# SHADOWPEFT: SHADOW NETWORK FOR PARAMETER EFFICIENT FINE-TUNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Popular low-rank parameter-efficient fine-tuning (PEFT) methods represent adaptation as separate updates to selected backbone weights, without maintaining an explicit task-specific state that is updated and reused across depth. These updates also require the backbone at inference and therefore cannot operate as standalone predictors. We propose ShadowPEFT, a stateful, detachable alternative to weight-space adaptation that consolidates trainable adaptation into a modular shadow component centered on a compact shadow model and lightweight Transformer layer-specific coupling modules. A persistent shadow state refines the frozen backbone representations and is updated from them in an interactive manner. Because the shadow model is trained as a complete predictor, it can be detached for shadow-only inference without executing the base model and can be initialized from a pretrained model. Experiments on text and image generation and understanding benchmarks show that ShadowPEFT matches or outperforms LoRA and DoRA under comparable trainable-parameter budgets. Additional analyses on shadow pretraining, cross-dataset transfer, parameter scaling, inference latency, and system-level evaluation suggest that centralized layer-space adaptation is a competitive and flexible alternative to conventional low-rank PEFT. The code is available at: `http://anonymous.4open.science/r/shadow-peft-submission-BA3A`.

## 1 INTRODUCTION

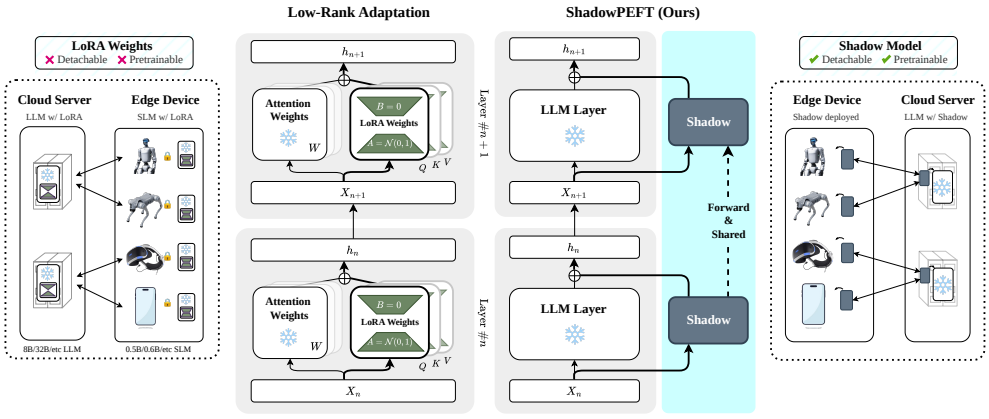



Figure 1: Comparison between (Left) conventional LoRA configuration and (Right) our proposed ShadowPEFT. LoRA-style methods adapt the backbone through linear-specific low-rank weight updates, whereas ShadowPEFT carries a persistent shadow state through depth. The shadow state can be attached to the backbone or detached for lightweight deployment.

Parameter-efficient fine-tuning (PEFT) trains a small set of task-specific parameters while keeping most backbone parameters frozen to make large-scale model adaptation computationally feasible (Han et al., 2024). Representative approaches include prompt and prefix tuning (Li & Liang, 2021),



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

adapter modules (Houlsby et al., 2019), and low-rank methods such as LoRA (Hu et al., 2022), QLoRA (Dettmers et al., 2023), and DoRA (Liu et al., 2024). LoRA-style methods are especially popular for large language models because they are simple and broadly compatible. They encode adaptation, however, as separate local modifications to selected backbone weights. Although backbone hidden representations naturally propagate through depth, the adaptation mechanism itself does not maintain a separate and decoupled task-specific state that is explicitly carried, updated, and reused across layers. Cross-layer coordination of adaptation therefore emerges only indirectly through the backbone computation, while the trainable updates remain distributed across layers and dependent on the backbone at inference. This motivates a complementary question: **can task adaptation be accompanied by an explicit depth-wise state that coordinates layer representations?**

We propose **ShadowPEFT**, which consolidates trainable adaptation into a modular shadow component coupled to a frozen backbone. The component combines a compact shadow model (architecturally similar to the base model but scaled down) with lightweight, layer-specific injection and update modules. The shadow model initializes and maintains a task state; at each Transformer layer, this state refines the backbone representation, and the resulting backbone state updates the shadow state for the next layer. This interactive coupling mechanism gives later refinements access to both accumulated shadow and the latest backbone representation, allowing more effective task adaptation through progressively coordinated representation learning across depth.

This design offers two complementary capabilities. First, as the shadow model is trained as a functional predictor, it can be detached and deployed without the base model. This feature supports deployment patterns such as edge–cloud collaboration, in which the compact shadow model handles simple requests locally and delegates more complex requests to an attached cloud model. Furthermore, since the shadow module is a self-contained functional model, it can be initialized from another pretrained model. For example, a smaller model such as Qwen3 0.6B can serve as the shadow model for its larger backbone Qwen3 8B, enabling the larger model to be adapted by training its smaller counterpart. In this configuration, the smaller model learns to refine the representations of the larger one, allowing adaptation capacity to be reused across model scales. This expands PEFT beyond lightweight parameter injection toward reusable and cross-scale adaptation dynamics.

We evaluate ShadowPEFT on six text and image generation and understanding benchmarks. Extensive experiments show that ShadowPEFT matches the performance of LoRA and DoRA under comparable trainable-parameter budgets, demonstrates high robustness in parameter scaling experiments, and achieves strong generalization ability. Additionally, its flexible design supports modular deployment modes that conventional low-rank PEFT methods do not naturally offer. Our contributions are as follows:

1. We introduce ShadowPEFT, a modular representation-space PEFT framework that organizes adaptation around a persistent shadow state and couples it to a frozen backbone through shadow-to-base injection and base-to-shadow gated updates.

2. The detachable shadow design supports both attached refinement and shadow-only inference without separate training, enabling edge–cloud collaboration. Notably, it reduces system latency by up to 80% compared to cloud-only PEFT baselines while maintaining competitive accuracy.

3. Across multiple benchmarks and backbone scales, ShadowPEFT performs competitively with LoRA and DoRA under similar trainable-parameter budgets, while exhibiting superior robustness and stronger out-of-distribution transfer capabilities.

## 2 Related Work

**Revisit parameter-efficient fine-tuning.** PEFT adapts pretrained models while updating only a small fraction of their parameters. Prompt Tuning (Lester et al., 2021), Prefix Tuning (Li & Liang, 2021), and P-Tuning (Liu et al., 2022) optimize continuous prompts while leaving the backbone unchanged. Adapter methods instead insert trainable bottleneck transformations into selected Transformer layers (Houlsby et al., 2019). Subsequent work improves adapter composition, transfer, or parameter sharing, including AdapterSoup (Chronopoulou et al., 2023), Tiny-Attention Adapter (Zhao et al., 2022), and Compacter (Karimi Mahabadi et al., 2021). Standard adapters generally use layer-specific parameters, for instance, Compacter shares hypercomplex "slow" factors while retaining layer-specific rank-one factors (Karimi Mahabadi et al., 2021); parallel adapters (He et al., 2021)

and ReFT (Wu et al., 2024) intervene on representations independently per layer. ShadowPEFT does not tie its injection and update parameters across depth. Instead, successive layers are connected by an evolving shadow state, making state persistence—rather than parameter sharing—the source of depth-wise coordination.

**Low-rank adaptation.** LoRA (Hu et al., 2022) freezes pretrained weights and learns low-rank updates for selected linear projections, substantially reducing the number of trainable parameters. Extensions improve rank allocation (AdaLoRA (Zhang et al., 2023) and DyLoRA (Valipour et al., 2023)), initialization (LoRA-GA (Wang et al., 2024), PiSSA (Meng et al., 2024), and OLoRA (Büyükakyüz, 2024)), quantization (QLoRA (Dettmers et al., 2023) and QA-LoRA (Xu et al., 2023)), and multi-task or compositional adaptation (Che et al., 2026; Li et al., 2024; Yang et al., 2025b; Hsu et al., 2024). These methods are effective, but their trainable adaptation remains distributed across layer-local weight updates. Although backbone activations propagate across layers, the adaptation mechanism does not explicitly maintain a task-specific state that is updated and reused through depth. ShadowPEFT instead organizes adaptation around a persistent state operating on layer-level representations, while retaining lightweight layer-specific coupling transformations.

**Side networks and two-way cross-pathway coupling.** Side-Tuning trains a lightweight side network and additively fuses its output with that of a frozen base network (Zhang et al., 2020). Ladder Side-Tuning (LST) exposes intermediate frozen-backbone activations to a separate ladder network through shortcut connections, enabling training without back-propagating through the backbone (Sung et al., 2022). These methods show that a trainable side network can efficiently use frozen backbone features. ShadowPEFT is related in maintaining an auxiliary path, but its data flows both ways at each layer: shadow injects into base, and base gates updates back to shadow. This creates a dependency chain instead of parallel execution, and unlike LST, it doesn't aim to avoid backprop through the base. The key difference is this coupled state transition, not just having a side network.

## 3 Our ShadowPEFT Framework

We introduce **ShadowPEFT**, a PEFT framework that consolidates trainable task adaptation into a modular *shadow component* coupled to a frozen pretrained LLM (the *base model*). Unlike LoRA-style methods (Figure 1(left)), ShadowPEFT organizes adaptation around a persistent *shadow state* carried through depth (Figure 1(right)).

