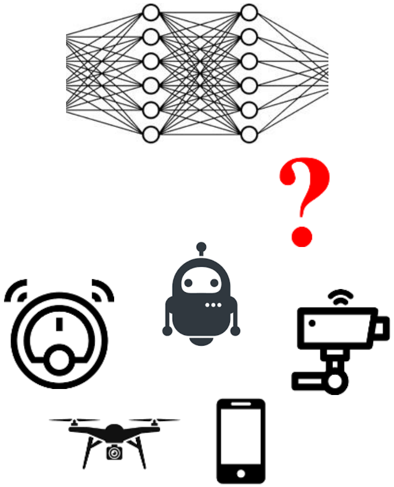
CAP 5516
Medical Image Computing
(Spring 2025)

Dr. Chen Chen
Associate Professor
Center for Research in Computer Vision (CRCV)
University of Central Florida
Office: HEC 221
Email: chen.chen@crcv.ucf.edu
Web: https://www.crcv.ucf.edu/chenchen/

Lecture 13
Efficient Deep Learning (3)

Research Challenges

Large models

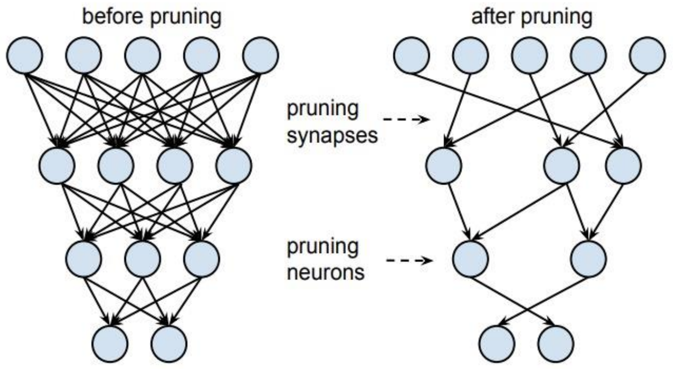


Devices with varying resources

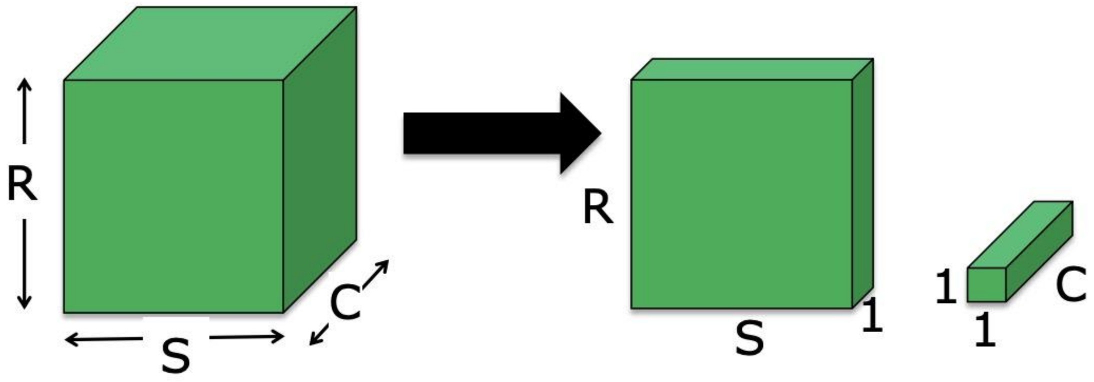
Efficient Neural Networks Design

Credit: Vivienne Sze

Network Pruning

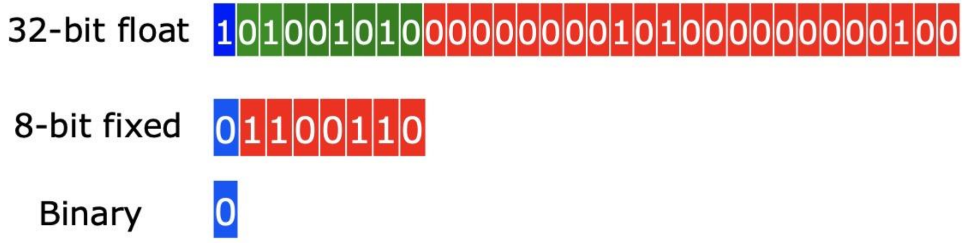


Efficient Network Architectures

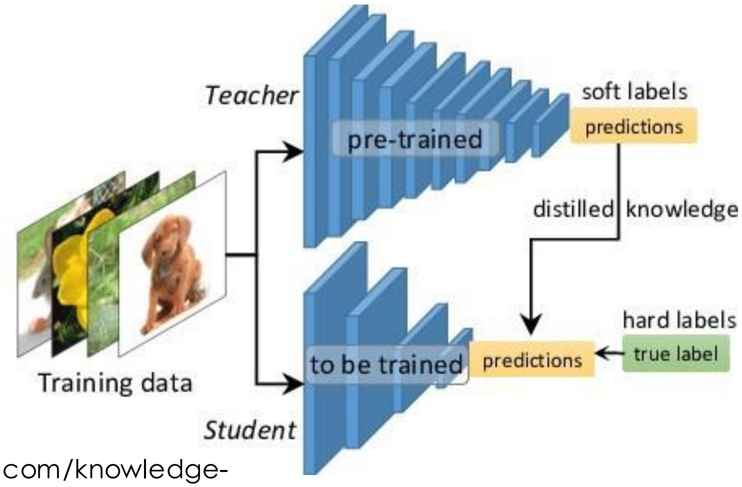


[MobileNets, ShuffleNets, AdderNet]

Reduce Precision



Knowledge Distillation



Source: https://towardsdatascience.com/knowledge-distillation-simplified-dd4973dbc764

UCF Center for Research in Computer Vision

Med-DANet-V2

- Shen, Haoran, Yifu Zhang, Wenxuan Wang, Chen Chen, Jing Liu, Shanshan Song, and Jiangyun Li. "Med-DANet V2: A Flexible Dynamic Architecture for Efficient Medical Volumetric Segmentation." In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 7871-7881. 2024.

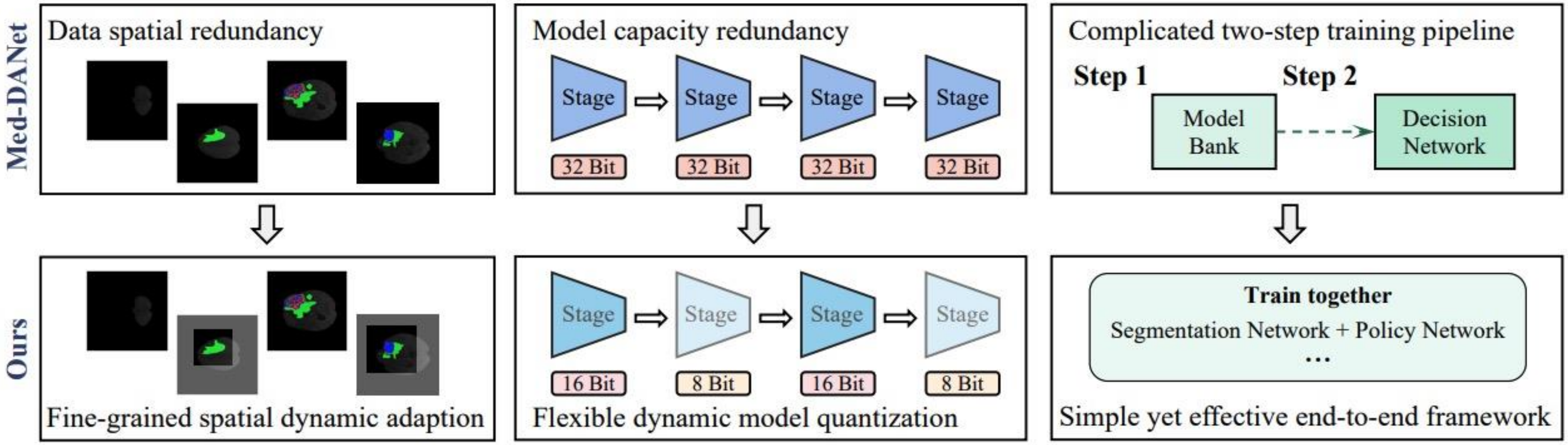
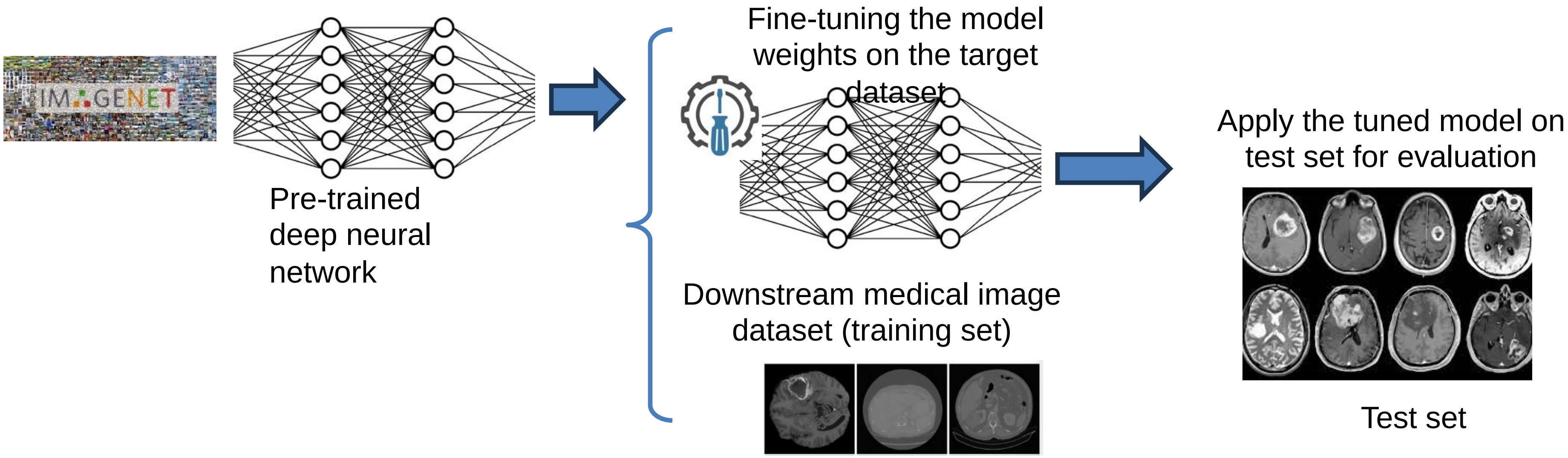

Med-DANet
Data spatial redundancy
Model capacity redundancy
Stage
Stage
Stage
Stage
32 Bit
32 Bit
32 Bit
32 Bit
Complicated two-step training pipeline
Step 1
Step 2
Model Bank
Decision Network
Ours
Stage
Stage
Stage
Stage
16 Bit
8 Bit
16 Bit
8 Bit
Train together
Segmentation Network + Policy Network
...
Fine-grained spatial dynamic adaption
Flexible dynamic model quantization
Simple yet effective end-to-end framework

Figure 1. The comparison between the previous dynamic network Med-DANet and our proposed Med-DANet V2 (Ours).

Motivation

- The standard “Pre-training then fine-tuning” paradigm for medical image segmentation



Pre-trained deep neural network

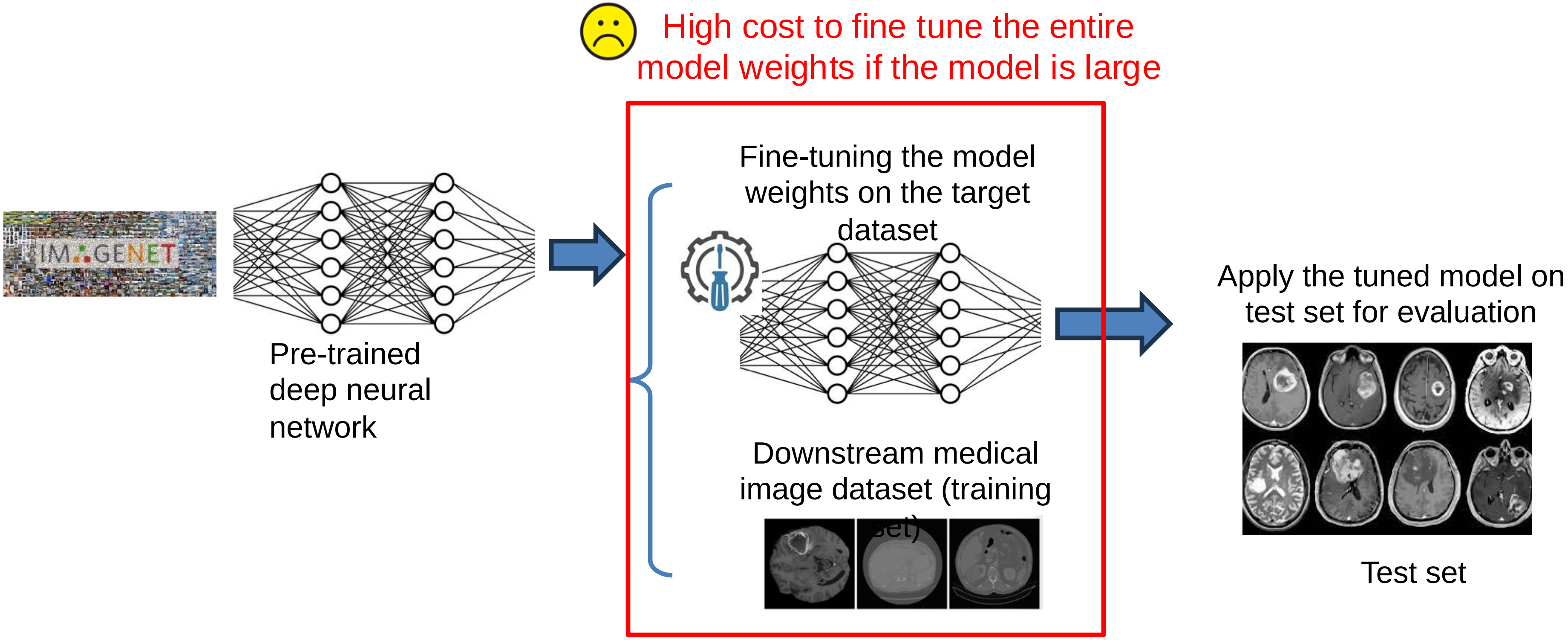
Fine-tuning the model weights on the target dataset

Downstream medical image dataset (training set)

Apply the tuned model on test set for evaluation

Test set

Motivation



☹ High cost to fine tune the entire model weights if the model is large

Pre-trained deep neural network

Fine-tuning the model weights on the target dataset

Downstream medical image dataset (training set)

Apply the tuned model on test set for evaluation

Test set

Parameter Efficient Fine Tuning for Transformer models

Parameter Efficient Fine Tuning

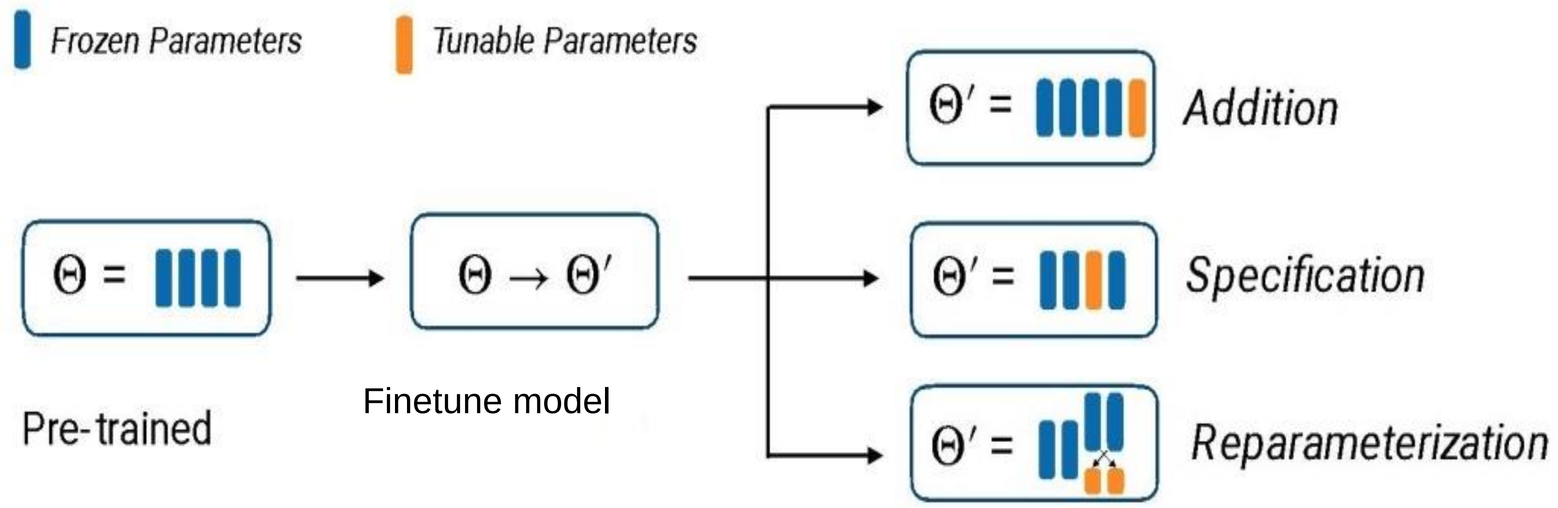
- Idea: Rather than finetuning the entire model, we finetune only small amounts of weights.

- **Efficiency:** Only a small set of parameters is trained, leading to faster convergence and reduced resource usage.

- **Practical Benefits:**
  - Reduces risk of overfitting when data is scarce.
  - Enables rapid adaptation to new tasks without altering the entire network.

Parameter Efficient Fine Tuning

- 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?



