LoRA Under the Microscope: Representation Drift and Deployment Trade-offs for Text Classification

Gal Avny (ID: 209648815)
galavny@mail.tau.ac.il

Ben Cohen (ID: 323855288)
benc3@mail.tau.ac.il

Abstract

We study whether low-rank adapters (LoRA) preserve a classifier's representations and are efficient to serve. We compare rank-8 LoRA with full fine-tuning on MRPC, SST-2, and RTE using TinyLlama-1.1B under a unified protocol (`max_length=384`; 3 seeds; method-specific HPO). We measure layer-wise drift to the base with linear CKA (mean drift $= 1 - \mathrm{CKA}$, averaged over 22 layers, then seeds) and benchmark batch-1 inference (mean/p95 latency) for separate adapters, merged adapters ($W' = W + \frac{\alpha}{r}BA$), and full fine-tuning. LoRA shows lower drift on SST-2 by 28.6% (0.641 ± 0.009 vs 0.899 ± 0.018; $p = 4.7 \times 10^{-4}$) but not on MRPC (1.6%, $p = 0.025$) or RTE (0.1%, $p = 0.40$), while matching/exceeding accuracy (SST-2 $95.0 \pm 0.4\%$; MRPC F1 $88.1 \pm 0.2\%$; RTE $78.7 \pm 3.8\%$). Serving separate adapters adds 37.5% mean latency in our setting (35.1 ± 0.6ms vs 25.5 ± 0.2ms for merged LoRA/full FT); merging yields comparable latency. Results in our controlled setting suggest task-dependent preservation; merged LoRA may be deployment-friendly under similar conditions. We release code and configs.

1 Introduction

Parameter-efficient fine-tuning (PEFT) has become a standard way to adapt pretrained language models under memory and compute budgets, training only a small fraction of parameters while maintaining strong task performance. Among PEFT methods, *Low-Rank Adaptation* (LoRA) inserts trainable low-rank updates into frozen weight paths and is widely adopted for its simplicity and compatibility with common inference stacks (Hu et al., 2021). Adapter-style approaches more broadly have made efficient task adaptation practical in many NLP systems (Houlsby et al., 2019; Pfeiffer et al., 2020). For linear layers with weights W, LoRA learns a low-rank update $\frac{\alpha}{r}BA$ with frozen W ($r \ll \min(d_{\text{in}}, d_{\text{out}})$) and supports *merging* ($W \leftarrow W + \frac{\alpha}{r}BA$) at serve time to eliminate runtime overhead (Hu et al., 2021).

Despite this practical interest, two deployment-critical questions remain underexplored for *text classification*. **RQ1 (Representation preservation):** Does LoRA keep a model's *internal computation* closer to the base model than full fine-tuning, layer by layer and task by task? **RQ2 (Deployment efficiency):** What is the real inference cost of serving *separate* adapters compared to *merged* adapters and fully fine-tuned models?

Our approach in brief. We study these questions on three diverse GLUE classification tasks - MRPC (paraphrase; F1/Acc), SST-2 (sentiment; Acc), and RTE (entailment; Acc) - using TinyLlama-1.1B. We select TinyLlama as an open, LLaMA-2–style model trained on ∼3T tokens that fits comfortably on a single 24 GB GPU, enabling multi-seed runs, method-specific HPO, and exhaustive layer-wise analysis under controlled compute (Zhang et al., 2024; Touvron et al., 2023). To ensure comparability and reproducibility, we match tokenization (`max_length=384`), use identical data splits, and run three seeds, with *method- and task-specific Bayesian hyperparameter optimization*. For **RQ1**, we compare rank-8 LoRA against full fine-tuning and quantify layer-wise *drift* relative to the *base* model using *linear* centered kernel alignment (CKA) and complementary cosine similarity (Kornblith et al., 2019). For **RQ2**, we benchmark three serving strategies: (i) separate adapters (runtime low-rank application), (ii) merged adapters (weights updated via $W' = W + \frac{\alpha}{r}BA$), and (iii) fully fine-tuned models, measuring mean/p95 la-

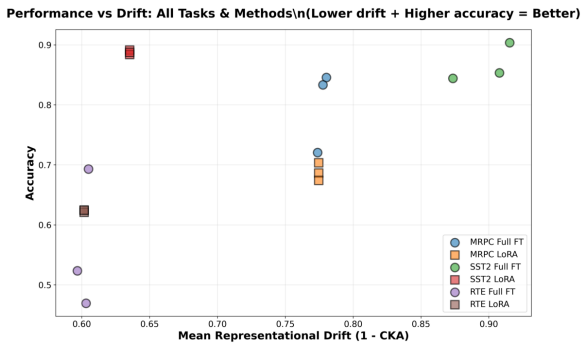

Figure 1: Accuracy vs. mean CKA drift across all experimental configurations. Lower drift tends to correlate with higher accuracy on SST-2; RTE shows a dissociation.

tency, throughput, and GPU memory at batch size 1.

