Evaluating LoRA Placement Strategies across Large Language Models: A Multi-Dimensional Efficiency Framework

Anonymous ACL submission

Abstract

Large language models (LLMs) excel across natural language processing tasks, but their massive parameter sizes make full fine-tuning computationally prohibitive. Low-Rank Adaptation (LoRA) provides an efficient parameter-efficient fine-tuning (PEFT) alternative. However, the optimal placement of LoRA modules across model components remains unclear. We propose a unified and reproducible evaluation framework that systematically assesses LoRA placement across accuracy, efficiency, and resource dimensions, and conduct a comprehensive comparative study across representative open-source LLMs. The framework integrates Pareto frontier and knee-point trade-off analysis to identify balanced configurations among multiple insertion strategies spanning attention, MLP, output projection, embedding, and their combinations. Our results reveal that not all placement strategies improve model performance, and that more trainable parameters do not necessarily yield better results. We further observe consistent patterns across different LLMs, suggesting generalizable placement principles. Our findings establish a systematic foundation for analyzing LoRA placement, provide practical guidelines for efficient fine-tuning under resource constraints, and offer a generalizable methodology for future parameter-efficient fine-tuning research.

Keywords: Parameter-Efficient Fine-Tuning, Low-Rank Adaptation, Large Language Models

1 Introduction

In recent years, large language models (LLMs) have developed rapidly, particularly in the field of natural language processing (NLP). Based on the Transformer architecture Vaswani et al. (2017), LLMs have evolved significantly—from early models such as BERT Devlin et al. (2019) and GPT-2 Radford et al. (2018) to later models like Llama Touvron et al. (2023) and the widely adopted GPT-4 (Achiam et al., 2023). These models have demonstrated remarkable performance across a wide range of tasks. However, their impressive general capabilities, these models are not universally effective in all domains. Tasks such as medical question answering, mathematical reasoning, and domain-specific QA often require specialized adaptation to achieve high accuracy. To address this, it is essential to enhance model performance on targeted datasets. However, massive parameters in these models make full training computationally expensive. As a cost-effective alternative, fine-tuning pre-trained models has become a common strategy for adapting LLMs to specific tasks El Boukkouri et al. (2022); Gupta (2021). Parameter-Efficient Fine-Tuning (PEFT) strategy is one kind of fine-tuning methods to reduce training costs and increase resource utility efficiency Houlsby et al. (2019); Li and Liang (2021); Lester et al. (2021). Low-Rank Adaptation (LoRA) Hu et al. (2022) serves as an efficient PEFT approach, significantly reducing training costs by freezing the pre-trained model weights and introducing trainable low-rank matrices.

Although LoRA has proven effective for fine-tuning LLMs, many research is still devoted to determining the optimal configuration for its application Hayou et al. (2024b); Liu et al. (2024a); Hayou et al. (2024a); Lu et al. (2024); Zhang et al. (2023). One of the critical aspects is LoRA placement. Microsoft researchers have investigated the impact of applying LoRA to different matrices within attention layers Hu et al. (2022). Subsequent work Fomenko et al. (2024) observed that LoRA placement can be effective when applied broadly. Some studies Ge et al. (2024); He et al. (2021) reported that LoRA placement on MLP layers performs better than on attention layers. While multiple studies suggest that placement significantly influences performance, researchers still face the challenge of selecting an optimal placement scheme.

This raises important questions: What is the optimal placement strategy across different blocks for fine-tuning LLMs? Answering this requires not only assessing performance improvements but also evaluating efficiency and resource utilization under comparable conditions. However, there still lack systematic experiments and evaluations across different placement strategies. Addressing this gap is crucial for understanding LoRA's behavior and for providing practical guidelines for its application to efficient LLM fine-tuning.

In this paper, we propose a unified and reproducible evaluation framework for systematically analyzing LoRA placement strategies in large language models (LLMs). We evaluate representative placement strategies—covering attention, MLP, output projection, embedding, and their combinations-across different representative models: Llama3 Liu et al. (2024b), Gemma3Team et al. (2025), Qwen3Yang et al. (2025), Falcon Penedo et al. (2023). Unlike prior studies that focused on isolated modules, our systematic evaluation framework jointly measures accuracy, efficiency, and resource usage to derive practical placement guidelines under different optimization goals. Our main contributions are summarized as follows:

- **Evaluation Framework.** We propose a reproducible framework that systematically compares nine LoRA placement strategies across four representative LLMs, integrating accuracy, efficiency, throughput, and memory metrics through Pareto frontier and knee-point analyses.

- **Empirical Findings.** We reveal that not all placements are beneficial: broader adaptation does not always improve performance, and consistent cross-model patterns identify the most efficient and most accurate strategies.

- **Practical Guidelines.** We derive actionable recommendations for choosing LoRA placements under varying priorities—accuracy-oriented, efficiency-critical, or resource-limited scenarios.

2 Background and Related Work

2.1 Overview of Low-Rank Representation

During fine-tuning, large language models (LLMs) often exhibit a much lower intrinsic dimensionality

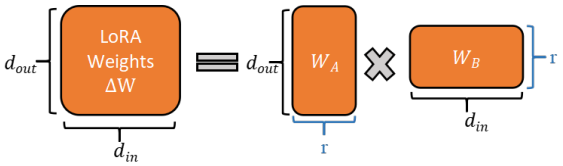

Figure 1: Low-rank decomposition of the LoRA weight update ΔW.