Finetune model

Slide Credit: Daphne Ippolito, Chenyan Xiong

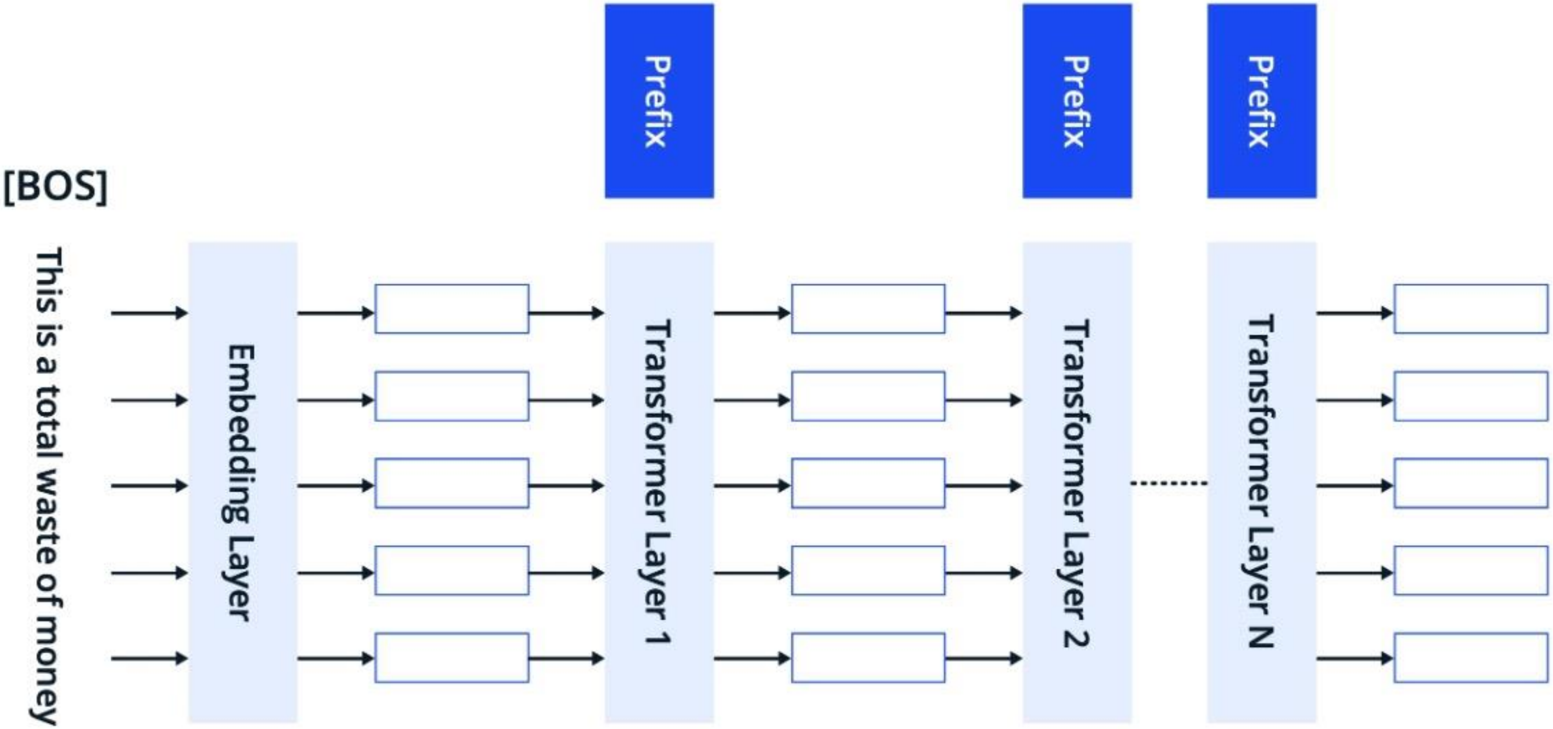
Parameter Efficient Fine Tuning – Prefix Tuning

Key Idea:

- Instead of updating all model weights, learn a set of additional “prefix” tokens.

Implementation:

- **Prefix Tokens:** Learnable continuous vectors prepended to the input sequence. These prefixes are injected into every layer of the Transformer
- **Frozen Backbone:** The main pretrained model remains unchanged during fine-tuning.



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

Parameter Efficient Fine Tuning – Prefix Tuning

Training Dynamics

- **Frozen Model:**

The original model parameters remain frozen, ensuring that only the prefix tokens are updated.

- **Backpropagation:**

Gradients are computed only for the prefix tokens, allowing them to learn task-specific representations.

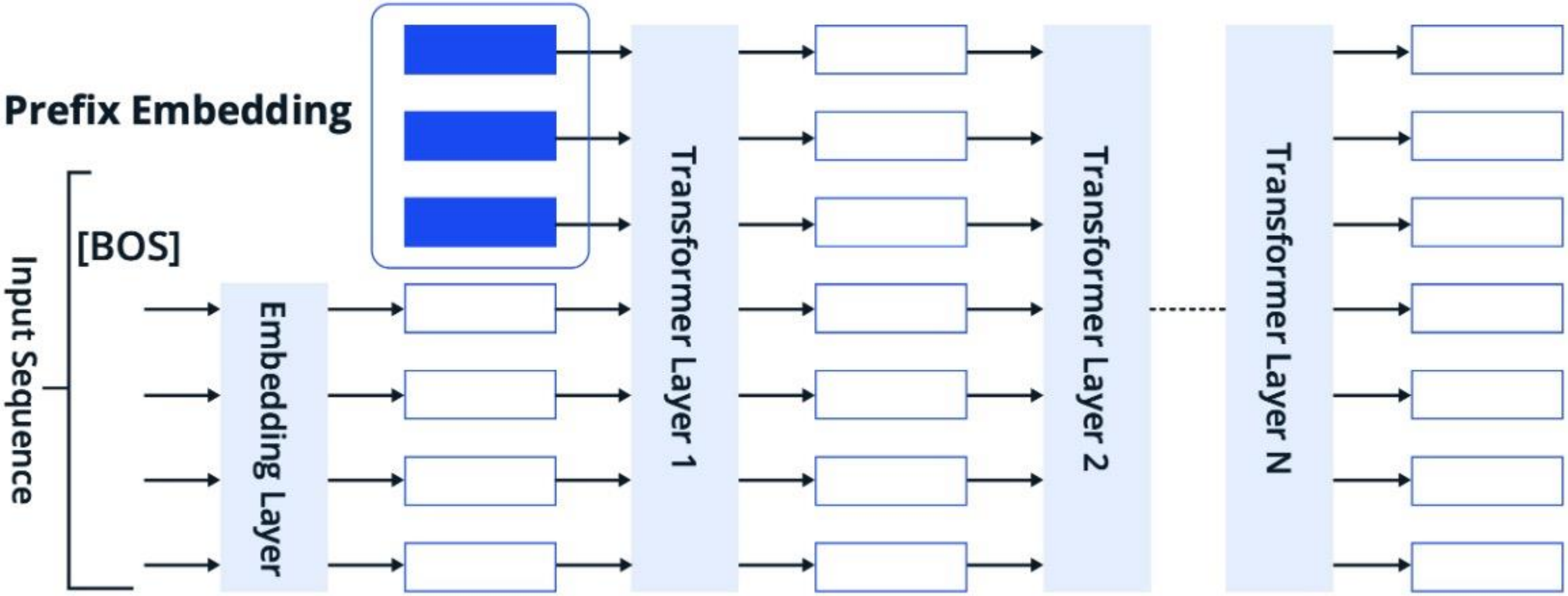
- **Outcome:**

The modified attention mechanism effectively “steers” the model toward desired behaviors, achieving performance comparable to full fine-tuning but at a fraction of the cost.

Parameter Efficient Fine Tuning – Prompt Tuning

Implementation:

- **Prompt Tokens:** A sequence of learnable vectors, prepended to the input.
- **Frozen Backbone:** The main pretrained model remains unchanged during training.



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

Parameter Efficient Fine Tuning – Prompt Tuning

During Training and Inference

- Prepend these prompt tokens to every input sequence.
- The model processes the prompt tokens along with the actual input, effectively incorporating task-specific context.

Training Dynamics

- **Frozen Model:**

Only the prompt tokens are updated; the rest of the model's parameters remain fixed.

- **Learning Process:**

The prompt tokens are optimized via backpropagation to maximize performance on the task.

Uses Cases and Trade-offs

Prefix Tuning:

Best For:

- Tasks requiring more nuanced control over the model's internal representations.
- Scenarios where achieving near full fine-tuning performance is critical.

Prompt Tuning:

Best For:

- Quick adaptation to new tasks with minimal changes.
- Situations with limited computational resources and simpler integration requirements.

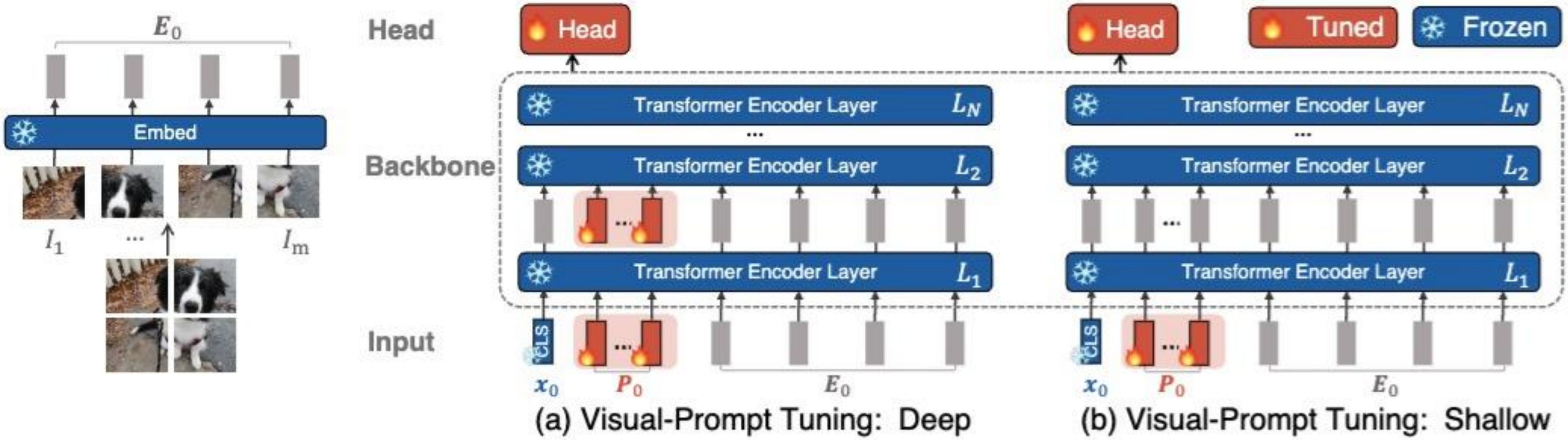
Trade-offs:

Complexity vs. Simplicity:

- Prefix tuning is slightly more complex but can yield finer control.
- Prompt tuning is simpler and more straightforward, but might not match the performance gains in all cases.

Parameter Efficient Fine Tuning – Visual Prompt Tuning

- Visual Prompt Tuning


E_0
Embed
I_1
...
I_m
Head
Backbone
Input
Head
Head
Tuned
Frozen
Transformer Encoder Layer L_N
Transformer Encoder Layer L_2
Transformer Encoder Layer L_1
CLS
x_0
P_0
E_0

(a) Visual-Prompt Tuning: Deep

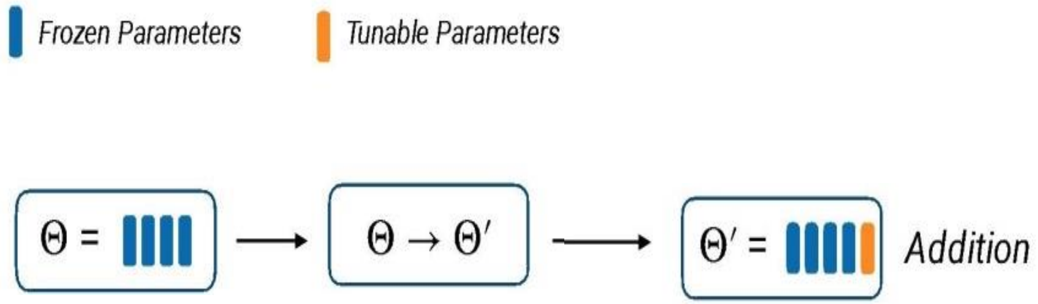
(b) Visual-Prompt Tuning: Shallow

Fig. 2. Overview of our proposed Visual-Prompt Tuning. We explore two variants: (a) prepend a set of learnable parameters to each Transformer encoder layer's input (VPT-DEEP); (b) only insert the prompt parameters to the first layer's input (VPT-SHALLOW). During training on downstream tasks, only the parameters of prompts and linear head are updated while the whole Transformer encoder is frozen.

Jia, M., Tang, L., Chen, B. C., Cardie, C., Belongie, S., Hariharan, B., & Lim, S. N. (2022, October). Visual prompt tuning. In European Conference on Computer Vision (pp. 709-727). Cham: Springer Nature Switzerland.

Parameter Efficient Fine Tuning - Adaptor

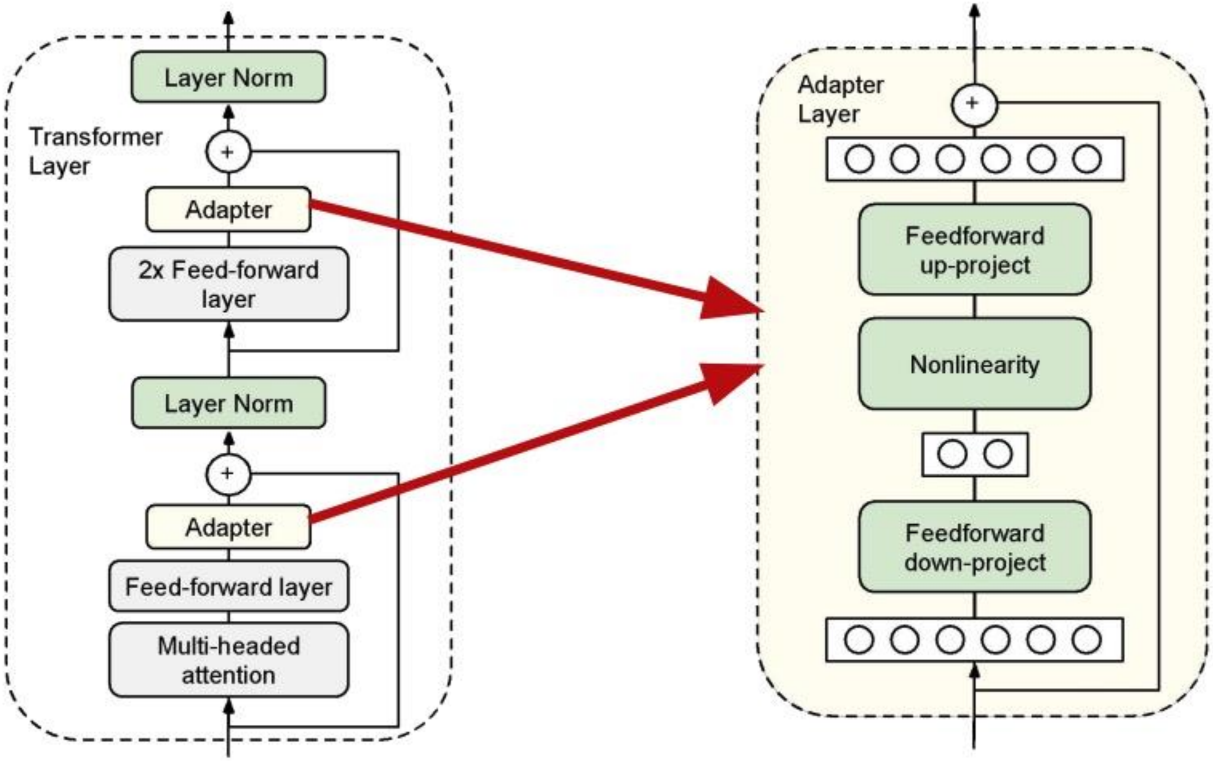
- Addition: What if we introduce additional (a small amount) trainable parameters to the neural network and just train those?



- 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.

Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." International Conference on Machine Learning. PMLR, 2019.

Parameter Efficient Fine Tuning - Adaptor

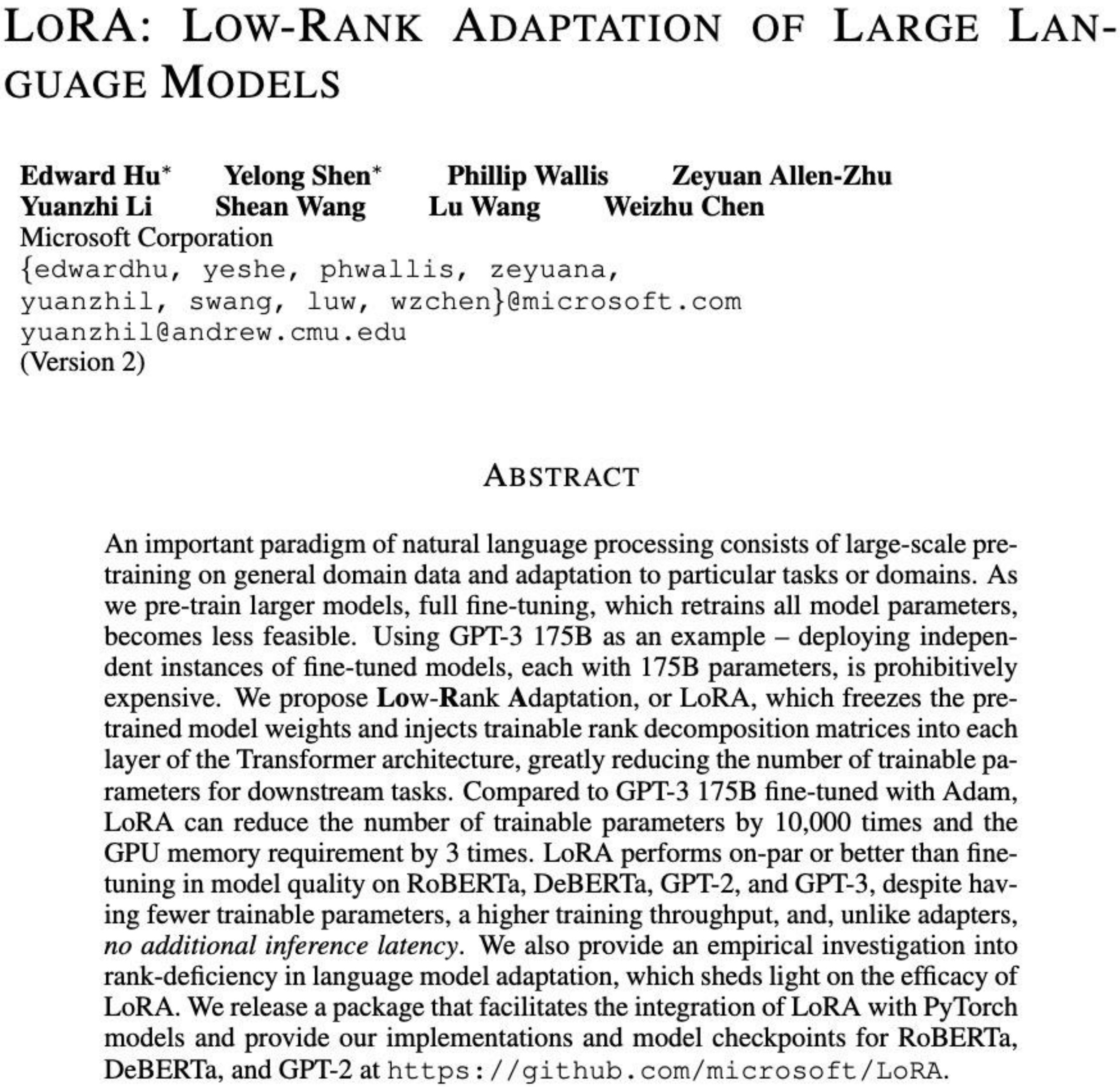


The adapter consists of a bottleneck which contains few parameters relative to the attention and feedforward layers in the original model.