### 3.1 ShadowPEFT Overview

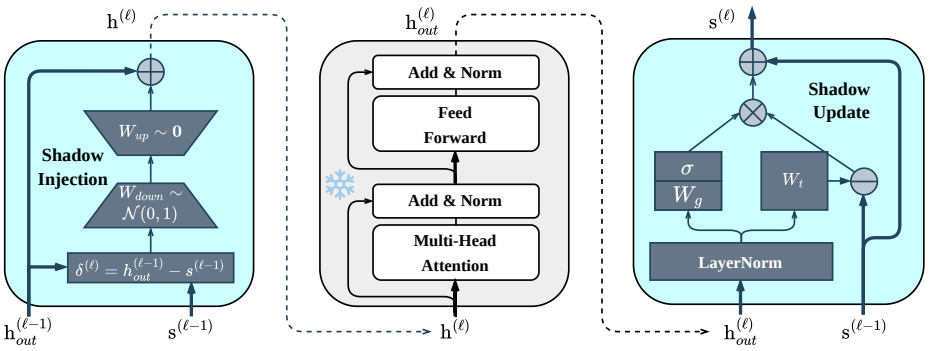
Each layer of ShadowPEFT consists of three operations: shadow injection, frozen base encoding, and shadow update. Let the frozen base consist of $L$ Transformer decoder layers. Given an input sequence $\mathbf{x}$, let $\mathbf{h}_{out}^{(\ell)} \in \mathbb{R}^{T\times d}$ denote the hidden state of the $\ell$-th LLM decoder layer (base layer), where $T$ is the sequence token length and $d$ is the hidden dimension. The *shadow state* $\mathbf{s}^{(\ell)} \in \mathbb{R}^{T\times d}$ is the persistent, task-adaptive state at depth $\ell$. The initial shadow state $\mathbf{s}^{(0)}$ is produced by a *shadow model* $f_{\text{shadow}}$:

$$\mathbf{s}^{(0)} = f_{\text{shadow}}(\mathbf{x};\, \theta_{\text{shadow}})\,. \tag{1}$$

For each base layer $\ell \geq 1$, ShadowPEFT repeats three operations. First, shadow injection uses the current shadow state $\mathbf{s}^{(\ell-1)}$ to refine the previous base hidden state $\mathbf{h}_{out}^{(\ell-1)}$. Second, the frozen base layer $f_{\text{base}}^{(\ell)}$ encodes the refined representation and produces $\mathbf{h}_{out}^{(\ell)}$. Third, the shadow update module advances the shadow state from $\mathbf{s}^{(\ell-1)}$ to $\mathbf{s}^{(\ell)}$ using the newly produced base representation. These operations are dependency-ordered in a forward pass: injection $\rightarrow$ base encoding $\rightarrow$ update at each depth, rather than concurrently parallel branches. More details about our shadow-backbone designs are provided in Appendix A.

### 3.2 Shadow Injection Module

The *Shadow Injection Module* carries the shadow-to-base direction of the coupling. Its goal is to inject task-specific corrections into the hidden states of the backbone by comparing the current




(a) Shadow Inject (b) Base Encode (c) Shadow Update

Figure 2: Architecture of ShadowPEFT. **(a) Shadow Injection Module.** The discrepancy $\boldsymbol{\delta}^{(\ell)}$ is projected through a low-rank bottleneck ($W_{\text{down}} \sim \mathcal{N}(0, \sigma^2)$, $W_{\text{up}}{=}0$) and added back to the base hidden state. **(b) Base Encoding Module.** The frozen base layer encodes the refined representation. **(c) Shadow Update Module.** The base output $\mathbf{h}_{\text{out}}^{(\ell)}$ is LayerNorm-normalized, then split into a transform $W_t$ and a sigmoid gate $\sigma(W_g)$; their product updates the shadow state via a gated residual.

representation of the backbone with the evolving shadow state. Unlike LoRA-style methods, ShadowPEFT performs adaptation in the transformer decoder layer: the persistent shadow state provides the depth-dependent reference, and each layer-specific injection module learns how to use the base–shadow discrepancy to refine the next backbone-layer input. At base layer $\ell$, the module takes the previous backbone output $\mathbf{h}_{out}^{(\ell-1)}$ and the current shadow state $\mathbf{s}^{(\ell-1)}$ as inputs, as illustrated in Figure 2(a). Since $\mathbf{s}^{(\ell-1)}$ has been initialized from the input and progressively updated through earlier layers, it encodes task-relevant information accumulated along the shadow pathway. The discrepancy between the two states therefore provides a compact signal indicating how the frozen backbone representation should be adjusted before entering layer $\ell$:

$$\boldsymbol{\delta}^{(\ell)} = \mathbf{h}_{out}^{(\ell-1)} - \mathbf{s}^{(\ell-1)}. \tag{2}$$

Rather than injecting this raw discrepancy to the backbone directly, ShadowPEFT passes it through a lightweight trainable bottleneck to obtain a refined task-specific correction: $\tilde{\boldsymbol{\delta}}^{(\ell)} = \text{Dropout}\left(\boldsymbol{\delta}^{(\ell)}\mathbf{W}_{\text{down}}^{(\ell)}\right)\mathbf{W}_{\text{up}}^{(\ell)}$, where $\mathbf{W}_{\text{down}}^{(\ell)} \in \mathbb{R}^{d\times r}$ and $\mathbf{W}_{\text{up}}^{(\ell)} \in \mathbb{R}^{r\times d}$ are layer-dependent trainable projection matrices with rank $r \ll d$.[1] This bottleneck preserves parameter efficiency while preventing the model from directly copying the base–shadow difference into the backbone. Instead, it learns which components of the discrepancy are useful for adaptation, making the injected signal a learned refinement rather than a direct overwrite. Finally, the correction is added to the backbone through a residual connection:

$$\mathbf{h}^{(\ell)} = \mathbf{h}_{out}^{(\ell-1)} + \alpha\,\tilde{\boldsymbol{\delta}}^{(\ell)}, \tag{3}$$

where $\alpha > 0$ controls the injection strength. The corrected representation $\mathbf{h}^{(\ell)}$ is then passed into base layer $\ell$ to produce $\mathbf{h}_{out}^{(\ell)}$, as illustrated in Figure 2(b).

### 3.3 Shadow Update Module

The *Shadow Update Module* carries the base-to-shadow direction of the coupling and keeps the shadow state aligned with the frozen backbone as representations evolve across depth. Since the injection module relies on the discrepancy between the hidden state of the backbone and the shadow state, the shadow cannot remain fixed after initialization. Instead, it should absorb useful information from each backbone layer while preserving its accumulated task-specific context. This allows the shadow state to serve as a stable yet adaptive reference for subsequent injections. To update the

[1] Inspired by LoRA (Hu et al., 2022), $\mathbf{W}_{\text{down}}^{(\ell)}$ is initialized from $\mathcal{N}(0,\,\sigma^2)$ and $\mathbf{W}_{\text{up}}^{(\ell)}$ is zero-initialized. This guarantees that $\tilde{\boldsymbol{\delta}}^{(\ell)} = \mathbf{0}$ at initialization, so the frozen base model remains unchanged at start of training.

shadow state after base layer $\ell$, we use a gated residual mechanism, as shown in Figure 2(c), that moves the prior shadow state $\mathbf{s}^{(\ell-1)}$ toward a candidate representation derived from the current backbone output $\mathbf{h}_{\text{out}}^{(\ell)}$:

$$\mathbf{t}^{(\ell)} = T^{(\ell)}\left(\mathbf{h}_{\text{out}}^{(\ell)}\right), \tag{4}$$

$$\mathbf{g}^{(\ell)} = \sigma\left(G^{(\ell)}\left(\mathbf{h}_{\text{out}}^{(\ell)}\right)\right), \tag{5}$$

$$\mathbf{s}^{(\ell)} = \left(1 - \mathbf{g}^{(\ell)}\right) \odot \mathbf{s}^{(\ell-1)} + \mathbf{g}^{(\ell)} \odot \mathbf{t}^{(\ell)}, \tag{6}$$

where $\odot$ denotes element-wise multiplication and $\sigma$ is the sigmoid function. Here, $T^{(\ell)}(\cdot)$ maps the current backbone output into a candidate shadow representation, while $G^{(\ell)}(\cdot)$ predicts an element-wise gate that controls how much new information is incorporated into the shadow state. The implementations of $T^{(\ell)}(\cdot)$ and $G^{(\ell)}(\cdot)$ are provided in Appendix B. This update can be viewed as an interpolation between the previous shadow state and the current layer-induced candidate state. The gate controls the balance between retaining information accumulated from earlier layers and incorporating information from the current backbone layer. Similar to GRU-style updates (Cho et al., 2014), it helps prevent abrupt state replacement and reduces the risk of shadow collapse.

