TESSA: RETHINKING LORA FOR EFFICIENT MULTI-ADAPTER SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-Rank Adaptation (LoRA) enables parameter-efficient fine-tuning by learning small, task-specific updates. Serving requests with different LoRA adapters on a shared base LLM requires executing adapter-specific low-rank computations during inference while reusing the base model, enabling inexpensive adapter swapping and heterogeneous-adapter batching. While these computations are theoretically lightweight, multi-LoRA serving can incur substantial practical overhead due to two sequential, GPU-unfriendly matrix multiplications and additional inter-GPU communication under tensor parallelism. Specifically, despite adding less than 1% to both computation and parameter memory, LoRA adapters can reduce serving throughput by 15.5% with the reduction growing to 42.9% as tensor parallelism scales. Existing multi-LoRA systems, such as S-LoRA and Punica, optimize adapter execution but retain the two-stage shrink–expand structure, leaving GPU-inefficient small sequential matrix products and adapter-specific inter-GPU communication as persistent bottlenecks. In this paper, we propose Tensor-parallel Shard-aligned Slice Adaptation (TESSA), a new adapter structure tailored to efficient multi-adapter serving, rather than optimizing execution around the existing LoRA structure. TESSA trains contiguous, shard-aligned subsets of the base weight matrix and represents each adapter as a weight delta over the selected subset. Contiguous partial updates enable base and adapter computations to be executed as a single wider, GPU-friendly matrix multiplication. Moreover, shard-aligned placement eliminates additional adapter-specific communication under tensor parallelism. We demonstrate that TESSA substantially improves serving efficiency across diverse workloads while maintaining adaptation quality comparable to LoRA under approximately matched adapter parameter budgets. Our vLLM implementation achieves up to 3.46× the throughput of S-LoRA at approximately matched adapter budgets.

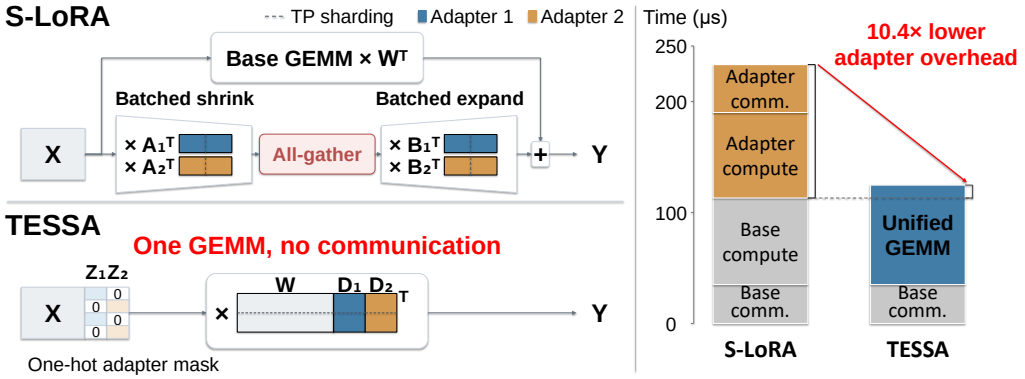
1 INTRODUCTION

Parameter-efficient finetuning allows a pretrained language model to support many tasks, domains, and users by training only a small set of parameters, thereby reducing gradient and optimizer-state storage (Houlsby et al., 2019; Li & Liang, 2021; He et al., 2022). Low-Rank Adaptation (LoRA) is the most widely used approach, representing each update as the product of two trainable low-rank factors (Hu et al., 2022). For a fixed adapter, the update can be merged into the base weight before serving, eliminating separate adapter computation and its associated inference overhead.

However, serving requests for many adapters on a shared base model requires a different execution strategy: maintaining a separately merged copy for each adapter duplicates the base weights, whereas switching among merged weights incurs additional work. Multi-LoRA serving systems therefore store adapters separately and share a single base model across hundreds to thousands of adapters (Sheng et al., 2024; Chen et al., 2026), batching requests that use different adapters to improve GPU utilization (Chen et al., 2024; Sheng et al., 2024). Consequently, the system batches shared base-model computation across requests while applying the adapter for each request online.

These LoRA computations can incur substantial latency despite their low parameter and floating-point operation (FLOP) counts, because the shrink–expand structure of LoRA entails two sequential, GPU-unfriendly matrix multiplications and intermediate memory traffic. For rank-32 LoRA, time to

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.



(a) Adapter computation under tensor parallelism (b) Execution time breakdown

Figure 1: Adapter computation and execution time of TESSA compared with S-LoRA under tensor parallelism. (a) S-LoRA sequentially performs batched shrink and expand operations with an intervening all-gather, whereas TESSA unifies base and adapter computations in one general matrix multiplication (GEMM) without additional adapter-specific communication. (b) Execution-time breakdown for the projections within a single Qwen3-32B decoder layer at TP8 with 32 tokens and four active adapters. S-LoRA uses rank 32, while TESSA uses suffix width 44 for a matched adapter budget. The $10.4\times$ ratio compares S-LoRA adapter computation and communication with TESSA combined computation minus S-LoRA base computation.

first token (TTFT) and time per output token (TPOT) reach 1.3–2.5$\times$ and 1.3–1.6$\times$ base latency on an H100, despite only 1.04–2.25% more parameters (Gowda et al., 2025a). Tensor parallelism (TP) further amplifies this overhead. As the TP degree increases, base model projection latency decreases, whereas the small sequential LoRA operations do not achieve proportional latency reduction and adapter sharding introduces additional inter-GPU collectives.

Prior works mitigate this adapter overhead through grouped kernels, sharding, and fusion (Chen et al., 2024; Sheng et al., 2024; Kong et al., 2024). S-LoRA aligns adapter sharding with the base-model partitioning to reduce communication volume (Sheng et al., 2024), while Block-diagonal LoRA (BD-LoRA) eliminates adapter-specific collectives through block-diagonal factors (Wang et al., 2025a). However, these approaches retain sequential shrink–expand computation of LoRA, leaving GPU-inefficient adapter computation as a persistent bottleneck. Specifically, S-LoRA reduces throughput by 15.5% at TP1, with the loss increasing to 42.9% at TP8 as shown in Figure 2(a).

To address these structural inefficiencies, we propose Tensor-parallel Shard-aligned Slice Adaptation (TESSA), which redesigns adapter training and inference for efficient multi-adapter serving. TESSA trains a contiguous suffix—the last r rows or columns—of the base weight and stores the corresponding weight delta as the adapter. The contiguous partial update replaces the sequential LoRA shrink–expand computation with a single matrix multiplication that can be jointly executed with the base projection through an extended GEMM (Figure 1(a)). Under TP, TESSA places the trainable suffix along the unsharded axis of the base weight and partitions adapter weights along the sharded axis, allowing each GPU to apply the adapter update locally without additional adapter-specific communication. TESSA maintains quality comparable to LoRA and achieves higher serving throughput than S-LoRA at approximately matched adapter budgets. In the fixed-length sweep, TESSA reaches up to $3.46\times$ S-LoRA throughput. Under scaled replay of real-world BurstGPT traffic (Wang et al., 2025c), TESSA achieves up to $127.51\times$ and $14.19\times$ speedups in mean TTFT and TPOT, respectively, relative to the better-performing baseline at each operating point.

In summary, our contributions are:

- **Serving-oriented adapter redesign that removes structural LoRA bottlenecks.** We propose TESSA, a new adapter structure for multi-adapter serving that replaces the sequential shrink–expand path of LoRA with a single matrix product and eliminates adapter-specific

TP communication through shard-aligned updates. TESSA maintains LoRA-comparable adaptation quality under similar parameter budgets.

- **Optimized kernels and workload-aware dispatch integrated into vLLM.** We integrate optimized TESSA kernels and workload-aware dispatch into vLLM (Kwon et al., 2023), achieving up to $3.46\times$ the throughput of S-LoRA at approximately matched adapter budgets.

2 Background and Motivation

2.1 Multi-adapter LoRA serving

Batched adapter computation. LoRA adapts a frozen weight $W \in \mathbb{R}^{h_{\text{out}} \times h_{\text{in}}}$ with a low-rank update $B_i A_i$ (Hu et al., 2022). For a fixed adapter, merging $B_i A_i$ into W eliminates separate adapter computation at inference. However, serving multiple adapters this way requires either maintaining a separate merged weight for each adapter or repeatedly updating the weight when switching adapters, increasing memory usage or switching overhead. Keeping adapters separate allows requests for different adapters to share one frozen base weight and a batched base projection.

Let X_i contain the input tokens assigned to adapter i, with factors $A_i \in \mathbb{R}^{r_i \times h_{\text{in}}}$ and $B_i \in \mathbb{R}^{h_{\text{out}} \times r_i}$. Its output is

$$Y_i = X_i W^\top + \underbrace{(X_i A_i^\top)}_{\text{shrink}} \underbrace{B_i^\top}_{\text{expand}}. \tag{1}$$

The base projection is shared across the mixed-adapter batch, while each adapter applies two sequential low-rank products. To improve adapter execution efficiency, multi-adapter serving systems, including Punica and S-LoRA, use grouped kernels to batch the per-adapter products (Chen et al., 2024; Sheng et al., 2024).

Tensor-parallel LoRA execution. Under Megatron-LM tensor parallelism (Shoeybi et al., 2019), column-parallel projections partition the output dimension across GPUs, whereas row-parallel projections partition the input dimension and combine partial outputs with an all-reduce. LoRA factor sharding must account for the corresponding base-model partitioning to avoid unnecessary communication. S-LoRA provides a representative fully sharded design, as illustrated in Figure 8 in Appendix A.3: the column-parallel up projection introduces an all-gather between shrink and expand, while the row-parallel down projection introduces an all-reduce (Sheng et al., 2024).

2.2 Sources of LoRA serving overhead

GPU execution overhead. LoRA executes two sequential, narrow GEMMs, incurring fixed execution costs and intermediate memory traffic (Gowda et al., 2025a; Li et al., 2026; Gowda et al., 2025b; Alsuradi et al., 2026). At small token batches, the narrow sequential GEMMs expose insufficient parallelism to fully utilize the GPU or amortize fixed overhead, causing adapter computation to occupy a disproportionately large fraction of projection latency, as shown in Figure 2(b). At larger batches, the bottleneck shifts to activation movement: the shrink–expand path reads and writes large activation tensors and materializes the low-rank intermediate, leaving LoRA memory-bound despite its small parameter and FLOP counts. Our time-based Roofline analysis characterizes these latency- and memory-bound regimes in Appendix A.2.

Tensor-parallel overhead. As described in Section 2.1, the fully sharded TP layout introduced by S-LoRA (Sheng et al., 2024) introduces substantial adapter-specific communication overhead. For Qwen3-32B under a decode-heavy workload, fully sharded LoRA retains 84.5% of the base-only throughput at TP1 but only 57.1% at TP8, as shown in Figure 2(a). Figure 2(b) shows adapter-collective overhead rising from 20% of base-projection latency at TP2 to 51% at TP8. Despite smaller messages, adapter collectives cost more than base collectives because they require twice as many launches, whose fixed latency dominates during decode (Appendix A.3).

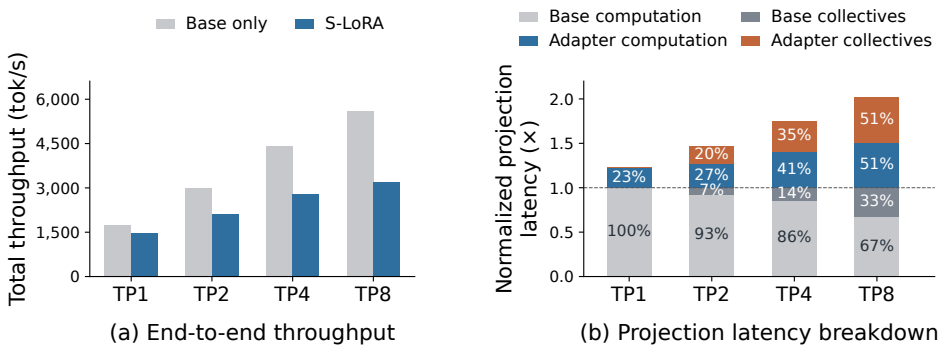


Figure 2: Tensor-parallel LoRA overhead on Qwen3-32B under a decode-heavy workload (H100, bf16, rank 32, one active adapter; input/output lengths 128/256, concurrency 32). (a) Absolute total throughput of the base-only and fully sharded LoRA servers across TP 1/2/4/8. (b) Latency decomposition of the QKV, output, and MLP projections within one transformer layer at $M = 32$, normalized to the corresponding base-only projection latency ($1\times$). Segment labels give percentages of that baseline. Prefill-heavy results are provided in Appendix A.3.

