CS769 Advanced NLP

Parameter-Efficient Fine-Tuning (PEFT)

Rheeya Uppaal, Junjie Hu

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

https://junjiehu.github.io/cs769-fall23/

Goals for Today

- 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 LMs

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

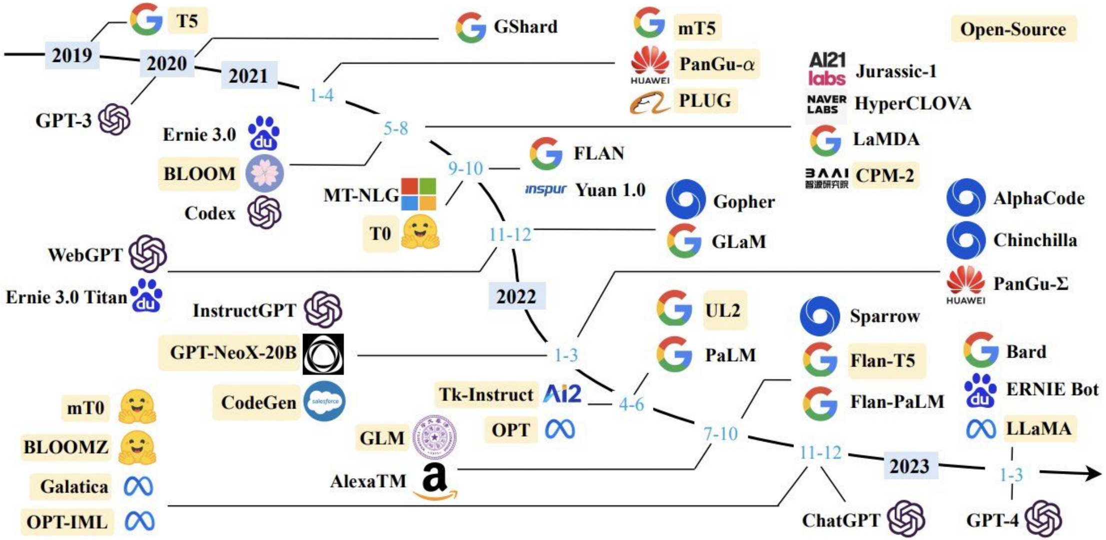
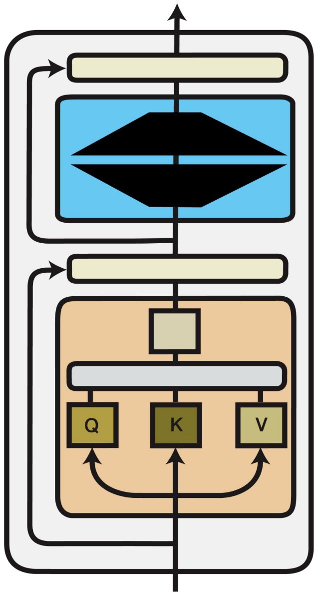
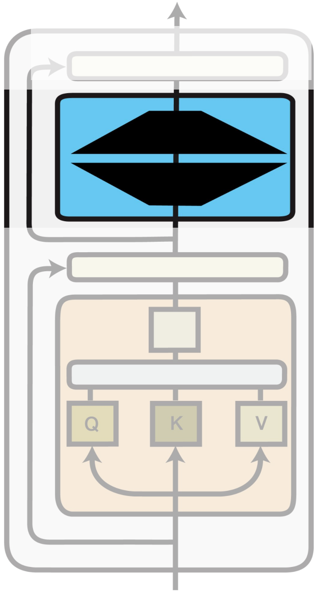


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

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

Slide from: EMNLP 2022 Tutorial on *Modular and Parameter-Efficient Fine-Tuning for NLP Models*

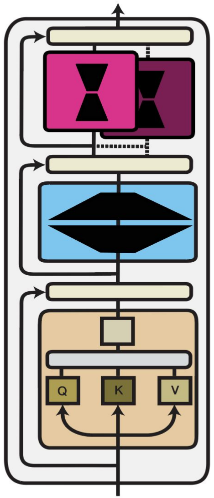
Motivation: Why PEFT?

- 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.
- 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

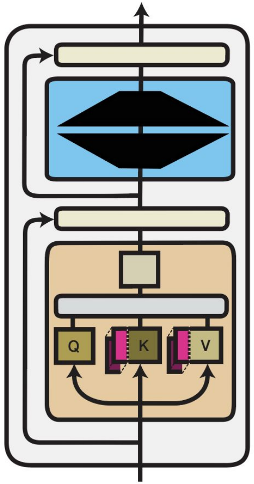
Comparison Between PEFT and Fine-tuning

	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

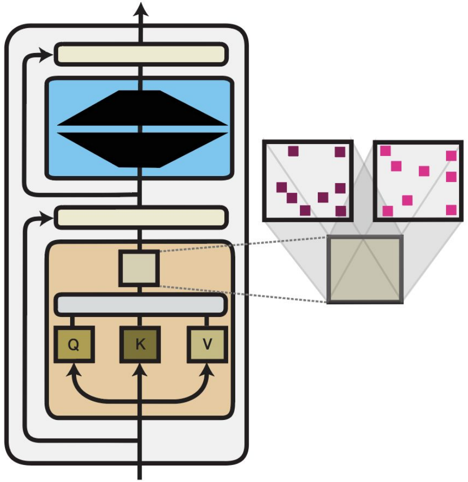
Three Computation Functions



Function Composition



Input Composition



Parameter Composition

Slide from: EMNLP 2022 Tutorial on *Modular and Parameter-Efficient Fine-Tuning for NLP Models*

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 $\theta_i, i = 1, \ldots, l$

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

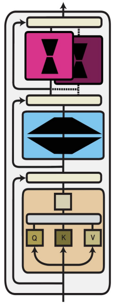
1. Function composition: $f_i'(\boldsymbol{x}) = f_{\theta_i} \odot f_\phi(\boldsymbol{x})$ — Function composition
2. Input composition: $f_i'(\boldsymbol{x}) = f_{\theta_i}([\boldsymbol{x}, \phi])$ — Concatenation
3. Parameter composition: $f_i'(\boldsymbol{x}) = f_{\theta_i \oplus \phi}(\boldsymbol{x})$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

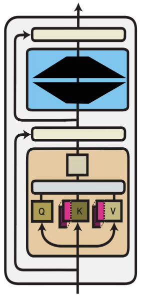
In practice, typically only the module parameters ϕ are updated while θ is fixed.