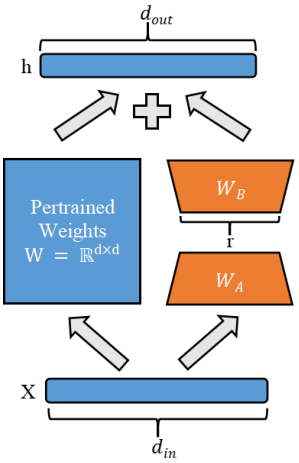

Figure 2: Workflow of LoRA integrated with pre-trained weights.

than the full parameter space. Based on this property, LoRA introduces a learnable update matrix ΔW constrained to a low-rank form

$$W' = W + \Delta W, \quad \Delta W = AB \tag{1}$$

where $A \in \mathbb{R}^{d_{out}\times r}$, $B \in \mathbb{R}^{r\times d_{in}}$, $r \ll min(d_{in}, d_{out})$ is the low-rank hyperparameter as illustrated in Figure 1. This low-rank approximation significantly reduces the number of learnable parameters when r is small. Instead of updating the full weight matrix, LoRA freezes pre-trained weights and only trains the low-rank adapters, thus effectively leveraging prior knowledge, as shown in Figure 2. Building on this foundation, we next describe the decoder-only LLM architecture, which provides the structural basis for LoRA placement.

2.2 Decoder-only LLMs

Decoder-only LLMs (Vaswani et al., 2017; Radford et al., 2018) consist of an input embedding layer, a stack of decoder blocks, and an output projection head. Each decoder block contains two main components: a multi-head self-attention layer and a feed-forward network (MLP), each containing multiple linear projection matrices. Figure 3 illustrates these core components, while omitting auxiliary structures such as normalization and positional

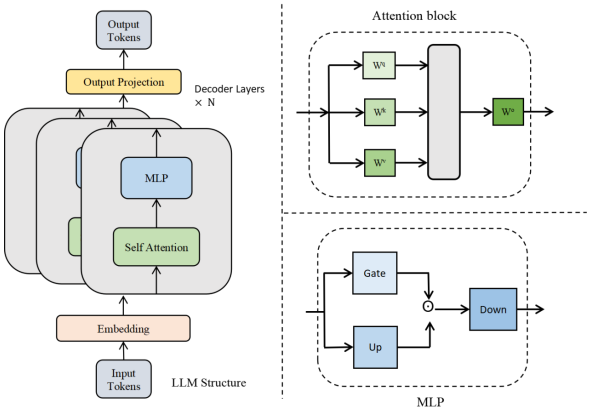

Figure 3: Structure of a decoder-only LLM, highlighting attention, MLP, embedding, and output projection matrices.

embeddings for clarity. The attention block contains four projection matrices (W_q, W_k, W_v, W_o) (Vaswani et al., 2017; Brown et al., 2020), while the MLP block includes up, gate, and down projections (Geva et al., 2020; Shazeer, 2020). In addition, the embedding layer Vaswani et al. (2017); Radford et al. (2018); Mikolov et al. (2013) and output projection head Vaswani et al. (2017); Press and Wolf (2016); Radford et al. (2019) also contain linear matrices, providing further insertion points. These projection matrices are the main targets for LoRA placement.

2.3 Related Works

The question of optimal LoRA placement remains open. Hu et al. (2022) examined varying the rank r and applying LoRA to different attention matrices, showing significant performance improvements. Beyond attention layers, (Fomenko et al., 2024) reported practical insights suggesting that embeddings, MLPs, or even all-layer placements may improve model performance. Other studies similarly observed that LoRA placement on MLP layers can outperform attention-only configurations (Ge et al., 2024; He et al., 2021). More recent work Hayou et al. (2025) proposed a method for identifying promising placement matrices within attention and MLP layers. In addition to placement, other study investigated complementary factors such as precision formats (FP16, BF16) and hardware architectures (e.g., A100, H100) that influence the efficiency of LoRA fine-tuning Hanindhito et al. (2025).

Overall, existing research provides valuable insights but remains fragmented—typically limited to specific modules and models, and do not establish a consistent pattern across architectures and metrics. In contrast, our work conducts a systematic comparison across multiple LLMs and different placement strategies, integrating accuracy, efficiency, and resource usage into a unified evaluation framework.

3 Methodology

Our study is grounded in a unified evaluation framework designed to systematically analyze LoRA placement across large language models (LLMs). The framework comprises four components:

- Defining the design space of LoRA placement.
- Maintaining consistent experimental settings for fair comparison.
- Evaluating performance across multi-dimensional metrics.
- Analyzing trade-offs via Pareto frontier and knee-point methods.

Together, these components enable a controlled, reproducible, and interpretable assessment of parameter-efficient fine-tuning strategies.

3.1 Low-Rank Placement Strategies

In this section, we describe the first component of this framework—Defining the design space of LoRA placement—by enumerating all possible insertion strategies within decoder-only LLMs. Based on the structural characteristics of decoder-only LLMs, we design a series of experiments to evaluate LoRA placement strategies across different layers. Building on the LLM components described in Section 2.2, we define a set of placement configurations ranging from single-layer to multi-layers placement strategies. Figure 4 visualizes the overall design, and Table 1 provides detailed specifications for each configuration. This design space definition constitutes the first component of our unified evaluation framework. The following sections describe the controlled experimental settings and evaluation metrics used to ensure consistent and reproducible comparisons across all placement strategies.

