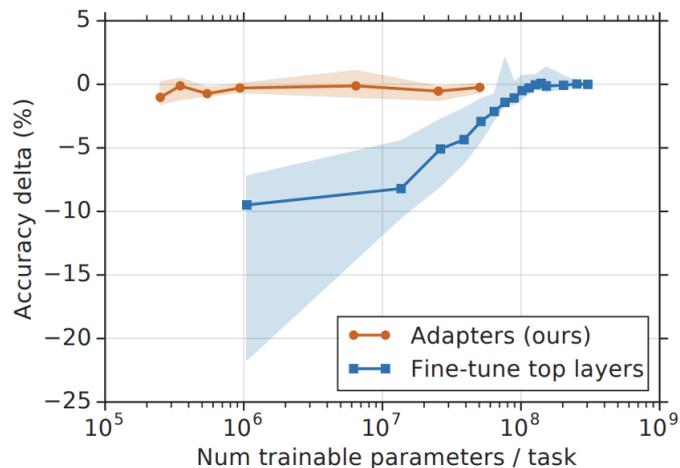


Adapters v.s. Full-parameter fine-tuning

- Compare **data-efficiency** and **parameter-efficiency**



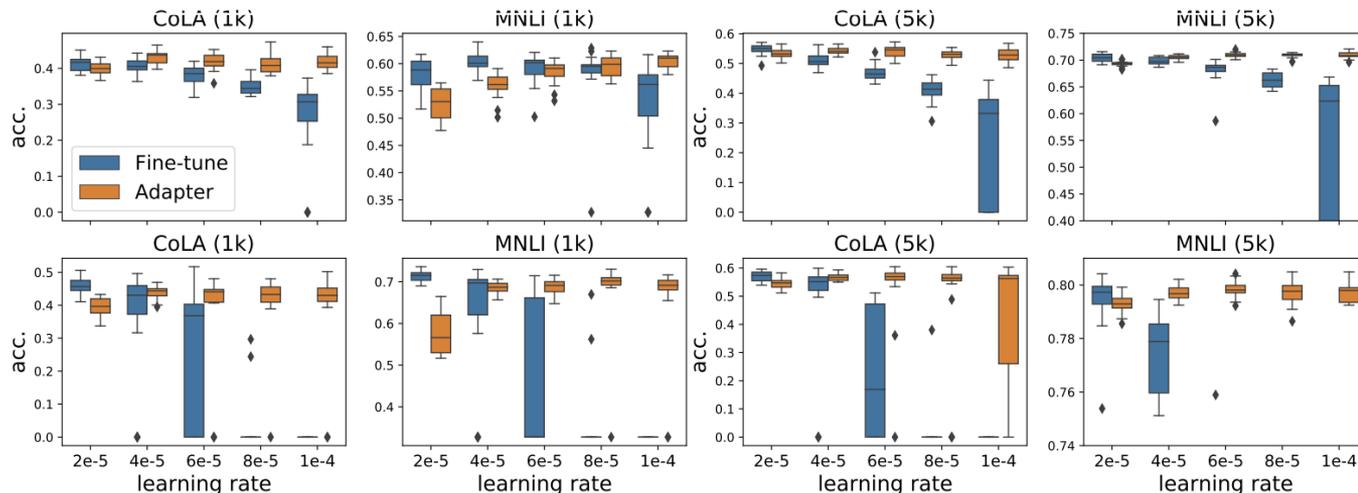
Adapters > FT with less data



Adapters > FT with less params

Adapters v.s. Full-parameter fine-tuning

- Run experiments multiple times with different hyperparameters, and compute the accuracy and standard deviation



