

Parameter-Efficient Fine-Tuning

Donald Shenaj

donald.shenaj@di.unipi.it

November 13, 2026

Lecture Outline

1. Parameter-Efficient Fine-Tuning
2. Low-Rank Adaptation (LoRA)
3. LoRA for Computer Vision

Recap

Given a pretrained model f_{θ_0}, where θ_0 denotes its pretrained parameters:

- **Linear probing**: freeze θ_0 and train only a task-specific head ϕ.
- **Partial fine-tuning**: train ϕ and a subset $\theta_{\text{train}} \subsetneq \theta_0$.
- **Full fine-tuning**: train ϕ and all pretrained parameters θ_0.

Parameter-Efficient Fine-Tuning (PEFT)

- Keep pretrained parameters frozen and train a small task-specific parameter set ϕ.
- $|\phi| \ll |\theta_0|$, where $|\cdot|$ counts scalar parameters.

Parameter-Efficient Fine-Tuning

The Model Ecosystem

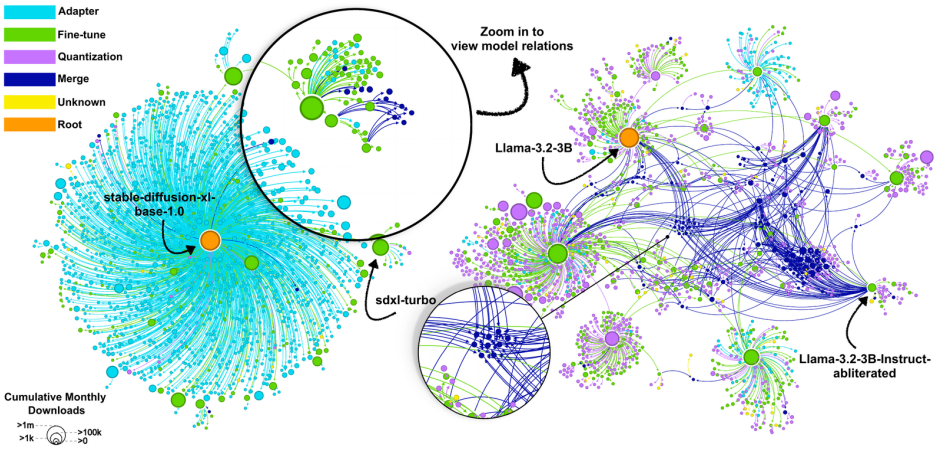


Figure: [*Horwitz et al., 2025: Charting an Atlas of All the World's Models*]

Why Parameter-Efficient Fine-Tuning?

Given a pretrained model with $|\theta_0|$ parameters:

- **Full fine-tuning**: update all $|\theta_0|$ pretrained parameters.
- **Memory requirements** with Adam and FP32 precision (32 bits = 4 bytes):

$$\underbrace{4|\theta_0|}_{\text{weights}} + \underbrace{4|\theta_0|}_{\text{gradients}} + \underbrace{8|\theta_0|}_{\text{Adam states}} = 16|\theta_0| \text{ bytes}$$

- **Weights**: 4 bytes per parameter.
- **Gradients**: 4 bytes per parameter.
- **Adam**: two FP32 states per parameter (first and second moments).
- **Multiple tasks**: full fine-tuning requires storing a separate model for each task.
- **PEFT**: train fewer parameters, reducing gradient, optimizer, and checkpoint memory.

Estimate excludes activations and other training buffers.

Bias-Term Fine-Tuning (BitFit)

BitFit trains bias parameters while keeping pretrained weight matrices frozen.

$$h = \sigma(Wx + b), \qquad \underbrace{W}_{\text{frozen}}, \quad \underbrace{b}_{\text{trained}}$$

- No additional modules or parameters.
- Approximately 0.08–0.09% of BERT parameters are trained.
- Performance comparable to full fine-tuning on several GLUE tasks.

Residual Adapters for Visual Domains

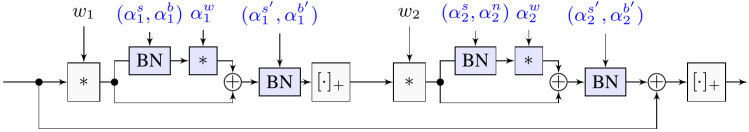


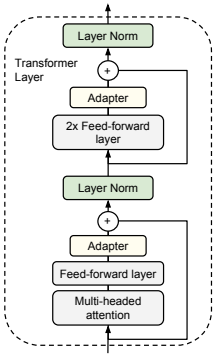
Figure: [*Rebuffi et al., 2017: Learning Multiple Visual Domains with Residual Adapters*]

For each domain d, introduce a residual adapter:

$$\boxed{h' = h + \alpha_d * h}$$

- α_d: learned 1×1 convolutional filter specific to domain d.
- Convolutional backbone shared across domains.
- Each domain has its own adapter and normalization parameters.
- Evaluated on the ten Visual Decathlon datasets.