### 3.4 Base-Shadow Joint Training

ShadowPEFT is trained end-to-end with a joint loss while keeping the base model frozen. The trainable components include the shadow model, the injection projections, the update networks, and the shadow prediction head. For **causal language modeling**, given shifted target labels $\mathbf{y}$, the training loss is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{CE}}(\mathbf{h}_{\text{base}}\,\mathbf{W}_{\text{lm}},\ \mathbf{y}) + \lambda\,\mathcal{L}_{\text{CE}}\left(\mathbf{s}^{(0)}\,\mathbf{W}_{\text{shadow}},\ \mathbf{y}\right), \tag{7}$$

where $\mathbf{h}_{\text{base}}$ is the final hidden state of the injected base model, $\mathbf{W}_{\text{lm}}$ and $\mathbf{W}_{\text{shadow}}$ are the base and shadow language-model heads, $\mathcal{L}_{\text{CE}}$ is the standard next-token cross-entropy loss, and $\lambda$ is a scalar hyperparameter (default $\lambda = 0.05$). For **sequence classification** with $C$ classes, following Li et al. (2026), the shadow hidden state $\mathbf{s}^{(L)} \in \mathbb{R}^{B\times T\times d}$ is reduced to a pooled vector. This pooled representation is then scored by a shadow classifier head $\mathbf{W}_{\text{cls}}^{\text{shadow}} \in \mathbb{R}^{d\times C}$, initialized as a copy of the base model's classifier head. The joint classification loss is:

$$\mathcal{L}_{\text{CLS}} = \mathcal{L}_{\text{CE}}(\mathbf{h}_{\text{base}}\,\mathbf{W}_{\text{cls}},\ y) + \lambda\,\mathcal{L}_{\text{CE}}\left(\bar{\mathbf{s}}^{(L)}\,\mathbf{W}_{\text{cls}}^{\text{shadow}},\ y\right), \tag{8}$$

where $y \in \{1, \ldots, C\}$ is the ground-truth class label. The shadow loss can be interpreted as a regularizer, detailed in the Appendix C. At inference time, ShadowPEFT supports two deployment configurations: 1) ***Shadow-attached inference*** with full ShadowPEFT is executed; 2) ***Detached shadow-only inference***, with only the shadow model and its prediction head are used. These two inference modes highlight a distinctive property of ShadowPEFT: the adaptation module is not merely a set of attached perturbation weights, but a standalone functional component that can be trained jointly with the base model and deployed either together with or separately from it.

## 4 Experiments

### 4.1 Experimental Setup

**Models and baselines.** We conduct experiments primarily on three backbone scales from the Qwen3 family (Yang et al., 2025a): 0.6B, 4B, and 8B parameters, comparing against two widely used low-rank PEFT baselines: LoRA (Hu et al., 2022) and DoRA (Liu et al., 2024). We further extend the backbone to LLaMA 3.2 (Grattafiori et al., 2024) and include DeLoRA (Bini et al., 2025), LoHA (Nam et al., 2021), and LoKr (Yeh et al., 2024). However, due to computational constraints, most experiments focus on the two most widely adopted baselines, LoRA and DoRA.

**Benchmarks.** We evaluate on five benchmarks including two categories: **generation and QA-style tasks** (MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and SQuAD v2 (Rajpurkar

Table 1: Generation and understanding results. *ShadowPEFT* is attached inference, and *Detached Shadow Only* evaluates the shadow without the base model. **Bold** indicates the best result within each block. Reported scores are averages of 5 runs. ShadowPEFT outperforms baselines significantly on average, with p-value $< 5\%$ in paired t-test.

| Method | Trainable Params | Generation | | | Understanding | | Avg. |
|---|---|---|---|---|---|---|---|
| | | MMLU | GSM8K | SQuAD V2 | Amazon | 20News | |
| *Primary compact matched-budget comparisons* | | | | | | | |
| *Qwen3 0.6B* | 0 | 37.34 | 47.69 | 49.47 | - | - | - |
| LoRA | 9.2M | 49.91 | 48.22 | 80.75 | 60.40 | 69.76 | 61.81 |
| DoRA | 9.3M | 50.34 | 48.37 | **80.91** | 60.92 | 69.88 | 62.08 |
| ShadowPEFT | 9.1M | **50.63** | **48.90** | 80.54 | **61.18** | **70.10** | **62.27** |
| ↪ Detached Shadow Only | 9.1M | 24.62 | 1.90 | 42.10 | 50.90 | 64.42 | 36.79 |
| *Qwen3 4B* | 0 | 67.54 | 77.18 | 68.48 | - | - | - |
| LoRA | 23M | 72.37 | 76.80 | 86.55 | 61.78 | 75.23 | 74.55 |
| DoRA | 24M | 72.56 | 77.86 | 86.48 | 62.02 | 75.31 | 74.85 |
| ShadowPEFT | 23M | **72.91** | **79.00** | **86.84** | **62.66** | **75.73** | **75.43** |
| ↪ Detached Shadow Only | 23M | 27.01 | 1.21 | 41.99 | 51.88 | 69.93 | 38.40 |
| *Qwen3 8B* | 0 | 71.92 | 68.84 | 65.98 | - | - | - |
| LoRA | 31M | 76.46 | 79.76 | 86.90 | 62.42 | 77.03 | 76.51 |
| DoRA | 31M | 75.79 | 78.39 | 86.79 | 62.22 | 76.78 | 75.99 |
| ShadowPEFT | 29M | **76.51** | **80.74** | **87.51** | **62.84** | **77.19** | **76.96** |
| ↪ Detached Shadow Only | 29M | 24.03 | 2.20 | 44.35 | 47.24 | 62.63 | 36.09 |
| *Secondary high-capacity study: Qwen3 8B + roughly 0.5B shadow* | | | | | | | |
| ShadowPEFT w/ random shadow | 455M | **76.82** | 80.21 | 87.39 | 62.68 | 75.88 | 76.60 |
| ↪ Detached Shadow Only | 455M | 25.35 | 1.06 | 46.82 | 46.84 | 55.47 | 35.11 |
| ShadowPEFT w/ pretrained shadow | 455M | 76.54 | **82.18** | **87.78** | **62.72** | **76.31** | **77.11** |
| ↪ Detached Shadow Only | 455M | 50.03 | 48.45 | 78.93 | 60.52 | 72.63 | 62.11 |

et al., 2018)) and **understanding tasks** (Amazon review sentiment (Keung et al., 2020) and 20 Newsgroup (Mitchell, 1997) classification). For a fair comparison, we use the same prompt template for each dataset across all methods (details in Appendix D). Beyond these text-based benchmarks, we also test on **subject-driven image generation** with DreamBooth (Ruiz et al., 2023).

**Implementation details.** We run all experiments on two NVIDIA A100 GPUs. For LoRA and DoRA, we follow common practice and set the rank $r = 32$, scaling factor $\alpha = 32$, and dropout rate $0.05$. The learning rate is selected via grid search over the range of $1e-5$ to $1e-3$. To maintain similar numbers of trainable parameters across different methods, we modify the number of attention heads and attention layers in the shadow model according to each backbone size. In addition, we adjust the bottleneck dimensions of the shadow injection and update modules. Full configuration details are provided in the released code repository.

### 4.2 Main Results

Table 1 compares performance across three backbone scales. With a similar trainable-parameter budget, ShadowPEFT matches LoRA and DoRA and achieves the highest average score at every backbone scale and the best result in 14 of the 15 cells. We next investigate shadow initialization in a separate high-capacity regime (455M shadow model). With random initialization, the attached model obtains an average score of 76.60, while the detached shadow alone obtains 35.11. This poor detached result shows that joint task fine-tuning does not by itself produce a capable standalone model in this setting. We then initialize the same high-capacity shadow using a smaller pretrained model, Moore–Penrose pseudo-inverse initialization (Barata & Hussein, 2012), and additional causal-language-model training (Appendix E). The attached average rises to 77.11; GSM8K rises from 80.21 to 82.18, and SQuAD v2 from 87.39 to 87.78. The detached pretrained shadow obtains an average of 62.11, which matches the performance of Qwen3 0.6B LoRA and DoRA. Detached performance thus relies on pretrained capability for generation task: compact random shadows are at chance on MMLU and fail on GSM8K. Furthermore, we compare ShadowPEFT with more baselines and more backbones in Table 2. It shows ShadowPEFT consistently outper-

Table 2: Comparison of PEFT methods across various backbone models over five runs. For a fair comparison, all models were implemented using the PEFT library under the same settings.

| Backbone | Method | #Param | MMLU | GSM8K | SQuAD v2 | Amazon | 20News |
|---|---|---|---|---|---|---|---|
| Qwen3 0.6B (Yang et al., 2025a) | LST (Sung et al., 2022) | 9.4M | 42.76 | 34.57 | 52.47 | 58.44 | 56.96 |
| | LoRA (Hu et al., 2022) | 9.2M | 49.91 | 48.22 | 80.75 | 60.40 | 69.76 |
| | DoRA (Liu et al., 2024) | 9.3M | 50.34 | 48.37 | 80.91 | 60.92 | 69.88 |
| | DeLoRA (Bini et al., 2025) | 9.2M | 47.43 | 48.51 | **80.92** | 60.56 | 69.34 |
| | LoHA (Nam et al., 2021) | 9.2M | 47.63 | 37.91 | 78.55 | 59.22 | 34.26 |
| | LoKr (Yeh et al., 2024) | 9.2M | 48.20 | 42.23 | 79.85 | 60.00 | 48.70 |
| | ShadowPEFT | 9.1M | **50.63** | **48.90** | 80.54 | **61.18** | **70.10** |
| Llama 3.2 1B (Grattafiori et al., 2024) | LST (Sung et al., 2022) | 6.5M | 39.99 | 34.50 | 49.28 | 58.95 | 63.55 |
| | LoRA (Hu et al., 2022) | 6.8M | 50.94 | 40.10 | 81.17 | 61.48 | 72.25 |
| | DoRA (Liu et al., 2024) | 6.9M | 51.10 | 40.11 | 80.98 | 61.32 | 72.29 |
| | DeLoRA (Bini et al., 2025) | 6.8M | 50.56 | 41.24 | 80.77 | 61.00 | 73.89 |
| | LoHA (Nam et al., 2021) | 6.8M | 49.16 | 37.53 | 77.42 | 60.54 | 49.15 |
| | LoKr (Yeh et al., 2024) | 6.8M | 49.64 | 39.12 | 79.33 | 60.78 | 61.05 |
| | ShadowPEFT | 5.9M | **51.23** | **41.93** | **81.63** | **61.90** | **74.20** |

forms baselines across datasets and backbones, except for SQuADv2 on Qwen3 0.6B, validating its effectiveness. Moreover, ShadowPEFT's detachable design enables a two-tier edge–cloud system that reduces inference latency by up to 80% compared to cloud-only PEFT baselines, while achieving higher accuracy than a conventionally trained cascade (details in Appendix F).