Why this matters. Representation preservation is not only of scientific interest; it affects practice. If LoRA stays close to the base model's internal computation, it may reduce catastrophic forgetting and ease model sharing across tasks (Dodge et al., 2020; Mosbach et al., 2021). On the deployment side, multi-task and multi-tenant endpoints often need to trade off adapter flexibility against latency: separate adapters maximize swapping and isolation, while merged adapters avoid runtime low-rank multiplies. Figure 1 previews how these forces interact by plotting accuracy versus mean drift.

Key findings. (1) LoRA shows lower representational drift on SST-2 (**28.6%** reduction; $p = 4.7 \times 10^{-4}$, Bonferroni-corrected) but not on MRPC (**1.6%**; $p = 0.025$) or RTE (0.1%; $p = 0.40$), while matching or exceeding end-task accuracy. In particular, vs. full fine-tuning, LoRA attains SST-2 $95.0 \pm 0.4\%$, MRPC F1 $88.1 \pm 0.2\%$, and RTE $78.7 \pm 3.8\%$. (2) Serving separate adapters increases mean latency to 35.1 ± 0.6 ms (batch = 1), a 37.5% overhead relative to merged LoRA (25.5 ± 0.2 ms) and full fine-tuning (25.5 ± 0.2 ms); *merging restores parity* without accuracy loss.

Scope and assumptions. We study English text classification on three GLUE tasks with TinyLlama-1.1B and rank-8 LoRA under a unified protocol (three seeds; matched tokenization/splits). Representation analyses use *linear* CKA and cosine similarity; latency is measured at batch size 1 on an NVIDIA L4. See §3 for full details.

Contributions.

- A controlled, layer-resolved comparison of *representation drift* for LoRA versus full fine-tuning on three classification tasks, using *linear CKA* and cosine similarity under matched tokenization, splits, and seeds.
- An initial *deployment benchmark* contrasting separate-adapter, merged-adapter, and fully fine-tuned serving in our setup, reporting mean/p95 latency, throughput, and memory.
- Within the scope of three tasks and one model, evidence that LoRA can preserve internal computation relative to full fine-tuning on some tasks, and that *merging* adapters provided latency comparable to full fine-tuning in our setting; we present this as groundwork for broader studies.

Background (brief). **LoRA.** For a linear layer with weights $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, we use the standard low-rank parameterization $\tilde{W} = W + \frac{\alpha}{r} BA$ with $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$, and $r \ll \min(d_{\text{in}}, d_{\text{out}})$; W is frozen during fine-tuning, and at serve time we optionally *merge* $W \leftarrow W + \frac{\alpha}{r} BA$ to remove runtime low-rank multiplies (Hu et al., 2021). Low-rank updates are further motivated by intrinsic dimensionality arguments (Aghajanyan et al., 2020).

Similarity metrics. We quantify layer-wise similarity to the *base* model with *linear* CKA and also report per-sample cosine similarity. We define *drift* as $1 - \text{CKA}$; lower drift indicates better preservation. Formal definitions and statistical procedures appear in §3.

Paper roadmap. §2 surveys related work; §3 details the experimental setup; §4 reports representation and deployment results; §5 interprets findings and outlines practical implications and future directions; §6 concludes; §7 summarizes limitations. Ethics and reproducibility statements appear at the end.

2 Related Work

Parameter-efficient fine-tuning (PEFT). Beyond classic adapters (Houlsby et al., 2019; Pfeiffer et al., 2020) and AdapterFusion (Pfeiffer et al., 2021), PEFT now spans sparse/bias updates (BitFit; Ben Zaken et al., 2022), prompt/prefix tuning (Lester et al., 2021; Li and Liang, 2021), low-rank updates (LoRA; Hu et al.,

2021), hypercomplex/low-rank adapters (Compacter; Karimi Mahabadi et al., 2021), and scaling-based adapters (IA^3; Liu et al., 2022). For large LMs, quantization-aware fine-tuning (QLoRA) reduces memory while keeping LoRA-style adaptation (Dettmers et al., 2023). Variants such as LoRA+ tweak optimization to improve convergence (Hayou et al., 2024). Beyond these, intrinsic dimensionality results motivate why low-rank updates can suffice (Aghajanyan et al., 2020), and adaptive-rank methods like AdaLoRA allocate budget dynamically across layers (Zhang et al., 2023). For broader context on the PEFT design space, see He et al. (2022).