Bottleneck Adapters for Transformers



[*Houlsby et al., 2019*]

For each downstream task t, insert a residual bottleneck adapter:

$$h' = h + W_{\text{up}}^{(t)} \sigma \left(W_{\text{down}}^{(t)} h \right)$$

$$W_{\text{down}}^{(t)} \in \mathbb{R}^{r \times d}, \qquad W_{\text{up}}^{(t)} \in \mathbb{R}^{d \times r}$$

- Bottleneck: reduce the feature dimension $d \to r \to d$, with $r \ll d$.
- $2dr$ trainable weights per adapter, excluding biases.
- Pretrained Transformer weights remain frozen.
- **Limitation**: additional sequential operations increase inference latency.

Adapters for 3D Medical Segmentation

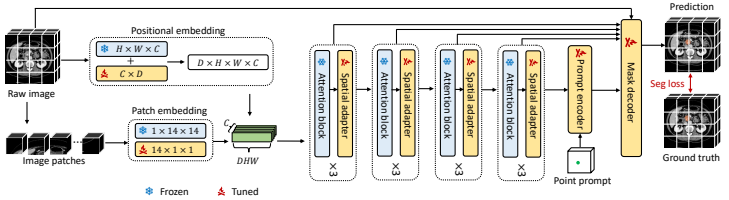


Figure: [*Gong et al., 2023: 3DSAM-adapter*]

Adapt a 2D-pretrained SAM to 3D medical volumes (e.g., CT and MRI).

- **2D SAM**: processes spatial information within individual image slices.
- **3DSAM-adapter**: introduces depth-wise 3D convolutions in spatial adapters.
- 3D convolution operates on features arranged along depth, height, and width:

$$X \in \mathbb{R}^{D \times H \times W \times C}$$

- Enables feature exchange across adjacent slices while reusing pretrained SAM weights.

CLIP-Adapter: Motivation

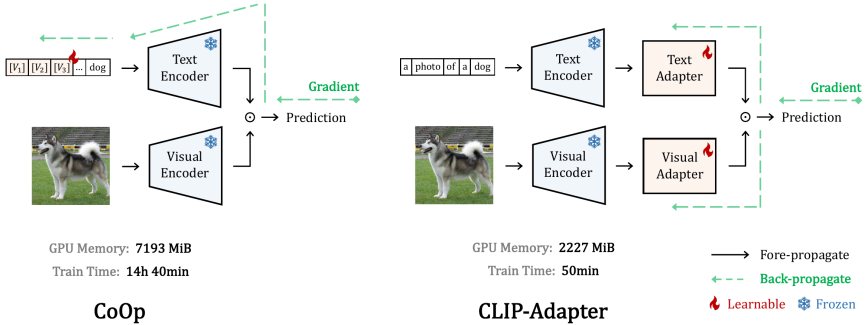


Figure: [*Gao et al., 2021: CLIP-Adapter*]

Few-shot classification: adapt a pretrained model using only a few labeled examples per class.

- **Example**: classify rare diseases from medical images with only 5 labeled examples per class.
- **CoOp**: learn prompt embeddings processed by the frozen CLIP text encoder.
- **CLIP-Adapter**: train a small MLP after the frozen CLIP image encoder.

CLIP-Adapter: Architecture

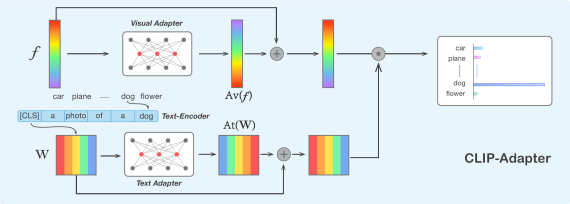


Figure: [*Gao et al., 2021: CLIP-Adapter*]

Given a few labeled images per class, adapt the features produced by CLIP:

- f: image embedding from the frozen CLIP encoder.
- $A_v(f)$: new embedding produced by the trainable visual adapter.

Combine the original and adapted features: $\boxed{f^\star = (1-\alpha)f + \alpha A_v(f)}$

- $f^\star$: final image embedding used for classification.
- Compare $f^\star$ with CLIP text embeddings using cosine similarity.

Prompt and Prefix Tuning

Prompt tuning: prepend p trainable embeddings to the input sequence X.

$$X' = [P; X]$$

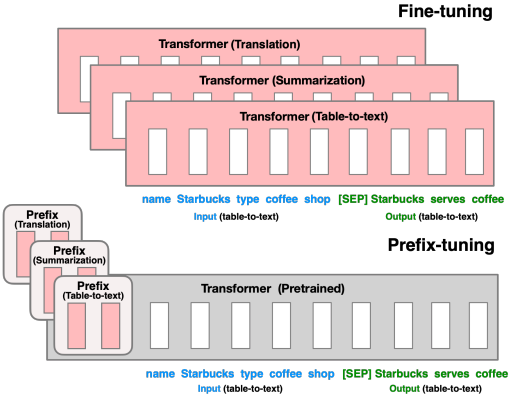
- $P \in \mathbb{R}^{p \times d}$: learned prompt embeddings.