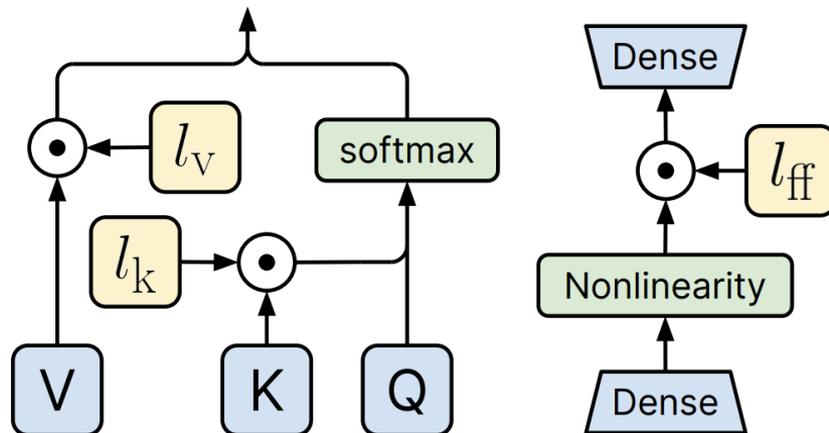
Finding: Adapters are less sensitive to hyperparameters like learning rate

(IA)³

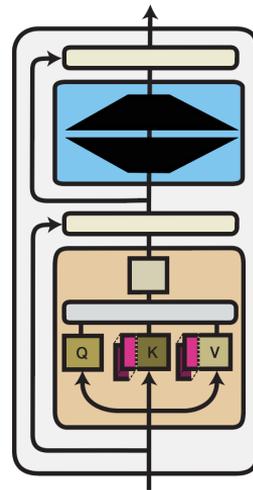
- Instead of learning a function, even rescaling via element-wise multiplication can be powerful:

$$f'_i(\mathbf{x}) = f_{\theta_i}(\mathbf{x}) \circ \phi_i$$

- Allows the model to select parameters that are more and less important for a given task



Input Composition: Prompt-tuning, Prefix-tuning

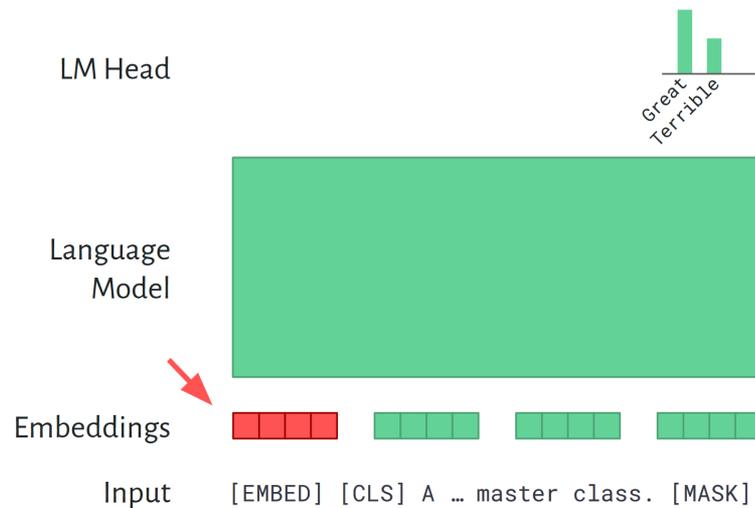


Motivation

- **Prompting with text:** Prepending instructive words or demonstrations before the actual test input
- 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 are sensitive to the formulation of the prompt and to the order of examples
- Why not skip the words and directly learn an appropriate ϕ_i ?

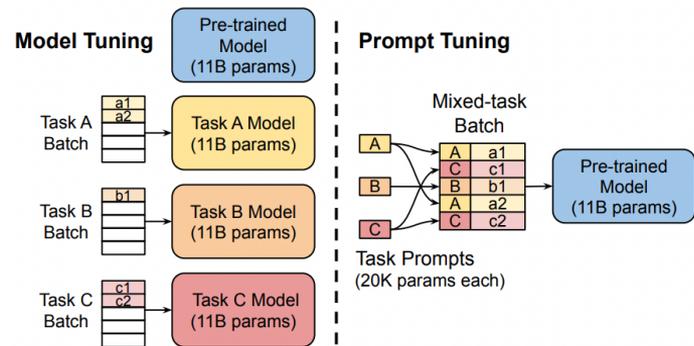
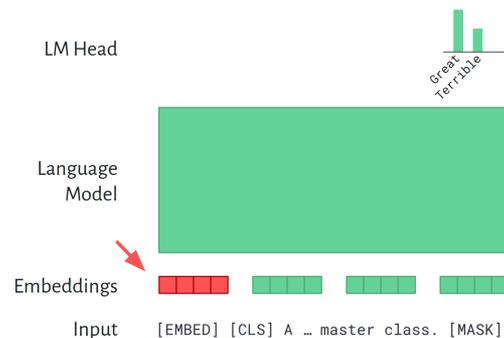
Prompt Tuning

