THE WEB CONFERENCE ACM
WWW '26 • Dubai, UAE • June 29 –July 3, 2026

Beyond Fine-Tuning

The Present and Future of Parameter-Efficient Fine-Tuning in Vision Transformers

Edwin Kwadwo Tenagyei • Lei Wang • Yongsheng Gao
Griffith University, Australia

Why parameter-efficient fine-tuning matters

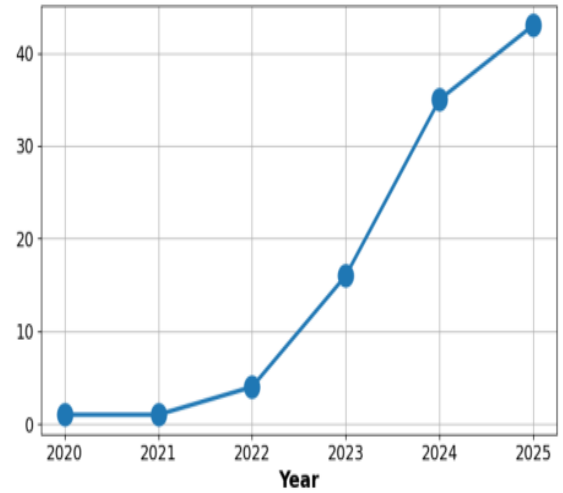
ViTs are powerful, but full fine-tuning is expensive to repeat across tasks.

Full fine-tuning

Updates the whole backbone
High GPU memory
High training time
One copy per task

PEFT

Freeze most weights
Train small task modules
Store lightweight adapters
Scale to many tasks



Research activity sharply increases from 2020–2025, motivating a structured synthesis of PEFT for ViTs.

Main message: PEFT changes the adaptation problem from “train everything” to “train the right small part.”

What this survey contributes

A focused review of PEFT techniques for Vision Transformers in image classification.

1. Taxonomy

We organize PEFT for ViTs into additive, reparameterization, selective, hybrid, and inference-efficient tuning.

2. Evaluation synthesis

We compare benchmark protocols, VTAB-1k results, and accuracy–efficiency trade-offs across representative methods.

3. Future agenda

We identify open directions: robustness, input-adaptive PEFT, theory, and deployment-aware benchmarking.

Scope: Vision Transformers • image classification • practical adaptation under resource constraints

From ViT fine-tuning to PEFT

The model is pretrained once, then adapted with a much smaller task-specific update.

Vision Transformer

Patch tokens + [CLS]
Self-attention layers
MLP blocks

Full tuning

Optimize all θ
Large gradients
Large storage

PEFT objective

Optimize $\Delta\theta$ only
with $|\Delta\theta| \ll |\theta|$
Backbone mostly frozen