Prefix tuning: prepend p trainable keys and values to the attention matrices.

$$K'_\ell = [P^\ell_K; K_\ell], \qquad V'_\ell = [P^\ell_V; V_\ell]$$

- K_ℓ, V_ℓ: keys and values at Transformer layer ℓ.
- P^ℓ_K, P^ℓ_V: learned key and value prefixes for that layer.

Pretrained Transformer weights remain frozen.



[*Li and Liang, 2021: Prefix-Tuning*]

Visual Prompt Tuning (VPT)

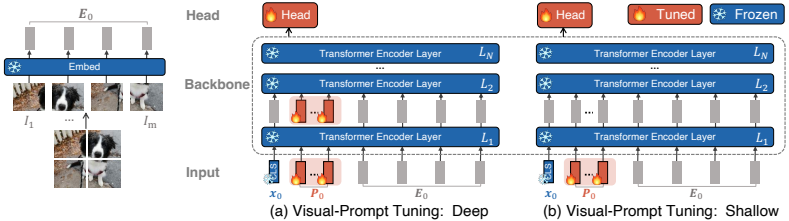


Figure: [*Jia et al., 2022: Visual Prompt Tuning*]

Learn p prompt embeddings $P \in \mathbb{R}^{p \times d}$, with the same dimension as ViT patch tokens.

- Prompts are trainable vectors, not patches extracted from the image.
- **VPT-Shallow**: insert prompts only at the input layer.
- **VPT-Deep**: insert separate trainable prompts at each Transformer layer.
- Train the prompts and classification head. Keep the ViT backbone frozen.

Visual Prompt Tuning: Results

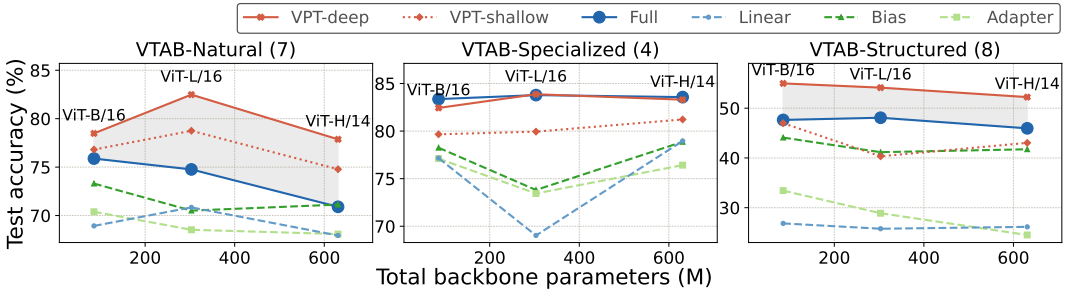


Figure: [*Jia et al., 2022: Visual Prompt Tuning*]

- VTAB-1k: 19 vision tasks, with 1,000 labeled examples per task.
- VPT-Deep outperforms full fine-tuning on Natural and Structured tasks for several ViT configurations.
- Performance depends on the task group and pretrained backbone.

Low-Rank Adaptation (LoRA)

Motivation for LoRA

Full fine-tuning updates pretrained weight matrices:

$$W' = W_0 + \Delta W$$

- Full fine-tuning requires storing task-specific model weights.
- Bottleneck adapters introduce additional operations during inference.
- Prompt and prefix tuning modify the input or attention context.

LoRA: learn low-rank weight updates that can be merged into pretrained matrices.

Bottleneck Adapters: Inference Latency

Bottleneck adapters add sequential operations inside Transformer blocks.

GPT-2 medium, batch size	**32**	**16**	**1**
Fine-tuning / LoRA	1449.4 ms	338.0 ms	19.8 ms
Adapter$^{\mathrm{L}}$	1482.0 ms	354.8 ms	23.9 ms
Adapter$^{\mathrm{H}}$	1492.2 ms	366.3 ms	25.8 ms

- Adapter overhead is larger at smaller batch sizes in this experiment.
- Merged LoRA introduces no additional adapter operations at inference.

Prefix Tuning: Parameter Budget

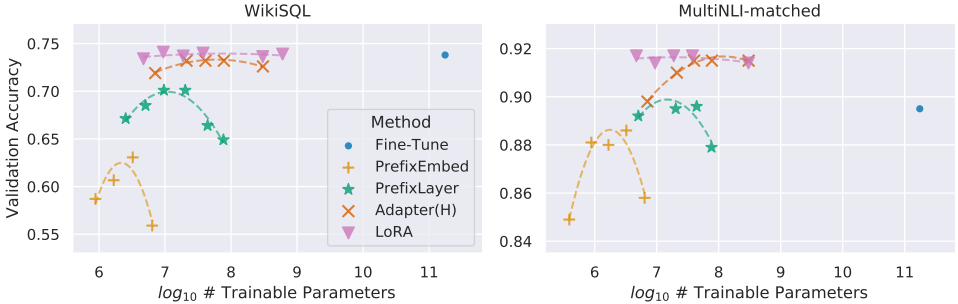
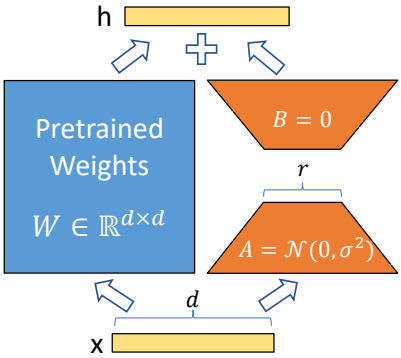