- Prompt tuning only updates **a small task-specific prompt parameters for each task**, enables mixed task inference.



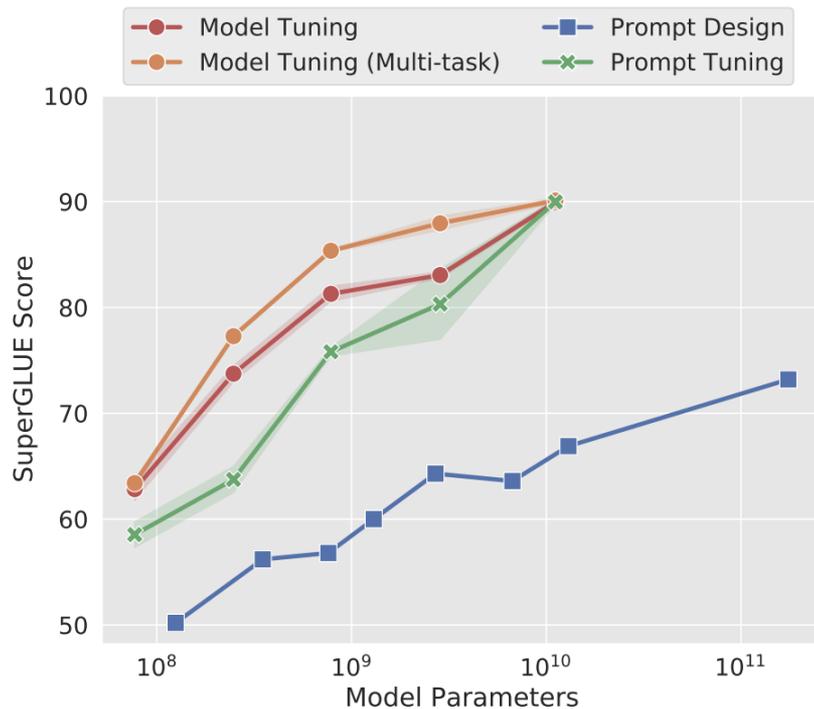
Prompt Tuning

- Prompt tuning only updates **a small task-specific prompt parameters for each task**, enables mixed task inference.
- Fine-tuning (Model tuning): make a task-specific copy of the entire pre-trained LMs for each task, and inference must be performed in separate batches.