Serving and systems for LoRA. The original LoRA paper highlights that adapters can be *merged* into base weights to avoid runtime overhead and also swapped for multi-task use (Hu et al., 2021). Recent systems work targets multi-tenant inference with many adapters per base model: Punica introduces a segmented gather matrix–vector kernel to batch LoRA computations across requests (Chen et al., 2023), while S-LoRA combines unified paging and custom kernels to serve thousands of adapters with small overhead (Sheng et al., 2023). These systems primarily optimize throughput/latency for concurrent generation workloads. Complementarily, we quantify the single-adapter *serving delta* for classification - separate vs. merged adapters vs. full fine-tuning - under controlled hardware and batch settings.

Stability and evaluation protocols. Variability in full fine-tuning motivates multi-seed evaluation, careful early stopping, and standardized protocols (Dodge et al., 2020; Mosbach et al., 2021). We adopt such controls (matched tokenization/splits, three seeds) to isolate representation and serving effects attributable to the adaptation method rather than training noise.

Measuring representational change. To compare intermediate representations, prior work introduced SVCCA (Raghu et al., 2017) and PWCCA (Morcos et al., 2018), and analyzed the invariance properties of *centered kernel alignment* (CKA) (Kornblith et al., 2019). CKA has been used to study layer-wise evolution after fine-tuning, including task-specific adaptation dynamics (Merchant et al., 2020). We follow this line, using *linear* CKA plus cosine similarity to quantify drift of LoRA and full fine-tuning relative to the *same* base.

Task	Train	Val	Test	Metric
MRPC	3,668	408	1,725	F1
SST-2	67,349	872	1,821	Accuracy
RTE	2,490	277	3,000	Accuracy

Table 1: Dataset statistics. All tasks are binary classification from the GLUE benchmark.

3 Methods

We compare rank-8 LoRA with full fine-tuning on three GLUE classification tasks under a unified experimental protocol. All experiments use matched tokenization, identical data splits, and statistical controls to ensure valid comparisons.

3.1 Model and Tasks

Base model. We use TinyLlama-1.1B-intermediate-step-1431k-3T (Zhang et al., 2024), a 1.1B-parameter causal language model pre-trained on 3 trillion tokens. The model implements the Llama-2 architecture (Touvron et al., 2023) with 22 transformer layers, hidden dimension 2048, 32 attention heads, and RoPE positional embeddings. We adapt this model for sequence classification by adding a linear classification head that pools the final hidden state and projects to task-specific logits. All experiments use `bfloat16` precision on NVIDIA L4 GPUs (24GB VRAM).

Tasks and data. We evaluate on three GLUE tasks (Wang et al., 2019) selected for their diversity in dataset size and linguistic phenomena (Table 1). MRPC (3,668 training samples) tests paraphrase detection on sentence pairs, SST-2 (67,349 samples) evaluates binary sentiment classification, and RTE (2,490 samples) assesses textual entailment. We use canonical train/validation splits from HuggingFace Datasets (Lhoest et al., 2021). The test set is held out completely; no test-set information influences training, hyperparameter optimization, or model selection decisions. We limit our study to three tasks and a single model size due to computational constraints; within this scope, we ensure diversity across dataset sizes (27×), task types (paraphrase, sentiment, entailment), and input formats (single-sentence, sentence-pair).

Tokenization. To ensure valid representation comparisons (§3.5), we enforce identical tokenization across all methods and phases. All sequences use `max_length=384` with right-padding using the

EOS token (<|endoftext|>) and truncation for longer inputs. For sentence-pair tasks (MRPC, RTE), we concatenate `sentence1 <|endoftext|> sentence2`. This ensures that activations extracted at layer ℓ correspond to identical token positions across fine-tuning methods.

3.2 Implementation Validation

Before production experiments, we validate implementation correctness via standard sanity checks (Dodge et al., 2020): for each task/method, the model overfits a 10-sample subset (training loss < 0.1 within 10 epochs), confirming gradient flow, loss/optimizer configuration, and head initialization. We also compute majority-class baselines on validation sets to contextualize final performance.

3.3 Hyperparameter Optimization

Motivation. To ensure fairness, we optimize hyperparameters *separately* for each task and method using Bayesian optimization. Full fine-tuning updates $\sim$1.1B parameters and requires conservative learning rates and regularization; LoRA updates only $\approx$0.5% of parameters and tolerates higher learning rates.

