Systematic Comparison of Parameter-Efficient Fine-Tuning Methods for Large Language Models

Maanas Aditya Popuri
Anish Sai Nimmagadda
Depak Ravinuthala

Abstract

With the growing size of LLMs, full fine-tuning of all parameters has become computationally and memory intensive. Parameter-Efficient Fine-Tuning (PEFT) techniques address this by adapting large pre-trained models to new tasks while being substantially less expensive. Methods such as Low-Rank Adaptation (LoRA), Prefix Tuning, Prompt Tuning, and their hybrids have shown strong performance under these constraints, but systematic comparisons across diverse tasks remain limited. In this report, we compare LoRA, Prompt Tuning, Prefix Tuning, and LoRA+Prompt Tuning across sentiment analysis, code generation, and mathematical reasoning tasks. By analyzing task metrics and examining the trade-offs between parameter efficiency and model performance, our results demonstrate how different PEFT methods behave across domains. Our findings show that LoRA achieves 79% accuracy on sentiment analysis with only 0.01% trainable parameters, while Prompt Tuning reaches 89.63% Pass@1 on code generation, highlighting task-dependent trade-offs in PEFT method selection.

1 Introduction

Large language models (LLMs) have grown rapidly, with architectures like GPT-4 containing over a trillion parameters and models continuing to scale, making full fine-tuning computationally expensive and storage intensive. The central challenge is how to adapt such pre-trained models efficiently to downstream tasks without updating every parameter. Parameter-efficient fine-tuning (PEFT) methods such as Low-Rank Adaptation (LoRA) [1], Prefix Tuning [2], and Prompt Tuning [3] address this by modifying only a small subset of weights while aiming to match the performance of full fine-tuning. While prior work has analyzed these methods individually, systematic cross-method comparisons across diverse task types remain limited.

This work addresses three key questions: (1) Which PEFT methods perform best across different downstream tasks? (2) How do parameter efficiency and performance trade off? (3) What hyperparameter configurations optimize LoRA's performance?

We systematically compare these PEFT techniques across three domains: sentiment analysis (SST-2 to IMDB transfer), mathematical reasoning (DeepMath-103K, MATH-500, GSM8K, AIME 2024), and code generation (CodeParrot training, HumanEval evaluation). Our methodology applies a common training and evaluation framework to LoRA, Prefix Tuning, Prompt Tuning, and a LoRA+Prompt Tuning hybrid, enabling controlled comparison of parameter counts, training efficiency, and task performance. We also conduct ablation studies on LoRA hyperparameters to identify optimal configurations.

Our results demonstrate that no single PEFT method dominates across all tasks. LoRA achieves the highest accuracy (79.8%) on sentiment classification with minimal parameters (0.01%), Prompt Tuning excels at code generation (89.63% Pass@1), and the Prompt+LoRA hybrid performs best on mathematical reasoning (68% on MATH-500). These findings provide practical guidance for selecting PEFT methods based on task characteristics and resource constraints.

2 Related Work

Parameter-efficient fine-tuning has emerged as a critical research area as LLMs continue to scale. While individual PEFT methods have been extensively studied, comprehensive cross-method comparisons remain limited. We organize existing literature into three main categories that contextualize our work.

2.1 Overview and Classification of PEFT Methods

Several surveys [4, 5] categorize PEFT approaches by their architectural principles and computational characteristics. These taxonomies distinguish between (1) *additive methods* that augment the backbone with small, task-specific components (e.g., adapter layers, soft prompts), (2) *selective methods* that update only a subset of existing parameters (e.g., bias terms, top layers), and (3) *reparameterization methods* such as LoRA that factor weight updates into low-rank matrices. While such frameworks clarify the broader PEFT design space, they typically provide limited empirical comparison among the specific families we study-low-rank, prefix-based, and prompt-based tuning-on common benchmarks and model sizes.

Our work instantiates four widely used methods-LoRA, Prefix Tuning, Prompt Tuning, and LoRA+Prompt Tuning-within a unified experimental framework, enabling controlled performance comparison across three distinct task types.

2.2 Direct Comparisons between PEFT and Full Fine-Tuning

Existing empirical studies often compare individual PEFT methods to full fine-tuning on specific benchmarks. For instance, Hu et al. [1] demonstrate that LoRA achieves performance comparable to full fine-tuning on language modeling while inducing distinct spectral properties in weight updates. Similarly, additive and prefix-style approaches have been evaluated on classification and generation tasks [2, 3]. However, these studies typically examine one technique at a time on a narrow set of domains, making it difficult to assess relative performance across diverse task types.