Prompt Tuning vs Model Tuning

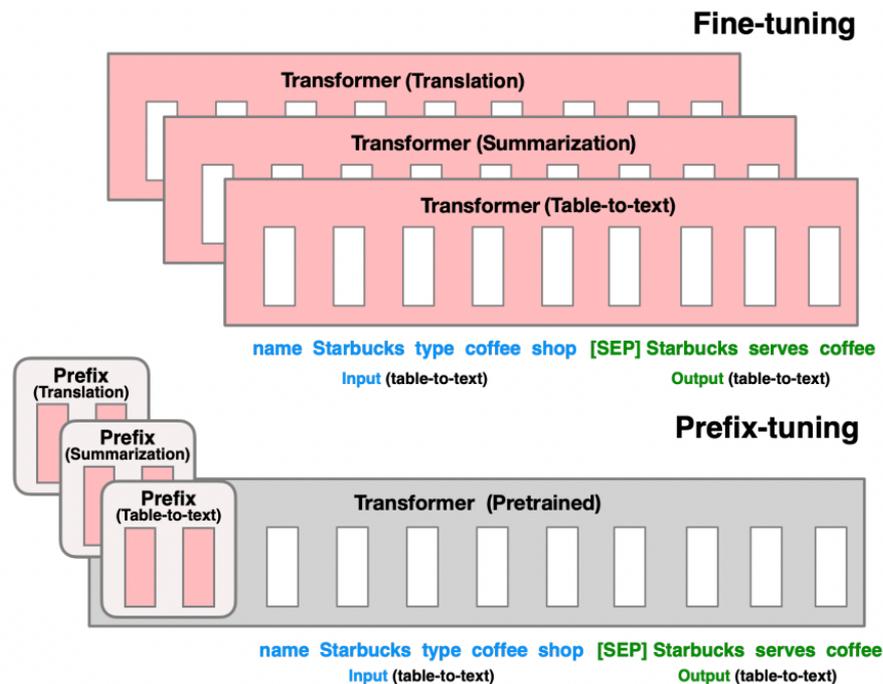
- As model size increases (e.g., T5-XXL 11B model), prompt tuning of T5 (green curve) matches the performance of (full) model tuning (red/orange curves) on SuperGLUE.
- Prompt design: few-shot in-context prediction by GPT-3 (blue curve) is still way worse than fine-tuning.



Prefix Tuning / Multi-Layer Prompt Tuning

- Add learnable parameters at the beginning of the input sequence over **all Transformer layers**.
- Use different prefix parameters for different tasks, and keep the other parameters frozen

$$z = [\mathbf{PREFIX}; x; y]$$



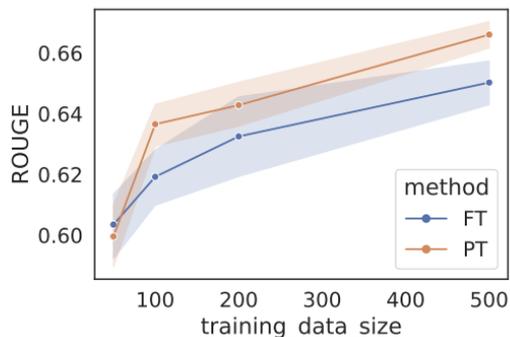
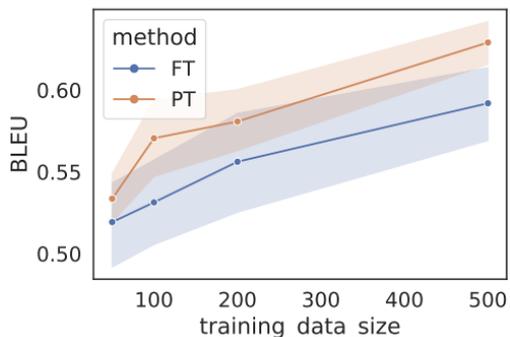
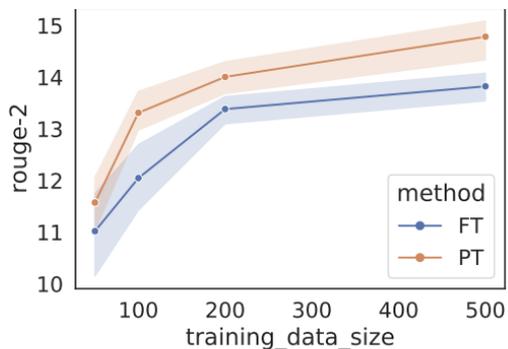
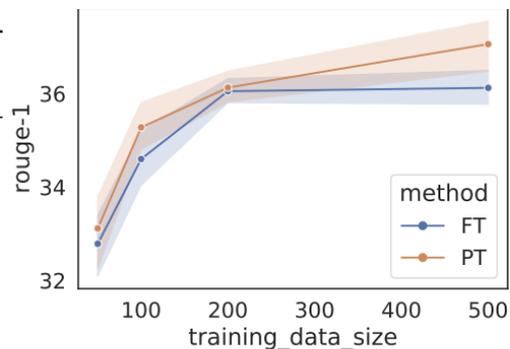
Effective on NLG tasks w/ 0.1% parameters