3.2 Experimental Settings

This section corresponds to the second component of our unified evaluation framework—maintaining

Strategy	Module Name	Placement Modules
S0	Baseline (no fine-tuning)	frozen pretrained model
S1	Attention only	W^q, W^k, W^v, W^o
S2	MLP only	gate, up, down
S3	Output projection only	lm_head
S4	Embedding only	embed_tokens
S5	Attention + Projection	W^q, W^k, W^v, W^o, lm_head
S6	MLP + Output projection	gate, up, down, lm_head
S7	Attention + MLP	W^q, W^k, W^v, W^o, gate, up, down
S8	Attention + MLP + Output Projection	W^q, W^k, W^v, W^o, gate, up, down, lm_head
S9	All layers	W^q, W^k, W^v, W^o, gate, up, down, lm_head, embed_tokens

Table 1: Summary of LoRA placement strategies (S0–S9) evaluated in this study.

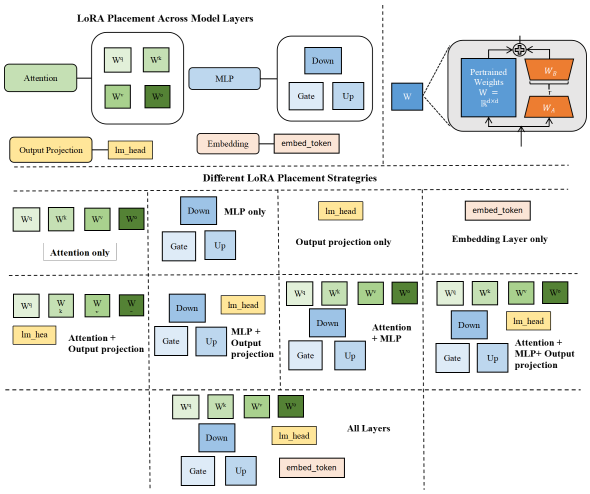

Figure 4: LoRA placement strategies in decoder-only LLMs. The top shows the internal structures of attention, MLP, embedding, and output projection layers, while the bottom summarizes the nine strategies, ranging from single-layer to full-layer placements.

consistent experimental settings to ensure fair and reproducible comparison across placement strategies. To ensure consistency across experiments, all configurations are fine-tuned under identical training settings. We adopt uniform hyperparameters across all experiments, including rank $r = 64$, scaling factor $\alpha = 128$, dropout (0.05), learning rate $lr = 2e - 4$, and precision format (BF16). The only varying factor across experiments is the LoRA placement strategy, which enables a controlled comparison of structural effectiveness. This design allows us to directly evaluate the trade-offs among different placement strategies in terms of accuracy, efficiency, and resource utilization.

We select four representative open-source LLMs from different families and design paradigms as evaluation backbones: Llama3.2-1B, Gemma3-1B, Qwen3-0.6B, and Falcon-1B. For training, we use three widely adopted multiple-choice question answering datasets, ARC-Challenge Clark et al. (2018), OpenBookQA Mihaylov et al. (2018), and CommonsenseQA Talmor et al. (2018).

LoRA-based fine-tuning is implemented with the following open-source libraries :

- Hugging Face Transformers Wolf et al. (2019): Model loading and training pipeline.
- Hugging Face Datasets Lhoest et al. (2021): Dataset downloading, processing, and preparation.
- Hugging Face PEFT Mangrulkar et al. (2022): Provides the LoRA integration API for efficiently fine-tuning pre-trained models with minimal additional parameters.

All experiments are conducted on a single NVIDIA H100 80GB GPU, which provides sufficient memory and compute capability to support large batch sizes and mixed-precision training.

For evaluation, we use the Massive Multitask Language Understanding (MMLU) benchmark Hendrycks et al. (2020), which evaluates both general-purpose reasoning and subject-specific accuracy across diverse domains. To ensure robustness, each placement strategy is evaluated over 5 independent runs under identical conditions. Reported results correspond to the mean values, reducing variance and improving reliability.

3.3 Evaluation Metrics

This section corresponds to the third component of our unified evaluation framework—defining the metrics used to assess accuracy, efficiency, and resource utilization. To evaluate model performance and efficiency, we record the following metrics:

- Accuracy: Measured on the MMLU benchmark.
- Parameters: Total number of parameters and number of trainable parameters.
- Throughput: Token processing rate (tokens per second) during training.
- Memory usage: Peak GPU memory consumption.

To quantify parameter efficiency, we define the *Improvement-per-Parameter Ratio (Im/P)* as:

$$Im/P = \frac{\Delta Accuracy(\%)}{\Delta Parameters(\%)} \quad (2)$$

This ratio normalizes performance gains by the relative increase in trainable parameters with respect to the full model size. It reflects the accuracy improvement achieved per 1% increase in trainable parameters, offering an interpretable measure of fine-tuning efficiency. Using these metrics, we evaluate and compare the accuracy, efficiency, and trade-offs among different LoRA placement strategies in the following section.

4 Results and Analysis

This section presents the experimental results and analysis under the proposed evaluation framework. This analysis corresponds to the final component of our unified framework—examining accuracy–efficiency trade-offs through multi-objective comparison and Pareto-based interpretation. We systematically compare various LoRA placement strategies across four representative models: Llama3.2-1B, Gemma3-1B, Qwen3-0.6B, and Falcon-1B. All strategies are evaluated in terms of accuracy, stability, parameter efficiency, throughput, and memory usage, providing a comprehensive view of the trade-offs between performance and efficiency. Our experiment results are shown in Table 2

4.1 Accuracy Comparison Across Layer Strategies

We begin by examining the accuracy performance of different LoRA placement strategies across the representative models. Complete results are provided in Table 3 (Appendix 5), while Figure 5 visualizes taverage accuracy across all strategies.

