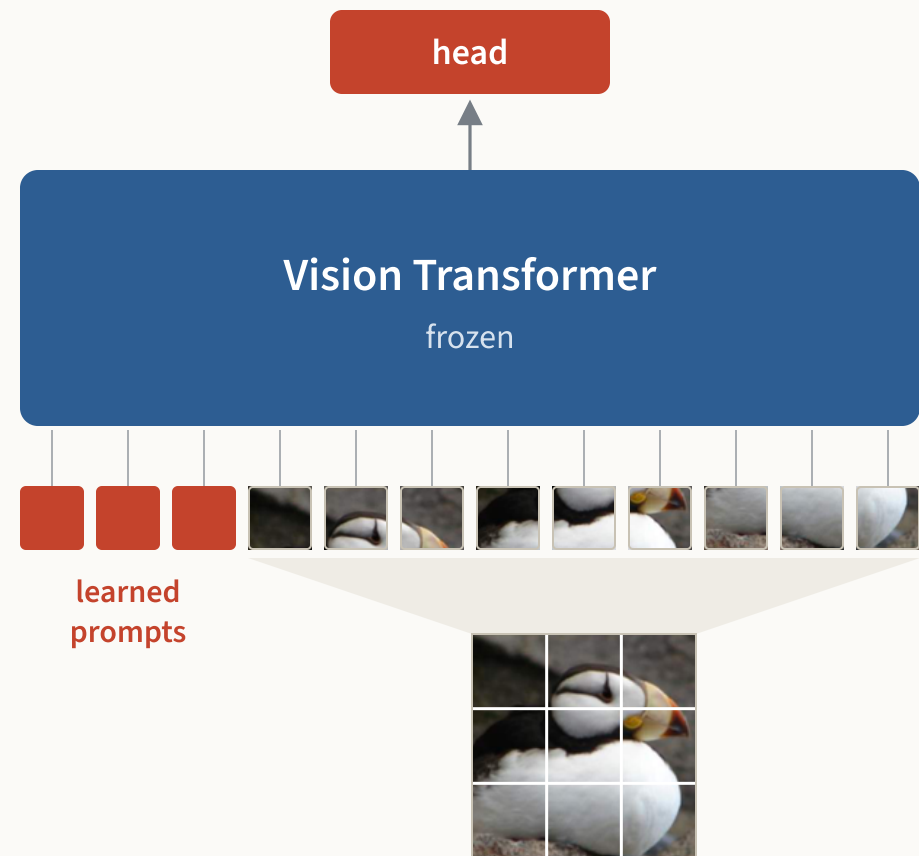
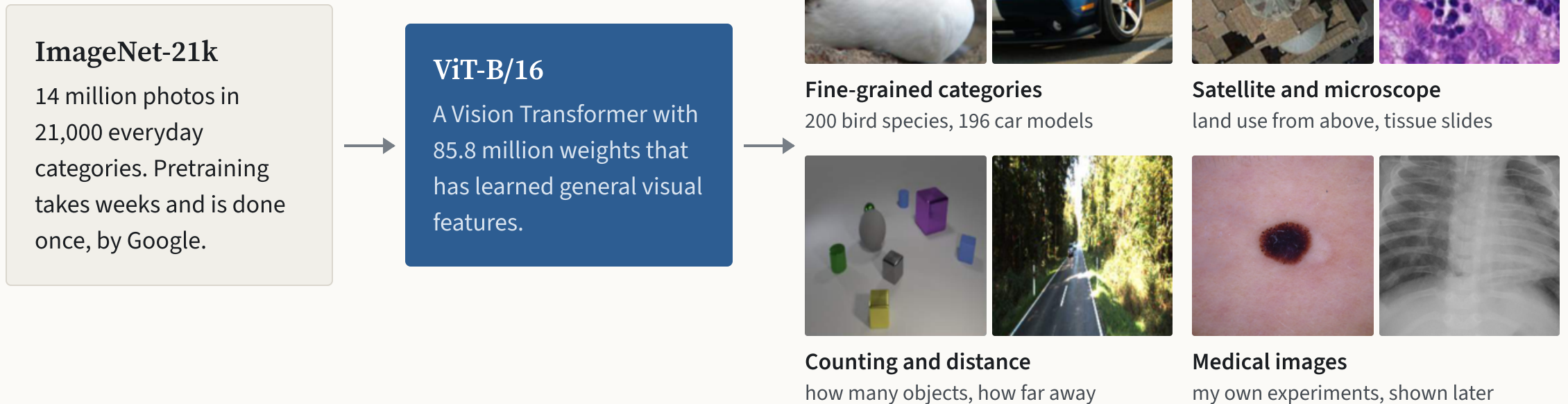


Visual Prompt Tuning

Teach a frozen Vision Transformer a new task by learning only a few extra input vectors.



Pretrain one big model once, then adapt it to each task



Each new task is small: most tasks in the paper have only **1,000 training images**, far too few to train from scratch. So we start from the pretrained model and **adapt** it.

The paper asks how to adapt it best: accurately, and cheaply enough to store for every task. Next, the standard answer and what it costs, then the cheaper methods tried before VPT.

Full fine-tuning stores a new copy of the model per task

Full fine-tuning retrains every weight of the pretrained model on the new task, and it is usually the most accurate option. The cost is that each task keeps its own retrained copy, so storage grows with every task and with the size of the model.

Shown for **1 task** on **ViT-B**, **86 million weights**. The next pages show 24 tasks, then the largest model.

Full fine-tuning: 1 retrained ViT-B copy



343 MB

1 copy at 343 MB each

VPT: one shared ViT-B plus 1 small file



each red speck is one task file, to scale

345 MB

343 MB once, plus 2.2 MB per task

Full fine-tuning stores a new copy of the model per task

Full fine-tuning retrains every weight of the pretrained model on the new task, and it is usually the most accurate option. The cost is that each task keeps its own retrained copy, so storage grows with every task and with the size of the model.

Shown for 24 tasks on ViT-B, 86 million weights.

Full fine-tuning: 24 retrained ViT-B copies



8.2 GB

24 copies at 343 MB each

VPT: one shared ViT-B plus 24 small files



395 MB

343 MB once, plus 2.2 MB per task

Full fine-tuning stores a new copy of the model per task

Full fine-tuning retrains every weight of the pretrained model on the new task, and it is usually the most accurate option. The cost is that each task keeps its own retrained copy, so storage grows with every task and with the size of the model.

Shown for 24 tasks on ViT-H, 632 million weights.

Full fine-tuning: 24 retrained ViT-H copies



61 GB

24 copies at 2.5 GB each

VPT: one shared ViT-H plus 24 small files



2.7 GB

2.5 GB once, plus 8.7 MB per task

Cheaper methods existed, but none matched full fine-tuning

		Mean accuracy (%)	Weights trained
Full fine-tuning	retrain every weight	73.9	100%
Only the top of the model is trained			
Partial-1	train the head and the last of 12 layers	66.2	8.35%
Linear probe	train only a new output layer	63.1	0.09%
MLP-3 head	train a bigger, three-layer head	62.8	1.45%
Small parts in or beside the backbone are trained			
Bias	train only the bias terms of every layer	71.0	0.21%
Adapter	insert small trainable blocks in every layer	66.7	0.36%
Sidetune	train a small network beside the model	57.0	10.09%

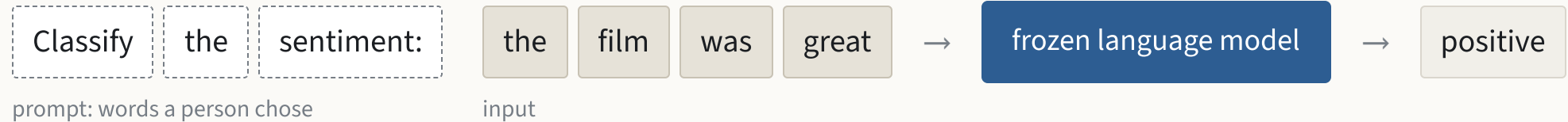
Bias came closest and even beat full fine-tuning on 8 of the 24 tasks, but on average no cheaper method reached it. VPT takes a third route: it leaves the model alone and changes the input.

Mean test accuracy over the paper's 24 tasks, ViT-B/16 (Jia et al. 2022, Table 1, mean of the four group averages). The right column is the share of weights trained for each task.

Where the word "prompt" comes from

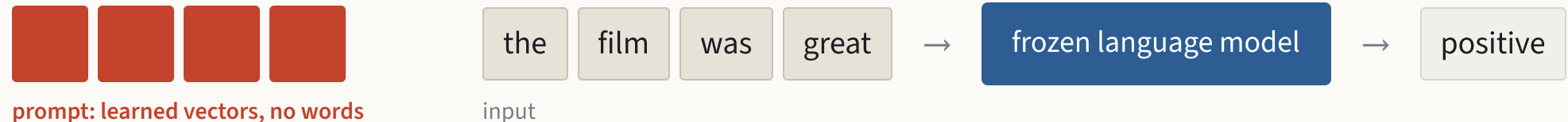
2020

GPT-3



2021

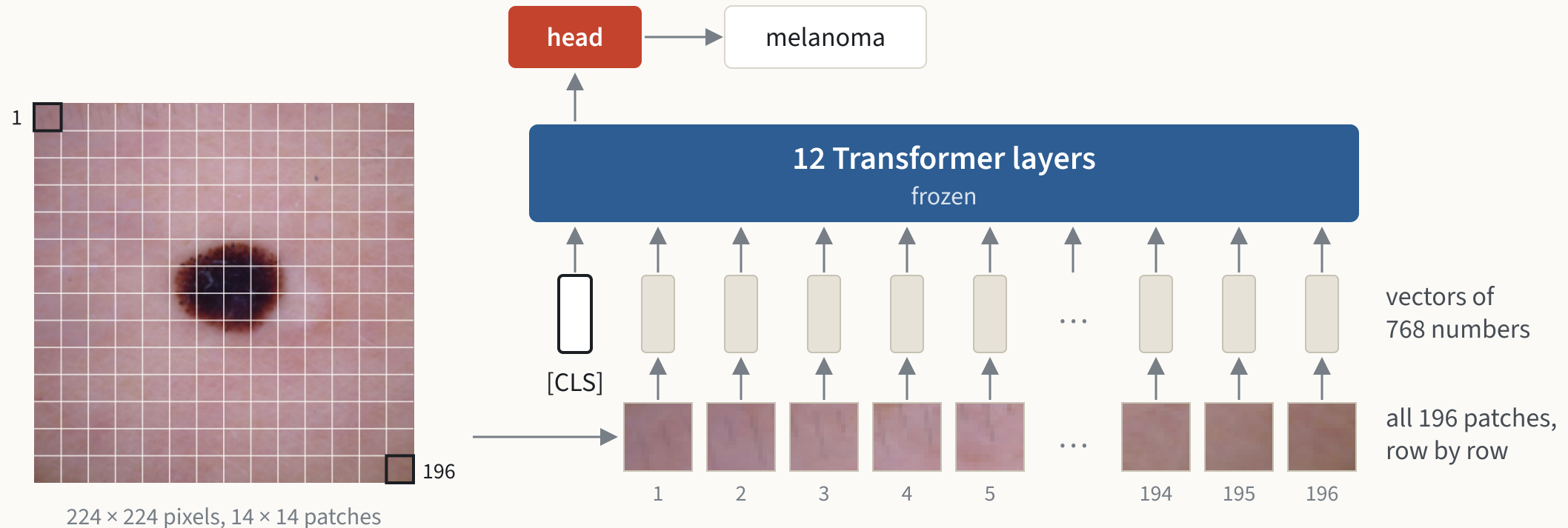
Prompt tuning



In 2020, a prompt was text. GPT-3, a frozen model with 175 billion weights, was only trained to continue text, yet it does a new task when you type a description of the task in front of the input. Not a single weight changes. But words are discrete, so the only way to improve a prompt is trial and error.

In 2021, prompt tuning (Lester et al.) replaced the words with **learned vectors** that correspond to no word. They are trained by gradient descent on the task's examples while the model stays frozen, and they are all you store per task. Prefix-tuning (Li & Liang) does the same at every layer.

How a Vision Transformer reads an image



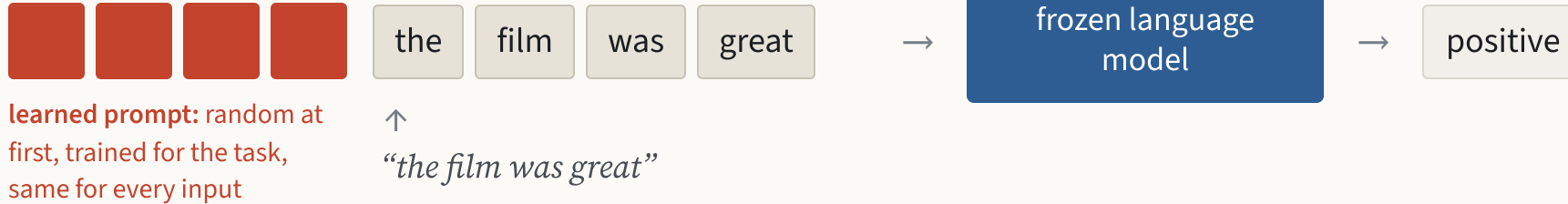
A fixed grid cuts the image into 196 patches of 16 × 16 pixels, and each patch, read row by row, becomes a vector of 768 numbers. One extra vector, [CLS], goes in front to collect a summary of the whole image: the 12 layers mix all 197 vectors, and the head reads only [CLS] to name the class.

The layers only see a sequence of vectors, so we can add a few more.

A visual prompt is the same idea applied to image patches

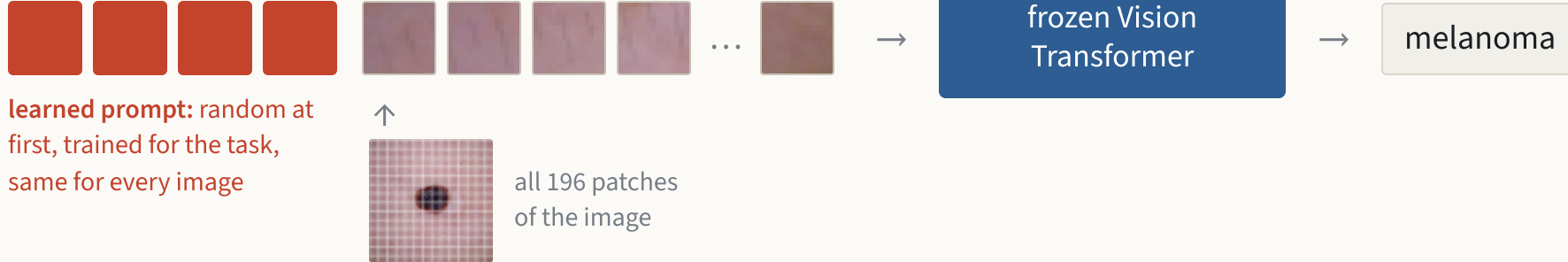
Language

Lester et al. 2021



Vision

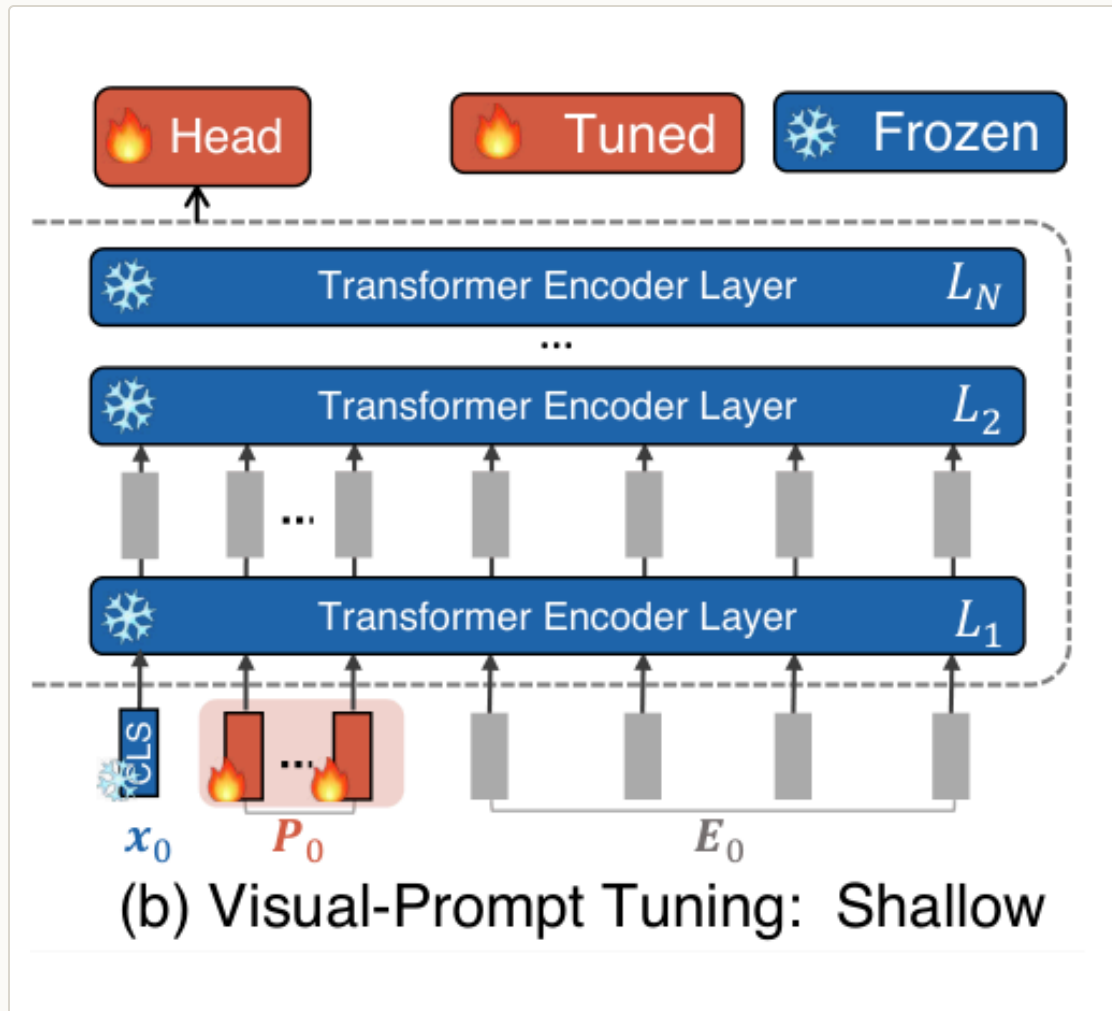
Jia et al. 2022



The grey vectors come from the input, as before. The red ones do not: they start as random numbers, are trained on the task's examples while the model stays frozen, and are then put in front of every input of that task. A visual prompt is therefore not text and not a pattern painted on the pixels, only learned numbers in the model's input space.

In language, prompt tuning matched full fine-tuning only once the model had about 10 billion weights. VPT beats full fine-tuning on 20 of 24 tasks with an 86-million-weight ViT-B.

VPT trains a few extra input vectors and the head



VPT-shallow

VPT-deep

The p prompt vectors P are added once, before the first layer. After that they pass through the 12 frozen layers like any other token.

$$[x_1, Z_1, E_1] = L_1([x_0, P, E_0]) \quad \text{prompts enter once}$$

$$[x_i, Z_i, E_i] = L_i([x_{i-1}, Z_{i-1}, E_{i-1}]) \quad i = 2 \dots 12$$

$$y = \text{Head}(x_{12})$$

Per task, only the red parts are trained and saved: 50 prompt vectors and the head, **0.14%** of the model.

VPT trains a few extra input vectors and the head

VPT-shallow

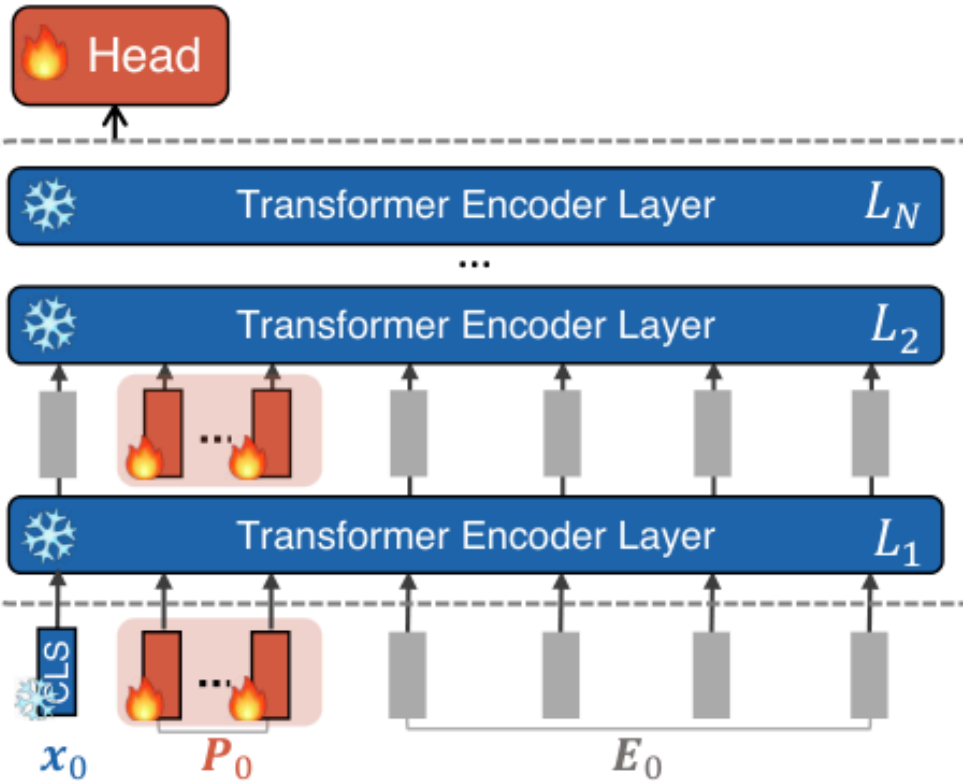
VPT-deep

Every layer gets its own fresh set of p prompt vectors. What the previous layer produced at those positions is thrown away: the blank in the equation.

$$[x_i, _, E_i] = L_i([x_{i-1}, P_{i-1}, E_{i-1}]) \quad i = 1 \dots 12$$

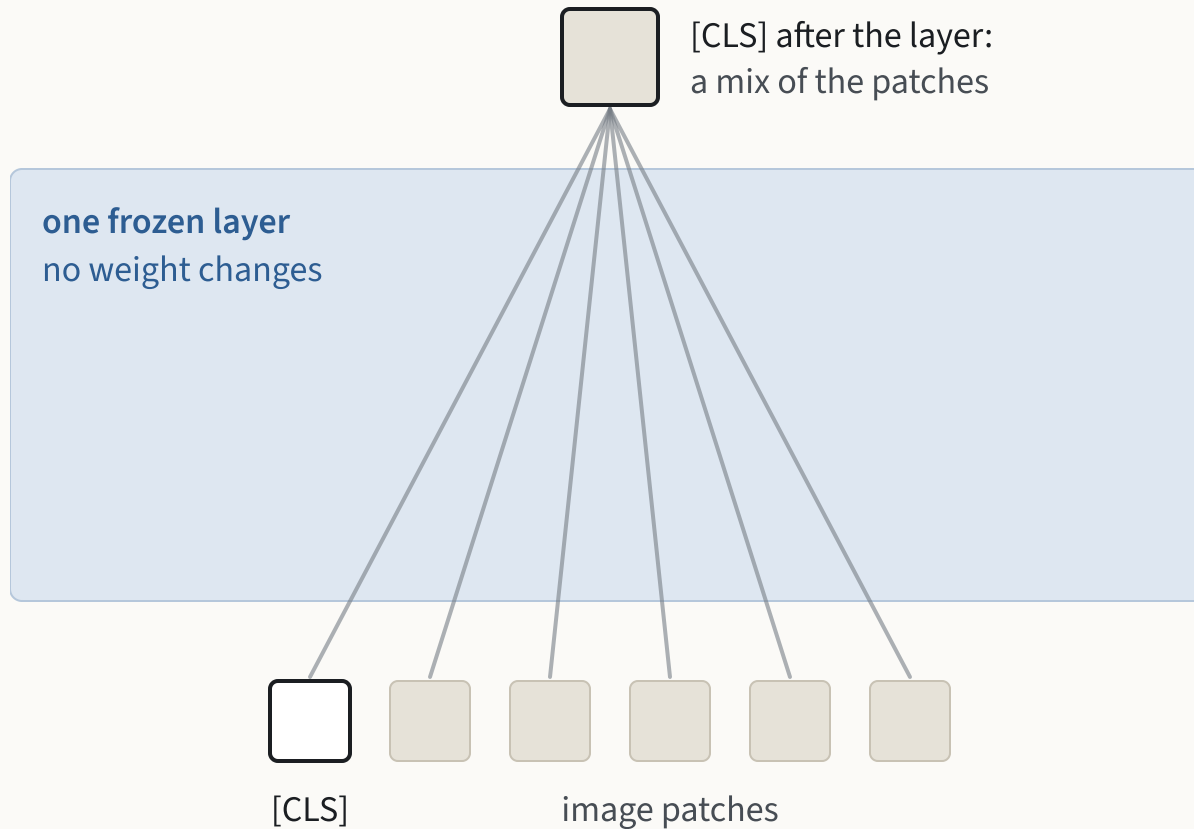
$$y = \text{Head}(x_{12})$$

Per task, only the red parts are trained and saved: 50 prompt vectors for each layer and the head, **0.63%** of the model.



(a) Visual-Prompt Tuning: Deep

Why a few extra tokens can steer a frozen network



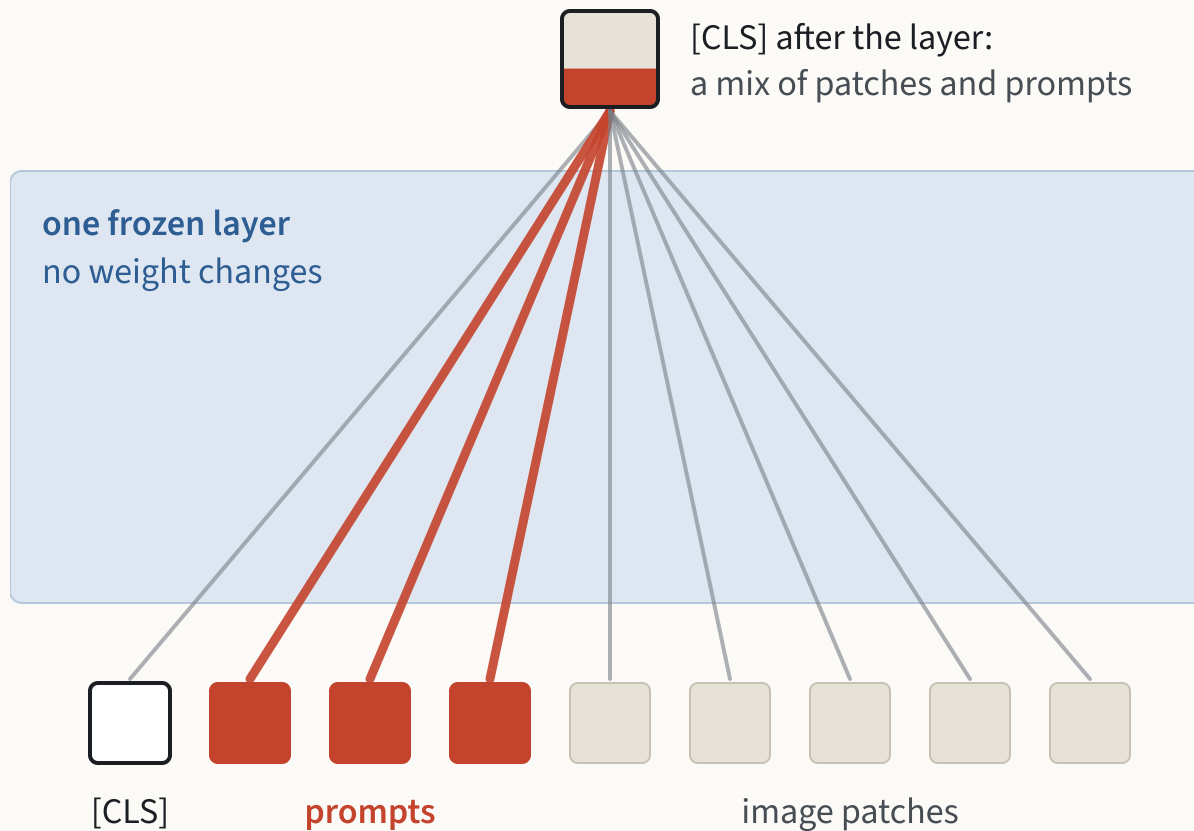
Without prompts

With prompts

In every layer, each token takes in information from all the other tokens; this is attention. With prompts in the sequence, [CLS] and every patch also take in information from the prompts.

So training the prompts changes what the frozen network computes, without changing a single weight.

Why a few extra tokens can steer a frozen network



Without prompts

With prompts

In every layer, each token takes in information from all the other tokens; this is attention. With prompts in the sequence, [CLS] and every patch also take in information from the prompts.

So training the prompts changes what the frozen network computes, without changing a single weight.

How much VPT trains and stores per task

$N \cdot p \cdot d = 12 \times 50 \times 768 = 460,800$ prompt numbers, plus 78,438 for a 102-class head

Numbers trained and stored per task

Full fine-tuning	all 85.8 million weights, retrained for this task	343 MB 100%
VPT-deep	the same 85.8 million weights, frozen and shared by every task	2.2 MB 0.63%

Compute per image

Plain ViT	197 tokens	1.00×
VPT-deep	197 tokens	1.27× +50 prompt tokens

Shown for $p = 50$; at $p = 200$ the file is 7.7 MB and compute is about 2.1×. More prompts keep the file small, but every image carries them through all 12 layers.

ViT-B/16, $N = 12$ layers, $d = 768$, 32-bit numbers. Compute counts multiply-adds per layer for n tokens ($12nd^2 + 2n^2d$). The paper measures about 2× test latency at $p = 200$ (Jia et al. 2022, Table 12).

VPT in code: a few lines and one new hyperparameter

```
# VPT-deep on a frozen ViT-B/16, simplified from demo/vpt/vpt_model.py
vit = timm.create_model(BACKBONE, pretrained=True)           # ImageNet-21k ViT-B/16
vit.requires_grad_(False)                                    # backbone frozen
prompts = nn.Parameter(torch.empty(12, p, 768).uniform_(-b, b)) # trained: 12 layers × p × 768
head = nn.Linear(768, num_classes)                           # trained: the classifier

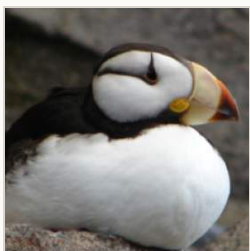
def forward(img):
    x = vit._pos_embed(vit.patch_embed(img))                  # [CLS] + 196 patch vectors
    for i, layer in enumerate(vit.blocks):
        rest = x[:, 1:] if i == 0 else x[:, 1 + p:]           # drop last layer's prompts
        P = prompts[i].expand(len(x), -1, -1)                 # this layer's prompts
        x = layer(torch.cat([x[:, :1], P, rest], dim=1))       # frozen layer on [CLS, P, rest]
    return head(vit.norm(x)[:, 0])                             # classify from [CLS]
```

The prompts start random, with Xavier-uniform values as in the official code. The paper tried smarter starts, such as class prototypes, and random did as well or better.

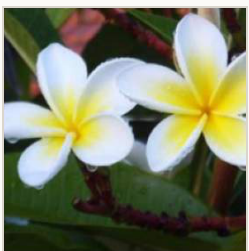
Training is plain SGD with momentum for 100 epochs on a cosine schedule. Learning rate and weight decay are picked on a validation split, for every method alike.

The one new hyperparameter is the prompt length p , chosen per task from {1, 5, 10, 50, 100, 200} on the validation split.

The paper tests VPT on 24 very different tasks



CUB-200



Oxford Flowers



Caltech101



DTD



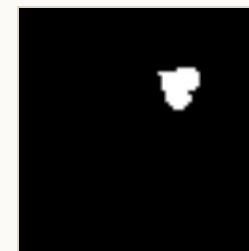
EuroSAT



Resisc45



CLEVR count



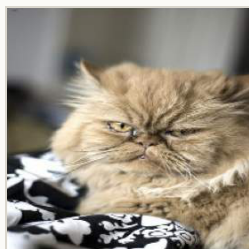
dSprites location



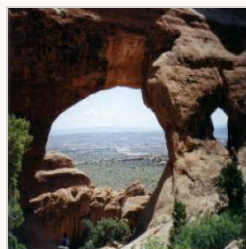
Stanford Cars



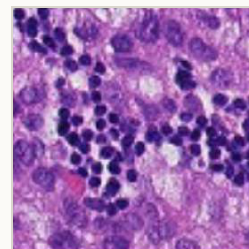
Stanford Dogs



Oxford Pets



SUN397



Patch Camelyon



Retinopathy



KITTI distance



SmallNORB azimuth

FGVC, 5 tasks

Fine-grained categories, such as 200 bird species, with thousands of training images.

VTAB Natural, 7 tasks

Ordinary photos of objects, textures, animals and scenes, close to the pretraining data.

VTAB Specialized, 4 tasks

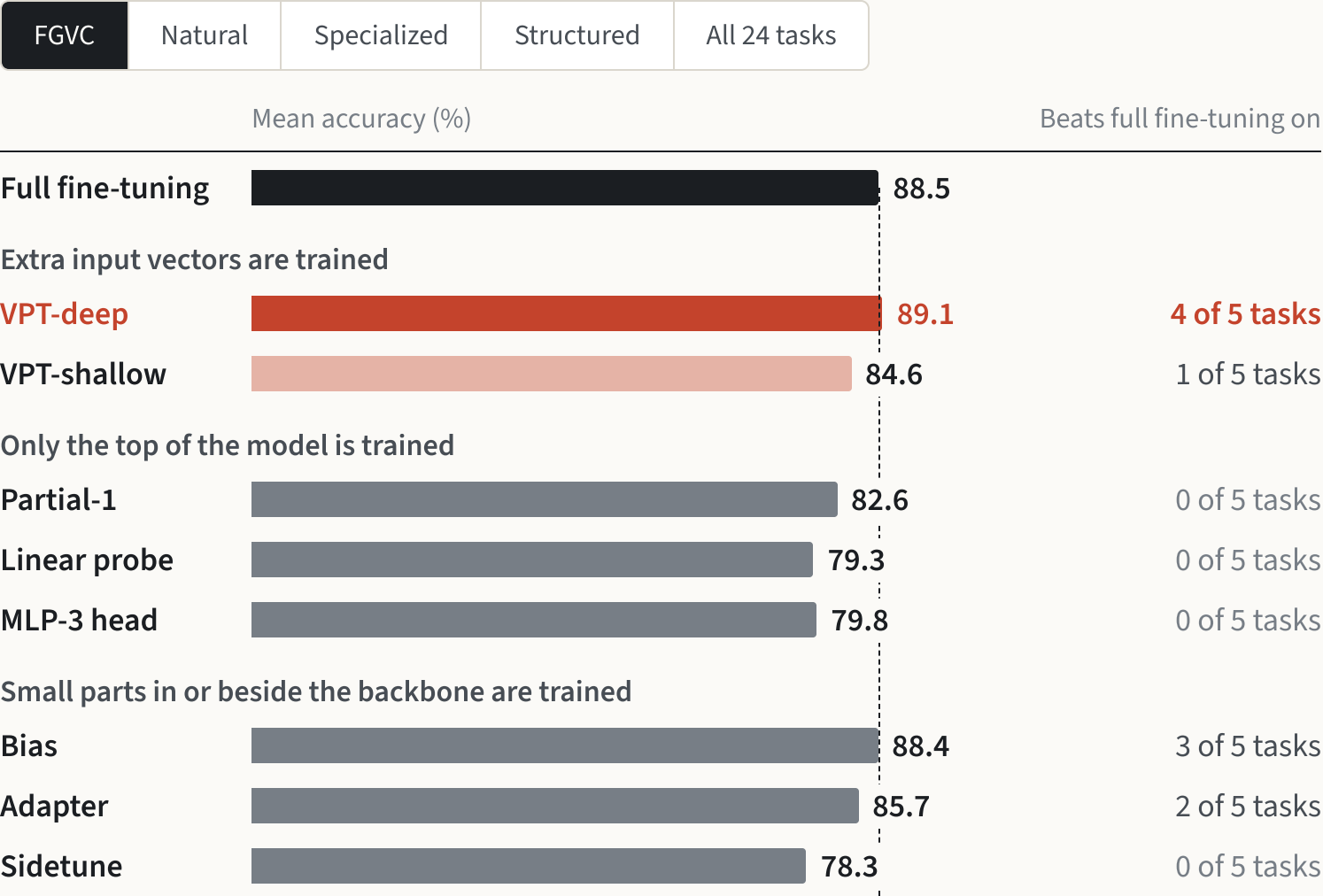
Images from special sensors: satellites, tissue slides under a microscope, and retina cameras.