Our study differs by evaluating multiple PEFT strategies side by side under controlled training conditions across sentiment classification, mathematical reasoning, and code generation. We use full fine-tuning results from the literature as reference points rather than conducting exhaustive full fine-tuning baselines, prioritizing PEFT-to-PEFT comparisons under realistic resource constraints.

2.3 Hybrid and Novel PEFT Strategies

Recent research explores hybrid schemes that combine multiple PEFT principles, including compositions of adapter-style modules with prompt-based methods, structured low-rank decompositions, and multi-task frameworks. These approaches aim to further reduce trainable parameters while preserving or improving task performance, but are often evaluated on specialized benchmarks with heterogeneous experimental setups, complicating direct comparison.

Our work examines a simple hybrid approach-LoRA+Prompt Tuning-that combines low-rank weight adaptation with soft prompt tuning. We evaluate this hybrid alongside its constituent methods (LoRA alone and Prompt Tuning alone) to assess whether combining PEFT techniques yields additive benefits. Our experiments span three task types: sentiment classification (GPT-2 on SST-2/IMDB), mathematical reasoning (multiple models on DeepMath-103K and related benchmarks), and code generation (DeepSeek-R1-Distill-Qwen-1.5B on CodeParrot/HumanEval).

3 Methodology

We evaluate four such PEFT configurations: LoRA, Prefix Tuning, Prompt Tuning, and LoRA+Prompt Tuning. To ensure fair comparison, all methods use a shared supervised fine-tuning framework with identical data preprocessing, optimization settings, and evaluation protocols. This controlled setup isolates the impact of adaptation strategy on task performance and parameter efficiency.

3.1 Datasets and Task Setup

3.1.1 DeepMath-103K

DeepMath-103K [6] contains over 100,000 mathematically challenging questions spanning algebra, calcu-

lus, number theory, and geometry. Problems are decontaminated against popular benchmarks (GSM8K, MATH) and calibrated to college-level difficulty, with each problem including a verifiable final answer and step-by-step solution. We use this dataset to evaluate whether PEFT methods can learn structured mathematical reasoning, a capability that requires both symbolic manipulation and multi-step inference.

3.1.2 Sentiment Analysis

To assess classification and transfer learning capability, we fine-tune GPT-2 (124M parameters) on sentiment analysis using the SST-2 dataset [7]. Models are trained exclusively on SST-2's sentence-level movie reviews (67,349 training examples) and evaluated on the IMDB test set [8] (25,000 reviews), creating an out-of-distribution evaluation setting. This setup tests how effectively different PEFT methods transfer from compact, sentence-level supervision to longer, more varied full-length movie reviews.

3.1.3 Code Generation

To evaluate code synthesis capability, we fine-tune DeepSeek-R1-Distill-Qwen-1.5B (1.5B parameters) on the CodeParrot dataset using three PEFT configurations: LoRA, Prompt Tuning, and LoRA+Prompt Tuning. We train on 50,000 Python code examples sampled from CodeParrot, balancing computational cost with dataset coverage. Models are then evaluated on the OpenAI HumanEval benchmark [9], which contains 164 programming problems with unit tests. This setup assesses whether PEFT methods can learn code generation patterns and generalize to held-out programming challenges.

3.2 Data Formatting

All datasets are converted into a unified instruction-following format consisting of a system instruction, user query, and target assistant response. For DeepMath-103K, responses contain step-by-step solutions followed by an explicit "Final Answer:" delimiter. Sentiment examples return binary classification labels. Code examples produce complete Python function implementations. This standardization enables consistent training across models and tasks while preserving task-specific output structures. Complete template examples are provided in Appendix A.

3.3 PEFT Configuration and Training

We implement all methods using Hugging Face PEFT [10] with shared hyperparameters: AdamW optimizer, learning rate 5×10^{-4}, batch size 16, 3 epochs.

For LoRA, we set rank $r = 8$, $\alpha = 16$, targeting query and value projections. Prompt Tuning uses 8 learnable tokens. Prefix Tuning attaches length-10 prefixes to each layer. The LoRA+Prompt hybrid combines $r = 8$ LoRA with 8 soft prompts. All experiments run on NVIDIA RTX 6000 Ada (49 GB VRAM). Training typically takes 2-4 hours per configuration depending on dataset size.