Only the adaptor layers (i.e., a small set of parameters) are trained on the downstream data

Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." International Conference on Machine Learning. PMLR, 2019.

Parameter Efficient Fine Tuning - LoRA



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`

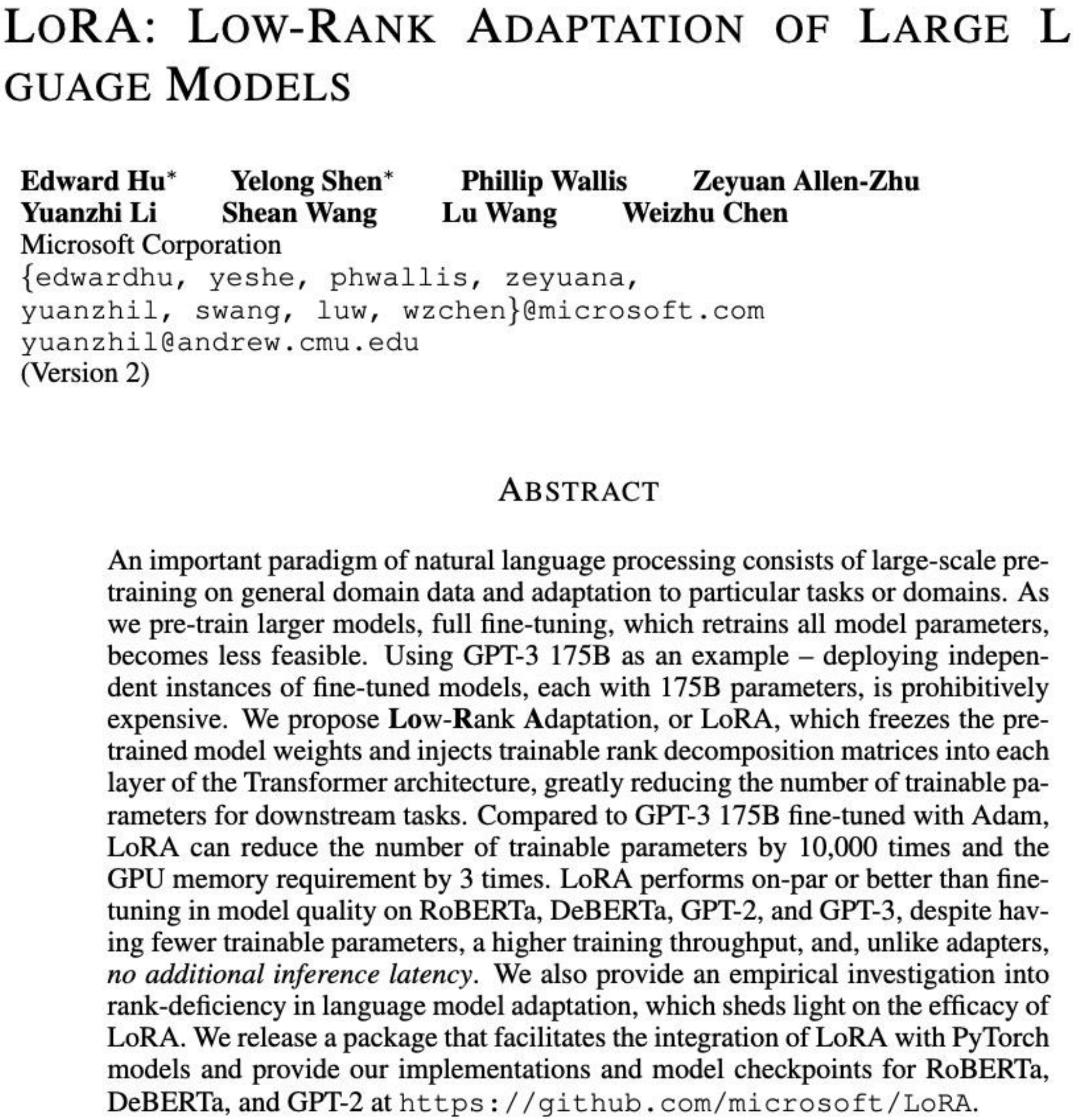
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pretraining 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 **Low-Rank Adaptation**, or LoRA, which freezes the pretrained 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`.

Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

Parameter Efficient Fine Tuning - LoRA



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)

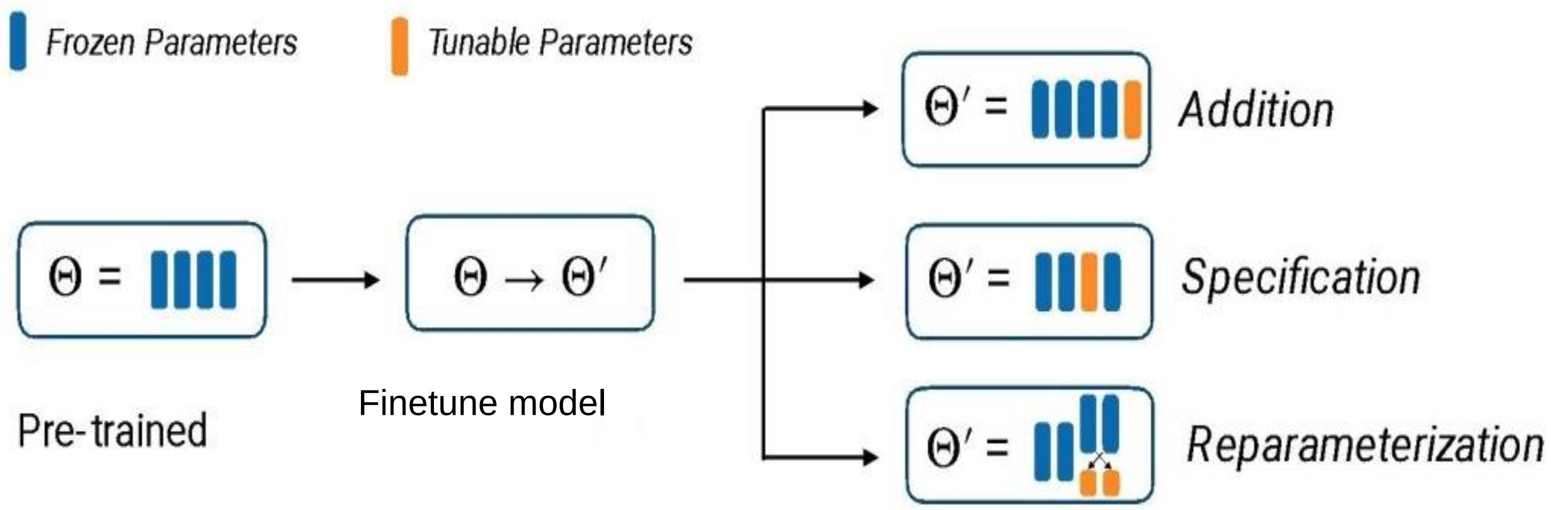
Abstract

An important paradigm of natural language processing consists of large-scale pretraining 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 **Low-Rank Adaptation**, or LoRA, which freezes the pretrained 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 finetuning 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`.

Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

Recall: Parameter Efficient Fine Tuning

- 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?

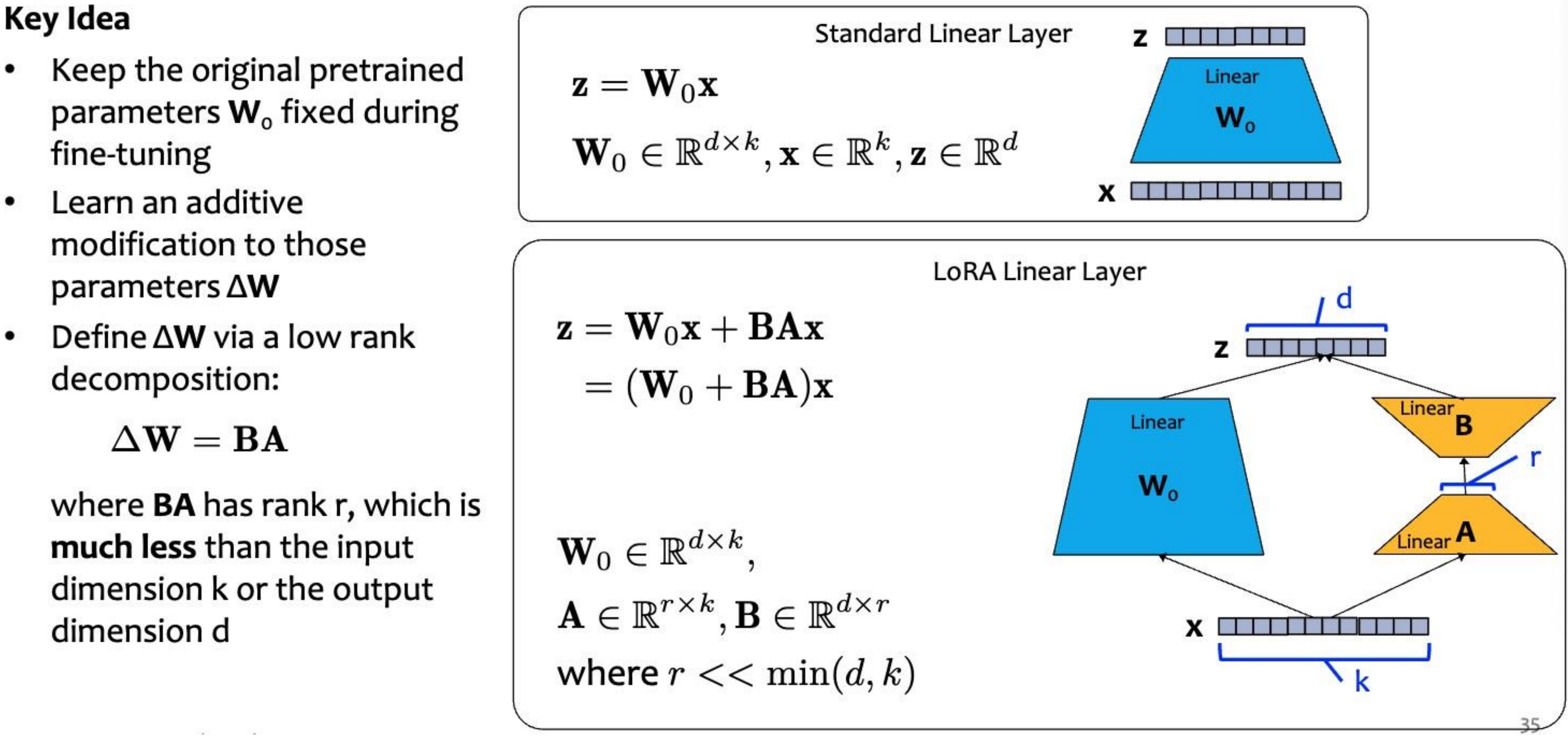


Slide Credit: Daphne Ippolito, Chenyan Xiong

Parameter Efficient Fine Tuning - LoRA

- Previous study shows that
  - Pre-trained LLMs have a “low intrinsic dimension”
  - LLMs can still learn efficiently despite a low-dim reparametrization

Parameter Efficient Fine Tuning - LoRA



Key Idea

- Keep the original pretrained parameters $\mathbf{W}_0$ fixed during fine-tuning
- Learn an additive modification to those parameters $\Delta\mathbf{W}$
- Define $\Delta\mathbf{W}$ via a low rank decomposition:

$$\Delta\mathbf{W} = \mathbf{BA}$$

where **BA** has rank r, which is **much less** than the input dimension k or the output dimension d

Standard Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k}, \mathbf{x} \in \mathbb{R}^k, \mathbf{z} \in \mathbb{R}^d$$

LoRA Linear Layer

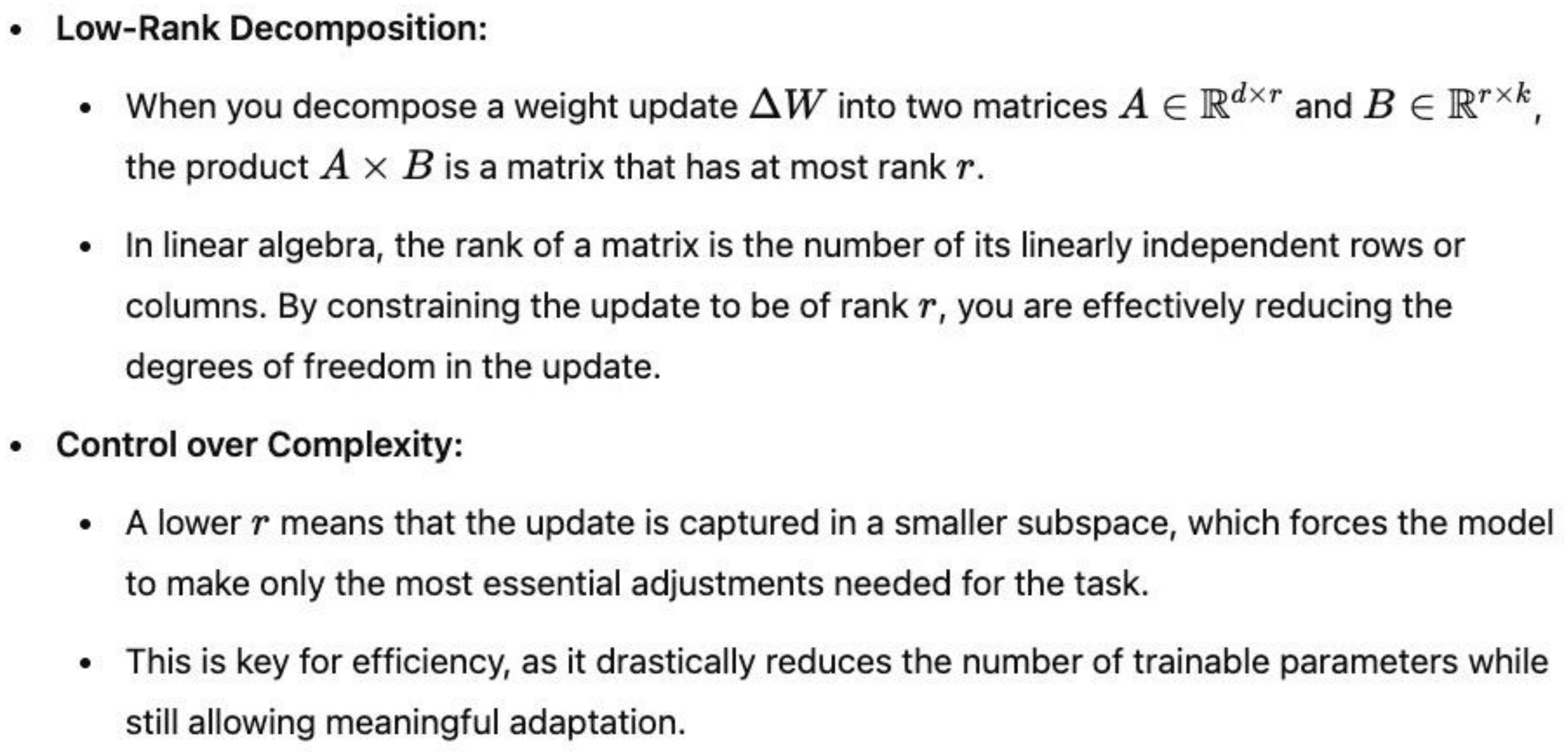
$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k},$$
$$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$$

where $r << \min(d, k)$

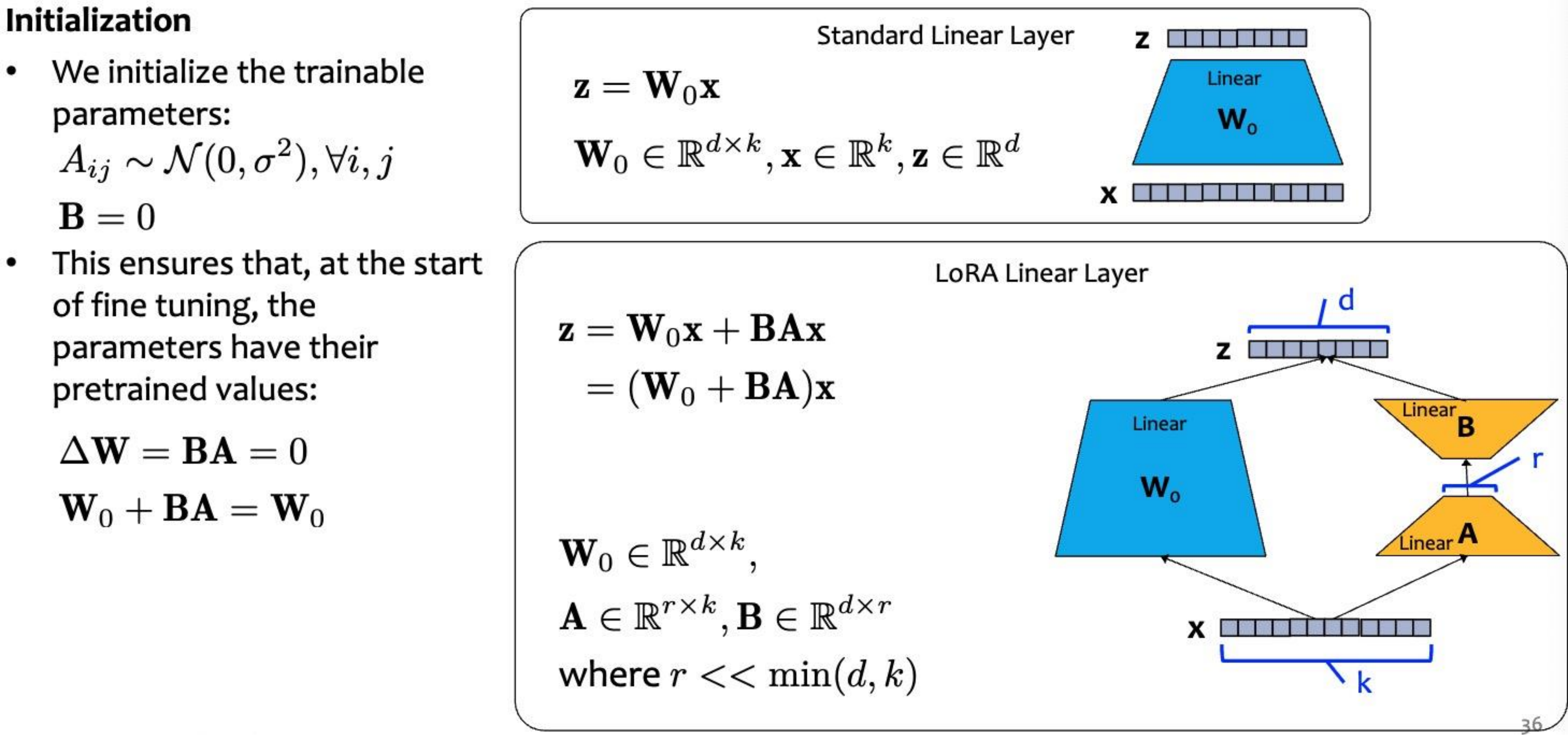
Slide credit: Matt Gormley

Parameter Efficient Fine Tuning - LoRA



- **Low-Rank Decomposition:**
  - When you decompose a weight update ΔW into two matrices $A \in \mathbb{R}^{d \times r}$ and $B \in \mathbb{R}^{r \times k}$, the product $A \times B$ is a matrix that has at most rank r.
  - In linear algebra, the rank of a matrix is the number of its linearly independent rows or columns. By constraining the update to be of rank r, you are effectively reducing the degrees of freedom in the update.
- **Control over Complexity:**
  - A lower r means that the update is captured in a smaller subspace, which forces the model to make only the most essential adjustments needed for the task.
  - This is key for efficiency, as it drastically reduces the number of trainable parameters while still allowing meaningful adaptation.