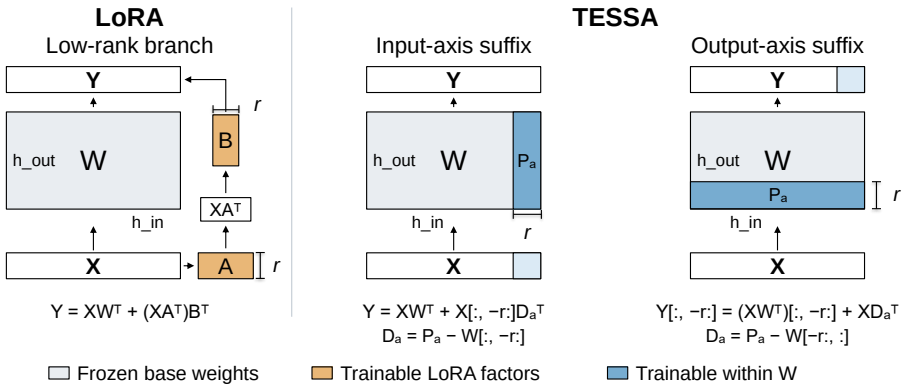


Figure 3: Comparison of LoRA and TESSA for a projection with $W \in \mathbb{R}^{h_{\text{out}} \times h_{\text{in}}}$. LoRA uses two sequential low-rank products, whereas TESSA trains a contiguous suffix of the base weight, reducing adapter computation to one product and enabling joint execution with the base GEMM. Input- and output-axis variants train the last r columns and rows, respectively. The adapter weight D_a is the difference between the trained suffix P_a and its frozen base values.

3 TESSA

TESSA redesigns adapter computation to address the execution and communication overheads identified in Section 2.2. Contiguous suffix updates enable unified GEMM execution (Section 3.2), while shard-aligned placement eliminates additional adapter-specific TP communication (Section 3.3).

3.1 Fixed-suffix adaptation

As illustrated in Figure 3, TESSA trains a contiguous suffix P_a of width r in the base weight W, rather than introducing separate low-rank factors, while keeping the remaining weights frozen. Let W_a denote the resulting adapted weight. After training, we store only the suffix difference $D_a = P_a - W_{\text{slice}}$ as the adapter, where W_{slice} is the corresponding frozen base suffix. The suffix axis is selected before training and remains fixed. We use zero-based, Python-style slicing notation: $Z[:, -r:]$ denotes the last r columns of Z, while $Z[:, :-r]$ denotes the remaining columns.

Suffix-axis variants. TESSA supports input- and output-axis suffix variants. The input-axis variant trains the last r columns of W and reads the corresponding input suffix, whereas the output-axis

Algorithm 1 Unified GEMM for multi-adapter suffix updates

Require: Input $X \in \mathbb{R}^{M \times h_{\text{in}}}$, base weight $W \in \mathbb{R}^{h_{\text{out}} \times h_{\text{in}}}$,
A active adapter weights $\{D_a\}_{a=1}^{A}$, suffix width r,
and an assignment $a(m) \in \{1, \ldots, A\}$ specifying
the active adapter used by each token $m = 1, \ldots, M$
Ensure: Adapted output $Y \in \mathbb{R}^{M \times h_{\text{out}}}$

procedure INPUT-AXIS SUFFIX
  $Z_a[m, :] \leftarrow \mathbf{1}[a(m) = a]\, X[m, -r:]$ for all a, m
  $\widetilde{X} \leftarrow [\, X \mid Z_1 \mid \cdots \mid Z_A \,]$
  $\widetilde{W} \leftarrow [\, W \mid D_1 \mid \cdots \mid D_A \,]$
  $Y \leftarrow \widetilde{X}\widetilde{W}^{\top}$ ▷ Unified GEMM
  return Y
end procedure

procedure OUTPUT-AXIS SUFFIX
  $\widetilde{W} \leftarrow [\, W^{\top} \mid D_1^{\top} \mid \cdots \mid D_A^{\top} \,]^{\top}$
  $[\, Y_{\text{base}} \mid \Delta Y_1 \mid \cdots \mid \Delta Y_A \,] \leftarrow X\widetilde{W}^{\top}$ ▷ Unified GEMM
  $Y \leftarrow Y_{\text{base}}$
  $Y[m, -r:] \leftarrow Y[m, -r:] + \Delta Y_{a(m)}[m, :]$ for all m
  return Y
end procedure

variant trains the last r rows and updates only the corresponding output suffix, leaving the remaining output columns unchanged from the base projection. At inference, the two variants compute

$$\begin{aligned} \text{Input-axis:} \quad & Y = XW^{\top} + X[:, -r:]D_a^{\top}, \\ \text{Output-axis:} \quad & Y[:, -r:] = (XW^{\top})[:, -r:] + XD_a^{\top}. \end{aligned} \tag{2}$$

For the input-axis variant, $D_a \in \mathbb{R}^{h_{\text{out}} \times r}$, whereas for the output-axis variant, $D_a \in \mathbb{R}^{r \times h_{\text{in}}}$. Equivalently, both variants can be interpreted as the LoRA update in Equation 1 with one factor fixed to a coordinate-selection matrix. The input-axis variant replaces the learned shrink with input slicing, while the output-axis variant replaces the learned expand with placement into the output suffix. Each therefore requires only one adapter matrix product.

Adaptation capacity. Prior work provides convergence analysis of partial-weight adaptation (Woo et al., 2025) and demonstrates LoRA-comparable quality by updating selected pretrained-weight connections (Woo et al., 2025; Namkung et al., 2026). TESSA further restricts these connections to a fixed, contiguous suffix. Our evaluations show that this restriction preserves LoRA-comparable quality under approximately matched parameter budgets (Section 4.1). We also observe faster early validation-loss reduction than PaCA with randomly selected input columns in a controlled comparison (Appendix C.5).

3.2 EFFICIENT EXECUTION OF ADAPTER UPDATES

Unified GEMM execution. Requests for different adapters sharing the same weight W can be batched. Let $a(m)$ denote the adapter assigned to the m-th token. The batch shares a single base projection $XW^{\top}$, while each token receives its assigned adapter update by applying Equation 2 token-wise with $D_{a(m)}$.

Beyond sharing the base projection across requests, TESSA combines the base and adapter matrix products into a single GEMM (Algorithm 1 and Figure 3). For the input-axis variant, we concatenate W and the adapter weights along the input axis, and X with input-suffix blocks masked by adapter assignment. The resulting GEMM directly computes the base output with each token's assigned adapter update. For the output-axis variant, we stack the adapter weights below W to compute the base and all adapter outputs in one GEMM, then add each token's assigned adapter output to its output suffix.

Benefits over LoRA. The single-product adapter formulation provides two complementary efficiency benefits. In latency-bound regimes, replacing the sequential LoRA shrink–expand path with a single adapter product reduces fixed execution overhead; unified GEMM further avoids separate adapter kernel launches and exposes more parallel work. In memory-bound regimes, TESSA reduces activation traffic: the input-axis variant reads only the selected input suffix and produces a full output, while the output-axis variant reads the full input but produces only the selected output suffix. In contrast, LoRA reads the full input in shrink and writes the full output in expand, additionally writing and reading the low-rank intermediate if its two stages execute separately, as in Punica and S-LoRA. For square projections of width h under exactly matched adapter budgets, either TESSA variant saves Mh activation accesses at the same adapter FLOPs. The resulting arithmetic intensity of the adapter operation approaches twice that of LoRA for small ranks and large token batches (Appendix A.1).

Workload-aware execution. Despite these latency and memory efficiency benefits, unified GEMM execution can be slower than computing adapter updates separately from the base projection. As shown in Algorithm 1, multi-adapter batching introduces redundant computation on masked input blocks or unselected adapter outputs. For A active adapters with suffix width r, the ratios of redundant FLOPs to base-projection FLOPs are $(A-1)r/h_{\text{in}}$ and $(A-1)r/h_{\text{out}}$ for the input- and output-axis variants, respectively, potentially outweighing the benefits of unified execution as A or r increases. We therefore introduce workload-aware dispatch to select between separate and unified execution based on the suffix width, hidden dimension, number of active adapters, and token count. Appendix B.1 derives the overhead, and Appendix B.2 details the dispatch.

3.3 Shard-aligned suffix

To avoid the additional communication incurred by fully sharded LoRA (Section 2.2), TESSA selects the suffix form from Section 3.1 according to the tensor-parallel partitioning of the base weight: the suffix lies along the unsharded axis. Shard-aligned placement allows each rank to compute and apply the adapter update locally, while evenly distributing adapter parameters and computation across ranks. Placing the suffix along the sharded axis would instead concentrate adapter parameters and computation on the ranks overlapping the suffix.

Concretely, for a column-parallel projection, where each rank k holds an output-axis shard $W^{(k)}$ of the base weight, we choose an input-axis suffix and partition D_a along the output axis. Each rank computes its output shard as

$$Y^{(k)} = X(W^{(k)})^\top + X[:, -r:](D_a^{(k)})^\top. \tag{3}$$

For a row-parallel projection, where each rank k holds an input-axis shard $W^{(k)}$ and the corresponding input $X^{(k)}$, we choose an output-axis suffix and partition D_a along the input axis. Each rank computes its partial output as

$$\begin{aligned} P^{(k)} &= X^{(k)}(W^{(k)})^\top, \\ P^{(k)}[:, -r:] &\leftarrow P^{(k)}[:, -r:] + X^{(k)}(D_a^{(k)})^\top. \end{aligned} \tag{4}$$

The existing all-reduce for the base projection then sums these partial outputs, requiring no additional collective. Figure 4 illustrates this strategy on a two-projection MLP following the Megatron-LM tensor-parallel layout. The same shard placement applies to multi-adapter serving, with each token using its assigned adapter.

4 Evaluation

We compare the fine-tuning quality of TESSA with that of LoRA at approximately matched adapter parameter budgets, and its serving performance with that of S-LoRA (Sheng et al., 2024) and BD-LoRA (Wang et al., 2025a) on fixed-length and scaled BurstGPT workloads (Wang et al., 2025c).

4.1 Quality at matched parameter budgets

Setup. We compare TESSA and LoRA at approximately matched adapter parameter budgets on LLaMA-3.1-8B (Grattafiori et al., 2024), Qwen3-8B, and Qwen3-14B (Yang et al., 2025) across

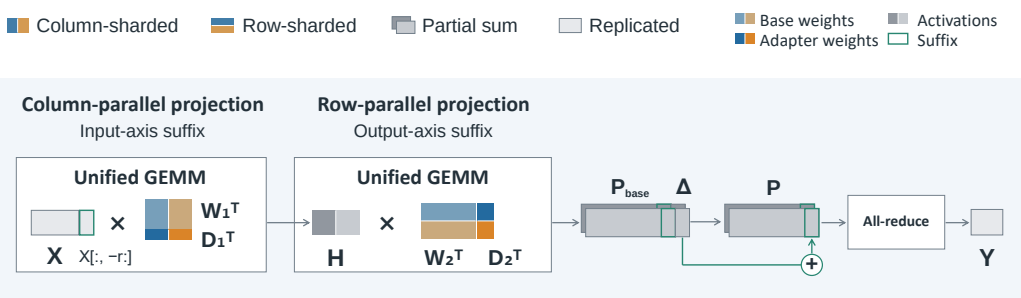


Figure 4: Shard-aligned TESSA on a two-projection MLP. The column-parallel up and row-parallel down projections use input- and output-axis suffixes, respectively (green outlines). Both apply adapter updates locally without additional adapter-specific communication.

Table 1: Downstream task performance at approximately matched adapter parameter budgets. Adapted scores are means over three seeds. Parameter counts and confidence intervals are reported in Table 5. The commonsense average covers eight tasks (Table 4). *Base uses no fine-tuning.