4 Evaluation

4.1 Mathematical Reasoning

Benchmarks: We evaluate mathematical reasoning across four diverse benchmarks. GSM8K tests grade school-level arithmetic and chain-of-thought reasoning through word problems. MATH-500 contains competition-level problems requiring symbolic manipulation and advanced algebra. DeepMath-103K focuses on college-level mathematics including calculus and formal proofs. AIME 2024 comprises problems from the American Invitational Mathematics Examination, designed for top high school students and requiring multi-step problem-solving strategies. This range spans arithmetic fundamentals through advanced mathematical reasoning.

Metrics: We report Exact Match (EM) accuracy on the extracted final answer. A prediction is counted as correct only if the numerical or symbolic result exactly matches the ground truth, regardless of intermediate solution steps. We initially considered BLEU to score similarity between generated and reference solutions, but found that it mainly measures surface n-gram overlap rather than the logical correctness of step-by-step reasoning, so we ultimately rely on EM as our primary metric.

Domain	Training	Test	Preferred Method	Key Metric
Sentiment	SST-2	IMDB	LoRA	Accuracy: 79%
Math	DeepMath	MATH-500	Prompt+LoRA	EM: 68%
Code	CodeParrot	HumanEval	Prompt Tuning	Pass@1: 89.63%

Table 1: Domain-specific performance comparison and preferred methods

Method	DeepMath	AIME 2024	GSM8K	MATH-500
Base Model	16.00%	2.00%	24.00%	46.00%
LoRA	38.00%	6.00%	34.00%	60.00%
Prompt Tuning	26.00%	4.00%	18.00%	62.00%
QLoRA	14.00%	2.00%	20.00%	36.00%
Prompt+LoRA	46.00%	10.00%	32.00%	68.00%

Table 2: Evaluation results across mathematical reasoning benchmarks

4.2 Sentiment Analysis

Benchmark: IMDB movie reviews (25,000 test examples).

We evaluate transfer learning from SST-2 (sentence-level) to IMDB (document-level) reviews. This out-of-distribution setting tests whether PEFT methods learn robust sentiment representations that generalize across text lengths and review styles.

Metrics: We report accuracy, F1 score, and precision. We also report parameter efficiency as the percentage of trainable parameters relative to the full model. Together, these metrics characterize both classification performance and adaptation cost.

4.3 Code Generation

Benchmark: OpenAI HumanEval (164 hand-written programming problems).

HumanEval evaluates functional correctness of generated Python code. Each problem provides a function signature and docstring; models must generate a complete implementation. Correctness is verified by executing a suite of unit tests for each problem.

Metrics: We report Pass@1, the fraction of problems where the first generated solution passes all unit tests. Unlike EM accuracy, Pass@1 measures functional correctness rather than exact code matching-multiple valid implementations may solve the same problem. We also report Pass@5 (success rate when generating 5 candidates per problem) and Token Jaccard similarity (lexical overlap between generated and reference solutions). Parameter efficiency is reported to assess the performance-cost trade-off across methods.

5 Results

5.1 Domain-wise Summary

LoRA achieves the highest accuracy on sentiment classification, Prompt+LoRA excels at mathematical reasoning, and Prompt Tuning performs best on code generation, as summarized in Table 1.

5.2 Mathematical Reasoning Results

We evaluate PEFT methods across four mathematical reasoning benchmarks (Table 2). Prompt+LoRA achieves the highest accuracy on MATH-500 (68%) and AIME 2024 (10%), indicating that combining parameter-efficient methods benefits complex reasoning tasks. Table 3 shows complementary results using T5-Medium as the foundation model, where Prompt+LoRA again achieves the best performance (24% on DeepMath-103K).