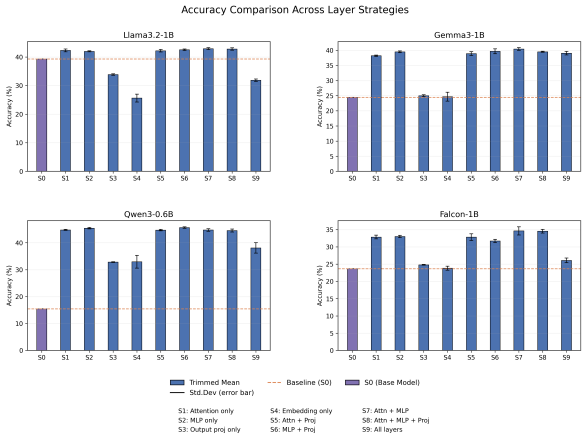

Figure 5: Accuracy results of different placement Strategies

Single-layer strategies. S1 (Attention) and S2 (MLP) consistently delivered strong improvements across all models. Notably, S2 outperforms S1 on Gemma3-1B, Qwen3-0.6B, and Falcon-1B, suggesting that MLP layers provide stronger adaptation capacity in certain architectures. By contrast, S3 (Output projection) and S4 (Embedding-only) showed limited or even negative gains: on Llama3.2-1B both fell below the baseline, and on other models their improvements were marginal at best. Even for Qwen3-0.6B, where S3 and S4 provided noticeable boosts, their absolute performance remained far behind S1 and S2.

Multi-layer strategies. Among combined placements, S7 (Attention+MLP) achieved the highest or near-highest accuracy across most models, confirming that jointly adapting attention and MLP layers yields substantial performance gains. S6 (MLP+Output projection) and S8 (Attention+MLP+Output projection) also performed competitively, with effectiveness varying by model: S6 was particularly strong on Gemma3-1B and Qwen3-0.6B, while S8 achieved slightly better results on Llama3.2-1B and Falcon-1B. S5 (Attention+projection) offered moderate improvements—superior to projection-only strategies but generally weaker than S7. By contrast, S9 (All layers) frequently underperformed and occasionally degraded accuracy, suggesting that overly broad adaptation—particularly involving embeddings—can introduce redundancy and hinder generalization efficiency.

Strategy	Accuracy (%)	Improvement vs. Base (%)	Trainable Params (%)	Im/P	Throughput (tok/s)
LLaMA 3.2 1B					
S0 (Baseline)	39.32	–	–	–	–
S1 (Attention)	42.33	+3.01	1.09	2.76	69.11
S7 (Attn+MLP)	42.93	+3.61	3.52	1.03	55.86
S9 (All)	31.88	–7.44	4.76	–1.56	52.72
Gemma 3 1B					
S0 (Baseline)	24.47	–	–	–	–
S1 (Attention)	38.22	+13.75	1.18	11.65	48.24
S7 (Attn+MLP)	40.44	+15.97	4.96	3.22	39.13
S9 (All)	39.06	+14.59	7.91	1.84	37.13
Qwen 3 0.6B					
S0 (Baseline)	15.45	–	–	–	–
S1 (Attention)	44.80	+29.35	2.99	9.82	54.36
S7 (Attn+MLP)	44.75	+29.30	6.34	4.62	47.34
S9 (All)	38.10	+22.65	9.14	2.48	45.40
Falcon 1B					
S0 (Baseline)	23.69	–	–	–	–
S1 (Attention)	32.85	+9.16	1.42	6.45	58.42
S7 (Attn+MLP)	34.64	+10.95	3.70	2.96	47.09
S9 (All)	26.10	+2.41	4.17	0.58	46.23

Table 2: Key results of LoRA placement strategies across four models, showing baseline (S0), Attention-only (S1), Attention+MLP (S7), and all-layer (S9). Full results for all strategies (S0–S9) are reported in Appendix Table 3.

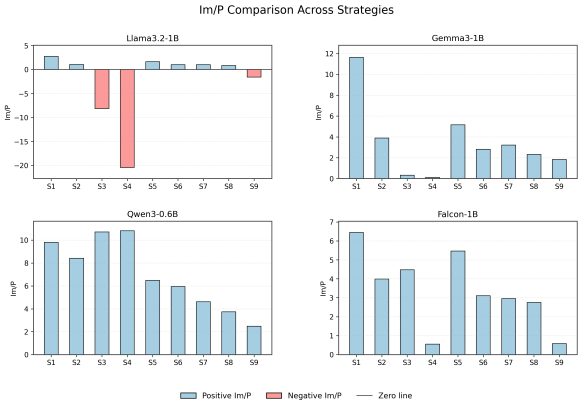

Figure 6: Comparison of parameter efficiency (Im/P) across different strategies.

4.2 Efficiency Comparison

This section focuses on the efficiency dimension of our unified evaluation framework. Efficiency measured in terms of parameter efficiency (*Im/P*) and throughput (tokens per second). These two metrics capture how effectively LoRA placements exploit trainable parameters and how efficiently the adapted models process tokens. We further combine them into a Pareto analysis to identify strategies that achieve balanced trade-offs between computational efficiency and performance..