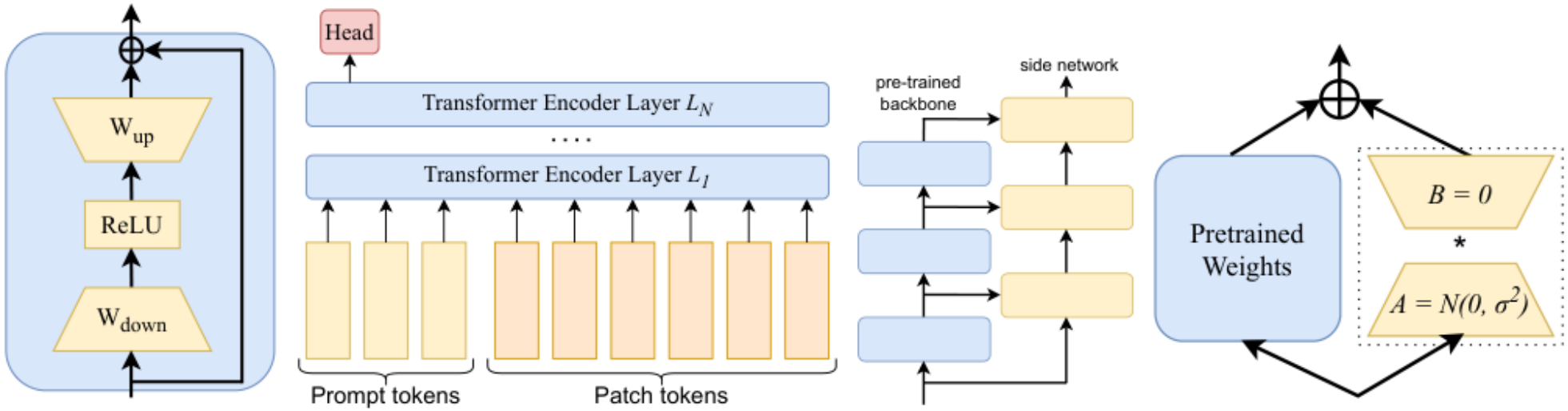
Key optimization idea

minimize task loss by updating a small trainable subset, while preserving pretrained visual representations

small update $\Delta\theta$ + frozen backbone θ_0 → task model

A taxonomy of PEFT for Vision Transformers

Five families describe where and how adaptation occurs.



(a) Adapter Tuning (b) Prompt Tuning (c) Side Tuning (d) Reparameterization Tuning

Additive **Reparameterization** **Selective** **Hybrid** **Inference-efficient**

The taxonomy separates methods by whether they add modules/tokens, transform weights, tune selected parameters, combine mechanisms, or reduce inference cost.

Additive-based tuning

Adapt by adding lightweight trainable components to a frozen ViT.

Adapters

Bottleneck modules are inserted into transformer layers; only the adapter weights are trained.

Prompts

Learnable visual prompt tokens are concatenated with patch embeddings or inserted across layers.

Prefixes

Trainable key/value prefix matrices modify multi-head self-attention.

Side tuning

A small auxiliary network learns task features while the backbone remains frozen.

Strength: flexible representation adaptation. Limitation: extra modules can add inference latency.

Reparameterization and Selective tuning

Two different routes to smaller updates.

Reparameterization-based tuning

Example: LoRA factorizes updates as low-rank matrices.

$$W' = W + \Delta W,\ \Delta W = BA$$

Goal: preserve expressiveness while training a compact parameterization.

Selective tuning

Example: linear probe, BitFit, LayerNorm tuning, salient channels.

Goal: update only the most relevant parameters.

Trade-off: extreme efficiency, but reduced expressive capacity.

Takeaway: reparameterization often offers stronger accuracy–efficiency balance; selective tuning is valuable when parameter budgets are extremely tight.

Hybrid and inference-efficient tuning

Recent work moves from parameter efficiency toward deployment efficiency.

Hybrid methods

Combine complementary mechanisms such as adapters, LoRA, and prompts. Examples include NOAH, U-Tuning, and unified PEFT analyses. These approaches can improve robustness across different task types.

Inference-efficient methods

Reduce latency and memory during deployment through structural reparameterization, token reduction, or token merging. These methods target the gap between low trainable parameters and real inference speed.

Key distinction: fewer trainable parameters do not automatically mean faster inference.

Evaluation datasets and protocol issues

PEFT comparisons depend heavily on benchmark choice and reporting practice.

FGVC

CUB-200-2011, NABirds, Oxford Flowers, Stanford Dogs, Stanford Cars. Useful for fine-grained adaptation, but often small and homogeneous.

VTAB-1k

Natural, specialized, and structured tasks with 1,000 labeled examples each. Strong for standardized low-data evaluation.

Robustness / scale

ImageNet variants and domain-shift benchmarks are less frequently reported, limiting understanding of robustness.

Protocol concern: many studies use different backbones, training schedules, hyperparameters, and single-run reporting.