		Math		Code		Commonsense
Model	Method	GSM8K	MATH-500	HumanEval	MBPP	Avg.
LLaMA-3.1-8B	Base*	14.56	12.80	34.76	58.73	20.57
	LoRA	79.33	**38.47**	51.63	**67.11**	83.53
	TESSA	**79.63**	38.40	**53.25**	67.02	**87.00**
Qwen3-8B-Base	Base*	44.66	34.80	84.76	78.31	7.19
	LoRA	89.71	81.73	86.18	82.36	88.60
	TESSA	**94.44**	**87.53**	**86.59**	**82.98**	**88.84**
Qwen3-14B-Base	Base*	85.22	71.60	83.54	83.86	36.66
	LoRA	**95.15**	90.80	**89.23**	87.92	91.45
	TESSA	95.10	**91.20**	88.62	**88.10**	**91.51**

mathematical reasoning, code generation, and commonsense reasoning. We fine-tune on the 1K math subset selected by DDCF (Hoang et al., 2026) for mathematics, Magicoder-OSS-Instruct-75K (Wei et al., 2024) for code, Commonsense170K (Hu et al., 2023) for commonsense reasoning. We evaluate on GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021; Lightman et al., 2024), HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), and eight commonsense benchmarks. Both methods adapt the attention query/output and MLP gate/up/down projections; code generation additionally adapts the attention key/value projections. Results are averaged over three seeds. Full training and evaluation details are provided in Appendix C.1.

Downstream task performance. Table 1 shows that TESSA achieves broadly comparable performance to LoRA at approximately matched adapter budgets. For example, on Qwen3-8B, TESSA achieves 94.44% on GSM8K and 87.53% on MATH-500, exceeding the LoRA means by 4.73 and 5.80 percentage points. On LLaMA-3.1-8B, TESSA also achieves a higher commonsense average of 87.00%, compared with 83.53% for LoRA. The differences are smaller on Qwen3-14B: differences remain within 0.61 percentage points across all five metrics. A similar pattern holds for the mixture-of-experts (MoE) models Qwen1.5-MoE-A2.7B (Qwen Team, 2024) and Qwen3-30B-A3B across six downstream tasks, where TESSA achieves higher mean scores on four tasks for each model and slightly lower scores on the remaining two (Table 7, Appendix C.4). Additional GLUE and MMLU evaluations show competitive performance relative to LoRA, with an additional comparison to published PaCA results on MMLU (Appendix C.3).

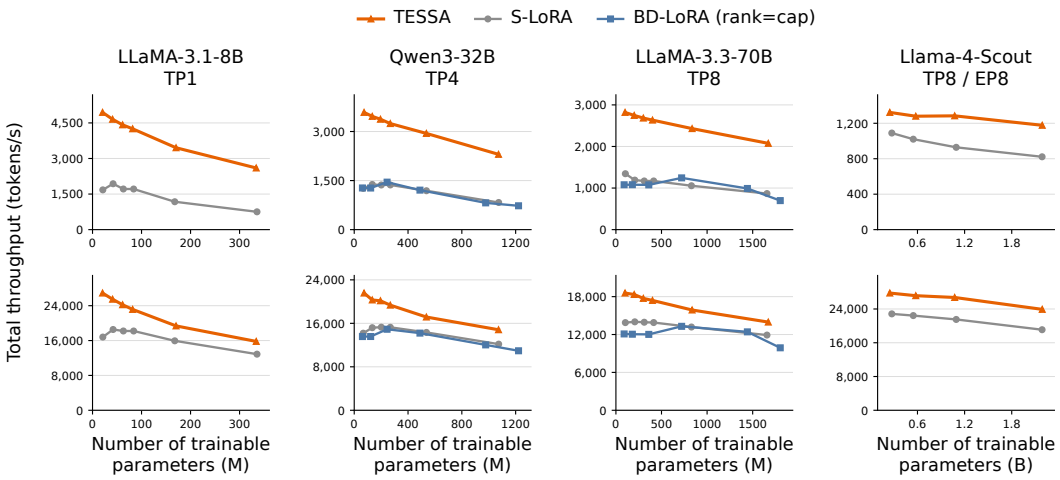


Figure 5: Serving throughput under decode-heavy (top) and prefill-heavy (bottom) workloads with a 32-adapter request pool. S-LoRA and TESSA use approximately matched parameter budgets; BD-LoRA uses an independent rank sweep, as detailed in Appendix D.1.

4.2 Serving efficiency under fixed-length workloads

Workload and metrics. We measure total throughput on H100 GPUs at a maximum request concurrency of 32, using decode-heavy and prefill-heavy workloads with input/output lengths $128/512$ and $8192/128$, respectively. We evaluate LLaMA-3.1-8B at TP1, Qwen3-32B at TP4, LLaMA-3.3-70B (Meta AI, 2024) at TP8, and Llama-4-Scout (Meta AI, 2025) at TP8 with expert parallelism (EP8), varying adapter parameter budgets over a fixed request pool of 32 adapters. S-LoRA and TESSA use approximately matched parameter budgets, while BD-LoRA is evaluated at supported rank sizes to avoid padding overhead in its implementation. We omit BD-LoRA at TP1, where it reduces to LoRA, and on MoE models, which its implementation does not support. Full settings are provided in Appendix D.1.

Serving performance. Figure 5 shows that TESSA outperforms S-LoRA across all evaluated configurations and BD-LoRA over their overlapping parameter ranges. On dense models, geometric-mean throughput gains over S-LoRA are 2.27–$2.78\times$ for decode-heavy workloads and 1.26–$1.33\times$ for prefill-heavy workloads. The larger decode gains are consistent with the greater impact of fixed execution and communication costs (Section 2.2), which TESSA reduces through unified GEMM execution and shard-aligned updates. Scout uses EP within MoE layers, leaving less adapter-specific TP communication to eliminate, yet achieves gains of 23.0–31.8%. This advantage over S-LoRA extends to other MoE models, as shown in Appendix D.2. BD-LoRA improves on S-LoRA in some decode-heavy configurations but provides no consistent advantage and is sometimes slower, despite eliminating adapter-specific TP communication. This highlights the remaining cost of its shrink–expand computation. TESSA addresses both computation and communication overhead, maintaining gains across both workloads.

4.3 Serving efficiency on production-trace replay

Workload and setup. We evaluate Qwen3-32B (TP4) and LLaMA-3.3-70B (TP8) on H100 GPUs using open-loop replay of scaled BurstGPT traces (Wang et al., 2025c), with 150 and 200 resident adapters, respectively. We vary the offered load from 10 to 90 req/s and report mean TTFT, mean TPOT, request throughput, and P95 end-to-end latency. S-LoRA and TESSA use approximately matched adapter budgets; BD-LoRA uses smaller adapters because approximately matched configurations run out of memory. Full settings are provided in Appendix D.1.

Results. The serving gains of TESSA extend to scaled production-trace replay, as shown in Figure 6. Throughput is similar at low loads, but as load increases, TESSA achieves higher throughput and

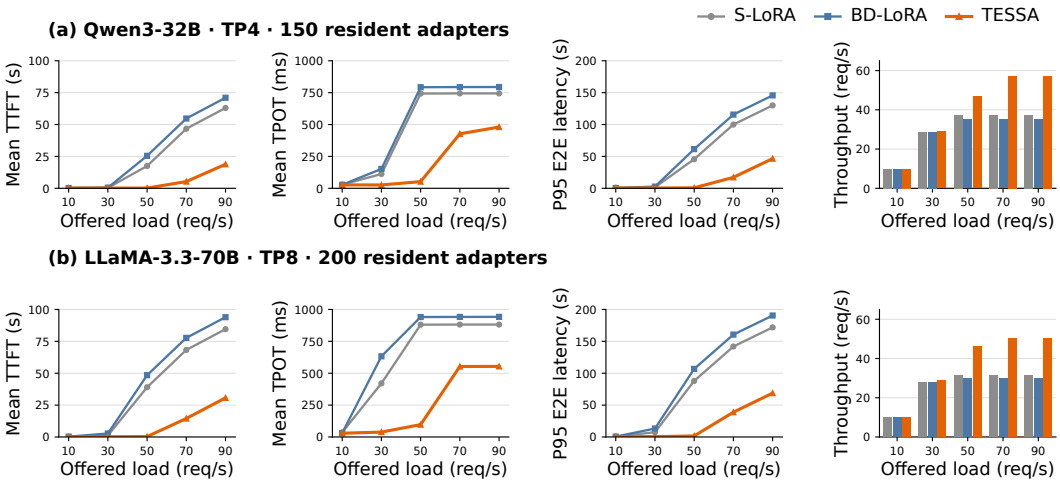


Figure 6: Serving performance on scaled BurstGPT workloads: (a) Qwen3-32B at TP4 with 150 resident adapters and (b) LLaMA-3.3-70B at TP8 with 200. Columns show mean TTFT, mean TPOT, P95 end-to-end request latency, and achieved request throughput. Each point is one replay. BD-LoRA uses a smaller adapter budget (Appendix D.1).

lower latency while the baselines saturate. At 50 req/s, mean TTFT stays below 0.31 seconds on both models, versus 17.47 and 39.01 seconds for S-LoRA on Qwen3-32B and LLaMA-3.3-70B. At 90 req/s, TESSA achieves 1.54× and 1.60× the throughput of S-LoRA, the stronger baseline at this load. Mean TPOT and P95 end-to-end latency are also lower across all measured loads.

5 Related Work

Several works improve serving efficiency by changing the adapter design. **BD-LoRA** uses block-diagonal factors to eliminate adapter-specific collectives, while retaining the shrink–expand computation of LoRA (Wang et al., 2025a). **zFLoRA** and **GAWS** redesign adapter computation for inference efficiency, but do not establish efficient multi-adapter serving under TP (Gowda et al., 2025a; Alsuradi et al., 2026). TESSA jointly addresses inefficient shrink–expand computation and adapter-specific TP communication for heterogeneous multi-adapter serving. A separate body of work targets training efficiency through adapter redesign or execution optimization (Woo et al., 2025; Khaki et al., 2025; Zhu et al., 2026; Wang et al., 2025b; Namkung et al., 2026). These methods primarily optimize fine-tuning rather than multi-adapter serving.

Another line of work optimizes LoRA serving without changing the adapter parameterization. These include heterogeneous batching and TP optimization (Chen et al., 2024; Sheng et al., 2024), kernel autotuning (Zhou et al., 2025), dynamic merging and request migration (Wu et al., 2024), and rank-aware scheduling with CPU-assisted execution during adapter loading (Li et al., 2024). These scheduling and adapter-management policies are largely complementary to TESSA.

6 Conclusion

We presented TESSA, a serving-oriented adapter design that addresses the computational and communication overheads of multi-tenant LoRA serving. By training contiguous, shard-aligned suffixes of base weights, TESSA replaces sequential shrink–expand computation of LoRA with a single adapter matrix product that can be combined with the base projection, and eliminates additional adapter-specific TP collectives. Integrated into vLLM with optimized kernels and workload-aware dispatch, TESSA improves serving performance over S-LoRA and BD-LoRA while maintaining adaptation quality comparable to LoRA under similar parameter budgets. These results highlight adapter structure as a key consideration in designing efficient multi-adapter serving systems.

AI USE STATEMENT

We used generative AI tools to assist with literature search, manuscript revision, mathematical derivations, and the preparation of figures and tables in the paper. The tools also assisted with code implementation and debugging, including TESSA kernels, kernel-parameter tuning, and benchmark scripts and execution. The core research ideas and hypotheses were developed by the authors. AI tools assisted with reviewing the experimental methodology and analyzing and interpreting results. The authors directed the research, selected the datasets, evaluation tasks, and training configurations, and made the final decisions on method design, experimental methodology, and interpretation of results. The authors reviewed all AI-assisted outputs and take full responsibility for the final manuscript, including its claims, results, code, and other artifacts. We did not use generative AI tools to generate synthetic datasets, to formulate the hypotheses of this work, to translate text, to clean or reformat datasets.

REFERENCES

Eman Alsuradi, Junhyun Lee, Kyenghun Lee, Hyeonmok Ko, and Fahed Jubair. Grouped adaptive weight sharing (GAWS): An inference-efficient adaptation method for large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 31790–31806, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, et al. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, et al. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.

Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Katz, and Nikolaos Aletras. LexGLUE: A benchmark dataset for legal language understanding in English. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4310–4330, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.297. URL https://aclanthology.org/2022.acl-long.297/.

Hongyu Chen, Letian Ruan, Zilin Xu, Yuchen Li, Xinyu Chen, Jingwen Leng, Bingsheng He, Minyi Guo, and Shixuan Sun. InfiniLoRA: Disaggregated multi-LoRA serving for large language models. *arXiv preprint arXiv:2604.07173*, 2026.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Christopher Clark, Kenton Lee, Ming-Wei Chang, et al. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 2924–2936, 2019. doi: 10.18653/v1/N19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, et al. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

GLM-4.5 Team. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Dhananjaya Gowda, Seoha Song, Harshith Goka, and Junhyun Lee. zFLoRA: Zero-latency fused low-rank adapters. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21401–21418, 2025a. URL https://aclanthology.org/2025.emnlp-main.1086/.

Dhananjaya Gowda, Seoha Song, Junhyun Lee, and Harshith Goka. FLoRA: Fused forward-backward adapters for parameter efficient fine-tuning and reducing inference-time latencies of LLMs. *arXiv preprint arXiv:2511.00050*, 2025b.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, et al. Measuring Mathematical Problem Solving With the MATH Dataset. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2103.03874.

Long P. Hoang, Wenxuan Zhang, and Wei Lu. Difficulty–diversity collaborative filtering for data-efficient LLM fine-tuning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=n9mXlqD2SJ.

Houston Hoffman and Fred Oh. Constant time launch for straight-line CUDA graphs and other performance enhancements. NVIDIA Technical Blog, September 2024. URL https://developer.nvidia.com/blog/constant-time-launch-for-straight-line-cuda-graphs-and-other-performance-enhancements/.

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 (ICML)*, 2019.

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 (ICLR)*, 2022.

Zhiqiang Hu, Lei Wang, Yihuai Lan, et al. LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5254–5276, 2023. doi: 10.18653/v1/2023.emnlp-main.319.

Samir Khaki, Xiuyu Li, Junxian Guo, Ligeng Zhu, Konstantinos N. Plataniotis, Amir Yazdanbakhsh, Kurt Keutzer, Song Han, and Zhijian Liu. SparseLoRA: Accelerating LLM fine-tuning with contextual sparsity. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://proceedings.mlr.press/v267/khaki25a.html.

Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, Barry Haddow, Philipp Koehn, Benjamin Marie, Christof Monz, Makoto Morishita, Kenton Murray, Masaaki Nagata, Toshiaki Nakazawa, Martin Popel, Maja Popović, Mariya Shmatova, and Jun Suzuki. Findings of the 2023 conference on machine translation (WMT23): LLMs are here but not quite there yet. In Philipp Koehn, Barry Haddow, Tom Kocmi, and Christof Monz (eds.), *Proceedings of the Eighth Conference on Machine Translation*, pp. 1–42, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.wmt-1.1. URL https://aclanthology.org/2023.wmt-1.1/.

Rui Kong, Qiyang Li, Xinyu Fang, Qingtian Feng, Qingfeng He, Yazhu Dong, Weijun Wang, Yuanchun Li, Linghe Kong, and Yunxin Liu. LoRA-Switch: Boosting the efficiency of dynamic LLM adapters via system-algorithm co-design. *arXiv preprint arXiv:2405.17741*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Qiyang Li, Rui Kong, Yuchen Li, Hengyi Cai, Shuaiqiang Wang, Linghe Kong, Guihai Chen, and Dawei Yin. AdaFuse: Accelerating dynamic adapter inference via token-level pre-gating and fused kernel optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. doi: 10.1609/aaai.v40i37.40435.

Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, and Wei Wang. CaraServe: CPU-assisted and rank-aware LoRA serving for generative LLM inference. *arXiv preprint arXiv:2401.11240*, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL*, 2021.

Hunter Lightman, Vineet Kosaraju, Yura Burda, et al. Let's Verify Step by Step. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.20050.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, et al. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.01210.

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 *International Conference on Machine Learning (ICML)*, 2024.

Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J. Ligocki, Matthew J. Cordery, Nicholas J. Wright, Mary W. Hall, and Leonid Oliker. Roofline model toolkit: A practical tool for architectural and program analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation (PMBS)*, volume 8966 of *Lecture Notes in Computer Science*, pp. 129–148, 2015. doi: 10.1007/978-3-319-17248-4_7.

Meta AI. Llama 3.3 model card. https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md, December 2024.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. Meta AI Blog, April 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

Todor Mihaylov, Peter Clark, Tushar Khot, et al. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2381–2391, 2018. doi: 10.18653/v1/D18-1260.

Sol Namkung, Sunghyeon Woo, Inho Jeong, and Dongsuk Jeon. RAPA: Recursively aligned pathway adaptation of large language models, 2026. URL https://openreview.net/forum?id=QTNkVX15o5.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. MedMCQA: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning (CHIL)*, volume 174 of *Proceedings of Machine Learning Research*, pp. 248–260, 2022. URL https://proceedings.mlr.press/v174/pal22a.html.

Qwen Team. Qwen1.5-MoE: Matching 7B model performance with 1/3 activated parameters. Qwen Blog, March 2024. URL https://qwenlm.github.io/blog/qwen-moe/.

Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency. Qwen Blog, September 2025. URL https://qwen.ai/blog?id=qwen3-next.

Ricardo Rei, Craig Stewart, Ana C. Farinha, and Alon Lavie. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2685–2702, 2020. doi: 10.18653/v1/2020.emnlp-main.213.

Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C. Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pp. 578–585, 2022. doi: 10.18653/v1/2022.wmt-1.52.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, et al. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399.

Maarten Sap, Hannah Rashkin, Derek Chen, et al. Social IQa: Commonsense Reasoning about Social Interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. doi: 10.18653/v1/D19-1454.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023. URL https://arxiv.org/abs/2307.09288.

Don Tuggener, Pius von Däniken, Thomas Peetz, and Mark Cieliebak. LEDGAR: A large-scale multi-label corpus for text classification of legal provisions in contracts. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (eds.), *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pp. 1235–1241, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.155/.

Xinyu Wang, Jonas M. Kübler, Kailash Budhathoki, Yida Wang, and Matthäus Kleindessner. Block-diagonal LoRA for eliminating communication overhead in tensor parallel LoRA serving. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a.

Yujin Wang, Shunan Dong, Zongle Huang, Yichen You, Liu He, Huazhong Yang, Yongpan Liu, and Hongyang Jia. HyC-LoRA: Memory efficient LoRA fine-tuning with hybrid activation compression. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025b. URL https://github.com/thu-ee-acts-lab/HyC-LoRA-release.

Yunsong Wang, Charlene Yang, Steven Farrell, Yan Zhang, Thorsten Kurth, and Samuel Williams. Time-based roofline for deep learning performance analysis. In *IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, pp. 10–19, 2020. doi: 10.1109/DLS51937.2020.00007.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A real-world workload dataset to optimize LLM serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*. ACM, 2025c. doi: 10.1145/3711896.3737413. URL https://doi.org/10.1145/3711896.3737413.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 52632–52657. PMLR, 2024. URL https://proceedings.mlr.press/v235/wei24h.html.

Sunghyeon Woo, Sol Namkung, Sunwoo Lee, Inho Jeong, Beomseok Kim, and Dongsuk Jeon. PaCA: Partial connection adaptation for efficient fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2025.

Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2024.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, et al. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4791–4800, 2019. doi: 10.18653/v1/P19-1472.

Changhai Zhou, Yuhua Zhou, Shiyang Zhang, Yibin Wang, and Zekai Liu. Dynamic operator optimization for efficient multi-tenant LoRA model serving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 22910–22918, 2025. doi: 10.1609/aaai.v39i21.34453. URL https://ojs.aaai.org/index.php/AAAI/article/view/34453.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, et al. Instruction-Following Evaluation for Large Language Models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

Zhanda Zhu, Qidong Su, Yaoyao Ding, Kevin Song, Shang Wang, and Gennady Pekhimenko. LoRAFusion: Efficient LoRA fine-tuning for LLMs. In *Proceedings of the 21st European Conference on Computer Systems (EuroSys)*, 2026. URL https://arxiv.org/abs/2510.00206.

A Analysis of LoRA execution overhead

This appendix supplements Section 2.2 with GPU execution analysis and tensor-parallel communication measurements. We first derive the arithmetic intensity of LoRA and TESSA for general projections at matched parameter budgets, then apply a time-based Roofline model to a square-projection case study, and finally examine collective costs under decode- and prefill-heavy workloads.

A.1 Arithmetic intensity of adapter computation

In this section, we compare the logical arithmetic intensity of the LoRA and TESSA adapter computation under an equal trainable-parameter budget.

Parameter budget and assumptions. Let $X \in \mathbb{R}^{M\times h_{\text{in}}}$ denote the input activations for M tokens and $W \in \mathbb{R}^{h_{\text{out}}\times h_{\text{in}}}$ the base weight, with the projection given by $Y = XW^\top$. A rank-r_L LoRA adapter uses factors $A \in \mathbb{R}^{r_L\times h_{\text{in}}}$ and $B \in \mathbb{R}^{h_{\text{out}}\times r_L}$, with output $(XA^\top)B^\top$ and parameter count

$$P = r_L(h_{\text{in}} + h_{\text{out}}). \tag{5}$$

The input-axis suffix variant of TESSA uses an adapter weight of shape $h_{\text{out}} \times r_I$, whereas the output-axis suffix variant uses one of shape $r_O \times h_{\text{in}}$. Matching P gives

$$r_I = \frac{P}{h_{\text{out}}}, \qquad r_O = \frac{P}{h_{\text{in}}}. \tag{6}$$

When calculating logical FLOPs and memory traffic, we count each multiply–add as two FLOPs and assume BF16 operands (two bytes per element). Following the implementations in S-LoRA and Punica (Sheng et al., 2024; Chen et al., 2024), we model LoRA adapter computation as two separate GEMMs, with the intermediate tensor written to and read from memory. We exclude the base projection, accumulation into its output, packing, masking, and inter-GPU communication.

Operation counts and arithmetic intensity. Each method performs $2MP$ FLOPs. LoRA reads Mh_{in} input elements and writes Mr_L during shrink, and reads Mr_L and writes Mh_{out} output elements during expand, in addition to reading P weight elements. The TESSA input-axis variant reads only Mr_I input elements and writes Mh_{out} output elements, whereas the output-axis variant reads Mh_{in} input elements and writes only Mr_O output elements. Their logical traffic in bytes is therefore

$$\begin{aligned} Q_{\text{LoRA}} &= 2\big[P + M(h_{\text{in}} + h_{\text{out}} + 2r_L)\big], \\ Q_{\text{input}} &= 2\big[P + M(h_{\text{out}} + r_I)\big], \\ Q_{\text{output}} &= 2\big[P + M(h_{\text{in}} + r_O)\big]. \end{aligned} \tag{7}$$

Dividing $2MP$ by these counts gives arithmetic intensity in FLOP/byte, where the superscripts in and out denote the input- and output-axis suffix variants of TESSA, respectively:

$$\begin{aligned} \text{AI}_{\text{LoRA}} &= \frac{MP}{P + M(h_{\text{in}} + h_{\text{out}} + 2r_L)}, \\ \text{AI}^{\text{in}}_{\text{TESSA}} &= \frac{MP}{P + M(h_{\text{out}} + r_I)}, \\ \text{AI}^{\text{out}}_{\text{TESSA}} &= \frac{MP}{P + M(h_{\text{in}} + r_O)}. \end{aligned} \tag{8}$$

For feasible, nonzero suffix widths, both TESSA denominators are smaller than the LoRA denominator due to smaller activation traffic.