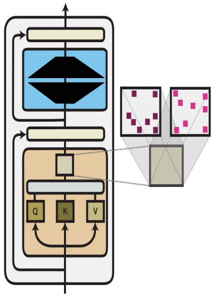
Slide from: EMNLP 2022 Tutorial on *Modular and Parameter-Efficient Fine-Tuning for NLP Models*

Three Computation Functions

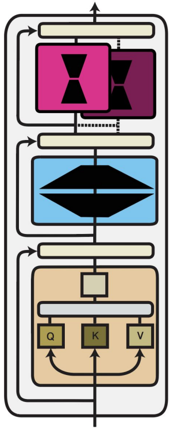
	Function Composition	Input Composition	Parameter Composition
Example Methods	Adapters, $(IA)^3$	Prompt Tuning, Prefix Tuning	LoRA, QLoRA, Pruning
Impact on Model Size	Additional modules in layers	Context window of model is increased	No increase in model size
Performance	Matches or outperforms fine-tuning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







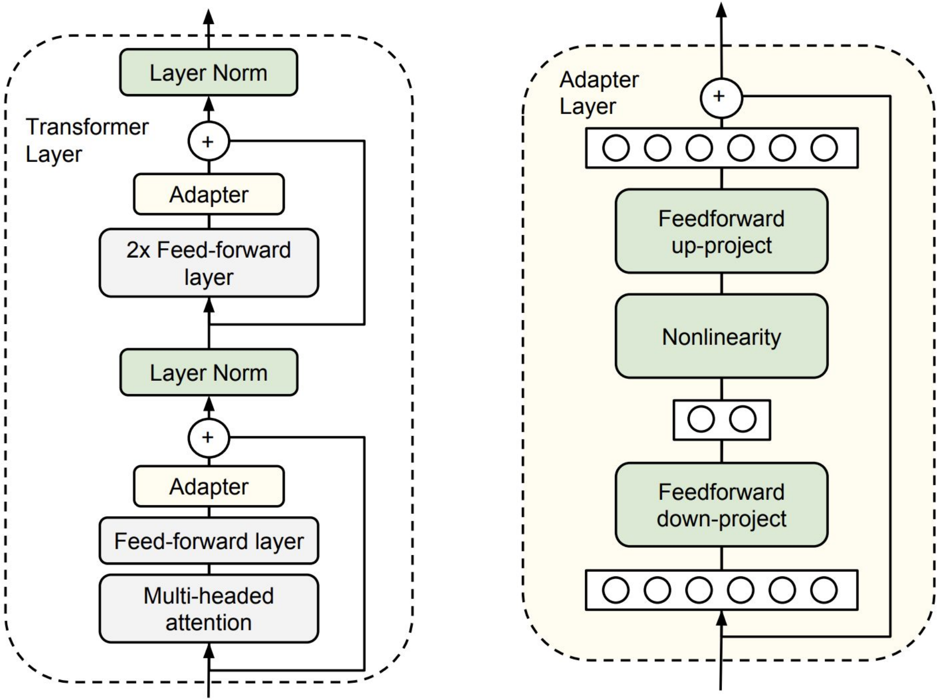
Function Composition:
Adapters, (IA)^3



Adapter

- An adapter is a MLP network.
- Add an adapter after the feed-forward layer in each Transformer layer

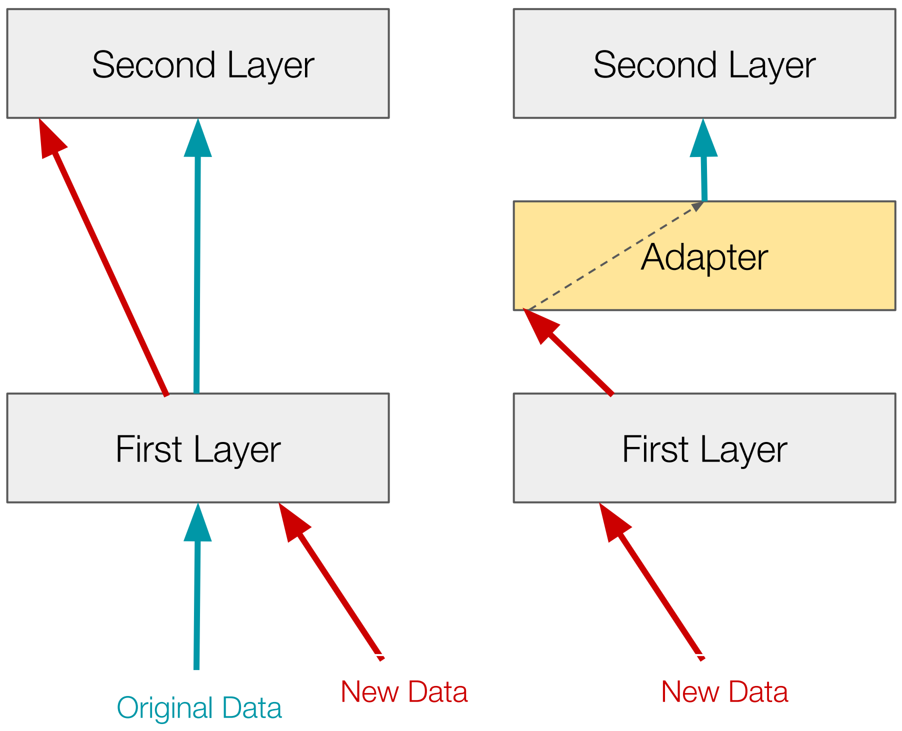
$$f_{\phi_i}(\boldsymbol{x}) = W^D(\sigma(W^U\boldsymbol{x}))$$