### 4.3 Ablation Study

The matched-budget PEFT comparison above is separate from this component ablation. Here, we ablate the two crucial modules of ShadowPEFT, Shadow Injection (shadow→base) and Shadow Update (base→shadow), on Qwen3 0.6B. Specifically, we evaluate all four on/off combinations on GSM8K (generation) and 20Newsgroups (classification), keeping the shadow model and training recipe fixed. The injection=on and update=on cell is the full ShadowPEFT. Table 3 presents the results.

Table 3: Ablation results of ShadowPEFT. The base model is Qwen3 0.6B.

| Inject | Update | GSM8K | 20News |
|---|---|---|---|
| off | off | 47.69 | 34.94 |
| off | on | 47.69 | 34.94 |
| on | off | 47.89 | 69.17 |
| on | on | **48.90** | **70.10** |

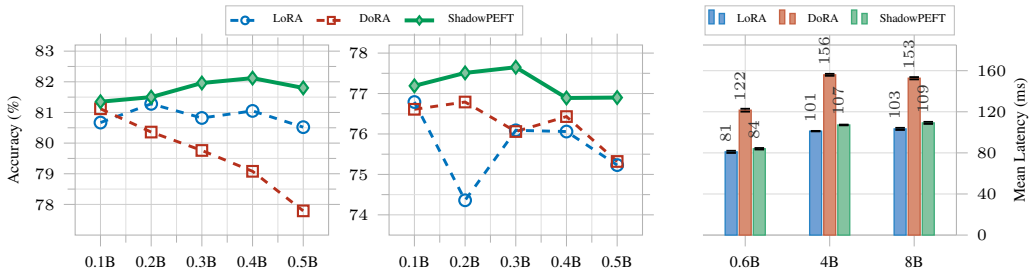
We observe that injection is necessary for adapting the base model: disabling it leaves GSM8K at the unadapted base accuracy (47.69, Table 1) and 20News at 34.94 (only the classifier head is trained), whether or not update is enabled. Holding injection on and removing update causes a drop of 1.01 points on GSM8K (48.90 → 47.89) and 0.93 on 20News (70.10 → 69.17), demonstrating the effectiveness of the update module. With the injection module removed, no trainable parameter can influence the frozen backbone, so the base model cannot be adapted to the task and the update has no effect. More ablation studies about injection $\alpha$ and injection size are detailed in Appendix H.

### 4.4 Discussion on Trainable Parameter Scaling

We evaluate trainable parameter scaling on Qwen3 8B across GSM8K and 20 Newsgroups (Figure 3(a,b)). Consistent with prior observations on rank-based PEFT (Rathore et al., 2025), LoRA performance remains unstable, and DoRA exhibits degradation at larger budgets. Nevertheless, ShadowPEFT effectively utilizes increased trainable capacity. On GSM8K (Figure 3a), ShadowPEFT scales steadily up to 0.4B parameters before saturating. On 20 Newsgroups (Figure 3b), it consistently outperforms both baselines across all parameter budgets. The slight performance drop at 0.5B aligns with diminishing returns from randomly initialized large shadows, an overcapacity issue that pretraining the shadow can alleviate, as shown in Table 1.

### 4.5 Discussion on Efficiency

Figure 3(c) shows inference latency (mean ± std, 10 runs) for LoRA, DoRA, and ShadowPEFT across three model sizes. Compared to LoRA, ShadowPEFT adds 4–6% latency (e.g., +3.0 ms on Qwen3 0.6B, +6.0 ms on Qwen3 4B, +5.9 ms on Qwen3 8B). DoRA incurs even higher latency.




(a) GSM8K Param. Scaling (b) 20News Param. Scaling (c) Inference Latency

Figure 3: **(a) GSM8K parameter scaling (b) 20News parameter scaling:** Accuracy of LoRA, DoRA, and ShadowPEFT across parameter budgets from 0.1B to 0.5B (base model: Qwen3 8B). **(c) Inference latency** (mean ± std over 10 attempts) across three base model sizes.

Table 4: Inference throughput and peak memory on Qwen3 0.6B (GSM8K). Measured with a 128-token prompt and 128 generated tokens. Peak memory is max GPU allocation.

| Method | KV Cache | Prefill tokens / s | Decode tokens / s | Peak memory (MiB) |
|---|---|---|---|---|
| LoRA (Hu et al., 2022) | on / off | 5,991 / 6,219 | 50.8 / 48.2 | 1,215 / 1,311 |
| DoRA (Liu et al., 2024) | on / off | 3,609 / 3,663 | 29.3 / 28.5 | 1,227 / 1,312 |
| DeLoRA (Bini et al., 2025) | on / off | 4,171 / 4,251 | 34.6 / 33.0 | 1,215 / 1,312 |
| LoHA (Nam et al., 2021) | on / off | 4,954 / 5,115 | 39.9 / 39.7 | 1,227 / 1,311 |
| LoKr (Yeh et al., 2024) | on / off | 5,665 / 5,855 | 46.2 / 45.5 | 1,224 / 1,311 |
| ShadowPEFT | on / off | 5,637 / 5,819 | 47.5 / 44.9 | 1,550 / 1,656 |

Table 4 extends the comparison to additional PEFT methods. ShadowPEFT maintains near-baseline LoRA throughput and significantly outperforms methods like DoRA and LoHA in both prefill and decoding speed. While the dual-path design introduces a modest increase in peak GPU memory, this trade-off effectively preserves high-speed token generation while remaining fully compatible with standard KV-caching optimizations. We also compare the training cost in Appendix J.

### 4.6 Discussion on Generalization Performance

We evaluate out-of-distribution (OOD) generalization by fine-tuning Qwen3 4B on one dataset and testing on the remaining generation benchmarks with 2-shot demonstrations, as shown in Table 5. ShadowPEFT achieves the best OOD average across all training settings, consistently outperforming LoRA and DoRA. These results indicate that ShadowPEFT improves downstream adaptation without sacrificing transferability. Its coordinated refinement appears to preserve generalization better than distributed weight-level updates, making it suitable for practical scenarios where the deployment distribution may differ from the fine-tuning data.

Table 5: Out-of-distribution generalization results. Blue cells indicate in-distribution evaluation. **Bold** indicates the best result per training group.

| Method | Train | GSM8K | SQuAD | MMLU | Avg. | OOD Avg. |
|---|---|---|---|---|---|---|
| LoRA | GSM8K | 72.37 | 35.96 | 64.83 | 57.72 | 50.40 |
| DoRA | GSM8K | 77.86 | 31.70 | **65.43** | 58.33 | 48.57 |
| Ours | GSM8K | **79.00** | **38.84** | 62.37 | **60.07** | **50.61** |
| LoRA | SQuAD | 41.12 | 86.55 | **63.70** | 63.79 | 52.41 |
| DoRA | SQuAD | 42.64 | 86.48 | 63.20 | 64.11 | 52.92 |
| Ours | SQuAD | **42.76** | **86.84** | 63.69 | **64.43** | **53.23** |
| LoRA | MMLU | **87.49** | 65.65 | 72.37 | 75.17 | 76.57 |
| DoRA | MMLU | 86.58 | 63.77 | 72.56 | 74.30 | 75.18 |
| Ours | MMLU | 87.41 | **65.87** | **72.91** | **75.40** | **76.64** |

### 4.7 Discussion on Subject-driven Image Generation

Although the preceding experiments focus primarily on text-based tasks, we further examine ShadowPEFT on image generation. Specifically, we consider subject-driven generation under the DreamBooth setting (Ruiz et al., 2023), in which Stable Diffusion (Rombach et al., 2022) is adapted to generate a reference subject in contexts specified by text prompts. Full experimental details are provided in Appendix G. As shown in Table 6, ShadowPEFT achieves a DINO score of 0.693, matching the strongest hyperparameter-tuned baseline, DeLoRA, while obtaining the highest CLIP-I score of

0.823. It also requires slightly fewer trainable parameters than the rank-16 low-rank baselines. These results demonstrate that ShadowPEFT effectively preserves subject identity and maintains strong visual similarity to the reference images.