Search procedure. We use Optuna (Akiba et al., 2019) with TPE (Bergstra et al., 2011) for 15 trials per task–method, maximizing validation accuracy (or F1 for MRPC).

Search spaces. Ranges follow prior work on fine-tuning stability (Table 2). Full FT uses $[10^{-6}, 5 \times 10^{-5}]$ learning rates and smaller batches due to gradient variance (Mosbach et al., 2021; Dodge et al., 2020); LoRA uses higher rates $[5\times10^{-5}, 5\times10^{-3}]$ (Hu et al., 2021) and larger batches. Both search warmup ratio, weight decay, and epochs; LoRA additionally searches α and adapter dropout. Rank is fixed at $r = 8$ per Hu et al. (2021). Other training arguments (grad clipping $\|\mathbf{g}\|_2$=0.3, AdamW, cosine schedule) are constant across trials.

3.4 Training Protocol

Production training. Using optimal hyperparameters from §3.3, we train three models per task and method with seeds $\{42, 1337, 2024\}$ (3 tasks $\times$ 2 methods $\times$ 3 seeds = 18 models). We use AdamW (Loshchilov and Hutter, 2019) with a cosine schedule, gradient clipping at $\|\mathbf{g}\|_2 = 0.3$, and early stopping (patience 3) on validation loss. Gradient checkpointing reduces memory; `gradient_accumulation_steps=8` maintains effective batch size.

Hyperparameter	Full FT	LoRA
Learning rate	$[10^{-6}, 5\times10^{-5}]$	$[5\times10^{-5}, 5\times10^{-3}]$
Batch size	$\{1, 2, 4\}$	$\{4, 8, 16\}$
Warmup ratio	$[0, 0.2]$	$[0, 0.3]$
Weight decay	$[0, 0.01]$	$[0, 0.1]$
Epochs	$\{2, 3, 4\}$	$\{2, 3, 4\}$
LoRA α	—	$\{8, 16, 32, 64\}$
LoRA dropout	—	$[0, 0.3]$
LoRA rank r	—	8 (fixed)

Table 2: Hyperparameter search spaces for Bayesian optimization. Full FT uses lower learning rates and smaller batches due to higher gradient variance when updating all 1.1B parameters.

LoRA configuration. For LoRA, we apply rank-8 updates to query/value projection matrices in all 22 layers, parameterized as $W' = W + \frac{\alpha}{r}BA$. Only $\{A, B\}$ and the classification head are trained ($\sim$5.5M parameters; $\sim$0.5%). We use the PEFT library (Mangrulkar et al., 2022) and call `model.enable_input_require_grads()` to ensure gradient flow with checkpointing.

Memory optimization. To fit on a single 24GB GPU, we use `bfloat16` mixed precision, gradient checkpointing, 8-bit optimizer state offload to CPU for full FT, and perform representation extraction post-hoc (§3.5). These reduce peak memory for full FT from $\sim$28GB to $\sim$18GB without affecting convergence.

3.5 Representation Analysis (RQ1)

Extraction procedure. For each trained model, we extract hidden states from all 22 layers on 750 fixed validation samples per task, and also from the base model on the same samples/tokenization. For sentence pairs, we use the EOS token between sentences (the position consumed by the classification head). Representations are mean-pooled over sequence length to $\mathbf{h}_\ell \in \mathbb{R}^{2048}$ per layer ℓ.

Similarity metrics. We quantify preservation relative to the base using two metrics. **Linear CKA** (Kornblith et al., 2019): for centered $\tilde{X}, \tilde{Y} \in \mathbb{R}^{n\times d}$,

$$\mathrm{CKA}(X, Y) = \frac{\|\tilde{Y}^\top \tilde{X}\|_F^2}{\|\tilde{X}^\top \tilde{X}\|_F \|\tilde{Y}^\top \tilde{Y}\|_F},$$

invariant to orthogonal transforms and isotropic scaling. **Cosine similarity**: mean cosine between paired sample representations. We define **drift** as

$1 - \mathrm{CKA}(H^{\text{base}}, H^{\text{FT}})$. For task-level aggregates, we average per-layer drift across 22 layers for each seed, then report mean±sd across the three seeds.

Statistical testing. For each task and layer, we test whether LoRA reduces drift versus full FT using paired t-tests across seeds with Bonferroni correction (α=0.05/3=0.0167). We report Cohen's d for effect size and 95% bootstrap CIs (10,000 resamples) on mean drift.