Figure: [*Hu et al., 2022: LoRA, Figure 2*]

- Prefix tuning introduces learned key and value vectors.
- Increasing the prefix length does not consistently improve accuracy in this experiment.
- LoRA achieves higher accuracy with a small parameter budget.

LoRA: Low-Rank Adaptation



[*Hu et al., 2022: LoRA*]

Pretrained linear layer:

$$h = W_0 x$$

Low-rank update:

$$\Delta W = \frac{\alpha}{r} BA$$

$$h = W_0 x + \frac{\alpha}{r} BAx$$

$$\boxed{h = \left(W_0 + \frac{\alpha}{r} BA\right) x}$$

$$B \in \mathbb{R}^{d_{\text{out}} \times r}, \quad A \in \mathbb{R}^{r \times d_{\text{in}}}$$

- W_0 frozen, A and B trainable.

LoRA: Trainable Parameters

Let $|W|$ denote the number of scalar elements in a matrix

$$W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \quad B \in \mathbb{R}^{d_{\text{out}} \times r}, \quad A \in \mathbb{R}^{r \times d_{\text{in}}}$$

- **Full fine-tuning**: $|W_0| = d_{\text{out}} d_{\text{in}}$ trainable weights.
- **LoRA**: $|A| + |B| = r d_{\text{in}} + d_{\text{out}} r$ trainable weights.

$$\begin{aligned} \frac{|A| + |B|}{|W_0|} &= \frac{r(d_{\text{in}} + d_{\text{out}})}{d_{\text{out}} d_{\text{in}}} \\ &= \frac{r d_{\text{in}}}{d_{\text{out}} d_{\text{in}}} + \frac{r d_{\text{out}}}{d_{\text{out}} d_{\text{in}}} \\ &= \boxed{r \left(\frac{1}{d_{\text{out}}} + \frac{1}{d_{\text{in}}} \right)} \end{aligned}$$

LoRA: Initialization

Initialize the LoRA matrices as

$$A \sim \mathcal{N}(0, \sigma^2), \qquad B = 0$$

The initial update is zero:

$$\Delta W(0) = \frac{\alpha}{r} BA = 0$$

- The initial forward pass matches the pretrained model.
- B receives gradients on the first optimization step.
- Gradients reach A after B changes.

LoRA: Gradient Updates

Matrix and component-wise notation (i output, j rank, k input):

$$h = W_0 x + \frac{\alpha}{r} BAx, \qquad h_i = (W_0 x)_i + \frac{\alpha}{r} \sum_j B_{ij} \sum_k A_{jk} x_k$$

Gradient with respect to B: B_{ij} affects only h_i.

$$\frac{\partial \mathcal{L}}{\partial B_{ij}} = \frac{\alpha}{r} \frac{\partial \mathcal{L}}{\partial h_i} (Ax)_j \quad \Longrightarrow \quad \frac{\partial \mathcal{L}}{\partial B} = \frac{\alpha}{r} \frac{\partial \mathcal{L}}{\partial h} (Ax)^\top$$

Gradient with respect to A: A_{jk} affects all outputs through B.

$$\frac{\partial \mathcal{L}}{\partial A_{jk}} = \frac{\alpha}{r} \sum_i \frac{\partial \mathcal{L}}{\partial h_i} B_{ij} x_k \quad \Longrightarrow \quad \frac{\partial \mathcal{L}}{\partial A} = \frac{\alpha}{r} B^\top \frac{\partial \mathcal{L}}{\partial h} x^\top$$

At initialization, $B = 0$:

$$\boxed{\frac{\partial \mathcal{L}}{\partial A} = 0, \qquad \frac{\partial \mathcal{L}}{\partial B} \neq 0 \quad \text{in general}}$$

LoRA in Transformer Layers

For token matrix $X \in \mathbb{R}^{N \times d}$:

$$Q = XW_Q^\top, \qquad K = XW_K^\top, \qquad V = XW_V^\top$$

LoRA can adapt all four attention projections:

$$W_Q' = W_Q + \frac{\alpha}{r} B_Q A_Q$$

$$W_K' = W_K + \frac{\alpha}{r} B_K A_K$$

$$W_V' = W_V + \frac{\alpha}{r} B_V A_V$$

$$W_O' = W_O + \frac{\alpha}{r} B_O A_O$$

- Each adapted matrix uses a separate pair A, B.
- Selected projections and MLP matrices can also be adapted.

LoRA: Which Projections to Adapt?

GPT-3 175B with a fixed LoRA parameter budget.