VTAB Structured, 8 tasks

Geometry instead of categories, such as counting objects or judging distance.

All 24 tasks use the same frozen ViT-B/16, pretrained on ImageNet-21k. The 19 VTAB tasks have only 1,000 training images each.

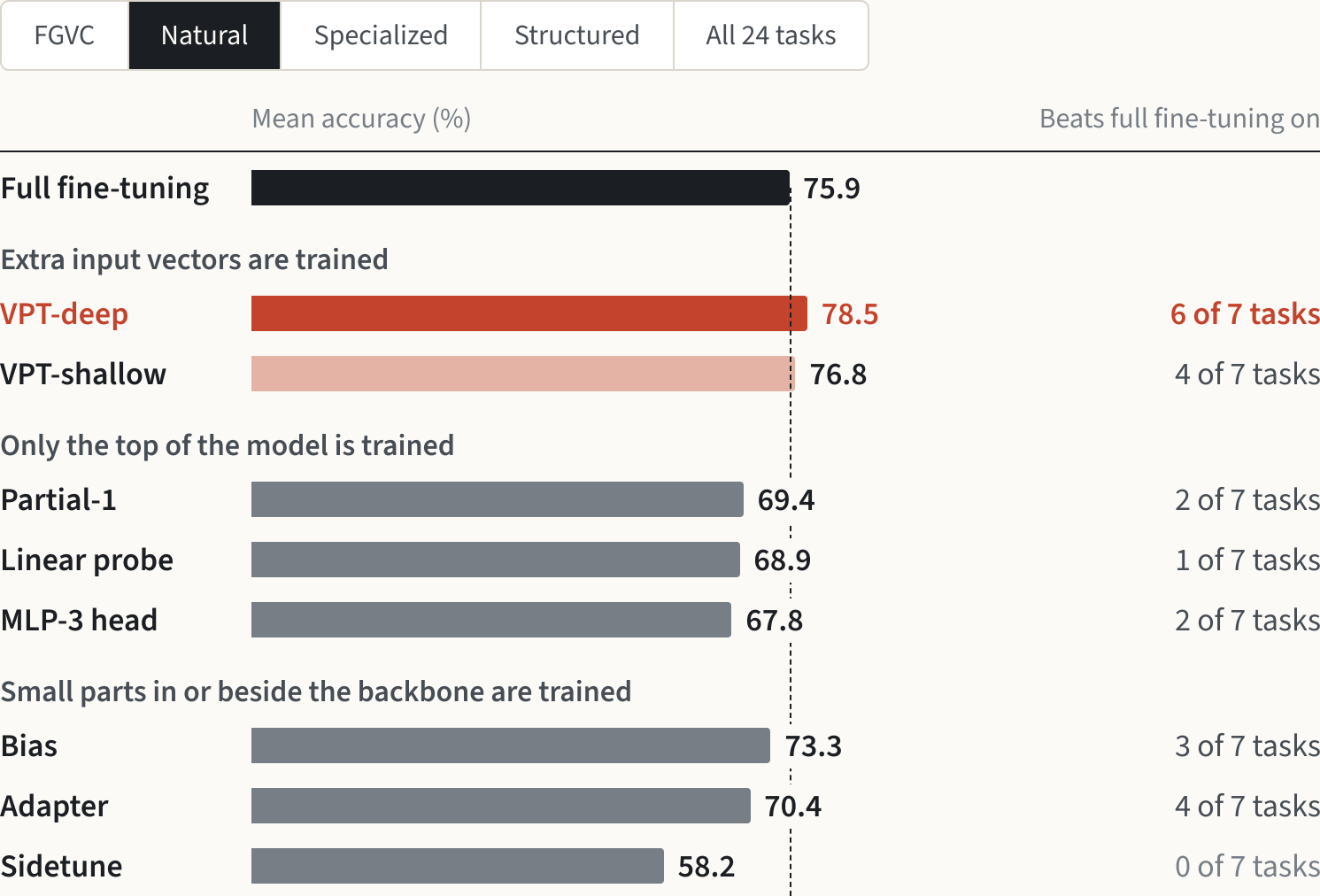
VPT-deep beats full fine-tuning on 20 of 24 tasks



These five tasks ask the model to tell apart 200 bird species, 196 car models, 120 dog breeds or 102 kinds of flower, with thousands of training images per task.

Every method does well on such photos, and VPT-deep only edges out full fine-tuning, 89.1 against 88.5.

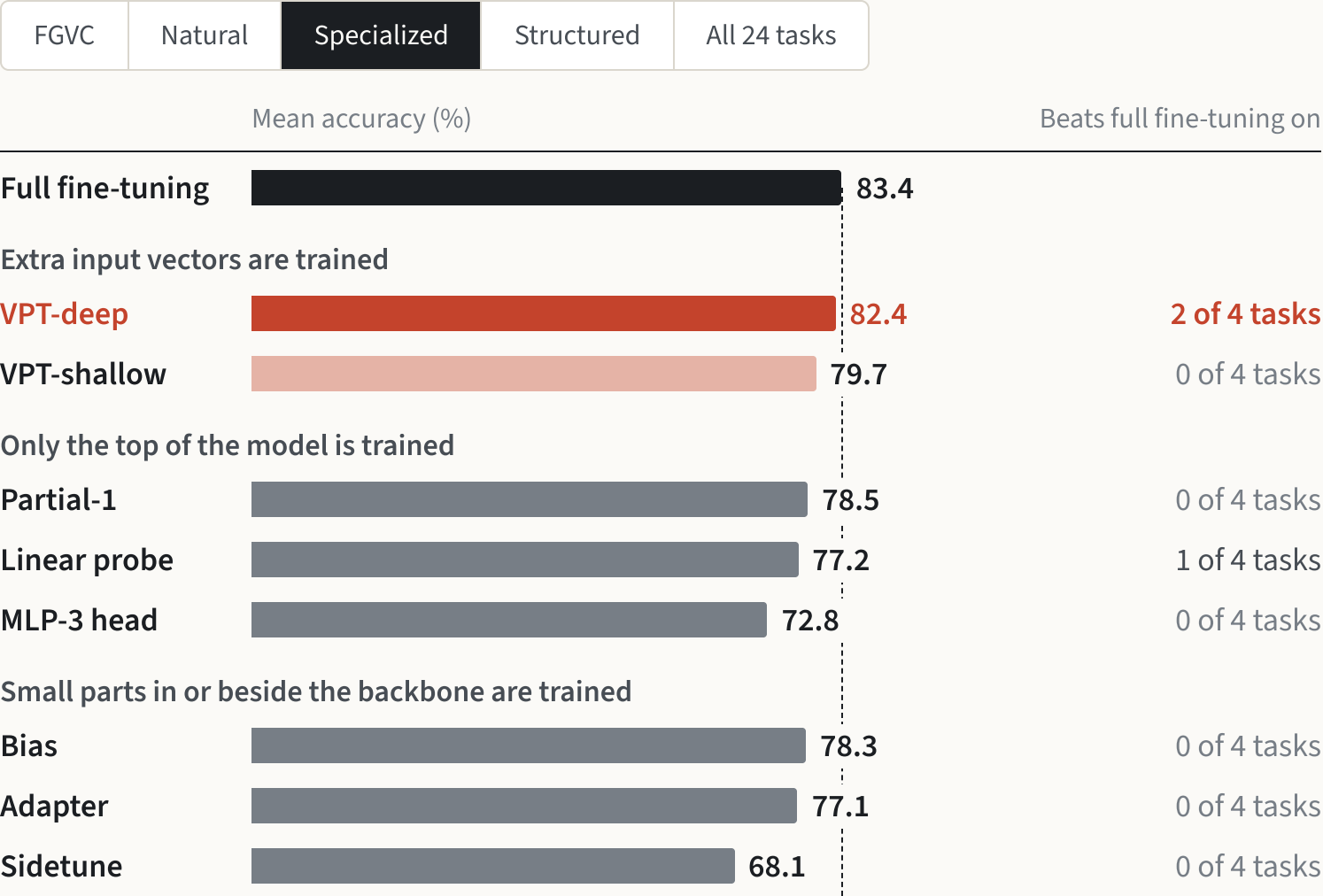
VPT-deep beats full fine-tuning on 20 of 24 tasks



These seven tasks use ordinary photos of objects, pets, textures, scenes and house numbers, close to what the model saw in pretraining, but with only 1,000 training images each.

With so little data, retraining all 86 million weights overfits, so even VPT-shallow beats full fine-tuning here.

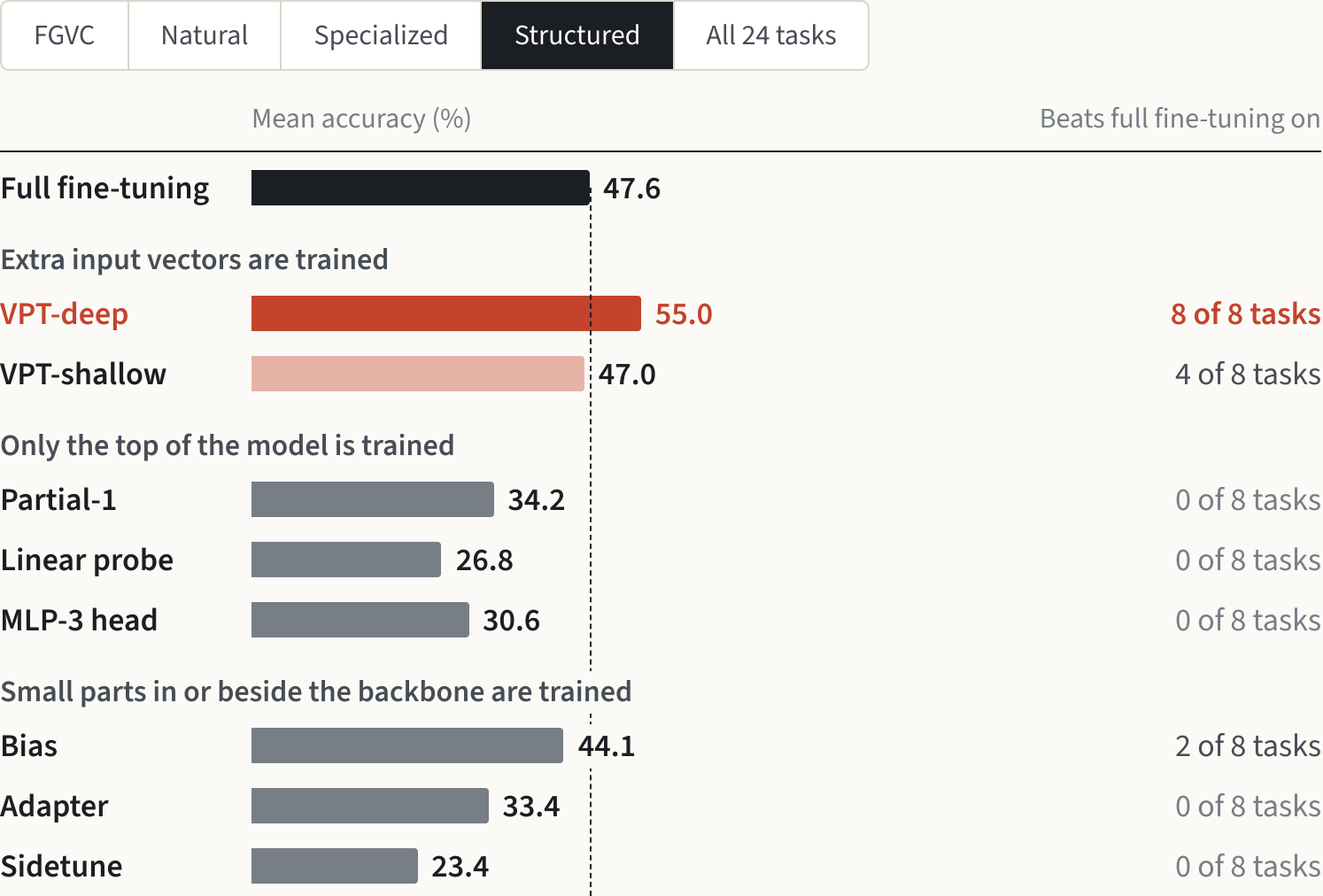
VPT-deep beats full fine-tuning on 20 of 24 tasks



These four tasks come from special cameras: satellite images of land use, microscope slides of tumour tissue and photos of the retina, with 1,000 training images each.

This is the one group where full fine-tuning stays ahead, by one point, but VPT-deep is still the best of the cheap methods.

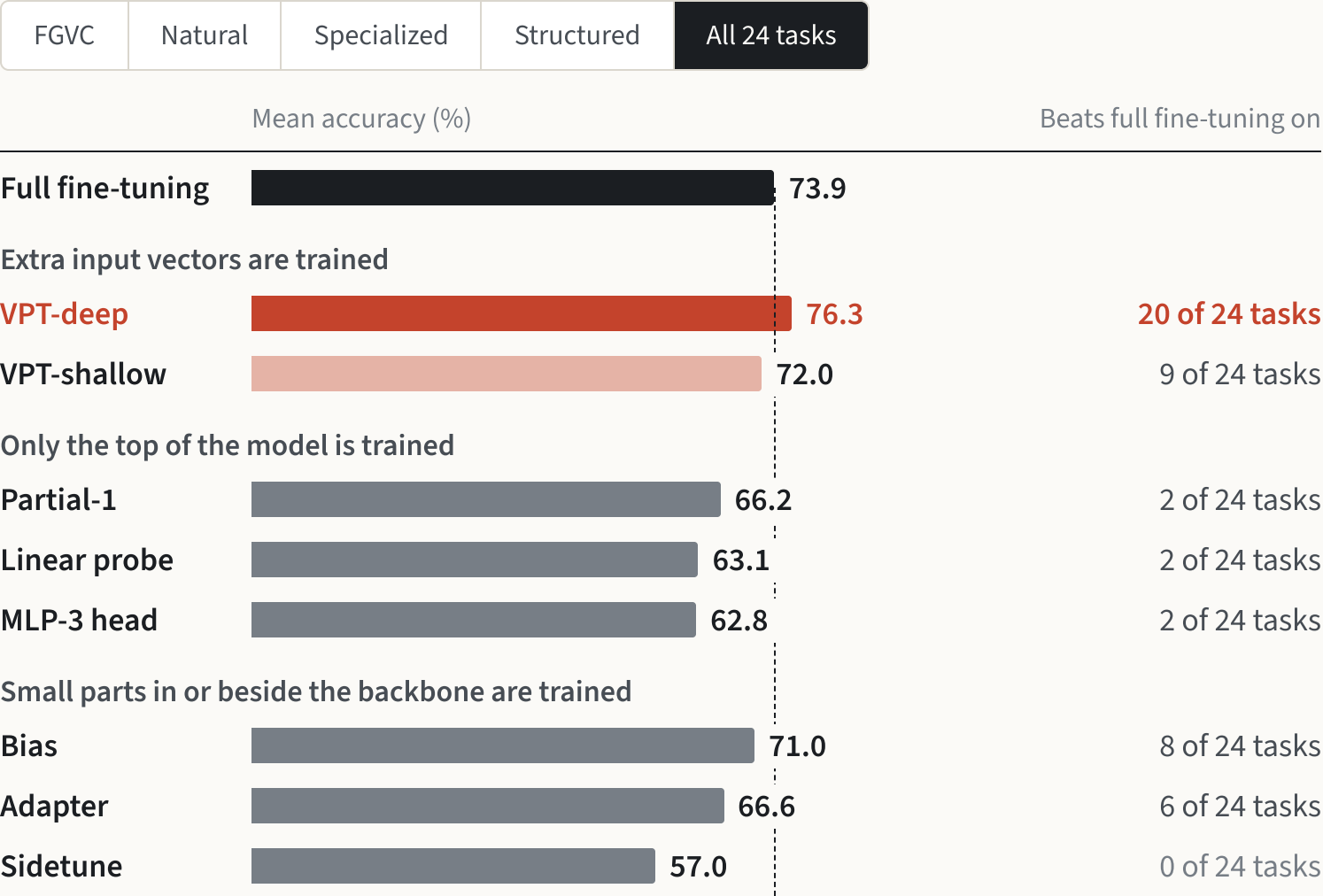
VPT-deep beats full fine-tuning on 20 of 24 tasks



These eight tasks ask about geometry rather than objects: how many things there are, how far away, at what angle. Most are synthetic scenes, far from ImageNet.

Here VPT-deep wins all eight, 55.0 against 47.6, while the linear probe falls to 26.8, so the frozen features alone do not hold this information.

VPT-deep beats full fine-tuning on 20 of 24 tasks



Averaged over all 24 tasks, VPT-deep is the only cheap method above full fine-tuning, 76.3 against 73.9.

It gets there while storing 1.18 backbones' worth of weights for the 24 tasks, where full fine-tuning stores 24.

VPT leads with little data and stays ahead with more

VPT-deep against

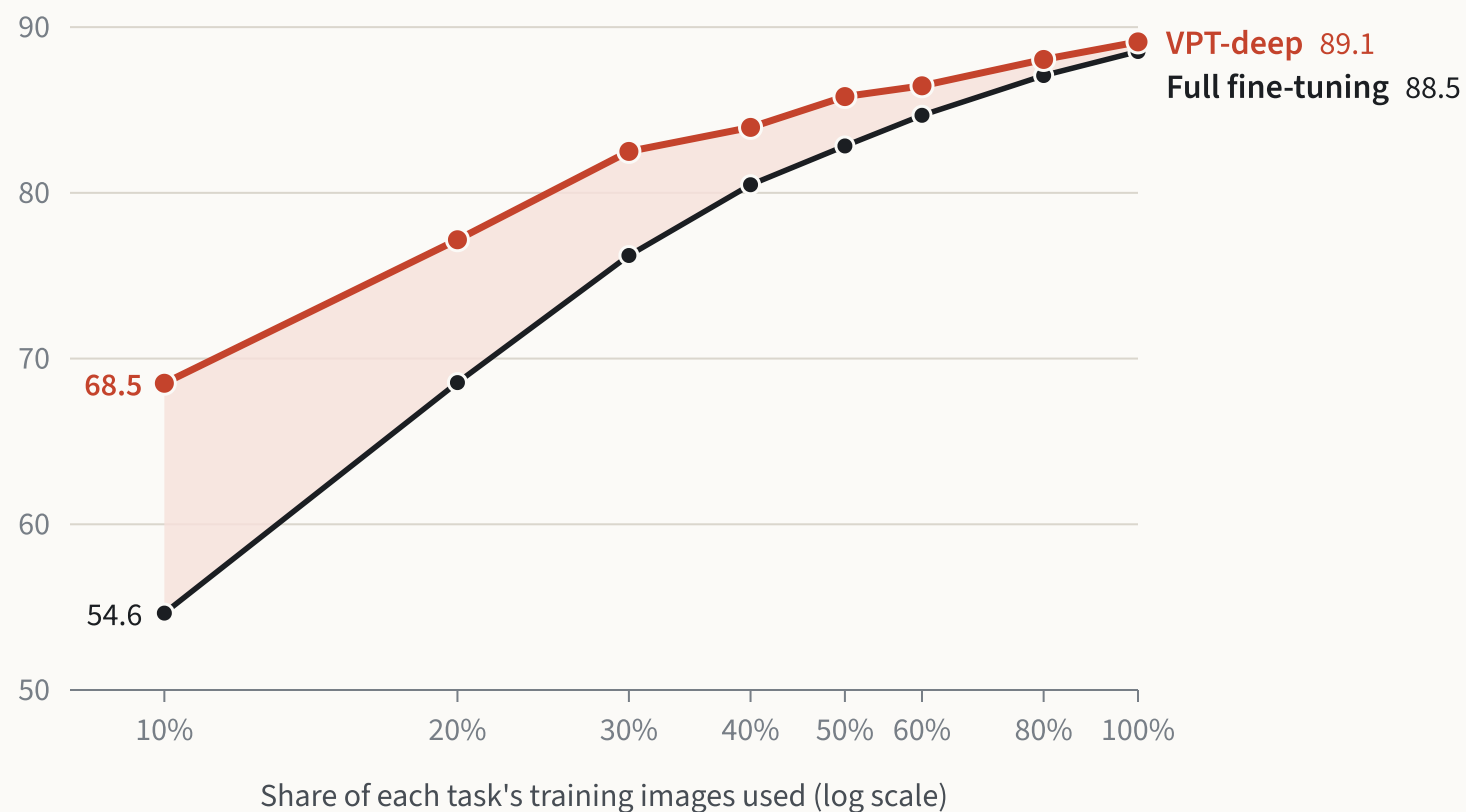
Full fine-tuning

Linear probe

Adapter

Bias

Mean test accuracy over the five FGVC tasks (%)

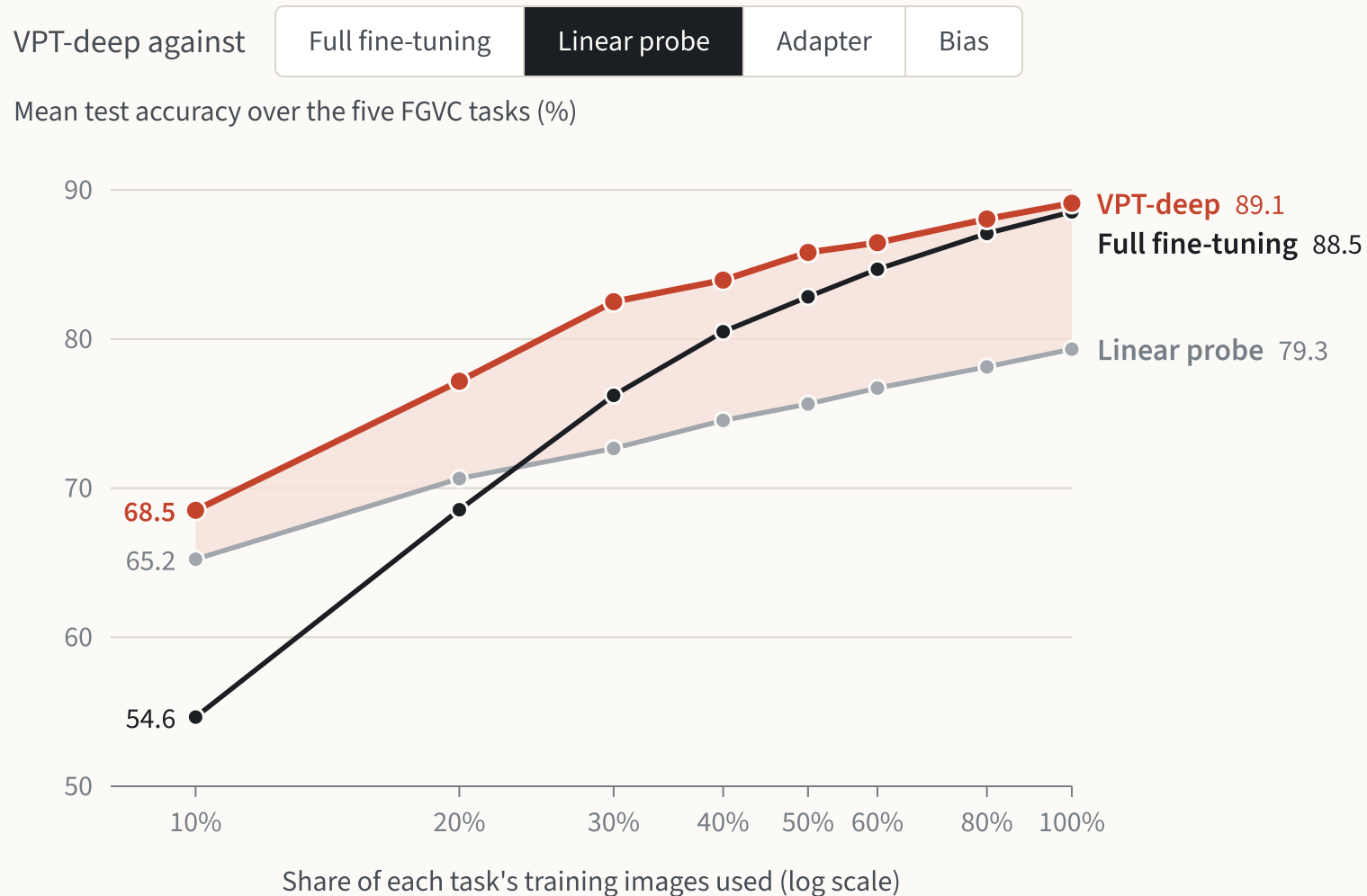


With a tenth of the training images, full fine-tuning has too many weights to fit and falls 14 points behind VPT-deep.

The gap shrinks as the data grows, but VPT-deep stays ahead all the way to the full training set.

Jia et al. 2022, Fig. 3: ViT-B/16, mean test accuracy over the five FGVC tasks, each method trained on a share of every task's training images. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the 100% points match Table 1.

VPT leads with little data and stays ahead with more

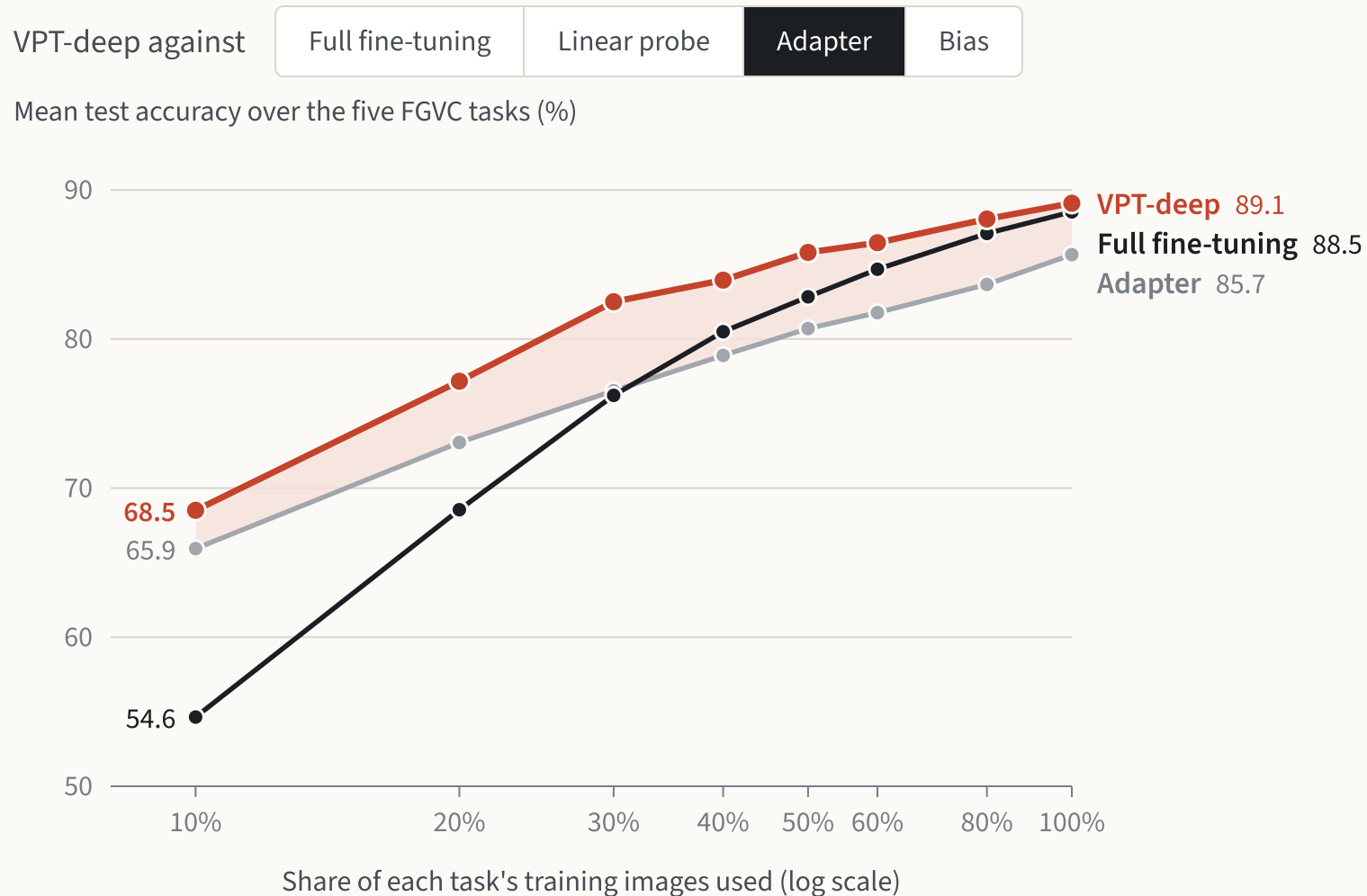


The linear probe trains only a new output layer, so it does well when data is scarce.

From 30% of the data on, full fine-tuning overtakes it, and with all the data it trails VPT-deep by almost 10 points.

Jia et al. 2022, Fig. 3: ViT-B/16, mean test accuracy over the five FGVC tasks, each method trained on a share of every task's training images. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the 100% points match Table 1.

VPT leads with little data and stays ahead with more

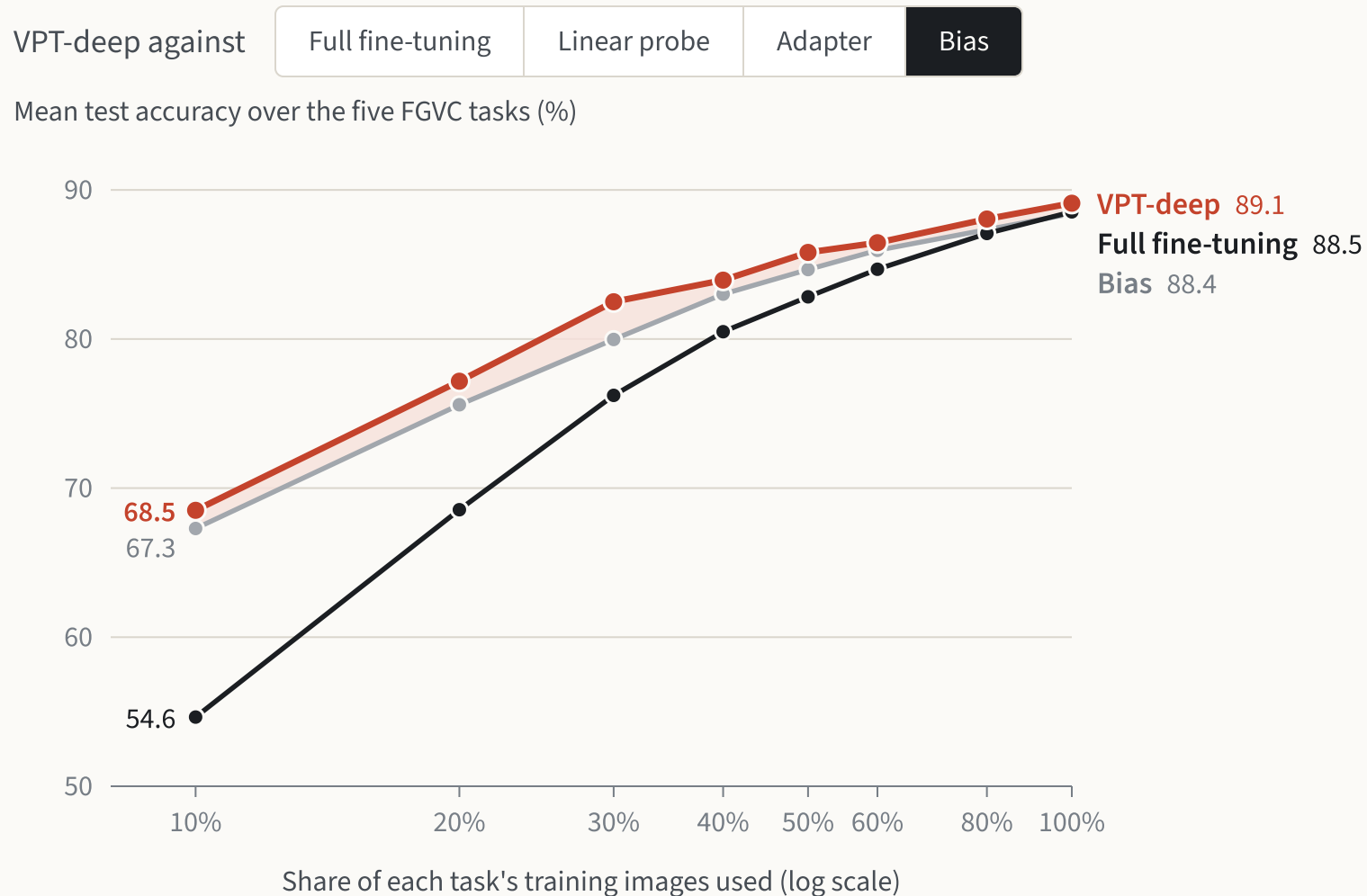


The adapter inserts small trained blocks into every layer, and it too beats full fine-tuning on little data.

Full fine-tuning passes it at 40% of the data, while VPT-deep stays 3 to 6 points above it throughout.

Jia et al. 2022, Fig. 3: ViT-B/16, mean test accuracy over the five FGVC tasks, each method trained on a share of every task's training images. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the 100% points match Table 1.

VPT leads with little data and stays ahead with more

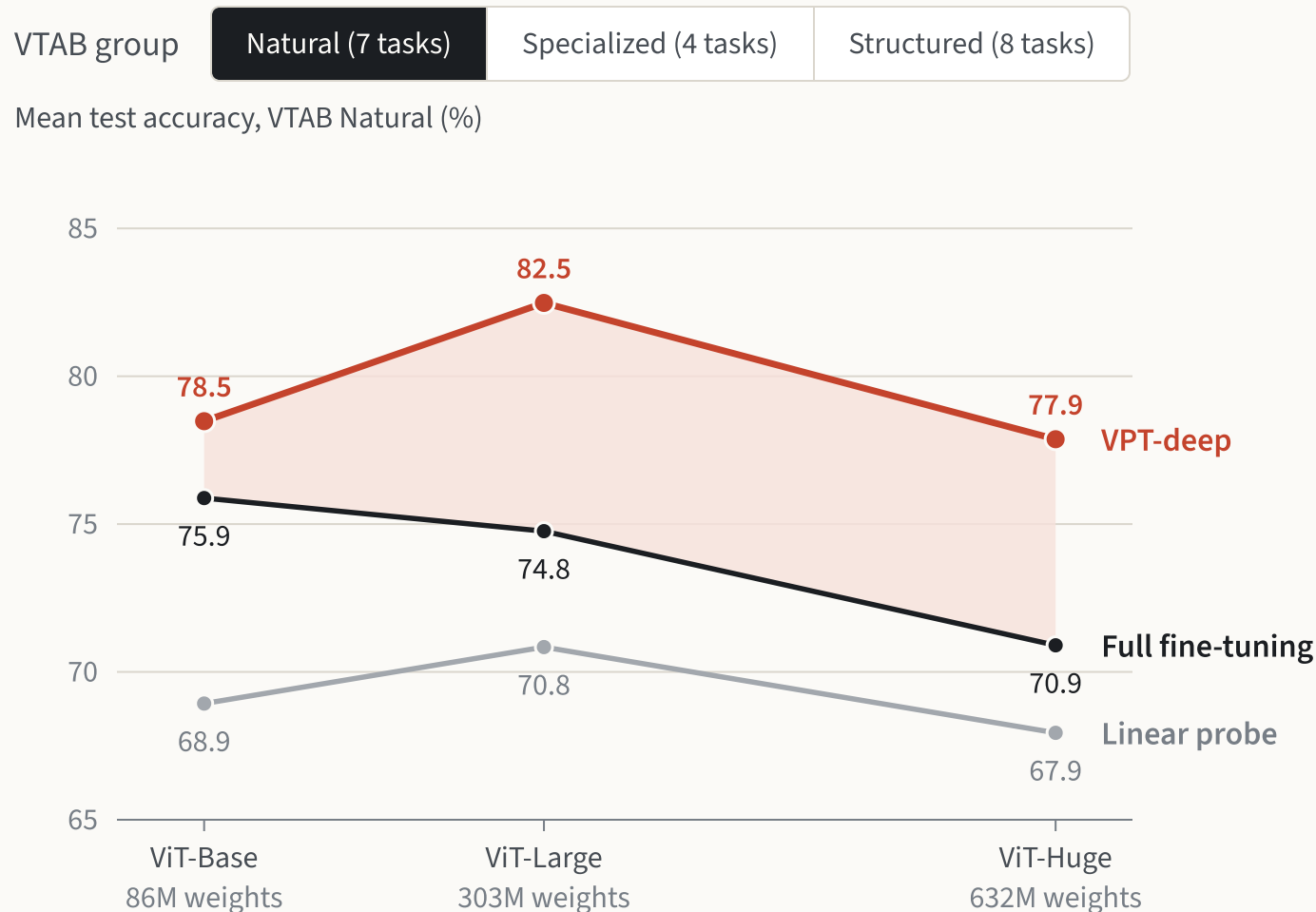


Bias trains only the bias terms and is the closest rival: it beats full fine-tuning up to 80% of the data.