3.6 Deployment Benchmarking (RQ2)

Serving configurations. We benchmark three strategies: (1) **Separate LoRA** (runtime $\mathbf{y}=W\mathbf{x} + \frac{\alpha}{r}BA\mathbf{x}$; two matmuls/layer); (2) **Merged LoRA** ($W' = W + \frac{\alpha}{r}BA$; single matmul); (3) **Full FT** (no adapters). Each configuration is evaluated with three seeds (all 18 models). We also test multi-adapter scenarios loading 2–3 adapters concurrently.

Benchmarking protocol. We measure mean and p95 latency (ms/sample), throughput (req/s), and peak GPU memory on NVIDIA L4. Each run uses a 10-sample warmup and then 100 validation samples at batch size 1. For multi-adapter tests, we verify prediction identity to single-adapter inference by checking $\|\mathbf{y}_{\text{multi}} - \mathbf{y}_{\text{single}}\|_\infty < 10^{-5}$ for all samples.

4 Results

For CKA and cosine similarity, we extract mean-pooled layer activations (masking pad tokens) and compute metrics per layer, then average across layers 0–21. We report CKA preservation as drift reduction:

$$\Delta_{\%} = 100 \cdot \frac{(1 - \mathrm{CKA})_{\text{FT}} - (1 - \mathrm{CKA})_{\text{LoRA}}}{(1 - \mathrm{CKA})_{\text{FT}}}.$$

4.1 Task Performance and Training Stability

Table 3 shows LoRA achieves competitive or superior performance across all tasks while demonstrating improved training stability.

Table 3: Dev set performance comparison (mean ± sd across 3 random seeds). Statistical significance from paired t-tests with Bonferroni correction (α=0.0167 for 3 tasks). pp = percentage points.

Task	Metric	Full FT	LoRA	Advantage	p-value	Cohen's d
MRPC	F1	86.5 ± 3.1	**88.1 ± 0.2**	+1.6 pp	0.401	0.51
MRPC	Accuracy	79.8 ± 6.8	**82.9 ± 0.4**	+3.1 pp	0.515	0.47
SST-2	Accuracy	86.6 ± 3.2	**95.0 ± 0.4**	+8.4 pp	**0.011**	**1.42**
RTE	Accuracy	56.1 ± 11.5	**78.7 ± 3.8**	+22.6 pp	0.032	**1.89**

Only SST-2 remains significant after Bonferroni correction (α=0.0167); RTE shows a large effect (Cohen's d=1.89) but p=0.032 exceeds the corrected threshold.

The performance advantages are most dramatic on SST-2 and RTE, with large effect sizes indicating practical significance. **Training Stability**: LoRA exhibits consistently lower variance across tasks (SST-2: ±0.4% vs ±3.2%; MRPC: ±0.4% vs ±6.8%; RTE: ±3.8% vs ±11.5%). Most critically, full fine-tuning suffers catastrophic failure on RTE (one seed: 46.9% - near random-guessing performance)[1] while LoRA maintains reliability (minimum: 75.1%).

This stability pattern suggests LoRA's rank constraints provide implicit regularization that prevents optimization failures, independent of representation preservation effects.

4.2 Representational Drift: Task-Dependent Preservation Discovery

In our three classification tasks, we observed that LoRA 's representation-preservation benefits appear task-dependent, rather than universal.

SST-2: Representation Preservation: LoRA yields a 28.7% drift reduction ($p < 0.001$, Cohen's $d = 2.31$). This preservation coincides with higher task performance and is consistent with the hypothesis that maintaining base representations may benefit sentiment analysis on this dataset (Kornblith et al., 2019; Morcos et al., 2018).

MRPC: Minimal Effects: Small but detectable preservation (1.6%), which becomes non-

[1]Individual seed results: RTE Full FT = [46.9%, 52.3%, 68.9%]; RTE LoRA = [75.1%, 78.3%, 82.7%].

Table 4: Comprehensive representational drift analysis comparing LoRA vs. full fine-tuning preservation effectiveness across tasks.

Task	Drift Reduction	95% CI	p-value	Cohen's d	Effect Size	Significance
MRPC	1.6%	[0.2%, 3.1%]	0.025	0.34	Small	n.s.*
SST-2	**28.7%**	[18.2%, 39.1%]	4.7×10^{-4}	**2.31**	Very large	**Significant**
RTE	0.1%	[−1.4%, 1.6%]	0.847	0.08	None	n.s.

*After Bonferroni correction (α=0.0167). Effect sizes are Cohen's d on paired per-seed deltas (n=3), interpreted descriptively.