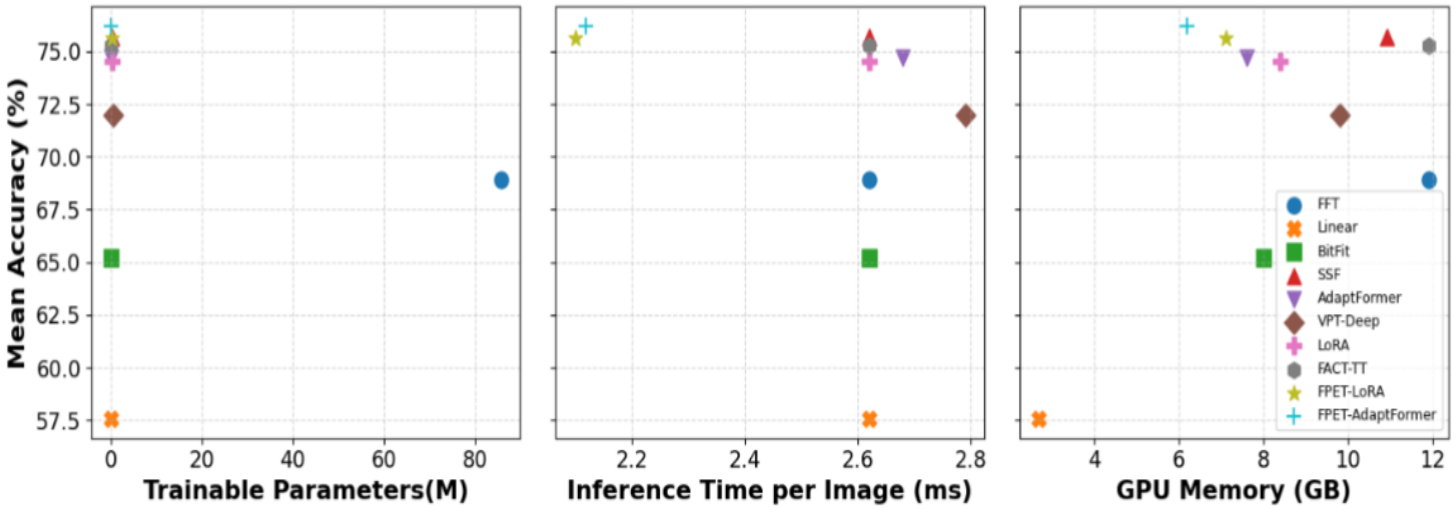
Accuracy–efficiency trade-offs on VTAB-1k