- Evaluate on three table-to-text generation datasets: E2E, WebNLG, and DART
- Continuous prompts in later layers are more important

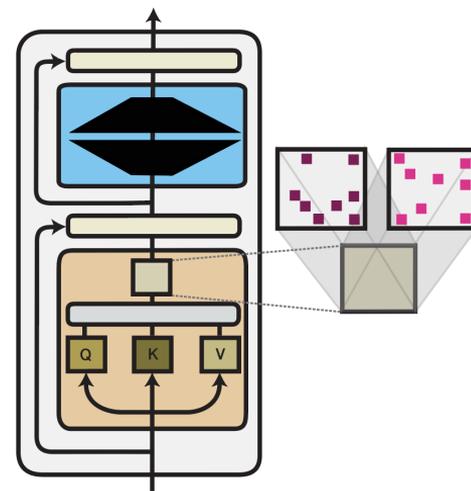
	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
GPT-2 _{MEDIUM}																				
FINE-TUNE	68.2	8.62	46.2	71.0	2.47	64.2	27.7	46.5	0.45	0.30	0.38	0.33	0.76	0.53	46.2	0.39	0.46	0.50	0.94	0.39
FT-TOP2	68.1	8.59	46.0	70.8	2.41	53.6	18.9	36.0	0.38	0.23	0.31	0.49	0.99	0.72	41.0	0.34	0.56	0.43	0.93	0.21
ADAPTER(3%)	68.9	8.71	46.1	71.3	2.47	60.4	48.3	54.9	0.43	0.38	0.41	0.35	0.45	0.39	45.2	0.38	0.46	0.50	0.94	0.39
ADAPTER(0.1%)	66.3	8.41	45.0	69.8	2.40	54.5	45.1	50.2	0.39	0.36	0.38	0.40	0.46	0.43	42.4	0.36	0.48	0.47	0.94	0.33
PREFIX(0.1%)	69.7	8.81	46.1	71.4	2.49	62.9	45.6	55.1	0.44	0.38	0.41	0.35	0.49	0.41	46.4	0.38	0.46	0.50	0.94	0.39
GPT-2 _{LARGE}																				
FINE-TUNE	68.5	8.78	46.0	69.9	2.45	65.3	43.1	55.5	0.46	0.38	0.42	0.33	0.53	0.42	47.0	0.39	0.46	0.51	0.94	0.40
Prefix	70.3	8.85	46.2	71.7	2.47	63.4	47.7	56.3	0.45	0.39	0.42	0.34	0.48	0.40	46.7	0.39	0.45	0.51	0.94	0.40
SOTA	68.6	8.70	45.3	70.8	2.37	63.9	52.8	57.1	0.46	0.41	0.44	-	-	-	-	-	-	-	-	-

Low-Data Setting

- Perform comparative to fine-tuning in low-data regimes where there are only a few training data



Parameter Composition: LoRA and Q-LoRA



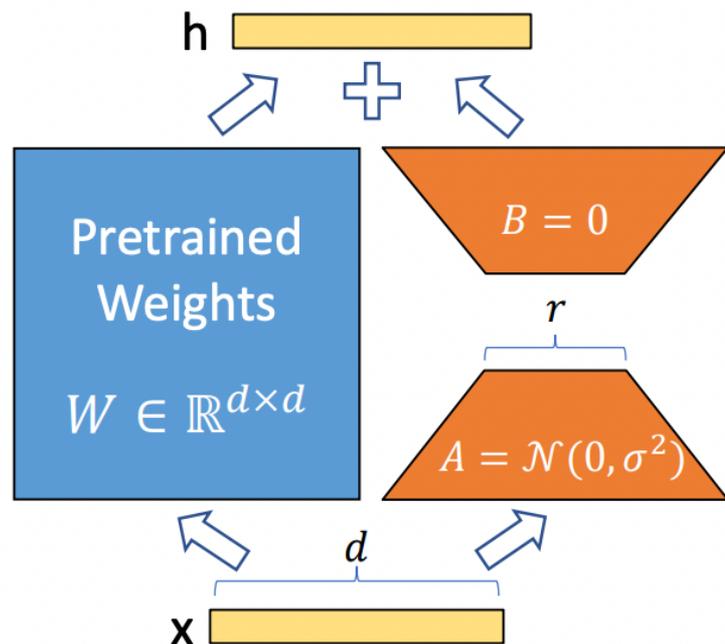
LoRA

- Approximate the self-attention update of a learnable weight by a low-rank matrix