Parameter Efficient Fine Tuning - LoRA



Initialization

- We initialize the trainable parameters:

  $A_{ij} \sim \mathcal{N}(0, \sigma^2), \forall i, j$

  $\mathbf{B} = 0$

- This ensures that, at the start of fine tuning, the parameters have their pretrained values:

  $\Delta\mathbf{W} = \mathbf{BA} = 0$

  $\mathbf{W}_0 + \mathbf{BA} = \mathbf{W}_0$

Standard Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k}, \mathbf{x} \in \mathbb{R}^k, \mathbf{z} \in \mathbb{R}^d$$

LoRA Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k},$$

$$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$$

where $r << \min(d, k)$

Slide credit: Matt Gormley

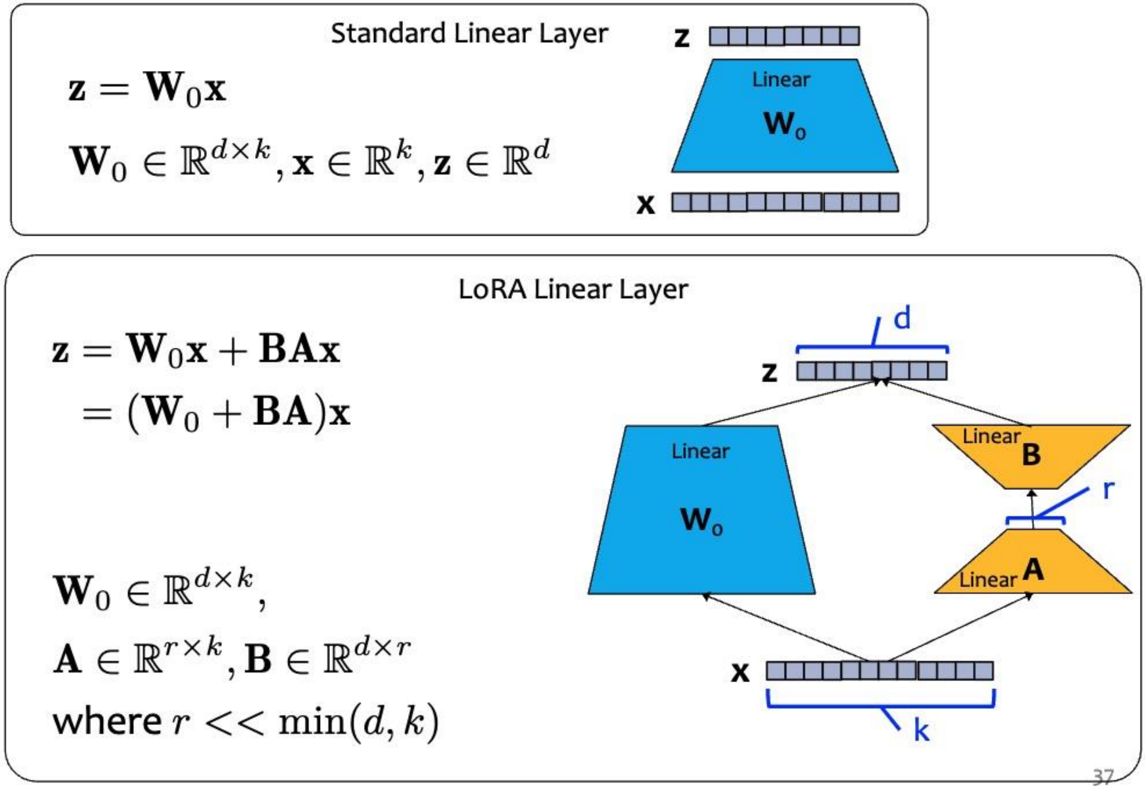
Parameter Efficient Fine Tuning - LoRA

- To get a Standard Linear Layer with parameters **W** that includes our LoRA fine tuning:

$$\mathbf{W} \leftarrow \mathbf{W}_0 + \mathbf{BA}$$

- To remove the LoRA fine tuning from that Standard Linear Layer:

$$\mathbf{W} \leftarrow \mathbf{W} - \mathbf{BA} = \mathbf{W}_0$$



Standard Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k}, \mathbf{x} \in \mathbb{R}^{k}, \mathbf{z} \in \mathbb{R}^{d}$$

LoRA Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

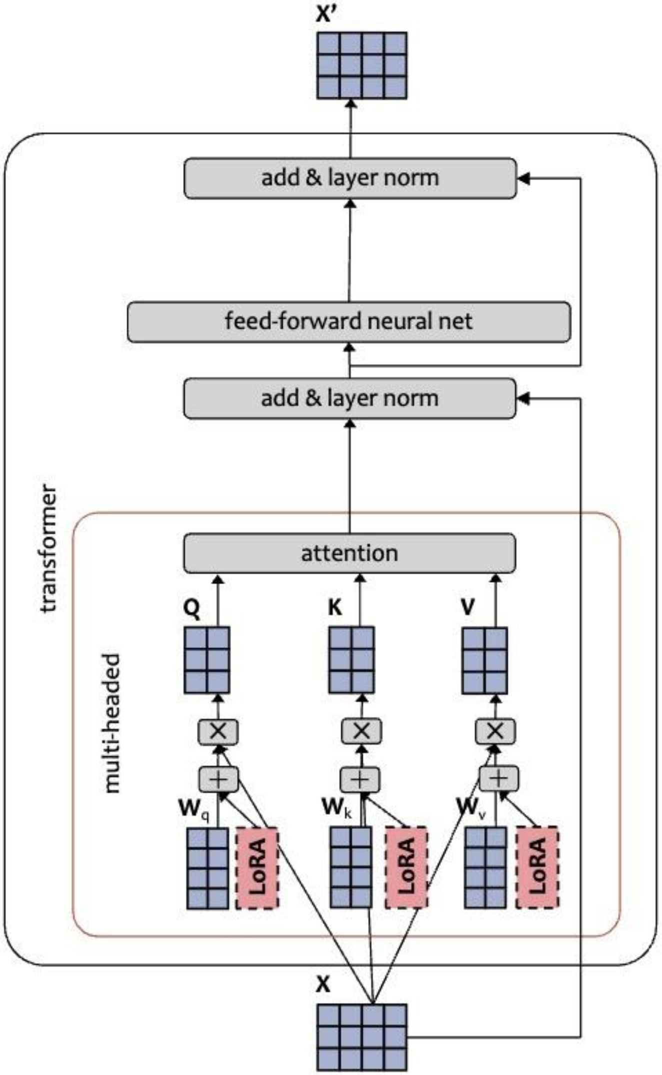
$\mathbf{W}_0 \in \mathbb{R}^{d\times k}$,

$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$

where $r << \min(d, k)$

Slide credit: Matt Gormley

LoRA for Transformer



- LoRA linear layers could replace *every* linear layer in the Transformer layer
- But the original paper only applies LoRA to the attention weights

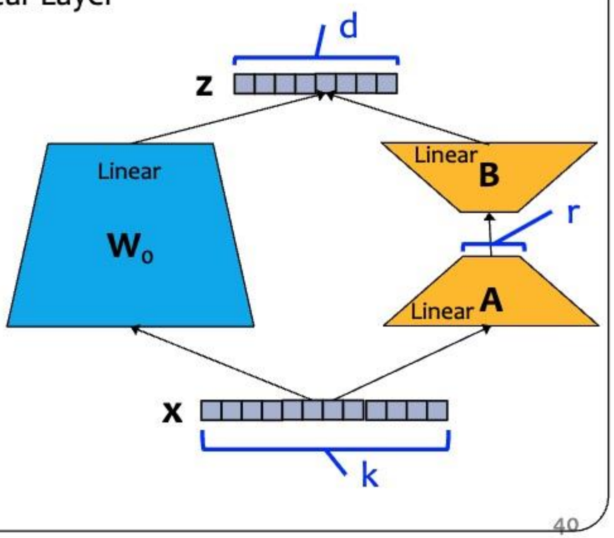
LoRA Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}$$
$$= (\mathbf{W}_0 + \mathbf{B}\mathbf{A})\mathbf{x}$$

$\mathbf{W}_0 \in \mathbb{R}^{d \times k}$,

$\mathbf{A} \in \mathbb{R}^{r \times k}, \mathbf{B} \in \mathbb{R}^{d \times r}$

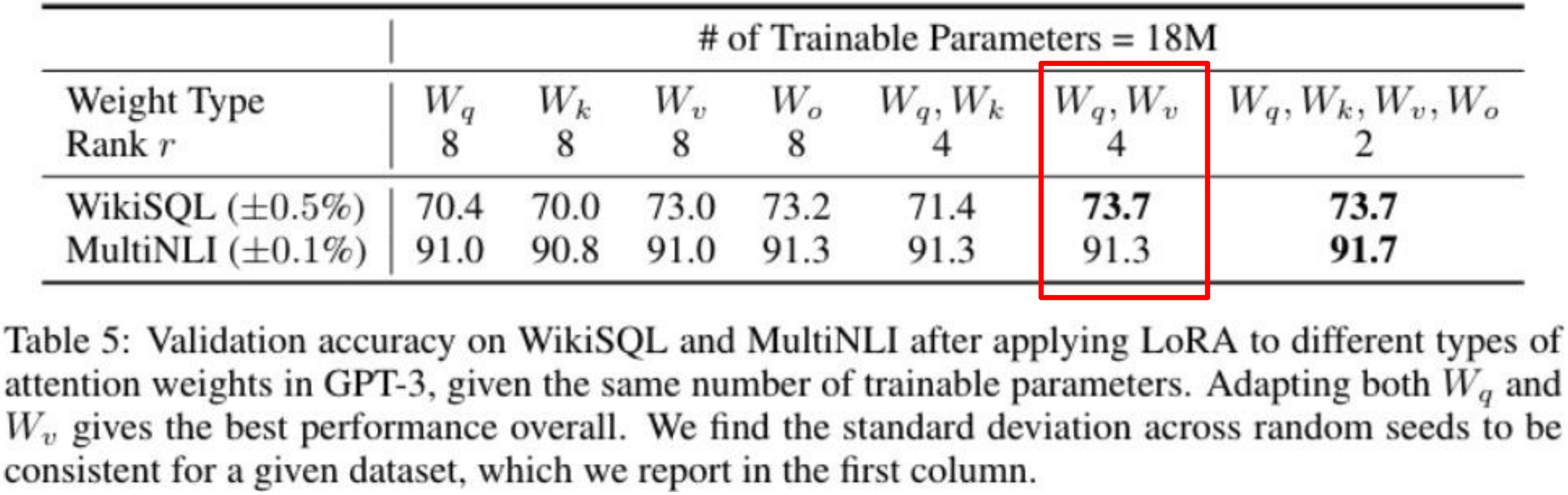
where $r << \min(d, k)$



Slide credit: Matt Gormley

LoRA for Transformer

- LoRA linear layers could replace *every* linear layer in the Transformer layer
- But the original paper only applies LoRA to the attention weights
- Empirically, for GPT-3, they also find that it is most efficient to include LoRA **only on the** Q **and** V linear layers:



	# of Trainable Parameters = 18M						
Weight Type Rank r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q, W_v 4	W_q, W_k, W_v, W_o 2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

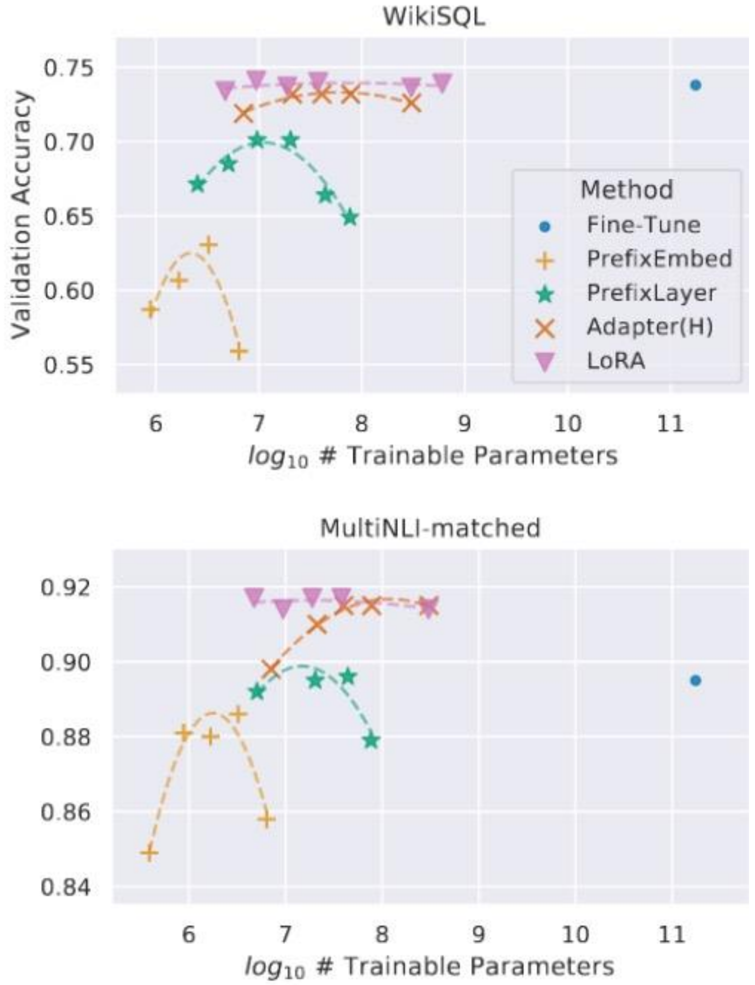
Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

Slide credit: Matt Gormley

LoRA Results

Takeaways

- Applied to GPT-3, LoRA achieves performance almost as good as full fine-tuning, but with far fewer parameters
- On some tasks it even outperforms full fine-tuning
- For some datasets a rank of r=1 is sufficient



Slide credit: Matt Gormley

LoRA Variants

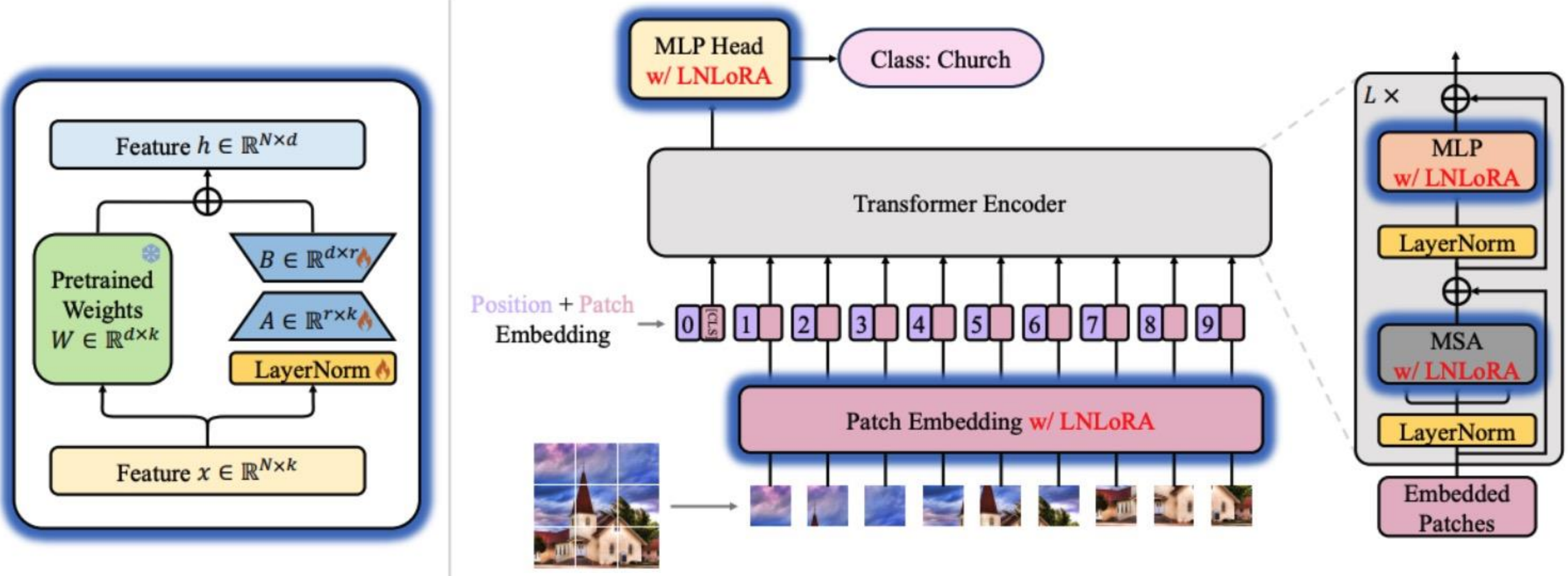
- LongLoRA [Chen et al., 2024]
  - Sparse Local attention to support longer context length during finetuning

- LoRA+ [Hayou et al., 2024]
  - different learning rates for the LoRA adapter matrices A and B improves finetuning speed

- DyLoRA [Valipou et al., 2023]
  - selects rank without requiring multiple runs of training

PEFT for Vision Transformer

- Since Vision Transfomer is just another transformer model, we can apply LoRA directly to it
- (LNLoRA is just a variant that includes LayerNorm in the LoRALinear module.)