Houlsby et al ICML 2019. Parameter-Efficient Transfer Learning for NLP. https://arxiv.org/pdf/1902.00751.pdf

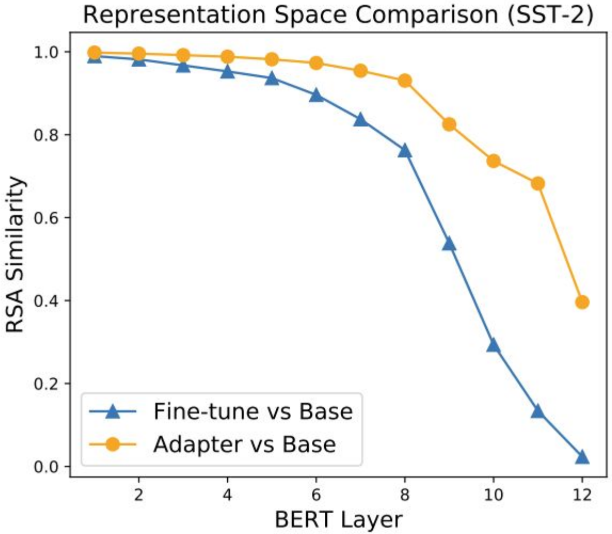
Why does this work? One Possible Intuition

- Oversimplified setting: Each layer is a matrix which transforms the input to a new space
- Adapters help "reroute" the data embeddings to what the upper layer expects



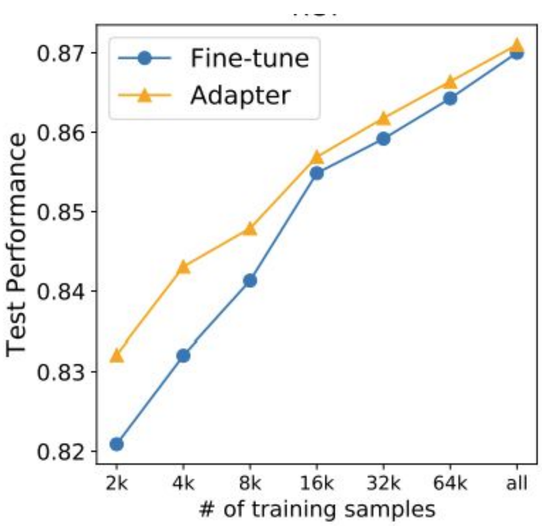
Why does this work? One Possible Intuition

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

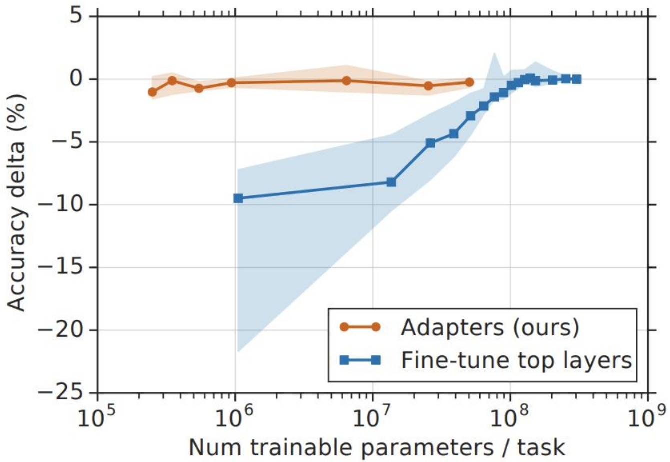


He et al. ACL 2021. On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. https://arxiv.org/abs/2106.03164

Adapters v.s. Full Fine-tuning



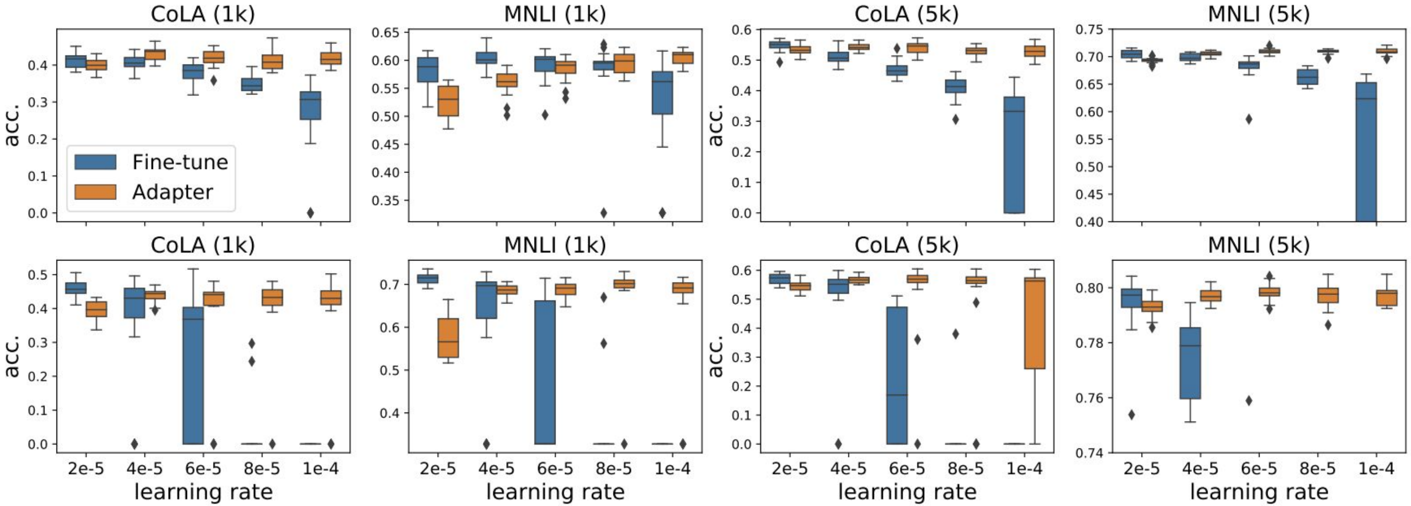
Adapters > FT with less data



Adapters > FT with less params

He et al. ACL 2021. On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. https://arxiv.org/abs/2106.03164
Houlsby et al ICML 2019. Parameter-Efficient Transfer Learning for NLP. https://arxiv.org/pdf/1902.00751.pdf

