Machine Learning for Natural Language Processing

Efficient Fine-tuning Methods for Language Models

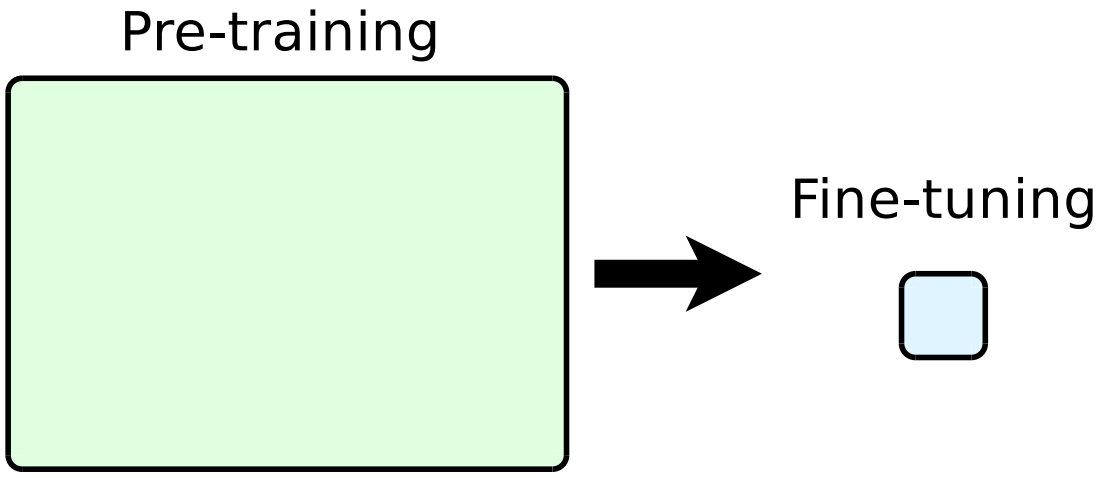
UNIVERSITY OF
GOTHENBURG

CHALMERS

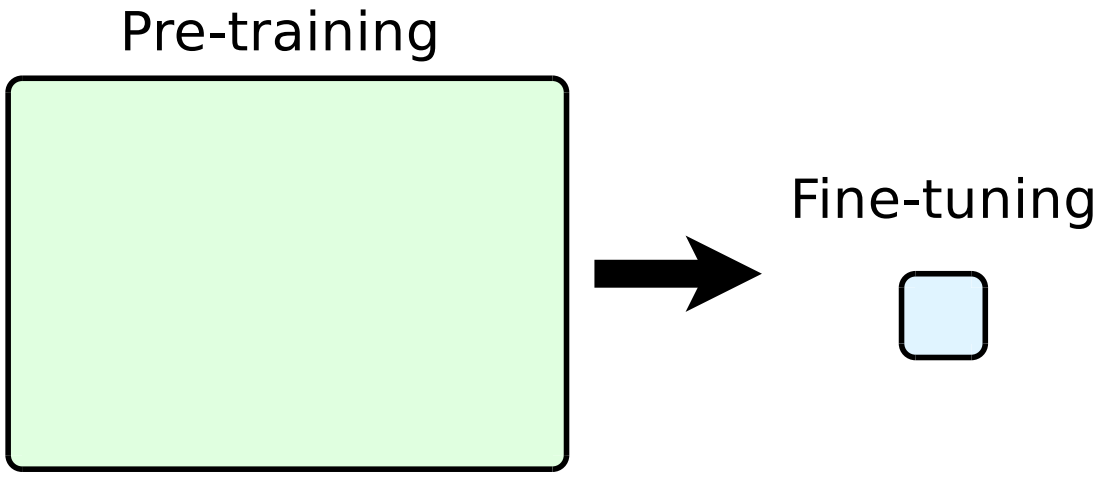
Richard Johansson

richajo@chalmers.se

Pre-training and fine-tuning



Pre-training and fine-tuning



Examples:

BERT $\rightarrow$ **BioBERT**

LLaMA $\rightarrow$ **Alpaca, Vicuna** (and other instruction tuning)

Full fine-tuning can be costly

It takes a lot of time

It takes a lot of space

It takes a lot of memory

Can we tune just the output head?