$$\Delta W = BA$$

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

- The initial update is 0
- After training, the updates are added back to the original checkpoint. So, the inference cost of the updated checkpoint is the same as the original checkpoint.



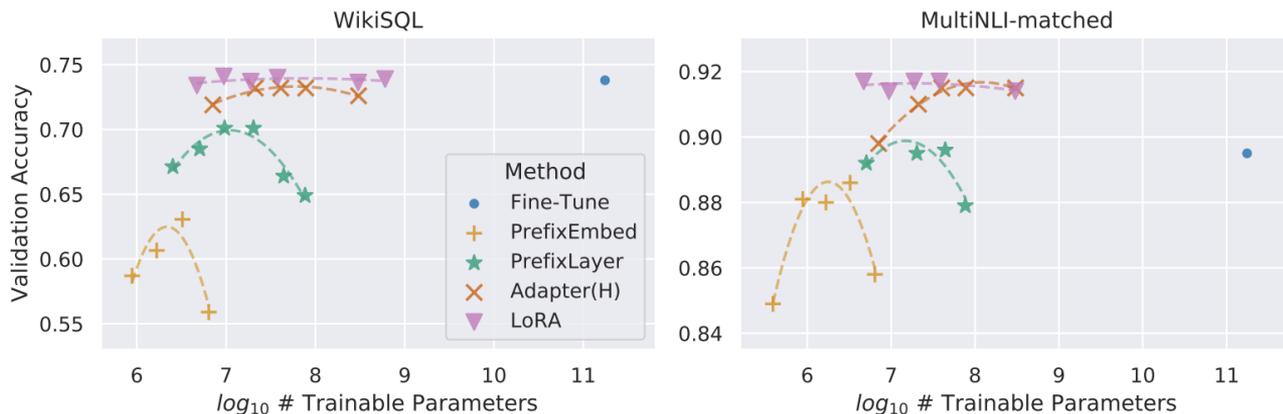
LoRA works better than other PEFT

- GPT-2 Median (355M) and Large (774M) models

Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter ^L)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter ^L)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter ^H)	11.09M	67.3 \pm .6	8.50 \pm .07	46.0 \pm .2	70.7 \pm .2	2.44 \pm .01
GPT-2 M (FT ^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	70.4\pm.1	8.85\pm.02	46.8\pm.2	71.8\pm.1	2.53\pm.02
<hr/>						
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter ^L)	0.88M	69.1 \pm .1	8.68 \pm .03	46.3 \pm .0	71.4 \pm .2	2.49\pm.0
GPT-2 L (Adapter ^L)	23.00M	68.9 \pm .3	8.70 \pm .04	46.1 \pm .1	71.3 \pm .2	2.45 \pm .02
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	70.4\pm.1	8.89\pm.02	46.8\pm.2	72.0\pm.2	2.47 \pm .02