The ratios of LoRA arithmetic intensity to that of the two TESSA variants are:

$$\begin{aligned} \frac{\text{AI}_{\text{LoRA}}}{\text{AI}^{\text{in}}_{\text{TESSA}}} &= \frac{P + M(h_{\text{out}} + r_I)}{P + M(h_{\text{in}} + h_{\text{out}} + 2r_L)}, \\ \frac{\text{AI}_{\text{LoRA}}}{\text{AI}^{\text{out}}_{\text{TESSA}}} &= \frac{P + M(h_{\text{in}} + r_O)}{P + M(h_{\text{in}} + h_{\text{out}} + 2r_L)}. \end{aligned} \tag{9}$$

When activation traffic dominates weight traffic and suffix terms are negligible in the traffic expressions, i.e., $P \ll M \min(h_{\text{in}}, h_{\text{out}})$ and $P \ll \min(h_{\text{in}}, h_{\text{out}})^2$, the ratios approach

$$\frac{\text{AI}_{\text{LoRA}}}{\text{AI}^{\text{in}}_{\text{TESSA}}} \approx \frac{h_{\text{out}}}{h_{\text{in}} + h_{\text{out}}}, \qquad \frac{\text{AI}_{\text{LoRA}}}{\text{AI}^{\text{out}}_{\text{TESSA}}} \approx \frac{h_{\text{in}}}{h_{\text{in}} + h_{\text{out}}}. \tag{10}$$

Equivalently, the input- and output-axis variants provide approximately $1 + h_{\text{in}}/h_{\text{out}}$ and $1 + h_{\text{out}}/h_{\text{in}}$ times the arithmetic intensity of LoRA, respectively.

Traffic reduction at matched parameter budgets. Subtracting the TESSA traffic counts from the LoRA count cancels weight reads under an exactly matched parameter budget:

$$\begin{aligned} \Delta Q_I &\equiv Q_{\text{LoRA}} - Q_{\text{input}} = 2M(h_{\text{in}} + 2r_L - r_I), \\ \Delta Q_O &\equiv Q_{\text{LoRA}} - Q_{\text{output}} = 2M(h_{\text{out}} + 2r_L - r_O). \end{aligned} \tag{11}$$

For $M > 0$ and $r_L > 0$, feasible suffix widths imply $\Delta Q_I, \Delta Q_O \geq 4Mr_L > 0$. Thus, both variants reduce logical traffic under these assumptions.

Special case: square projection. For $h_{\text{in}} = h_{\text{out}} = h$, the LoRA expressions reduce to

$$Q_{\text{LoRA}} = 4(Mh + hr_L + Mr_L), \qquad \text{AI}_{\text{LoRA}} = \frac{Mhr_L}{Mh + hr_L + Mr_L} < r_L. \tag{12}$$

Matching parameters exactly gives $r_I = r_O = 2r_L$. Equation 11 then simplifies to

$$\Delta Q_I = \Delta Q_O = 2Mh \quad \text{bytes}, \tag{13}$$

equivalent to Mh element accesses. Both TESSA variants then satisfy

$$\frac{\text{AI}_{\text{TESSA}}}{\text{AI}_{\text{LoRA}}} = \frac{2hr_L + M(2h + 2r_L)}{2hr_L + M(h + 2r_L)} \approx 2 \quad \text{when } M \gg r_L \text{ and } h \gg r_L. \tag{14}$$

A.2 GPU EXECUTION OVERHEAD OF LORA: SQUARE-PROJECTION CASE STUDY

To examine the inefficiency of LoRA in detail, we apply a time-based Roofline model (Wang et al., 2020) to LoRA on a 4096×4096 projection, varying the rank r and token batch size M on an NVIDIA H100 GPU. Here, $h_{\text{in}} = h_{\text{out}} = h = 4096$ and $r = r_L$. The measured adapter path consists of two sequential GEMMs with a materialized intermediate. Following empirical Roofline practice, we use microbenchmark-calibrated compute throughput F and memory bandwidth BW for the measured execution environment (Wang et al., 2020).

Figure 7 shows a transition from latency-bound to memory-bound execution across the measured configurations. At small M, the narrow, sequential GEMMs provide insufficient parallel work to fully utilize the GPU or amortize fixed launch overhead; the adapter takes 62–70% of the base projection's execution time and latency remains nearly constant in this regime. At large M, reading the input activations during shrink and writing the output updates during expand dominate memory traffic, leaving the adapter memory-bound despite its small parameter count. Although increasing the rank improves arithmetic intensity, practical LoRA ranks are generally too small to reach the compute-bound threshold: the arithmetic intensity of LoRA is bounded above by r_L.

Calculation of regime boundaries. We distinguish latency-, memory-, and compute-bound regions according to the largest of t_0, t_{memory}, and t_{compute}, where t_0 is the calibrated fixed execution overhead.

For a square projection of width h,

$$t_{\text{compute}} = \frac{4Mhr}{F}, \qquad t_{\text{memory}} = \frac{4(Mh + hr + Mr)}{BW}. \tag{15}$$

Here, F and BW are empirically calibrated compute throughput and HBM bandwidth (Wang et al., 2020); memory time uses logical GEMM traffic.

The latency–memory boundary follows from $t_0 = t_{\text{memory}}$:

$$M_{\text{lat}\to\text{mem}}(r) = \frac{t_0 BW/4 - hr}{h + r}. \tag{16}$$

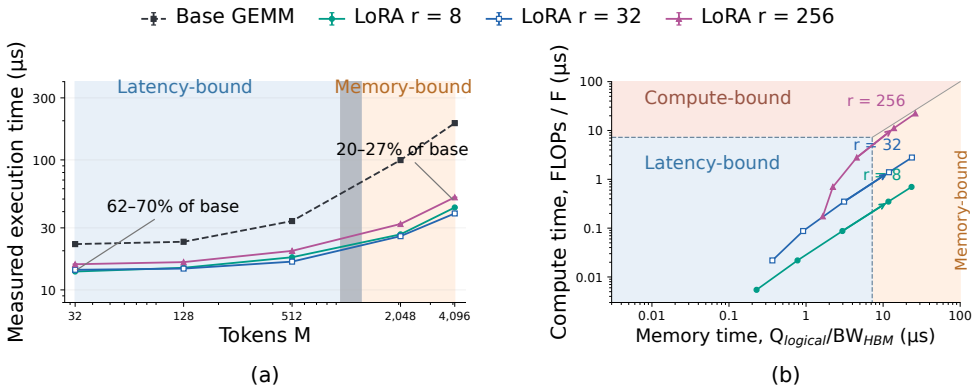


Figure 7: LoRA execution time and time-based Roofline analysis on an H100 GPU (BF16, 4096×4096 projection). (a) Measured latency of the base GEMM and the LoRA shrink–expand pair across token batch sizes M and ranks r. Shading marks latency- and memory-bound regimes; the gray band indicates rank-dependent transitions. (b) Time-based Roofline view with arrows indicating increasing M. The analysis uses empirically calibrated compute throughput and HBM bandwidth.

For $h = 4096$, $t_0 = 7.2192$ μs, and $BW = 2.880442$ TB/s, the boundary occurs at approximately 1259, 1228, and 954 tokens for $r = 8$, 32, and 256, respectively. Below this boundary, the fixed-execution reference dominates; above it, memory time dominates, provided compute time remains smaller.

The memory–compute boundary follows from $t_{\text{memory}} = t_{\text{compute}}$. This condition can be expressed in terms of arithmetic intensity:

$$\text{AI}_{\text{LoRA}} = \frac{Mhr_L}{Mh + hr_L + Mr_L} = \frac{F}{BW}. \tag{17}$$

Solving for M gives

$$M_{\text{memory}\to\text{compute}} = \frac{Fhr_L}{BWhr_L - F(h + r_L)}, \tag{18}$$

provided that $BWhr_L > F(h + r_L)$. Otherwise, no finite token batch size reaches the compute-bound regime under this model.

For any finite $M > 0$,

$$\text{AI}_{\text{LoRA}} < \frac{hr_L}{h + r_L} < r_L, \qquad \lim_{M\to\infty} \text{AI}_{\text{LoRA}} = \frac{hr_L}{h + r_L}. \tag{19}$$

This limit approaches r_L when $h \gg r_L$. With our calibrated rates, $F/BW \approx 267$ FLOP/byte on NVIDIA H100 GPU. Since all evaluated ranks satisfy $r_L \leq 256$, none reaches the compute-bound threshold, regardless of token batch size.

Empirical compute throughput and memory bandwidth. Following the empirical Roofline calibration approach of Lo et al. (2015), we estimated attainable compute throughput and memory bandwidth using microbenchmarks rather than nominal hardware peaks. Specifically, we used BF16 cuBLAS GEMMs with FP32 accumulation to calibrate F and a streaming read/write copy to calibrate BW. Measurements were performed on an H100 80GB HBM3 GPU with its SM clock fixed at 1590 MHz, using CUDA 13.0. We benchmarked square GEMMs with dimensions 2048, 4096, and 8192. For each GEMM size, we ran five independent processes, each with 20 warm-up calls and 15 timing samples of 20 GEMMs. Then, we took the median throughput within each process and averaged these medians across the five processes. We used the highest resulting average throughput, $F = 769.1137$ TFLOP/s, achieved at dimension 4096.

Empirical peak HBM bandwidth was measured using a streaming copy that sequentially read an input buffer and wrote to a separate output buffer. A sweep of buffer sizes and block counts selected 1024-MiB input and output buffers and 2112 blocks of 256 threads each, based on the highest mean

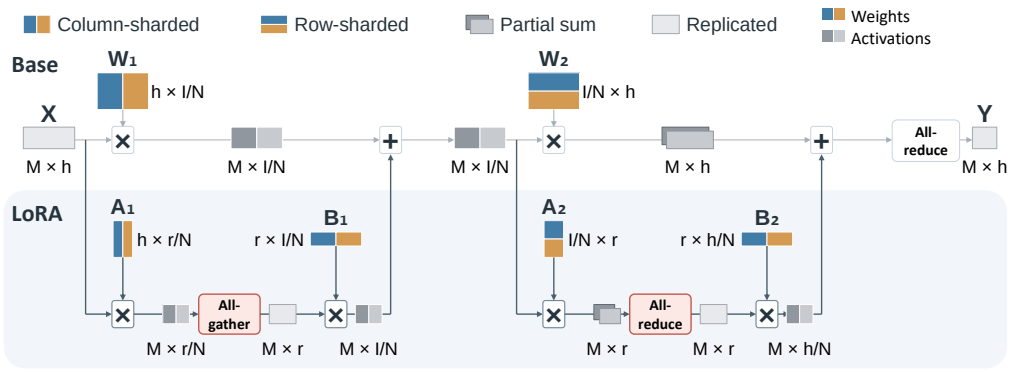


Figure 8: S-LoRA TP strategy for one simplified MLP module. The base shards the up projection along output axis I, where I denotes the intermediate dimension, and the down projection along input axis I, with one all-reduce at the block output. LoRA additionally all-gathers the shrink output of the up projection and all-reduces the shrink output of the down projection before expansion.

of the five process-level median bandwidths. The buffers exceeded the 50-MiB L2 cache capacity to limit cache reuse, and the block count corresponded to 16 times the 132 SMs on the tested GPU to provide sufficient parallel work across the device. We used five independent processes, each performing 30 warm-up executions followed by 15 timing samples, with each sample averaging 20 CUDA Graph replays. Bandwidth was computed as total accessed bytes divided by CUDA-event elapsed time, yielding $BW = 2.880442$ TB/s.

Fixed execution overhead measurement. Following NVIDIA's methodology for measuring repeat launch device runtime of straight-line CUDA Graphs (Hoffman & Oh, 2024), we used the `cudaGraphsPerfScaling` sample to measure the latency of a serial chain of three empty kernels. We chose three kernels because a small-shape cuBLAS shrink–expand pair launched three internal GPU kernels despite using two GEMM calls. We collected 600 samples in each of five independent processes and averaged the five process-level median latencies, obtaining $t_0 = 7.2192\ \mu$s.

A.3 Tensor-parallel overhead and collective costs

Measurement setup. In Figures 2 and 9, we use a BF16 Qwen3-32B model with a rank-32 adapter on all attention and MLP projections.

For panel (a) in each figure, we measured end-to-end throughput using vLLM, which employs Punica kernels and the S-LoRA TP strategy. A single client sent 64 measured requests without inter-arrival delays, subject to concurrency limits of 32 for decode-heavy and 12 for prefill-heavy workloads; all requests used the same adapter ID.

For panel (b) in each figure, we measured the combined latency of the QKV, output, and MLP gate/up/down projections within a single transformer layer, using the token batch size specified in each figure. Communication overhead was estimated from the latency difference between runs with and without the corresponding collectives.