4.2.1 Im/P

Parameter efficiency results are shown in Figure 6. Across models, S1 achieved the highest parameter efficiency, representing the best balance between accuracy gains and the proportion of trainable parameters. In contrast, S3, S4, and S9 consistently ranked lowest, with S3 and S4 even yielding negative efficiency on Llama-3.2-1B. The remaining strategies (S2, S5–S8) generally exhibit moderate efficiency, though none surpass S1. Interestingly, on Qwen3-0.6B, S3 and S4 obtained the highest Im/P values, suggesting that in smaller-scale models, even limited accuracy improvements combined with very few trainable parameters can yield higher normalized efficiency.

4.2.2 Throughput

Throughput comparison as shown in Figure 7, reveal a distinct trend from Im/P. The results indicate a general relationship between throughput and the number of trainable parameters: strategies with fewer trainable parameters typically achieved higher throughput, enabling faster token processing. In particular, S3 consistently delivered the highest throughput across all models. Interestingly, although S4 involves a comparable number of parameters, its throughput was notably lower than S3, suggesting that the computational cost is influenced not only by parameter quantity but also by the specific module where LoRA is applied.

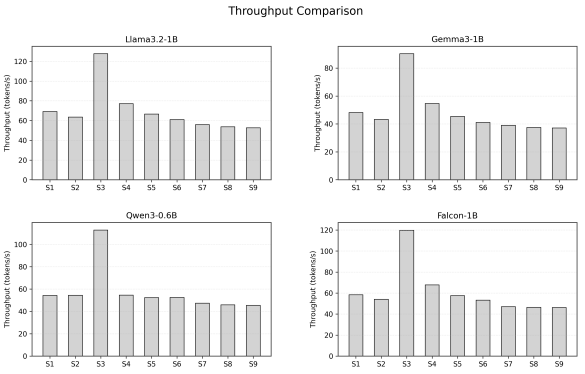

Figure 7: Throughput comparison across different models

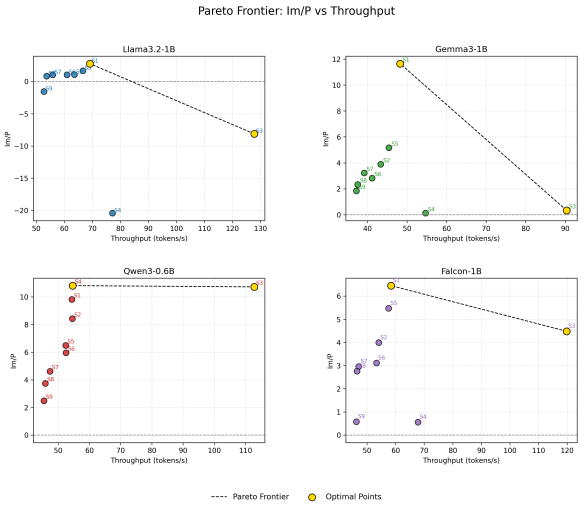

Figure 8: Pareto analysis of efficiency trade-offs (Im/P vs. Throughput).

4.2.3 Im/P vs Throughput

The combined efficiency analysis is illustrated in Figure 8. By combining both dimensions, we identify Pareto-optimal strategies that balance parameter efficiency (Im/P) and token processing speed. For Llama3.2-1B, Gemma3-1B, and Falcon-1B, the Pareto frontiers were dominated by S1 and S3. However, since S3 suffered from poor or even negative accuracy efficiency, S1 emerges as the preferred choice in practice. For Qwen3-0.6B, the frontier was primarily shaped by S3 and S4, followed by S1. Nevertheless, because S3 and S4 offer limited absolute accuracy improvements, S1 again provides the most balanced trade-off between efficiency and performance.

4.3 Tradeoff Analysis

This section corresponds to the final analytical component of our unified evaluation framework—examining multi-dimensional trade-offs between performance and resource usage. While accuracy and efficiency provide different insights, a complete evaluation also requires considering resource utilization, particularly GPU memory consumption. We analyze the trade-offs between performance gains and computational cost by analyzing the relationships between accuracy–memory and throughput–memory to highlight practical deployment strategies.

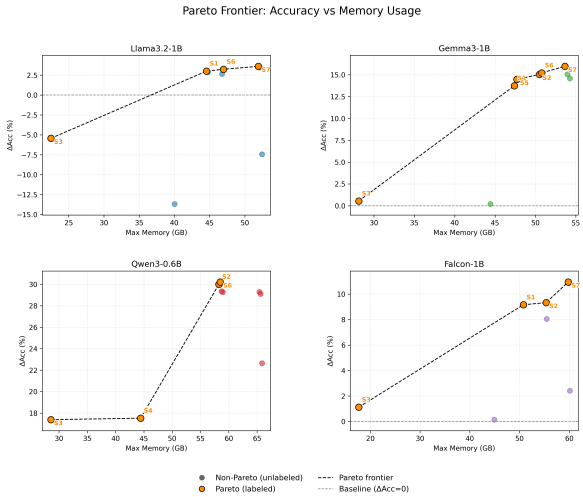

Figure 9: Accuracy–memory trade-off across LoRA placement strategies.