Method	DeepMath Accuracy
Base Model (T5-Medium)	8.00%
LoRA	18.00%
Prompt Tuning	20.00%
QLoRA	12.00%
Prompt+LoRA	24.00%
Prefix Tuning	18.00%

Table 3: Evaluation results for DeepMath-103K using T5-Medium as foundation model

Method	Pass@1	Pass@5	Token Jaccard
Foundation	69.51%	85.23%	0.021
LoRA	28.05%	47.18%	0.089
Prompt Tuning	89.63%	96.42%	0.003
Prompt+LoRA	65.85%	82.67%	0.023

Table 4: Performance comparison of fine-tuning methods on code generation

Method	Loss	Accuracy	F1 Score	Precision	Param Efficiency (%)
Foundation	0.95–1.15	0.70	0.70	0.70	–
Prefix Tuning	0.7088	0.5000	0.6667	0.5000	11.8784
Prompt Tuning	1.2067	0.7088	0.7054	0.7138	0.6530
LoRA	0.6215	0.7980	0.7764	0.8696	0.0136

Table 5: Sentiment analysis results on IMDB after training on SST-2

5.3 Code Generation Results

On code generation (Table 4), Prompt Tuning achieves the highest Pass@1 score of 89.63%, substantially outperforming LoRA (28.05%) and even surpassing the foundation model (69.51%). This 61-percentage-point gap over LoRA indicates that soft prompt methods better preserve the base model's code generation capabilities. LoRA's weight modifications appear to interfere with pre-trained code synthesis patterns, while prompt-based approaches leave the original weights intact, maintaining the model's strong coding abilities.

5.4 Sentiment Analysis Results

For sentiment classification (Table 5), LoRA achieves the highest accuracy (79.8%) and F1 score (77.64%) while using only 0.0136% trainable parameters. This represents a 1000× parameter efficiency improvement over Prefix Tuning (11.88% parameters), which achieves only 50% accuracy. Even Prompt Tuning, despite using 50× more parameters than LoRA (0.65%), achieves lower accuracy (70.88%). These results demonstrate that LoRA is particularly effective for classification tasks, achieving superior performance with extreme parameter efficiency.

6 Ablation Study

To identify optimal LoRA hyperparameters, we conduct a systematic ablation study varying rank (r) and learning rate on mathematical reasoning. For computational efficiency, we use GPT-2 (124M parameters) on a subset of DeepMath-103K rather than the larger models from main experiments. Table 6 presents results across 15 configurations.

Rank	Learning Rate	Accuracy	BLEU	Params
4	1e-4	52.3%	8.2	0.8M
4	5e-4	58.1%	10.5	0.8M
4	1e-3	54.7%	9.1	0.8M
8	1e-4	61.2%	11.8	1.6M
8	5e-4	68.4%	13.6	1.6M
8	1e-3	62.9%	12.1	1.6M
16	1e-4	64.8%	12.4	3.2M
16	5e-4	72.5%	14.9	3.2M
16	1e-3	70.1%	13.8	3.2M
32	1e-4	66.1%	12.9	6.4M
32	5e-4	**73.8%**	**15.2**	6.4M
32	1e-3	71.4%	14.3	6.4M
64	1e-4	66.9%	13.1	12.8M
64	5e-4	74.2%	15.4	12.8M
64	1e-3	71.8%	14.5	12.8M

Table 6: Ablation study of LoRA rank and learning rate on mathematical reasoning task. Params indicates trainable parameters.

The results reveal three key trends. First, performance improves with increased rank up to $r = 32$ (73.8% accuracy), after which gains plateau—rank 64 achieves only 74.2%, a minimal 0.4 percentage point improvement despite doubling parameters from 6.4M to 12.8M. This suggests diminishing returns beyond $r = 32$ for this model size. Second, learning rate 5e-4 consistently yields the best performance across all ranks, while 1e-4 underperforms (66.1% at rank 32) and 1e-3 causes instability (71.4% at rank 32). Third, parameter count scales linearly with rank, emphasizing the efficiency-performance trade-off. The optimal configuration is rank 32 with learning rate 5e-4, balancing accuracy and parameter efficiency.

7 Limitations

Our current evaluation is limited to a subset of popular PEFT methods (LoRA, Prefix Tuning, Prompt Tuning, and hybrids), applied on classification, reasoning, and code generation tasks with relatively controlled datasets. We do not yet account for very large models (>7B parameters), specialized domains, or advanced PEFT variants such as AdapterFusion or Representation Fine-Tuning (ReFT). Further, while our results highlight general trends in performance and efficiency, the optimal method and configuration may depend heavily on specific model architectures, hardware budgets, and task complexity. Results represent single training runs due to computational constraints; confidence intervals from multiple runs would strengthen conclusions. External benchmarks and interpretability remain open challenges.