Representative VTAB-1k comparison

	#Param (M)	Natural							Specialized				Structured								Average
		Cifar100	Caltech101	DTD	Flowers102	Pets	SVHN	Sun397	Camelyon	EuroSAT	Resisc45	Retinopathy	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Ele	
Traditional Fine-Tuning																					
Full	85.8	68.9	87.7	64.3	97.2	86.9	87.4	38.8	79.7	95.7	84.2	73.9	56.3	58.6	41.7	65.5	57.5	46.7	25.7	29.1	68.9
Linear Probe[50]	0.04	64.4	85.0	63.2	97.0	86.3	36.6	51.0	78.5	87.5	68.5	74.0	34.3	30.6	33.2	55.4	12.5	20.0	9.6	19.2	57.6
PETL Methods																					
BitFit [116]	0.10	72.8	87.0	59.2	97.5	85.3	59.9	51.4	78.7	91.6	72.9	69.8	61.5	55.6	32.4	55.9	66.6	40.0	15.7	25.1	65.2
GPS [123]	0.25	81.1	94.2	75.8	99.4	91.7	91.6	52.4	87.9	96.2	86.5	76.5	79.9	62.6	55.0	82.4	84.0	55.4	29.7	46.1	77.5
SCT [124]	0.11	75.3	91.6	72.2	99.2	91.1	91.2	55.0	85.0	96.1	86.3	76.2	81.1	65.1	51.7	80.2	75.4	46.2	33.2	45.7	76.0
Adapter [35]	0.16	69.2	90.1	68.0	98.8	89.9	82.8	54.3	84.0	94.9	81.9	75.5	80.9	65.3	48.6	78.3	74.8	48.5	29.9	41.6	73.9
AdaptFormer [9]	0.16	70.8	91.2	70.5	99.1	90.9	86.6	54.8	83.0	95.8	84.4	76.3	81.9	64.3	49.3	80.3	76.3	45.7	31.7	41.1	74.7
Convpass [41]	0.33	72.3	91.2	72.2	99.2	90.9	91.3	54.9	84.2	96.1	85.3	75.6	82.3	67.9	51.3	80.0	85.9	53.1	36.4	44.4	76.6
Bi-AdaptFormer [43]	0.59	74.1	92.4	72.1	99.3	91.6	89.0	56.3	88.2	95.2	86.0	76.2	83.9	63.9	53.0	81.4	86.2	54.8	35.2	41.3	77.0
VPT-Shallow [39]	0.24	77.7	86.9	62.6	97.5	87.3	74.5	51.2	78.2	92.0	75.6	72.9	50.5	58.6	40.5	67.1	68.7	36.1	20.2	34.1	67.8
VPT-Deep [39]	0.53	78.8	90.8	65.8	98.0	88.3	78.1	49.6	81.8	96.1	83.4	68.4	68.5	60.0	46.5	72.8	73.6	47.9	32.9	37.8	72.0
E^2VPT [27]	0.25	78.6	89.4	67.8	98.2	88.5	85.3	52.3	82.5	96.8	84.8	73.6	71.7	61.2	47.9	75.8	80.8	48.1	31.7	41.9	73.9
LoRA [36]	0.29	67.1	91.4	69.4	98.8	90.4	85.3	54.0	84.9	95.3	84.4	73.6	82.9	69.2	49.8	78.5	75.7	47.1	31.0	44.0	74.5
Bi-LoRA [43]	1.18	72.1	91.7	71.2	99.1	91.4	90.2	55.8	87.0	95.4	85.5	75.5	83.1	64.1	52.2	81.2	86.4	53.5	36.7	44.4	76.7
KARST [129]	0.33	76.8	93.2	75.1	99.3	92.2	91.9	57.6	88.3	96.2	88.4	75.7	83.8	69.0	52.9	82.0	86.0	52.9	33.8	47.0	78.1
FacT-TT [42]	0.04	71.3	89.6	70.7	98.9	91.0	87.8	54.6	85.2	95.5	83.4	75.7	82.0	69.0	49.8	80.0	79.2	48.4	34.2	41.4	75.3
FacT-TK [42]	0.01	70.6	90.6	70.8	99.1	90.7	88.6	54.1	84.8	96.2	84.5	75.7	82.6	68.2	49.8	80.7	80.8	47.4	33.2	43.0	75.6
OFT [77]	0.15	68.8	91.9	73.8	99.7	92.2	91.8	49.2	90.2	100	89.1	80.5	83.2	71.1	53.9	81.3	82.0	54.3	34.4	43.8	78.0
GOFT [61]	0.02	75.0	93.9	72.3	99.7	92.6	85.2	60.9	89.1	100	87.9	82.4	84.0	74.2	55.1	82.0	80.9	52.7	32.3	43.8	78.6
Hydra[48]	0.28	72.7	91.3	72.0	99.2	91.4	90.7	55.5	85.8	96.0	86.1	75.9	83.2	68.2	50.9	82.3	80.3	50.8	34.5	43.1	76.5
SSF[55]	0.24	69.0	92.6	75.1	99.4	91.8	90.2	52.9	87.4	95.9	87.4	75.5	75.9	62.3	53.3	80.6	77.3	54.9	29.5	37.9	75.7
NOAH [122]	0.36	69.6	92.7	70.2	99.1	90.4	86.1	53.7	84.4	95.4	83.9	75.8	82.8	68.9	49.9	81.7	81.8	48.3	32.8	44.2	75.5
RepAdapter [60]	0.22	69.0	92.6	75.1	99.4	91.8	90.2	52.9	87.4	95.9	87.4	75.5	75.9	62.3	53.3	80.6	77.3	54.9	29.5	37.9	76.1
LoSA [66]	0.19	82.5	92.8	76.1	99.7	90.5	82.0	55.8	86.6	97.1	87.0	76.7	81.5	62.3	48.6	82.1	94.2	61.7	47.9	45.6	78.4
DTL [22]	0.05	74.1	94.8	71.8	99.4	91.7	90.4	57.2	87.9	96.7	87.5	74.8	81.9	64.7	51.5	81.9	93.9	54.0	35.6	50.3	76.7
LAST [89]	0.66	66.7	93.4	76.1	99.6	89.8	86.1	54.3	86.2	96.3	86.8	75.4	81.9	65.9	49.4	82.6	87.9	46.7	32.3	51.5	76.5
DyT[125]	0.16	74.4	95.5	73.6	99.2	91.7	87.5	57.4	88.3	96.1	86.7	76.7	83.5	63.5	52.9	83.1	85.7	54.9	34.3	45.9	77.6
FPET-RepAdapter[46]	0.23	72.1	91.5	71.8	99.3	90.7	90.3	55.0	85.2	96.2	84.5	75.6	82.2	67.7	49.7	79.9	82.2	48.7	36.9	41.7	76.1
FPET-LoRA[46]	0.30	70.1	92.7	69.4	99.1	90.8	85.4	55.6	87.2	94.6	82.5	74.1	83.0	63.4	50.6	81.6	84.7	51.5	34.3	43.3	75.6
FPET-AdaptFormer[46]	0.17	71.3	93.5	69.9	99.3	90.7	87.0	54.7	87.5	95.1	84.5	76.2	83.6	63.1	52.2	81.3	87.1	54.1	33.5	40.2	76.2
FPET-Bi-LoRA[46]	1.18	71.9	91.1	70.9	99.1	90.5	89.4	55.9	87.4	94.7	84.4	74.9	83.5	65.1	52.1	79.7	85.8	54.2	36.7	44.4	76.4
FPET-Bi-AdaptFormer[46]	0.64	74.1	92.8	72.5	99.4	91.1	89.6	56.2	88.3	94.9	86.3	75.3	83.8	63.0	52.8	81.4	85.7	54.4	35.9	42.2	77.0