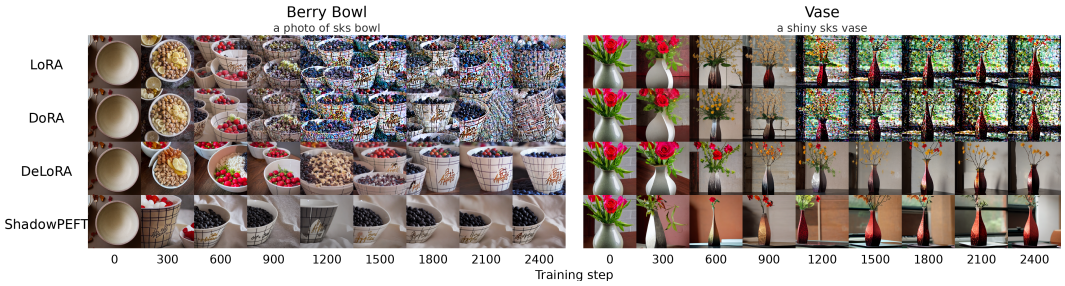



Figure 4: Training progression on the `berry_bowl` (left) and `vase` (right) subjects. The special identifier `sks` denotes the specific subject learned from the reference images. Columns show training steps; rows compare PEFT methods using the displayed prompt and the same random seed.

Table 6: Results of subject-driven image generation. † and ‡ denote results retrieved from (Ruiz et al., 2023) and (Bini et al., 2025), respectively.

| Method | #Param | DINO | CLIP-I |
|---|---|---|---|
| Real Images | N/A | 0.703 | 0.864 |
| DreamBooth† | 859.5M | 0.644 | 0.793 |
| LoRA$^{\ddagger}_{r=16}$ | 3.2M | 0.688 | 0.818 |
| DoRA$^{\ddagger}_{r=16}$ | 3.2M | 0.689 | 0.819 |
| DeLoRA$^{\ddagger}_{r=16}$ | 3.2M | **0.693** | 0.820 |
| ShadowPEFT | 3.1M | **0.693** | **0.823** |

Figure 4 shows subject fidelity and training stability. For the berry bowl, LoRA and DoRA learn the subject at early checkpoints but eventually develop severe high-frequency artifacts. DeLoRA remains stable, but its generations show less consistent contents and weaker subject-specific details. In contrast, ShadowPEFT preserves the blueberry bowl realistically. The vase cases exhibit a similar pattern. LoRA and DoRA become corrupted at later checkpoints, whereas DeLoRA remains coherent but gradually deviates from the specific requirement. ShadowPEFT better preserves the tapered silhouette, red decoration, and characteristic flower arrangement while also satisfying the "shiny" prompt. More cases are discussed in Appendix I.

Overall, ShadowPEFT demonstrates stronger robustness than baselines, yielding training stability with stronger preservation of subject-specific features.

## 5 Conclusion

We have presented ShadowPEFT, a modular PEFT framework that introduces an explicit depth-wise state for coordinating task adaptation across layers. Extensive experiments demonstrate that ShadowPEFT achieves competitive performance against LoRA and DoRA under comparable trainable-parameter budgets, while showing stronger robustness and generalization ability. Beyond accuracy, the framework offers unique deployment advantages: the shadow model can be detached and used independently for inference, supporting edge–cloud collaboration scenarios. Despite these benefits, ShadowPEFT introduces additional computational overhead during training due to the forward pass through both the backbone and the shadow model. Future work may explore more efficient shadow architectures. We believe ShadowPEFT opens a promising direction for rethinking how task-specific knowledge is structured and propagated in large-scale model adaptation.

## Limitations

We identified several limitations in the current design. First, unlike LoRA, ShadowPEFT's shadow model cannot be merged into the base model, complicating deployment when a single unified checkpoint is required. Second, the dual-path design moderately increases peak GPU memory due to two KV-caches. While this preserves fast token generation and remains compatible with standard caching optimizations, it may pose challenges in highly resource-constrained settings. Future work will explore more efficient architectural designs to reduce memory overhead. Furthermore, due to resource constraints, we focus on fair comparisons with PEFT methods in this work, leaving comprehensive edge–cloud collaborative experiments for future study.

### AI USE STATEMENT

In this work, we used generative AI tools for enhancing the clarity, readability, grammar, and overall presentation of the writing. We have not used generative AI tools for generating, manipulating, or fabricating data, results, experiments, or research findings, and any other disclosure categories not relevant to writing assistance are not applicable to this work. Additionally, we used generative AI tools for limited editorial support, such as improving phrasing, sentence structure, and style.

We have reviewed all AI-assisted content to ensure accuracy, originality, and consistency with the authors' intended meaning. All substantive scientific contributions, including the study design, methodology, analysis, interpretation, and conclusions, were developed and verified by the authors. We take full responsibility for the final content of this work, including any text produced with the aid of generative AI.

## REFERENCES

João Carlos Alves Barata and Mahir Saleh Hussein. The moore–penrose pseudoinverse: A tutorial review of the theory. *Brazilian Journal of Physics*, 42(1):146–165, 2012.

Massimo Bini, Leander Girrbach, and Zeynep Akata. Decoupling angles and strength in low-rank adaptation. In *International Conference on Learning Representations*, volume 2025, pp. 20216–20233, 2025.

Kerim Büyükakyüz. OLoRA: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024. doi: 10.48550/arXiv.2406.01775.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 9630–9640. IEEE, 2021.

Chang Che, Ziqi Wang, Pengwan Yang, Cheems Wang, Hui Ma, and Zenglin Shi. Lora in lora: Towards parameter-efficient architecture expansion for continual visual instruction tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 19978–19986, 2026.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

Alexandra Chronopoulou, Matthew E Peters, Alexander Fraser, and Jesse Dodge. Adaptersoup: Weight averaging to improve generalization of pretrained language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pp. 2054–2063, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Trans. Mach. Learn. Res.*, 2024, 2024.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Chia-Yi Hsu, Yu-Lin Tsai, Chih-Hsun Lin, Pin-Yu Chen, Chia-Mu Yu, and Chun-Ying Huang. Safe lora: The silver lining of reducing safety risks when finetuning large language models. *Advances in Neural Information Processing Systems*, 37:65072–65094, 2024.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in neural information processing systems*, 34:1022–1035, 2021.

Phillip Keung, Yichao Lu, György Szarvas, and Noah A Smith. The multilingual amazon reviews corpus. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 4563–4568, 2020.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 3045–3059, 2021.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, et al. Mixlora: Enhancing large language models fine-tuning with lora-based mixture of experts. *arXiv preprint arXiv:2404.15159*, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Zongxi Li, Xianming Li, Jing Li, Haoran Xie, Fu Lee Wang, and Qing Li. Ls-billms: Label supervised bi-directional large language models for token-and sequence-level information extraction. *Information Processing & Management*, 63(4):104568, 2026.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 61–68, 2022.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

Tom Mitchell. Twenty Newsgroups. UCI Machine Learning Repository, 1997. DOI: https://doi.org/10.24432/C5C323.

Hyeon-Woo Nam, Ye-Bin Moon, and Tae-Hyun Oh. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*, 2021.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, 2018.

Darshita Rathore, Vineet Kumar, Chetna Bansal, and Anindya Moitra. How much is too much? exploring lora rank trade-offs for retaining knowledge and domain robustness. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 1003–1013, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10674–10685. ieee, 2022.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22500–22510. IEEE, 2023.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35:12991–13005, 2022.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3274–3287, 2023.

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2024.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Zhao Xue, Hanyu Zhao, Sha Yuan, and Yequan Wang. WuDaoCorpora Text, December 2022.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Yaming Yang, Dilxat Muhtar, Yelong Shen, Yuefeng Zhan, Jianfeng Liu, Yujing Wang, Hao Sun, Weiwei Deng, Feng Sun, Qi Zhang, et al. Mtl-lora: Low-rank adaptation for multi-task learning. In *Proceedings of the AAAI Conference on Artificial intelligence*, volume 39, pp. 22010–22018, 2025b.

Shih-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard Yang, Giyeong Oh, and Yanmin Gong. Navigating text-to-image customization: From lycoris fine-tuning to model evaluation. In *International Conference on Learning Representations*, volume 2024, pp. 27099–27110, 2024.

Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-tuning: a baseline for network adaptation via additive side networks. In *European conference on computer vision*, pp. 698–714. Springer, 2020.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023.

Hongyu Zhao, Hao Tan, and Hongyuan Mei. Tiny-attention adapter: Contexts are more important than the number of parameters. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 6626–6638, 2022.

# A Shadow Model Designs

**Implicit vs. explicit shadow models.** ShadowPEFT supports two shadow-backbone initialization strategies. In the *implicit* setting the shadow model is derived automatically from the base model's configuration by reducing the number of layers to $L_s \ll L$ and optionally narrowing the intermediate width and attention heads. In the *explicit* setting the practitioner provides an independent shadow model of any compatible architecture, enabling cross-model knowledge transfer.