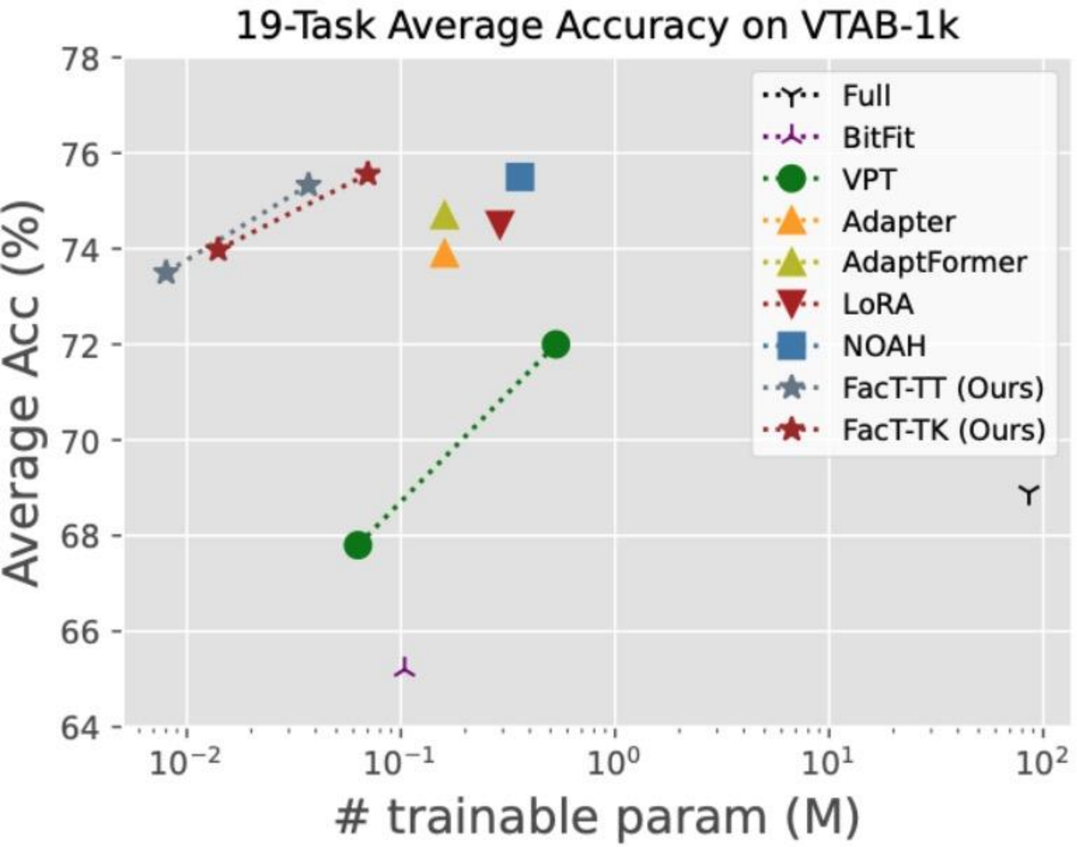


Feature $h \in \mathbb{R}^{N\times d}$
Pretrained Weights $W \in \mathbb{R}^{d\times k}$
$B \in \mathbb{R}^{d\times r}$
$A \in \mathbb{R}^{r\times k}$
LayerNorm
Feature $x \in \mathbb{R}^{N\times k}$
MLP Head w/ LNLoRA
Class: Church
Transformer Encoder
Position + Patch Embedding
0
[CLS]
1
2
3
4
5
6
7
8
9
Patch Embedding w/ LNLoRA
$L\times$
MLP w/ LNLoRA
LayerNorm
MSA w/ LNLoRA
LayerNorm
Embedded Patches

Yuan, Zheng, Jie Zhang, and Shiguang Shan. "Fulllora-at: Efficiently boosting the robustness of pretrained vision transformers." *arXiv preprint arXiv:2401.01752* (2024).

Slide credit: Matt Gormley

PEFT for Vision Transformer



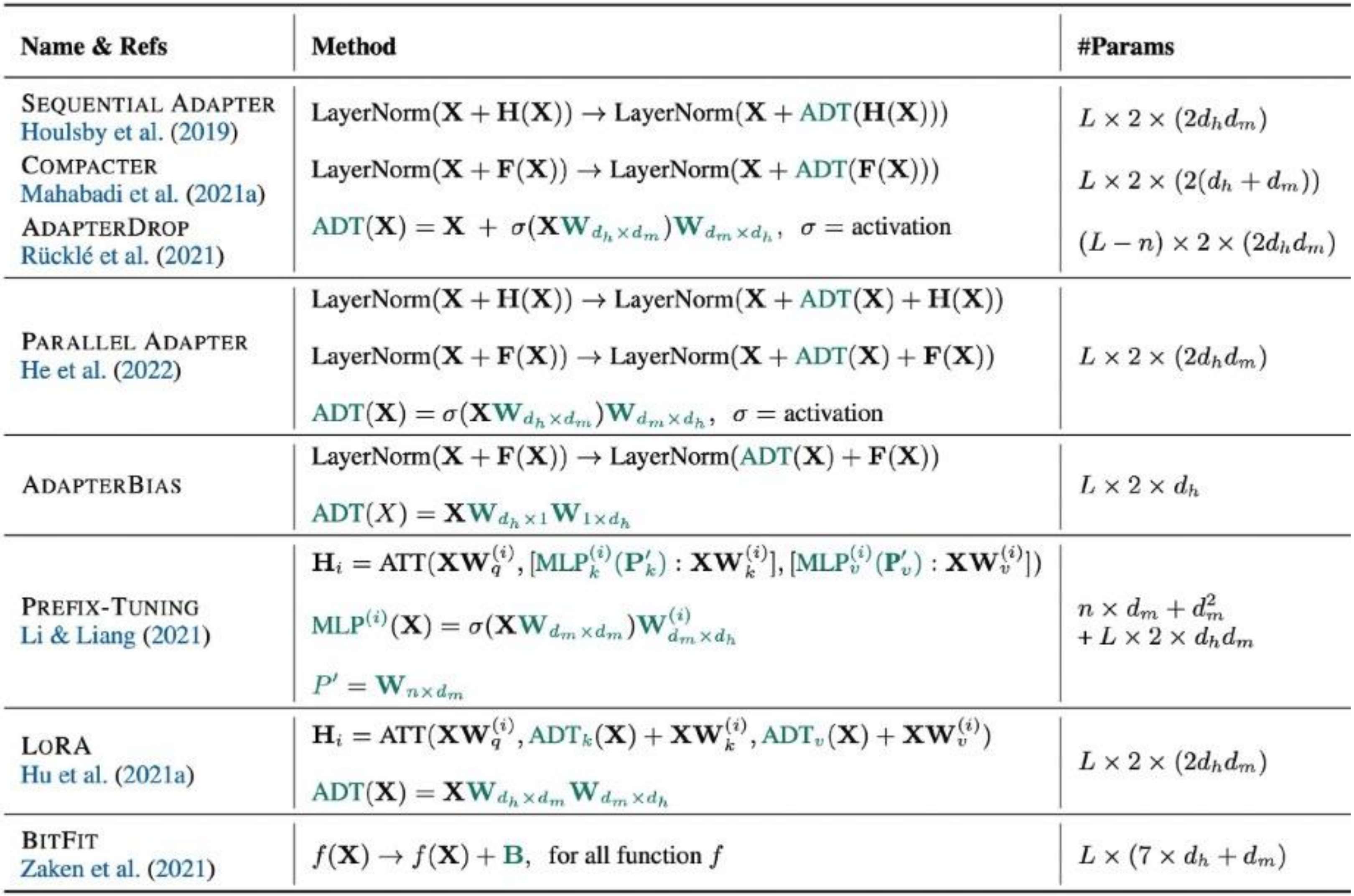
- For various computer vision tasks, parameter efficient transfer-learning (PETL) is sometimes **better** than full fine-tuning!
- VTAB-1k is a collection of 19 different vision tasks; here we're seeing average performance across tasks
- (FacT is another low-rank method capable of dramatically reducing the number of parameters tuned.)

Jie, Shibo, and Zhi-Hong Deng. "Fact: Factor-tuning for lightweight adaptation on vision transformer." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 37. No. 1. 2023.

Slide credit: Matt Gormley

Summary

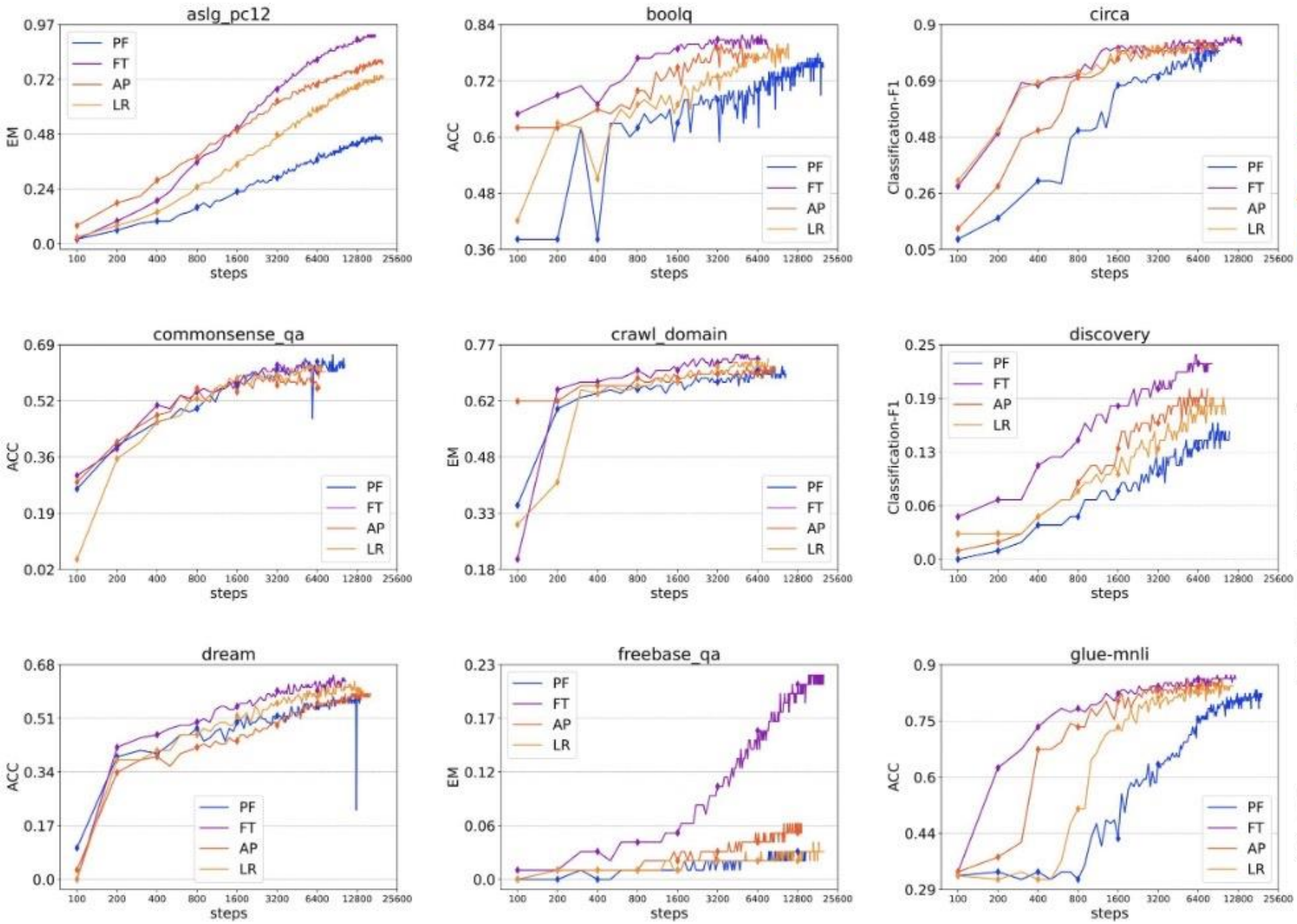
green color : tunable parameters and modules



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

Ding, Ning, et al. "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models." arXiv preprint arXiv:2203.06904 (2022).

Results



PF: prefix tuning

FT: full fine-tuning

AP: adapter

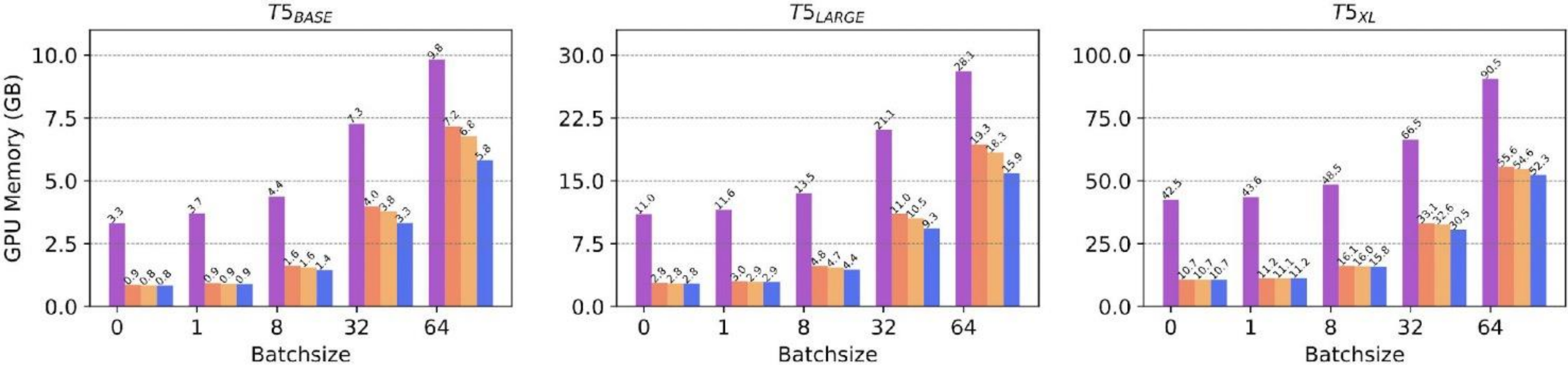
LR: LoRA

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.

Ding, Ning, et al. "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models." arXiv preprint arXiv:2203.06904 (2022).

Memory Usage



FT: full fine-tuning

AP: adapter

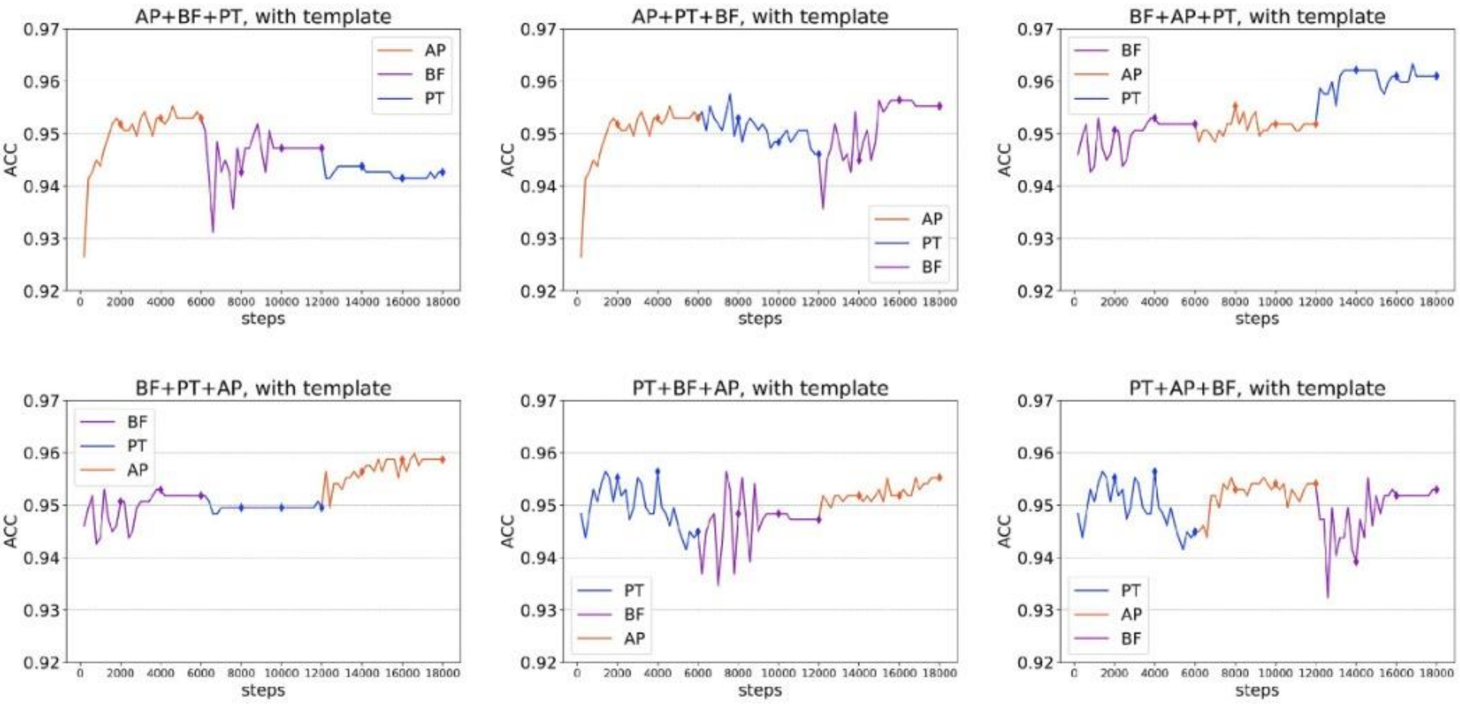
LR: LoRA

BF: BitFit

Prompt tuning and prefix tuning not included because they use the same amount of memory as full fine-tuning

Ding, Ning, et al. "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models." arXiv preprint arXiv:2203.06904 (2022).