LoRA overhead under prefill-heavy workload. Figure 9 shows that LoRA incurs substantial overhead under a prefill-heavy workload, with throughput loss relative to the base-only model increasing from 17.6% at TP1 to 38.2% at TP8. The layer-level breakdown shows that adapter computation accounts for most of this overhead. Larger token batches provide sufficient adapter work for TP sharding to reduce execution time more effectively than in decode, while making fixed collective-launch overhead less significant relative to computation. Consequently, the relative throughput loss grows more modestly with TP degree than in the decode-heavy case (Figure 2).

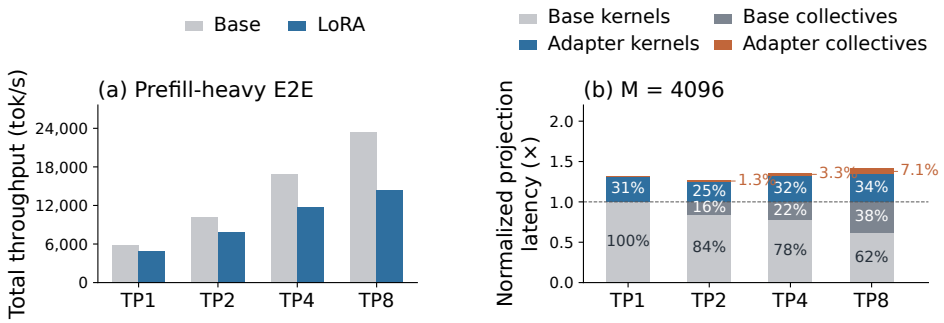


Figure 9: Tensor-parallel LoRA overhead on Qwen3-32B under a prefill-heavy workload (H100, bf16, rank 32, one active adapter; input/output lengths 2048/64, concurrency 12). (a) Absolute total throughput of the base-only and fully sharded LoRA servers across TP 1/2/4/8. (b) Latency decomposition of the QKV, output, and MLP projections within one transformer layer at $M = 4096$, normalized to the corresponding base-only projection latency ($1\times$). Segment labels give percentages of that baseline.

B Unified GEMM: computation cost and implementation

B.1 Redundant computation in multi-adapter batches

For M tokens and A active adapters with a common suffix width r, let $d = h_{\text{out}}$ for input-axis suffixes and $d = h_{\text{in}}$ for output-axis suffixes. Unified GEMM in Algorithm 1 computes the base projection and an adapter block for every token–adapter pair, requiring

$$F_{\text{unified}} = 2Mh_{\text{in}}h_{\text{out}} + 2MArd.$$

Only the assigned adapter update is needed for each token, so the useful computation is

$$F_{\text{useful}} = 2Mh_{\text{in}}h_{\text{out}} + 2Mrd.$$

The redundant computation is therefore

$$F_{\text{redundant}} = F_{\text{unified}} - F_{\text{useful}} = 2M(A-1)rd.$$

Relative to the base-projection FLOPs, this gives $(A-1)r/h_{\text{in}}$ for input-axis suffixes and $(A-1)r/h_{\text{out}}$ for output-axis suffixes.

B.2 Workload-aware dispatch

As discussed in Section 3.2 and Appendix B.1, unified GEMM performs redundant work that grows with the suffix width and the number of active adapters, and therefore can be slower than executing adapter updates separately. Figure 10 illustrates this trade-off using cuBLAS-based implementations of both paths, excluding packing and output assembly.

For a small token batch ($M = 64$), unified GEMM is faster in almost all measured configurations, as the small adapter computation makes reducing kernel-launch overhead and exposing more parallel work beneficial.

For a larger batch ($M = 4096$), separate execution becomes increasingly favorable as the active-adapter count and suffix width grow. Unlike the small-batch case, longer computation and memory-transfer times reduce the relative benefit of fewer kernel launches, making redundant computation more costly.

These workload-dependent trade-offs motivate a calibrated dispatcher that selects between unified GEMM and separate execution. Before serving, we benchmark both execution paths for each distinct local projection shape, suffix width, resident-adapter capacity, and TP configuration. The default calibration grid uses token counts $M \in \{1, 2, 4, \ldots, 8192\}$ and active-adapter counts at powers of two up to the resident count. Calibration is performed once per deployment configuration and reused during serving. In the tested configurations, it takes only about a minute. For an unmeasured

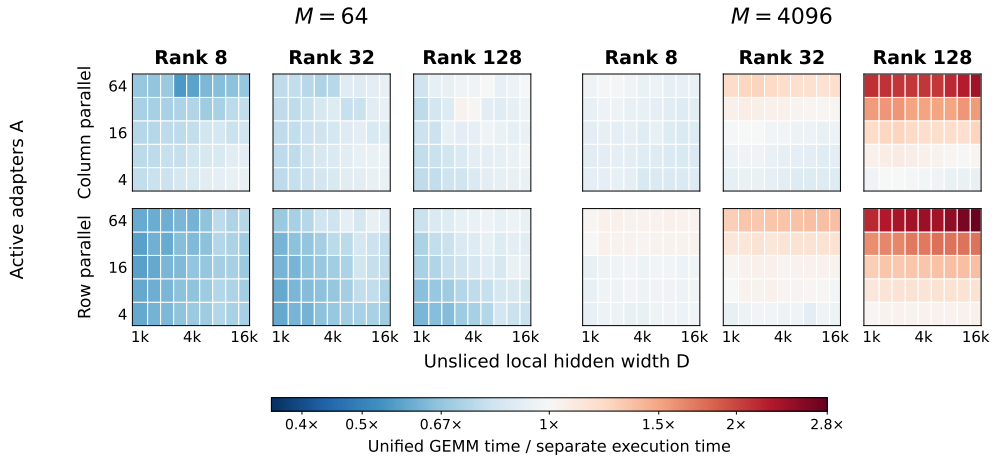


Figure 10: Measured latency ratio of unified GEMM to separate execution using cuBLAS on H100 GPUs (BF16), for $M = 64$ (left) and $M = 4096$ (right). Columns vary suffix width; rows distinguish column- and row-parallel projection shapes. The horizontal axis varies the unsliced local hidden width D, and the vertical axis varies active adapters A. The resident-adapter count equals A, and tokens are evenly grouped by adapter. The other base-weight dimension is fixed at 4096. Values above one favor separate execution. CUDA Graph measurements alternate the paths in ABBA order and cycle through weight copies exceeding four times L2 capacity. Timings include GEMMs only, excluding input packing, unified-output assembly, communication, and serving overhead.

Table 2: Dense-model training protocols. Batch denotes global batch size, and length is the maximum training sequence length.

Task	Training data	Examples	Epochs	Batch	Length
Mathematics	DDCF 1K	1,000	4	32	16,384
Code	Magicoder-OSS-Instruct-75K	38.3K	2	64	2,048
Commonsense	Commonsense170K	165,420 train + 5K val.	up to 3	64	512

configuration, the dispatcher linearly interpolates the execution time of each path between neighboring token counts and active-adapter counts. Separate execution is selected only when its predicted latency is more than 1% lower; otherwise, unified GEMM is preferred.

C Quality measurements in full

C.1 Training and evaluation setup

This section supplements the quality evaluation in Section 4.1. Tables 2 and 3 summarize the training configurations and adapter settings.

Common training settings. All dense-model experiments start from pretrained base checkpoints and use BF16 training. Within each task, LoRA and TESSA use the same data, preprocessing, and per-run training budget. All three tasks use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, cosine learning-rate decay, and warm-up over the first 3% of training steps. Weight decay is 10^{-4} for mathematics and 10^{-2} for code and commonsense reasoning. Method-specific learning rates and adapter widths are provided in Table 3.

Parameter matching. We select TESSA suffix widths to approximately match the number of trainable LoRA parameters over the task-specific target projections. Adapter widths and parameter counts are reported in Tables 3 and 5.

Table 3: Dense-model adapter widths, learning rates, and mathematics parameter counts. Entries are LoRA / TESSA; learning rates are in units of 10^{-4} and parameter counts are in millions. Code widths include attention key/value projections. CS denotes commonsense reasoning.

Model	Math width	Code width	CS width	Math LR	Code LR	CS LR
LLaMA-3.1-8B	128 / 180	128 / 196	32 / 44	3 / 3	1 / 1	3 / 1
Qwen3-8B-Base	128 / 188	128 / 206	32 / 48	3 / 2	1 / 1	3 / 3
Qwen3-14B-Base	128 / 180	128 / 200	32 / 44	3 / 3	1 / 1	1 / 1

Model	Mathematics parameters (M)
LLaMA-3.1-8B	293.6 / 294.9
Qwen3-8B-Base	302.0 / 304.9
Qwen3-14B-Base	450.9 / 449.7

Adapter initialization and scaling. LoRA initializes A with Kaiming uniform initialization ($a = \sqrt{5}$) and B to zero. TESSA initializes each trainable suffix from the corresponding base-weight slice, giving an initially zero adapter delta. Both methods scale adapter updates by α/r, with $\alpha = r$ for mathematics and code and $\alpha = 2r$ for commonsense, using the rank or suffix width of each method. The equations in Section 3 omit scalar adapter scaling for clarity; at inference, this factor can be absorbed into the stored adapter weight D_a. LoRA dropout is set to zero, and TESSA applies no adapter dropout.

Mathematical reasoning. All three models train on the same DDCF corpus.[1] Training uses completion-only supervised fine-tuning. We select the learning rate for each method using 256 GSM8K and 256 MATH validation questions drawn from their training splits; the selected rates are listed in Table 3. We select LoRA by maximizing the smaller validation-accuracy gain over the base model across the two benchmarks, and TESSA by maximizing the smaller gain over the selected LoRA configuration. Ties are resolved by the mean gain and then the smaller learning rate. The DDCF evaluation protocol uses a Qwen chat template and scores the final generated response on GSM8K and MATH-500. Results are averaged over three seeds.

Code generation. We train on the Python subset of Magicoder-OSS-Instruct-75K with packed 2,048-token sequences and evaluate using EvalPlus (Liu et al., 2023). Base and adapted models use the same Alpaca prompt format. Both methods use a fixed learning rate of 10^{-4} and train for two epochs. We evaluate the final two-epoch checkpoint and average results over three seeds.

Commonsense reasoning. Training uses early stopping based on multiple-choice accuracy on 5,000 held-out training examples, ranking answer candidates by summed log-likelihood (ties count as incorrect). Validation occurs twice per epoch, with a patience of two evaluations, and we select the checkpoint with the highest validation accuracy. We evaluate BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SocialIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), and OpenBookQA (Mihaylov et al., 2018). Following the generation-based protocol introduced by LLM-Adapters (Hu et al., 2023) and adopted by DoRA (Liu et al., 2024), we use its Alpaca prompt without a chat template, four-beam decoding without sampling, and at most 32 new tokens. We compare the first answer label matching each task's pattern with the reference label; responses without a match count as incorrect. The commonsense average in Table 1 is the mean of the eight task accuracies reported in Table 4. Additional GLUE and MMLU protocols are provided in Appendix C.3; confidence intervals are described in Appendix C.2.

C.2 Confidence intervals for downstream task performance

Table 5 reports the means and 95% confidence intervals corresponding to Table 1. Intervals use the Student-t distribution with $n-1$ degrees of freedom: $t_{0.975,n-1}s/\sqrt{n}$. Math, code, and commonsense use $n = 3$.

[1] https://huggingface.co/datasets/iNLP-Lab/DDCF-Qwen3-8B-Base-1k-0.2-math

Table 4: Per-task commonsense reasoning results. Param gives trainable adapter parameters in millions. Fine-tuned scores are means ± 95% confidence-interval half-widths over three seeds, in percent. Avg. is the mean across the eight tasks. HellaS. and WinoG. denote HellaSwag and WinoGrande. Bold indicates the higher LoRA/TESSA mean; ∗ Base denotes no fine-tuning.

