Systematic Evaluation of LoRA Adapter Placement and Rank Allocation for Resource-Constrained Fine-Tuning

Anonymous authors
Paper under double-blind review

Abstract

Parameter-efficient fine-tuning (PEFT) methods like Low-Rank Adaptation (LoRA) enable adapting large language models (LLMs) under tight computational and data constraints. However, optimal adapter placement and rank allocation remain underexplored in ultra-low-resource regimes. Here we systematically evaluate LoRA configurations-attention-only, MLP-only, and combined placements with ranks 8 and 16-on the MBPP-mini code generation and GSM8K-mini math reasoning datasets using the Qwen2.5-0.5B-Instruct model with fixed training budgets. Our hardened evaluation protocol includes deterministic prompting, AST-based code extraction, and import-safe execution. Results show combined placement significantly improves code generation pass@1 accuracy (34% vs. 19%), while rank scaling often outperforms increased training steps for parameter efficiency. Hardware profiling reveals all configurations fit within 2.2GB GPU memory, enabling consumer-grade fine-tuning. These findings inform automated configuration selection, facilitating efficient, reliable adaptation of LLMs on minimal hardware for specialized tasks.

1 Introduction

Fine-tuning Large Language Models is computationally expensive (Hu et al., 2022). Even compact models (0.5B-3B parameters) need > 1000 labeled examples and multi-GPU hours to deliver stable results (Dettmers et al., 2023; Liu et al., 2022b; Zhou et al., 2024). The cost and availability barriers to such hardware limit researchers, small teams, and practitioners from fine-tuning their models (Dettmers et al., 2023). It motivates the need to democratize low-budget adaptation of low-resource datasets (< 200 examples) accessible on consumer-grade hardware.

While LoRA (Hu et al., 2022) enables Parameter Efficient Fine Tuning (PEFT), reducing trainable parameters, the adapter placement and rank scaling (Attention vs MLP) have stayed as underrated resources to explore in small datasets (< 200 examples) where the focus on increasing the steps dominated the rank scaling (Liu et al., 2022b; Zhou et al., 2024).

Research Questions: This work is a systematic empirical study of parameter-efficient fine-tuning under resource constraints. We address the following questions:

- **RQ1:** How does LoRA adapter placement (attention-only, MLP-only, combined) affect task performance and computational cost under fixed fine-tuning budgets?

- **RQ2:** How does rank scaling compare to increasing training steps when the total trainable-parameter budget is controlled?
- **RQ3:** Are the observed trends consistent across code generation (MBPP-mini) and mathematical reasoning (GSM8K-mini) tasks, and stable across random seeds?

Contributions: Our contributions are primarily empirical:

1. We present the first controlled empirical evaluation of LoRA adapter placement strategies (attention-only, MLP-only, combined) under ultra-low-resource settings with fixed token budgets (<200 examples).
2. We compare rank scaling versus step scaling under matched trainable-parameter budgets, demonstrating that a higher rank with fewer steps often outperforms a lower rank with more steps.
3. We introduce a hardened evaluation protocol for code generation that distinguishes functional correctness from syntactic validity, revealing that naive string-matching inflates reported performance by up to 25% relative.
4. We provide systematic hardware profiling linking each LoRA configuration to concrete resource requirements (GPU memory, training time), enabling pre-training feasibility checks.
5. We report results across multiple tasks (MBPP-mini, GSM8K-mini) and random seeds, and provide anonymized supplementary artifacts for reproducing the main results.

2 Related Work

2.1 Parameter-Efficient Fine-Tuning (PEFT)

Parameter-efficient fine-tuning (PEFT) includes methods that adapt large pretrained models by updating a small set of parameters while freezing the base weights at their original precision (Hu et al., 2022; Houlsby et al., 2019). Earlier methods for fine-tuning included prompt tuning (Lester et al., 2021), which learns prompt embeddings, and prefix-tuning (Li & Liang, 2021), which prepends trainable vectors to the model's attention layer as keys and values. Apart from prompt tuning, we also used adapter modules that added less than 1% trainable parameters (Houlsby et al., 2019; Pfeiffer et al., 2020) as lightweight layers after the attention and feed-forward sublayers. After these methods, we had LoRA, which enabled PEFT fine-tuning by injecting trainable low-rank decomposition matrices into pre-existing weight matrices to control the effective learning rate (Hu et al., 2022). After LoRA, we saw significant changes leading to QLoRA (Dettmers et al., 2023), combining LoRA with 4-bit NormalFloat quantization and paged optimizers. QLoRA enabled 65B-scale tuning on a single GPU by quantizing base weights while keeping LoRA adapters at higher precision (Dettmers et al., 2023). Other variants, such as Infused Adapter by Inhibiting and Amplifying Inner Activations (**IA**^{3}) (Liu et al., 2022a), inject three learnable vectors per layer to rescale activations without adding new layers. In contrast, Weight-Decomposed Low-Rank Adaptation (DoRA) (Liu et al., 2024), which extends LoRA, decomposes the weight into direction and magnitude. Despite progress in various methods, the placement of ranks and adapters remains underexplored (Liu et al., 2022b; Zhou et al., 2024).