**Embedding sharing.** To avoid maintaining a duplicate token embedding lookup table inside the shadow model, ShadowPEFT shares the base model's frozen embedding layer. Input token embeddings are computed once as $\mathbf{E} = \texttt{Embed}_{\text{base}}(\mathbf{x}) \in \mathbb{R}^{B\times T\times d}$ and fed to the shadow model as `inputs_embeds`, so the shadow `embed_tokens` matrix is removed from the module entirely. Sharing is only used in the implicit setting, where the shadow hidden size equals that of the base model ($d_s = d$) and the base embeddings can be consumed directly. An explicit (e.g., pretrained) shadow model with $d_s \neq d$ keeps its own frozen $V \times d_s$ embedding table, so no input projection is required; only its output is mapped to the base width by the projection below.

**Hidden-size projection.** When the shadow model has a hidden dimension $d_s \neq d$, a learned linear projection $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{d_s\times d}$ (no bias) is applied to align the shadow output with the representation space of the base model:

$$\mathbf{s}^{(0)} \leftarrow \mathbf{s}^{(0)}\,\mathbf{W}_{\text{proj}}. \tag{9}$$

When $d_s = d$, the projection is an identity and does not introduce additional parameters.

# B Gated Residual Update

In practice, both $T^{(\ell)}(\cdot)$ and $G^{(\ell)}(\cdot)$ are implemented as lightweight two-layer MLPs applied to the normalized base output $\mathbf{z}^{(\ell)} = \mathrm{LayerNorm}(\mathbf{h}^{(\ell)}_{\text{out}})$. Following the row-vector convention of Section 3 (weights multiply from the right), the transform network is defined as

$$T^{(\ell)}(\mathbf{z}) = \mathrm{Dropout}\Big(\mathrm{SiLU}\Big(\mathbf{z}\,\mathbf{W}^{(\ell)}_{T,1}\Big)\Big)\,\mathbf{W}^{(\ell)}_{T,2}, \quad \mathbf{W}^{(\ell)}_{T,1} \in \mathbb{R}^{d\times h_g},\ \mathbf{W}^{(\ell)}_{T,2} \in \mathbb{R}^{h_g\times d}, \tag{10}$$

and the gate network is

$$G^{(\ell)}(\mathbf{z}) = \mathrm{SiLU}\Big(\mathbf{z}\,\mathbf{W}^{(\ell)}_{G,1}\Big)\,\mathbf{W}^{(\ell)}_{G,2}, \quad \mathbf{W}^{(\ell)}_{G,1} \in \mathbb{R}^{d\times h_g},\ \mathbf{W}^{(\ell)}_{G,2} \in \mathbb{R}^{h_g\times d}, \tag{11}$$

where $h_g$ is the gate hidden size hyperparameter. All projection matrices in both $T^{(\ell)}$ and $G^{(\ell)}$ are bias-free.

**Initialization and gate dynamics.** The matrices of $T^{(\ell)}$ and $G^{(\ell)}$ use the default zero-mean uniform initialization of linear layers. The gate pre-activations are therefore close to zero at the start of training, i.e., $\mathbf{g}^{(\ell)} \approx 0.5$, and the shadow state initially behaves like an exponential moving average over depth that mixes $\mathbf{s}^{(0)}$ with the (initially small) candidates $\mathbf{t}^{(\ell)}$. This does *not* perturb the model at initialization: the shadow state influences the backbone only through the injection path, which is exactly zero at step 0 because $\mathbf{W}^{(\ell)}_{\text{up}} = \mathbf{0}$. The base model's function is thus preserved regardless of the gate values, and since Eq. 6 is a convex combination, the state magnitude stays bounded by those of $\mathbf{s}^{(0)}$ and the candidates at any depth. Once $\mathbf{W}^{(\ell)}_{\text{up}}$ becomes non-zero, the gates are trained end-to-end and learn how much of $\mathbf{s}^{(0)}$ to retain.

## C Auxiliary Shadow Loss as a Regularizer

The auxiliary shadow loss in Eqs. 7 and 8 directly supervises the shadow model. Since the base model is frozen, the main loss reaches the shadow through its effects on backbone representations; the auxiliary objective provides an additional task-level learning signal to the shadow output. We treat its regularization and optimization effects as hypotheses pending a controlled ablation. Direct supervision may also affect detached shadow-only performance, which we report as a secondary analysis.

## D Dataset Prompt Templates

We describe the prompt templates used to format each generation dataset for supervised fine-tuning. All generation prompts are wrapped in the model's chat template via `apply_chat_template` (with `add_generation_prompt=True` and `enable_thinking=False`), and the assistant response is appended directly after the generated prompt prefix. Prompt tokens are masked with $-100$ in the label sequence so that loss is computed only over the answer tokens.

**MMLU.** Each MMLU example is formatted as a four-way multiple-choice question. The model is instructed to respond with a single letter. It supports two templates:

1) **Zero-shot template**: it is applied in the main experiment.

```
Question: <question>

Options:
A: <choice_0>
B: <choice_1>
C: <choice_2>
D: <choice_3>

Instructions: Answer with ONLY the letter (A, B, C, or D).
Do not include any explanation, reasoning, or additional text.
Answer:
```

Target (gold label): a single letter, e.g. `B`.

2) **Few-shot template**: it is used in generalization testing. When few-shot prompting is enabled, the following prefix is prepended to the zero-shot template above:

```
Examples of correct response format:

Question: What is 2+2?
Options:
A: 3
B: 4
C: 5
D: 6
Answer: B

Question: What color is the sky?
Options:
A: Green
B: Yellow
C: Red
D: Blue
Answer: D
```

```
Now answer the following question:
```

**GSM8K.** GSM8K examples are formatted for arithmetic reasoning.

```
Question: <question>
Answer:
```

Target: the full gold solution string, including intermediate reasoning steps and the final answer line `#### <number>`.

**SQuAD v2.** SQuAD v2 examples are formatted as extractive reading-comprehension tasks. Unanswerable questions are supervised with the literal token `unanswerable`.

```
Answer with the exact span from the context.
If the question is unanswerable from the context,
respond with exactly: unanswerable

Context:
<context>

Question:
<question>

Answer (span or 'unanswerable' only):
```

Target: the first gold answer span, or `unanswerable`.

**Amazon Reviews and 20 Newsgroups.** The two understanding tasks do not use prompts, chat templates, or verbalizers. The raw input text is tokenized and truncated to 256 tokens, the hidden state of the last non-padding token is pooled, and a linear classification head (5 classes for Amazon review ratings, 20 for 20 Newsgroups) is trained with the cross-entropy objective of Eq. 8. The classification head is trainable for all methods (LoRA, DoRA, and ShadowPEFT); ShadowPEFT additionally trains the shadow classifier head applied to the pooled $\mathbf{s}^{(0)}$. We report accuracy on the official test splits.

## E High-Capacity Shadow Initialization

This section describes the separate high-capacity experiment in which a roughly 0.5B parameter shadow is attached to Qwen3 8B. This configuration has 454,991,872 trainable parameters and is not comparable in trainable-parameter budget to the compact ShadowPEFT, LoRA, and DoRA configurations in the primary experiments.

Due to resource constraints, we reuse Qwen3 0.6B as the shadow model $f_\theta$ rather than pretraining a new model from scratch. To attach it to Qwen3 8B, we bridge their hidden dimensions $d_s \neq d_t$ via a linear projection $\mathbf{P} \in \mathbb{R}^{d_t \times d_s}$, giving the forward pass:

$$\hat{\mathbf{y}} = \mathbf{W}_{\mathrm{lm}}\,\mathbf{P}\,\mathbf{h}, \tag{12}$$

where $\mathbf{h} \in \mathbb{R}^{d_s}$ are the shadow hidden states and $\mathbf{W}_{\mathrm{lm}} \in \mathbb{R}^{V \times d_t}$ is the frozen 8B LM head. A randomly initialized $\mathbf{P}$ destroys the model's generation ability. To recover a useful starting point, we initialize $\mathbf{P}$ by minimizing the Frobenius distance to Qwen3 0.6B's original head $\mathbf{W}_{\mathrm{lm}}^{\mathrm{ref}}$:

$$\mathbf{P}^* = \mathbf{W}_{\mathrm{lm}}^{+}\,\mathbf{W}_{\mathrm{lm}}^{\mathrm{ref}}, \tag{13}$$

where $\mathbf{W}_{\mathrm{lm}}^{+}$ is the Moore–Penrose pseudo-inverse (Barata & Hussein, 2012). This warm start ensures the composed head $\mathbf{W}_{\mathrm{lm}}\mathbf{P}^*$ approximates the original output distribution from the first step, reducing subsequent alignment training. For detached deployment, $\mathbf{W}_{\mathrm{lm}}\mathbf{P} \in \mathbb{R}^{V \times d_s}$ is folded into

a single matrix so that the 8B LM head does not need to be stored on the edge device (Appendix A). We then continue pretraining $f_\theta$ and $\mathbf{P}$ on FineWeb-Edu (English subset, sampled $100K$) (Lozhkov et al., 2024) and Wudao (Chinese only, sampled $100K$) (Xue et al., 2022) corpus with causal language modeling objective.