Simple approach: fine-tune just the top layer

But: as shown in the BERT paper (and elsewhere), fine-tuning the whole model is generally more effective

Prompt and prefix tuning

Prefix-Tuning: Optimizing Continuous Prompts for Generation

Xiang Lisa Li
Stanford University
xlisali@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

Abstract

Fine-tuning is the de facto way to leverage large pretrained language models to perform downstream tasks. However, it modifies all the language model parameters and therefore necessitates storing a full copy for each task. In this paper, we propose prefix-tuning, a lightweight alternative to fine-tuning for natural language generation tasks, which keeps language model parameters frozen, but optimizes a small *continuous task-specific* vector (called the prefix). Prefix-tuning draws inspiration from prompting, allowing subsequent tokens to attend to this prefix as if it were "virtual tokens". We apply prefix-tuning to GPT-2 for table-to-text generation and to BART for summarization. We find that by learning only 0.1% of the parameters, prefix-tuning obtains comparable performance in the full data setting, outperforms fine-tuning in low-data settings, and extrapolates better to examples with topics unseen during training.

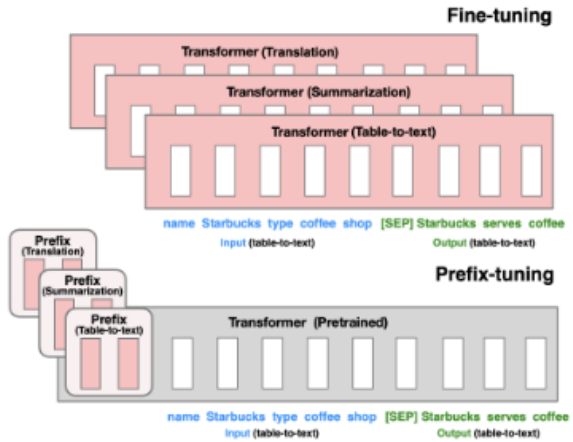
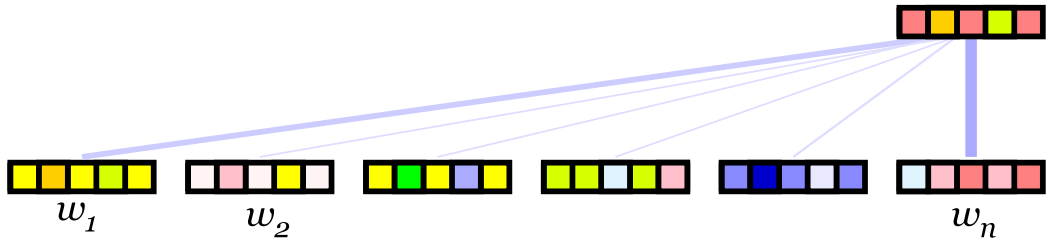
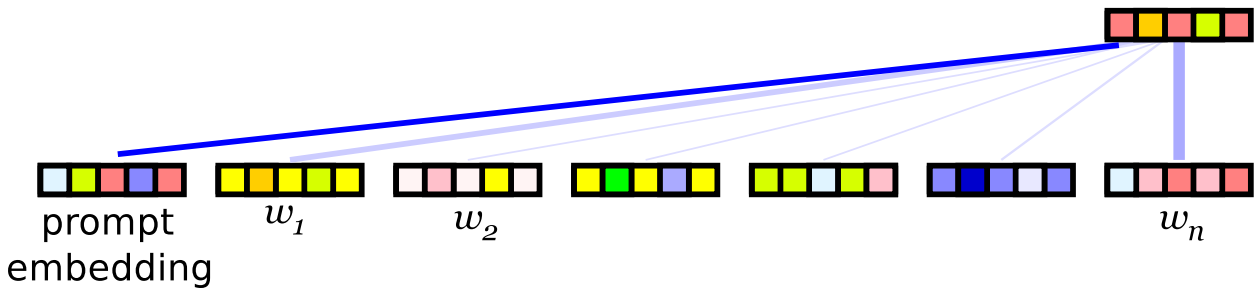