4.3.1 Accuracy vs. Memory

The accuracy–memory trade-off highlights how different LoRA placement strategies balance performance gains against resource demands. We first apply a Pareto frontier analysis to identify non-dominated strategies for each model. As shown in Figure 9, multiple Pareto-optimal points typically emerge. To further determine the most balanced configuration, we adopt a knee-point criterion, which identifies the point beyond which additional memory provides diminishing accuracy returns. The knee point is defined as the point of maximum curvature on the Pareto frontier, following standard diminishing-return heuristics. Detailed computational steps and model-specific case studies are provided in Appendix 5. This combined approach offers a principled and reproducible way to recommend strategies that achieve a practical compromise between accuracy and memory efficiency. The results reveal similar patterns across models. For Llama3.2-1B, the Pareto-optimal points include S1, S3, S6, and S7. Since S3 yields negative improvement, it is excluded from consideration. Among the remaining strategies, our knee-point

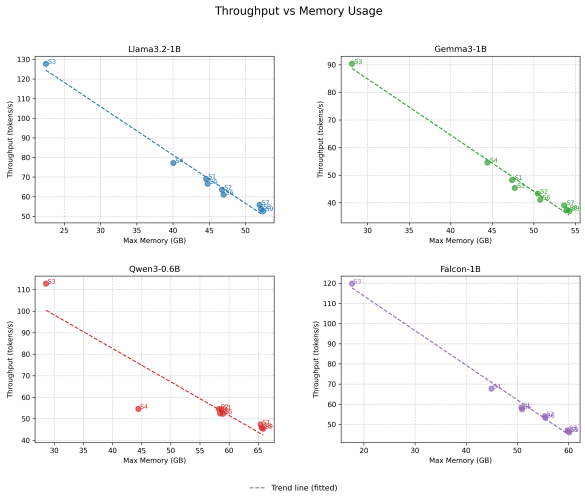

Figure 10: Throughput–memory trade-off across LoRA placement strategies.

analysis identifies S1 as the most balanced option, because the additional accuracy gains offered by S6 and S7 are marginal relative to their significantly higher memory costs. Applying the same procedure to Gemma3-1B and Falcon-1B, we also find that S1 consistently emerges as the best compromise between accuracy and memory.

In contrast, for Qwen3-0.6B, the analysis highlights S2 as the optimal trade-off point. This model exhibits a slightly different pattern: while MLP-based strategies introduce more trainable parameters, consuming slightly less memory than attention-only strategies. Nevertheless, the differences in accuracy gains and memory overhead between the two remain small. Therefore, after considering both accuracy improvements and resource consumption, S1 can still be regarded as a generally robust and practical choice across models.

4.3.2 Throughput vs. Memory

The analysis of throughput against memory usage reveals a consistent negative correlation across all models: strategies with larger memory footprints inevitably lead to lower token processing speed. This phenomenon arises because increasing the number of trainable parameters not only raises memory consumption but also introduces additional computational overhead, thereby reducing throughput. As shown in Figure 10, this trend is evident across different architectures, confirming a generally observed trade-off in efficient fine-tuning. To achieve a practical balance between speed and memory without incurring substantial accuracy loss, lightweight single-layer strategies emerge as favorable candidates. In particular, S1 and S2 consistently demonstrate strong performance across models, offering competitive accuracy while maintaining higher throughput and lower memory demand. Overall, these analyses complete the proposed framework by jointly evaluating accuracy, efficiency, and resource utilization, providing actionable insights for practical LoRA deployment.

5 Conclusion

In this paper, we presented a unified evaluation framework for systematically analyzing LoRA placement strategies and conducted a comprehensive investigation of their impact across LLMs. We applied this framework to four representative models—Llama3.2-1B, Gemma3-1B, Qwen3-0.6B, and Falcon-1B—and systematically compared accuracy, parameter efficiency, throughput, and memory usage to identify optimal configurations under varying resource conditions.

Our findings reveal several consistent trends. First, not all placements lead to performance gains. Strategies targeting only the output projection, embedding layer, or all layers generally underperformed, and in some cases even degraded accuracy—challenging the intuition that broader adaptation yields better results. Second, based on our framework's efficiency metrics, the Attention-only placement consistently achieved the highest parameter and computational efficiency across all models. Third, trade-off analysis based on Pareto frontier and knee-point evaluations show that Attention-only strategies also provide the best balance between accuracy, memory, and throughput, whereas expanding LoRA coverage increases memory usage and consequently slows down token processing.

In summary, the proposed framework enables a multi-dimensional understanding of LoRA placement, bridging accuracy, efficiency, and resource trade-offs. Attention+MLP placement is recommended when accuracy is prioritized, whereas attention-only serves as a robust and resource-efficient default. This framework provides a principled basis and practical reference for selecting LoRA placement strategies, and future work will extend it to larger-scale models, broader tasks, and adaptive placement mechanisms.

Limitations

While our framework provides a systematic analysis of LoRA placement, it has several limitations.

- First, our experiments are limited to medium-scale LLMs (up to 1B parameters) due to computational constraints. Although these models capture key trends, larger-scale validation (e.g., 7B–70B) would provide stronger generalization evidence.
- Second, our evaluation focuses on multiple-choice question answering, we expect the framework to generalize across task types
- Finally, while we analyze LoRA placements statically, exploring adaptive or data-dependent placement during fine-tuning remains an open direction.

Ethics Statement

All datasets used in this study (ARC-Challenge, OpenBookQA, and CommonsenseQA) are publicly available and commonly used in academic research. No private or sensitive data were employed, and all experiments comply with the ACL Code of Ethics. Our work focuses on efficiency and resource optimization rather than generation of user-facing content; therefore, it does not raise specific ethical or societal concerns beyond standard considerations for model reproducibility and transparency.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, and Pierre Zweigenbaum. 2022. Re-train or train from scratch? comparing pre-training strategies of bert in the medical domain. In *LREC 2022-Language Resources and Evaluation Conference*, pages 2626–2633.

Vlad Fomenko, Han Yu, Jongho Lee, Stanley Hsieh, and Weizhu Chen. 2024. A note on lora. *arXiv preprint arXiv:2404.05086*.