Adapted matrices	Rank per matrix	WikiSQL	MultiNLI-m
W_Q	8	70.4	91.0
W_K	8	70.0	90.8
W_V	8	73.0	91.5
W_O	8	73.2	91.3
W_Q, W_K	4	71.4	91.3
W_Q, W_V	4	73.7	91.3
W_Q, W_K, W_V, W_O	2	73.7	91.7

- Adapting multiple projections can improve performance under the same budget.
- The preferred subset depends on the task.

LoRA: Rank Selection

GPT-3 175B results for different adaptation ranks.

	Adapted matrices	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL	W_Q	68.8	69.6	70.5	70.4	70.0
	W_Q, W_V	73.4	73.3	73.7	73.8	73.5
	W_Q, W_K, W_V, W_O	74.1	73.7	74.0	74.0	73.9
MultiNLI	W_Q	90.7	90.9	91.1	90.7	90.7
	W_Q, W_V	91.3	91.4	91.3	91.6	91.4
	W_Q, W_K, W_V, W_O	91.2	91.7	91.7	91.5	91.4

- Small ranks can already match larger ranks in these experiments.
- The adapted matrices matter as much as the rank.

LoRA: Merging for Inference

After training, merge the update into the pretrained matrix:

$$W_{\text{merged}} = W_0 + \frac{\alpha}{r} BA$$

The forward pass becomes

$$h = W_{\text{merged}} x$$

- **Merged**: no additional LoRA matrix multiplications at inference.
- **Unmerged**: retain separate task-specific adapters.
- Different adapters can share the same pretrained backbone.

LoRA: Memory Requirements

Memory component	**LoRA**
Pretrained weights	Still stored
Gradients for pretrained weights	Not stored
Optimizer states for pretrained weights	Not stored
LoRA gradients and optimizer states	Stored
Task-specific checkpoints	Small adapter weights
Activation memory	Still needed for backpropagation

LoRA: Experimental Results

GPT-3 175B	Trainable	WikiSQL	MNLI-m	SAMSum	
Method	**params**	**Acc.**	**Acc.**	**R1/R2**	**RL**
Full fine-tuning	175,255.8M	73.8	89.5	52.0/28.0	44.5
BitFit	14.2M	71.3	91.0	51.3/27.4	43.5
PreEmbed	3.2M	63.1	88.6	48.3/24.2	40.5
PreLayer	20.2M	70.1	89.5	50.8/27.3	43.5
$\text{Adapter}^{\text{H}}$	7.1M	71.9	89.8	53.0/28.9	44.8
$\text{Adapter}^{\text{H}}$	40.1M	73.2	91.5	53.2/29.0	45.1
LoRA	**4.7M**	73.4	**91.7**	**53.8/29.8**	**45.9**
LoRA	**37.7M**	**74.0**	91.6	53.4/29.2	45.1

- LoRA with 4.7M trainable parameters achieves comparable accuracy to full fine-tuning.
- Increasing LoRA parameters to 37.7M improves WikiSQL accuracy in this experiment.

QLoRA: Quantized Low-Rank Adaptation

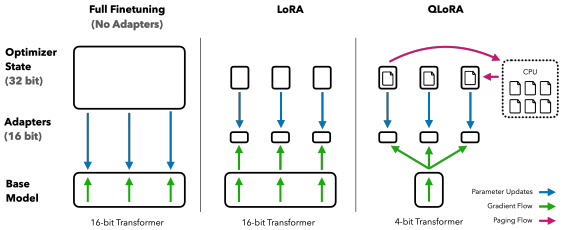


Figure: [*Dettmers et al., 2023: QLoRA*]

$$h = \text{dequant}(W_{\text{NF4}})x + \frac{\alpha}{r} BAx$$

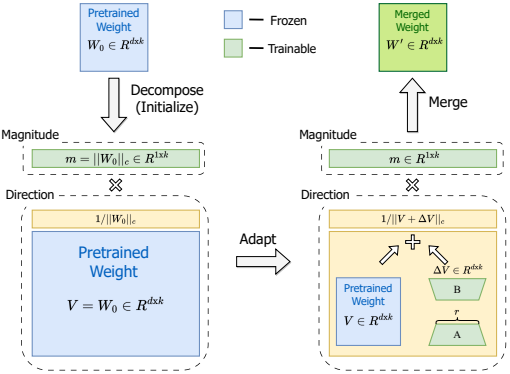
- Store frozen pretrained weights using 4-bit NF4 quantization.
- Dequantize for higher-precision matrix multiplication.
- Train LoRA parameters without updating the quantized backbone.

QLoRA: Memory Optimizations

- **NF4**: 4-bit quantization for approximately normally distributed weights.
- **Double quantization**: quantize the quantization scale constants.
- **Paged optimizers**: use CPU/GPU unified memory to manage optimizer-memory spikes.

$$\underbrace{W_{\mathrm{NF4}}}_{\text{stored in 4 bits}} \xrightarrow{\text{dequantize}} \underbrace{W_{\mathrm{compute}}}_{\text{higher-precision operations}}$$

Weight-Decomposed Low-Rank Adaptation (DoRA)



[*Liu et al., 2024: DoRA*]

Separate weight magnitudes and directions:

$$W = V \operatorname{diag}\left(\frac{m}{\|V\|_{\text{col}}}\right)$$

where

$$m_j = \|W_{:,j}\|_2$$

- $m \in \mathbb{R}^{d_{\text{in}}}$ controls column magnitudes.
- V determines weight directions.
- Normalization is performed column-wise.