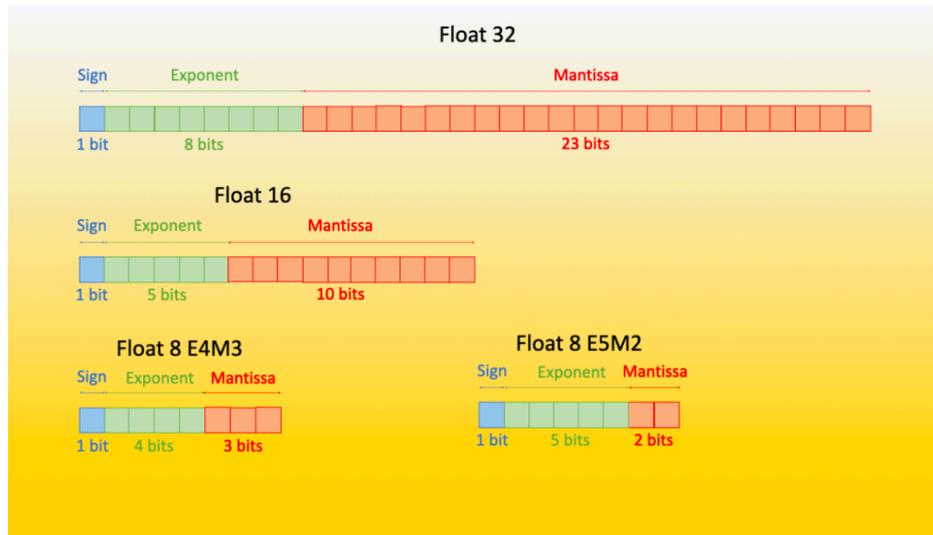
Scale to GPT-3 175 B

- Key benefit is the reduction in memory and storage usage:
 - Do not need to store the gradients of the frozen parameters
 - Reduce the VRAM consumption from **1.2TB to 350GB** during training
 - Use fewer GPUs and fewer I/O operations
- But LoRA still requires forward computation and back-propagation.
 - So, LoRA gives a 25% speedup (not 10x) compared to full fine-tuning.



QLoRA: Further Reduce Memory Usage

- Convert information in a high-precision data type to a low-precision data type
- Allows training a LLM in a single consumer GPU, e.g., 33B LLAMA in a single 24GB GPU



QLoRA

- Define a quantization method to convert a 16-bit model into a 4-bit model, using CPU before training
- Store the model weights in a special data type (4-bit NF), and compute the update using another data type (16-bit BF)