Figure 1: Fine-tuning (top) updates all Transformer parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the Transformer parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.

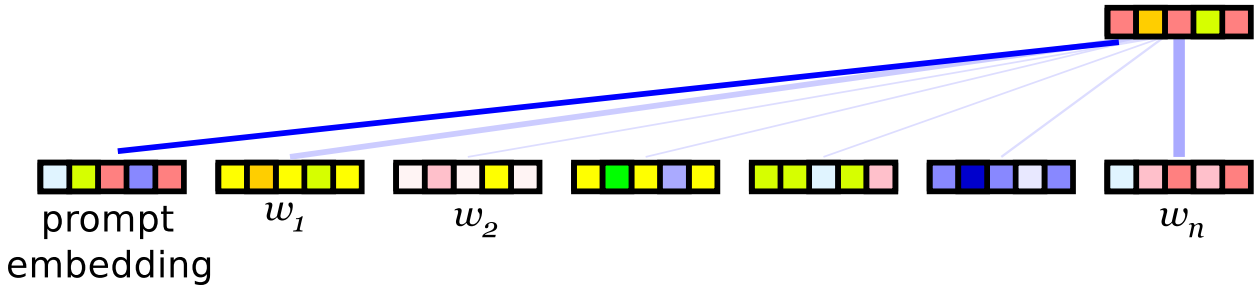
Prompt and prefix tuning: intuition



Prompt and prefix tuning: intuition



Prompt and prefix tuning: intuition



Variations:

One or more additional embeddings?

Parameters only in bottom or in several layers?

Adapters

Parameter-Efficient Transfer Learning for NLP

Neil Houlsby [1] **Andrei Giurgiu** [1 *] **Stanisław Jastrzębski** [2 *] **Bruna Morrone** [1] **Quentin de Laroussilhe** [1]
Andrea Gesmundo [1] **Mona Attariyan** [1] **Sylvain Gelly** [1]

Abstract

Fine-tuning large pre-trained models is an effective transfer mechanism in NLP. However, in the presence of many downstream tasks, fine-tuning is parameter inefficient: an entire new model is required for every task. As an alternative, we propose transfer with adapter modules. Adapter modules yield a compact and extensible model; they add only a few trainable parameters per task, and new tasks can be added without revisiting previous ones. The parameters of the original network remain fixed, yielding a high degree of parameter sharing. To demonstrate adapter's effectiveness, we transfer the recently proposed BERT Transformer model to 26 diverse text classification tasks, including the GLUE benchmark. Adapters attain near state-of-the-art performance, whilst adding only a few parameters per task. On GLUE, we attain within 0.4% of the performance of full fine-tuning, adding only 3.6% parameters per task. By contrast, fine-tuning trains 100% of