Xiou Ge, Ali Mousavi, Edouard Grave, Armand Joulin, Kun Qian, Benjamin Han, Mostafa Arefiyan, and Yunyao Li. 2024. Time sensitive knowledge editing through efficient finetuning. *arXiv preprint arXiv:2406.04496*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2020. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*.

Neeraj Gupta. 2021. A pre-trained vs fine-tuning methodology in transfer learning. In *Journal of Physics: Conference Series*, volume 1947, page 012028. IOP Publishing.

Bagus Hanindhito, Bhavesh Patel, and Lizy K John. 2025. Large language model fine-tuning with low-rank adaptation: A performance exploration. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering*, pages 92–104.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024a. The impact of initialization on lora finetuning dynamics. *Advances in Neural Information Processing Systems*, 37:117015–117040.

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

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2025. Plop: Precise lora placement for efficient finetuning of large models. *arXiv preprint arXiv:2506.20629*.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

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 vSavsko, 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. *ArXiv*, abs/2109.02846.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

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

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024b. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Haodong Lu, Chongyang Zhao, Jason Xue, Lina Yao, Kristen Moore, and Dong Gong. 2024. Adaptive rank, reduced forgetting: Knowledge retention in continual learning vision-language models with dynamic rank-selective lora. *arXiv preprint arXiv:2412.01004*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and 1 others. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and 1 others. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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*.

Appendix: Full Table

Appendix: Knee Point Decision Rule

This appendix details the procedure used to identify the knee point on the Pareto frontier, which determines the most balanced configuration between accuracy and memory efficiency.

Step1. Calculate the slope between consecutive points

Let the Pareto points be

$$p_1(x_1, y_1), p_2(x_2, y_2), \ldots, p_k(x_k, y_k) \quad (3)$$

with $x_1 < x_2 < \ldots < x_k$ and $y_1 < y_2 < \ldots < y_k$. The slope between consecutive points is defined as:

$$s_i = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}, \quad i = 1, 2, \ldots, k. \quad (4)$$

For our analysis, this corresponds to:

$$s_i = \frac{\Delta Accuracy}{\Delta Memory}. \quad (5)$$

Step 2: Identify the reference slope

$$s_m = \max_i s_i. \quad (6)$$

This is the highest marginal accuracy gain per unit memory.

Step 3. Classify each Segment

Compare each s_i with s_m. Segments are classified as follows:

- **High efficiency:** $s_i > 0.5 s_m$.
- **Moderate efficiency:** $0.2 s_m \leq s_i \leq 0.5 s_m$.
- **Low efficiency:** $s_i < 0.2 s_m$ or when $\Delta Accuracy < 1\%$.

Step 4. Select the Knee Point

The knee point is chosen as the last point before the first low or flattening segment, representing the best balance between accuracy improvement and memory cost. To ensure robustness against noise and spurious fluctuations, we introduce the following guard rules:

- **Noise filtering.** If a segment has $\Delta Accuracy < 1.0\%$ and consumes a large memory increase (e.g., $>$ 10 GB), it is regarded as noise rather than a valid efficiency drop. Such segments are ignored during knee detection.
- **Plateau detection.** When the first non-noise low-efficiency segment appears ($\Delta Accuracy < 1.0\%$), the model is considered to have entered a plateau region. By default, the knee point is selected as the last point immediately before this plateau.
- **Overriding the plateau.** If, after entering a plateau, subsequent points exhibit substantial cumulative accuracy improvements, the plateau phase is regarded as broken and the search for the knee point is resumed. We formalize "substantial improvements" using the following threshold:

$$\tau_T = Mean(\{\Delta Accuracy\}) \quad (7)$$

  computed across all non-noise segments.

  - **Single-segment improvement:** A segment achieves $\Delta Accuracy > \tau_T$ and slope $s_i > 0.5 s_m$.
  - **Cumulative improvement:** Across multiple consecutive segments after the plateau, the cumulative $\sum \Delta Accuracy > \tau_T$, and the average slope $\bar{s} \geq 0.2 s_m$.

These rules prevent small local oscillations ("false rebounds") from being treated as new knees, while allowing genuine second-phase improvements to be captured.

Case Studies

(1) Llama3.2-1B.
Pareto points: $S1 \rightarrow S6 \rightarrow S7$

- Max slope: $s_m = 0.0958$.
- Both segments exhibit low efficiency, indicating that the model has entered a plateau region.
- **Knee point: S1**.

(2) Gemma3-1B.
Pareto points: $S3 \rightarrow S1 \rightarrow S5 \rightarrow S2 \rightarrow S6 \rightarrow S7$

- Max slope: $s_m = 2.323$.
- The first segment is classified as Moderate, while all subsequent segments are Low.
- **Knee point: S1**.

(3) Qwen3-0.6B.
Pareto points: $S3 \rightarrow S4 \rightarrow S2 \rightarrow S6$

- Max slope: 1.048
- $S3 \rightarrow S4$ is the noise ($\Delta Accuracy < 1\%, \Delta Memory > 10GB$).
- $S2 \rightarrow S6$ is Low, indicating a plateau.
- **Knee point: S12**.

(4) Falcon-1B.
Pareto points: $S3 \rightarrow S1 \rightarrow S2 \rightarrow S7$

- Max slope: $s_m = 0.36$.
- $S3 \rightarrow S1$ is High, $S1 \rightarrow S2$ indicates a plateau, and $S2 \rightarrow S7$ is a false rebound.
- **Knee point: S1**.