DoRA: Low-Rank Direction Updates

Apply LoRA to the weight direction:

$$V = W_0 + \frac{\alpha}{r} BA$$

The adapted weight matrix is:

$$\boxed{W' = V \,\mathrm{diag}\left(\frac{m}{\|V\|_{\mathrm{col}}}\right)}$$

- Freeze the pretrained matrix W_0.
- Train A, B, and the magnitude vector m.
- Unlike standard LoRA, column magnitudes are adapted explicitly.

AdaLoRA: Adaptive Rank Allocation

AdaLoRA uses an SVD-like parameterization:

$$\Delta W = P \Lambda Q, \qquad \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_r)$$

- Estimate the importance of low-rank components during training.
- Prune less important components and allocate different ranks across layers.

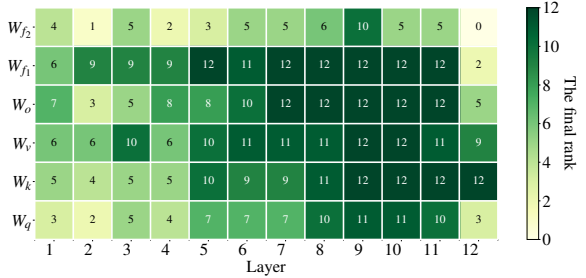


Figure: [*Zhang et al., 2023: AdaLoRA*]

Vector-based Random Matrix Adaptation (VeRA)

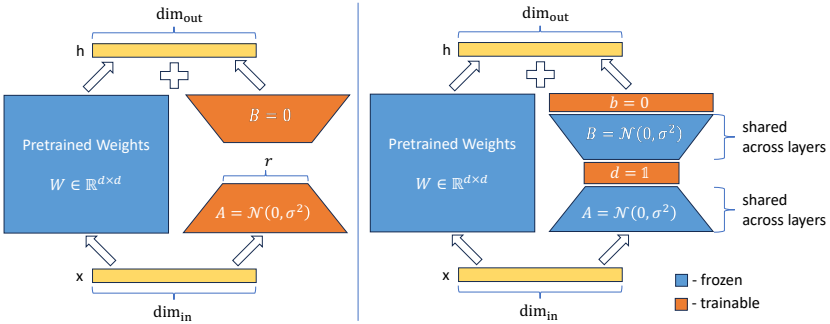


Figure: [*Kopiczko et al., 2024: VeRA*]

Reuse frozen random projections:

$$\boxed{\Delta W = \text{diag}(b) B \, \text{diag}(d) A}$$

$$b \in \mathbb{R}^{d_{\text{out}}}, \qquad d \in \mathbb{R}^{r}$$

LoRA for Computer Vision

LoRA for Vision Transformers

Recall the attention projections in a ViT, with $X \in \mathbb{R}^{N \times D}$:

$$Q = XW_Q^\top, \qquad K = XW_K^\top, \qquad V = XW_V^\top$$

LoRA modifies selected projection matrices:

$$\boxed{W_i' = W_i + \frac{\alpha}{r} B_i A_i, \qquad i \in \{Q, K, V, O\}}$$

- Pretrained ViT weights remain frozen.
- Only the low-rank matrices A_i, B_i are trained.
- LoRA can also be applied to MLP projections.

LoRA for ViTs: Comparison with Other PEFT Methods

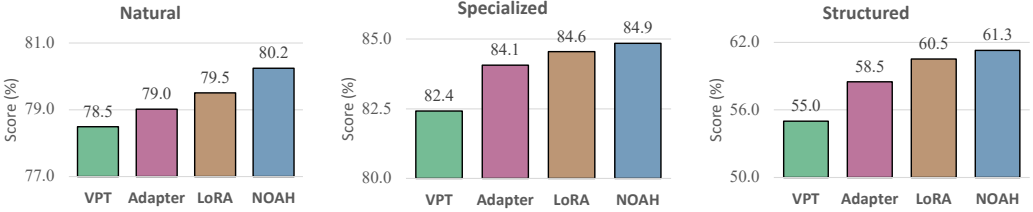


Figure: [*Zhang et al., 2022: Neural Prompt Search (NOAH)*]

- LoRA outperforms VPT and standard adapters across all three VTAB-1k task groups.
- NOAH combines multiple adaptation mechanisms and achieves the highest accuracy.