Adapters v.s. Full Fine-tuning



Adapters are less sensitive to hyperparameters like learning rate

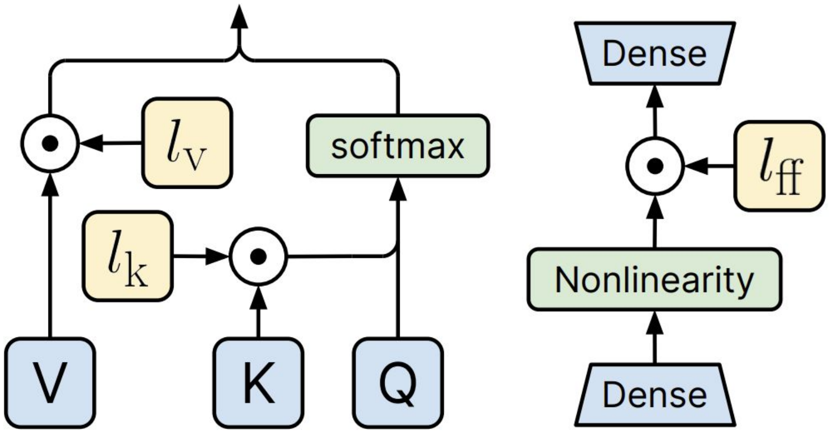
He et al. ACL 2021. On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. https://arxiv.org/abs/2106.03164

(IA)^3

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

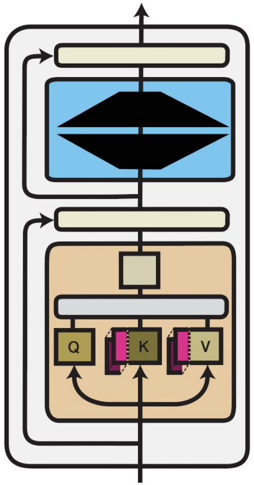
$$f_i'(\boldsymbol{x}) = f_{\theta_i}(\boldsymbol{x}) \circ \phi_i$$

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



Liu et al NeurIPS 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.
https://arxiv.org/abs/2205.05638

Input Composition:
Prompt-tuning, Prefix-tuning

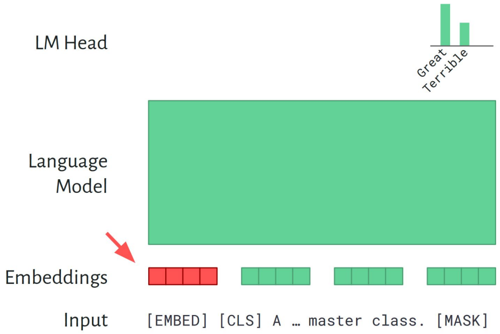


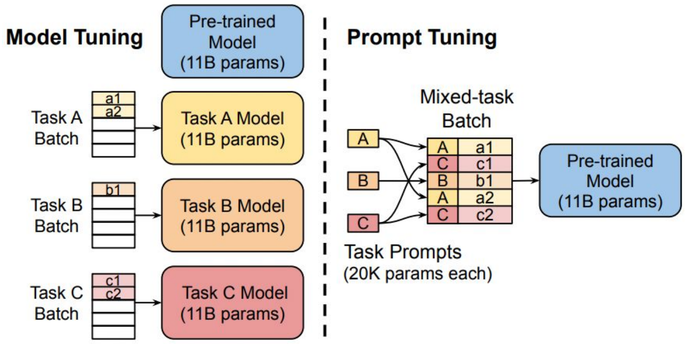
Motivation

- 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.
- 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.

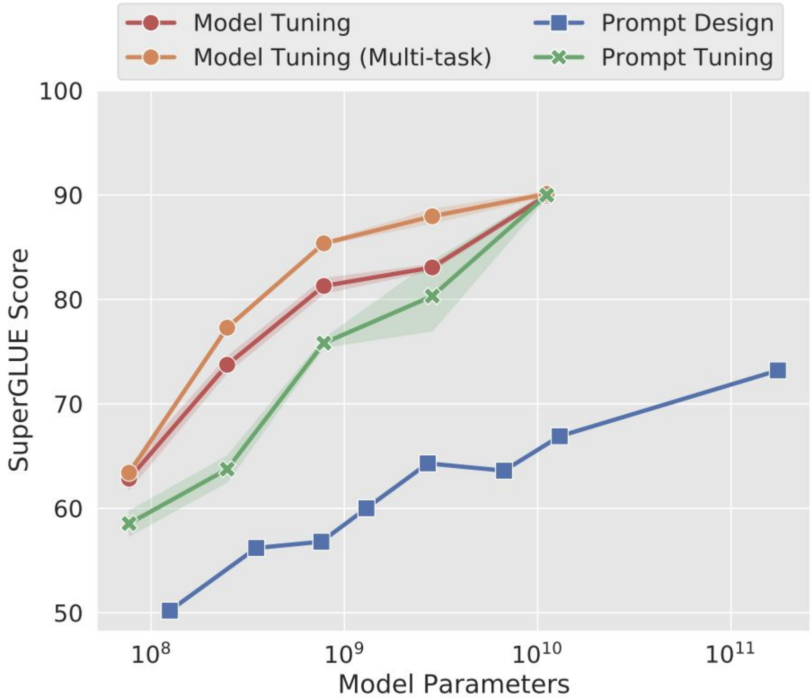




Lester et al. EMNLP 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. https://arxiv.org/abs/2104.08691