VPT-deep is still ahead at every point, by half a point to 2.5 points.

Jia et al. 2022, Fig. 3: ViT-B/16, mean test accuracy over the five FGVC tasks, each method trained on a share of every task's training images. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the 100% points match Table 1.

The result holds for larger backbones too

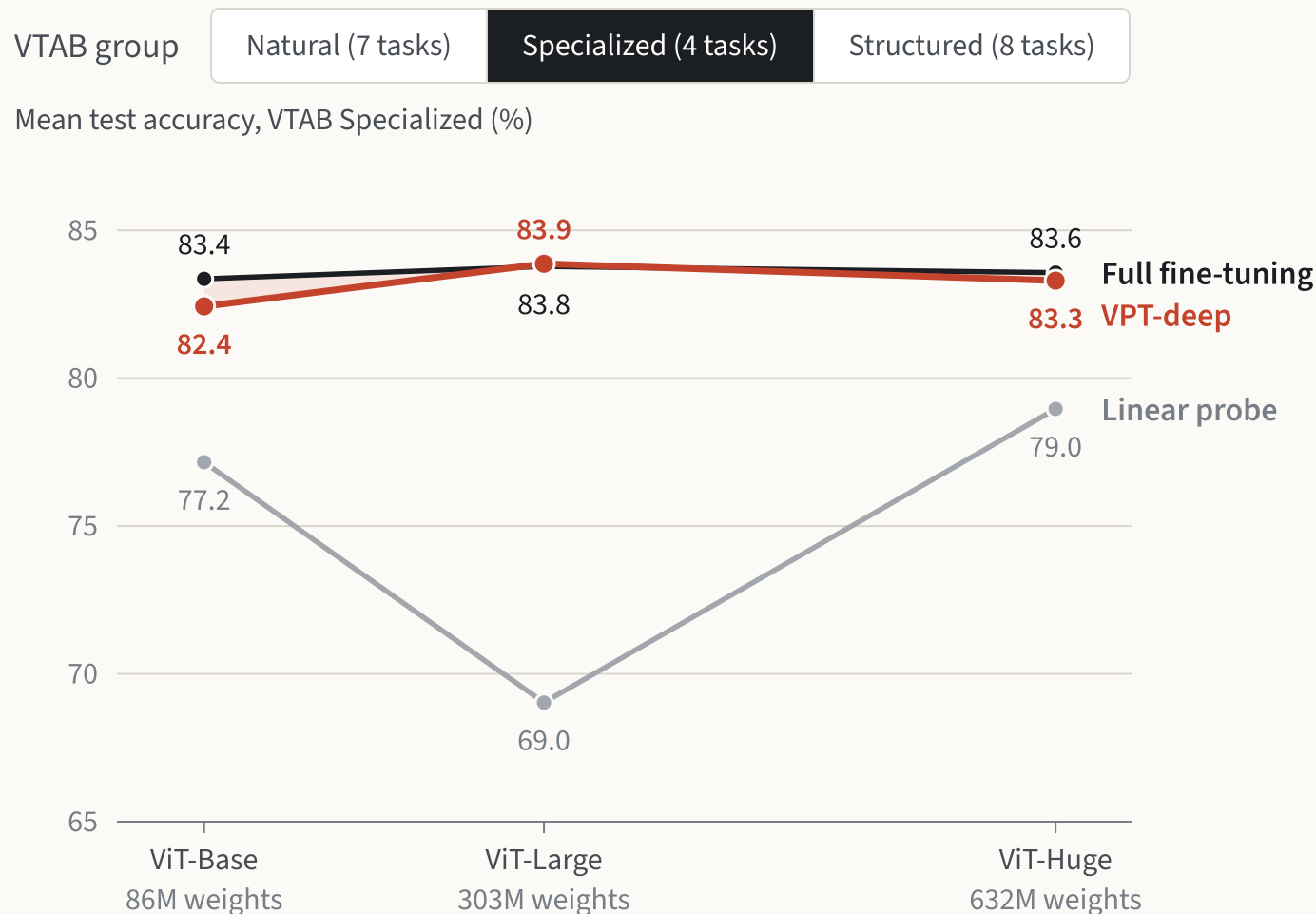


On ordinary photos, VPT-deep beats full fine-tuning with all three backbones: by 2.6 points on ViT-Base and by 7 to 8 points on the larger two.

Full fine-tuning even gets worse as the model grows, from 75.9 to 70.9.

Jia et al. 2022, Fig. 4: VTAB-1k, mean test accuracy per group, backbones pretrained on ImageNet-21k. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the ViT-B points match Table 1.

The result holds for larger backbones too

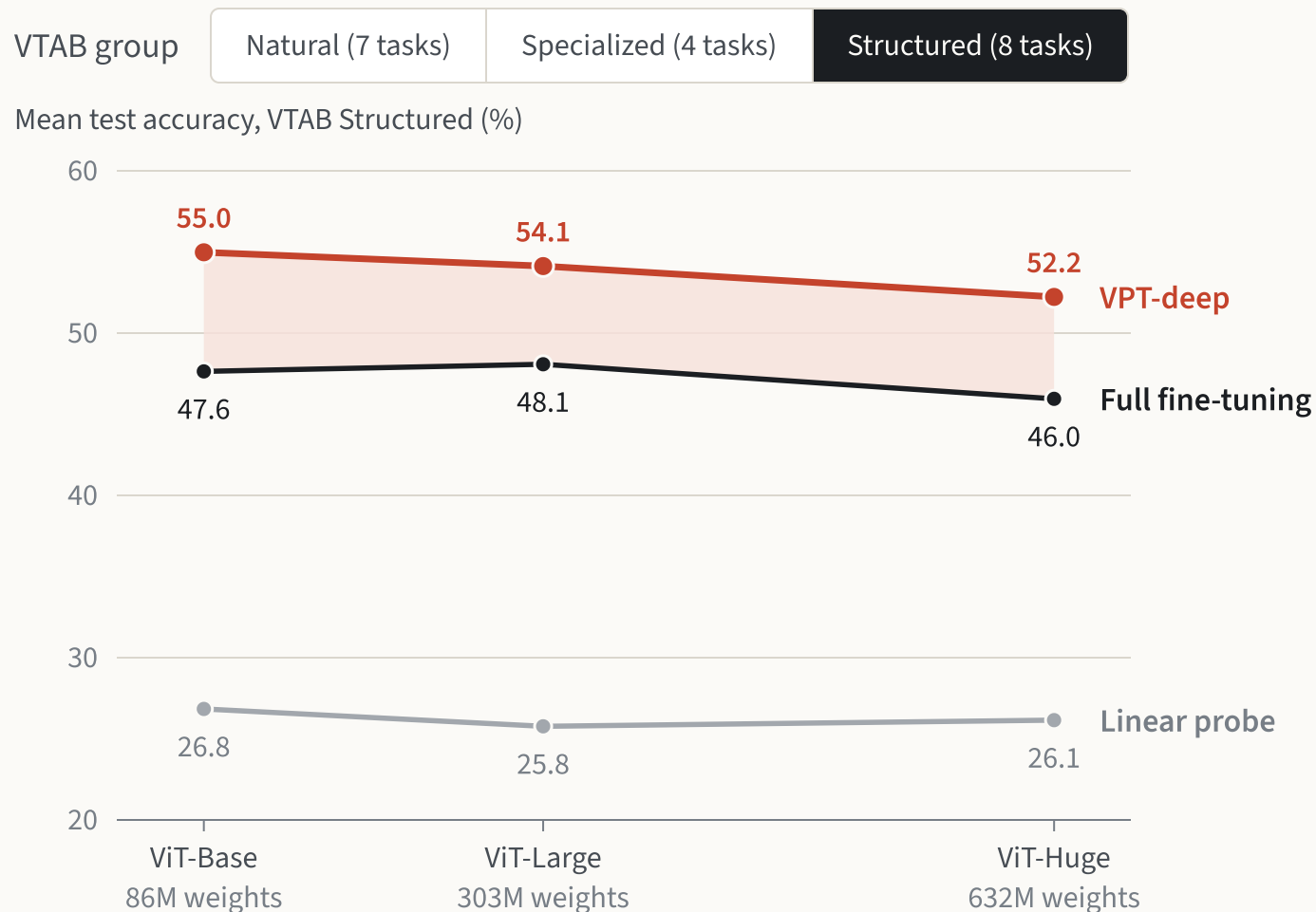


On medical and satellite images, the two are level with every backbone and never differ by more than one point.

This was already the one group where full fine-tuning kept a small lead on ViT-Base, and larger backbones do not change that.

Jia et al. 2022, Fig. 4: VTAB-1k, mean test accuracy per group, backbones pretrained on ImageNet-21k. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the ViT-B points match Table 1.

The result holds for larger backbones too

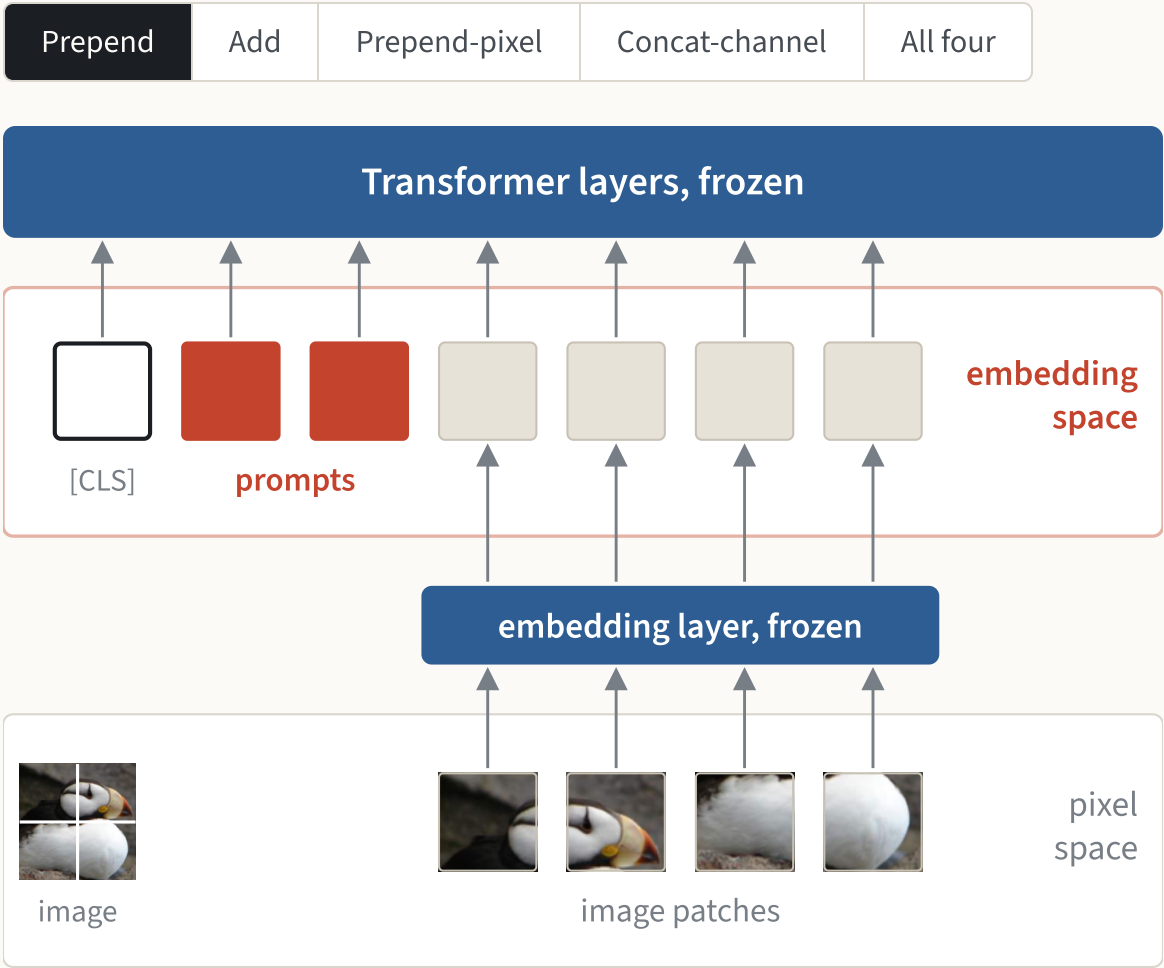


On counting and distance tasks, VPT-deep leads by 6 to 7 points with every backbone.

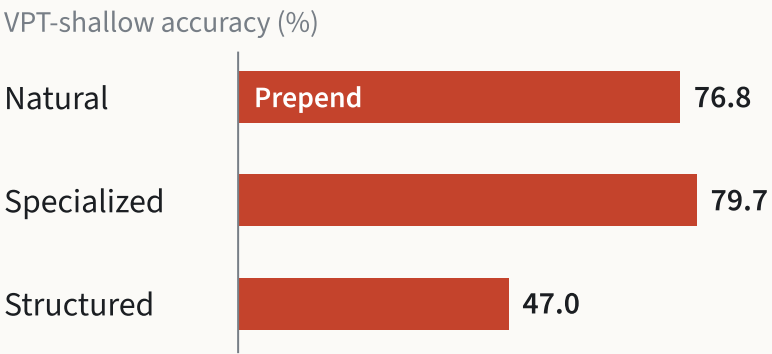
The linear probe stays near 26%, so the frozen features alone are not enough here, at any size.

Jia et al. 2022, Fig. 4: VTAB-1k, mean test accuracy per group, backbones pretrained on ImageNet-21k. The paper gives these values only as a figure; they were read from its vector drawing in the PDF, and the ViT-B points match Table 1.

Four places to put the prompt

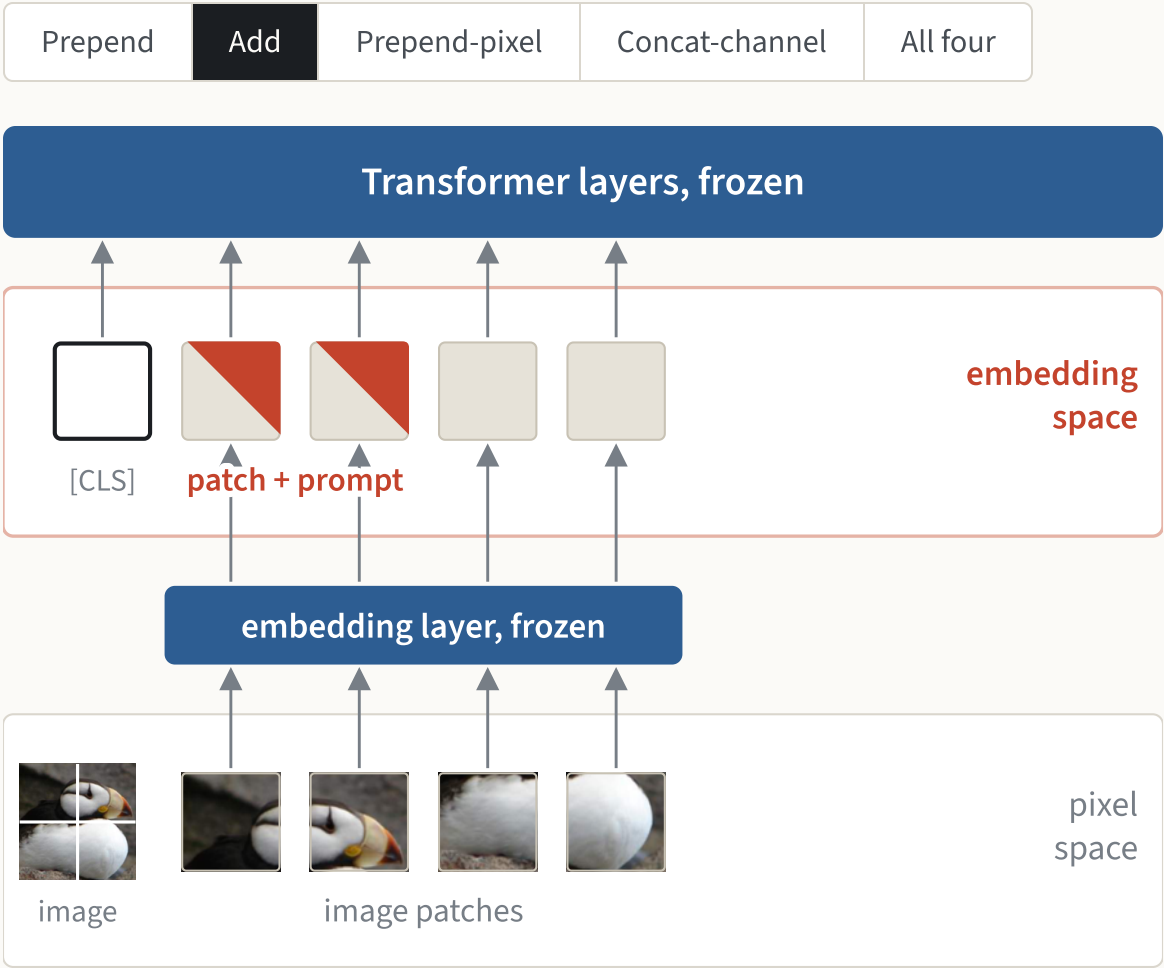


This is the default. The prompts are extra vectors placed next to the patch vectors, after the embedding layer, so every layer can attend to them as tokens of their own.

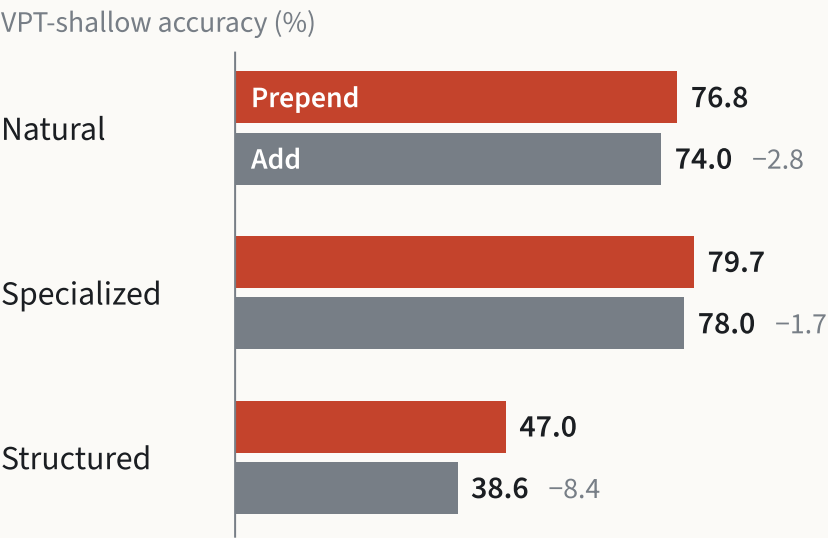


Numbers from Jia et al. 2022, Fig. 5 (ViT-B/16, prompts in the first layer only); "All four" is the mean of the three groups. Diagrams redrawn.

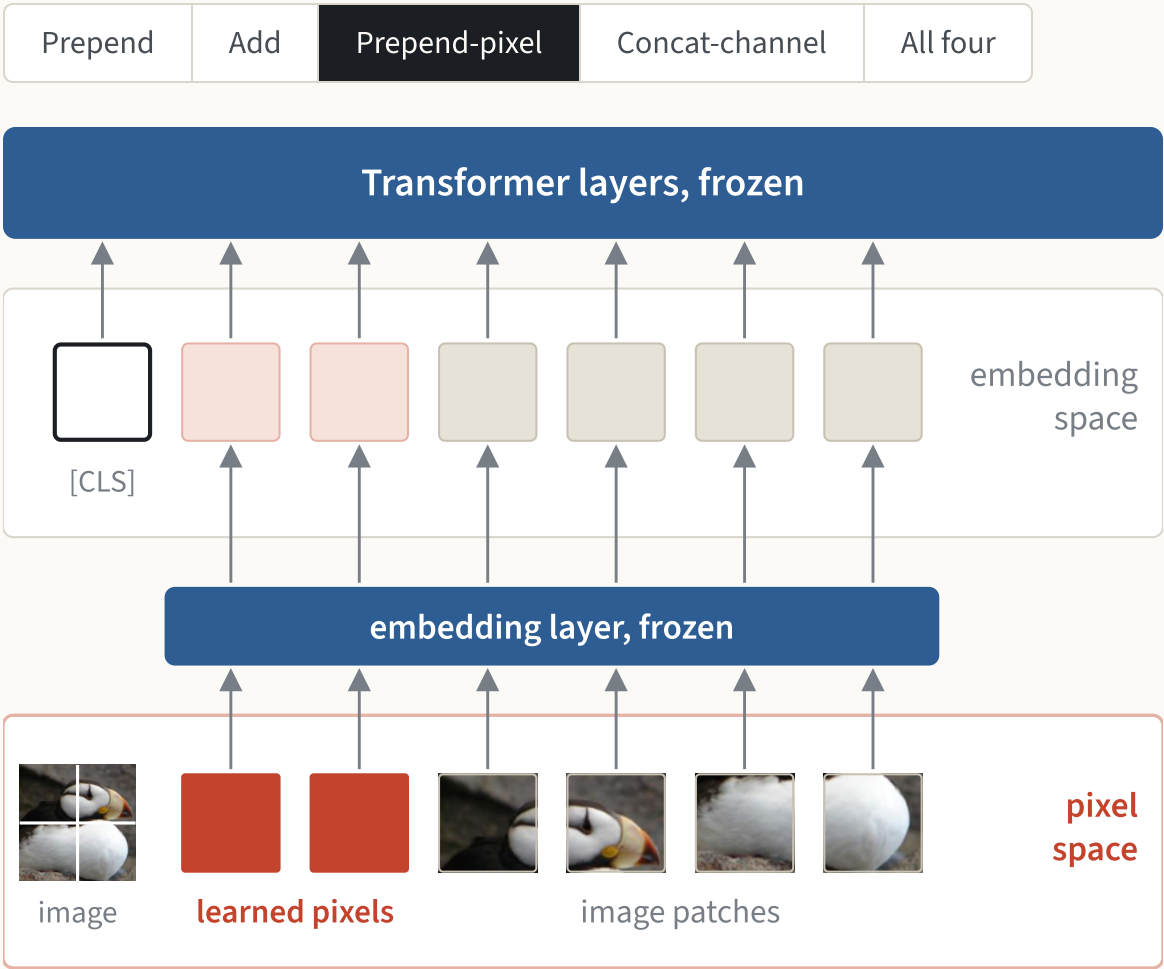
Four places to put the prompt



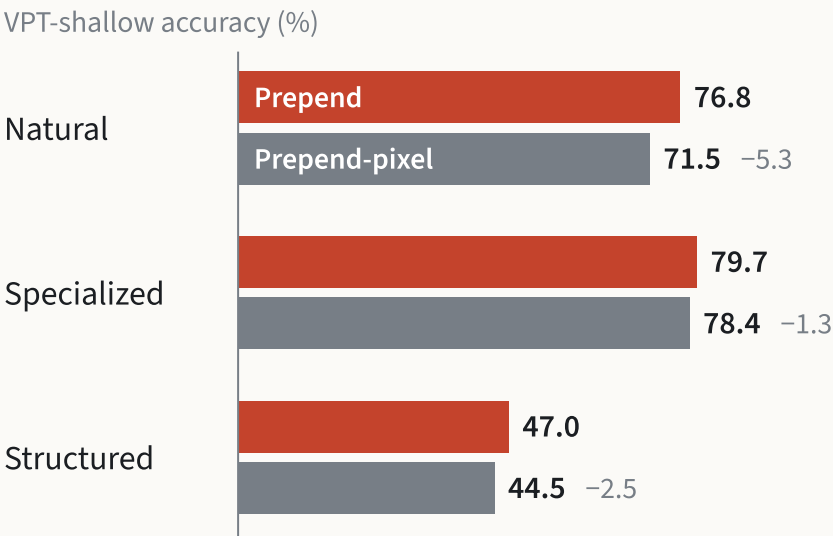
Here each prompt vector is added onto a patch vector, so the sequence keeps its length. This scores lower in all three groups, by 8.4 points on Structured.



Four places to put the prompt

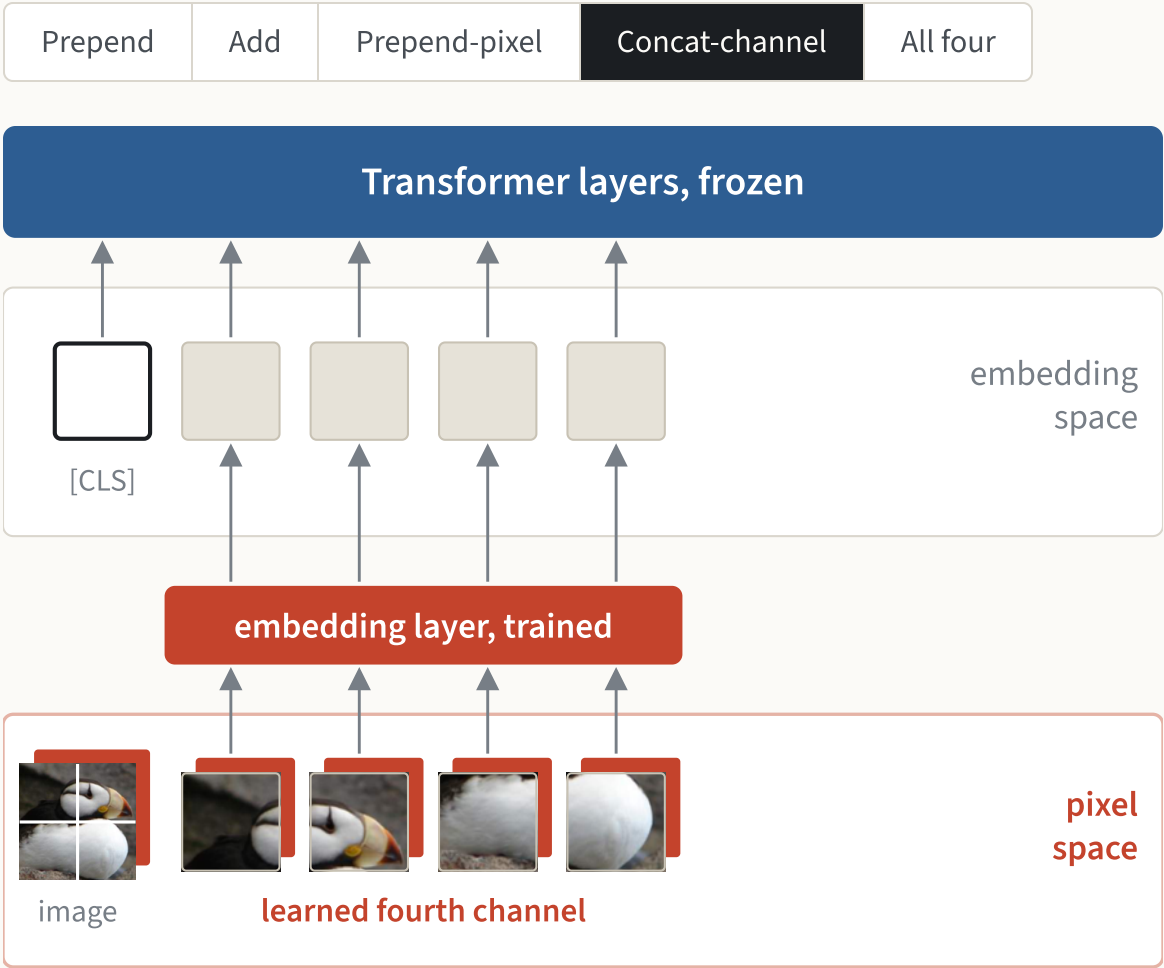


Here the prompts are learned pixel patches, placed next to the image patches before the frozen embedding layer. This also scores lower in every group, by 5.3 points on Natural.

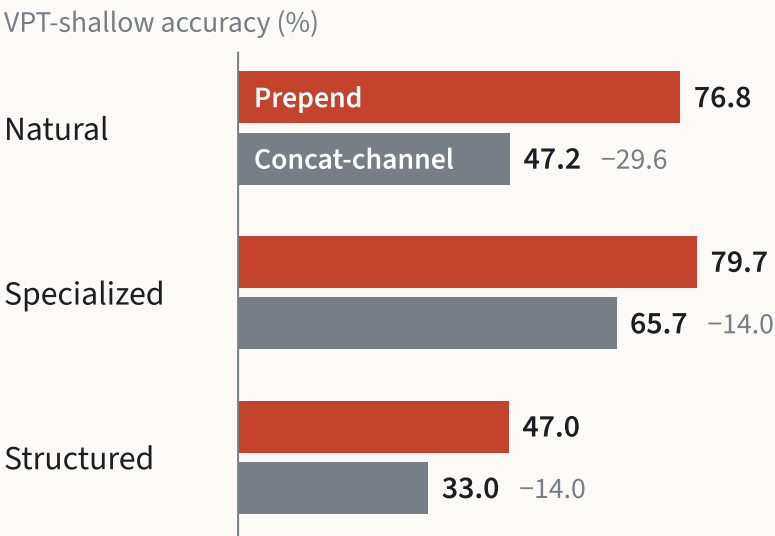


Numbers from Jia et al. 2022, Fig. 5 (ViT-B/16, prompts in the first layer only); "All four" is the mean of the three groups. Diagrams redrawn.

Four places to put the prompt



Here the prompt is a learned fourth colour channel on the image, and the embedding layer is trained to read it. This is by far the worst: 29.6 points lower on Natural, below even a linear probe there.



Four places to put the prompt

Prepend

Add

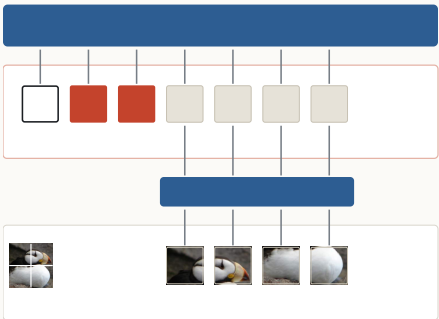
Prepend-pixel

Concat-channel

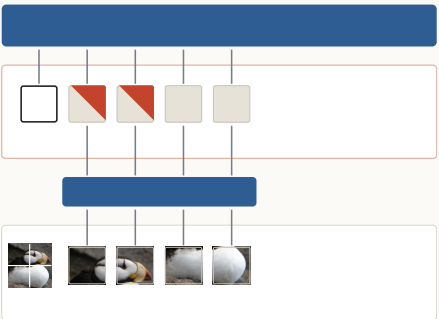
All four

Prompt in the embedding space

Prepend

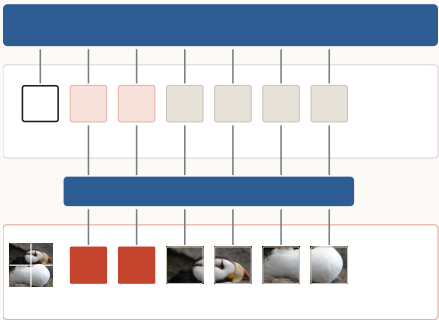


Add

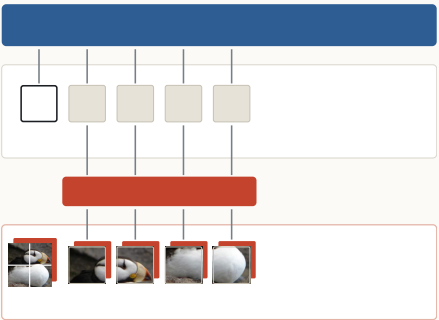


Prompt in pixel space

Prepend-pixel

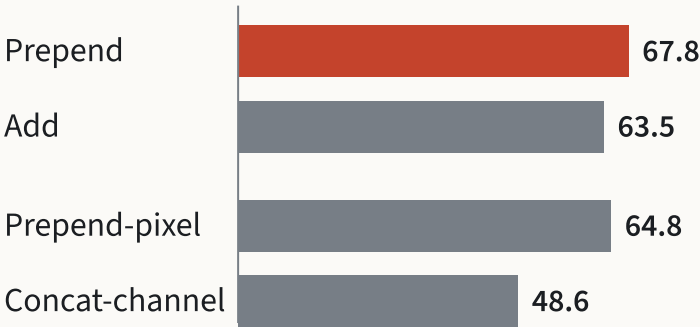


Concat-channel



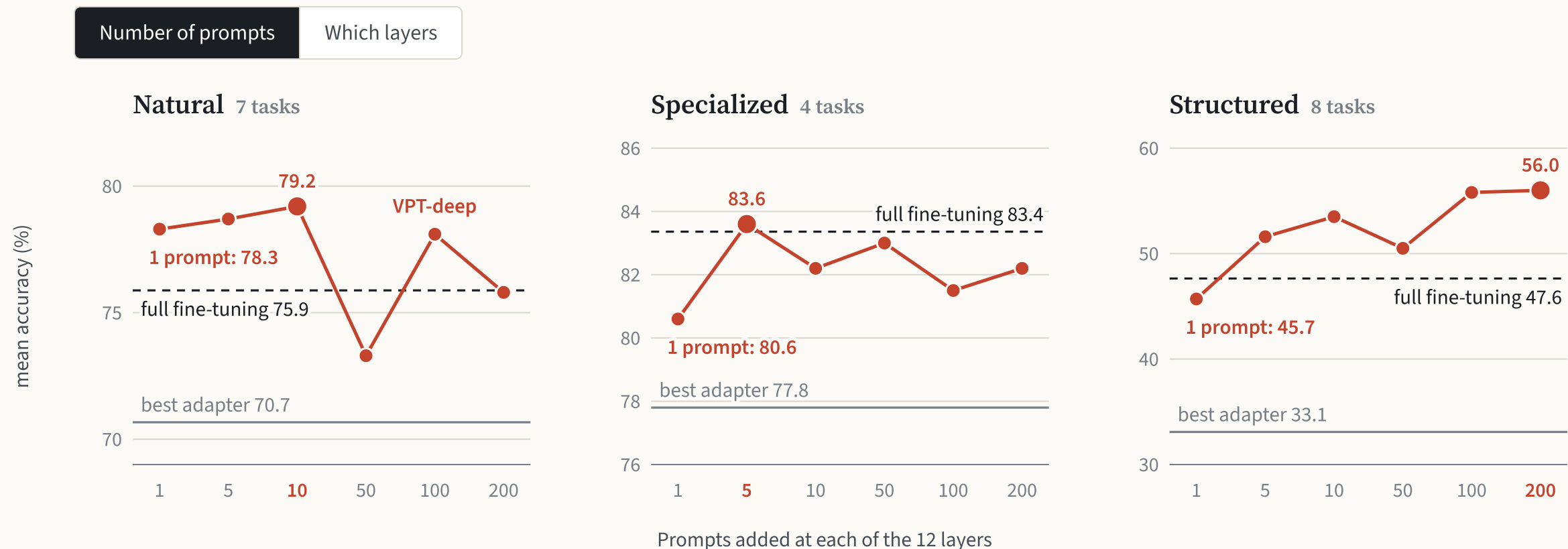
Prepend scores highest in every group. A prompt works best as separate vectors in the embedding space; adding it onto the patches or moving it into the pixels costs accuracy.