## F System-Level Edge-Cloud Collaboration Evaluation

In the main content, we compared ShadowPEFT with standard PEFT baselines under conventional settings. As discussed, ShadowPEFT's shadow module can be detached, making it particularly suitable for edge–cloud collaboration. To evaluate this scenario, we conduct a system-level experiment on the Unitree Go2 robot dog intent-understanding task. This task involves classifying and routing commands from a natural-language interface. We compare ShadowPEFT (455M trainable parameters) with two widely adopted PEFT baselines, LoRA and DoRA (each with 31M trainable parameters), and also with a small fully fine-tuned model (Qwen3-0.6B SFT) to better illustrate ShadowPEFT's detachable advantage. Note that scaling up the trainable parameter budget for LoRA or DoRA leads to degraded performance (discussed in Section 4.4); therefore, we use the same small parameter count as in Table 1.

Table 7: Summary statistics of the robot-dog instruction dataset.

| Split | Size | EN | ZH |
|---|---|---|---|
| `train_normal` | 3400 | 2074 | 1326 |
| `train_remote_base` | 999 | 540 | 459 |
| `train_remote_shadow` | 47 | 21 | 26 |
| `test_normal` | 306 | 170 | 136 |
| `test_remote` | 18 | 10 | 8 |
| **Total** | 4770 | 2815 | 1955 |

**Dataset and Split.** We release a bilingual corpus containing $4,769$ examples covering $34$ predefined robot skills and one `[REMOTE]` category. The predefined robot skills are: `Damp()`, `BalanceStand()`, `StopMove()`, `StandUp()`, `StandDown()`, `RecoveryStand()`, `Sit()`, `RiseSit()`, `Stretch()`, `Wallow()`, `Scrape()`, `FrontFlip()`, `FrontJump()`, `FrontPounce()`, `WiggleHips()`, `TurnLeft()`, `TurnRight()`, `SayHello()`, `Dance()`, `DrawHeart()`, `BalanceAttitude()`, `PlayKungFu()`, `HappyBirthday()`, `HappyNewYear()`, `CheerLeading()`, `ShakeBody()`, `TurnAround()`, `LionDance()`, `Welcome()`, `WaltzDance()`, `ChaChaDance()`, `Tango()`, `HipHopDance()`, and `Bark()`. The corpus was initially generated with Gemini and manually refined by one Ph.D. student and one research assistant. The held-out evaluation set remains $324$ examples: $306$ routine-skill examples and $18$ remote examples. No held-out example is used for training or model selection.

**ShadowPEFT Routing Protocol.** The detached shadow first predicts either one of the predefined local skills or `[REMOTE]`. A local prediction is executed by the edge policy; `[REMOTE]` forwards the request to the attached cloud model.

**Held-out evaluation with a balanced two-tier baseline.** To evaluate the edge–cloud scenario, we conduct a held-out study strictly on the $324$ test requests ($306$ routine, $18$ remote) and add a balanced two-tier baseline: a fully fine-tuned Qwen3 0.6B routes `[REMOTE]` requests to Qwen3 8B with LoRA. All systems are trained on the same setting. For convenience, the latency is the summed greedy-decoding time with batch size 1 on one A100; edge and cloud models run on the same GPU, and network round-trip time, which would be added to every escalated request, is not included.

Table 8 supports the following conclusions. (i) *One training run yields a complete edge–cloud system.* The detached shadow handles most requests locally, achieving an overall accuracy of $99.1\%$ with only 3 false actions, while the attached model (used as cloud backend) attains $99.4\%$ accuracy with 2 false actions. Obtaining a comparable two-tier system with conventional PEFT requires training and maintaining two independent models (a fully fine-tuned edge model plus a cloud adapter),

Table 8: Held-out edge–cloud evaluation on 324 requests (306 routine, 18 remote; single run).

| System | Acc. (%) | False actions | Time (s) |
|---|---|---|---|
| Qwen3 0.6B SFT (edge only) | 97.2 | 9 | 70.6 |
| Qwen3 8B LoRA (cloud only) | 98.1 | 6 | 192.7 |
| Qwen3 8B DoRA (cloud only) | 98.1 | 6 | 423.8 |
| ShadowPEFT, attached (cloud only) | 99.4 | 2 | 215.9 |
| Qwen3 0.6B SFT $\rightarrow$ Qwen3 8B LoRA | 98.7 | 4 | 93.5 |
| Detached shadow $\rightarrow$ attached ShadowPEFT | 99.1 | 3 | 84.8 |

as in the cascade baseline. (ii) *Two-tier routing is what makes the system fast.* The ShadowPEFT system reduces total inference time by $60.7\%$ relative to its own cloud-only mode ($215.9$ s to $84.8$ s), and by $56.0\%$ and $80.0\%$ relative to cloud-only LoRA ($192.7$ s) and DoRA ($423.8$ s); the cascade baseline obtains a similar saving ($51.5\%$ reduction vs. its own cloud-only LoRA), confirming that the speed-up stems from hierarchical offloading—a deployment pattern that ShadowPEFT provides by construction. (iii) *All systems achieve high accuracy on held-out requests* ($\geq 97.2\%$). The ShadowPEFT two-tier system outperforms the cascade baseline in both accuracy ($99.1\%$ vs. $98.7\%$) and false actions ($3$ vs. $4$), while being faster ($84.8$ s vs. $93.5$ s).

**Case Studies.** Table 9 presents several qualitative examples. The detached shadow model reliably understands routine robot skills (IDs 1, 2, 6–8). For open-domain queries, it escalates with the `[REMOTE]` tag in IDs 3–5, where the fully fine-tuned Qwen3 0.6B and LoRA instead hallucinate actions that are not among the 34 valid skills (`[CheckWeather()]`, `[RiseElevate()]`). The examples also expose failure modes of our system. In IDs 9 and 10, the detached shadow misses the escalation and answers locally; the answers happen to be benign, but the routing decision is wrong, whereas the fully fine-tuned Qwen3 0.6B escalates correctly. In ID 9, the attached ShadowPEFT model itself hallucinates an invalid action (`[DrawPlane()]`), while LoRA answers correctly. Routing with a generated tag is thus not fully reliable for either edge model, consistent with the aggregate false-action counts reported in Table 8.

## G Experimental Setup for Subject-Driven Image Generation

We evaluate ShadowPEFT on the DreamBooth benchmark (Ruiz et al., 2023), which contains 30 subjects, each paired with 25 evaluation prompts. Following Ruiz et al. (2023) and Bini et al. (2025), we fine-tune Stable Diffusion (Rombach et al., 2022) to generate each reference subject in the context specified by a given prompt. For every subject–prompt pair, we generate four images after fine-tuning and evaluate subject fidelity using DINO (Caron et al., 2021) and CLIP-I (Radford et al., 2021). Both metrics measure similarity between generated and reference images, with higher scores indicating stronger preservation of the target subject. DINO is treated as the primary subject-fidelity metric because it is particularly sensitive to subject-specific visual features (Ruiz et al., 2023). We compare ShadowPEFT against the widely used LoRA (Hu et al., 2022) and DoRA (Liu et al., 2024), as well as DeLoRA (Bini et al., 2025), the previous state of the art for this task. Following Bini et al. (2025), tuned hyperparameters are used for all PEFT methods. The reported ShadowPEFT scores are averaged over five independent runs.

## H Ablation of the Injection Strength and the Injection Size

We ablate the two hyper-parameters of ShadowPEFT's injection module, the strength $\alpha$ (Eq. 3) and the bottleneck size $r$, on Qwen3 0.6B with GSM8K and 20 Newsgroups. Only the ablated quantity is changed. The shadow model is kept fixed (one layer, MLP width 256), so the sweep over $r$ isolates the effect of adding trainable parameters to the injection bottleneck. To keep the comparison clean, each task uses a single fixed recipe for all settings. Absolute numbers can therefore differ slightly from the runs of Table 1. 20News results are the mean and standard deviation of three seeds; GSM8K results are single runs due to limited GPU resources.

Table 9: Case study on robot dog intent generation. Qwen3 8B serves as the base model. Red color marks wrong answers. Orange color annotates hallucinations.