Combine Different Methods



FT: BitFit

AP: adapter

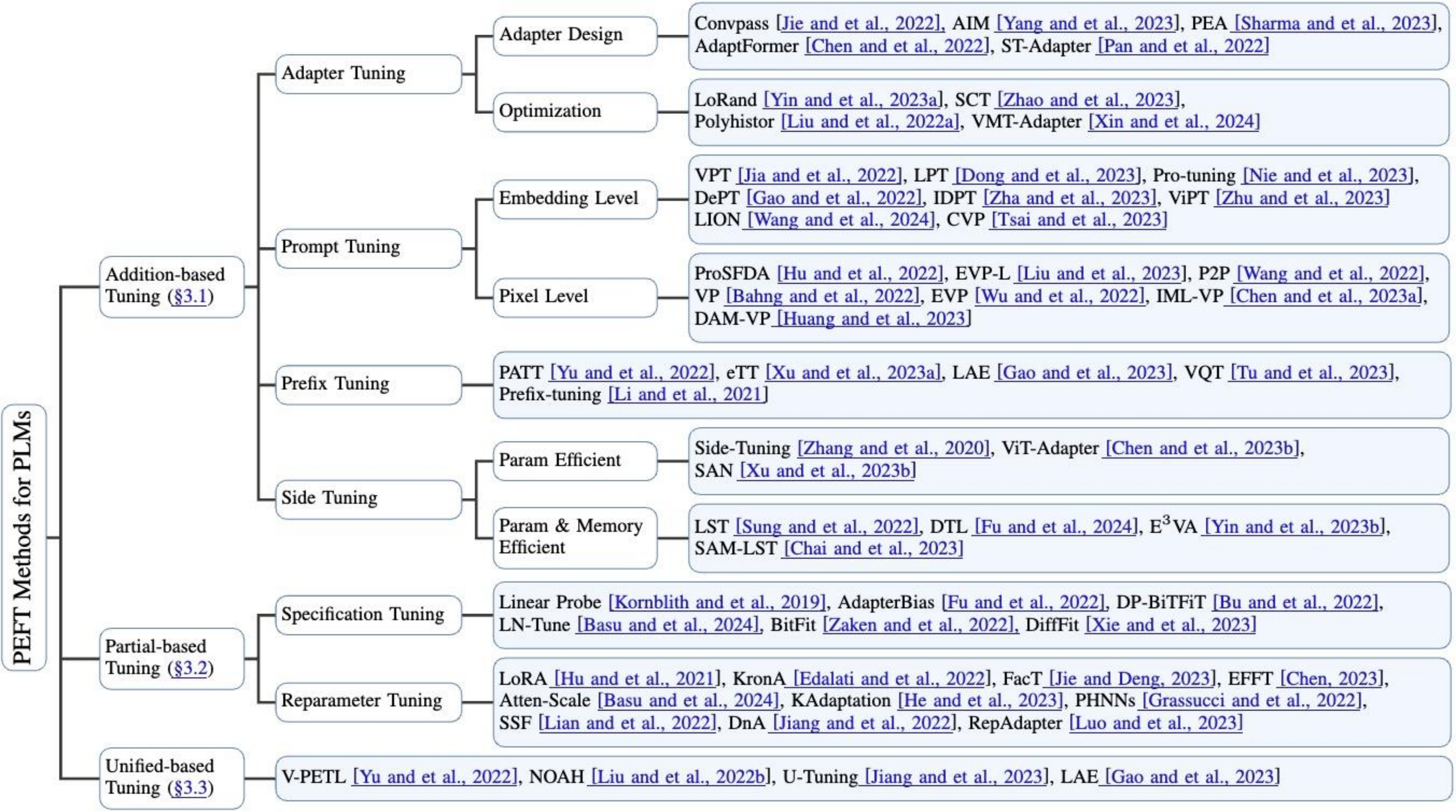
PT: prompt tuning

Results on SST-2 sentiment classification

Ding, Ning, et al. "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models." arXiv preprint arXiv:2203.06904 (2022).

Surveys

- Han, Zeyu, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. "Parameter-efficient fine-tuning for large models: A comprehensive survey." *arXiv preprint arXiv:2403.14608* (2024).
- Xu, Lingling, et al. "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment." *arXiv preprint arXiv:2312.12148* (2023).
- Xin, Yi, et al. "Parameter-efficient fine-tuning for pre-trained vision models: A survey." *arXiv preprint arXiv:2402.02242* (2024).
- Ding, Ning, et al. "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models." *arXiv preprint arXiv:2203.06904* (2022).



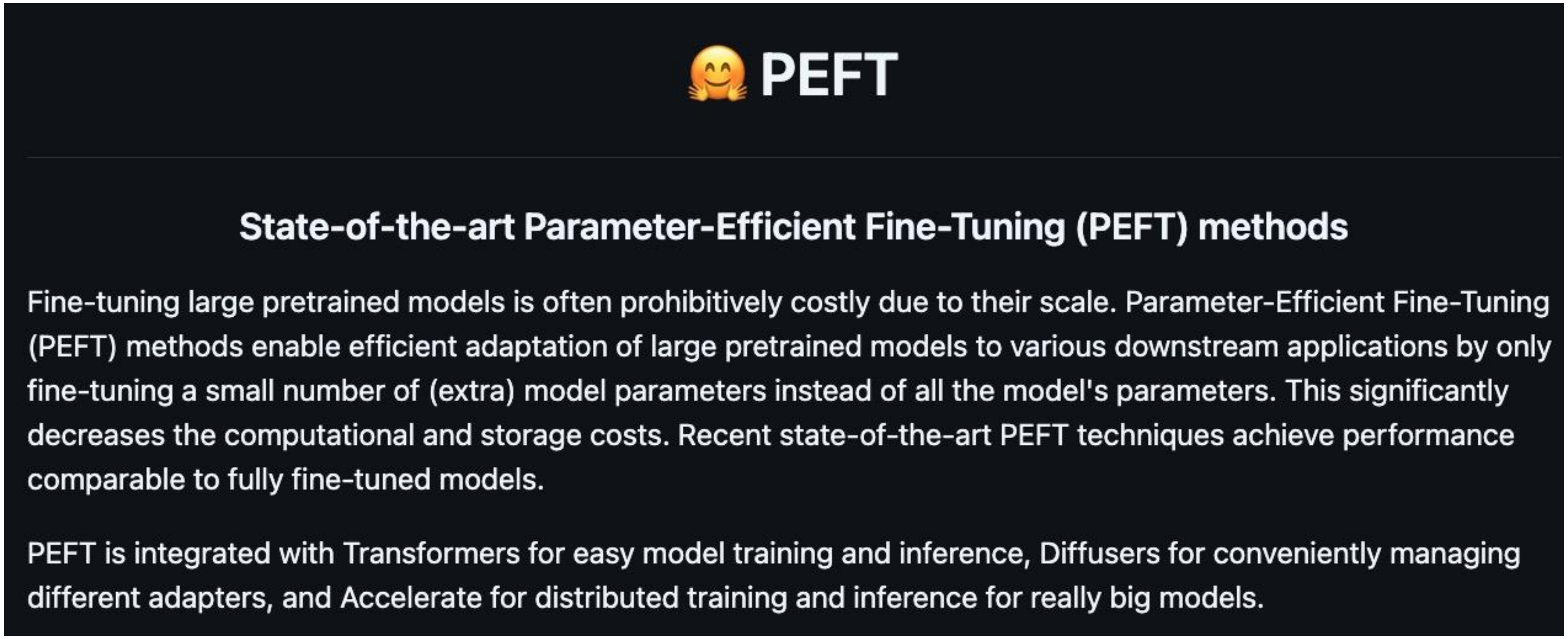
- PEFT Methods for PLMs
  - Addition-based Tuning (§3.1)
    - Adapter Tuning
      - Adapter Design
        - Convpass [Jie and et al., 2022], AIM [Yang and et al., 2023], PEA [Sharma and et al., 2023], AdaptFormer [Chen and et al., 2022], ST-Adapter [Pan and et al., 2022]
      - Optimization
        - LoRand [Yin and et al., 2023a], SCT [Zhao and et al., 2023], Polyhistor [Liu and et al., 2022a], VMT-Adapter [Xin and et al., 2024]
    - Prompt Tuning
      - Embedding Level
        - VPT [Jia and et al., 2022], LPT [Dong and et al., 2023], Pro-tuning [Nie and et al., 2023], DePT [Gao and et al., 2022], IDPT [Zha and et al., 2023], ViPT [Zhu and et al., 2023] LION [Wang and et al., 2024], CVP [Tsai and et al., 2023]
      - Pixel Level
        - ProSFDA [Hu and et al., 2022], EVP-L [Liu and et al., 2023], P2P [Wang and et al., 2022], VP [Bahng and et al., 2022], EVP [Wu and et al., 2022], IML-VP [Chen and et al., 2023a], DAM-VP [Huang and et al., 2023]
    - Prefix Tuning
      - PATT [Yu and et al., 2022], eTT [Xu and et al., 2023a], LAE [Gao and et al., 2023], VQT [Tu and et al., 2023], Prefix-tuning [Li and et al., 2021]
    - Side Tuning
      - Param Efficient
        - Side-Tuning [Zhang and et al., 2020], ViT-Adapter [Chen and et al., 2023b], SAN [Xu and et al., 2023b]
      - Param & Memory Efficient
        - LST [Sung and et al., 2022], DTL [Fu and et al., 2024], E^3VA [Yin and et al., 2023b], SAM-LST [Chai and et al., 2023]
  - Partial-based Tuning (§3.2)
    - Specification Tuning
      - Linear Probe [Kornblith and et al., 2019], AdapterBias [Fu and et al., 2022], DP-BiTFiT [Bu and et al., 2022], LN-Tune [Basu and et al., 2024], BitFit [Zaken and et al., 2022], DiffFit [Xie and et al., 2023]
    - Reparameter Tuning
      - LoRA [Hu and et al., 2021], KronA [Edalati and et al., 2022], FacT [Jie and Deng, 2023], EFFT [Chen, 2023], Atten-Scale [Basu and et al., 2024], KAdaptation [He and et al., 2023], PHNNs [Grassucci and et al., 2022], SSF [Lian and et al., 2022], DnA [Jiang and et al., 2022], RepAdapter [Luo and et al., 2023]
  - Unified-based Tuning (§3.3)
    - V-PETL [Yu and et al., 2022], NOAH [Liu and et al., 2022b], U-Tuning [Jiang and et al., 2023], LAE [Gao and et al., 2023]

Figure 1: Taxonomy of Parameter-Efficient Fine-Tuning Methods for Pre-trained Vision Models.

Xin, Yi, et al. "Parameter-efficient fine-tuning for pre-trained vision models: A survey." *arXiv preprint arXiv:2402.02242* (2024).

Tools

- https://github.com/huggingface/peft



🤗 PEFT

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

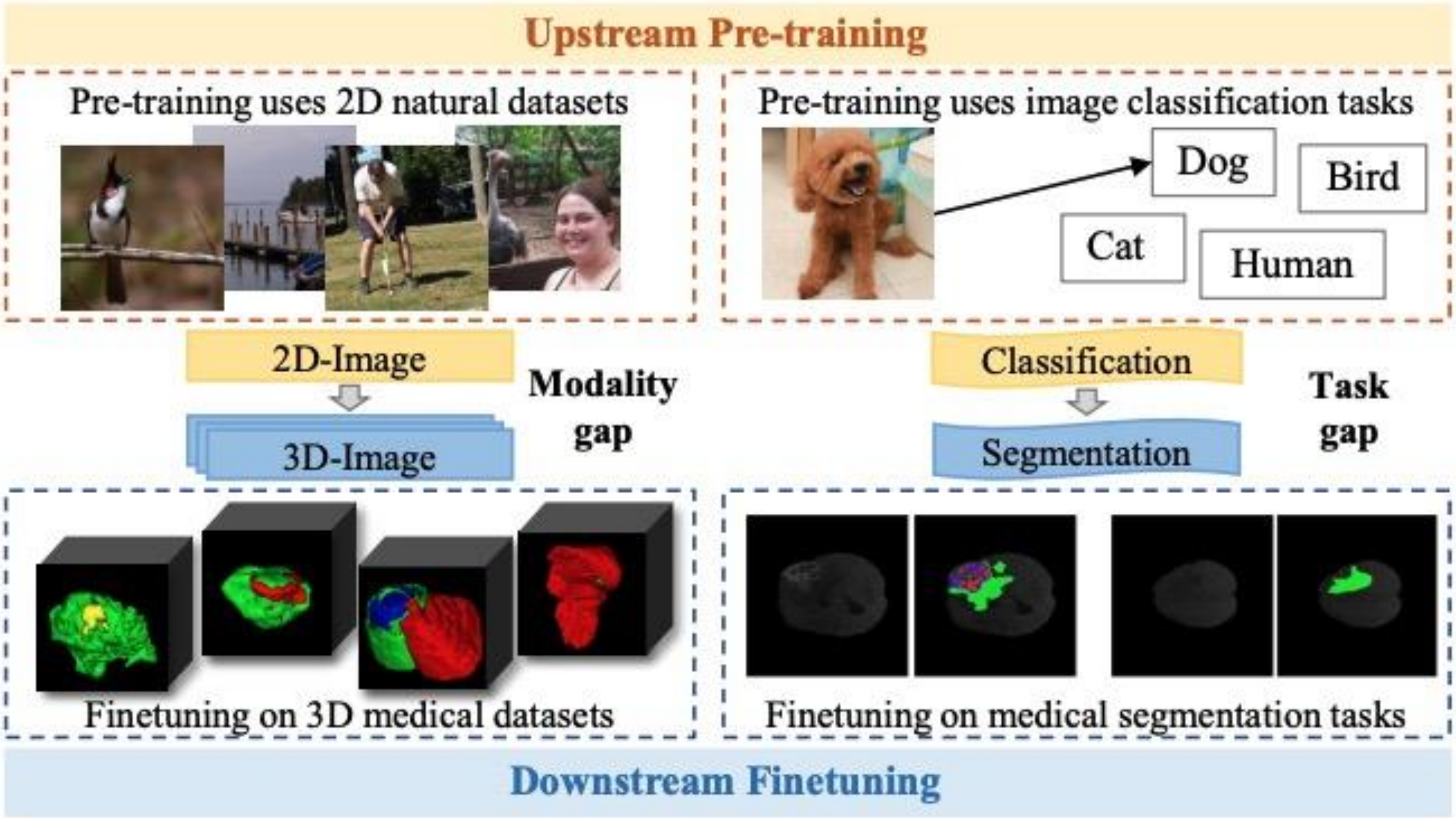
Fine-tuning large pretrained models is often prohibitively costly due to their scale. Parameter-Efficient Fine-Tuning (PEFT) methods enable efficient adaptation of large pretrained models to various downstream applications by only fine-tuning a small number of (extra) model parameters instead of all the model's parameters. This significantly decreases the computational and storage costs. Recent state-of-the-art PEFT techniques achieve performance comparable to fully fine-tuned models.

PEFT is integrated with Transformers for easy model training and inference, Diffusers for conveniently managing different adapters, and Accelerate for distributed training and inference for really big models.

Parameter Efficient Fine Tuning for Medical Volumetric Segmentation

Parameter Efficient Fine Tuning for Medical Volumetric Segmentation

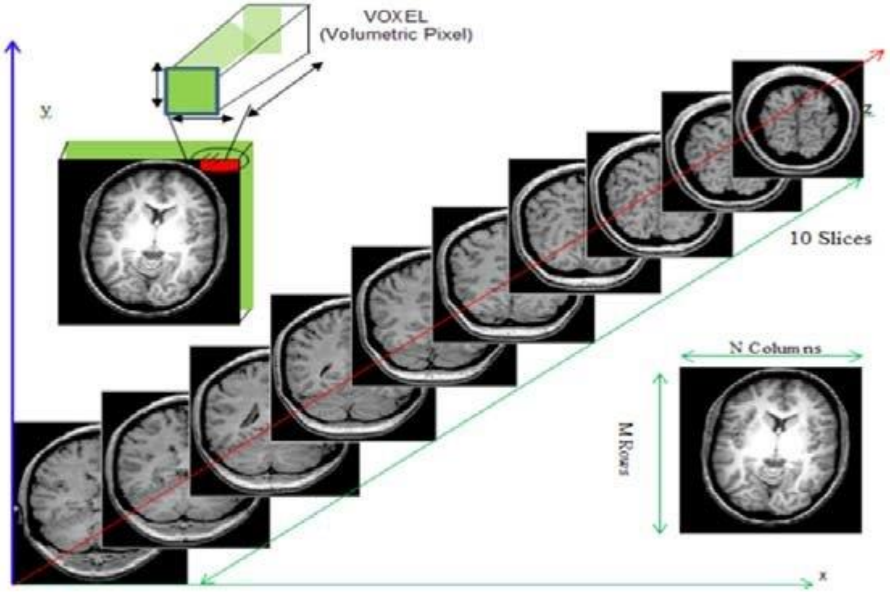
- Challenges



Upstream Pre-training
Pre-training uses 2D natural datasets
Pre-training uses image classification tasks
Dog
Bird
Cat
Human
2D-Image
3D-Image
Modality gap
Classification
Segmentation
Task gap
Finetuning on 3D medical datasets
Finetuning on medical segmentation tasks
Downstream Finetuning

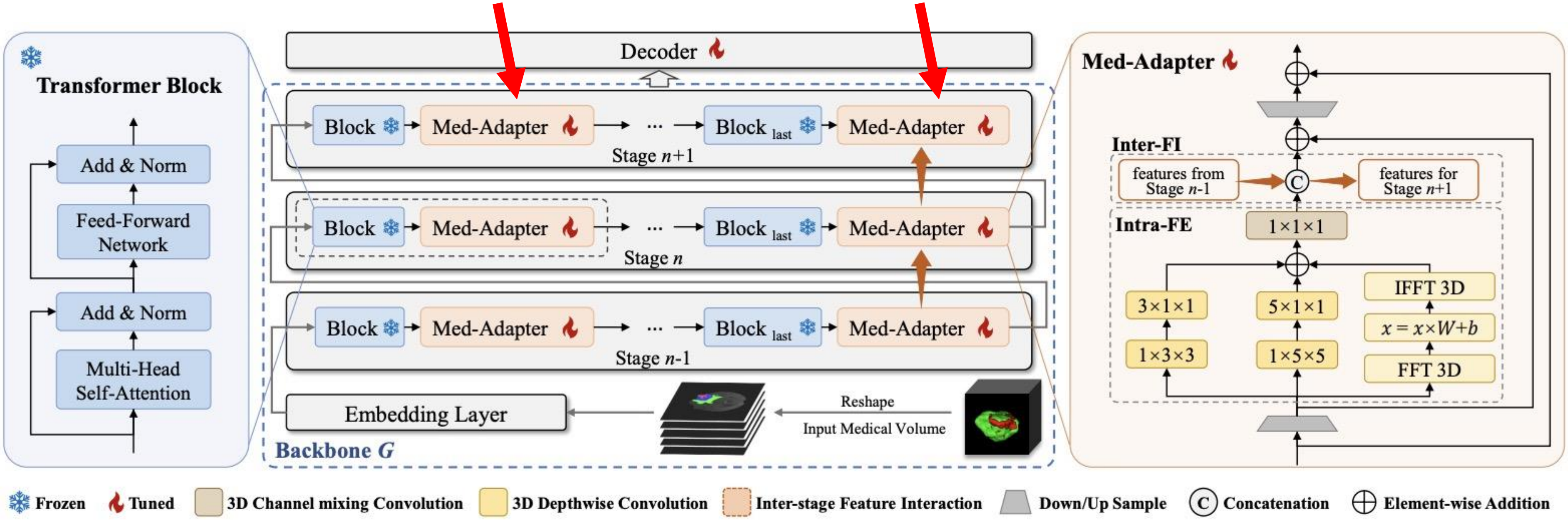
Parameter Efficient Fine Tuning for Medical Volumetric Segmentation

- Goal
  - Empower a 2D Transformer model pre-trained on natural images to gain the capability of **spatial and temporal feature modeling** for medical volumetric data in a **parameter-efficient manner**