Prompt Tuning vs Model Tuning

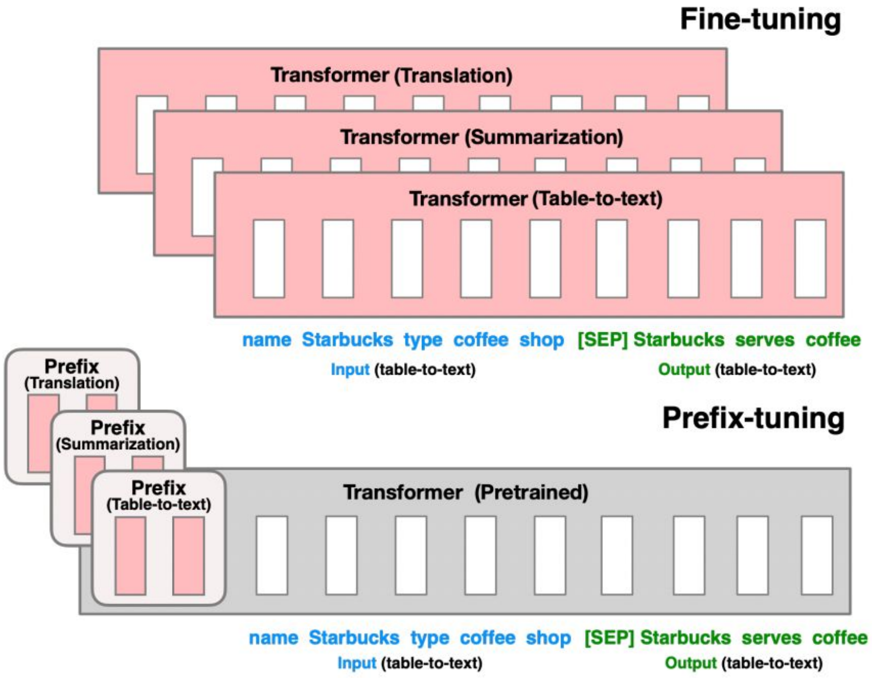
- As model size increases (e.g., T5-XXL 11B model), prompt tuning of T5 (green curve) matches the performance of (full) model tuning (red/orange curves) on SuperGLUE.
- Prompt design: few-shot in-context prediction by GPT-3 (blue curve) is still way worse than fine-tuning.



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

$$z = [\textsc{Prefix}; x; y],$$



Li et al. ACL 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. https://arxiv.org/pdf/2101.00190.pdf

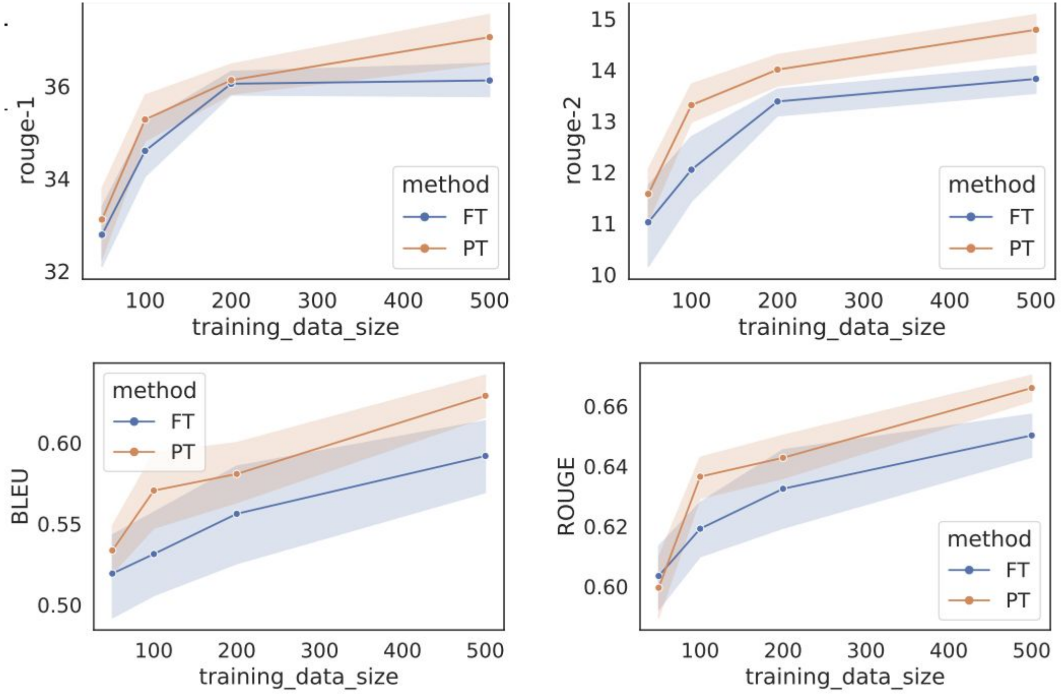
Effective on NLG tasks w/ 0.1% parameters

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

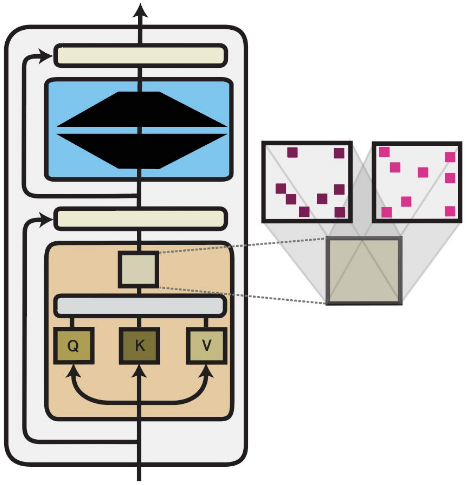
	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
GPT-2$_{MEDIUM}$																				
FINE-TUNE	68.2	8.62	**46.2**	71.0	2.47	**64.2**	27.7	46.5	**0.45**	0.30	0.38	**0.33**	0.76	0.53	46.2	**0.39**	**0.46**	**0.50**	**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