Commonsense reasoning											
Model	Method	Param (M)	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
LLaMA-3.1-8B	Base*	0	24.22	54.41	17.76	24.96	2.13	12.63	15.02	13.40	20.57
	LoRA	73.4	73.90 ±1.47	85.46 ±1.74	79.58 ±1.04	93.82 ±0.42	85.48 ±3.15	89.10 ±1.09	76.39 ±2.56	84.53 ±2.56	83.53 ±0.67
	TESSA	72.1	**75.46** ±1.17	**89.10** ±0.92	**81.51** ±0.27	**96.27** ±0.32	**87.74** ±2.56	**92.92** ±0.97	**83.85** ±1.09	**89.20** ±1.32	**87.00** ±0.22
Qwen3-8B-Base	Base*	0	24.68	3.32	6.86	8.32	7.50	2.44	2.39	2.00	7.19
	LoRA	75.5	75.14 ±1.84	**90.28** ±0.94	**81.65** ±1.34	95.59 ±0.40	88.29 ±1.12	96.48 ±0.77	89.96 ±1.39	91.40 ±1.99	88.60 ±0.17
	TESSA	77.9	**75.52** ±0.82	89.81 ±2.01	81.28 ±0.70	**95.64** ±0.15	**88.69** ±0.75	**96.84** ±0.27	**90.53** ±0.22	**92.40** ±1.32	**88.84** ±0.20
Qwen3-14B-Base	Base*	0	1.90	57.18	38.89	6.79	45.78	55.01	37.12	50.60	36.66
	LoRA	112.7	77.69 ±1.02	93.22 ±0.65	**82.92** ±1.29	**97.43** ±0.05	**92.24** ±0.79	**98.18** ±0.15	**94.31** ±0.87	95.60 ±0.87	91.45 ±0.27
	TESSA	109.9	**78.35** ±1.44	**93.54** ±0.47	82.87 ±1.29	97.32 ±0.35	92.03 ±2.26	98.02 ±0.45	94.25 ±1.17	**95.73** ±0.30	**91.51** ±0.22

Table 5: Full results for Table 1, including 95% confidence intervals. Param (M) gives trainable adapter parameters in millions for code and commonsense, respectively (Code / CS); math budgets are listed in Table 3. Scores are means ± 95% confidence-interval half-widths over the task-specific seed counts stated above; bold marks the higher LoRA/TESSA mean. CS Avg. averages eight commonsense tasks (Table 4). *Base uses no fine-tuning and has no confidence interval.

Mathematical reasoning, code generation, and commonsense reasoning							
Model	Method	Param (M) Code / CS	GSM8K	MATH-500	HumanEval	MBPP	CS Avg.
LLaMA-3.1-8B	Base*	0	14.56	12.80	34.76	58.73	20.57
	LoRA	335.5 / 73.4	79.33 ±7.95	**38.47** ±7.13	51.63 ±0.87	**67.11** ±2.01	83.53 ±0.67
	TESSA	334.0 / 72.1	**79.63** ±2.36	38.40 ±15.18	**53.25** ±3.50	67.02 ±5.27	**87.00** ±0.22
Qwen3-8B-Base	Base*	0	44.66	34.80	84.76	78.31	7.19
	LoRA	349.2 / 75.5	89.71 ±23.77	81.73 ±32.47	86.18 ±0.87	82.36 ±0.77	88.60 ±0.17
	TESSA	349.3 / 77.9	**94.44** ±3.11	**87.53** ±3.78	**86.59** ±4.00	**82.98** ±0.77	**88.84** ±0.20
Qwen3-14B-Base	Base*	0	85.22	71.60	83.54	83.86	36.66
	LoRA	513.8 / 112.7	**95.15** ±1.86	90.80 ±5.66	**89.23** ±0.87	87.92 ±1.37	91.45 ±0.27
	TESSA	516.1 / 109.9	95.10 ±0.72	**91.20** ±7.50	88.62 ±3.15	**88.10** ±2.36	**91.51** ±0.22

C.3 Additional evaluations on GLUE and MMLU

We extend the comparison between TESSA and LoRA to GLUE using LLaMA-3.1-8B at approximately matched adapter parameter budgets. We also evaluate both methods on MMLU using LLaMA2-7B (Touvron et al., 2023), following the PaCA protocol (Woo et al., 2025) and including its published results as references. Both experiments adapt all seven attention and MLP projections; Table 6 reports the results and trainable parameter counts.

Table 6: Additional quality evaluations on GLUE and MMLU. † denotes published MMLU means from Woo et al. (2025), Table 1. Unmarked numeric results are our runs. Base denotes no fine-tuning. Param gives adapter parameters in millions; our GLUE task heads add 4,096–12,288 parameters per method. Metrics are scaled by 100: Matthews for CoLA, Pearson for STS-B, accuracy otherwise. Our GLUE entries show seed mean ± sample standard deviation (SD). GLUE scores use the validation split; task-specific learning rates and selection procedures are described in the text. MNLI reports matched accuracy. Our GLUE uses ten seeds for CoLA/MRPC/RTE/STS-B and three otherwise; MMLU uses three. Published rows are reference results rather than controlled comparisons against our runs; their parameter counts and training settings are not uniformly matched.

LLaMA-3.1-8B, GLUE

Method	Param (M)	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
LoRA	335.5	68.04±0.94	91.79±0.17	83.50±3.11	95.78±0.01	92.25±0.07	89.10±0.72	96.79±0.11	91.55±0.58
TESSA	334.0	66.92±1.11	91.59±0.21	82.97±2.43	95.74±0.07	92.40±0.06	88.27±1.31	96.64±0.07	91.22±0.79

LLaMA2-7B, MMLU auxiliary training

Method	Param (M)	MMLU (5-shot)
Base†	0	45.9
LoRA†	20.0	50.6
PaCA†	11.0	50.4
PaCA†	22.0	51.2
LoRA	20.0	51.40
TESSA	11.07	51.59

GLUE. We use LoRA rank 128 and TESSA width 196, with 335.54M and 333.97M adapter parameters, respectively (a 0.47% difference). Both methods use a learning rate of 10^{-5} for MNLI, MRPC, QNLI, QQP, and SST-2. For CoLA, RTE, and STS-B, we evaluate $\{10^{-5}, 3 \times 10^{-5}, 10^{-4}\}$ for both methods; the highest seed-mean validation scores select 3×10^{-5} for CoLA and 10^{-4} for RTE and STS-B for both methods. Training uses effective batch size 64, sequence length 128, AdamW, linear learning-rate decay, and a 5% warm-up fraction. We evaluate four times per epoch and use early stopping with patience three, after at least two epochs for CoLA, MRPC, RTE, and STS-B and one epoch for the other tasks.

TESSA closely matches LoRA on MNLI, QNLI, QQP, and SST-2, with differences of at most 0.20 points. LoRA obtains higher mean scores on the four smaller tasks, with gaps ranging from 0.33 points on STS-B to 1.12 points on CoLA.

MMLU. On MMLU, TESSA uses suffix width $r = 7$ and scaling parameter $\alpha = 28$, with 11.07M trainable parameters, approximately matching the 11.0M-parameter PaCA configuration. TESSA achieves accuracy comparable to the published PaCA result (Table 6). LoRA achieves 51.40% accuracy with 20.0M parameters, compared with 51.59% for TESSA.

C.4 Downstream evaluation of MoE models

We evaluate Qwen1.5-MoE-A2.7B (Qwen Team, 2024) and Qwen3-30B-A3B (Yang et al., 2025) on six downstream tasks, adapting expert gate, up, and down projections while freezing the router. Training data sources and example counts for the two models are summarized in Table 8. The final evaluation uses GSM8K for mathematical reasoning, HumanEval for code generation, IFEval (Zhou et al., 2023) for instruction following, MedMCQA (Pal et al., 2022) for medical question answering, WMT23 (Kocmi et al., 2023) for translation, and LEDGAR (Tuggener et al., 2020; Chalkidis et al., 2022) for legal classification. Table 7 reports available public-task evaluations. Both models use the same task-specific prompt format. GSM8K uses five shots for Qwen1.5-MoE and zero shots for Qwen3-30B-A3B. For the latter, the base uses flexible answer extraction and the adapted models use strict answer matching. For LEDGAR, we predict the label with the highest conditional log-likelihood among 100 fixed label candidates and report macro-F1 over the resulting predictions. For translation, both models train on 26,550 deduplicated WMT human-reference translations for one epoch with global batch 64; WMT23 is evaluated with COMET (Rei et al., 2020) using the `wmt22-comet-da` checkpoint (Rei et al., 2022) over en–cs, en–de, and en–zh.

Table 7: Downstream task performance of MoE models. Adapted-model scores are means ± SD over three seeds. GSM8K and MedMCQA report accuracy, HumanEval pass@1, IFEval prompt-level strict accuracy, and LEDGAR macro-F1 (all in percent). WMT23 reports COMET ×100, averaged over en–cs, en–de, and en–zh.

Method	Param (M)	GSM8K	HumanEval	IFEval	MedMCQA	WMT23	LEDGAR
Qwen1.5-MoE-A2.7B							
Base	0	59.82	37.20	20.70	40.66	51.56	8.78
LoRA	238.9	64.62 ±0.24	45.33 ±1.96	29.76 ±1.51	**52.82** ±0.19	**80.59** ±0.25	83.03 ±0.28
TESSA	243.3	**65.88** ±0.26	**47.15** ±0.35	**30.19** ±2.45	52.74 ±0.09	80.42 ±0.06	**83.13** ±0.17
Qwen3-30B-A3B							
Base	0	85.37	77.44	51.20	64.14	81.90	15.63
LoRA	1660.9	87.39 ±0.23	74.59 ±1.53	53.54 ±0.53	**71.73** ±0.30	**85.04** ±0.04	78.38 ±1.59
TESSA	1642.1	**87.52** ±0.34	**76.63** ±1.86	**54.41** ±0.28	71.44 ±0.10	84.95 ±0.03	**79.69** ±1.30

Table 8: MoE fine-tuning data. Example counts are listed as Qwen1.5-MoE-A2.7B / Qwen3-30B-A3B. Counts refer to training examples after preprocessing and exclusion of validation examples.

Task	Training data	Examples
Mathematics	MetaMathQA	59,000 / 59,000
Code	CodeAlpaca-20k	18,995 / 18,995
Instruction following	Tulu-v3.1 mix preview	59,000 / 59,000
Medical QA	MedMCQA	59,000 / 59,000
Translation	WMT human-reference translations	26,550 / 26,550
Legal classification	LEDGAR	58,982 / 58,982

C.5 The effect of suffix alignment on task adaptation

We examine whether fixed suffix alignment, described in Section 3.1, helps adapters learn task-relevant updates during training. As illustrated in Figure 11(a), TESSA aligns the residual coordinates updated by output-side adapters with those accessed by subsequent input-side adapters. We compare TESSA with PaCA using independently sampled input columns on Qwen3-8B-Base. Both methods train 77.86M parameters on 10,000 mathematical instruction examples from the archived `math_train.jsonl` corpus, separate from the DDCF experiment in Appendix C. Training uses one epoch, global batch size 32 (microbatch 4), maximum sequence length 384, and AdamW with zero weight decay, gradient clipping at 1, cosine decay, and 5% warmup. Both methods use a learning rate of 2×10^{-5}. We select this common rate from $\{1, 2, 3, 4, 5\} \times 10^{-5}$ using seed 42 and 128 validation examples: it is the largest rate at which TESSA has lower loss at every post-initial tuning checkpoint. The reported curves use a disjoint set of 128 validation examples and training seeds 123, 456, and 789. This selection criterion favors an early TESSA advantage, so the comparison characterizes learning dynamics under the selected setting rather than learning-rate-independent superiority. TESSA uses the final 48 residual coordinates for adapter reads and writes across layers.

We use two prompts that differ at a single token position and have different correct answers. First, we run one prompt as the reference and record the additive adapter updates from the attention output and MLP down projections at each token and layer. We then run the other prompt without changing its input. At one selected token and layer, we replace these two updates with those recorded at the corresponding projections in the reference run. Only the additive adapter updates are replaced; the frozen base projections and subsequent operations are evaluated normally using the current activations.

We then measure the change in the probability assigned to the first token of the reference answer, relative to the second prompt's run without replacement. We repeat this procedure separately for each token–layer location to obtain Figure 11(b). After four optimizer updates, TESSA shows stronger

Table 9: Task-specific training settings for MoE models. Width and learning rate are listed as LoRA / TESSA; length is the maximum training sequence length, and batch is global. Translation uses the human-reference retraining described above.