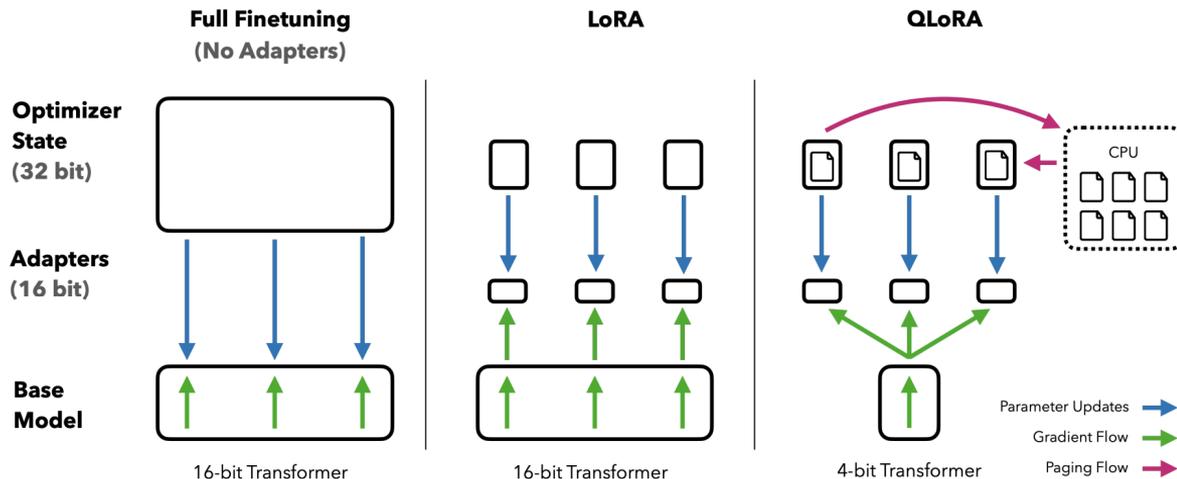


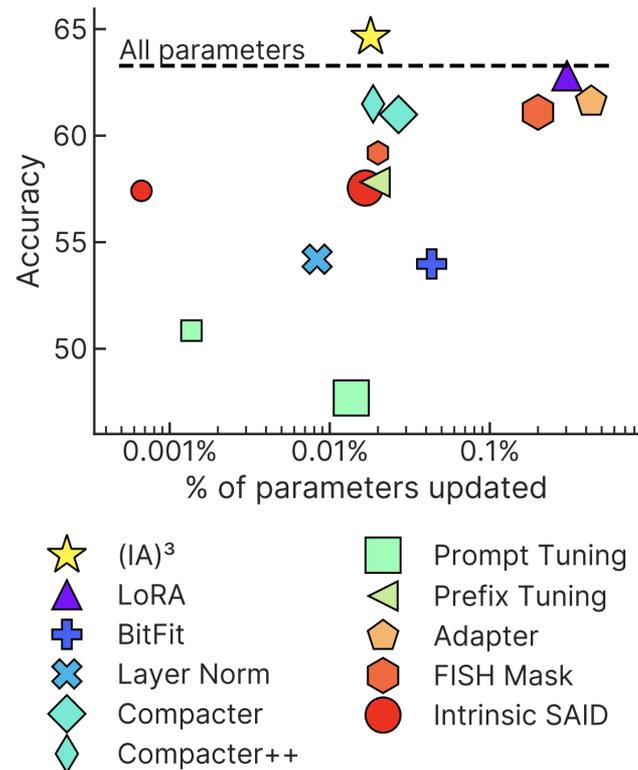
Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Fine-tuning a LLM in a single GPU

- Fine-tune a 65B LLM on a 48GB GPU (e.g., A6000)
- Fine-tune a 33B LLM on a 24GB GPU (e.g., RTX 3090, RTX 4090, A5000)

Model	Size	Elo
GPT-4	-	1348 \pm 1
Guanaco 65B	41 GB	1022 \pm 1
Guanaco 33B	21 GB	992 \pm 1
Vicuna 13B	26 GB	974 \pm 1
ChatGPT	-	966 \pm 1
Guanaco 13B	10 GB	916 \pm 1
Bard	-	902 \pm 1
Guanaco 7B	6 GB	879 \pm 1

Summary: How do these methods compare?



Aside: Extending PEFT to Compositionality

Extending PEFT to Compositionality

- Can we leverage the modular nature of PEFT methods as a means to model compositionality?
- Learning smaller and task specific modules allows us to compose them in different ways as a “mixture of experts”
- Combining known experts facilitates multi-task models and out of distribution generalization!

DEMIX Layers



References

- <https://www.leewayhertz.com/parameter-efficient-fine-tuning/>
- <https://huggingface.co/blog/peft>
- Ruder, S., Pfeiffer, J., & Vulić, I. (2022, December). Modular and Parameter-Efficient Fine-Tuning for NLP Models. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts (pp. 23-29).
- Houshy, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., ... & Gelly, S. (2019, May). Parameter-efficient transfer learning for NLP. In International Conference on Machine Learning (pp. 2790-2799). PMLR.
- He, R., Liu, L., Ye, H., Tan, Q., Ding, B., Cheng, L., ... & Si, L. (2021, August). On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers) (pp. 2208-2222).
- Lester, B., Al-Rfou, R., & Constant, N. (2021, November). The Power of Scale for Parameter-Efficient Prompt Tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (pp. 3045-3059).
- Li, X. L., & Liang, P. (2021, August). Prefix-Tuning: Optimizing Continuous Prompts for Generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers) (pp. 4582-4597).

References

- Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., & Raffel, C. A. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35, 1950-1965.
- Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021, October). LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.
- Li, C., Farkhoor, H., Liu, R., & Yosinski, J. (2018, February). Measuring the Intrinsic Dimension of Objective Landscapes. In *International Conference on Learning Representations*.
- Aghajanyan, A., Gupta, S., & Zettlemoyer, L. (2021, August). Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (pp. 7319-7328).
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.
- Gururangan, S., Lewis, M., Holtzman, A., Smith, N. A., & Zettlemoyer, L. (2022, July). DEMix Layers: Disentangling Domains for Modular Language Modeling. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 5557-5576).