VPT-shallow accuracy (%), mean of the three groups



Numbers from Jia et al. 2022, Fig. 5 (ViT-B/16, prompts in the first layer only); "All four" is the mean of the three groups. Diagrams redrawn.

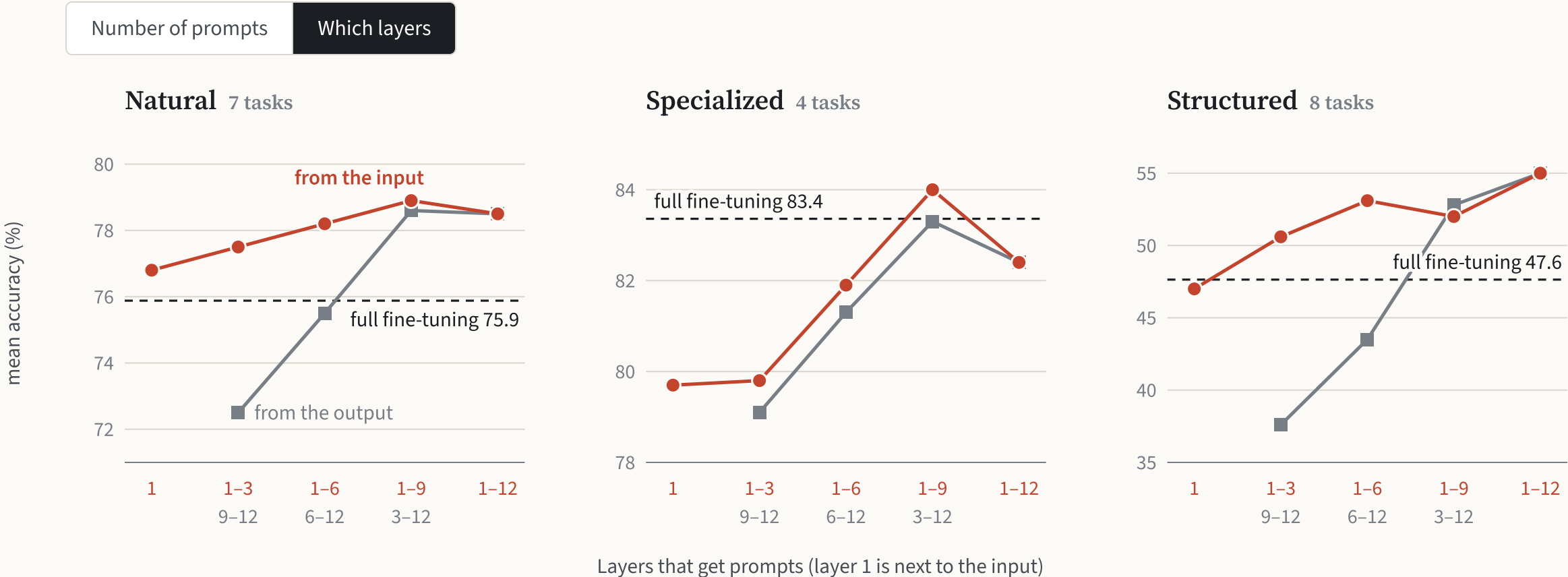
How many prompts, and at which layers



Each group prefers a different number: Natural does best with 10 prompts per layer, Specialized with 5 and Structured with 200. Even a single prompt per layer beats the best adapter in every group, and on Natural it already beats full fine-tuning.

Jia et al. 2022, Fig. 6: VPT-deep on ViT-B/16, mean test accuracy per VTAB group. The paper gives these only as a figure; values are read off its plotted points. Full fine-tuning and the adapter (best of three sizes): Table 13.

How many prompts, and at which layers



Prompting more layers generally helps: layer 1 alone is VPT-shallow, all twelve is VPT-deep. Layers next to the input matter most: prompts placed only in the layers near the output score lower, the opposite of what was found for text prompts.

Jia et al. 2022, Fig. 7, each variant with its best prompt length. Values are read off the figure's plotted points; layer 1 alone and layers 1-12 match VPT-shallow and VPT-deep in Table 13.

With prompts, the same frozen network separates the classes

House numbers (SVHN)

Satellite land use (EuroSAT)

Counting objects (CLEVR)

Linear probe

36.6%

frozen network, no prompts



VPT-deep

78.1%

same frozen network, with prompts



Full fine-tuning

87.4%

every weight retrained



Each dot is one test image, coloured by its true class; the number is test accuracy. With the frozen features alone, the classes overlap. With prompts, the same frozen network produces clearly separated groups, so a simple linear head can tell them apart.

With prompts, the same frozen network separates the classes

House numbers (SVHN)

Satellite land use (EuroSAT)

Counting objects (CLEVR)

Linear probe

87.5%

frozen network, no prompts



VPT-deep

96.1%

same frozen network, with prompts



Full fine-tuning

95.7%

every weight retrained



Each dot is one test image, coloured by its true class; the number is test accuracy. With the frozen features alone, the classes overlap. With prompts, the same frozen network produces clearly separated groups, so a simple linear head can tell them apart.

With prompts, the same frozen network separates the classes

House numbers (SVHN)

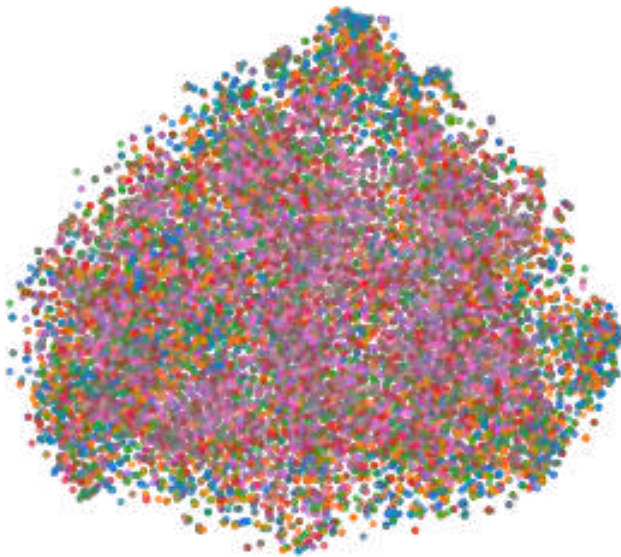
Satellite land use (EuroSAT)

Counting objects (CLEVR)

Linear probe

34.3%

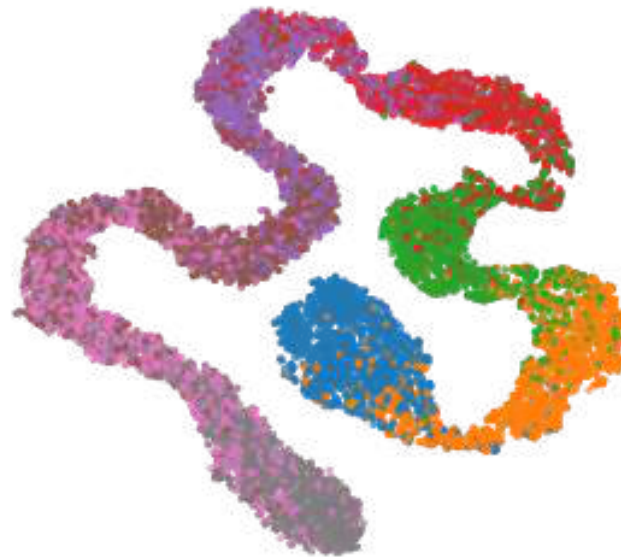
frozen network, no prompts



VPT-deep

68.5%

same frozen network, with prompts



Full fine-tuning

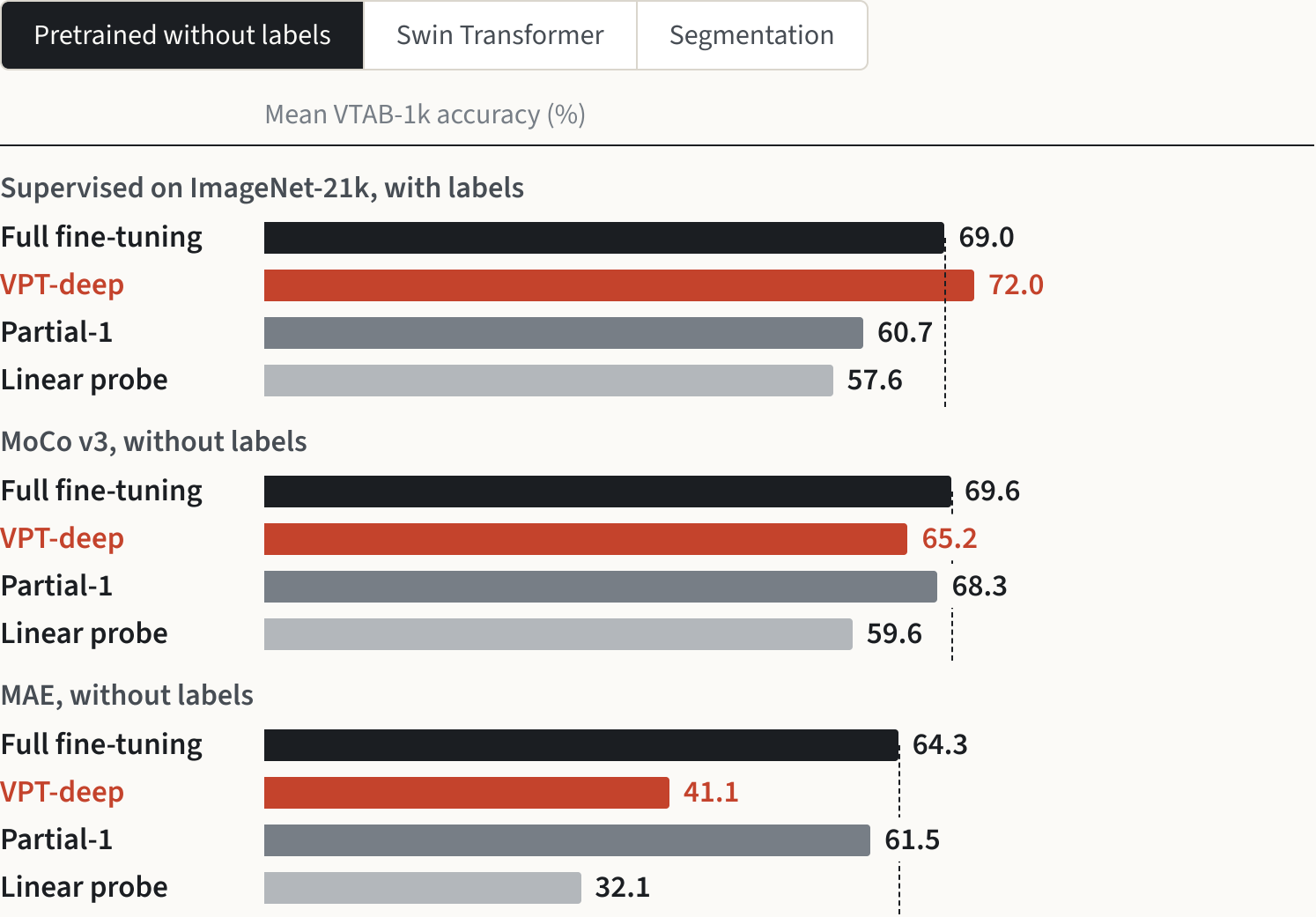
56.3%

every weight retrained



Each dot is one test image, coloured by its true class; the number is test accuracy. With the frozen features alone, the classes overlap. With prompts, the same frozen network produces clearly separated groups, so a simple linear head can tell them apart.

Where VPT falls short

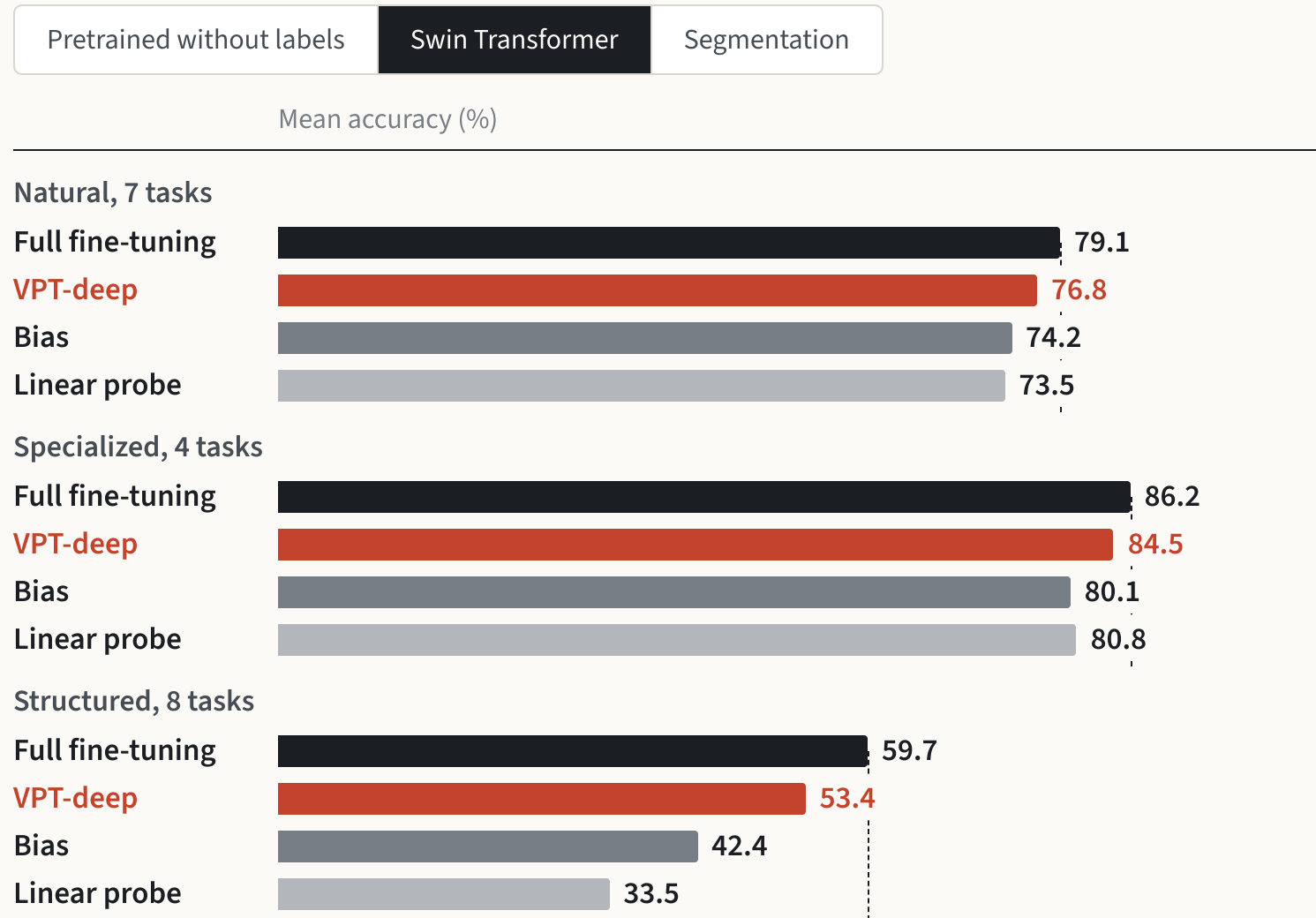


Everything so far used a ViT pretrained with labels. The paper also tries the same ViT-B pretrained without labels, and there VPT loses its lead.

With MoCo v3 it ends four points behind full fine-tuning. With MAE it drops towards the linear probe, while Partial-1, which retrain only the last block, stays close to full fine-tuning. The authors leave the reason open.

Jia et al. 2022, Table 1 (with labels) and Table 4 (MoCo v3, MAE): ViT-B/16, mean of the three VTAB-1k group averages.

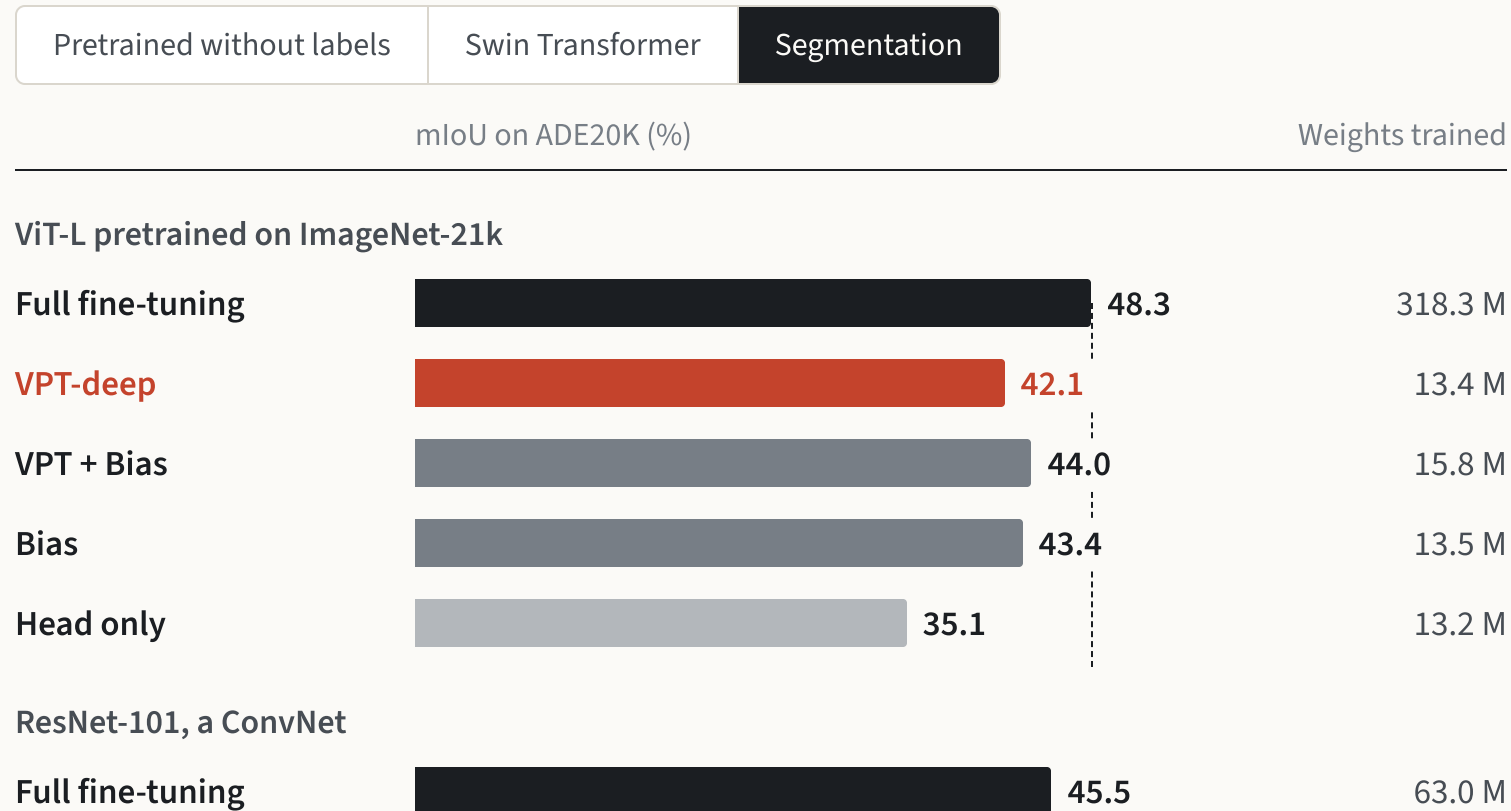
Where VPT falls short



Swin is a vision Transformer that looks at small windows of the image and merges patches as it goes deeper.

On Swin, VPT still beats the other cheap methods. But full fine-tuning is ahead of VPT-deep in all three groups, by 6.3 points on Structured.

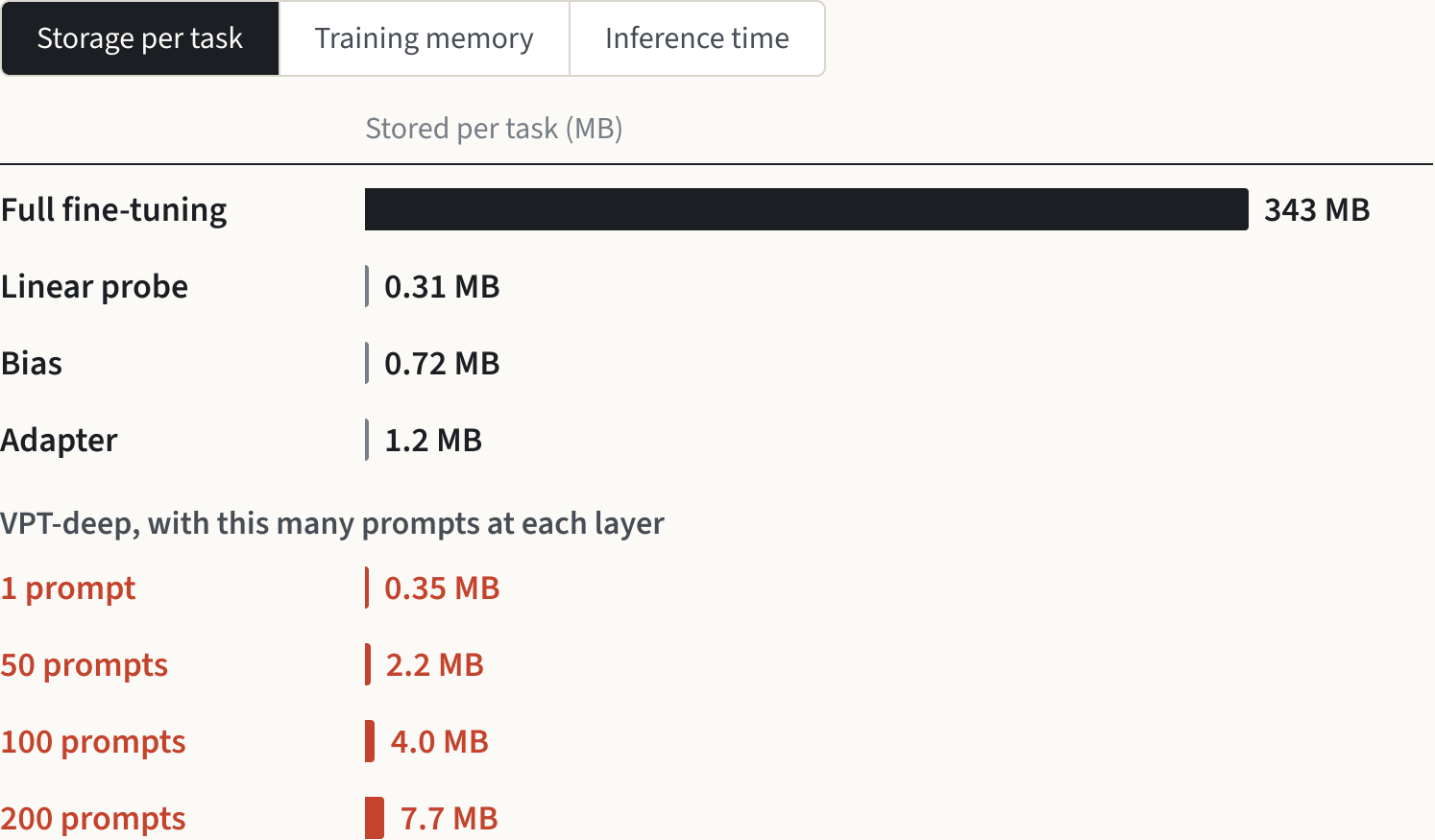
Where VPT falls short



Segmentation labels every pixel of a scene. The score, mIoU, measures how well the predicted regions overlap the true ones.

Here VPT-deep reaches 42.1 against 48.3 for full fine-tuning, about the same as tuning only the biases. Prompts and biases together come within 1.5 points of a fully fine-tuned ResNet-101.

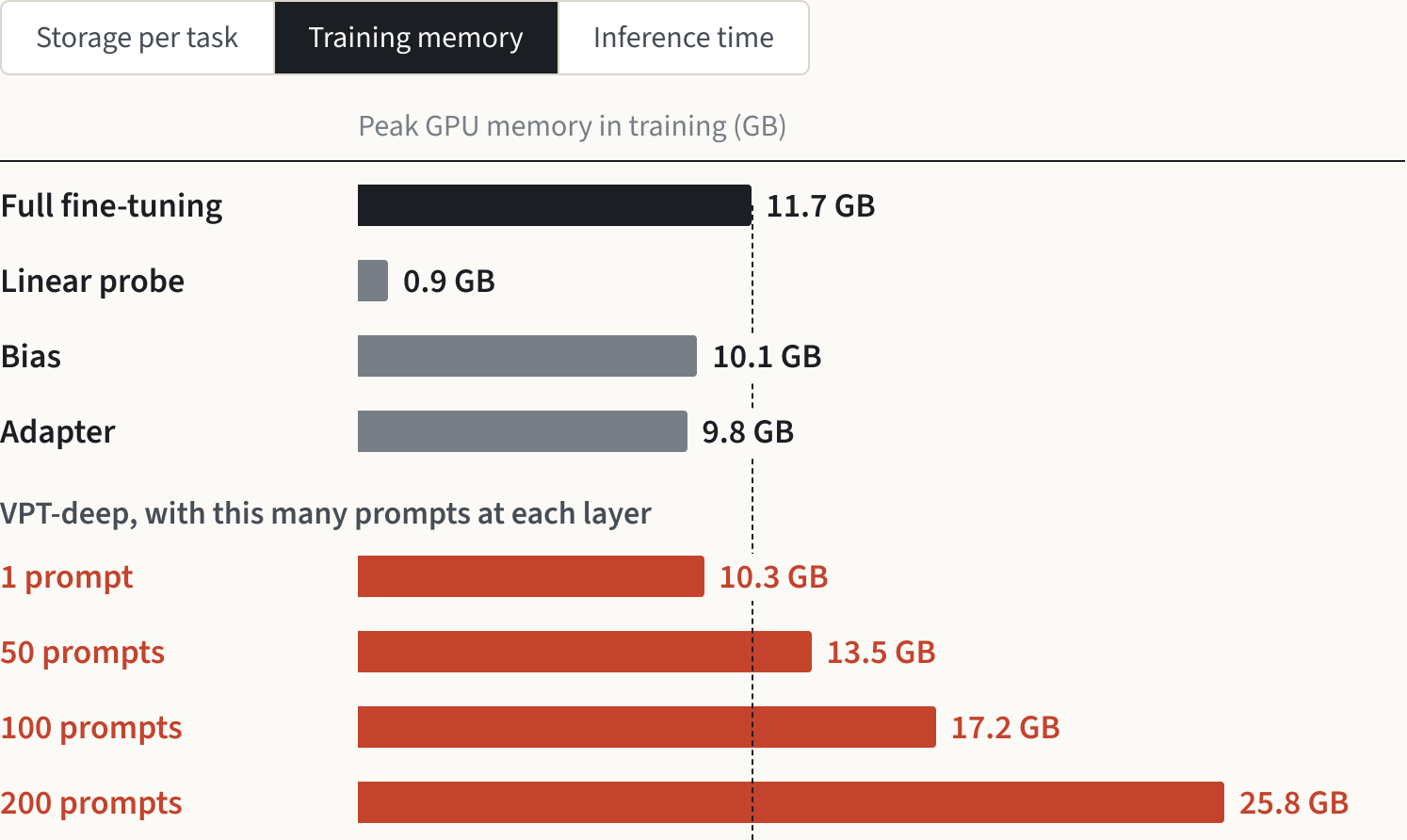
VPT saves storage, not training memory or inference time



A task trained with VPT-deep keeps only its prompts and its head: 0.35 MB with one prompt per layer, 7.7 MB with 200. A fully fine-tuned copy is 343 MB for every task.

All tasks share one frozen backbone, so this is the real saving, and it grows with every task.

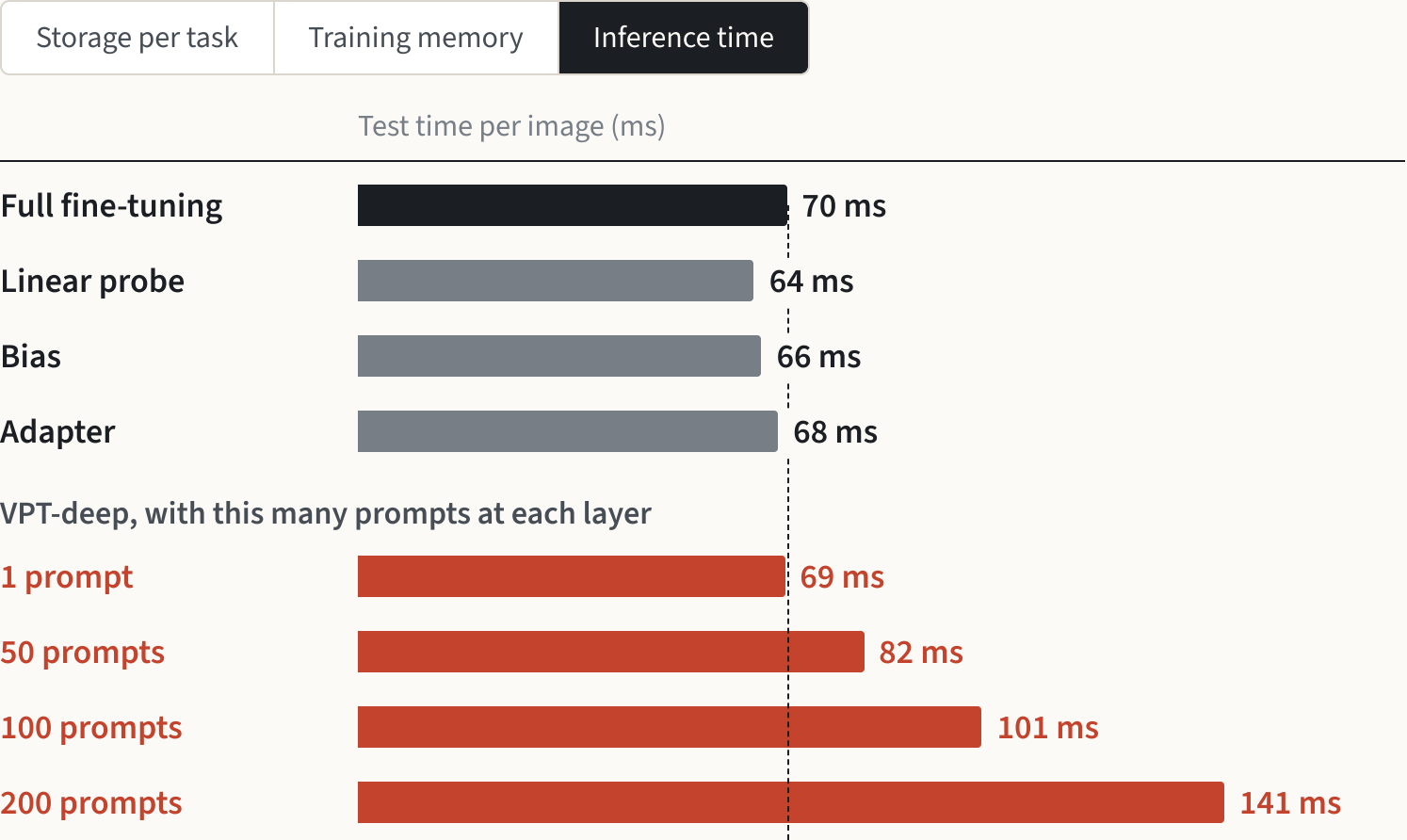
VPT saves storage, not training memory or inference time



In training, VPT-deep needs about as much GPU memory as full fine-tuning, and with 200 prompts more than twice as much.

To train the prompts in the first layer, the gradient has to travel back through all twelve frozen layers, so their activations must be kept, as in full fine-tuning. Only the linear probe avoids this.

VPT saves storage, not training memory or inference time

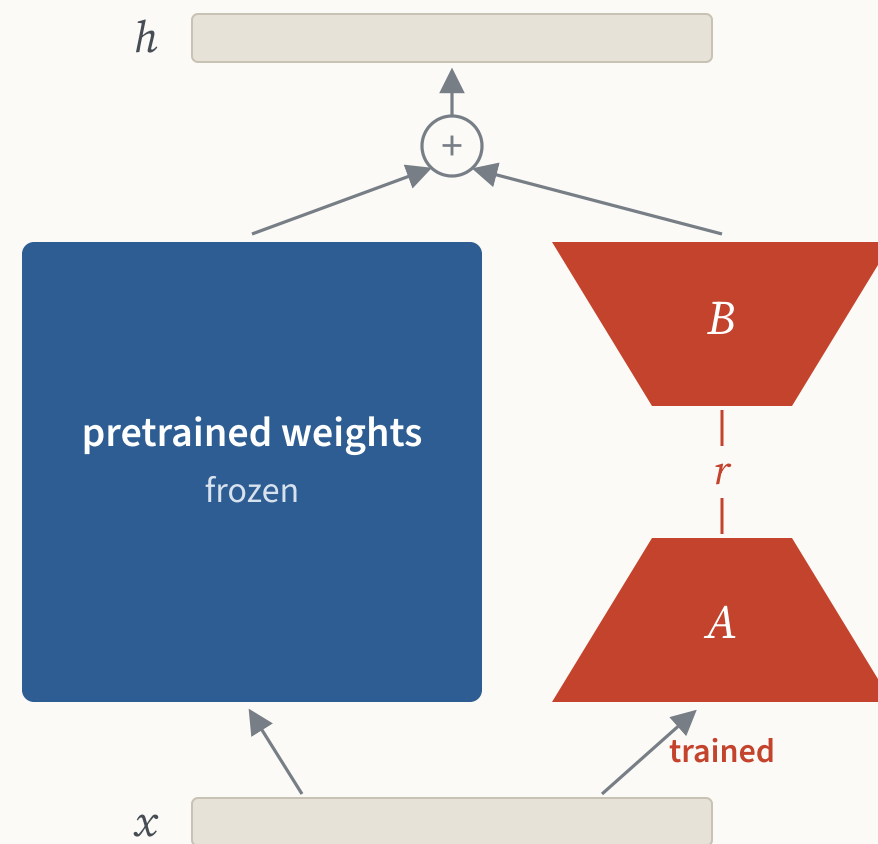


At test time every image carries the prompts through all twelve layers, so the sequence is longer. Fifty prompts make it 82 ms instead of 70, and 200 prompts double it.

The prompts cannot be folded into the weights, so this cost stays. LoRA, the next paper, avoids it.

Low-Rank Adaptation

Teach a frozen model a new task by learning a small, low-rank change to its weights.



Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen
Microsoft. ICLR 2022. Results on images from Zhang, Zhou & Liu 2022 and Zhu et al. 2024.

Presented by **Ertugrul Senturk**
October 2026

From VPT to LoRA

VPT

Jia et al. 2022



frozen Vision Transformer



head

Learned vectors join the image patches, so every image carries a longer sequence through every layer.

LoRA

Hu et al. 2022



frozen Vision Transformer

 small trained change to its weights



head

The input stays as it is. **A small change to the weights** is learned, and after training it is added into them.

VPT trained 0.6% of the weights and beat full fine-tuning on 20 of 24 tasks. But its prompts make every image slower, and they can never be folded into the model.