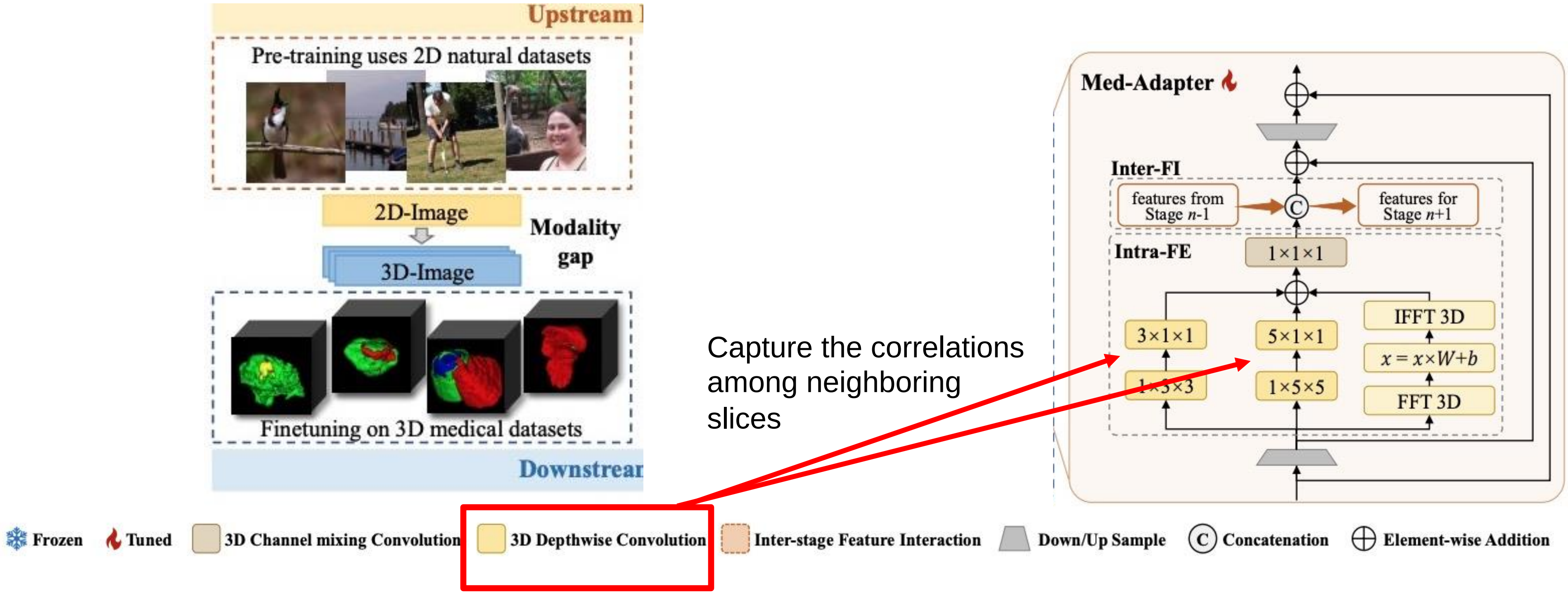


Proposed Method – Med-Tuning

- Follow the overall structure of the adapter module with tailored designs



Proposed Method – Med-Tuning

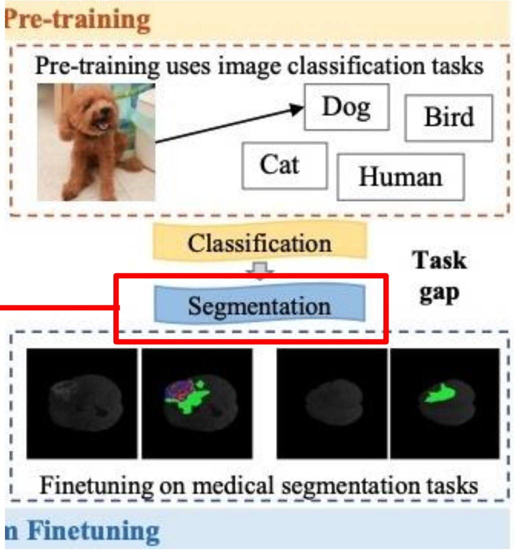


Capture the correlations among neighboring slices

Proposed Method – Med-Tuning

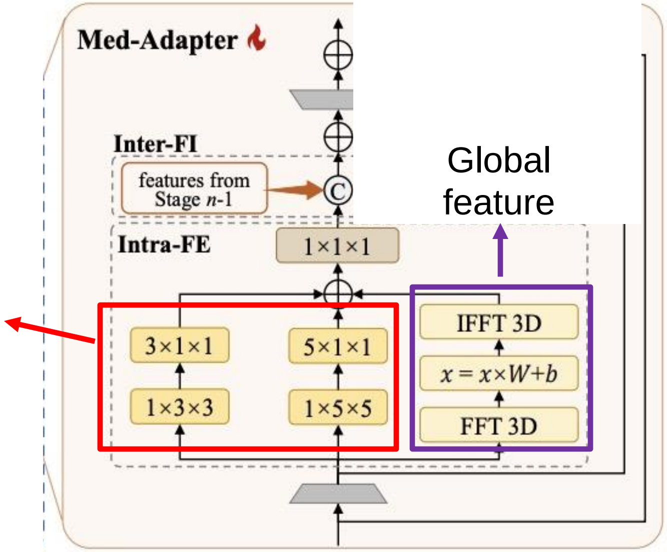
Dense prediction task (assign a label for each pixel)

Both local (fine-grained) feature representations and global semantic information are important


Pre-training
Pre-training uses image classification tasks
Dog
Bird
Cat
Human
Classification
Task gap
Segmentation
Finetuning on medical segmentation tasks
n Finetuning

Multi-scale feature representation

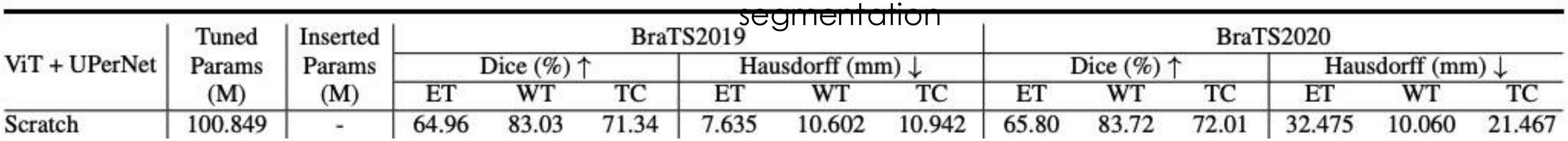
Global feature



❄ Frozen 🔥 Tuned ▭ 3D Channel mixing Convolution ▭ 3D Depthwise Convolution ▭ Inter-stage Feature Interaction ⏢ Down/Up Sample Ⓒ Concatenation ⊕ Element-wise Addition

Results - BraTS

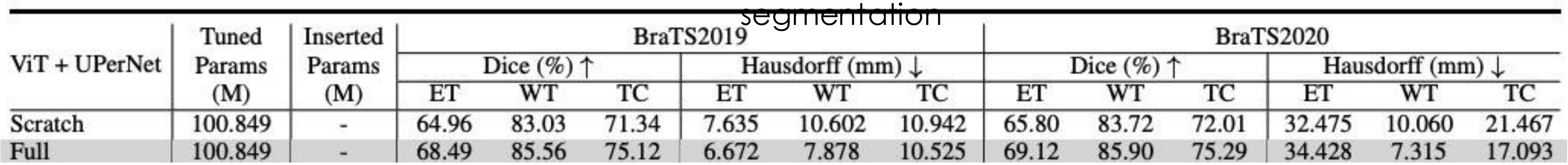
MRI brain tumor segmentation



ViT + UPerNet	Tuned Params (M)	Inserted Params (M)	BraTS2019						BraTS2020					
			Dice (%) ↑			Hausdorff (mm) ↓			Dice (%) ↑			Hausdorff (mm) ↓		
			ET	WT	TC	ET	WT	TC	ET	WT	TC	ET	WT	TC
Scratch	100.849	-	64.96	83.03	71.34	7.635	10.602	10.942	65.80	83.72	72.01	32.475	10.060	21.467

Results

MRI brain tumor segmentation

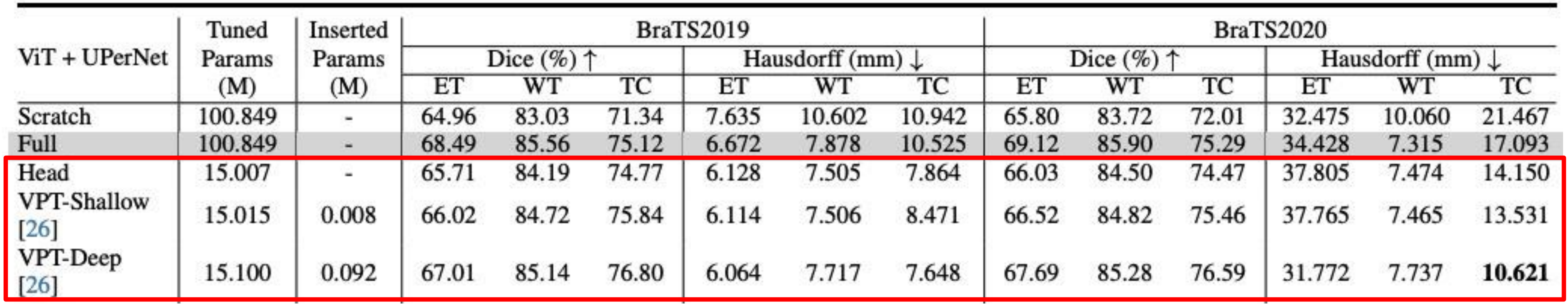


ViT + UPerNet	Tuned Params (M)	Inserted Params (M)	BraTS2019						BraTS2020					
			Dice (%) ↑			Hausdorff (mm) ↓			Dice (%) ↑			Hausdorff (mm) ↓		
			ET	WT	TC	ET	WT	TC	ET	WT	TC	ET	WT	TC
Scratch	100.849	-	64.96	83.03	71.34	7.635	10.602	10.942	65.80	83.72	72.01	32.475	10.060	21.467
Full	100.849	-	68.49	85.56	75.12	6.672	7.878	10.525	69.12	85.90	75.29	34.428	7.315	17.093

UCF CENTER FOR RESEARCH IN COMPUTER VISION

Results

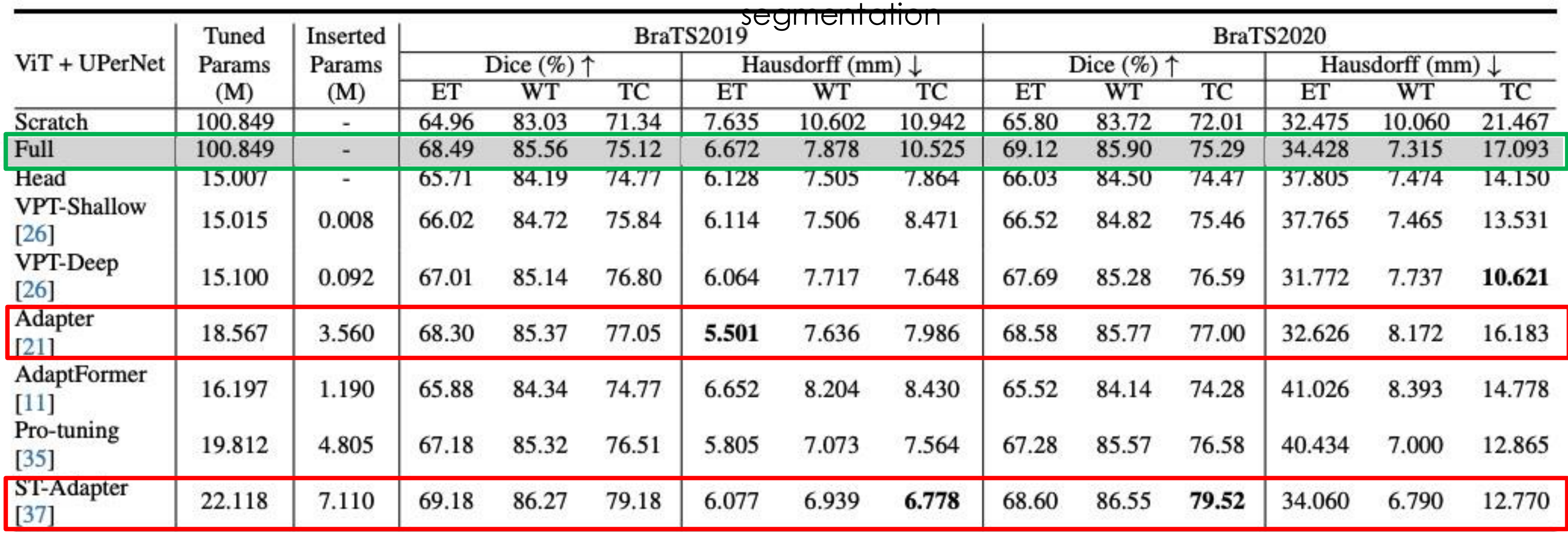
MRI brain tumor segmentation



ViT + UPerNet	Tuned Params (M)	Inserted Params (M)	BraTS2019						BraTS2020					
			Dice (%) ↑			Hausdorff (mm) ↓			Dice (%) ↑			Hausdorff (mm) ↓		
			ET	WT	TC	ET	WT	TC	ET	WT	TC	ET	WT	TC
Scratch	100.849	-	64.96	83.03	71.34	7.635	10.602	10.942	65.80	83.72	72.01	32.475	10.060	21.467
Full	100.849	-	68.49	85.56	75.12	6.672	7.878	10.525	69.12	85.90	75.29	34.428	7.315	17.093
Head	15.007	-	65.71	84.19	74.77	6.128	7.505	7.864	66.03	84.50	74.47	37.805	7.474	14.150
VPT-Shallow [26]	15.015	0.008	66.02	84.72	75.84	6.114	7.506	8.471	66.52	84.82	75.46	37.765	7.465	13.531
VPT-Deep [26]	15.100	0.092	67.01	85.14	76.80	6.064	7.717	7.648	67.69	85.28	76.59	31.772	7.737	**10.621**

Results

MRI brain tumor segmentation

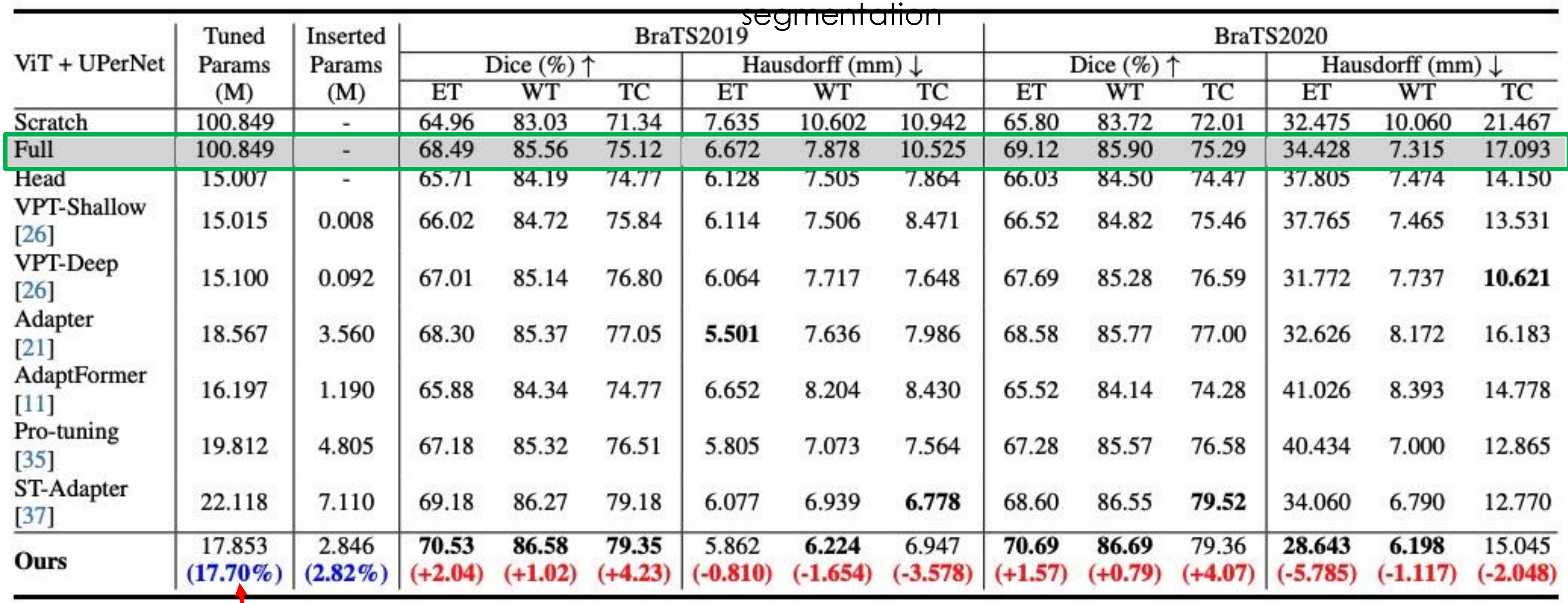


ViT + UPerNet	Tuned Params (M)	Inserted Params (M)	BraTS2019						BraTS2020					
			Dice (%) ↑			Hausdorff (mm) ↓			Dice (%) ↑			Hausdorff (mm) ↓		
			ET	WT	TC	ET	WT	TC	ET	WT	TC	ET	WT	TC
Scratch	100.849	-	64.96	83.03	71.34	7.635	10.602	10.942	65.80	83.72	72.01	32.475	10.060	21.467
Full	100.849	-	68.49	85.56	75.12	6.672	7.878	10.525	69.12	85.90	75.29	34.428	7.315	17.093
Head	15.007	-	65.71	84.19	74.77	6.128	7.505	7.864	66.03	84.50	74.47	37.805	7.474	14.150
VPT-Shallow [26]	15.015	0.008	66.02	84.72	75.84	6.114	7.506	8.471	66.52	84.82	75.46	37.765	7.465	13.531
VPT-Deep [26]	15.100	0.092	67.01	85.14	76.80	6.064	7.717	7.648	67.69	85.28	76.59	31.772	7.737	**10.621**
Adapter [21]	18.567	3.560	68.30	85.37	77.05	**5.501**	7.636	7.986	68.58	85.77	77.00	32.626	8.172	16.183
AdaptFormer [11]	16.197	1.190	65.88	84.34	74.77	6.652	8.204	8.430	65.52	84.14	74.28	41.026	8.393	14.778
Pro-tuning [35]	19.812	4.805	67.18	85.32	76.51	5.805	7.073	7.564	67.28	85.57	76.58	40.434	7.000	12.865
ST-Adapter [37]	22.118	7.110	69.18	86.27	79.18	6.077	6.939	**6.778**	68.60	86.55	**79.52**	34.060	6.790	12.770