Parameter Composition: LoRA and Q-LoRA



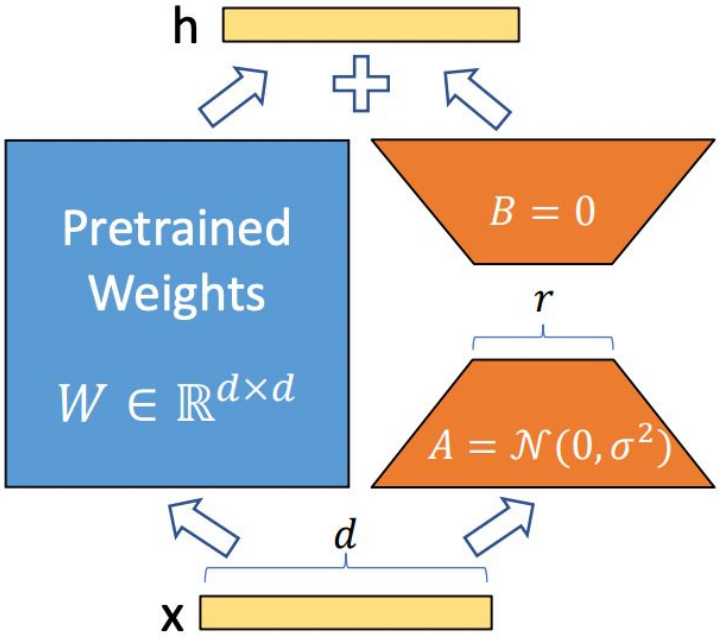
LoRA

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

$$\Delta W = BA$$

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- 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.



Hu et al. ICLR 2021. LoRA: Low-Rank Adaptation of Large Language Models. https://arxiv.org/pdf/2106.09685.pdf

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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Why does this work? Intrinsic Dimensions

- 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:

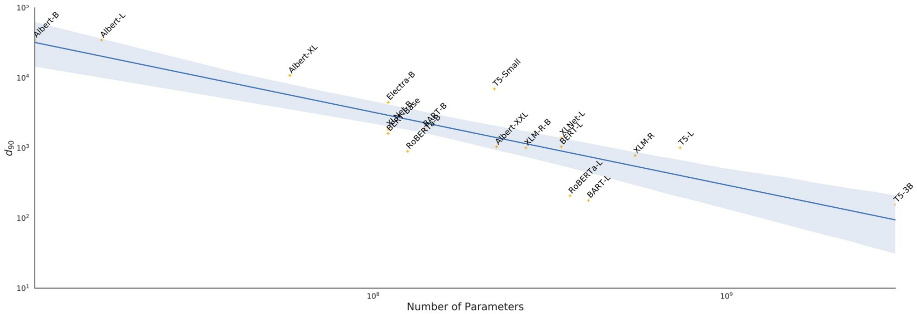
$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)}$$

- Intrinsic Dimensionality: Smallest *d* for which models achieve 90% of original accuracy
  - Intrinsic dimensionality decreases during pre-training
  - Larger models have lower intrinsic dimensionality

Li et al. ICLR 2018. Measuring the Intrinsic Dimension of Objective Landscapes. https://arxiv.org/abs/1804.08838
Aghajanyan et al. ACL 2021. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. https://arxiv.org/abs/2012.13255

One Possible Intuition

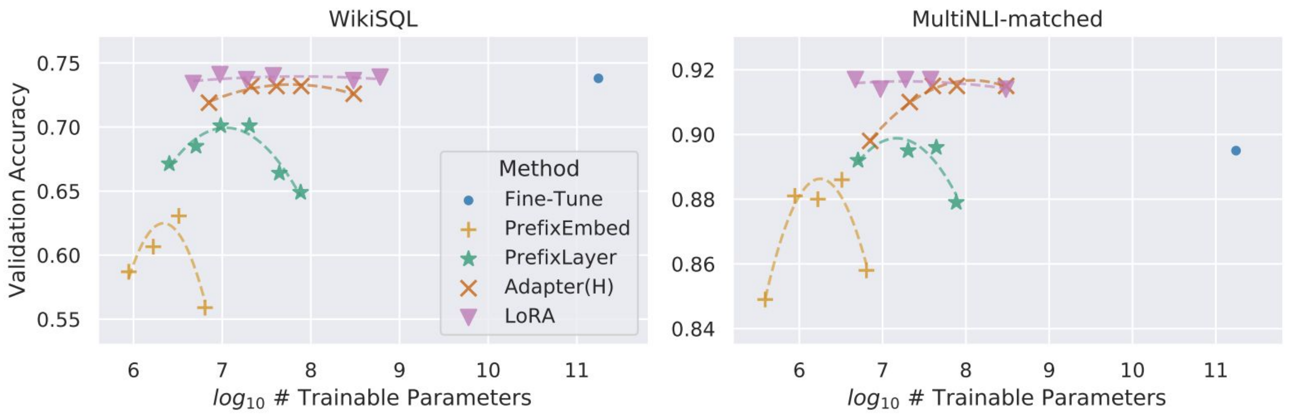
- Pre-training provides a strong initialization, i.e. a good $\theta_0^{(D)}$ in *D* dimensional space
- Due to this, the model only needs to explore a subspace of *d* dimensions during fine-tuning (through $\theta^{(d)}$), to learn the final weights $\theta^{(D)}$



Li et al. ICLR 2018. Measuring the Intrinsic Dimension of Objective Landscapes. https://arxiv.org/abs/1804.08838
Aghajanyan et al. ACL 2021. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. https://arxiv.org/abs/2012.13255