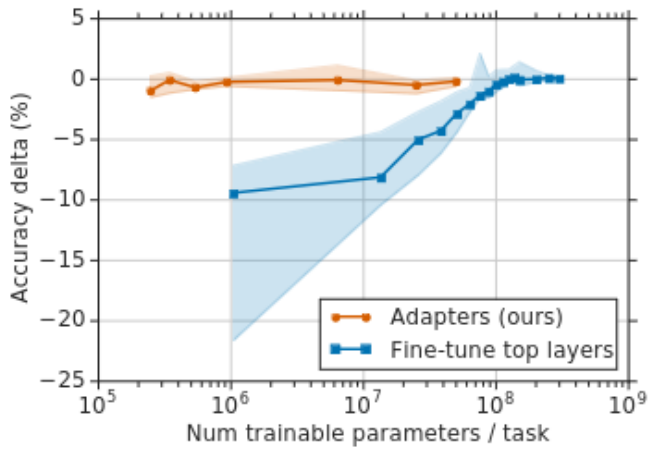
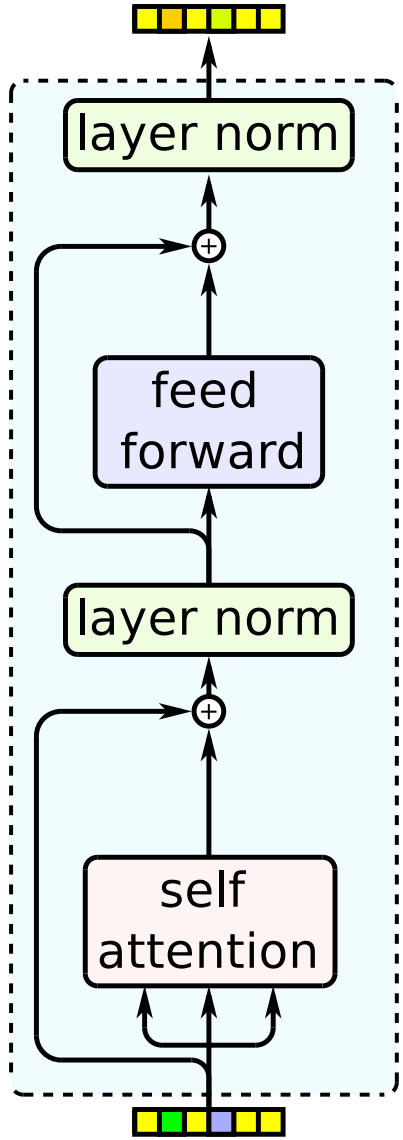
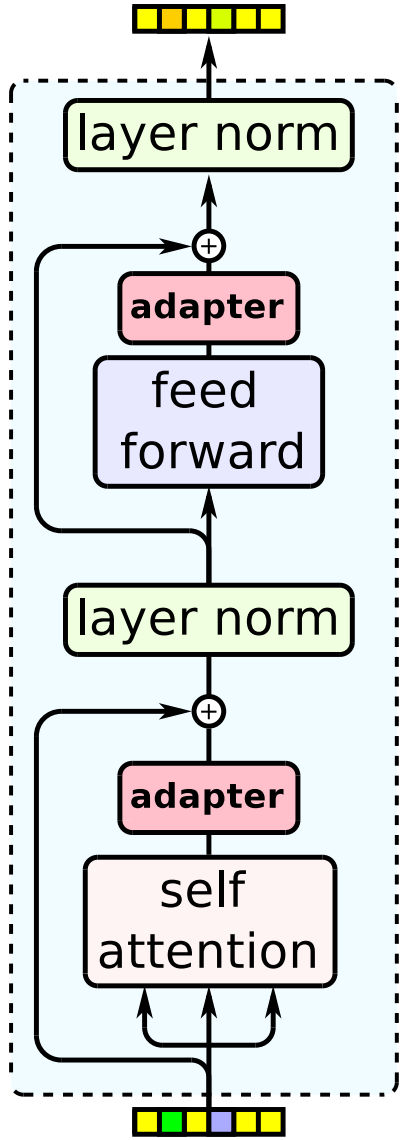