LoRA for Medical Image Segmentation

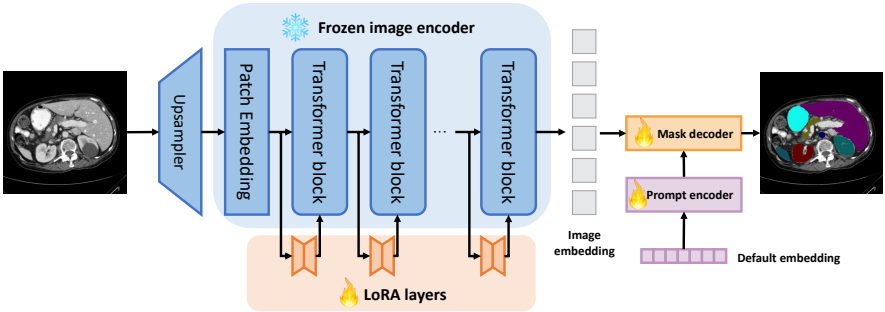


Figure: [*Zhang and Liu, 2023: SAMed*]

Adapt a pretrained SAM image encoder to medical image segmentation.

- **Input**: 2D medical images instead of natural images.
- **Adaptation**: apply LoRA to query and value projections.
- **Output**: medical segmentation masks from the mask decoder.
- Pretrained image-encoder weights remain frozen.

Conv-LoRA: Spatial Adaptation

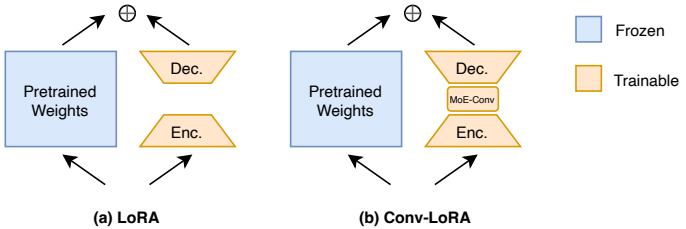


Figure: [*Zhong et al., 2024: Convolution Meets LoRA*]

Conv-LoRA inserts a convolutional module between A and B:

$$\boxed{h = W_0 x + \frac{\alpha}{r} B \ \text{MoEConv}(Ax)}$$

- A: reduce feature channels to rank r. B: project back to the output dimension.
- MoEConv: select convolutional experts operating at different spatial scales.
- Each expert uses interpolation and a 3×3 convolution.

SemLA: Domain-Specific LoRA Library

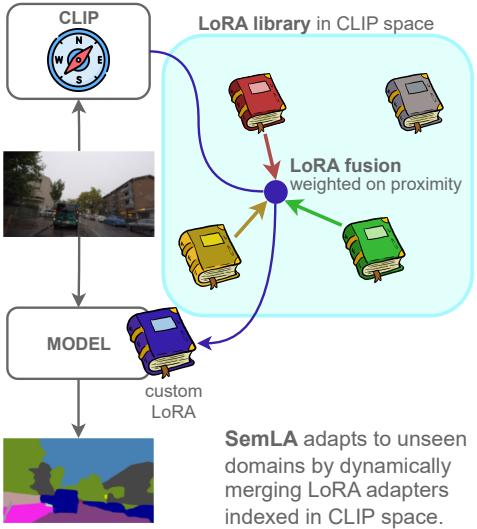
Task: open-vocabulary semantic segmentation under domain shift.

- Example: segment street scenes captured in rain, fog, or at night.
- Start from the same pretrained segmentation model for each domain.
- Train a separate LoRA adapter on each source domain.

$$\Delta W_i = B_i A_i, \qquad i = 1, \ldots, M$$

- M: number of source domains. Each domain has its own A_i, B_i.
- Compute a CLIP embedding centroid for each domain and store it with its LoRA.

SemLA: LoRA Retrieval and Fusion



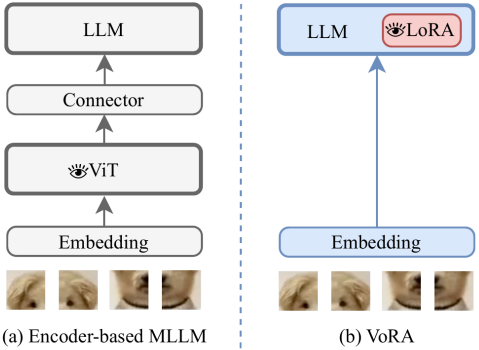
For a test image x:

1. Extract its CLIP embedding.
2. Retrieve the k most similar domain adapters.
3. Combine their LoRA updates:

$$\Delta W(x) = \sum_{i=1}^{k} w_i(x) B_i A_i$$

- $w_i(x)$: similarity-based weight for retrieved adapter i.
- The resulting update is specific to the input image.
- No gradient updates at inference.

Vision as LoRA (VoRA)



[*Wang et al., 2025: Vision as LoRA*]

Integrate visual processing into a pretrained language model.