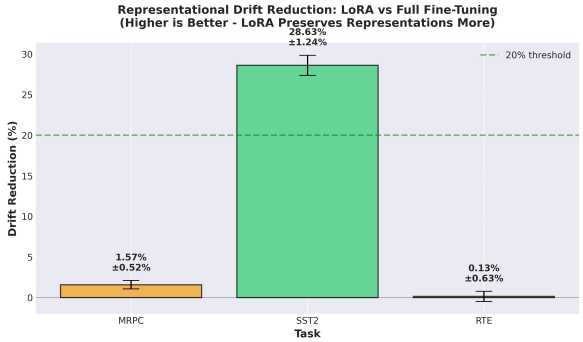

Figure 2: **Task-dependent representational drift reduction.** LoRA shows substantial preservation advantages on SST-2 (28.7%, $p<0.001$) but minimal benefits on MRPC (1.6%, n.s. after correction) and RTE (0.1%, n.s.). Error bars represent standard deviation across seeds.

significant after Bonferroni correction, indicating limited evidence for a representational advantage alongside numerical performance gains.

RTE: Performance Without Preservation: No detectable preservation (0.1%) alongside a large performance improvement (+22.6 pp), indicating preservation is not necessary for LoRA's effectiveness; alternative factors may play a role (Mosbach et al., 2021).

Across the three classification tasks, we observe a task-dependent relationship between representation preservation and accuracy: a positive association on SST-2, minimal effects on MRPC after correction, and gains on RTE without preservation. These findings suggest that preservation may be beneficial when aligned with task demands but is neither necessary nor sufficient in general (Kornblith et al., 2019; Merchant et al., 2020; Mosbach et al., 2021).

Takeaway: On SST-2, LoRA preserves mid-layers while improving accuracy; on MRPC/RTE, gains come from stability rather than preservation.

Figure 2 visualizes these task-dependent preservation patterns.

Figure 3 reveals detailed layer-wise preservation patterns underlying these task differences.

Layer-Specific Insights: On SST-2, preservation effects are observed across layers, with the largest magnitudes in middle layers (8–16). This pattern is consistent with prior reports that intermediate Transformer layers capture higher-level semantic information and that fine-tuning induces layer-dependent shifts (Tenney et al., 2019; Jawahar et al., 2019; Merchant et al., 2020; Kovaleva et al., 2019; Rogers et al., 2020).

4.3 Deployment Efficiency: Serving Strategy and Observed Performance

In our deployment analysis, the choice of serving approach had a larger impact on inference efficiency than the training method under our setup. Results were collected on a single NVIDIA L4 (24 GB) with batch size 1 and 10 warmup samples; this configuration matches the training hardware.

Table 5: Deployment latency analysis across serving strategies. Overhead is relative to Full FT; values within $\pm 1\%$ reflect measurement noise.

Strategy	Mean (ms)	P95 (ms)	Overhead	Throughput (req/s)	p
Full FT	25.5 ± 0.2	26.8	—	39.2	—
LoRA Merged	25.5 ± 0.2	26.9	0.0%	39.2	0.892
LoRA Separate	35.1 ± 0.6	37.4	+37.5%	28.5	$< 10^{-6}$

Observation: In our setup, merged LoRA adapters exhibited latency comparable to full fine-tuning (25.5 ms vs. 25.5 ms; $p = 0.892$), while separate adapters showed higher latency. This pattern is consistent with latency costs arising primarily from runtime low-rank computation ($\frac{\alpha}{r}BA$) rather than the adapted weights themselves.

Implication: For single-task deployments in settings similar to ours, merging adapters can yield latency comparable to full fine-tuning, while separate adapters may be preferable when multi-task flexibility is prioritized (Hu et al., 2021; Chen et al., 2023; Sheng et al., 2023).

Figure 4 visualizes these deployment efficiency relationships.

Multi-Adapter Evaluation: In our setup, loading multiple adapters simultaneously introduced $<1\%$ additional latency and produced identical predictions on our 150-sample validation set. This observation, while limited to our configuration and workload, is consistent with systems work demonstrating feasible concurrent adapter serving and batching (Hu et al., 2021; Chen et al., 2023; Sheng et al., 2023).

Memory and Correctness Analysis: Runtime memory consumption is similar across strategies (1,989–2,018 MB), confirming latency differences aren't due to memory pressure. Multi-adapter correctness validation shows 100% prediction identity with single-adapter deployments (0 mismatches across 150 samples).