Accuracy–efficiency trade-offs on VTAB-1k

No single PEFT family dominates every task and constraint.



What the trade-off shows

- Full fine-tuning is parameter-heavy
- PEFT reaches comparable accuracy
- Low parameters ≠ zero latency
- Memory efficiency improves strongly

Main conclusion: method choice should be application-specific—accuracy, trainable parameters, memory, and inference cost must be considered together.

Strengths and weaknesses of PEFT

PEFT is a trade-off space, not a single universal solution.

Strengths

Competitive accuracy with fewer trainable parameters
Lower memory use and training cost
Scalable storage for multi-task settings
Regularization benefits in low-data regimes

Weaknesses

Sensitive to rank, prompt length, and insertion points
Not uniformly strong across datasets
Some methods add inference latency
Limited theoretical understanding

The design goal is not “smallest update,” but the best balance for the target deployment scenario.

Future directions and closing message

The future of PEFT should be robust, adaptive, principled, and deployable.

1. **Structure-aware adaptation** — Use token graphs, locality, and semantic relations.
2. **Dynamic PEFT** — Activate modules or ranks based on input difficulty and token importance.
3. **Robustness** — Evaluate under distribution shift, scarcity, and perturbations.
4. **Theory** — Explain expressivity, optimization, and generalization.
5. **Deployment benchmarks** — Report accuracy, memory, latency, and compute together.

Closing: PEFT makes large Vision Transformers practically adaptable, but the next frontier is deployment-aware PEFT.

Thank you

Questions?