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

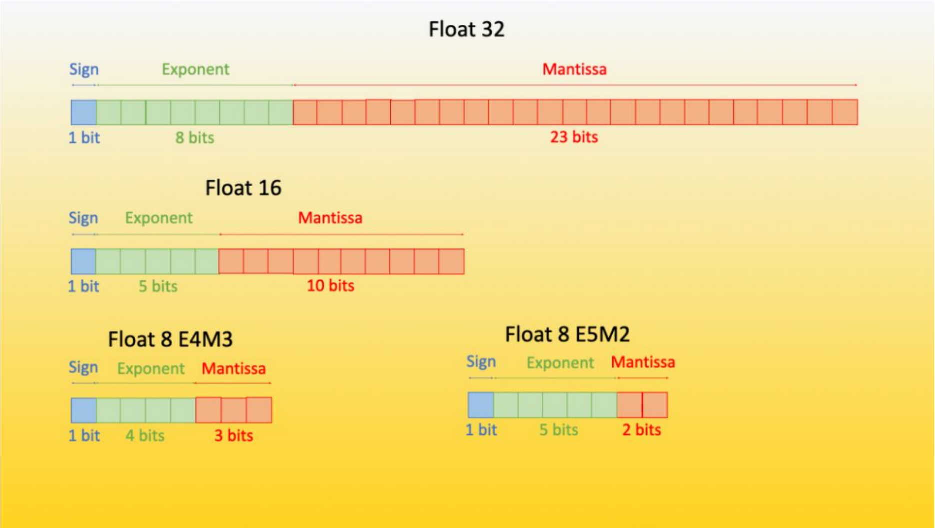


Image source: https://huggingface.co/blog/4bit-transformers-bitsandbytes

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)

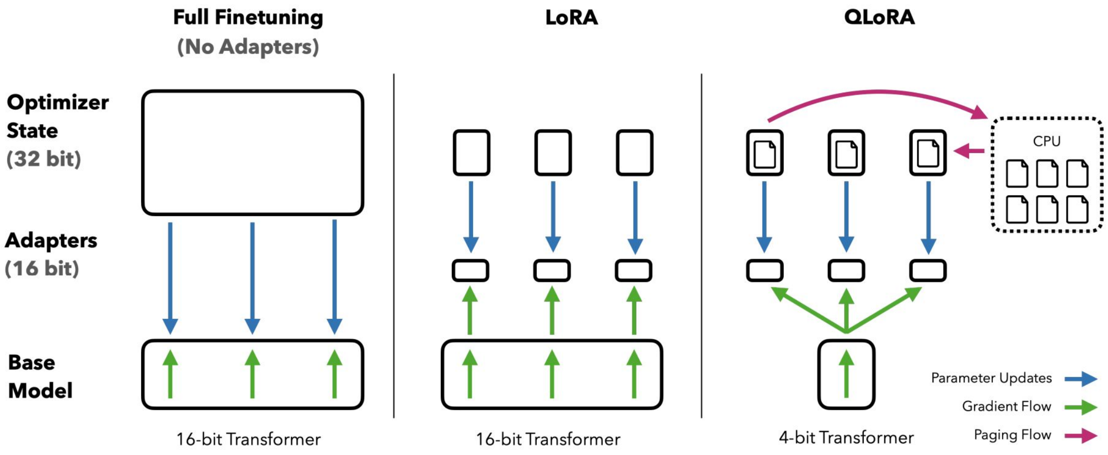


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

QLoRA: Background

- **Block-wise k-bit Quantization**: discretize an input from a high-precision representation to a low-precision representation.
  - Example: quantize a 32-bit float tensor into a 8-bit integer tensor with range [-127, 127] with a quantization constant *c* (input dependent).

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}),$$

  - Dequantization is the inverse operation:

$$\text{dequant}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}}$$

QLoRA: Double Quantization

- **Double Quantization**: (1) first quantize the weight matrix, and (2) then further quantize the quantization constants for additional memory savings.
    - Example: using 32-bit constants and a blocksize of 64 for a weight W, quantization constants add 32/64 = 0.5 bits per parameter on average

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \underbrace{\text{dequant}(\underbrace{\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}})}_{\text{First dequantization}}, \mathbf{W}^{\text{4bit}})}_{\text{Second dequantization}} = \mathbf{W}^{\text{BF16}}$$

Second dequantization

First dequantization

QLoRA

- QLoRA use a single linear layer in the quantized based model with a single LoRA adapter (recall LoRA update: $h = W_0 x + BAx$

$$\mathbf{Y}^{\mathrm{BF16}} = \mathbf{X}^{\mathrm{BF16}} \mathrm{doubleDequant}(c_1^{\mathrm{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\mathrm{NF4}}) + \mathbf{X}^{\mathrm{BF16}} \mathbf{L}_1^{\mathrm{BF16}} \mathbf{L}_2^{\mathrm{BF16}},$$

- Summary: QLoRA has one **storage data type** (usually **4-bit NormalFloat**) and a **computation data type** (**16-bit BrainFloat**). They dequantize the storage data type to the computation data type to perform the forward and backward pass, but they only compute the weight gradients for the LoRA parameters which use 16-bit BrainFloat.

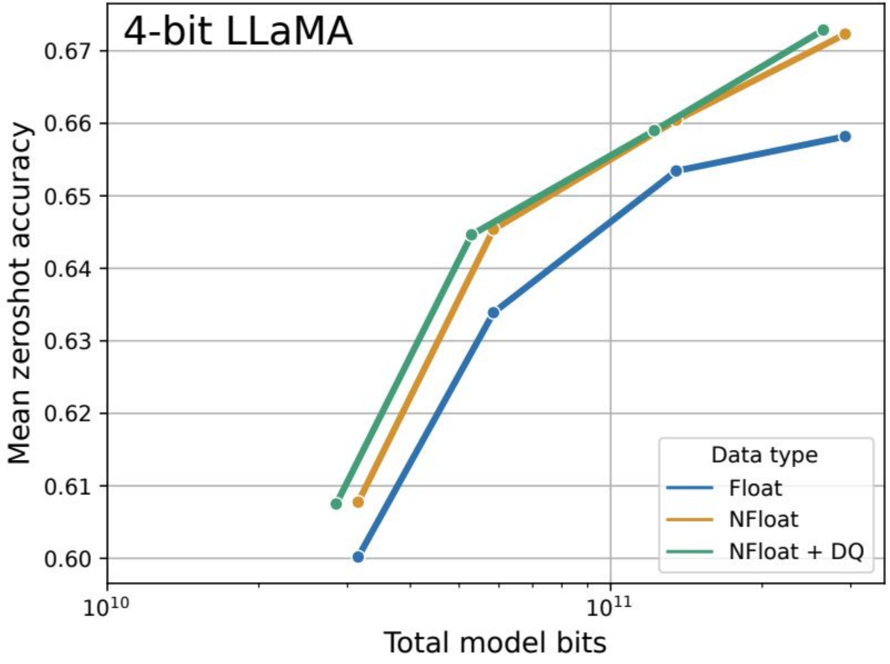
Fine-tuning a LLM in a single GPU

- 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 ± 1
Guanaco 65B	41 GB	1022 ± 1
Guanaco 33B	21 GB	992 ± 1
Vicuna 13B	26 GB	974 ± 1
ChatGPT	-	966 ± 1
Guanaco 13B	10 GB	916 ± 1
Bard	-	902 ± 1
Guanaco 7B	6 GB	879 ± 1