UCF Center for Research in Computer Vision

Results

MRI brain tumor segmentation



ViT + UPerNet	Tuned Params (M)	Inserted Params (M)	BraTS2019						BraTS2020					
			Dice (%) ↑			Hausdorff (mm) ↓			Dice (%) ↑			Hausdorff (mm) ↓		
			ET	WT	TC	ET	WT	TC	ET	WT	TC	ET	WT	TC
Scratch	100.849	-	64.96	83.03	71.34	7.635	10.602	10.942	65.80	83.72	72.01	32.475	10.060	21.467
Full	100.849	-	68.49	85.56	75.12	6.672	7.878	10.525	69.12	85.90	75.29	34.428	7.315	17.093
Head	15.007	-	65.71	84.19	74.77	6.128	7.505	7.864	66.03	84.50	74.47	37.805	7.474	14.150
VPT-Shallow [26]	15.015	0.008	66.02	84.72	75.84	6.114	7.506	8.471	66.52	84.82	75.46	37.765	7.465	13.531
VPT-Deep [26]	15.100	0.092	67.01	85.14	76.80	6.064	7.717	7.648	67.69	85.28	76.59	31.772	7.737	**10.621**
Adapter [21]	18.567	3.560	68.30	85.37	77.05	**5.501**	7.636	7.986	68.58	85.77	77.00	32.626	8.172	16.183
AdaptFormer [11]	16.197	1.190	65.88	84.34	74.77	6.652	8.204	8.430	65.52	84.14	74.28	41.026	8.393	14.778
Pro-tuning [35]	19.812	4.805	67.18	85.32	76.51	5.805	7.073	7.564	67.28	85.57	76.58	40.434	7.000	12.865
ST-Adapter [37]	22.118	7.110	69.18	86.27	79.18	6.077	6.939	**6.778**	68.60	86.55	**79.52**	34.060	6.790	12.770
Ours	17.853 (17.70%)	2.846 (2.82%)	**70.53** (+2.04)	**86.58** (+1.02)	**79.35** (+4.23)	5.862 (-0.810)	**6.224** (-1.654)	6.947 (-3.578)	**70.69** (+1.57)	**86.69** (+0.79)	79.36 (+4.07)	**28.643** (-5.785)	**6.198** (-1.117)	15.045 (-2.048)

Results

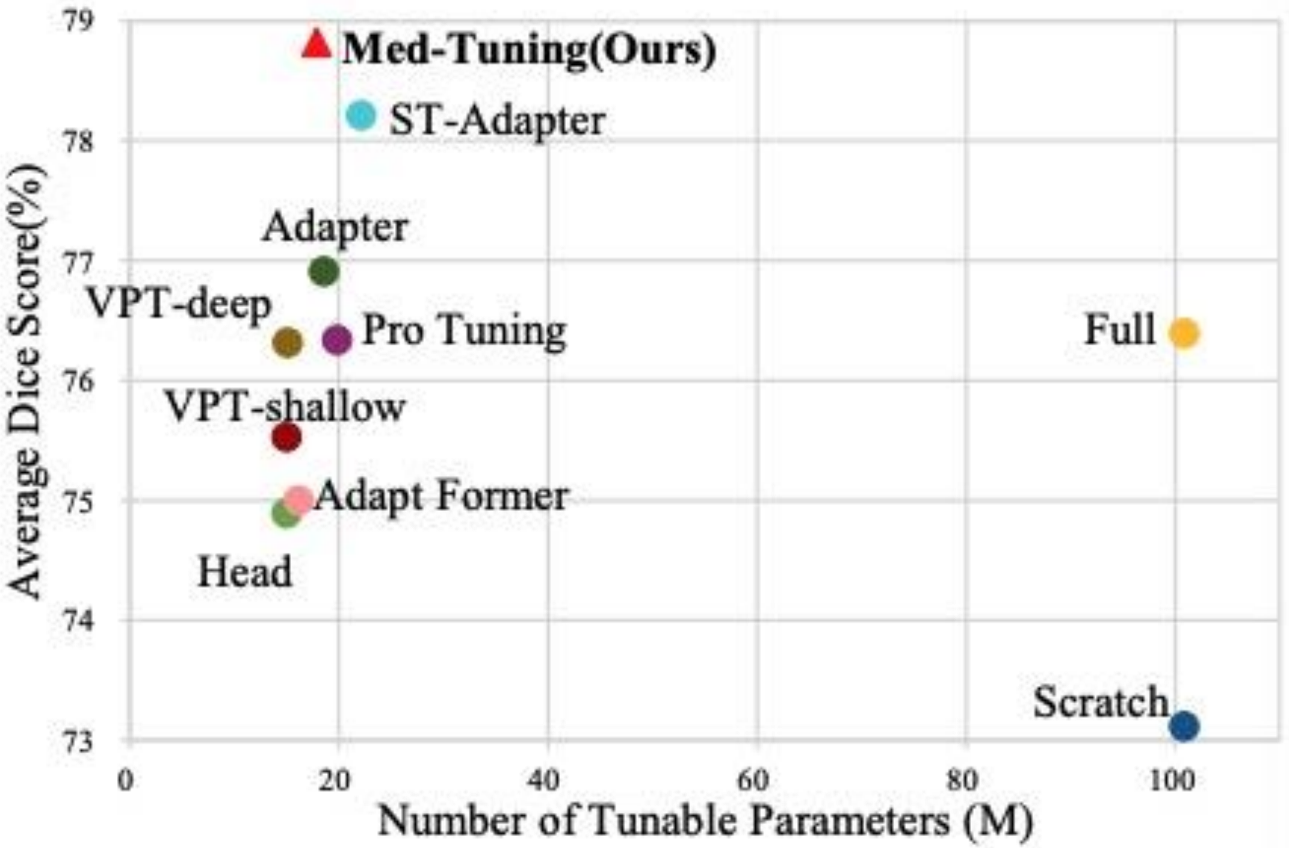
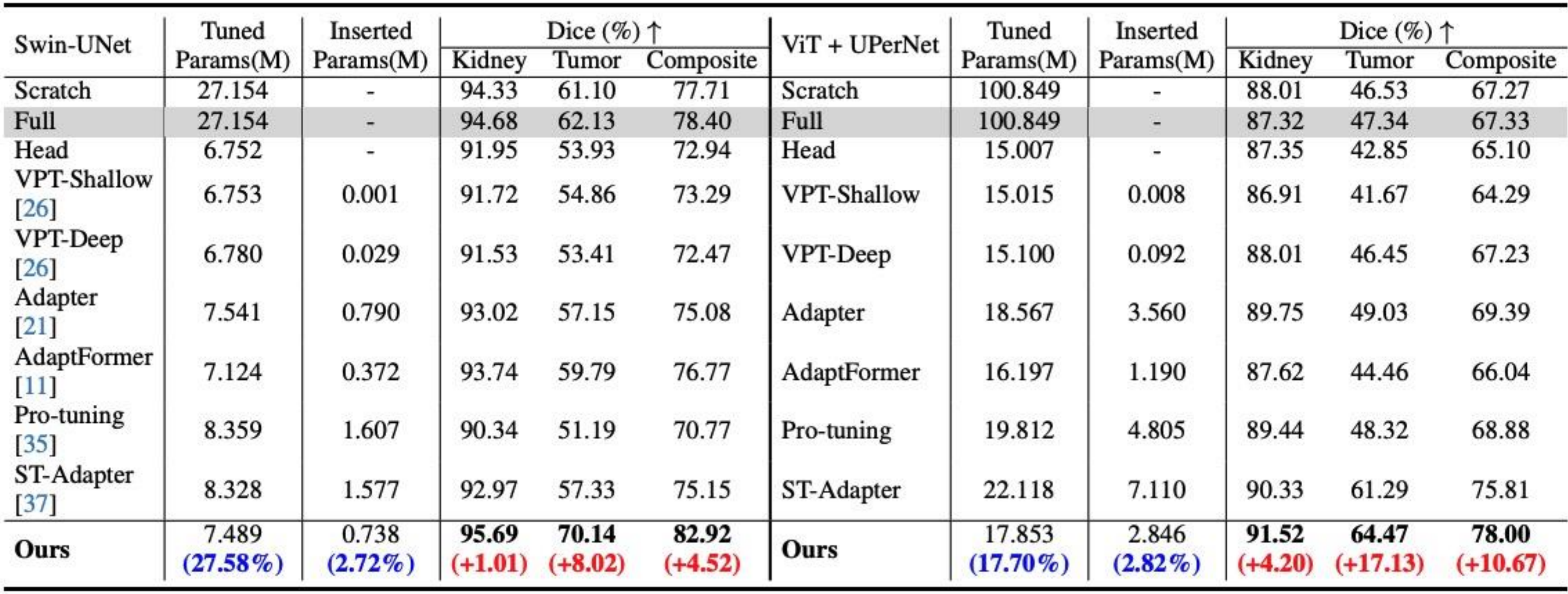


Figure 2. Comparison with previous PETL methods in terms of the trade-off between the number of tuned parameters and segmentation accuracy. The backbone ViT-B/16 is pre-trained on ImageNet-21k and fine-tuned on BraTS2019 dataset.

Results - KiTS

Kidney tumor segmentation on CT



Swin-UNet	Tuned Params(M)	Inserted Params(M)	Dice (%) ↑			ViT + UPerNet	Tuned Params(M)	Inserted Params(M)	Dice (%) ↑		
			Kidney	Tumor	Composite				Kidney	Tumor	Composite
Scratch	27.154	-	94.33	61.10	77.71	Scratch	100.849	-	88.01	46.53	67.27
Full	27.154	-	94.68	62.13	78.40	Full	100.849	-	87.32	47.34	67.33
Head	6.752	-	91.95	53.93	72.94	Head	15.007	-	87.35	42.85	65.10
VPT-Shallow [26]	6.753	0.001	91.72	54.86	73.29	VPT-Shallow	15.015	0.008	86.91	41.67	64.29
VPT-Deep [26]	6.780	0.029	91.53	53.41	72.47	VPT-Deep	15.100	0.092	88.01	46.45	67.23
Adapter [21]	7.541	0.790	93.02	57.15	75.08	Adapter	18.567	3.560	89.75	49.03	69.39
AdaptFormer [11]	7.124	0.372	93.74	59.79	76.77	AdaptFormer	16.197	1.190	87.62	44.46	66.04
Pro-tuning [35]	8.359	1.607	90.34	51.19	70.77	Pro-tuning	19.812	4.805	89.44	48.32	68.88
ST-Adapter [37]	8.328	1.577	92.97	57.33	75.15	ST-Adapter	22.118	7.110	90.33	61.29	75.81
Ours	7.489 **(27.58%)**	0.738 **(2.72%)**	**95.69** **(+1.01)**	**70.14** **(+8.02)**	**82.92** **(+4.52)**	**Ours**	17.853 **(17.70%)**	2.846 **(2.82%)**	**91.52** **(+4.20)**	**64.47** **(+17.13)**	**78.00** **(+10.67)**

Visual Results

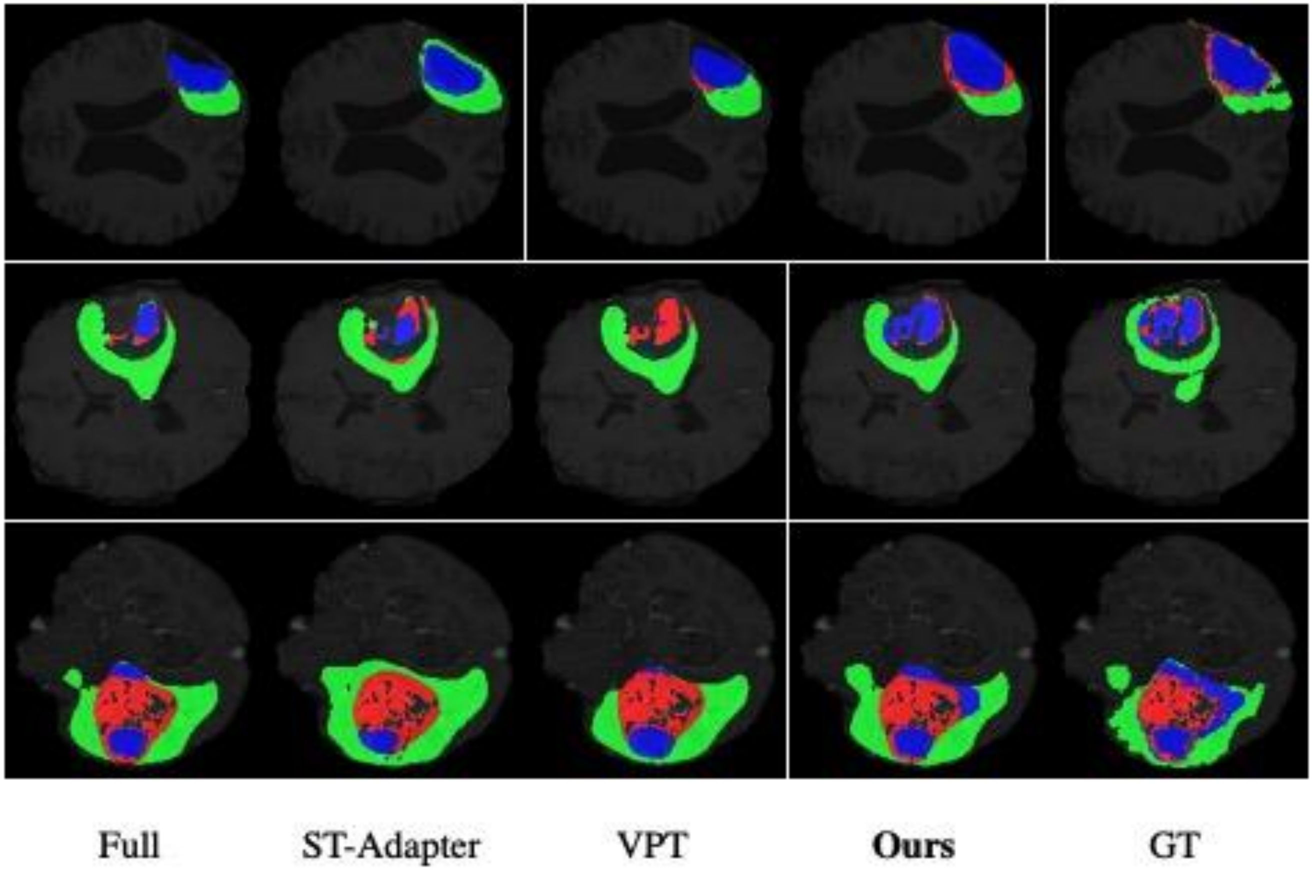


Figure 4. The visual comparison of segmentation results on BraTS 2019 dataset. The blue, red and green regions denote the enhancing tumors, non-enhancing tumors, and peritumoral edema. Full and GT denote full fine-tuning and ground truth.

Visual Results

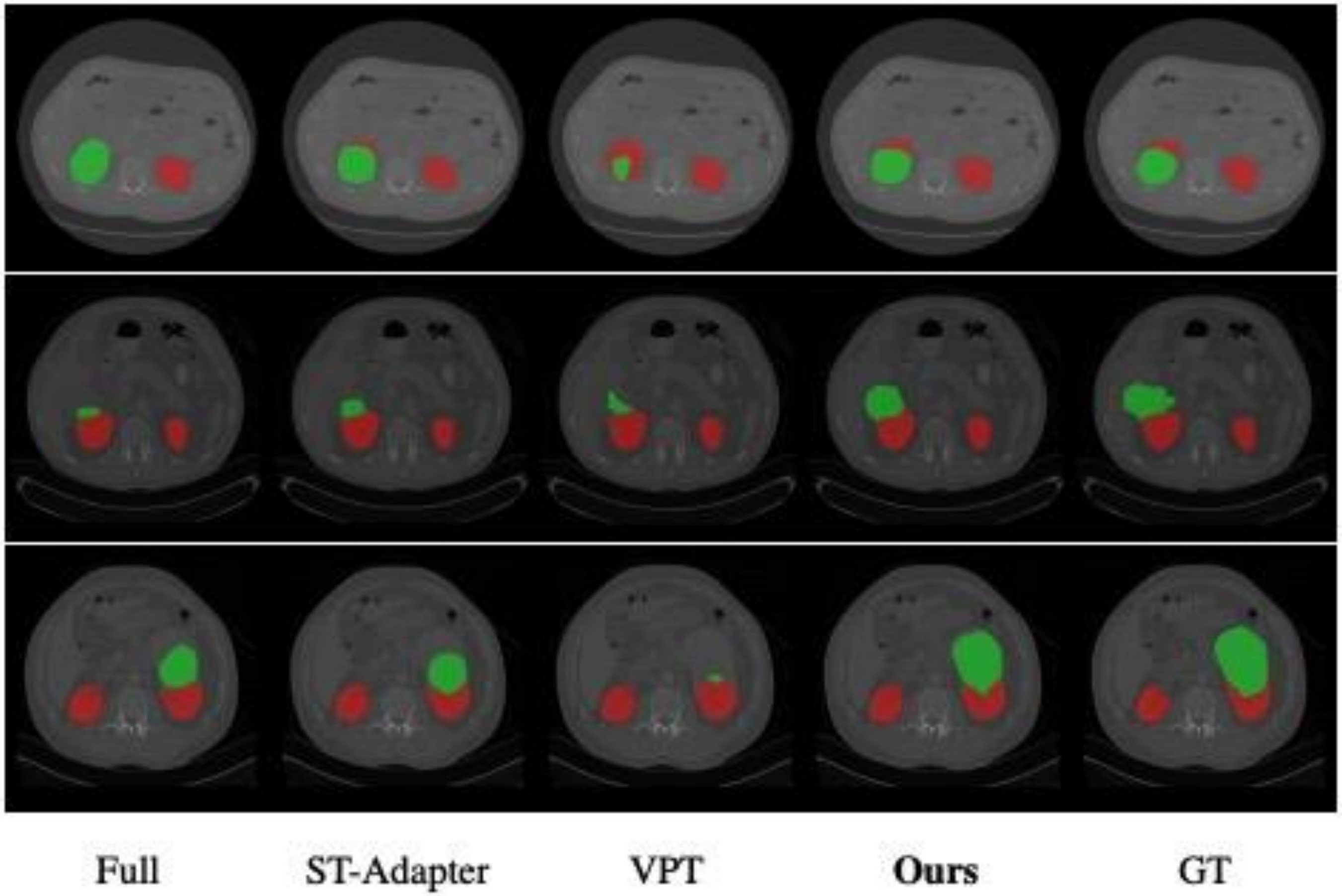


Figure 5. The visual comparison of segmentation results on KiTS 2019 dataset. The red and green regions denote the kidneys and kidney tumors. Full, GT denote full finetuning and ground truth.

Thank you!

Question?