(5) Summary.

- Llama3.2-1B: S1
- Gemma3-1B: S1
- Qwen3-0.6B: S2
- Falcon-1B: S1

Strategy	Trimmed mean acc.	Std. Dev	Improvement v.s. Base (%)	Trainable params (%)	Im/P	Throughput (tok/s)	Max Memory (GB)
Llama3.2-1B							
S0	39.32%	–	–	–	–	–	–
S1	42.33%	0.49	3.01%	1.09%	2.76	69.11	44.56
S2	41.98%	0.18	2.66%	2.48%	1.07	63.57	46.75
S3	33.88%	0.26	-5.44%	0.67%	-8.12	127.77	22.47
S4	25.63%	1.39	-13.69%	0.67%	-20.42	77.20	40.03
S5	42.20%	0.44	2.88%	1.75%	1.65	66.62	44.78
S6	42.56%	0.26	3.24%	3.12%	1.04	60.97	46.96
S7	42.93%	0.32	3.61%	3.52%	1.03	55.86	51.92
S8	42.80%	0.41	3.48%	4.14%	0.84	53.71	52.13
S9	31.88%	0.42	-7.44%	4.76%	-1.56	52.72	52.44
Gemma3-1B							
S0	24.47%	–	–	–	–	–	–
S1	38.22%	0.22	13.75%	1.18%	11.65	48.24	47.39
S2	39.51%	0.31	15.04%	3.87%	3.89	43.33	50.49
S3	25.02%	0.29	0.55%	1.66%	0.33	90.34	28.11
S4	24.67%	1.47	0.20%	1.66%	0.12	54.63	44.42
S5	38.94%	0.63	14.47%	2.80%	5.17	45.40	47.70
S6	39.69%	0.80	15.22%	5.40%	2.82	41.14	50.77
S7	40.44%	0.43	15.97%	4.96%	3.22	39.13	53.65
S8	39.53%	0.16	15.06%	6.46%	2.33	37.47	53.96
S9	39.06%	0.59	14.59%	7.91%	1.84	37.13	54.27
Qwen3-0.6B							
S0	15.45%	–	–	–	–	–	–
S1	44.80%	0.19	29.35%	2.99%	9.82	54.36	58.72
S2	45.45%	0.29	30.00%	3.56%	8.43	54.50	58.25
S3	32.83%	0.15	17.38%	1.62%	10.73	112.78	28.56
S4	32.97%	2.37	17.52%	1.62%	10.82	54.63	44.42
S5	44.73%	0.28	29.28%	4.51%	6.49	52.35	58.94
S6	45.67%	0.27	30.22%	5.07%	5.96	52.47	58.46
S7	44.75%	0.48	29.30%	6.34%	4.62	47.34	65.39
S8	44.56%	0.56	29.11%	7.76%	3.75	45.82	65.61
S9	38.10%	1.93	22.65%	9.14%	2.48	45.40	65.87
Falcon-1B							
S0	23.69%	–	–	–	–	–	–
S1	32.85%	0.54	9.16%	1.42%	6.45	58.42	50.77
S2	33.02%	0.36	9.33%	2.34%	3.99	54.06	55.34
S3	24.81%	0.11	1.12%	0.25%	4.48	119.80	17.66
S4	23.83%	0.59	0.14%	0.25%	0.56	67.79	44.93
S5	32.83%	0.98	9.14%	1.67%	5.47	57.56	50.90
S6	31.74%	0.47	8.05%	2.59%	3.11	53.30	55.47
S7	34.64%	1.18	10.95%	3.70%	2.96	47.09	59.80
S8	34.55%	0.54	10.86%	3.93%	2.76	46.49	59.94
S9	26.10%	0.67	2.41%	4.17%	0.58	46.23	60.13

Table 3: Full results for LoRA placement configurations (S0–S9) across models.

Segment	ΔAccuracy	ΔMemory (GB)	Slope s_i	s_i/s_m	Classification
S1→S6	+0.23%	+2.40	0.0958	1.000	Low (ΔAcc <1%)
S6→S7	+0.37%	+4.96	0.0746	0.780	Low (ΔAcc <1%)

Table 4: Efficiency classification for Llama3.2-1B.

Segment	ΔAccuracy	ΔMemory (GB)	Slope s_i	s_i/s_m	Classification
S3→S1	+13.20%	+19.28	0.685	0.295	Moderate
S1→S5	+0.72%	+0.31	2.323	1.000	Low (ΔAcc <1%)
S5→S2	+0.57%	+2.79	0.204	0.088	Low
S2→S6	+0.18%	+0.28	0.643	0.277	Low
S6→S7	+0.75%	+2.88	0.260	0.112	Low

Table 5: Efficiency classification for Gemma3-1B.

Segment	ΔAccuracy	ΔMemory (GB)	Slope s_i	s_i/s_m	Classification
S3→S4	+0.14%	+15.86	0.009	0.009	Low
S4→S2	+12.48%	+13.83	0.902	0.861	High
S2→S6	+0.22%	+0.21	1.048	1.000	Low (ΔAcc <1%)

Table 6: Efficiency classification for Qwen3-0.6B.

Segment	ΔAccuracy	ΔMemory (GB)	Slope s_i	s_i/s_m	Classification
S3→S1	+8.04%	+33.11	0.243	0.669	High
S1→S2	+0.17%	+4.57	0.037	0.102	Low
S2→S7	+1.62%	+4.46	0.363	1.000	High

Table 7: Efficiency classification for Falcon-1B.