LoRA leaves the input alone and changes the weights instead. It ends up storing about as little per task as VPT, while the model runs exactly as fast as before.

Every fine-tuned copy is as big as the model

The paper starts from the largest model of its time, GPT-3, with 175 billion weights. Each fine-tuned copy takes 350 GB, so a service with a hundred tasks would store 35 TB. LoRA keeps one shared model and only a small file per task, and the frozen weights also need no optimizer state, which cuts training memory from 1.2 TB to 350 GB.

Shown for **1 task on ViT-B/16, 86 million weights.**

The next pages show 100 tasks, then GPT-3.

Full fine-tuning: 1 retrained ViT-B copy



343 MB

1 copy at 343 MB each

LoRA: one shared ViT-B plus 1 small file



each red speck is one task file, to scale

345 MB

343 MB once, plus 1.5 MB per task

GPT-3 in 16-bit numbers: 350 GB per copy, 35 MB per LoRA file at rank 4 on the query and value weights (Hu et al. 2022, Sec. 4.2). ViT-B/16 in 32-bit: 343 MB per copy; its LoRA file is rank 8 on the same weights plus a 102-class head. Squares are sized by area.

Every fine-tuned copy is as big as the model

The paper starts from the largest model of its time, GPT-3, with 175 billion weights. Each fine-tuned copy takes 350 GB, so a service with a hundred tasks would store 35 TB. LoRA keeps one shared model and only a small file per task, and the frozen weights also need no optimizer state, which cuts training memory from 1.2 TB to 350 GB.

Shown for **100 tasks** on ViT-B/16, 86 million weights.

Full fine-tuning: 100 retrained ViT-B copies



34 GB

100 copies at 343 MB each

LoRA: one shared ViT-B plus 100 small files



493 MB

343 MB once, plus 1.5 MB per task

GPT-3 in 16-bit numbers: 350 GB per copy, 35 MB per LoRA file at rank 4 on the query and value weights (Hu et al. 2022, Sec. 4.2). ViT-B/16 in 32-bit: 343 MB per copy; its LoRA file is rank 8 on the same weights plus a 102-class head. Squares are sized by area.

Every fine-tuned copy is as big as the model

The paper starts from the largest model of its time, GPT-3, with 175 billion weights. Each fine-tuned copy takes 350 GB, so a service with a hundred tasks would store 35 TB. LoRA keeps one shared model and only a small file per task, and the frozen weights also need no optimizer state, which cuts training memory from 1.2 TB to 350 GB.

Shown for 100 tasks on GPT-3, 175 billion weights.

Full fine-tuning: 100 retrained GPT-3 copies



35 TB

100 copies at 350 GB each

LoRA: one shared GPT-3 plus 100 small files



354 GB

350 GB once, plus 35 MB per task

GPT-3 in 16-bit numbers: 350 GB per copy, 35 MB per LoRA file at rank 4 on the query and value weights (Hu et al. 2022, Sec. 4.2). ViT-B/16 in 32-bit: 343 MB per copy; its LoRA file is rank 8 on the same weights plus a 102-class head. Squares are sized by area.

Earlier cheap methods cost time or sequence length

	Time for one forward pass	Extra time
One input of 128 tokens		
Fine-tuned, or LoRA merged	19.8 ms	
Adapter, one per block	23.9 ms	+20.7%
Adapter, two per block	25.8 ms	+30.3%
16 inputs of 256 tokens		
Fine-tuned, or LoRA merged	338 ms	
Adapter, one per block	354.8 ms	+5.0%
Adapter, two per block	366.3 ms	+8.4%
32 inputs of 512 tokens		
Fine-tuned, or LoRA merged	1449.4 ms	
Adapter, one per block	1482 ms	+2.2%
Adapter, two per block	1492.2 ms	+3.0%

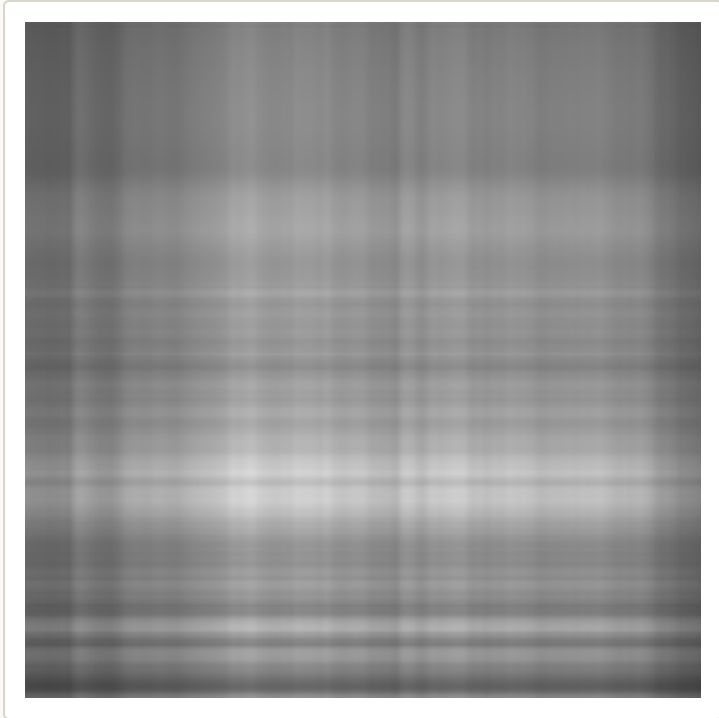
Adapters insert small extra layers into every block. They have few weights, but an image or a sentence must pass through them one after another, so serving a single input takes 20 to 30% longer. With large batches the gap shrinks to 2 or 3%.

Prompt methods such as prefix tuning and VPT lengthen the input instead. That leaves less room for the real input and costs compute in every layer, and on GPT-3 prefix tuning got worse beyond 256 prompt tokens.

LoRA adds no layer and no token at test time.

Hu et al. 2022, Table 1: one forward pass of GPT-2 medium, mean of 100 trials on one NVIDIA Quadro RTX 8000; bars are drawn relative to the fine-tuned model in each setting. Prefix tuning: Sec. 5.5 and Table 15.

What a low-rank matrix is



Rank 1: $1 \times (274 + 274) = \mathbf{548}$ numbers, 0.7% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

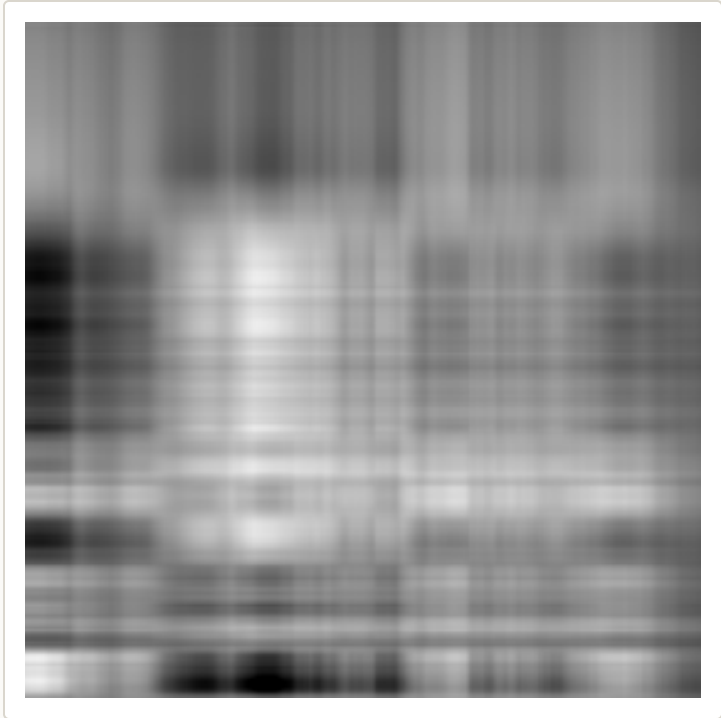
A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

One piece gives only bands of light and dark, because every row of it is the same pattern, scaled.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.

$$\begin{array}{c} \text{photo} \\ 274 \times 274 \end{array} = \begin{array}{c} | \\ B \\ 274 \times 1 \end{array} \times \begin{array}{c} \text{---} \\ A \\ 1 \times 274 \end{array}$$

What a low-rank matrix is



Rank 2: $2 \times (274 + 274) = \mathbf{1,096}$ numbers, 1.5% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

Two pieces are still mostly bands, now in two patterns.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.

$$\begin{array}{c} \text{photo} \\ 274 \times 274 \end{array} = \begin{array}{c} | \\ B \\ 274 \times 2 \end{array} \times \begin{array}{c} \text{---} \\ A \\ 2 \times 274 \end{array}$$

What a low-rank matrix is



Rank 4: $4 \times (274 + 274) = \mathbf{2,192}$ numbers, 2.9% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

With four pieces the main light and dark areas appear.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.

$$\begin{array}{c} \text{photo} \\ 274 \times 274 \end{array} = \begin{array}{c} | \\ B \\ 274 \times 4 \end{array} \times \begin{array}{c} \text{---} \\ A \\ 4 \times 274 \end{array}$$

What a low-rank matrix is



Rank 8: $8 \times (274 + 274) = 4,384$ numbers, 5.8% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

With eight pieces, 6% of the numbers, the rough shape of the cat appears, because the first pieces carry most of the picture.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.

$$\begin{array}{c} \text{photo} \\ 274 \times 274 \end{array} = \begin{array}{c} | \\ B \\ 274 \times 8 \end{array} \times \begin{array}{c} \text{---} \\ A \\ 8 \times 274 \end{array}$$

What a low-rank matrix is



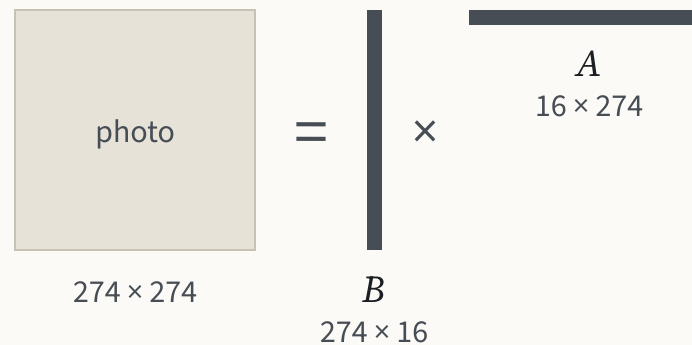
Rank 16: $16 \times (274 + 274) = \mathbf{8,768}$ numbers,
11.7% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

With 16 pieces the cat is recognisable, from 12% of the numbers.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.



What a low-rank matrix is



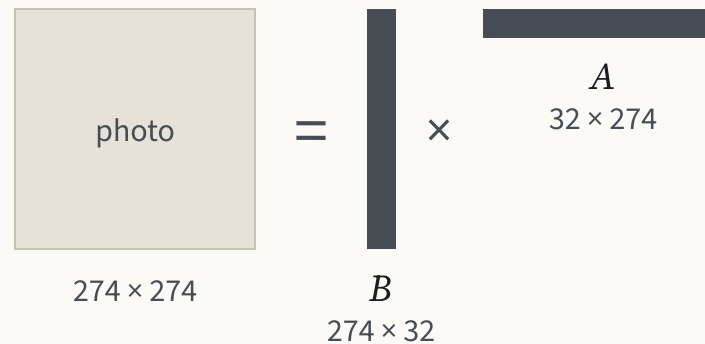
Rank 32: $32 \times (274 + 274) = \mathbf{17,536}$ numbers,
23.4% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

With 32 pieces the face and the fur appear.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.



What a low-rank matrix is



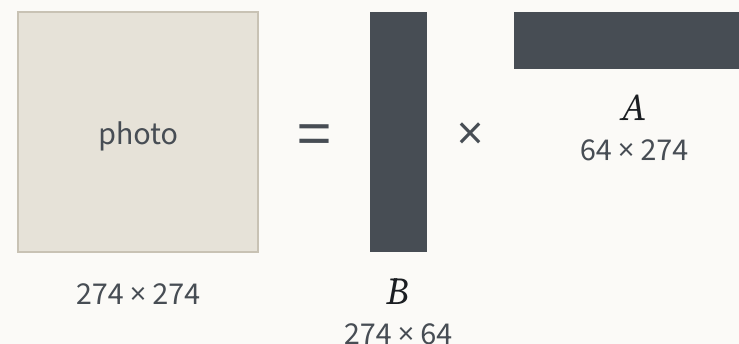
Rank 64: $64 \times (274 + 274) = \mathbf{35,072}$ numbers,
46.7% of the photo's 75,076

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

With 64 pieces it is hard to tell from the original.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.



What a low-rank matrix is



75,076 numbers: the full photo

1	2	4	8	16	32	64	all 274
---	---	---	---	----	----	----	---------

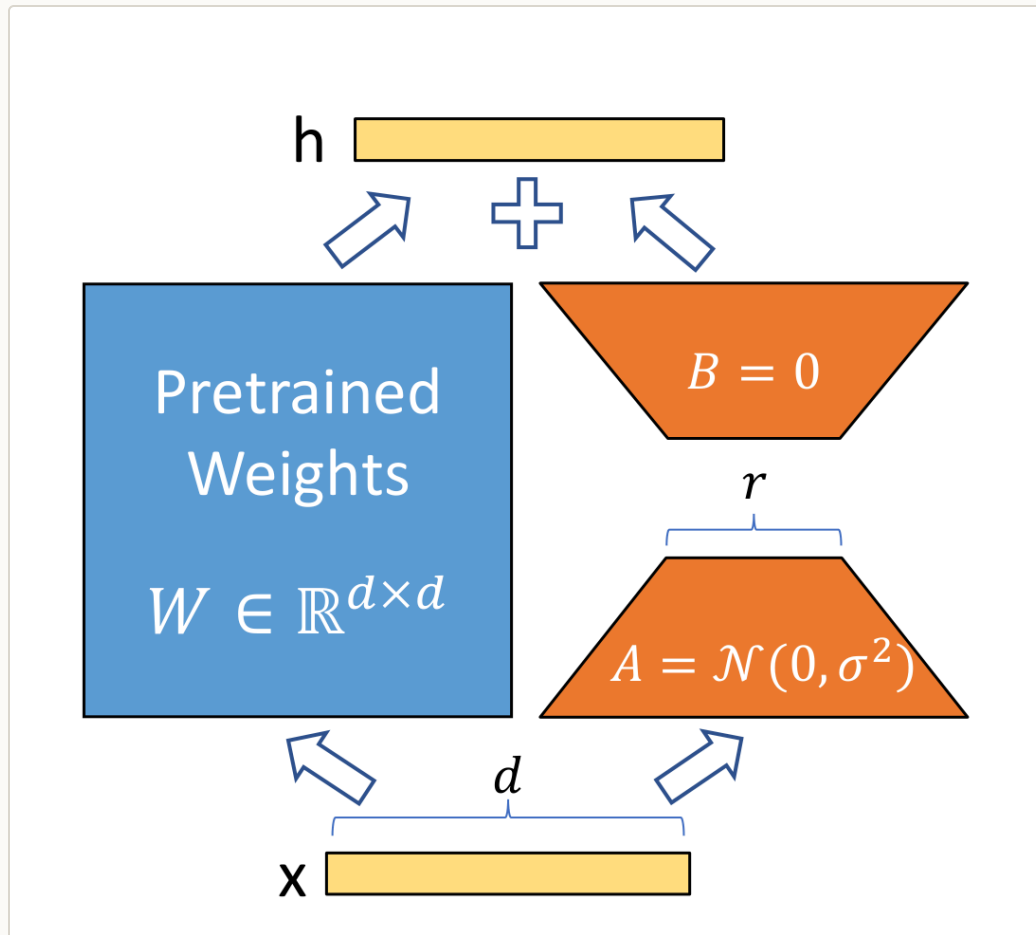
A grayscale photo is a matrix of 274×274 brightness values. Any matrix can be written as a sum of simple pieces, where each piece is one column of numbers times one row. The rank is the number of pieces.

All 274 pieces give back the photo exactly.

LoRA bets that the change fine-tuning makes to a weight matrix is like this photo: a few pieces carry almost all of it.

$$\begin{array}{ccc} \begin{array}{c} \text{photo} \\ 274 \times 274 \end{array} & = & \begin{array}{c} B \\ 274 \times 274 \end{array} \times \begin{array}{c} A \\ 274 \times 274 \end{array} \end{array}$$

LoRA adds a trainable bypass to a frozen weight



During training

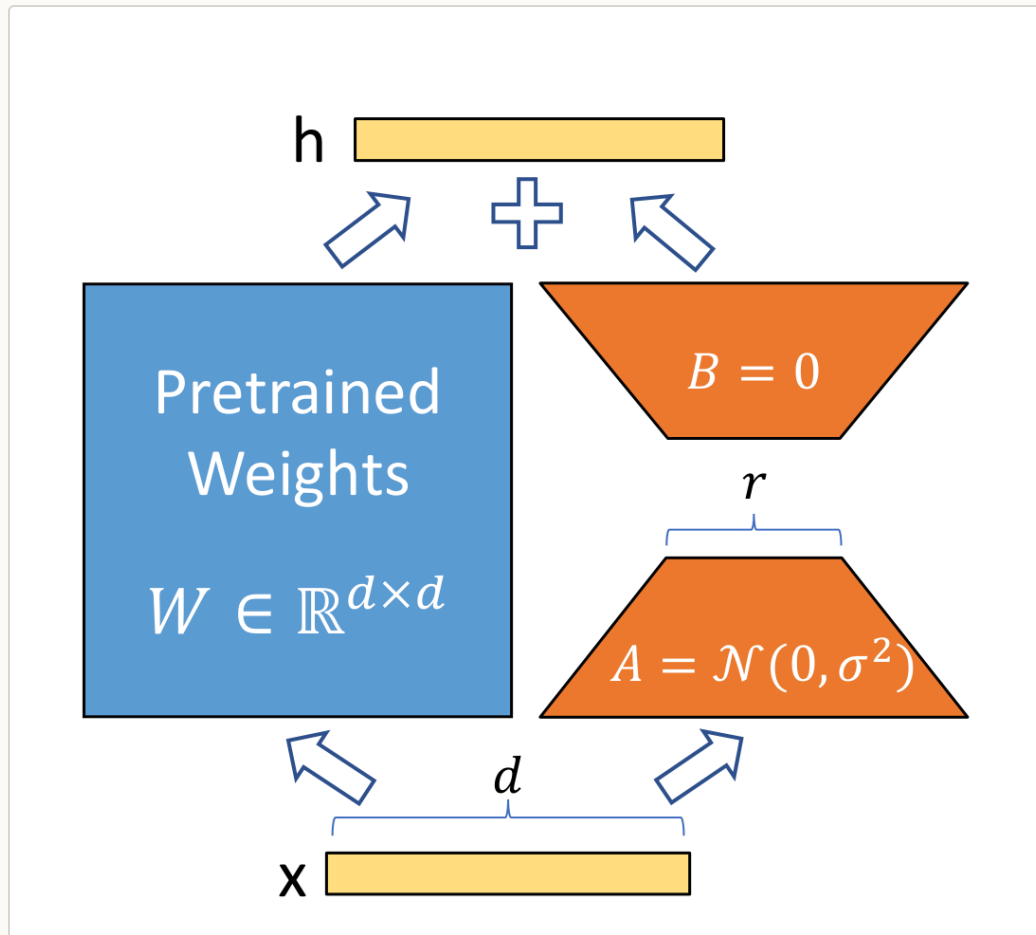
After training

The input goes through the frozen weight W_0 and, in parallel, through A and B , which are trained. The two results are added. The full change ΔW is never built; LoRA trains the thin matrices directly, with a rank r chosen by hand.

$$h = W_0 x + (\alpha/r) B A x$$

B starts at zero, so training begins exactly at the pretrained model. Only A and B get gradients and optimizer state.

LoRA adds a trainable bypass to a frozen weight



During training

After training

After training, the product BA is added into the weight once. The model is again one matrix per layer, of the same size, so it runs exactly as fast as before.

$$W = W_0 + (\alpha/r) BA$$

$$h = Wx$$

To switch task, subtract this BA and add that task's $B'A'$. Per task, only A and B are stored.

LoRA on a 3×3 weight matrix

Frozen path	LoRA path	Merging	Merged layer
-------------	-----------	---------	--------------

During training

$$\begin{array}{ccc|c|c} \boxed{2} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{5} \\ \boxed{1} & \boxed{3} & \boxed{0} & \boxed{2} & \boxed{7} \\ \boxed{0} & \boxed{1} & \boxed{2} & \boxed{3} & \boxed{8} \end{array} =$$

W x Wx

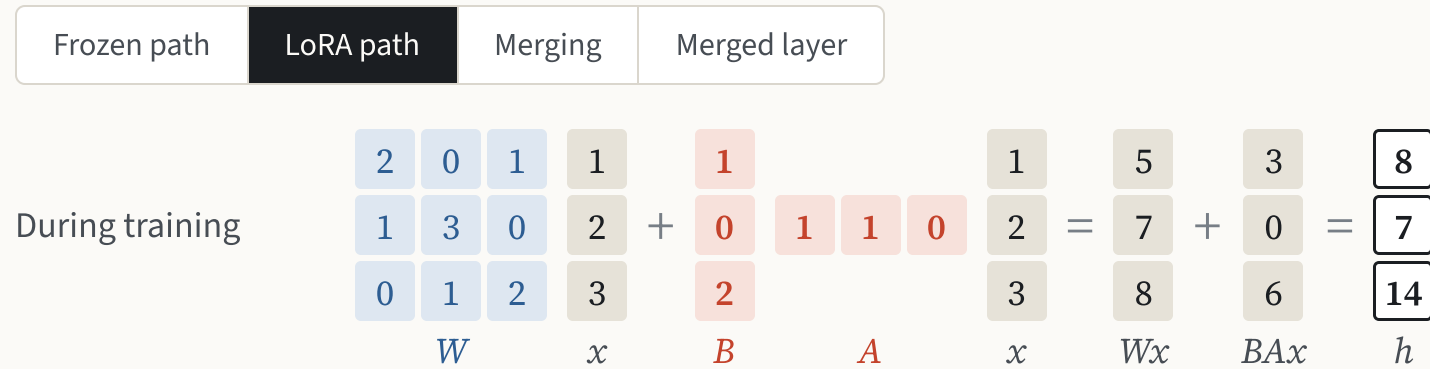
After training, once

At test time

Take a tiny weight W and an input x . The frozen path is just the pretrained layer, and Wx gives 5, 7 and 8.

Made-up numbers with rank $r = 1$ and $\alpha/r = 1$, so every product can be checked by eye. In ViT-B the same happens with 768×768 weights and $r = 8$.

LoRA on a 3×3 weight matrix



After training, once

At test time

LoRA adds a second path. A squeezes the input to r numbers, here a single 3, and B expands it back to three; the two paths are added. In training, only the six red numbers change.

Made-up numbers with rank $r = 1$ and $\alpha/r = 1$, so every product can be checked by eye. In ViT-B the same happens with 768×768 weights and $r = 8$.

LoRA on a 3×3 weight matrix

Frozen path	LoRA path	Merging	Merged layer
-------------	-----------	---------	--------------

During training

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|} \hline 2 & 0 & 1 & 1 \\ \hline 1 & 3 & 0 & 2 \\ \hline 0 & 1 & 2 & 3 \\ \hline \end{array} & & \begin{array}{|c|c|c|c|} \hline 1 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 \\ \hline 2 & 0 & 0 & 0 \\ \hline \end{array} \\
 W & & B \quad A \\
 x & &
 \end{array}
 +
 \begin{array}{|c|c|c|c|} \hline 1 & 5 & 3 & 8 \\ \hline 2 & 7 & 0 & 7 \\ \hline 3 & 8 & 6 & 14 \\ \hline \end{array}
 =
 \begin{array}{|c|c|c|c|} \hline 1 & 5 & 3 & 8 \\ \hline 2 & 7 & 0 & 7 \\ \hline 3 & 8 & 6 & 14 \\ \hline \end{array}$$

$x \quad Wx \quad BAx \quad h$

After training, once

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|} \hline 1 & 0 & 2 & 0 \\ \hline 0 & 1 & 1 & 0 \\ \hline 2 & 0 & 0 & 0 \\ \hline \end{array} & = & \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline 2 & 2 & 0 \\ \hline \end{array} \\
 B & & BA \\
 A & &
 \end{array}
 +
 \begin{array}{ccc}
 \begin{array}{|c|c|c|} \hline 2 & 0 & 1 \\ \hline 1 & 3 & 0 \\ \hline 0 & 1 & 2 \\ \hline \end{array} & + & \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline 2 & 2 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 3 & 1 & 1 \\ \hline 1 & 3 & 0 \\ \hline 2 & 3 & 2 \\ \hline \end{array} \\
 W & & BA \quad W'
 \end{array}$$

At test time

After training, B times A is worked out once. A column times a row is a full 3×3 matrix, the same shape as W , so it is simply added to it; the red cells are the ones that changed.

Made-up numbers with rank $r = 1$ and $\alpha/r = 1$, so every product can be checked by eye. In ViT-B the same happens with 768×768 weights and $r = 8$.

LoRA on a 3×3 weight matrix

Frozen path	LoRA path	Merging	Merged layer
-------------	-----------	---------	--------------

During training

2

0

1

1

1

3

0

2

0

1

2

3

1

3

0

2

0

1

2

3

1

0

2

0

1

1

0

1

5

3

2

7

8

0

0

6

8

7

14

2

0

1

1

3

0

0

1

2

1

5

3

2

7

8

0

0

6

8

7

14

After training, once

1

0

2

1

3

0

0

1

2

1

5

3

2

7

8

0

0

6

8

7

14

1

0

2

1

3

0

0

1

2

1

5

3

2

7

8

0

0

6

8

7

14

At test time

3

1

1

0

0

1

1

0

2

2

0

1

5

3

2

7

8

0

0

6

8

7

14

3

1

1

0

1

1

2

2

0

1

5

3

2

7

8

0

0

6

8

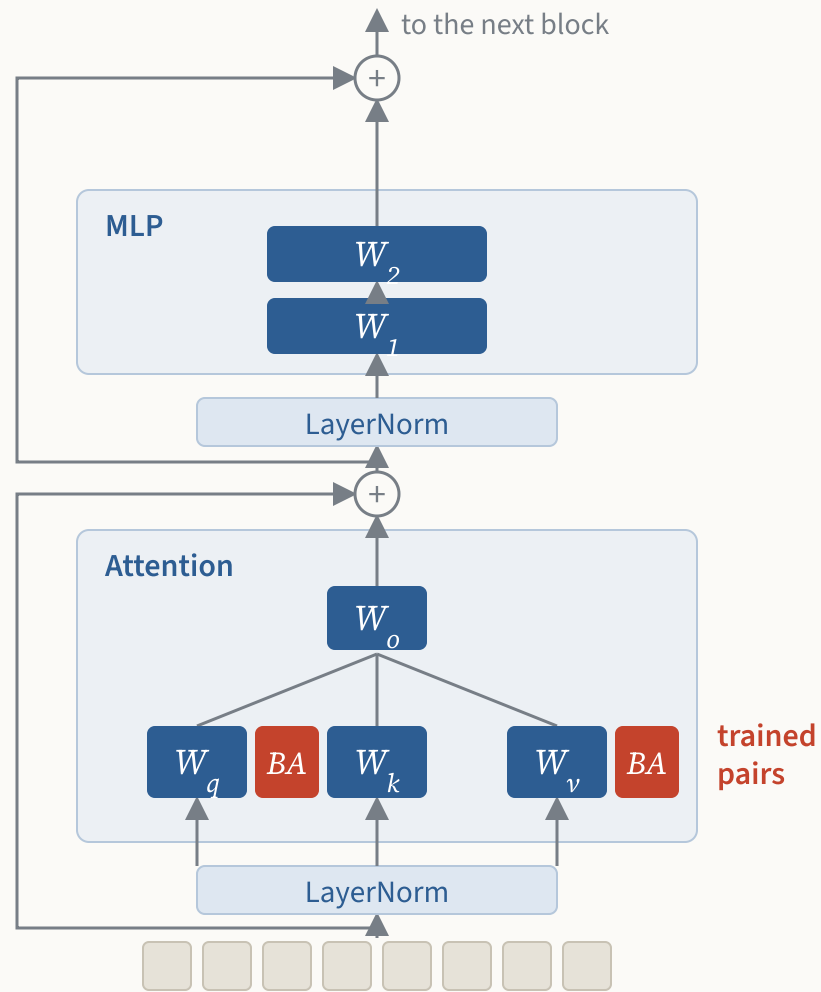
7

14

The merged weight gives exactly the same output, because $Wx + B(Ax) = (W + BA)x$. B and A are deleted, and the layer is one ordinary matrix again.

Made-up numbers with rank $r = 1$ and $\alpha/r = 1$, so every product can be checked by eye. In ViT-B the same happens with 768×768 weights and $r = 8$.

Where LoRA goes in a Vision Transformer

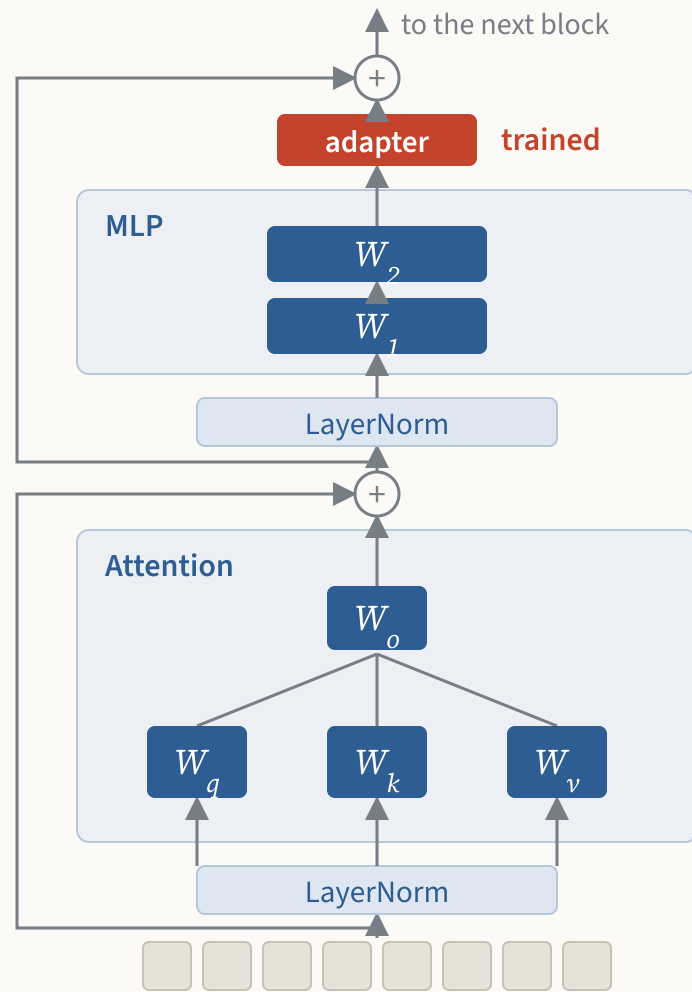


LoRA

Adapter

LoRA puts a trained pair B A beside the query and value weights of the attention, in all 12 blocks. The key and output weights, the MLP and the layer norms stay frozen. After merging, the block is exactly the original.

Where LoRA goes in a Vision Transformer



An adapter, for contrast, is a small trained two-layer network inserted after the MLP. It is an extra layer that every input must pass through, which is where its extra time comes from; LoRA sits beside weights that already exist.

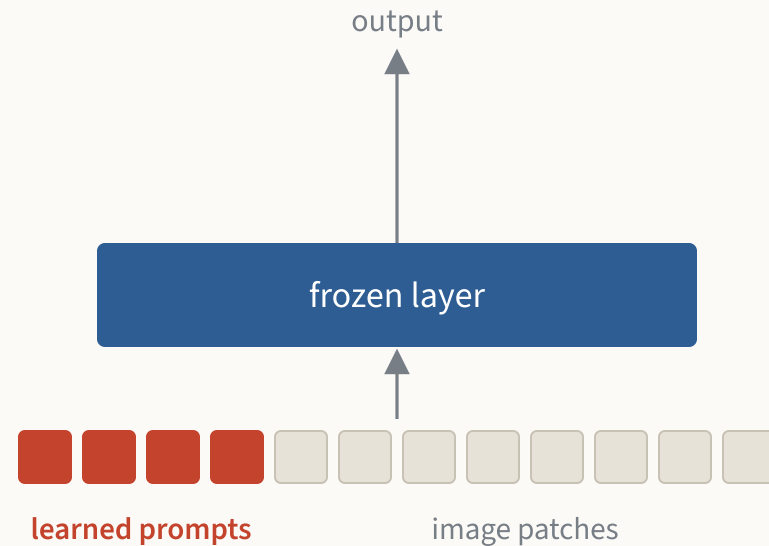
VPT and LoRA side by side

What is learned

During training

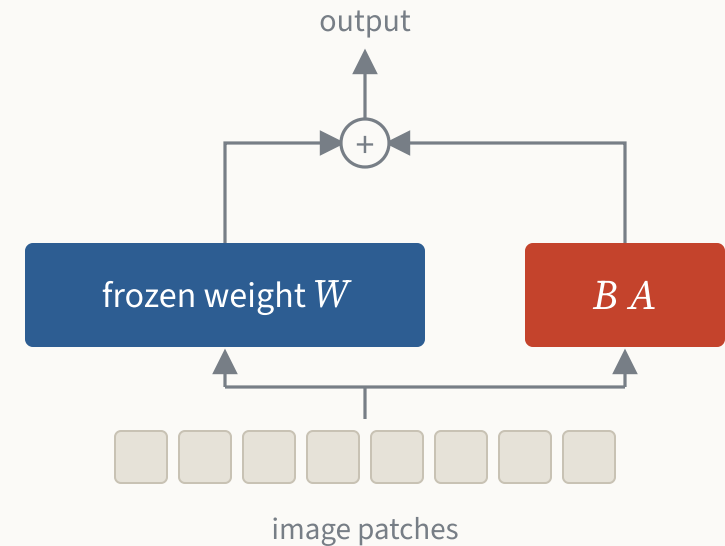
At test time

VPT



VPT learns extra input vectors. The same 50 per layer are placed in front of every image of the task; they start random and are not taken from any image.

LoRA



LoRA learns a correction to the weights, the pair B and A beside the query and value weights. It is also one pair for the whole task, and B starts at zero.

One layer of ViT-B/16 drawn for each. VPT-deep with 50 prompts per layer (Jia et al. 2022); LoRA with rank 8 on the query and value weights (Hu et al. 2022). Both train the same 102-class head.