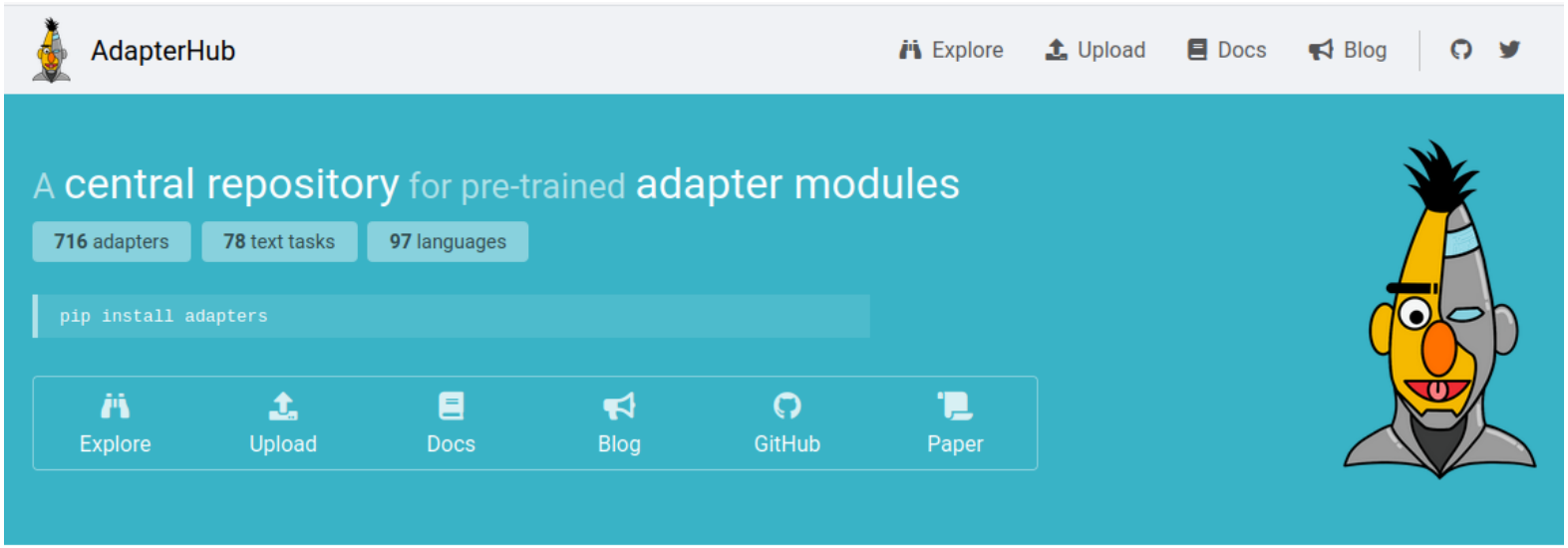
Figure 1. Trade-off between accuracy and number of trained task-specific parameters, for adapter tuning and fine-tuning. The y-axis is normalized by the performance of full fine-tuning, details in Section 3. The curves show the 20th, 50th, and 80th performance percentiles across nine tasks from the GLUE benchmark. Adapter-based tuning attains a similar performance to full fine-tuning with two orders of magnitude fewer trained parameters.

Adapters: intuition



Adapters: intuition





https://adapterhub.ml/

LoRA: Low-rank Adaptation

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

LoRA: Key ideas

Rewrite fine-tune weights in terms of difference to pre-trained:

$$W_{\mathsf{FT}} = W_{\mathsf{PT}} + \Delta W$$

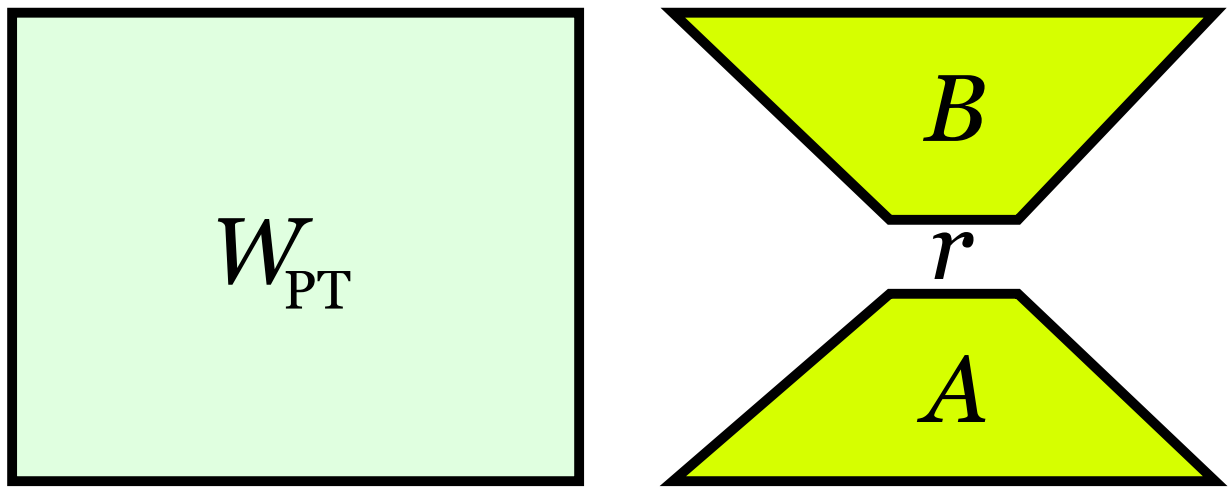
LoRA: Key ideas

Rewrite fine-tune weights in terms of difference to pre-trained:

$$W_{\text{FT}} = W_{\text{PT}} + \Delta W$$

Use a low-rank decomposition of the difference term:

$$W_{\text{FT}} = W_{\text{PT}} + BA$$



PEFT library in Hugging Face

🤗 PEFT

State-of-the-art Parameter-Efficient Fine-Tuning (PEFT) methods

Parameter-Efficient Fine-Tuning (PEFT) methods enable efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. Fine-tuning large-scale PLMs is often prohibitively costly. In this regard, PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs. Recent State-of-the-Art PEFT techniques achieve performance comparable to that of full fine-tuning.

Seamlessly integrated with 🤗 Accelerate for large scale models leveraging DeepSpeed and Big Model Inference.

Supported methods:

1. LoRA: LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS
2. Prefix Tuning: Prefix-Tuning: Optimizing Continuous Prompts for Generation, P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks
3. P-Tuning: GPT Understands, Too
4. Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning
5. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
6. $(IA)^3$: Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning
7. MultiTask Prompt Tuning: Multitask Prompt Tuning Enables Parameter-Efficient Transfer Learning
8. LoHa: FedPara: Low-Rank Hadamard Product for Communication-Efficient Federated Learning
9. LoKr: KronA: Parameter Efficient Tuning with Kronecker Adapter based on Navigating Text-To-Image Customization:From LyCORIS Fine-Tuning to Model Evaluation implementation
10. LoftQ: LoftQ: LoRA-Fine-Tuning-aware Quantization for Large Language Models
11. OFT: Controlling Text-to-Image Diffusion by Orthogonal Finetuning

https://github.com/huggingface/peft

Takeaways

Practical adaptation of LLMs typically involves lightweight fine-tuning

We typically add some small set of parameters that are fine-tuned

When combined with quantization (e.g. QLoRA), we can make it even more efficient in time and memory

References

T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. arXiv:2305.14314.

N. Houlsby, A. Giurgiu, and S. Jastrzebski et al. 2019. Parameter-efficient transfer learning for NLP. In *ICML*.

E. Hu, Y. Shen, and P. Wallis et al. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR*.

X. L. Li and P. Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL-IJCNLP*.

C. Poth, H. Sterz, and Indraneil Paul et al. 2023. Adapters: A unified library for parameter-efficient and modular transfer learning. arXiv:2311.11077.

R. Zhang, J. Han, and C. Liu et al. 2023. LLaMA-adapter: Efficient fine-tuning of language models with zero-init attention. arXiv:2303.16199.