8 Conclusions and Future Work

Key Findings:

Our systematic comparison yields three key findings: (1) No single PEFT method dominates across all tasks—LoRA excels at sentiment classification, Prompt Tuning at code generation, and Prompt+LoRA at mathematical reasoning; (2) Parameter efficiency varies by 1000×, from LoRA's 0.01% to Prefix Tuning's 11.88%; (3) LoRA rank 32 with learning rate 5e-4 provides optimal accuracy-efficiency balance for mathematical reasoning on smaller models.

Future Directions:

In future work, we plan to broaden our PEFT library and make the benchmarking more systematic. This includes incorporating additional techniques and more advanced adapter-based methods. We also aim to evaluate these methods on further benchmarks, including medical question answering [11] and additional reasoning-heavy datasets, and to run detailed ablations over rank, layer placement, and training

regime to clarify when each method is most advantageous.

Ultimately, a more comprehensive comparison across tasks and architectures will help establish practical guidelines for selecting PEFT strategies that balance resource constraints with task performance, and may motivate new hybrid approaches that combine the strengths of multiple methods.

9 Acknowledgements

We would like to extend our sincere gratitude to all those who supported and guided us throughout the duration of this project. First and foremost, we express our deepest appreciation to Professor Dhruv Kumar, our Faculty In-Charge, for his invaluable insights, encouragement, and unwavering support that significantly contributed to the progress and completion of this work.

We are equally grateful to Mr. Ayush Gupta, our Industry Mentor, whose expert guidance and practical expertise greatly enriched our learning experience. Despite the geographical distance, being located in the United States while we are based in India, he generously adjusted to our time zone and consistently provided remote assistance, ensuring seamless mentorship and constructive support throughout the project.

A Implementation Details

A.1 Chat Template

For math reasoning, training examples use a JSON chat format:

```
{
"messages": [
{"role": "system",
"content": "You are a math problem
solver. Provide step-by-step
reasoning."},
{"role": "user",
"content": example["question"]},
{"role": "assistant",
"content": f"{example['solution_steps']}
Final Answer: {example['final_answer']}"}
]
}
```

A.2 LoRA Configuration

We use PEFT LoRA with rank $r = 8$, $\alpha = 16$, and dropout 0.1 on q_proj and v_proj.[1]

```
from peft import LoraConfig
lora_config = LoraConfig(
r=8,
lora_alpha=16,
lora_dropout=0.1,
target_modules=["q_proj", "v_proj"],
bias="none",
task_type="CAUSAL_LM",
)
```

A.3 Prompt Tuning

Prompt tuning uses 8 virtual tokens with random initialization.

```
from peft import PromptTuningConfig
prompt_config = PromptTuningConfig(
    num_virtual_tokens=8,
    prompt_tuning_init="RANDOM",
    task_type="CAUSAL_LM",
)
```

A.4 Prefix Tuning

Prefix tuning uses 10 virtual tokens per layer without projection.

```
from peft import PrefixTuningConfig
prefix_config = PrefixTuningConfig(
    num_virtual_tokens=10,
    prefix_projection=False,
    task_type="CAUSAL_LM",
)
```

A.5 Training Setup

All methods share the same training hyperparameters:

```
from transformers import TrainingArguments
training_args = TrainingArguments(
    learning_rate=5e-4,
    per_device_train_batch_size=16,
    num_train_epochs=3,
    warmup_steps=100,
```

[1] Hyperparameters follow the referenced PEFT LoRA setup.

```
    weight_decay=0.01,
    logging_steps=50,
    save_strategy="epoch",
    optim="adamw_torch",
)
```

A.6 Hardware

Experiments run on a single NVIDIA RTX 6000 Ada (49 GB VRAM) with PyTorch 2.0, Transformers 4.35, and PEFT 0.7.

References

[1] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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

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

[4] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pages 7319–7328, 2021.

[5] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*, 2018.

[6] Zhen Wang, Zhengyang Zhang, and Hao Xu. Deepmath-103k: A benchmark for mathematical reasoning. Hugging Face Datasets, `https://huggingface.co/datasets/zwhe99/DeepMath-103K`, 2023.

[7] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013.

[8] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 142–150, 2011.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[10] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

[11] Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. *arXiv preprint arXiv:2203.14371*, 2022.