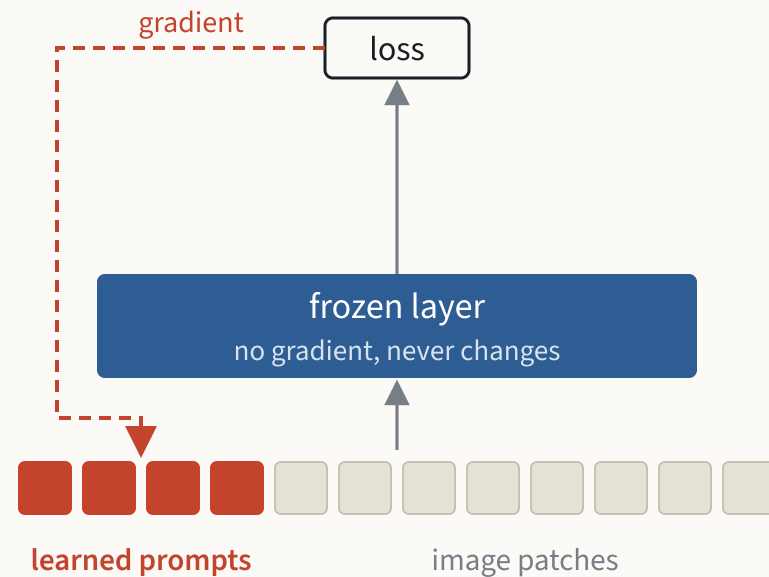
VPT and LoRA side by side

What is learned

During training

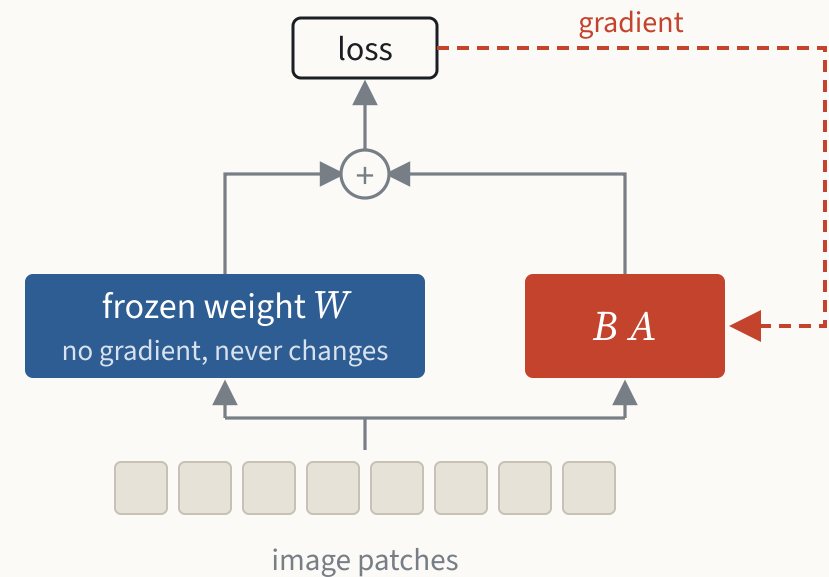
At test time

VPT



Each training image goes through the frozen model with the prompts attached. The loss sends gradients back, and only the prompts and the head change.

LoRA



Each training image takes both paths, Wx and BAx . Only B , A and the head change; W never receives a gradient.

One layer of ViT-B/16 drawn for each. VPT-deep with 50 prompts per layer (Jia et al. 2022); LoRA with rank 8 on the query and value weights (Hu et al. 2022). Both train the same 102-class head.

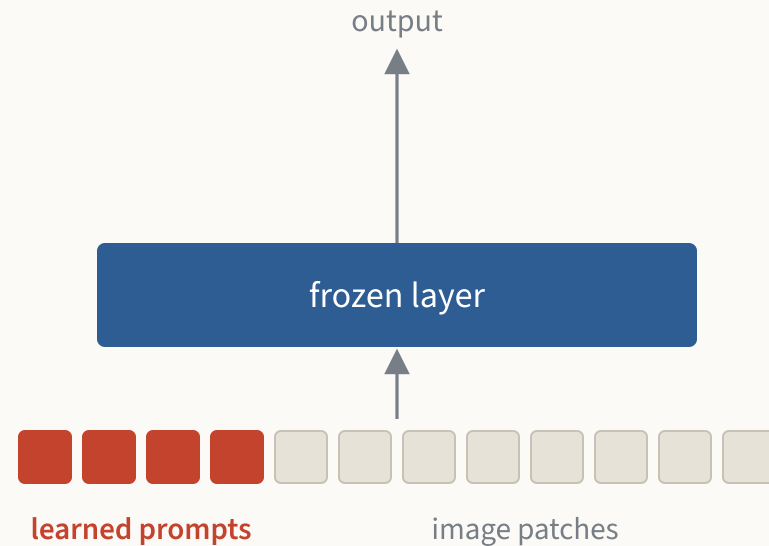
VPT and LoRA side by side

What is learned

During training

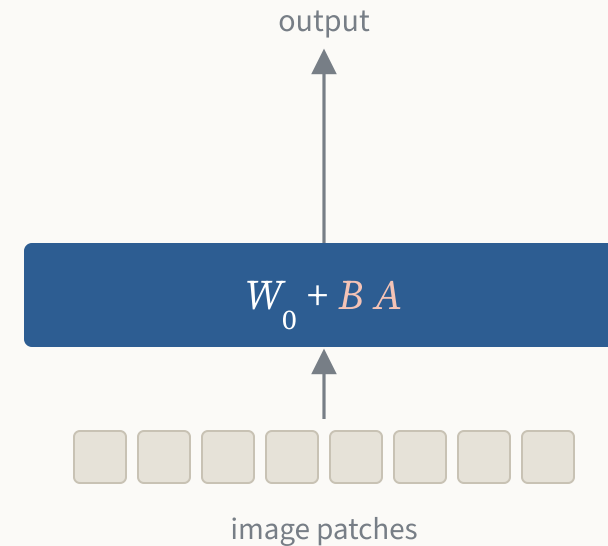
At test time

VPT



The prompts must be attached to every test image, so each layer works on 247 tokens instead of 197. They are inputs, so they can never be folded into the weights.

LoRA



BA is added into W once, and the layer is one plain matrix again. The test image goes through exactly the original model.

One layer of ViT-B/16 drawn for each. VPT-deep with 50 prompts per layer (Jia et al. 2022); LoRA with rank 8 on the query and value weights (Hu et al. 2022). Both train the same 102-class head.

How much LoRA trains and stores per task

$2 \cdot N \cdot m \cdot d \cdot r = 2 \times 12 \times 2 \times 768 \times 8 = \mathbf{294,912}$ numbers in the pairs, plus 78,438 for a 102-class head

Numbers trained and stored per task

Full fine-tuning	all 85.8 million weights, retrained for this task	343 MB 100%
VPT-deep, 50 prompts	frozen and shared by every task	2.2 MB 0.63%
LoRA	frozen and shared by every task	1.5 MB 0.44%

Compute per image at test time

Plain ViT	197 tokens	1.00×
VPT-deep, 50 prompts	197 tokens	1.27× +50 tokens
LoRA, merged	197 tokens	1.00× nothing extra

Shown for rank 8 on the query and value weights. At rank 64 on all four attention weights the file grows to 19 MB, 5.6% of the model, and the compute per image does not change.

LoRA's saving grows with the width of the model

Full change ΔW	$d \times d = d^2$	589,824 numbers on ViT-B
B and A together	$d \times r + r \times d = 2rd$	12,288 numbers on ViT-B
Share of a full weight change stored by B and A, rank 8		width d
ViT-B/16	 2.08%	768
ViT-L/16	 1.56%	1,024
ViT-H/14	 1.25%	1,280
GPT-3	 0.13%	12,288

A full change to one $d \times d$ weight has d^2 numbers, while B and A together have $2rd$. Since d is the square root of d^2 , LoRA's cost grows like the square root of the full size, times the small constant $2r$.

So the wider the model, the smaller LoRA's share. Together with adapting only the query and value weights, this is why GPT-3's task file is 10,000 times smaller than a copy.

One $d \times d$ attention weight, rank $r = 8$ for every model. Widths: ViT-B/16 768, ViT-L/16 1,024, ViT-H/14 1,280 (Dosovitskiy et al. 2021); GPT-3 12,288 (Hu et al. 2022, Sec. 1). 10,000 times: Hu et al. 2022, Sec. 4.2.

LoRA on a ViT in a few lines

```
vit = timm.create_model(BACKBONE, pretrained=True, num_classes=C)
vit.requires_grad_(False); vit.head.requires_grad_(True)

class LoRAqv(nn.Module):
    # wraps qkv = Linear(768, 2304)
    def __init__(self, qkv, r=8, alpha=8):
        super().__init__()
        d = qkv.in_features
        self.qkv, self.s = qkv, alpha / r # frozen W0, scale  $\alpha/r$ 
        self.Aq = nn.Parameter(torch.randn(r, d) / d ** 0.5)
        self.Av = nn.Parameter(torch.randn(r, d) / d ** 0.5)
        self.Bq = nn.Parameter(torch.zeros(d, r)) # B = 0
        self.Bv = nn.Parameter(torch.zeros(d, r))

    def forward(self, x):
        q, k, v = self.qkv(x).chunk(3, dim=-1) # W0 x
        q = q + self.s * x @ self.Aq.T @ self.Bq.T # + B A x
        v = v + self.s * x @ self.Av.T @ self.Bv.T
        return torch.cat([q, k, v], dim=-1)

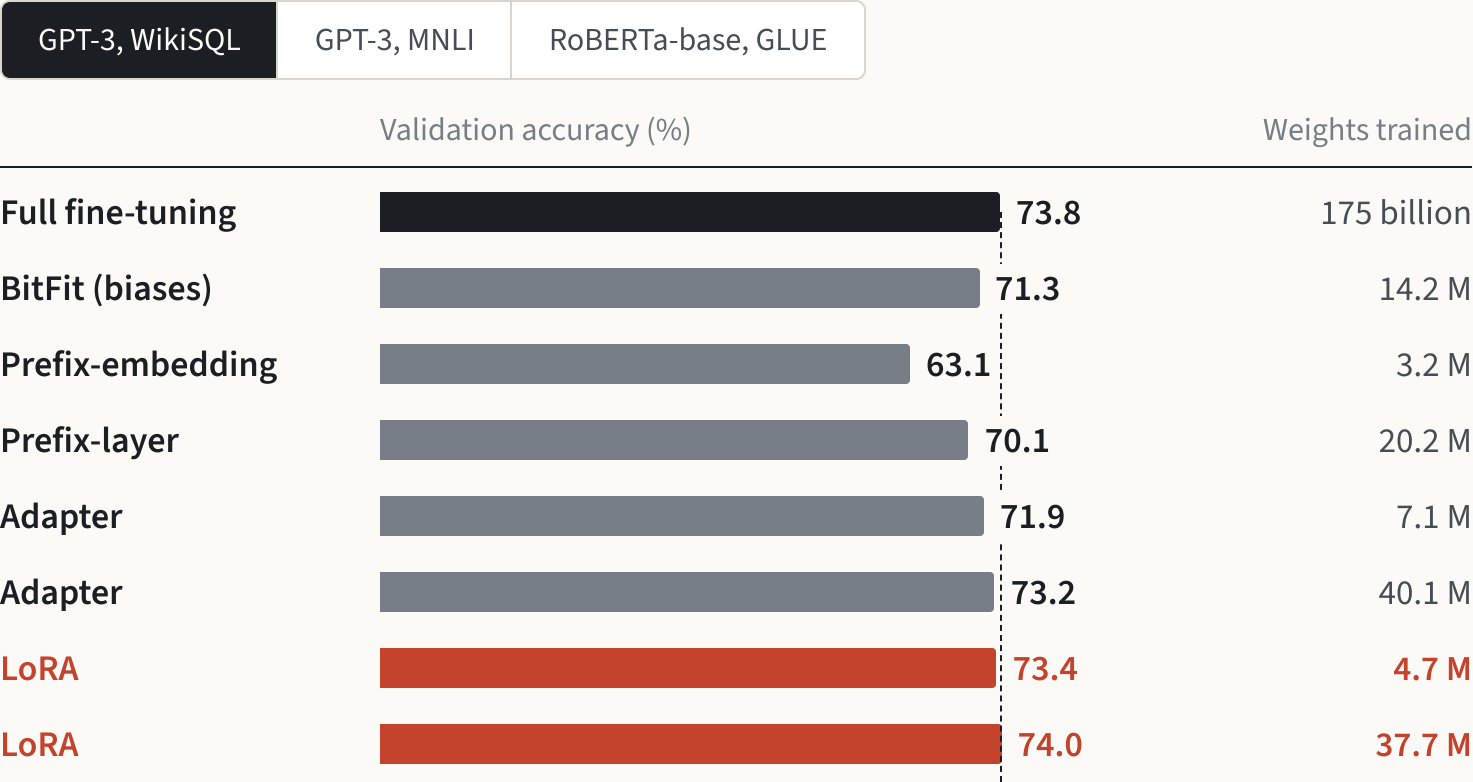
for blk in vit.blocks: blk.attn.qkv = LoRAqv(blk.attn.qkv)
```

B starts at zero and A at small random values, so the first forward pass is exactly the pretrained model.

The two new choices are the rank r and which weights get a pair. On ViT-B, Zhang et al. use rank 8 with AdamW at a learning rate of 0.001 for 100 epochs.

After training, each pair is added into its qkv weight once, and the model is a plain timm ViT again.

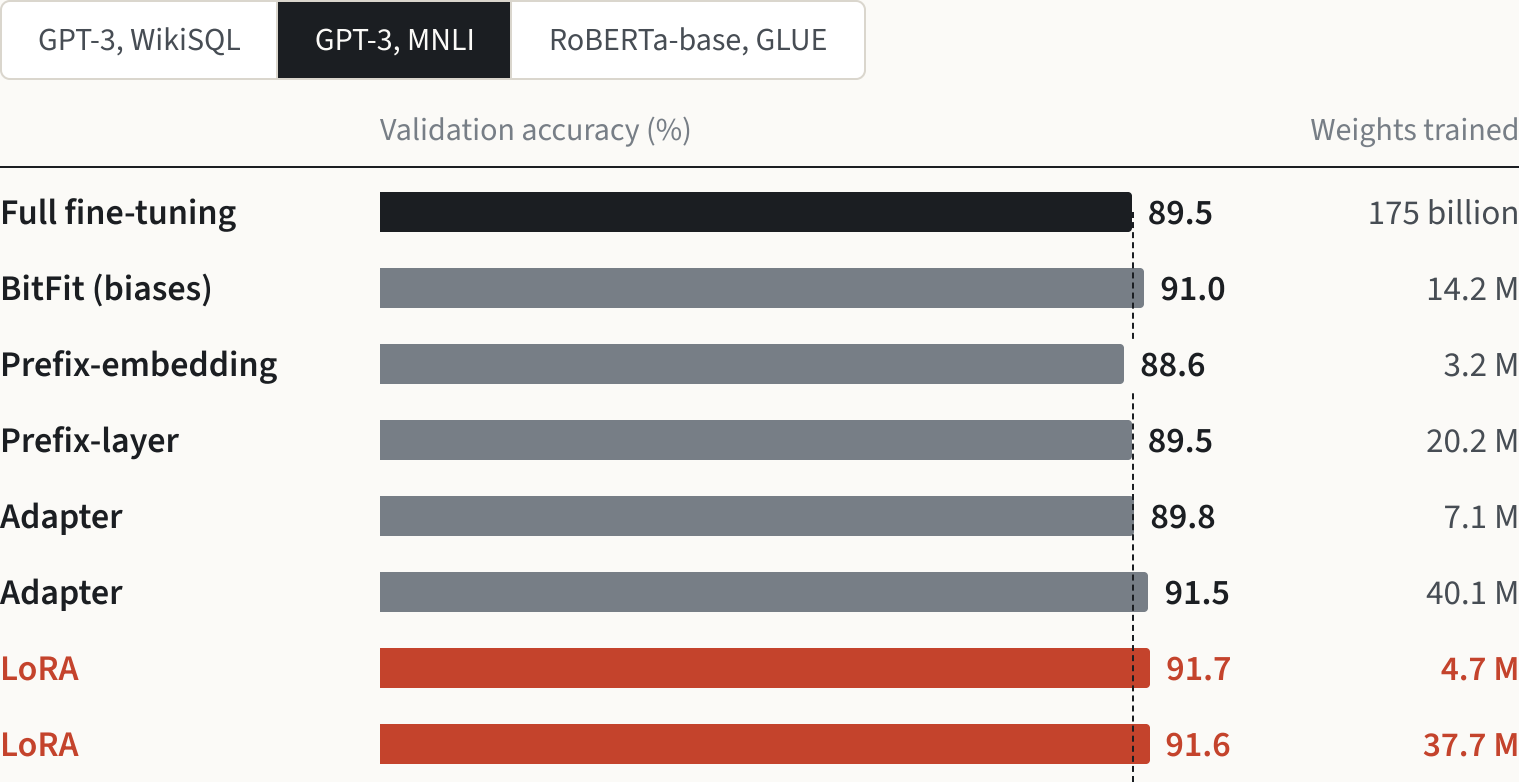
Results in the paper: language models



WikiSQL turns a question into an SQL query. On GPT-3, LoRA with 4.7 million trained weights, 0.003% of the model, is within the run-to-run noise of full fine-tuning, and slightly above it with 37.7 million.

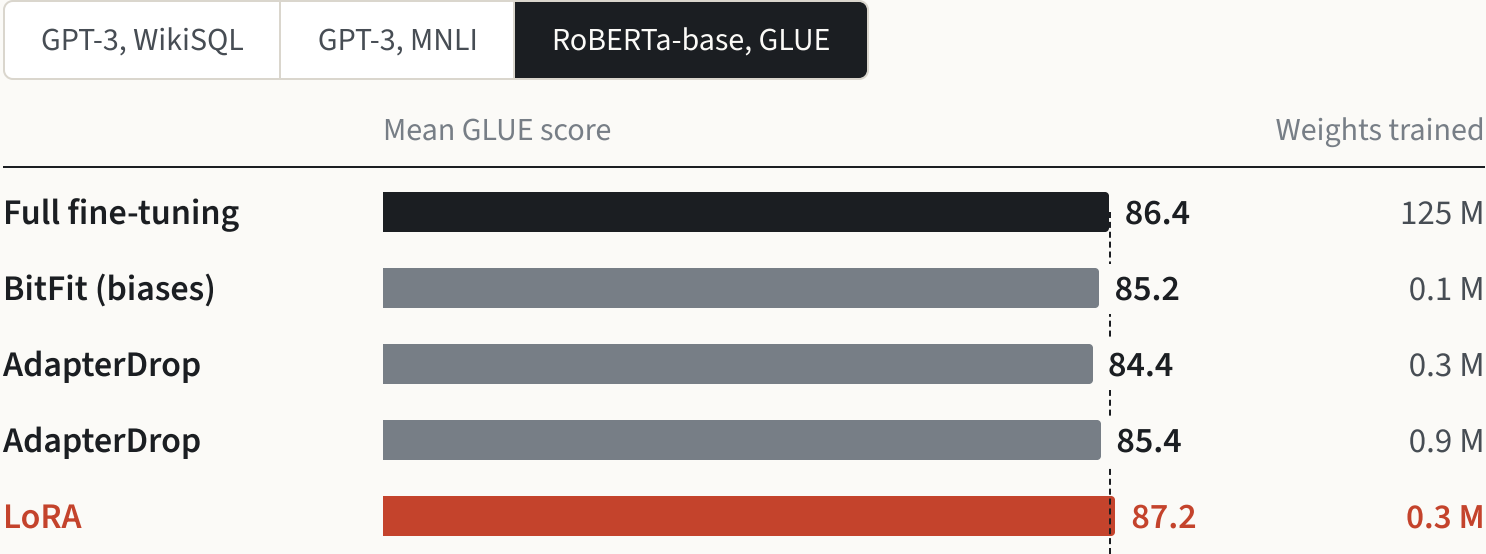
The prefix methods are far behind, and the adapter needs more weights to come close.

Results in the paper: language models



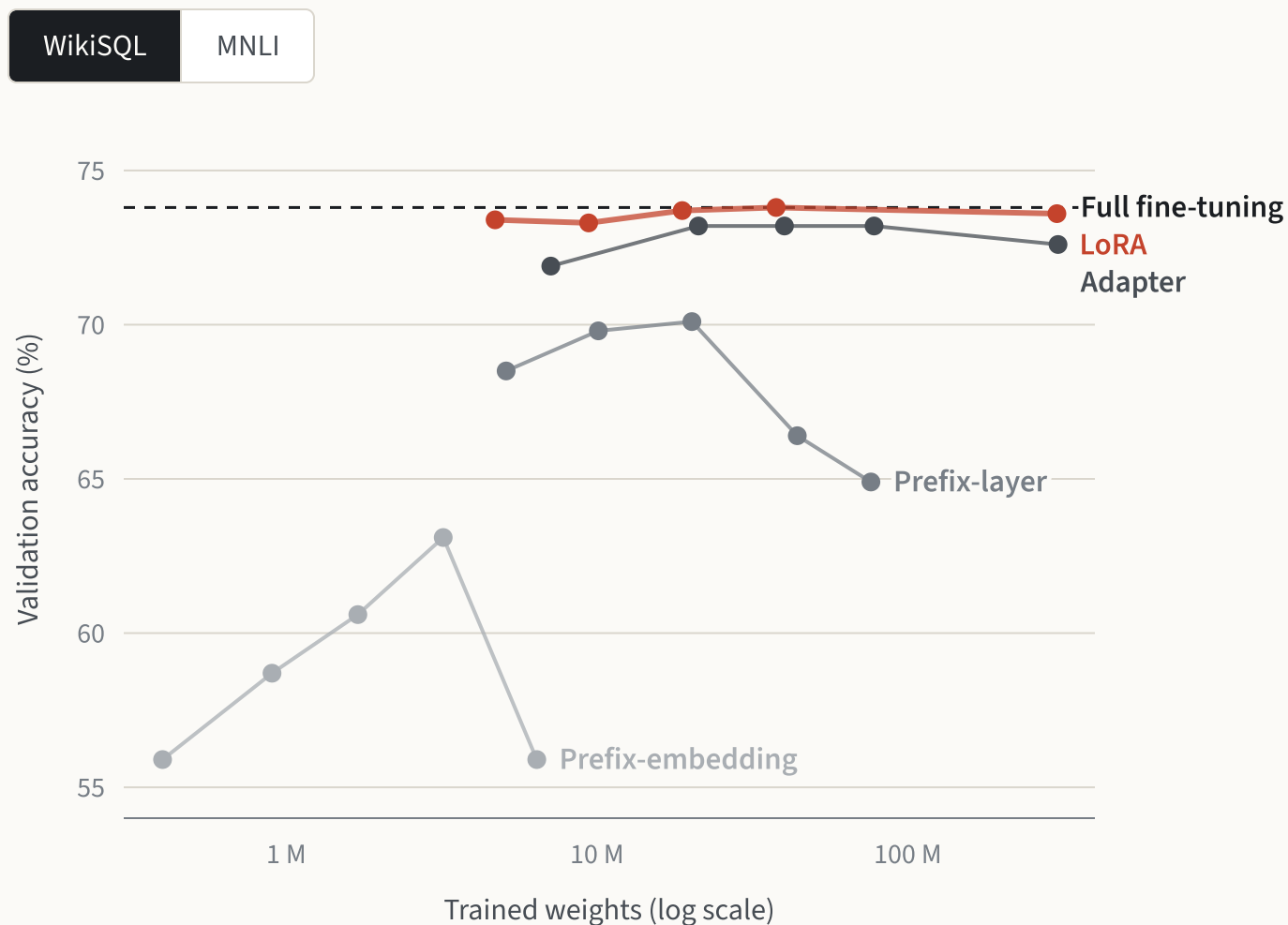
MNLI asks whether one sentence follows from another. Here LoRA is two points above full fine-tuning, 91.7 against 89.5, and above every other cheap method.

Results in the paper: language models



RoBERTa-base has 125 million weights, close to ViT-B. Averaged over the eight GLUE language tasks, LoRA with 0.3 million trained weights scores 87.2, against 86.4 for full fine-tuning.

What happens when more weights are trained

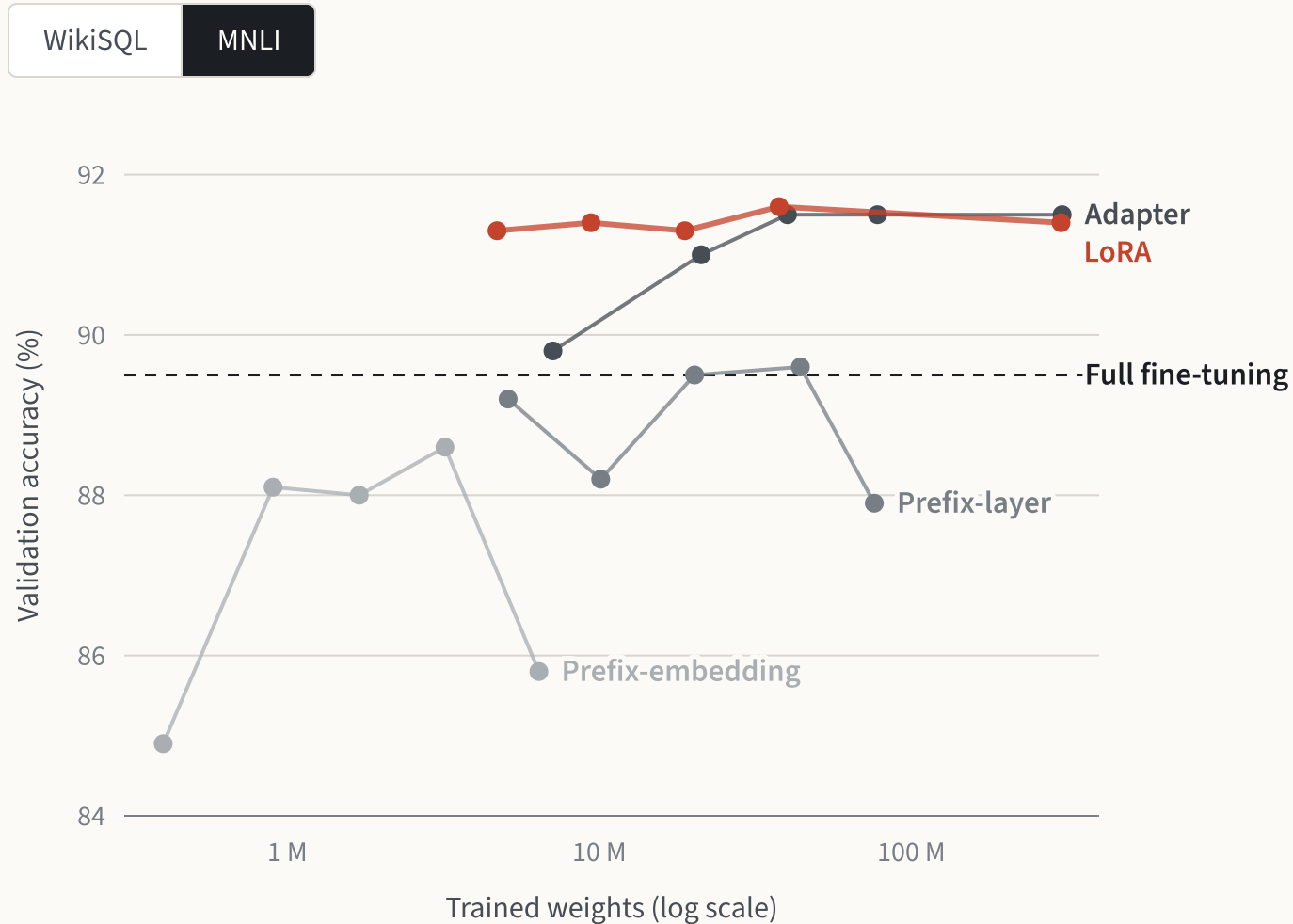


Each point is one size of a method. Prefix-embedding tuning improves up to 256 prompt tokens, then drops from 63.1 to 55.9. Prefix-layer tuning also falls after 20 million weights.

LoRA is flat from rank 1 to rank 64, at the level of full fine-tuning.

Hu et al. 2022, Table 15 (the points of Fig. 2): GPT-3 175B, validation accuracy; LoRA with equal rank on query and value, from rank 1 to 64. Full fine-tuning trains all 175 billion weights.

What happens when more weights are trained



On MNLI the prompt methods again rise and fall, and the adapter needs about 40 million weights to catch up.

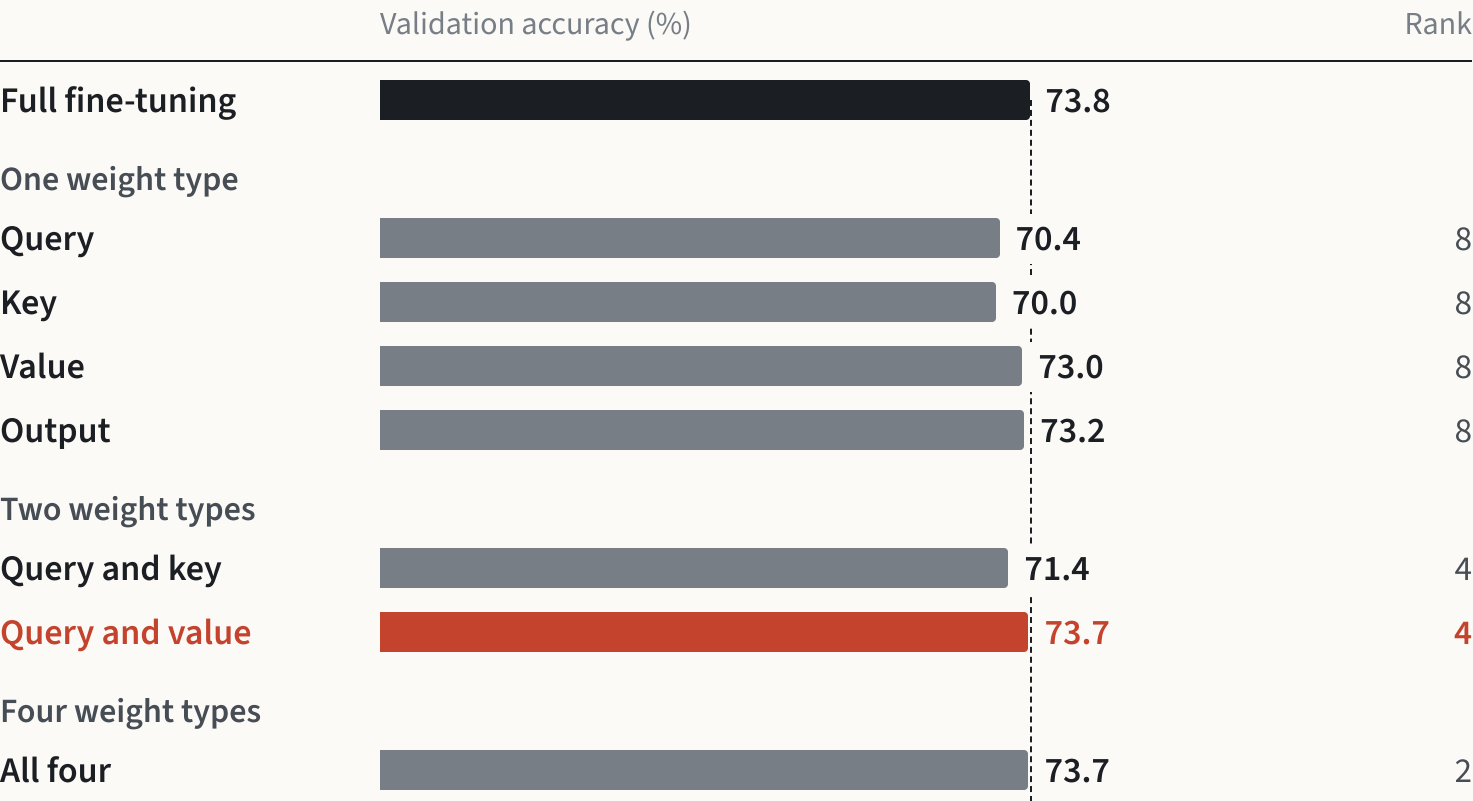
LoRA is above full fine-tuning at every size, so its rank is an easy choice.

Hu et al. 2022, Table 15 (the points of Fig. 2): GPT-3 175B, validation accuracy; LoRA with equal rank on query and value, from rank 1 to 64. Full fine-tuning trains all 175 billion weights.

Which weight matrices to adapt

WikiSQL

MNLI

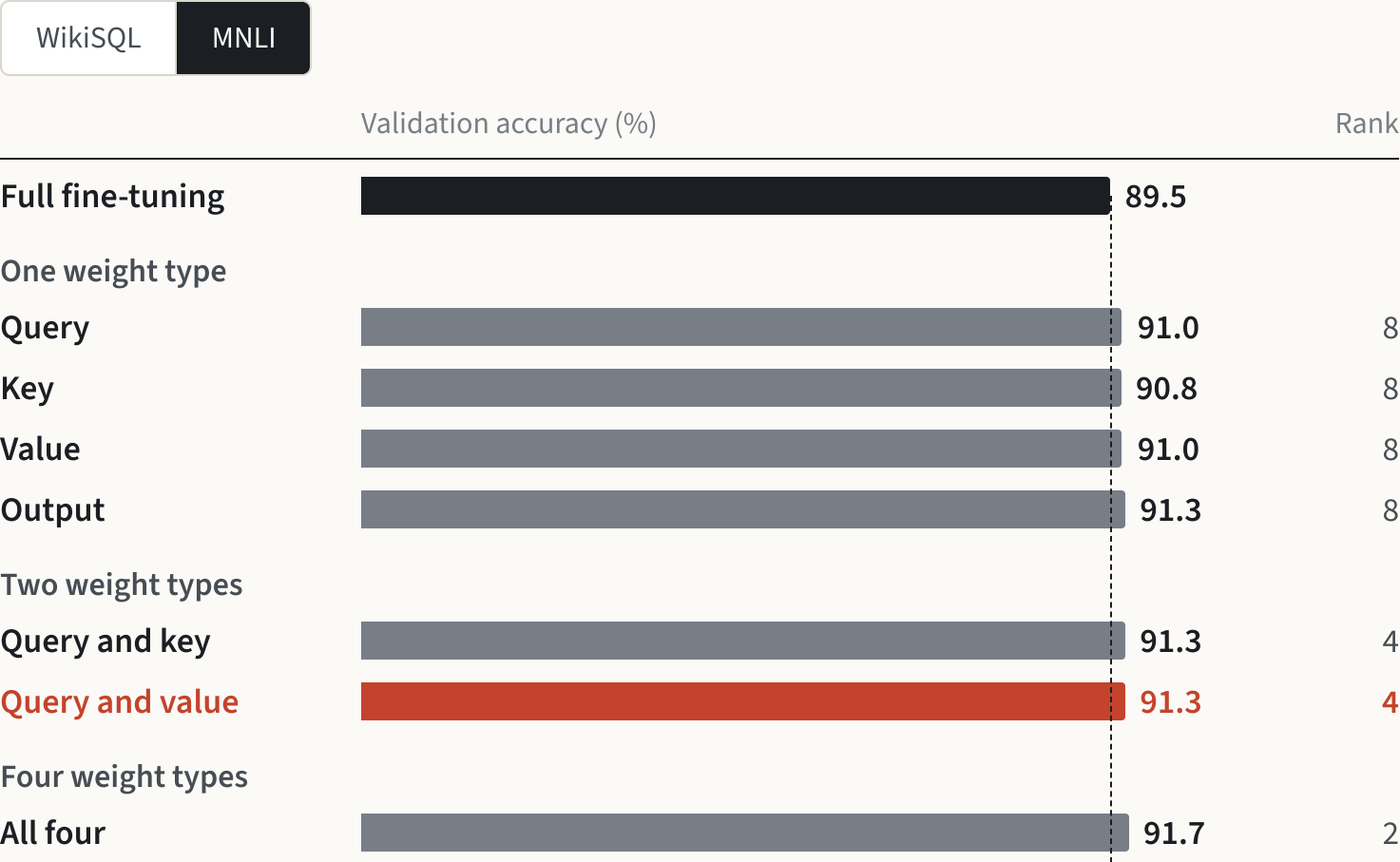


Every LoRA row trains the same 18 million weights. Putting them all into the query or the key weight is about three points worse than spreading them over query and value at rank 4.

Adapting all four at rank 2 does as well, so it pays to adapt more matrices with a smaller rank.

Hu et al. 2022, Table 5: GPT-3 175B with the same 18 million trained weights in every LoRA row, validation accuracy (noise ± 0.5 on WikiSQL, ± 0.1 on MNLI). Full fine-tuning: Table 4.