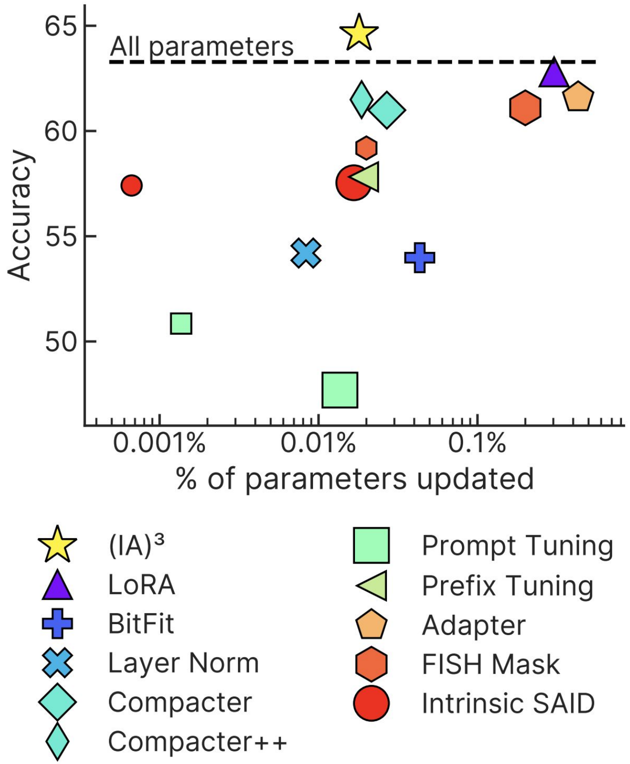
4-bit NF vs 4-bit Floating Point

- Using the same amount of model bits, **4-bit NF** yields better performance than **4-bit Floating point** (orange vs blue curves).
- **Double quantization** reduces the memory footprint without degrading performance (orange vs green curves). For instance, save ~3GB GPU RAM for a 65B LLM



Mean zero-shot accuracy on 5 datasets using LLAMA w/ different 4-bit data type.

Summary: How do these methods compare?



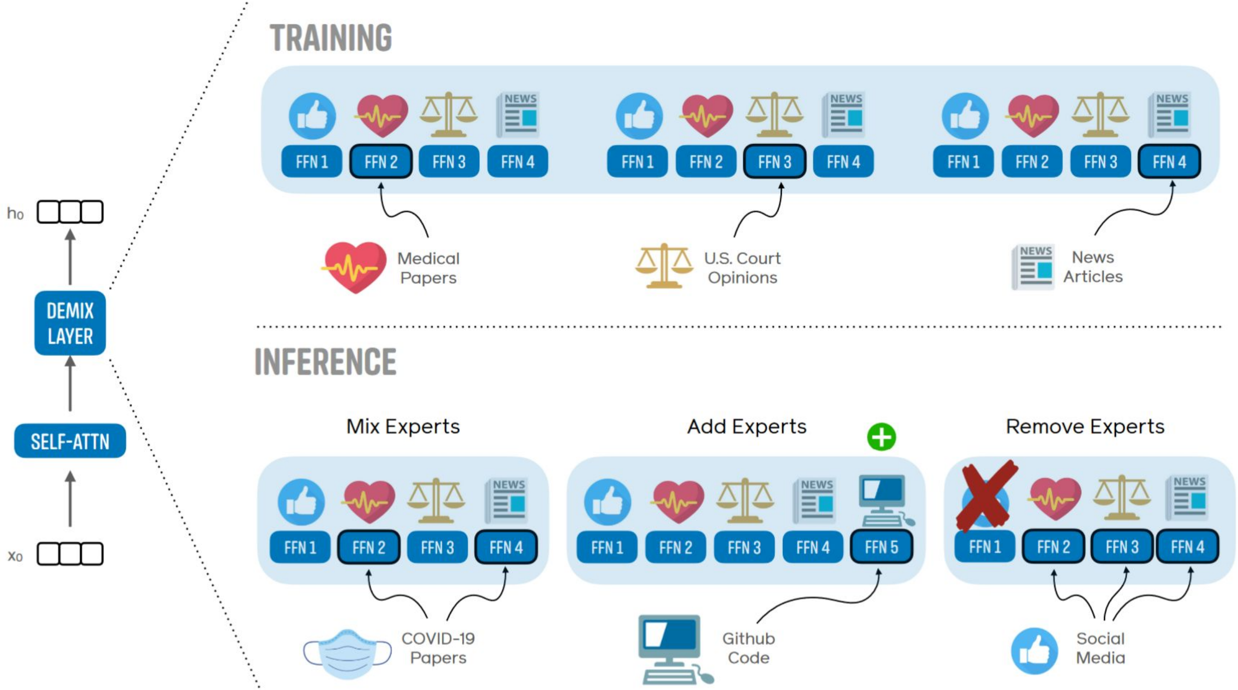
Liu et al NeurIPS 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.
https://arxiv.org/abs/2205.05638

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



Gururangan et al. NAACL 2022. DEMix Layers: Disentangling Domains for Modular Language Modeling. https://arxiv.org/abs/2108.05036

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).
- Houlsby, 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).