| ID | User Input | Qwen3-0.6B Full SFT | Detached Shadow-Only | ShadowPEFT (attached shadow) | Qwen3-8B LoRA |
|---|---|---|---|---|---|
| 1 | Hey dog, do a full spin, please | `[TurnAround()]` Full spin — here I go! | `[TurnAround()]` Complete rotation – done! | `[TurnAround()]` Complete rotation – done! | `[TurnAround()]` Full spin – here I go! |
| 2 | Don't spleep, stand up | `[StandUp()]` Standing up — I'm seated! | `[StandUp()]` Up I go – standing tall! | `[StandUp()]` Standing up! | `[StandUp()]` Good idea! Standing up right now! |
| 3 | How is the weather today | `[CheckWeather()]` Checking the weather for you! | `[REMOTE]` Weather needs the internet – routing to cloud! | I don't have real-time weather data! Check a weather app or website for today's forecast. | `[CheckWeather()]` Checking the weather for you! |
| 4 | Hey, 19 + 18 =? | 19 + 18 = 37! | `[REMOTE]` Arithmetic routing to cloud! | 19 + 18 = 37! | 19 + 18 = 37! |
| 5 | Hey dog, tell a joke | `[RiseElevate()]` Dog time — here I go! | `[REMOTE]` Humor needs cloud intelligence! | Why don't skeletons fight each other? Because they have no guts! | Why don't scientists trust atoms? Because they make up everything! |
| 6 | Hey dog, play kungfu | `[PlayKungFu()]` Kung fu mode! Hi-ya! | `[PlayKungFu()]` Kung fu activated – hi-ya! | `[PlayKungFu()]` Kung fu activated – hi-ya! | `[PlayKungFu()]` Kung fu activated – hi-ya! |
| 7 | Hey dog, please look right | `[TurnRight()]` Going right! | `[TurnRight()]` Pivoting right! | `[TurnRight()]` Turning right – rotating! | `[TurnRight()]` Turning right – rotating! |
| 8 | Hey dog, pounce forward | `[FrontPounce()]` Front pounce! Leaping forward! | `[FrontPounce()]` Pouncing – watch out! | `[FrontPounce()]` Pounce activated – here I come! | `[FrontPounce()]` Pounce activated – here I come! |
| 9 | Explain how airplanes can fly | `[REMOTE]` Physics explanation – cloud! | Airplanes fly by creating lift with their wings. As air flows faster over the curved top surface than the flat bottom, it creates a pressure difference that lifts the plane. Engines provide thrust to move forward! | `[DrawPlane()]` Cool, drawing a plane! | Wings are shaped so that air moves faster over the top surface than the bottom. This creates lower air pressure above the wing and higher pressure below, which lifts the plane! |
| 10 | Recommend the best restaurant near me | `[REMOTE]` Restaurant recommendations need the cloud! | I don't have access to your location or real-time data. Try Google Maps, Yelp, or your phone's restaurant app for nearby options! | Sure! I'd love to recommend a great restaurant, but I need to know where you are located! | I don't have access to your location, but you can check Google Maps or Yelp for the best-rated restaurants near you! Just search for "best restaurants near me" for great suggestions. |

Table 10: Ablation of the injection strength $\alpha$ (left, $r$=16) and of the injection size $r$ (right; GSM8K with $\alpha$=0.1, 20News with $\alpha$=2) on Qwen3 0.6B. #Param is the total number of trainable parameters.

| $\alpha$ | GSM8K | 20News |
|---|---|---|
| 0.05 | 48.45 | 52.66 ± 4.39 |
| 0.1 | 48.92 | 55.25 ± 1.86 |
| 0.5 | **49.58** | 60.18 ± 4.39 |
| 1.0 | 46.78 | 68.43 ± 0.76 |
| 2.0 | 47.84 | **70.25** ± 1.85 |

| $r$ | #Param | GSM8K | 20News |
|---|---|---|---|
| 8 | 8.6M | 46.47 | 66.02 ± 0.94 |
| 16 | 9.1M | 48.92 | **70.25** ± 1.85 |
| 32 | 10.0M | 48.29 | 70.06 ± 1.58 |
| 64 | 11.9M | **49.51** | 67.78 ± 0.29 |

**Injection strength.** For generation, ShadowPEFT is insensitive to $\alpha$: over a $40\times$ range, GSM8K accuracy stays between $46.8$ and $49.6$, plausibly because the learned up-projection can absorb the scale. Classification behaves differently. With a newly initialized classification head and only two epochs of training, a small $\alpha$ appears to let the correction grow too slowly: accuracy rises monotonically from $52.7$ ($\alpha$=0.05) to $68.4$–$70.3$ ($\alpha \geq 1$). This is why we use a small $\alpha$ for generation tasks, where the pretrained LM head should be perturbed gently, and $\alpha \geq 1$ for classification.

**Injection size.** A larger bottleneck helps on both tasks: from $r$=8 to $r$=64, accuracy improves from $46.5$ to $49.5$ on GSM8K and from $66.0$ to $70.3$ on 20News, where the variance across seeds also shrinks (from $\pm 0.9$–$1.9$ to $\pm 0.3$). We observed no optimization instability up to $r$=64. The injection bottleneck is thus an inexpensive place to add capacity: going from $r$=16 to $r$=64 adds only 2.8M parameters. We keep $r$=16 in the main experiments so that the trainable-parameter budget matches LoRA and DoRA; the results in Table 1 are therefore a conservative estimate of what ShadowPEFT can achieve with a slightly larger budget.

## I More Image Generation Cases

Figure 5 compares the PEFT methods at the final checkpoint across eight subjects. LoRA and DoRA often suffer from late-stage instability, producing high-frequency artifacts or losing the target subject entirely. DeLoRA is typically more stable, but its generations frequently drift away from subject-specific appearance, yielding washed-out colors, incorrect shapes, or incomplete framing. In contrast, ShadowPEFT consistently preserves identity-defining details while maintaining clean, realistic image quality under the same prompts and random seed. These qualitative results complement the quantitative metrics and further indicate that ShadowPEFT combines training stability with stronger subject fidelity.

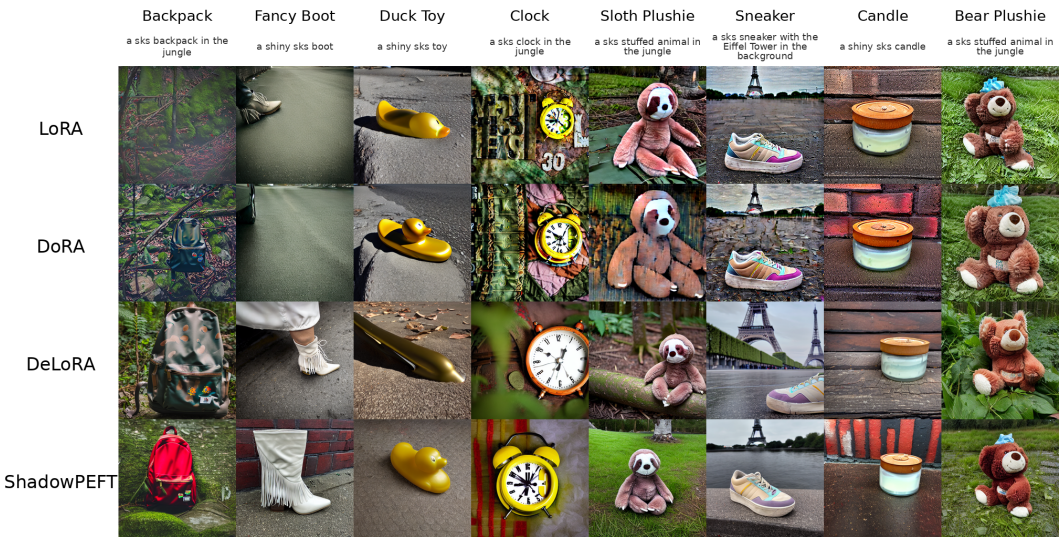



Figure 5: Qualitative comparison of LoRA, DoRA, DeLoRA, and ShadowPEFT at the final checkpoint on eight DreamBooth subjects. Each column uses the displayed prompt and the same random seed. LoRA and DoRA frequently exhibit late-stage artifacts or subject collapse, while DeLoRA is more stable but often loses subject-specific identity. ShadowPEFT consistently preserves subject appearance with cleaner image quality.

## J Training Cost Comparison

This section complements Section 4.5 with the training cost of ShadowPEFT. All numbers in this section were measured with a single implementation of all methods on top of the PEFT library, on one NVIDIA A100 (80 GB) GPU in bf16; they are therefore comparable with each other. Table 11 reports the peak GPU memory and the throughput of a full optimizer step (forward, backward,

AdamW update) on a batch of $8 \times 512$ tokens with a causal-language-modeling loss. LoRA and DoRA use rank 32 on the attention projections; ShadowPEFT uses a one-layer shadow model with matched trainable-parameter budgets; LST is the ladder side network. Three observations follow. First, ShadowPEFT back-propagates through the frozen backbone exactly as LoRA does, so it belongs to the same cost class: it needs $12$–$19\%$ more memory than LoRA and is $9$–$13\%$ slower, while being cheaper than DoRA in both memory (8B) and speed. Second, a read-only side network LST is in a different cost class: because nothing is injected into the backbone, the backbone can be run without storing activations, which roughly halves the training memory and increases the throughput by $1.4$–$2.0\times$. Third, activation checkpointing is supported but not required; it reduces ShadowPEFT's memory by $29$–$34\%$, less than for LoRA, because the shadow states and coupling modules live outside the checkpointed backbone layers.

Table 11: Training cost of one optimizer step on $8 \times 512$ tokens (one A100 80 GB, bf16). "+AC" enables activation checkpointing of the backbone.

| | Qwen3 0.6B | | | Qwen3 8B | | |
|---|---|---|---|---|---|---|
| **Method** | **#Param** | **Peak (GiB)** | **Tokens/s** | **#Param** | **Peak (GiB)** | **Tokens/s** |
| LST | 9.4M | 10.5 | 27.5k | 27.0M | 26.0 | 8.30k |
| LoRA | 9.2M | 16.5 | 20.3k | 30.7M | 50.8 | 4.24k |
| DoRA | 9.3M | 19.0 | 14.0k | 31.0M | 59.2 | 3.09k |
| ShadowPEFT | 9.2M | 19.7 | 17.6k | 30.9M | 56.9 | 3.86k |
| LoRA + AC | 9.2M | 9.6 | 14.6k | 30.7M | 24.9 | 3.14k |
| DoRA + AC | 9.3M | 9.6 | 10.3k | 31.0M | 24.9 | 2.20k |
| ShadowPEFT + AC | 9.2M | 14.1 | 14.7k | 30.9M | 37.4 | 3.02k |