Which weight matrices to adapt

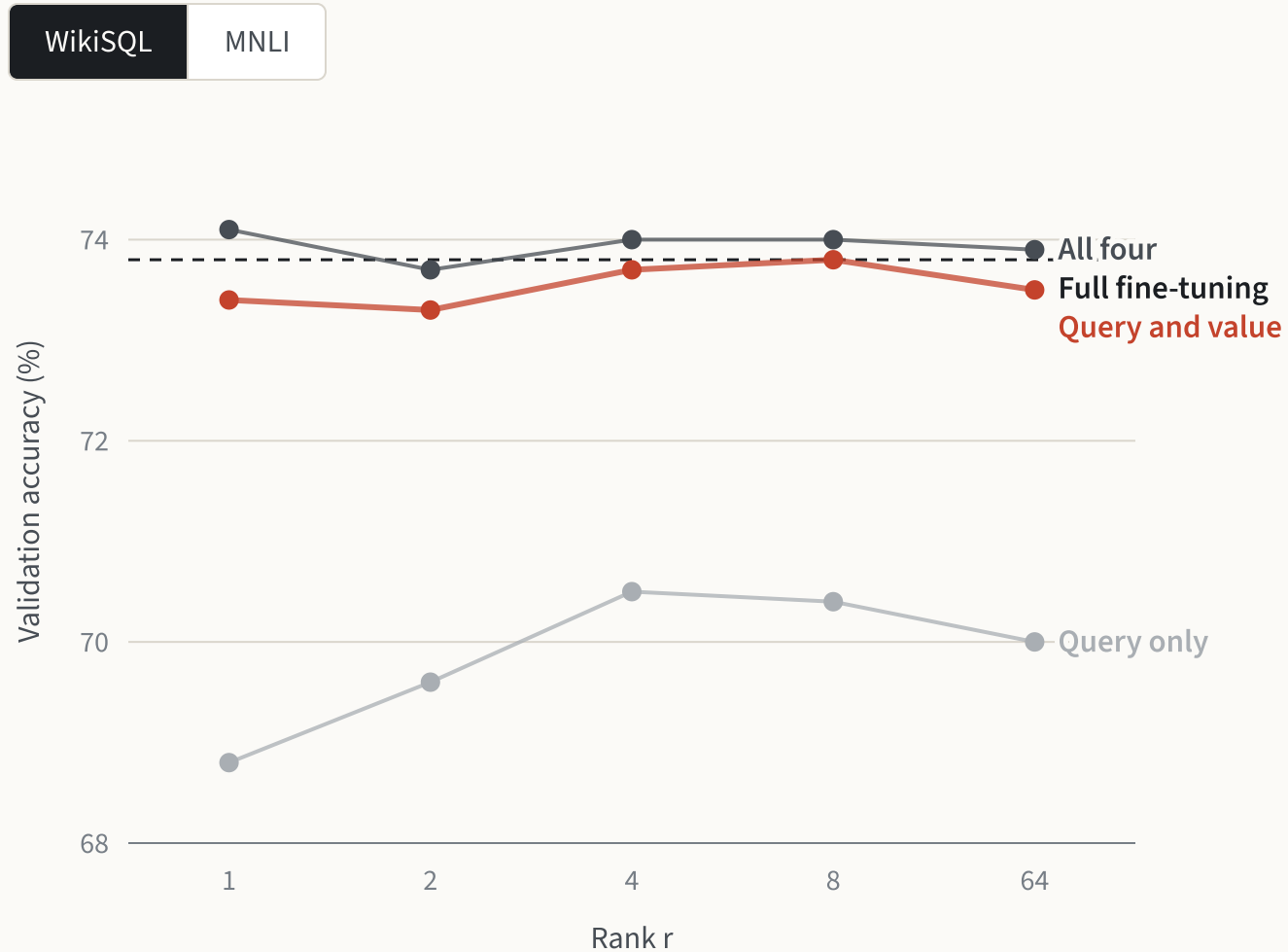


On MNLI the differences are smaller, but the order is similar: the query or key alone is lowest, and all four at rank 2 is highest, 91.7.

Every LoRA setting is above full fine-tuning here.

Hu et al. 2022, Table 5: GPT-3 175B with the same 18 million trained weights in every LoRA row, validation accuracy (noise ± 0.5 on WikiSQL, ± 0.1 on MNLI). Full fine-tuning: Table 4.

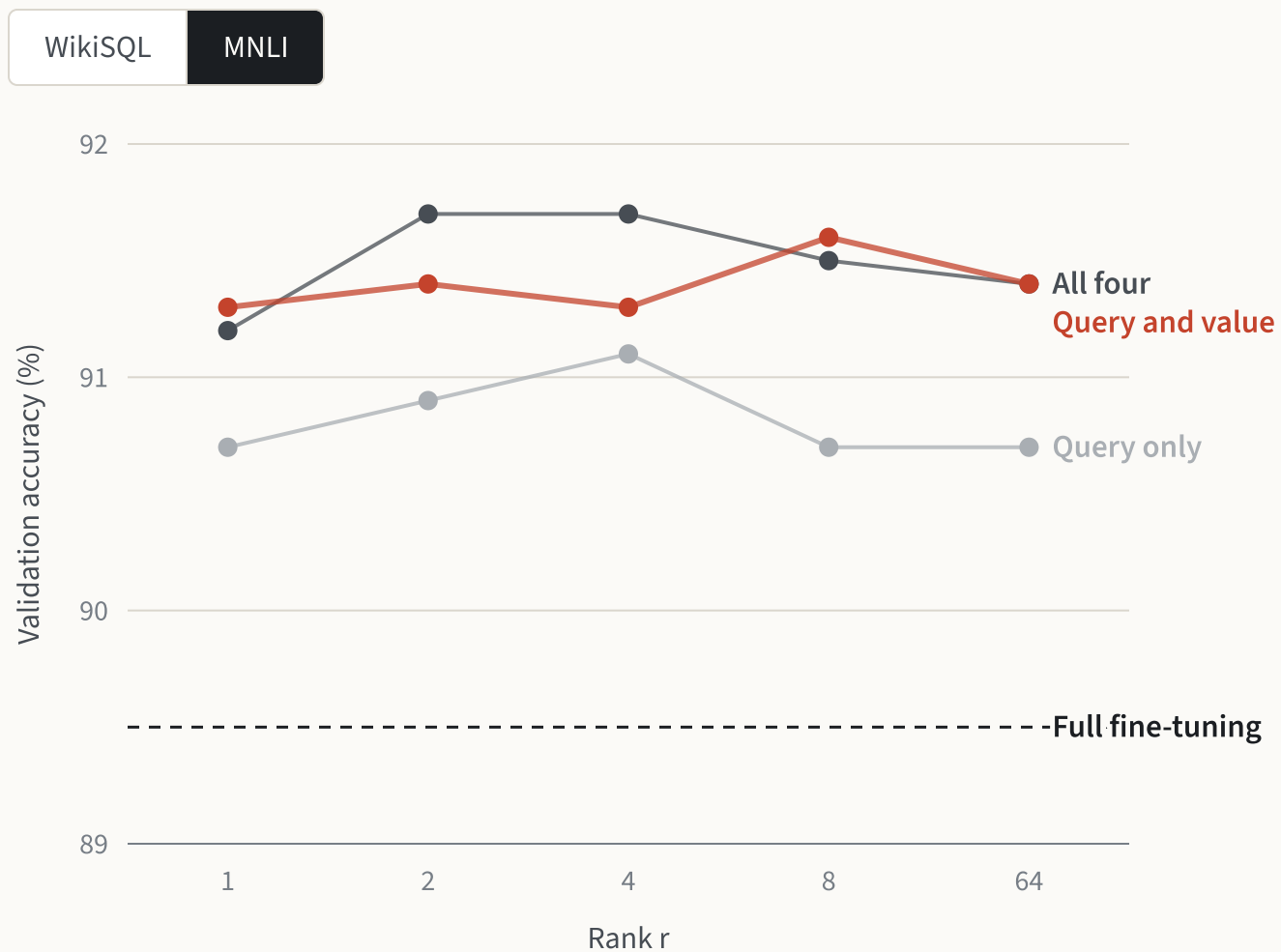
How small the rank can be



With the query and value weights together, rank 1 already gives 73.4, and no rank does better than 73.8, inside the noise of ± 0.5 .

The query weight alone needs rank 4 to reach its best, and stays three points lower.

How small the rank can be



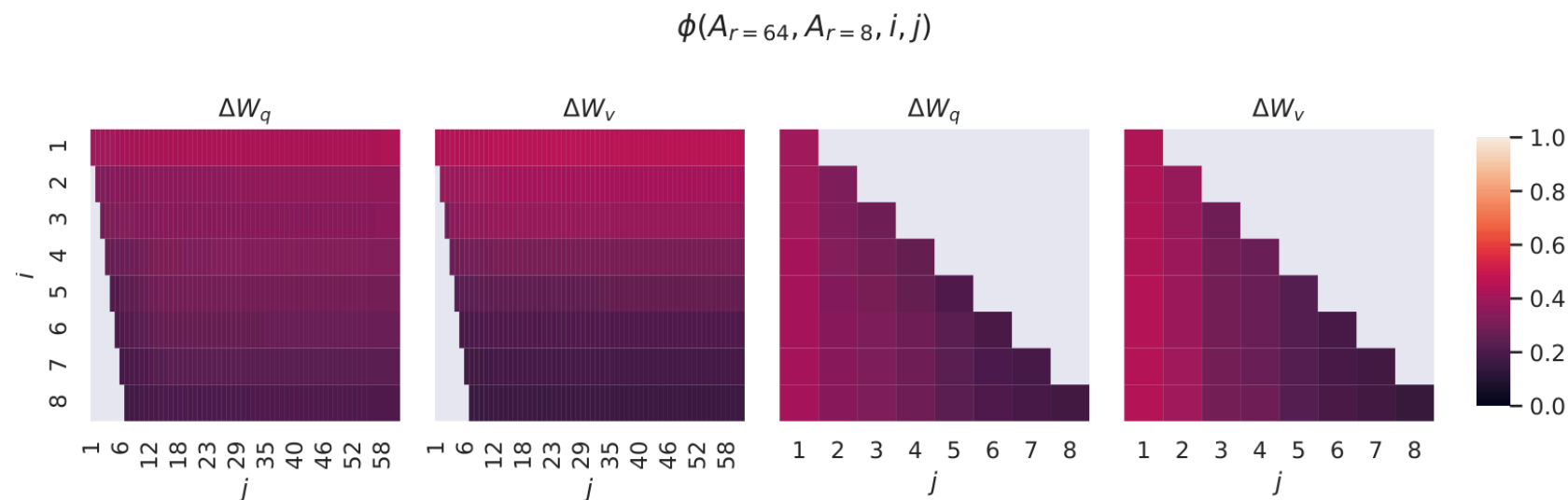
On MNL, too, rank 1 for query and value is within 0.3 points of the best rank.

So the change fine-tuning needs has very few real directions, at least on GPT-3.

Why a small rank is enough

Rank 8 against rank 64

The change against the weight



The authors trained rank 8 and rank 64 separately and compared the directions each learned. Bright means the two runs found the same direction. Only the first direction is clearly shared, with similarity above 0.5, and the rest overlap little, so most of the extra rank holds noise.

Why a small rank is enough

Rank 8 against rank 64

The change against the weight

21.67

is how strongly the pretrained weight W_q acts along its own top 4 directions.

0.32

is how strongly it acts along the 4 directions of the learned change ΔW_q : weak, but more than the 0.02 of random directions.

6.91

is the strength of the change along those directions, about 21 times what the pretrained weight had there.

The change does not repeat what the model already does. It strongly amplifies directions the model learned in pretraining but used weakly.

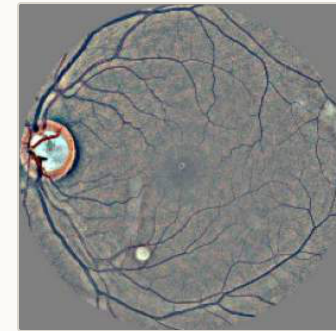
LoRA on images

The paper itself has no image experiments. Vision groups tested it next, first on the same 19 VTAB-1k tasks as VPT, then on medical images.

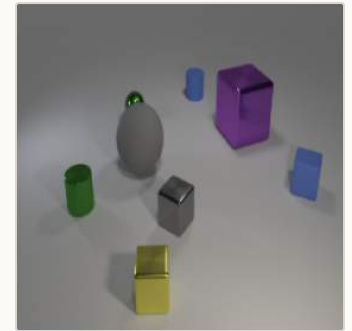
Both use a ViT-B/16, so the numbers sit next to the VPT results.



Natural

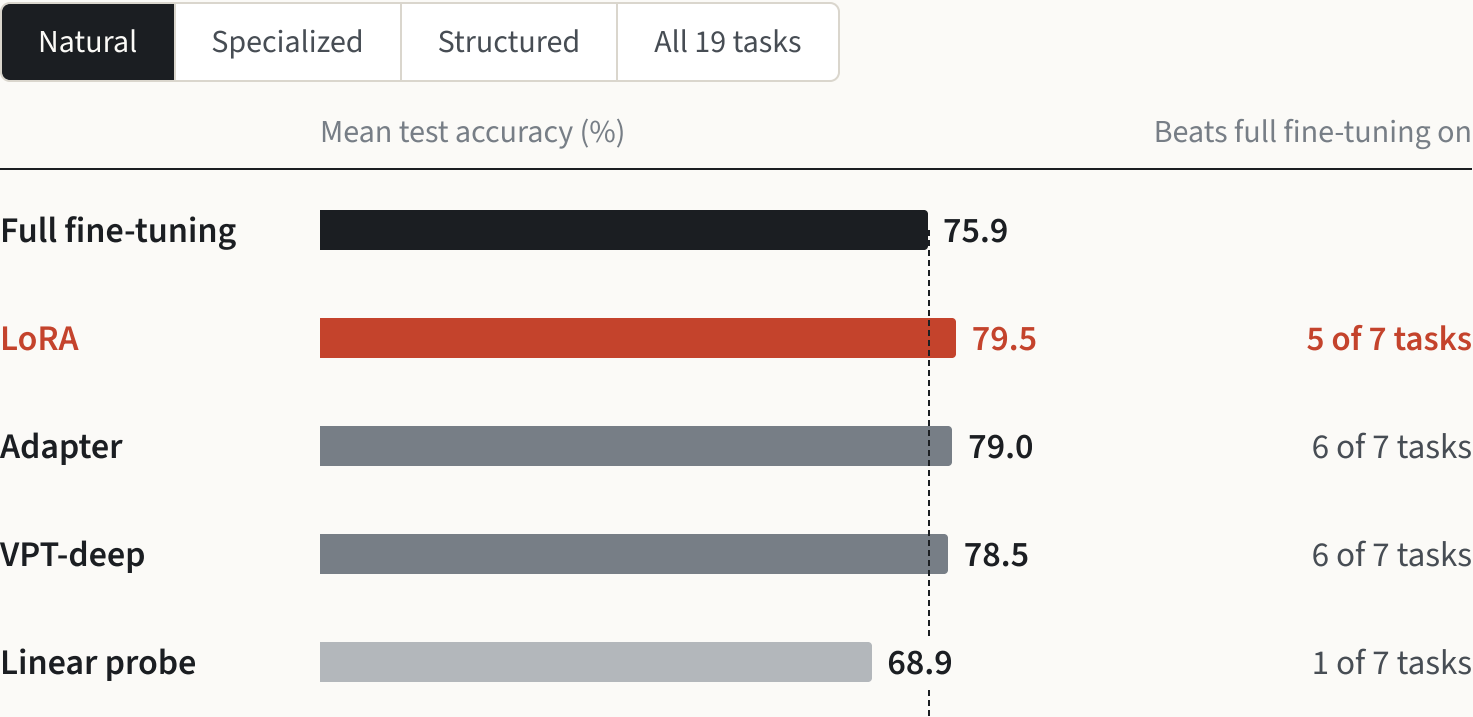


Specialized



Structured

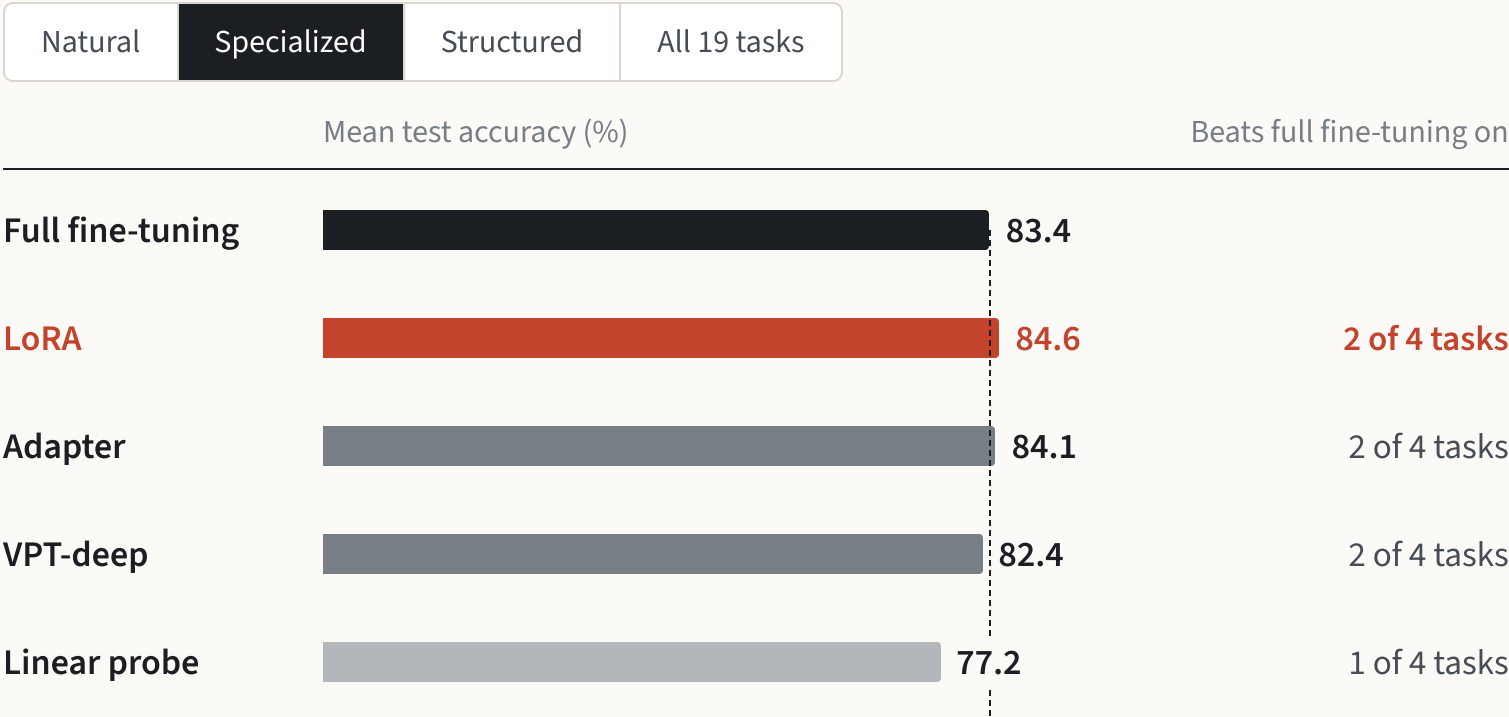
LoRA, VPT and adapters on the same ViT



Ordinary photos, with 1,000 training images per task. LoRA, the adapter and VPT-deep all beat full fine-tuning by two to four points, and LoRA is narrowly the best.

Zhang et al. 2022, Table 1: ViT-B/16 pretrained on ImageNet-21k, VTAB-1k test accuracy. LoRA (rank 8) and the adapter were trained by Zhang et al.; the other rows are Jia et al. 2022, Table 1. Group means computed from the per-task values; "all 19" is the mean of the three groups.

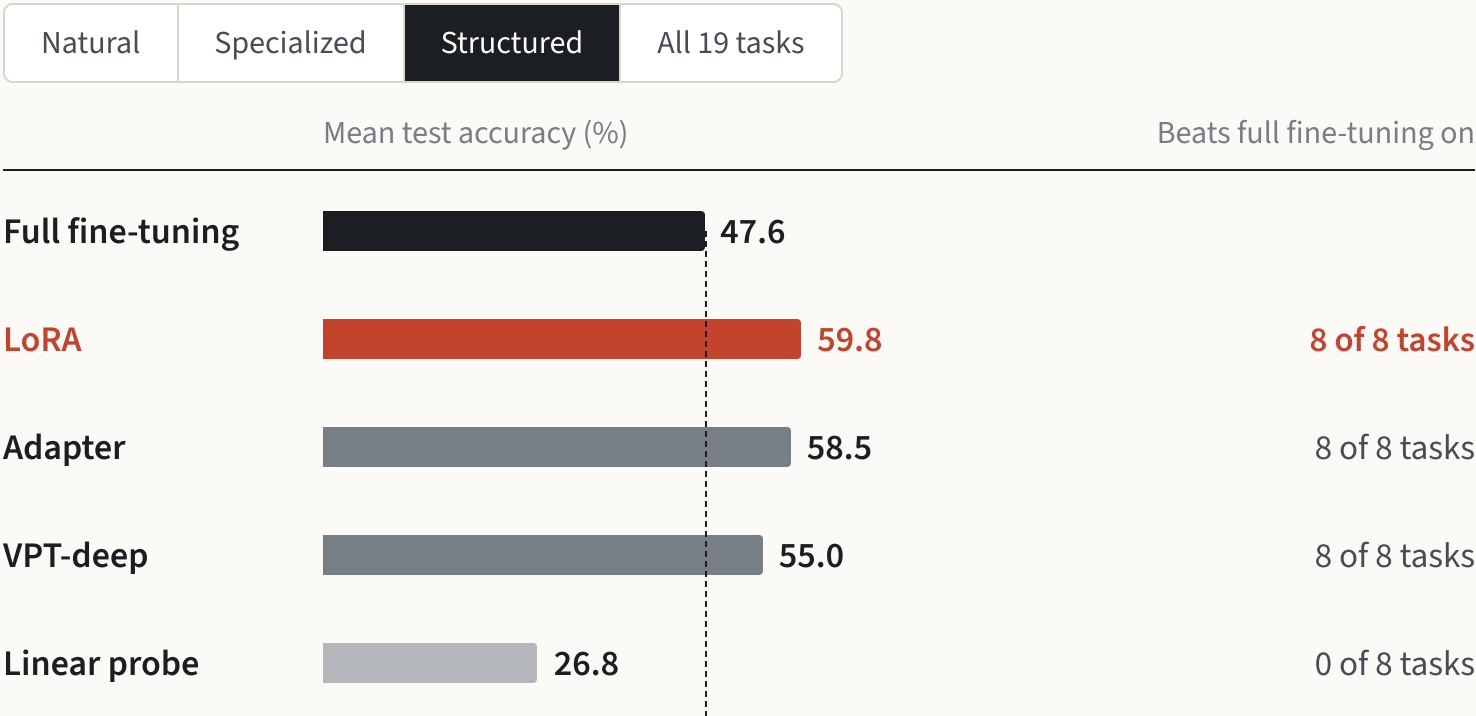
LoRA, VPT and adapters on the same ViT



Satellite, microscope and retina images.
LoRA and the adapter are slightly above full fine-tuning, 84.6 and 84.1 against 83.4, while VPT-deep is a point below.

Zhang et al. 2022, Table 1: ViT-B/16 pretrained on ImageNet-21k, VTAB-1k test accuracy. LoRA (rank 8) and the adapter were trained by Zhang et al.; the other rows are Jia et al. 2022, Table 1. Group means computed from the per-task values; "all 19" is the mean of the three groups.

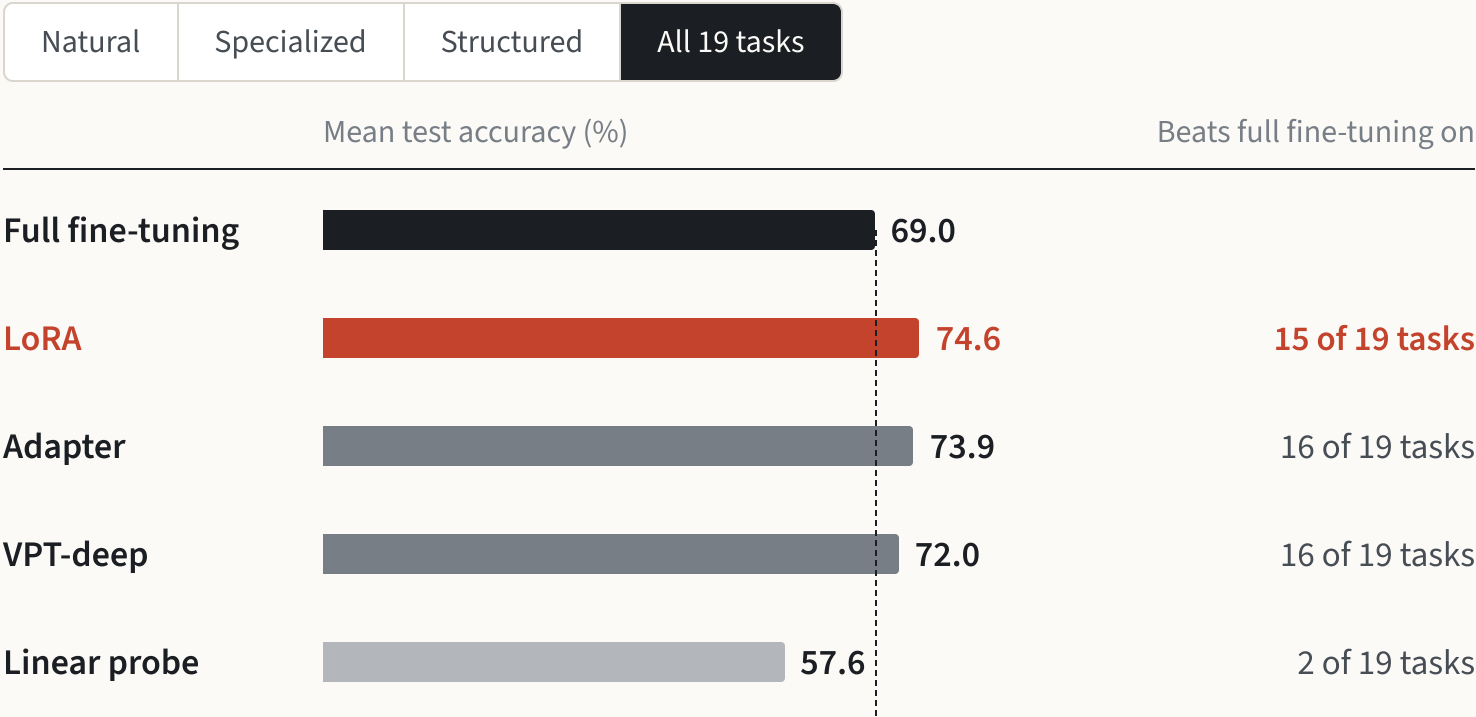
LoRA, VPT and adapters on the same ViT



Counting, distances and angles, far from ImageNet. All three cheap methods beat full fine-tuning on all eight tasks, LoRA by the most: 59.8 against 47.6, with VPT-deep at 55.0.

Zhang et al. 2022, Table 1: ViT-B/16 pretrained on ImageNet-21k, VTAB-1k test accuracy. LoRA (rank 8) and the adapter were trained by Zhang et al.; the other rows are Jia et al. 2022, Table 1. Group means computed from the per-task values; "all 19" is the mean of the three groups.

LoRA, VPT and adapters on the same ViT

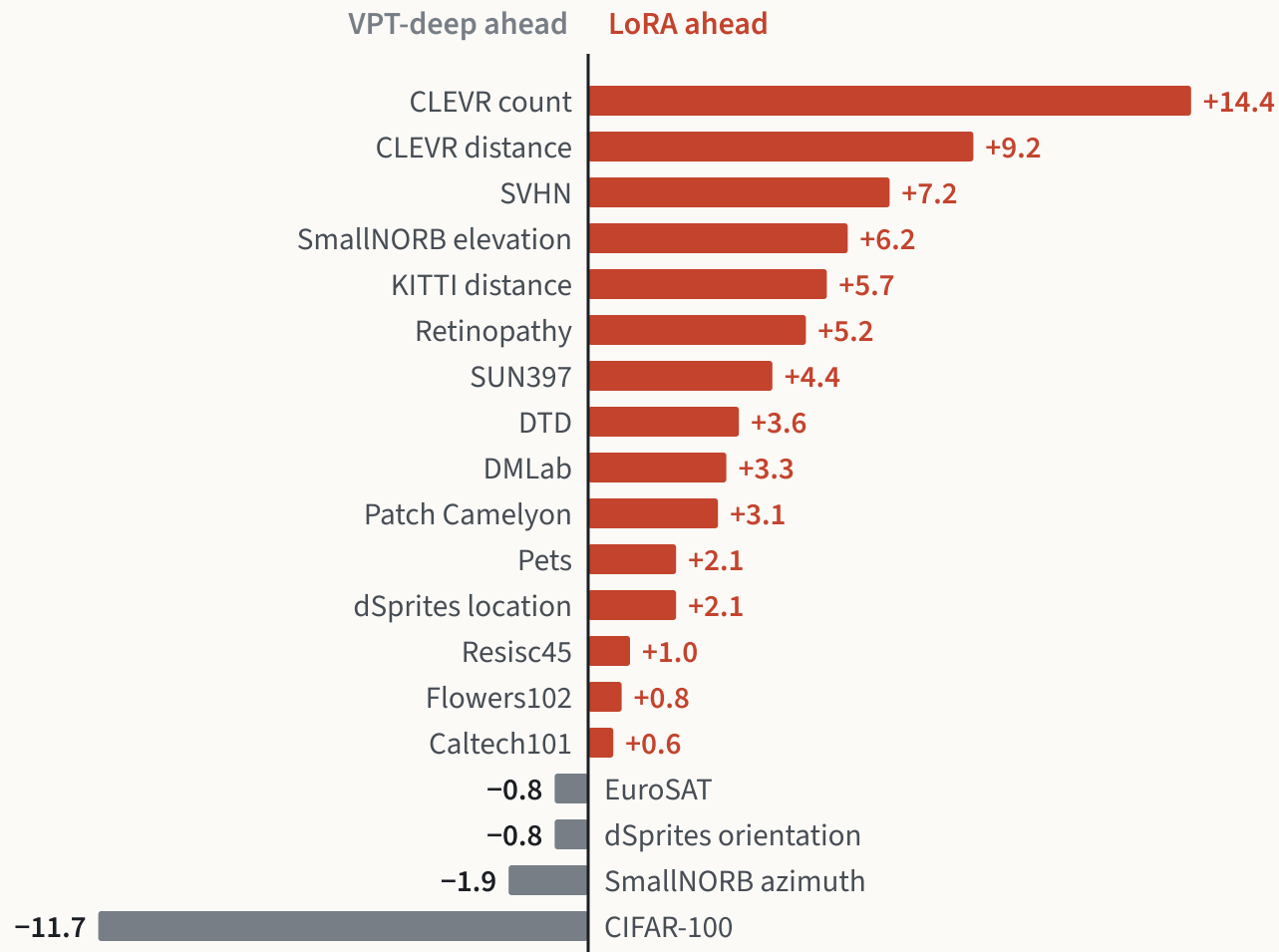


Over all three groups LoRA reaches 74.6, VPT-deep 72.0 and full fine-tuning 69.0.

LoRA trains 0.29 million weights here and VPT-deep 0.64 million on average, so LoRA is both smaller and more accurate.

Zhang et al. 2022, Table 1: ViT-B/16 pretrained on ImageNet-21k, VTAB-1k test accuracy. LoRA (rank 8) and the adapter were trained by Zhang et al.; the other rows are Jia et al. 2022, Table 1. Group means computed from the per-task values; "all 19" is the mean of the three groups.

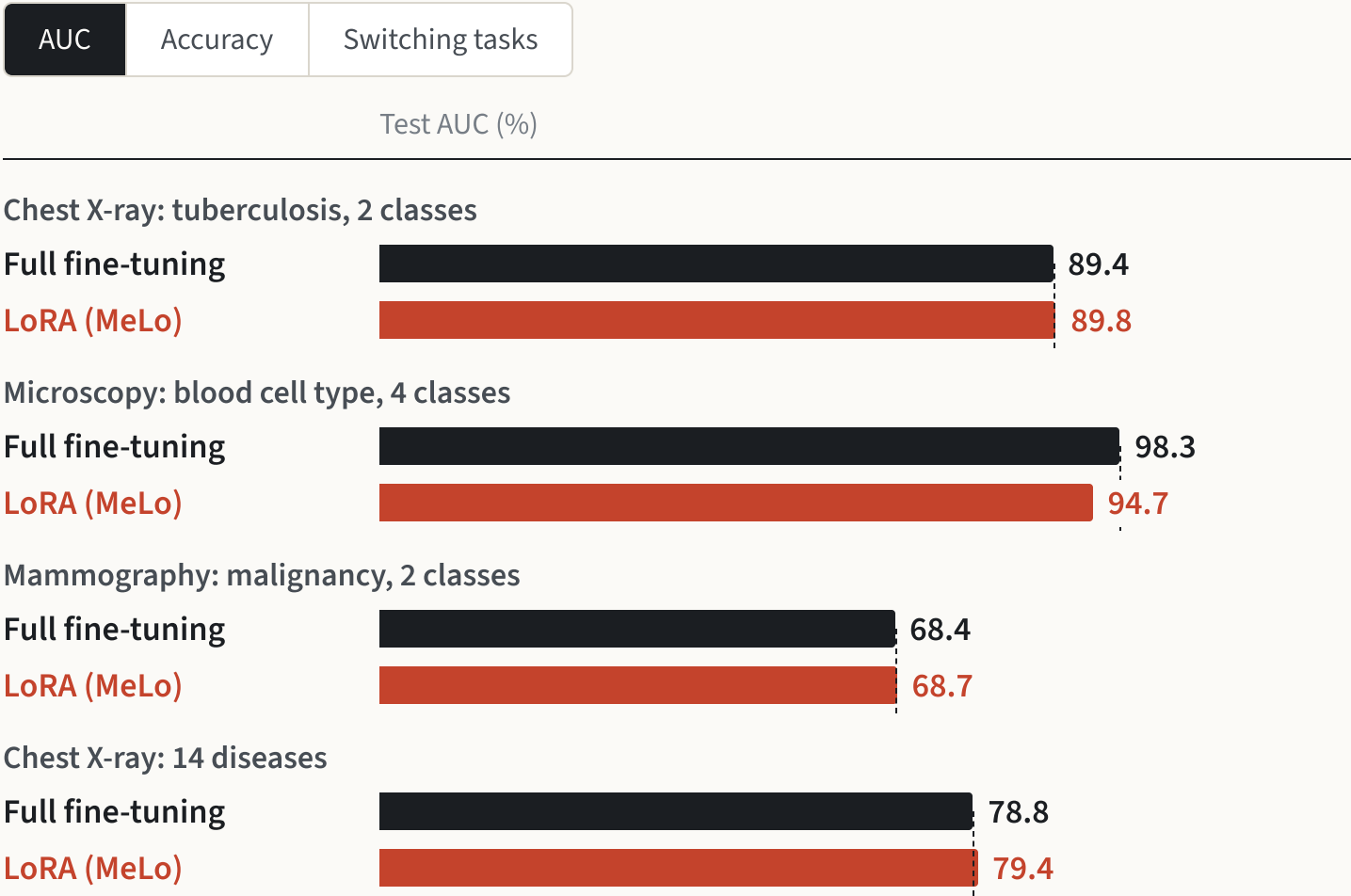
Neither LoRA nor VPT wins every task



LoRA is ahead on 15 of the 19 tasks, by up to 14 points on counting objects in CLEVR. VPT-deep is ahead on 4, and on CIFAR-100 by 12 points.

Zhang et al. conclude that no single method fits every task. Their method, NOAH, searches for a mix of LoRA, adapter and prompts per task, and reaches 75.5 on average.