Model	Task	Width	Learning rate	Length	Epochs	Batch
Qwen1.5-MoE-A2.7B	Math	16 / 40	$3\times10^{-4}/10^{-4}$	512	1	64
	Code	16 / 40	$3\times10^{-4}/10^{-4}$	512	1	64
	Instruction	16 / 40	$3\times10^{-4}/10^{-4}$	4,096	1	64
	Medical	16 / 40	$3\times10^{-4}/10^{-4}$	512	2	64
	Translation	16 / 40	$3\times10^{-4}/10^{-4}$	512	1	64
	Legal	16 / 40	$3\times10^{-4}/10^{-4}$	512	2	64
Qwen3-30B-A3B	Math	32 / 116	$10^{-4}/3.33\times10^{-5}$	512	1	64
	Code	32 / 116	$10^{-4}/3.33\times10^{-5}$	512	1	64
	Instruction	32 / 116	$10^{-4}/3.33\times10^{-5}$	4,096	1	64
	Medical	32 / 116	$10^{-4}/3.33\times10^{-5}$	512	1	64
	Translation	32 / 116	$10^{-4}/3.33\times10^{-5}$	512	1	64
	Legal	32 / 116	$10^{-4}/3.33\times10^{-5}$	512	1	64

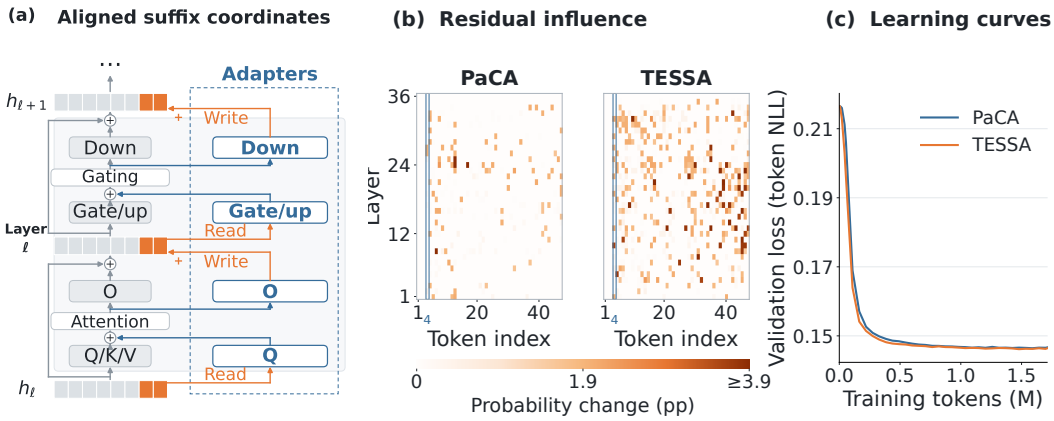


Figure 11: Suffix alignment and learning dynamics. (a) Suffix-aligned adaptation within one transformer layer. Query and gate/up adapters read the final 48 residual coordinates, while output and down adapters write to the same coordinates. Residual connections carry the updated state to subsequent blocks and layers; normalization is omitted for clarity. (b) Intervention maps for PaCA (left) and TESSA (right) on one prompt pair after four optimizer updates. Color shows the positive change in reference-answer token probability, measured over the full vocabulary in percentage points and averaged over three seeds, with a shared color scale. Token 4 marks the changed input. (c) Validation loss under matched training conditions.

positive effects across more locations than PaCA for this prompt pair, indicating that transferred updates of TESSA more strongly shift the prediction toward the reference answer.

Figure 11(c) tracks corresponding validation loss throughout the same training runs, without interventions. TESSA reduces loss more rapidly during early training, while the two methods approach similar loss values later in training. Together, the intervention results and learning curves are consistent with fixed suffix alignment helping task-relevant residual updates emerge earlier.

D Serving Measurements in Full

D.1 Detailed Serving Setup

All end-to-end serving experiments use H100 80GB GPUs. We use the native fully-sharded LoRA implementation of vLLM (Kwon et al., 2023) as the S-LoRA baseline, which uses Punica kernels

Table 10: S-LoRA and TESSA rank pairs in Figure 5, with 32 resident adapters and a request adapter pool of 32. r is the LoRA rank, R_T is the TESSA width, and ΔP_T is the signed parameter-count difference relative to S-LoRA. D and P are TESSA/S-LoRA throughput ratios for decode and prefill. BD-LoRA uses the separate rank-cap sweep described in the text. Each configuration is measured once.

Model	r	R_T	ΔP_T	TESSA / S-LoRA D	P
LLaMA-3.1-8B	8	12	-2.50%	2.949	1.601
	16	24	-2.50%	2.405	1.375
	24	36	-2.50%	2.579	1.331
	32	48	-2.50%	2.483	1.272
	64	100	$+1.56\%$	2.935	1.215
	128	196	-0.47%	3.459	1.227
Qwen3-32B	8	12	$+8.98\%$	2.842	1.521
	16	22	-0.10%	2.508	1.336
	24	32	-3.12%	2.473	1.318
	32	44	-0.10%	2.374	1.267
	64	88	-0.10%	2.474	1.198
	128	176	-0.10%	2.774	1.217
LLaMA-3.3-70B	8	12	-3.16%	2.096	1.337
	16	24	-3.16%	2.306	1.309
	24	36	-3.16%	2.292	1.271
	32	48	-3.16%	2.251	1.252
	64	100	$+0.87\%$	2.301	1.204
	128	200	$+0.87\%$	2.393	1.173
Llama-4-Scout	8	12	-9.35%	1.214	1.216
	16	28	$+5.76\%$	1.254	1.210
	32	52	-1.80%	1.383	1.242
	64	106	$+0.09\%$	1.432	1.253

Table 11: Main and additional serving workloads. Adapter pool denotes request-assignment IDs, not the number necessarily active in every forward.

Workload	Models and parallelism	Requests and adapters
Fixed-length	LLaMA-3.1-8B TP1; Qwen3-32B TP4; LLaMA-3.3-70B TP8; Scout, Qwen3-30B-A3B, Qwen3-Next, and GLM-4.5-Air (GLM-4.5 Team, 2025) TP8/EP8	Input/output 128/512 (decode) or 8192/128 (prefill); concurrency 32; 32 resident / 32 pool
BurstGPT	Qwen3-32B TP4; LLaMA-3.3-70B TP8	Open-loop, 10–90 req/s; 150/200 resident, respectively; Zipf ($s = 1.2$)

and an S-LoRA-style tensor-parallel strategy. For BD-LoRA, we ported the public vLLM implementation released by the BD-LoRA authors to our vLLM fork with minimal modifications, leaving most of its core implementation files unchanged (Wang et al., 2025a).[2] Table 11 summarizes the model, parallelism, and request settings for each workload. Projection-level measurements use the separate protocols in Appendix A.3.

Adapter budgets. For each LoRA rank in the fixed-length sweeps, we select a TESSA suffix width to approximately match the trainable adapter parameter count over the actual adapted tensors. Table 10 reports these paired ranks, signed parameter-count differences, and measured throughput ratios. Most LoRA/TESSA pairs differ by about 3% or less in trainable parameters; the larger residuals at low ranks arise from rounding suffix widths, typically to multiples of four. For BD-LoRA, the vLLM implementation zero-pads adapter factors to buffers sized by the configured `max-lora-rank` cap and executes the kernels over these padded dimensions. Ranks between supported caps therefore incur padding to a larger cap. To avoid this additional padding overhead, we independently sweep BD-LoRA ranks equal to their configured caps: 16, 32, 64, 128, 256, and 320. Figure 5 plots these points at their actual parameter counts.

[2] See `https://github.com/vllm-project/vllm/pull/28136`.

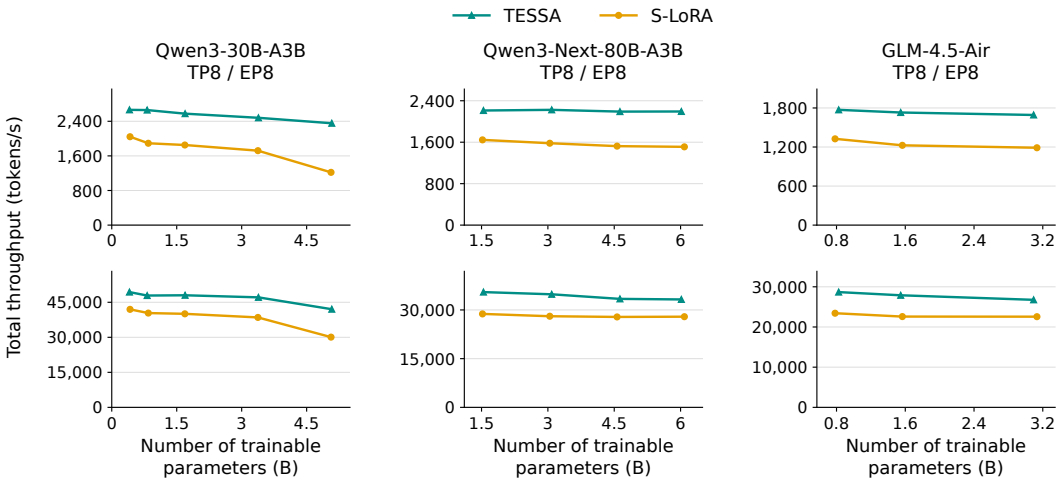


Figure 12: Serving throughput under fixed-length workloads on the additional MoE models with 32 resident adapters and a request pool of 32 adapters. Columns show models; rows show decode-heavy (top) and prefill-heavy (bottom) total throughput. Higher is better. The horizontal axis shows each method's actual trainable parameters per adapter in billions (B), under approximately matched parameter budgets. Qwen3-30B-A3B, Qwen3-Next, and GLM-4.5-Air include five, four, and three parameter budgets, respectively. The GLM-4.5-Air sweep stops at LoRA rank 32 because rank 64 required fewer resident adapters.

Fixed-length workloads. For the parameter-budget sweeps in Figures 5 and 12, we use two fixed-length workloads: prefill-heavy requests with 8,192 input and 128 output tokens, and decode-heavy requests with 128 input and 512 output tokens. Each configuration uses a concurrency limit of 32 and a request pool of 32 resident adapters; after 32 warm-up requests, we measure total throughput over 128 requests. S-LoRA enables fully sharded adapters in non-expert layers; expert adapters remain unsharded within experts distributed by EP. The MoE sweeps use LoRA ranks 8/16/32/64 for Scout, 8/16/32/64/96 for Qwen3-30B-A3B, 8/16/24/32 for Qwen3-Next (Qwen Team, 2025), and 8/16/32 for GLM-4.5-Air.

Production-trace replay. For Figure 6, we replay 9,769 requests from the busiest 1,200-second BurstGPT window (Wang et al., 2025c). We scale inter-arrival times to offered loads of 10, 30, 50, 70, and 90 req/s while preserving request order and input/output lengths. Each request is assigned an adapter from a truncated Zipf distribution ($s = 1.2$, seed 0), with 150 resident adapters for Qwen3-32B and 200 for LLaMA-3.3-70B. Per-request assignments are reused across methods and loads for each model.

All methods adapt the attention query/key/value/output and MLP gate/up/down projections. Fully-sharded S-LoRA uses rank 64; TESSA uses suffix widths 88 and 100 for Qwen3-32B and LLaMA-3.3-70B, respectively. Their per-adapter parameter counts are 536.87M/536.35M and 828.38M/835.58M (S-LoRA/TESSA), differing by less than 1%. Approximately matched BD-LoRA ranks 140 and 144 require a rank cap of 256 and run out of memory during adapter-buffer allocation. We therefore use BD-LoRA rank/cap 128, with 488.64M and 720.90M parameters, respectively—9.0% and 13.0% fewer than S-LoRA.

D.2 Additional MoE adapter budget sweeps

To assess whether TESSA's gains extend to MoE models, we evaluate Qwen3-30B-A3B, Qwen3-Next, and GLM-4.5-Air with 32 resident adapters and a 32-adapter request pool, as in Figure 12. Figure 12 shows that TESSA achieves higher throughput than S-LoRA in all measured configurations. Geometric-mean decode/prefill gains are $1.48\times/1.23\times$ for Qwen3-30B-A3B, $1.41\times/1.22\times$ for Qwen3-Next, and $1.39\times/1.21\times$ for GLM-4.5-Air.