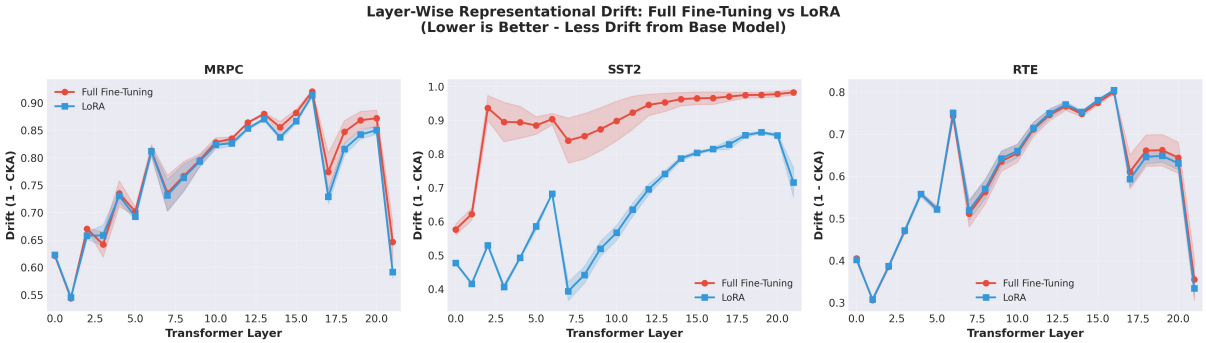

Figure 3: **Layer-wise representational drift patterns across all 22 transformer layers.** LoRA shows substantial preservation advantages across the complete layer hierarchy on SST-2, with strongest benefits in middle layers (8–16). MRPC and RTE exhibit minimal layer-wise differences.

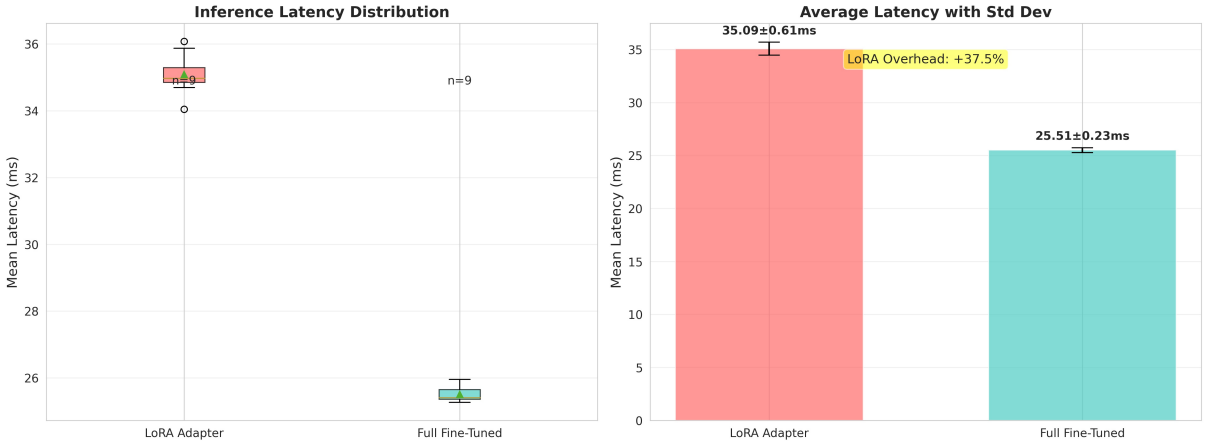

Figure 4: **Deployment latency comparison across serving strategies.** Merged LoRA achieves complete parity with full fine-tuning, while separate adapters show substantial overhead ($+37.5\%$, $p<10^{-6}$). Error bars represent measurement standard deviation.

5 Discussion

Interpreting task-dependent preservation. Across three classification tasks, we observe stronger preservation on SST-2, minimal effects on MRPC after correction, and gains on RTE without preservation. This pattern suggests that the link between representation preservation and accuracy may depend on task properties (e.g., dataset size, label geometry, reliance on lexical vs. relational cues), consistent with prior observations that intermediate representations and downstream accuracy are only loosely coupled (§2).

Mechanistic considerations. Merged adapters matched full fine-tuning latency in our setup, whereas separate adapters were slower. This is consistent with the added runtime low-rank multiply when adapters are applied at inference time, rather than differences in the learned weights themselves (Hu et al., 2021; Chen et al., 2023; Sheng et al., 2023). We view these as implementation-contingent observations specific to our configuration.