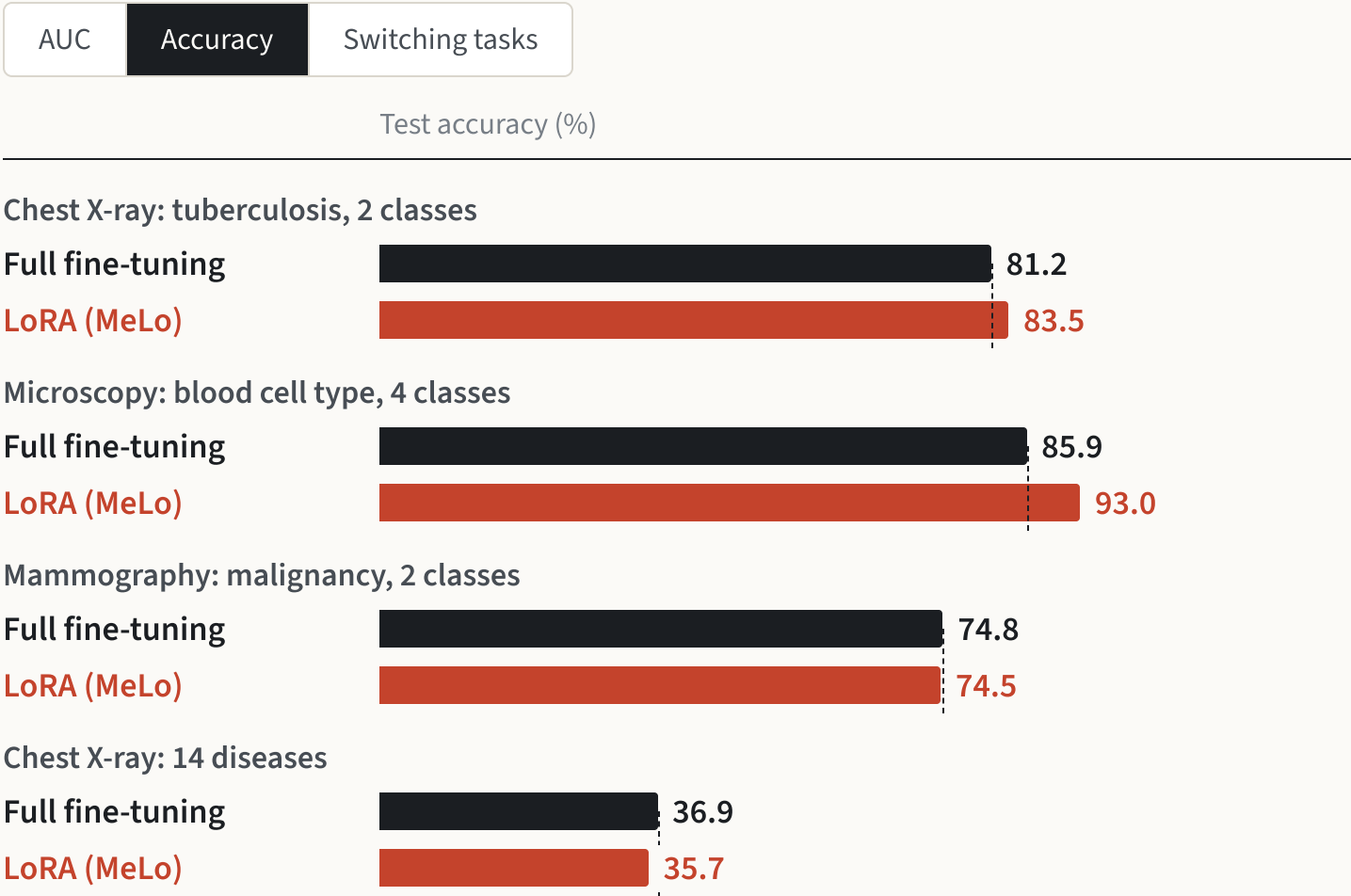
LoRA on medical images



MeLo adds LoRA pairs of rank 4 to the query and value weights of an ImageNet ViT-B, training 0.14 million weights, 0.17% of the model.

Its AUC is within a point of full fine-tuning on three tasks and 3.6 points lower on blood cells.

LoRA on medical images



In accuracy MeLo is ahead on two tasks and slightly behind on two.

These are single runs on small test sets, about 80 mammograms for INbreast, so the fair reading of the title, "better than fine-tuning", is "about the same".

LoRA on medical images

	AUC	Accuracy	Switching tasks
	Time for 100 images in random order (s)		Weights in GPU memory
Fine-tuning, reload a copy per image			1,759 million
Fine-tuning, all four copies loaded			7,034 million
LoRA, one model and four pairs			1,760 million

Their practical point is switching between tasks, measured with a giant CLIP ViT on 25 images from each task in random order.

Reloading a fine-tuned copy for each image takes 218 s, and keeping all four copies loaded needs four times the memory. One frozen model with four LoRA pairs takes 10.6 s at the memory of one model.

Where LoRA falls short

The rank and the weights are chosen by hand

The paper picks the query and value weights and a small rank by trying settings, and leaves a principled choice to future work. On VTAB-1k the best method even changes from task to task.

A small rank assumes a nearby task

The authors expect a small rank to fail when the new task is far from pretraining, such as a different language. The low-rank analysis was only done on GPT-3, not on vision models.

Merged weights serve one task at a time

Once BA is added into W , every input in a batch must belong to that task. Mixing tasks in one batch means keeping the pairs separate, which brings back a small extra cost; VPT's prompts mix freely.

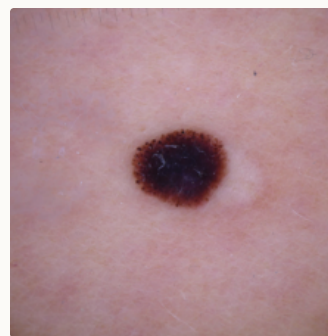
Training still runs through the whole model

LoRA saves the optimizer state of the frozen weights, but the gradient still passes back through every layer. On ViT-B most memory is activations: freezing the backbone for VPT only cut it from 11.7 to 10.3 GB.

My experiments

I re-implemented VPT and LoRA and trained both on three medical imaging datasets from MedMNIST. A linear probe and full fine-tuning ran next to them, all four starting from the same ViT-B/16.

First the results, then all four models running live in this browser.



Skin lesions



Blood cells



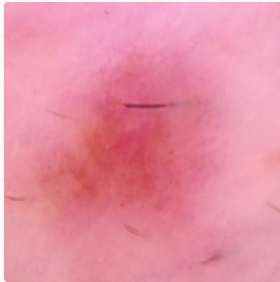
Chest X-rays

I tested VPT and LoRA on three medical datasets

Each method was trained twice: on 1,000 images, as in VTAB-1k, and on the full training set.



melanoma

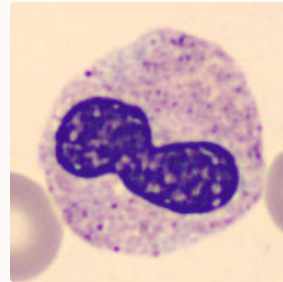


nevus

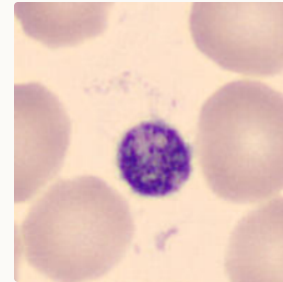
Skin lesions

Dermoscopy, 7 classes


1,000 or all 7,007 training images



neutrophil



platelet

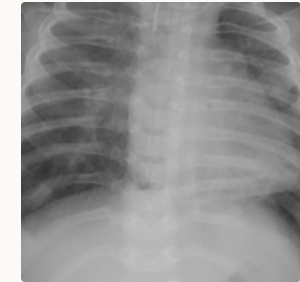
Blood cells

Microscopy, 8 cell types


1,000 or all 11,959 training images



normal



pneumonia

Chest X-rays

Radiographs, pneumonia or normal

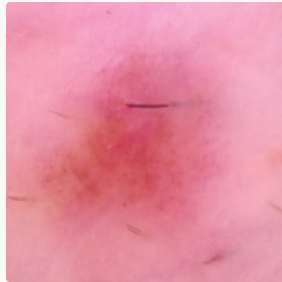

1,000 or all 4,708 training images

I tested VPT and LoRA on three medical datasets

Each method was trained twice: on 1,000 images, as in VTAB-1k, and on the full training set.



melanoma

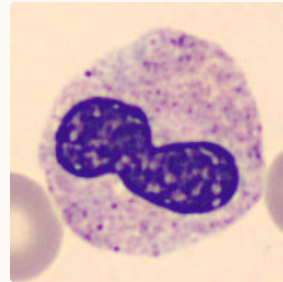


nevus

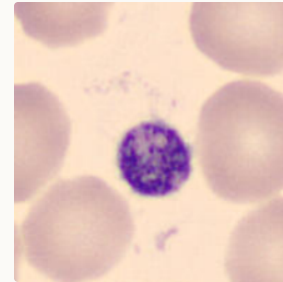
Skin lesions

Dermoscopy, 7 classes


1,000 or all 7,007 training images



neutrophil



platelet

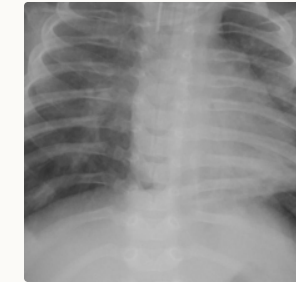
Blood cells

Microscopy, 8 cell types


1,000 or all 11,959 training images



normal



pneumonia

Chest X-rays

Radiographs, pneumonia or normal


1,000 or all 4,708 training images



Linear probe

Trains only a new head.



VPT-deep

Trains a head and 50 prompts per layer.



LoRA

Trains a head and rank-8 pairs beside W_q and W_v .

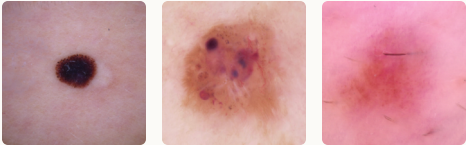
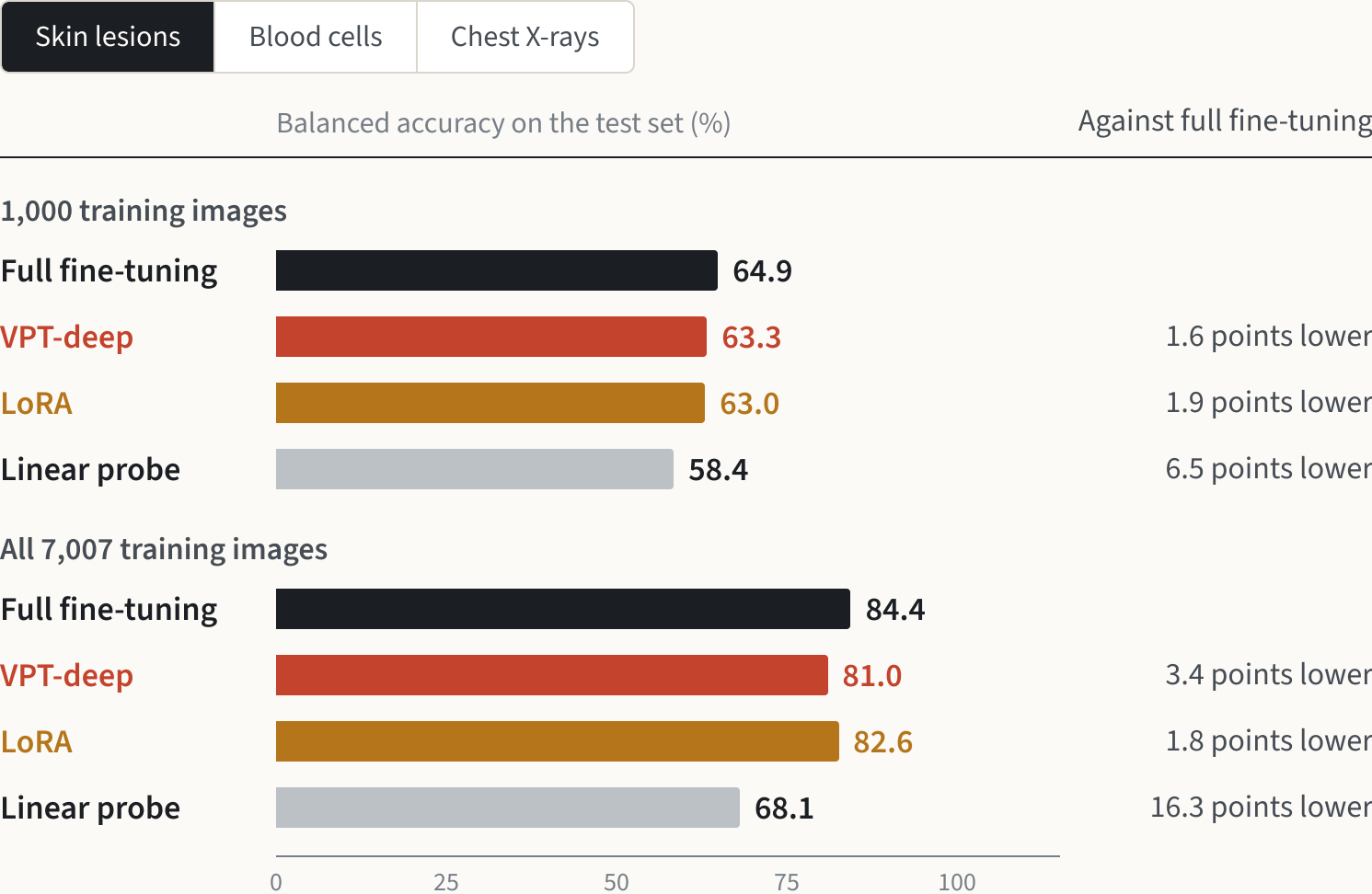


Full fine-tuning

Retrains all 85.8 million weights.

MedMNIST+ at 224×224 (Yang et al. 2023); skin lesions from HAM10000 (Tschandl et al. 2018). RTX 5070 Ti laptop GPU. SGD for linear probe and VPT, AdamW for LoRA and full fine-tuning; learning rate picked on validation.

VPT and LoRA on my three datasets



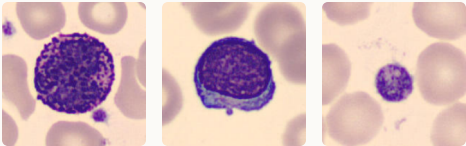
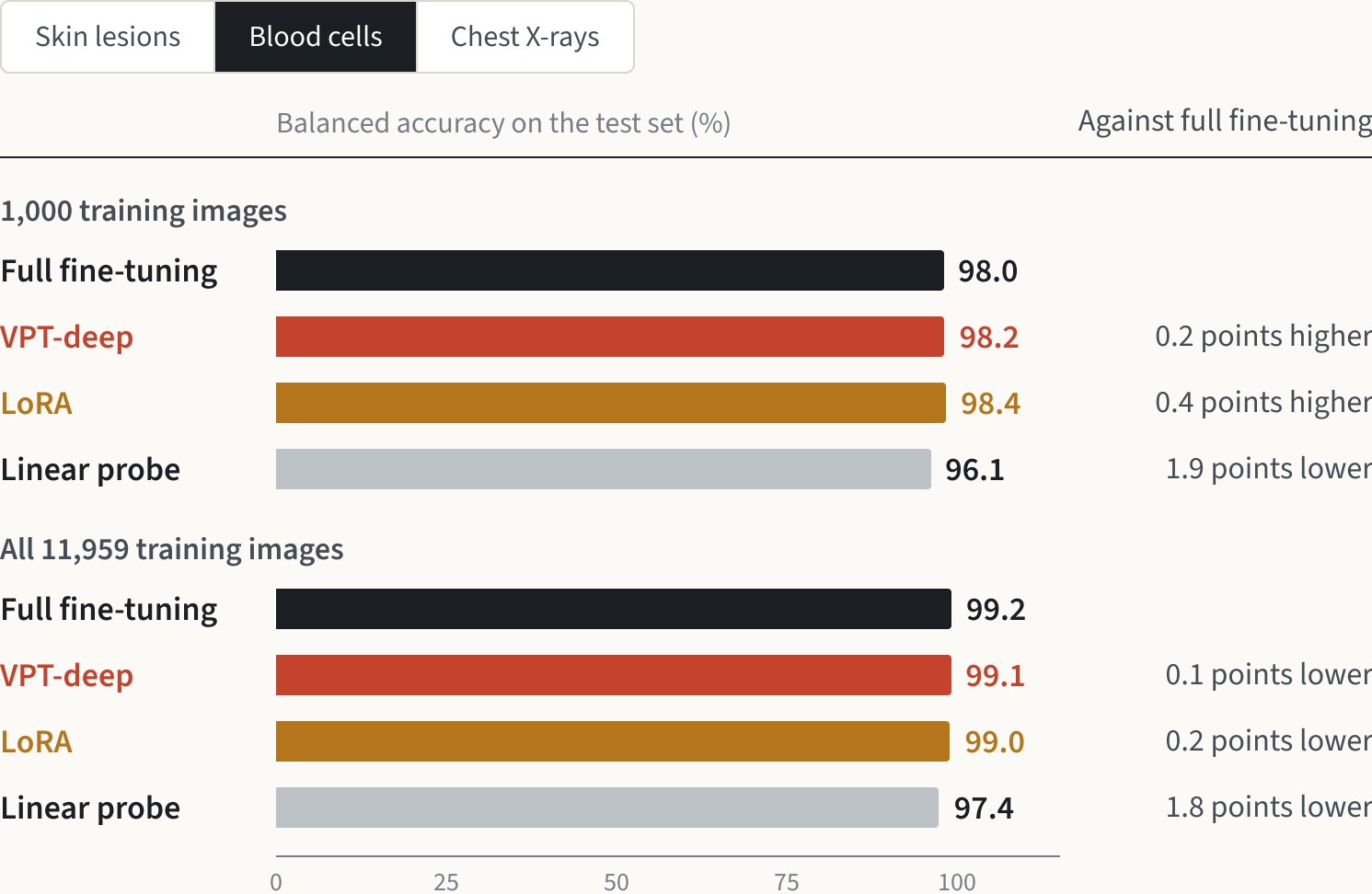
Seven kinds of skin lesion, and two thirds of the images show harmless moles. This is the hardest of the three tasks.

With all 7,007 images, LoRA is 1.6 points ahead of VPT-deep and 1.8 points behind full fine-tuning.

With 1,000 images, LoRA is 0.3 points behind VPT-deep and 1.9 points behind full fine-tuning.

My runs: ViT-B/16 (ImageNet-21k), MedMNIST+ 224 × 224, balanced accuracy on the test set (the mean per-class recall), bars start at 0. Each method was trained once with 1,000 images and once with the full training set; there are no points in between.

VPT and LoRA on my three datasets



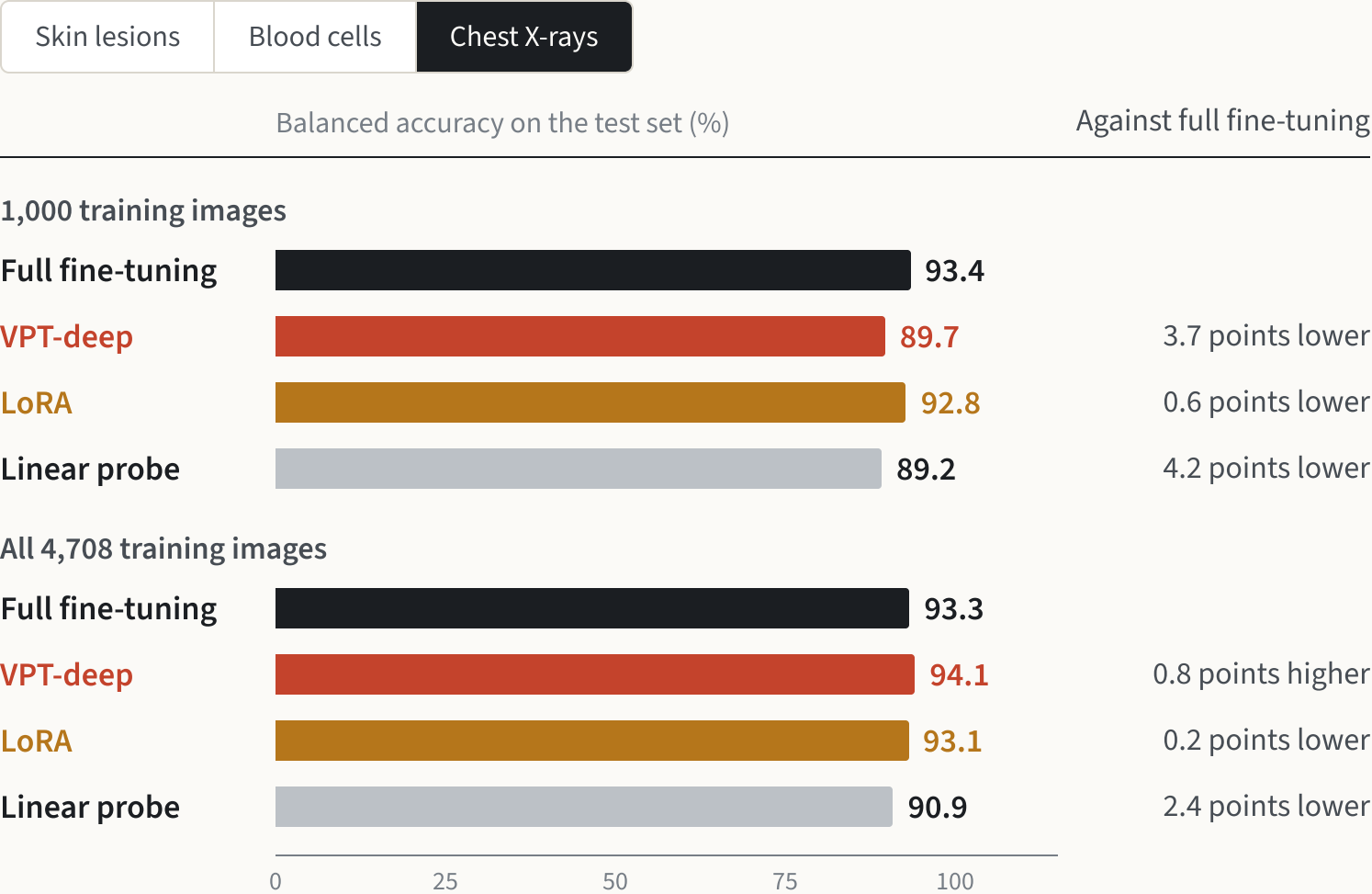
Eight kinds of blood cell under a microscope. Every method does well here, so the bars are almost the same length.

With all 11,959 images, LoRA is 0.1 points behind VPT-deep and 0.2 points behind full fine-tuning.

With 1,000 images, LoRA is 0.2 points ahead of VPT-deep and 0.4 points ahead of full fine-tuning.

My runs: ViT-B/16 (ImageNet-21k), MedMNIST+ 224 × 224, balanced accuracy on the test set (the mean per-class recall), bars start at 0. Each method was trained once with 1,000 images and once with the full training set; there are no points in between.

VPT and LoRA on my three datasets



Chest X-rays of children, labelled normal or pneumonia.

With all 4,708 images, LoRA is 1.0 points behind VPT-deep and 0.2 points behind full fine-tuning.

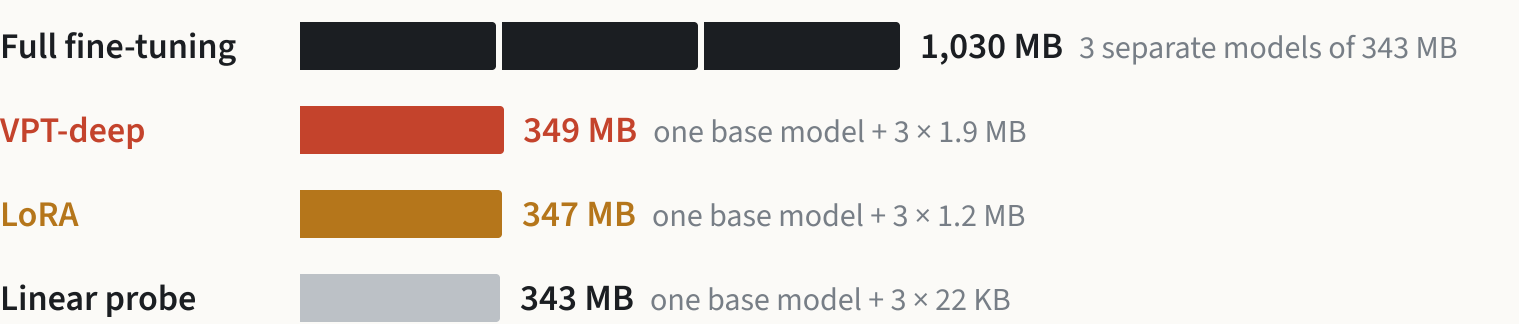
With 1,000 images, LoRA is 3.1 points ahead of VPT-deep and 0.6 points behind full fine-tuning.

My runs: ViT-B/16 (ImageNet-21k), MedMNIST+ 224 × 224, balanced accuracy on the test set (the mean per-class recall), bars start at 0. Each method was trained once with 1,000 images and once with the full training set; there are no points in between.

What each method costs on skin lesions

Storage and accuracy	Training	Inference
----------------------	----------	-----------

Stored for all three tasks



Balanced accuracy on the test set



LoRA stores 1.2 MB per task and VPT-deep 1.9 MB, against 343 MB for a fine-tuned copy. Both share one frozen base model across all tasks.

Here LoRA reaches 82.6% and VPT-deep 81.0%, against 84.4% for full fine-tuning and 68.1% for the linear probe.

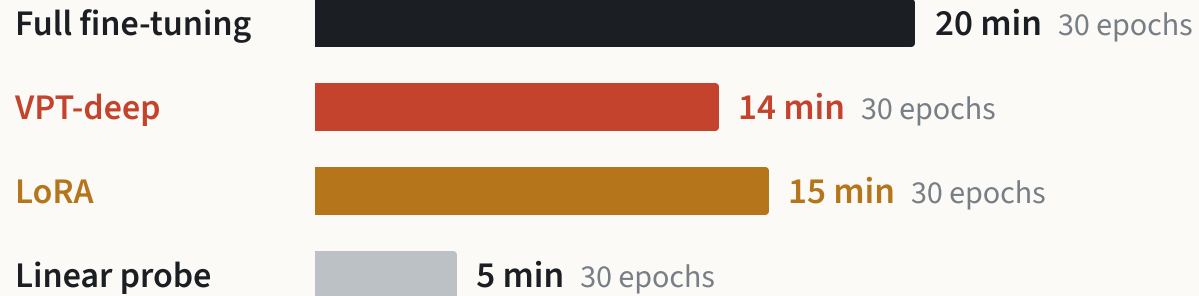
What each method costs on skin lesions

Storage and accuracy

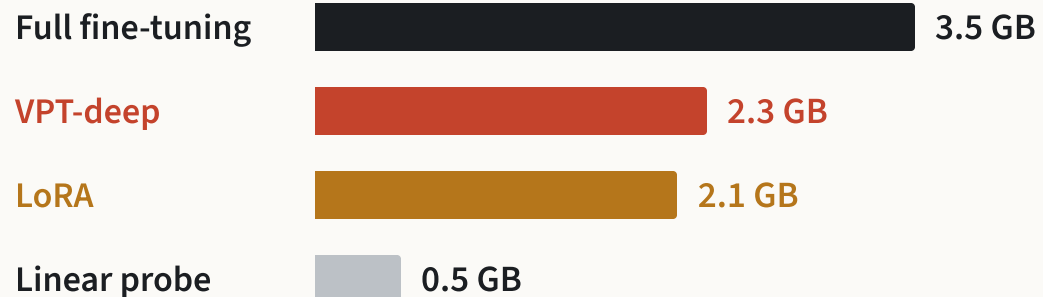
Training

Inference

Training time on all 7,007 images



Peak GPU memory during training, batch 32



LoRA needed 2.1 GB of GPU memory and VPT-deep 2.3 GB, against 3.5 GB for full fine-tuning.

Both still pass the error signal back through all twelve frozen layers, so only the linear probe is really cheap to train.

What each method costs on skin lesions

Storage and accuracy

Training

Inference

Time per image on the CPU, 2 threads



Tokens that every layer processes



VPT-deep needs 250 ms per image, because every layer processes 50 extra tokens. LoRA with separate pairs needs 190 ms.

Merged into the weights, LoRA takes 188 ms, the time of the unchanged model (189 ms).

Live demo: one image, four models

Skin lesion, one test image, models trained on **all training images**.



True label: **melanoma**

Linear probe

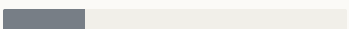
shared model + head

68.5% test accuracy

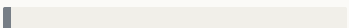
melanocytic n... **73.7%**



melanoma **23.8%**



benign keratosi... **2.1%**



Wrong

178 ms per image

VPT-deep

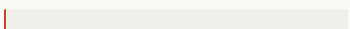
shared model + prompts

81.0% test accuracy

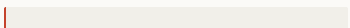
melanoma **99.5%**



benign keratosi... **0.3%**



melanocytic nevi **0.1%**



Correct

204 ms per image

LoRA

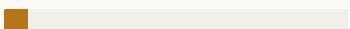
shared model + pairs

82.5% test accuracy

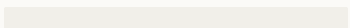
melanocytic n... **92.8%**



melanoma **7.1%**



benign keratosi... **0.0%**



Wrong

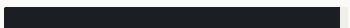
173 ms per image

Full fine-tuning

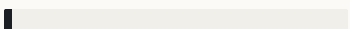
own retrained model

84.4% test accuracy

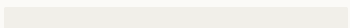
melanoma **97.7%**



melanocytic nevi **2.2%**



benign keratosi... **0.0%**



Correct

163 ms per image

Test accuracy is balanced accuracy on the whole test set, for exactly these models. Changing the training set swaps one 1.2 MB file for LoRA, but a whole second 172 MB model for full fine-tuning.

Live demo: one image, four models

Skin lesion, one test image, models trained on 1,000 training images.



True label: **melanoma**

Linear probe

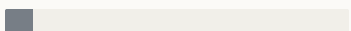
shared model + head

58.1% test accuracy

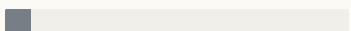
melanocytic n... 84.5%



benign keratosi... 7.9%



melanoma 7.5%



Wrong

142 ms per image

VPT-deep

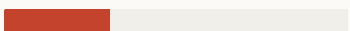
shared model + prompts

62.5% test accuracy

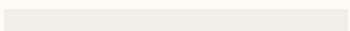
melanoma 69.2%



melanocytic n... 30.8%



dermatofibroma 0.0%



Correct

218 ms per image

LoRA

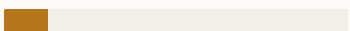
shared model + pairs

63.0% test accuracy

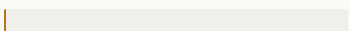
melanocytic n... 86.6%



melanoma 12.8%



benign keratosi... 0.4%



Wrong

166 ms per image

Full fine-tuning

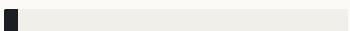
own retrained model

64.8% test accuracy

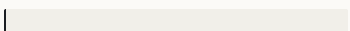
melanocytic n... 95.1%



melanoma 4.2%



benign keratosi... 0.2%



Wrong

169 ms per image

Test accuracy is balanced accuracy on the whole test set, for exactly these models. Changing the training set swaps one 1.2 MB file for LoRA, but a whole second 172 MB model for full fine-tuning.

Summary

20 of 24

tasks where VPT-deep beat full fine-tuning, training 0.6% of a ViT-B pretrained with labels.

99.4%

less storage per task than a fully fine-tuned copy: 2.2 MB for VPT instead of 343 MB, and 1.5 MB for LoRA.

1.00×

the test-time cost of the original model with LoRA, because its change is added into the weights. VPT's prompts cost 1.27×

74.6

mean VTAB-1k accuracy of LoRA on ViT-B/16, against 72.0 for VPT-deep and 69.0 for full fine-tuning.

88.1%

average balanced accuracy of LoRA on my three medical datasets, against 87.6% for VPT-deep, 88.8% for full fine-tuning and 83.3% for the linear probe.

References

The two papers

Jia, Tang, Chen, Cardie, Belongie, Hariharan & Lim 2022. *Visual prompt tuning*. ECCV. arXiv:2203.12119.

Hu, Shen, Wallis, Allen-Zhu, Li, Wang, Wang & Chen 2022. *LoRA: low-rank adaptation of large language models*. ICLR. arXiv:2106.09685.

Prompting before VPT

Lester, Al-Rfou & Constant 2021. *The power of scale for parameter-efficient prompt tuning*. EMNLP.

Li & Liang 2021. *Prefix-tuning: optimizing continuous prompts for generation*. ACL.

Zhou, Yang, Loy & Liu 2022. *Learning to prompt for vision-language models (CoOp)*. IJCV. arXiv:2109.01134.

Bahng, Jahanian, Sankaranarayanan & Isola 2022. *Exploring visual prompts for adapting large-scale models*. arXiv:2203.17274.

Ideas LoRA builds on

Li, Farkhoor, Liu & Yosinski 2018. *Measuring the intrinsic dimension of objective landscapes*. ICLR.

Aghajanyan, Zettlemoyer & Gupta 2021. *Intrinsic dimensionality explains the effectiveness of language model fine-tuning*. ACL.

Other parameter-efficient methods

Houlsby et al. 2019. *Parameter-efficient transfer learning for NLP (adapter)*. ICML.

Lin, Madotto & Fung 2020. *Exploring versatile generative language model via parameter-efficient transfer learning*. Findings of EMNLP.

Rücklé et al. 2021. *AdapterDrop: on the efficiency of adapters in Transformers*. EMNLP.

Ben Zaken, Goldberg & Ravfogel 2022. *BitFit: simple parameter-efficient fine-tuning for Transformer-based masked language-models (Bias)*. ACL.

Zhang, Sax, Zamir, Guibas & Malik 2020. *Side-tuning: a baseline for network adaptation via additive side networks (Sidetune)*. ECCV.

LoRA on images

Zhang, Zhou & Liu 2022. *Neural prompt search (NOAH)*. arXiv:2206.04673.

Zhu et al. 2024. *MeLo: low-rank adaptation is better than fine-tuning for medical image diagnosis*. ISBI. arXiv:2311.08236.

References, continued

Models

- Dosovitskiy et al. 2021. *An image is worth 16×16 words: Transformers for image recognition at scale* (ViT). ICLR.
- Liu et al. 2021. *Swin Transformer: hierarchical vision Transformer using shifted windows*. ICCV.
- Chen, Xie & He 2021. *An empirical study of training self-supervised vision Transformers* (MoCo v3). ICCV.
- He et al. 2022. *Masked autoencoders are scalable vision learners* (MAE). CVPR.
- Zheng et al. 2021. *Rethinking semantic segmentation from a sequence-to-sequence perspective with Transformers* (SETR). CVPR.
- Chen et al. 2018. *Encoder-decoder with atrous separable convolution for semantic image segmentation* (DeepLab v3+). ECCV.
- Radford et al. 2021. *Learning transferable visual models from natural language supervision* (CLIP). ICML.
- Brown et al. 2020. *Language models are few-shot learners* (GPT-3). NeurIPS.
- Radford et al. 2019. *Language models are unsupervised multitask learners* (GPT-2). OpenAI report.
- Liu et al. 2019. *RoBERTa: a robustly optimized BERT pretraining approach*. arXiv:1907.11692.

Datasets and benchmarks

- Zhai et al. 2019. *A large-scale study of representation learning with the visual task adaptation benchmark* (VTAB). arXiv:1910.04867.
- Parkhi, Vedaldi, Zisserman & Jawahar 2012. *Cats and dogs* (Oxford Pets). CVPR.
- Zhou et al. 2019. *Semantic understanding of scenes through the ADE20K dataset*. IJCV.
- Zhong, Xiong & Socher 2017. *Seq2SQL: generating structured queries from natural language using reinforcement learning* (WikiSQL). arXiv:1709.00103.
- Williams, Nangia & Bowman 2018. *A broad-coverage challenge corpus for sentence understanding through inference* (MNLI). NAACL.
- Wang et al. 2019. *GLUE: a multi-task benchmark and analysis platform for natural language understanding*. ICLR.
- Jaeger et al. 2014; Moreira et al. 2012; Wang et al. 2017. Shenzhen chest X-ray, INbreast and ChestX-ray14, used by MeLo.
- Yang et al. 2023. *MedMNIST v2: a large-scale lightweight benchmark for 2D and 3D biomedical image classification*. Scientific Data.
- Tschandl, Rosendahl & Kittler 2018. *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*. Scientific Data.