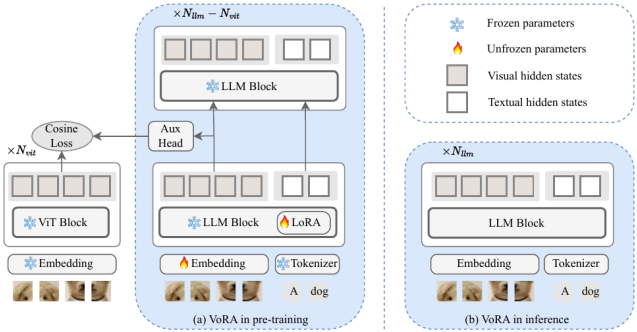
- Introduce visual patch embeddings and vision-specific LoRA modules.
- Reuse frozen pretrained language-model weights.
- Distill visual features from a pretrained ViT teacher.
- No separate ViT encoder is required at inference.

VoRA: Block-Wise Distillation

Align intermediate LLM visual representations with a ViT teacher:

$$\mathcal{L}^{i}_{\text{distill}} = \frac{1}{S}\sum_{s=1}^{S}\left[1 - \cos\left(g_i(h^{i,s}_{\text{llm}}), h^{i,s}_{\text{vit}}\right)\right]$$

- i: selected Transformer block; s: visual token index.
- S: number of aligned visual tokens; g_i: auxiliary projection head.
- Train visual adaptation modules and auxiliary heads.
- Combine distillation with the language-modeling objective.



[*Wang et al., 2025: Vision as LoRA*]

Comparing PEFT Methods

Method	Trainable components	Modification
BitFit	Bias vectors	Existing parameters
Bottleneck adapters	Down/up projections	Residual feature module
Prompt / VPT	Prompt embeddings	Additional input tokens
Prefix tuning	Keys and values	Additional attention context
LoRA	Low-rank matrices A, B	Weight update BA
QLoRA	LoRA matrices	Quantized frozen backbone
DoRA	A, B, column magnitudes	Direction and magnitude
AdaLoRA	Low-rank SVD-like factors	Variable rank by layer
VeRA	Scaling vectors	Shared random projections

Summary

- PEFT reduces the number of trainable parameters while retaining pretrained weights.
- LoRA represents weight updates as low-rank matrix products.
- QLoRA reduces backbone storage. DoRA and AdaLoRA modify the update parameterization.
- In vision, LoRA is used for ViT adaptation, segmentation, and vision-language models.

Next Lectures

- Lab: Vision-Language Models and PEFT.
- Diffusion models for image and video generation.
- Personalized diffusion models: Textual Inversion, DreamBooth, and LoRA.


Parameter-Efficient Fine-Tuning
Donald Shenaj
donald.shenaj@unipi.it
November 13, 2026
1


Lecture Outline
Motivation (PEFT)
Prefix Tuning
LoRA
Adapters
Beyond LoRA
Efficient Fine-Tuning for Vision
2

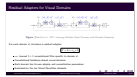

Recap
3

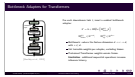

Parameter-Efficient Fine-Tuning
4


The Model Ecosystem
5


Why Parameter-Efficient Fine-Tuning?
6


How Does Fine-Tuning (SGFT)
7

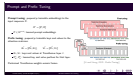

Residual Adapters for Visual Domains
8

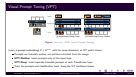

Sentence Adapters for Transformers
9


Adapters for 2D Multi-Head Representation
10


IA3 Adapter: Motivation
11


IA3 Adapter: Architecture
12


Prompt and Prefix Tuning
13


Visual Prompt Tuning (VPT)
14

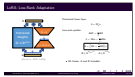

Visual Prompt Tuning: Results
15

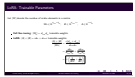

Low-Rank Adaptation (LoRA)
16


Motivation for LoRA
17


Sentence Adapters: Inference Latency
18


Prefix Tuning: Parameter Budget
19


LoRA: Low-Rank Adaptation
20


LoRA: Trainable Parameters
21


LoRA: Initialization
22

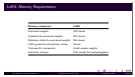

LoRA: Gradient Updates
23


LoRA in Transformer Layers
24


LoRA: Which Projections to Adapt?
25


LoRA: Rank Selection
26

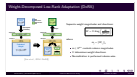

LoRA: Merging for Inference
27


LoRA: Memory Requirements
28


LoRA: Experimental Results
29

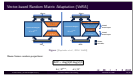

QLoRA: Quantized Low-Rank Adaptation
30


QLoRA: Memory Optimization
31


Weight-Decomposed Low-Rank Adaptation (DoRA)
32


DoRA: Loss/Rank-Direction Updates
33

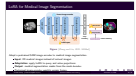

AdaLoRA: Adaptive Rank Allocation
34


Vector-based Random Matrix Adaptation (VeRA)
35


LoRA for Computer Vision
36


LoRA for Vision Transformers
37


LoRA for NTA: Comparison with Other PEFT Methods
38


LoRA for Medical Image Segmentation
39


Conv-LoRA: Spatial Adaptation
40

Bench: Domain-Specific LoRA Library
41

Bench: LoRA Retrieval and Fusion
42

Vision vs LoRA (VoRA)
43

LoRA: Block-Wise Distillation
44

Comparing PEFT Methods
45

Summary
46

Next Lectures
47