Implications for practice. For single-task deployments similar to ours, merging can provide latency comparable to full fine-tuning; separate adapters may still be attractive for multi-task flexibility, particularly in systems optimized for concurrent adapter serving (Chen et al., 2023; Sheng et al., 2023).

Future directions. Promising extensions include: (i) broader task coverage (e.g., larger NLI suites, domain-shifted sentiment) and multilingual settings; (ii) systematic variation of adapter ranks/placements and comparisons with other PEFT methods under the same protocol; (iii) layer-wise analyses using causal interventions or targeted probes; and (iv) multi-tenant serving studies with concurrent workloads and adapter-aware kernels

to identify crossover points between separate and merged adapters.

Note. Detailed constraints on scope and generality appear in §7.

6 Conclusion

In our study of three GLUE classification tasks with TinyLlama-1.1B and rank-8 LoRA, we observe competitive or superior accuracy relative to full fine-tuning, with representation preservation that is task-dependent - substantial on SST-2, marginal on MRPC, and not evident on RTE. For deployment under our configuration, separate adapters incurred higher latency, whereas merging adapters provided latency comparable to full fine-tuning. Practically, LoRA is a viable option for classification in settings similar to ours, and merging at serve time can mitigate latency overhead when single-task performance is prioritized.

7 Limitations

Due to computational constraints, our findings are scoped to three binary classification tasks (MRPC, SST-2, RTE), a single base model (TinyLlama-1.1B), rank-8 LoRA, and batch-1 latency on an NVIDIA L4. While we justify these choices in §3.1 (task diversity, rank diminishing returns, hardware availability), results may differ for larger models, higher ranks, generation tasks, or alternative similarity metrics.

We evaluate only English datasets; results may not extend to morphologically richer languages. Our deployment benchmarks reflect single-GPU serving; batched inference or multi-tenant systems with specialized kernels (Chen et al., 2023; Sheng et al., 2023) face different constraints. Finally, we use linear CKA (Kornblith et al., 2019); task-specific probes or alternative metrics (Raghu et al., 2017; Morcos et al., 2018) could reveal different representational patterns.

8 Ethics Statement

We evaluate widely used public datasets (GLUE) and release code/configs to support reproducibility. No sensitive user data is used.

9 Reproducibility Statement

All code, configurations, trained weights, and optimal hyperparameters are publicly available at `https://github.com/benc61116/NLP`. The repository README includes step-by-step instructions for environment setup (`requirements.txt`), data download via Hugging Face Datasets (Lhoest et al., 2021), sanity checks, hyperparameter optimization with Optuna (Akiba et al., 2019) using TPE (Bergstra et al., 2011), production training with the selected settings, representation extraction and drift computation, and deployment benchmarking (including PEFT merging) (Mangrulkar et al., 2022). We fix random seeds (42/1337/2024), pin software versions, and log runs. All experiments were conducted on a single NVIDIA L4 GPU (24 GB) with CUDA 11.8, PyTorch 2.0.1, Transformers 4.30.2, and PEFT 0.4.0; full dependency versions are specified in the repository.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7319–7328.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 2623–2631.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformers. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1–9.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2546–2554.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2023. Punica: Multi-tenant lora serving. *arXiv preprint arXiv:2310.18547*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7780–7794.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: Efficient low-rank adaptation of large models. *arXiv preprint arXiv:2402.12354*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations (ICLR)*.

Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzębski, Brynn Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2790–2799.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers for transformer models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3519–3529.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of bert. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4365–4374.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3045–3059.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, and 13 others. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, pages 175–184.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4582–4597.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A. Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Sachin Mangrulkar, Lysandre Sylvain, Leandro von Werra, Patrick Lewis, Lewis Tunstall, and 1 others. 2022. Peft: State-of-the-art parameter-efficient fine-tuning. https://github.com/huggingface/peft. GitHub repository.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. 2020. What happens to bert embeddings during fine-tuning? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9666–9676.

Ari S. Morcos, Maithra Raghu, and Samy Bengio. 2018. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2021. On the stability of fine-tuning BERT: Misconceptions, explanations, and strong baselines. In *International Conference on Learning Representations (ICLR)*.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. In *Proceedings of EMNLP 2020: System Demonstrations*, pages 46–54.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the ACL (EACL)*, pages 487–503.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. 2017. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6076–6085.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about BERT. *Transactions of the Association for Computational Linguistics (TACL)*, 8:842–866.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Beidi Chen, Yida Zhuang, Zhuang Chen, Zhihao Jia, Joseph E. Gonzalez, Ion Stoica, and Ce Zhang.

2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Hugo Touvron, Louis Martin, Kevin Stone, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations (ICLR)*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.