2.2 Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) (Hu et al., 2022) enables parameter-efficient fine-tuning by injecting trainable low-rank matrices into pre-existing weight matrices of a frozen transformer. For a weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA models the update as $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ are initialized with random Gaussian and zero values, respectively, and $r \ll \min(d, k)$ is the rank. During the forward pass, the output is computed as

$$W_0 x + \frac{\alpha}{r}(BAx),$$

where α is a scaling hyperparameter that controls the magnitude of the update independently of r. This design ensures that increasing r adds capacity without changing the learning dynamics when α is fixed. LoRA is typically applied to attention projection matrices (W_q, W_k, W_v, W_o) although placement remains configurable (Zhou et al., 2024). QLoRA (Dettmers et al., 2023) extends LoRA by quantizing base model weights to 4-bit precision of W_0, reducing memory by 60–75% compared to LoRA while preserving adapter precision. In practice, $r \in \{4, 8, 16, 32\}$ and $\alpha \approx 2r$ are common heuristics (Hu et al., 2022). We are the first to systematically ablate r vs. training steps under fixed token budgets in ultra-low-data settings (< 200 examples), revealing that rank scaling often outperforms step increases for parameter efficiency.

2.3 Adapter Placement Strategies

Adapters in PEFT are small, trainable modules that update only specific parts of the model, leaving the majority of the model unchanged. The effectiveness of adapters depends on their placement-whether in attention layers, MLP layers, or both, which influences output quality and parameter efficiency. Strategic placement can improve performance without unnecessary memory or compute overhead.

Prior to LoRA, Houlsby-style adapters (Houlsby et al., 2019) inserted “bottleneck modules" after both the attention and feed-forward (MLP) sub-layers of the BERT-base model, updating 3.6% of the parameters by compressing the data to a lower dimension. LoRA (Hu et al., 2022) improved the efficiency by updating low-rank matrices to specific weights instead of “bottleneck layers". LoRA targets the attention “projection" matrices (W_q, W_k, W_v, W_o) to handle how the module computes attention scores and combines results (Zhou et al., 2024). Configuring attention is crucial, as most relational knowledge resides in the model’s attention, making it impactful for adaptation with minimal parameters (Diao & Loynd, 2023; Ranjan et al., 2025). However, no prior work systematically compares attention-only, MLP-only, and combined placements under ultra-low-data regimes (< 200 examples) with fixed token budgets. In this paper, we run controlled ablations on MBPP-mini and GSM8K-mini data sets, revealing that in task-dependent scenarios, for example, in code generation, it favors combined placement and consistent improvements for mathematical reasoning.

Fine-tuning a model in low-resource settings (< 200 examples) introduces challenges such as overfitting, poor generalization, and optimizer instability. With limited data, models are prone to memorizing the training set rather than learning generalizable patterns, leading to poor performance on unseen data despite high training accuracy. Regularization techniques such as weight decay, dropout, and early stopping can help mitigate these issues. However, their effectiveness diminishes when the number of trainable parameters exceeds the number of examples (Zhang et al.,

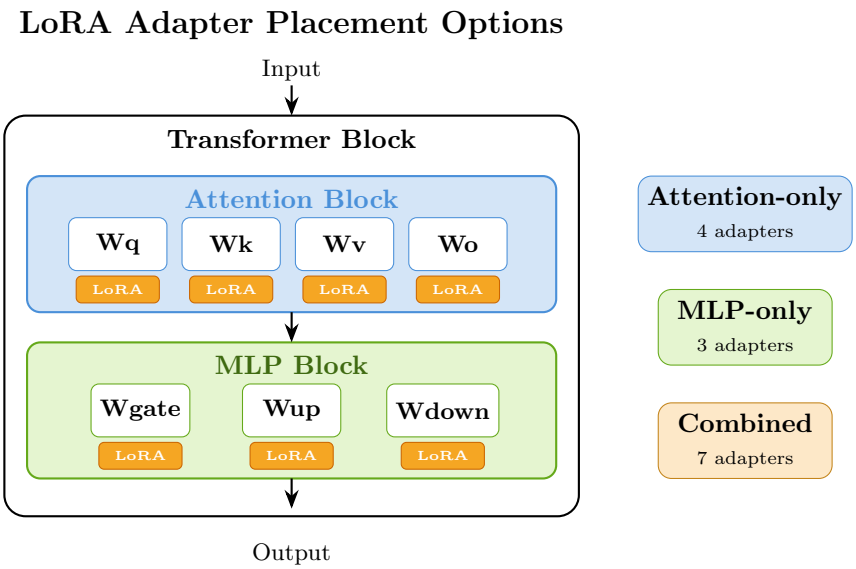

Figure 1: LoRA adapter placement options in a Transformer block. **Attention-only** targets query, key, value, and output projections (W_q, W_k, W_v, W_o). **MLP-only** targets gate, up, and down projections ($W_{gate}, W_{up}, W_{down}$). **Combined** applies adapters to all seven modules. In our experiments, combined placement yields the best performance for code generation, while MLP-only performs comparably for mathematical reasoning.

2021). Data augmentation methods, such as shuffling sample order or rewording prompts, can help build a more effective dataset, but are limited by potential mismatches in data distribution and label consistency.

PEFT approaches, such as LoRA (Hu et al., 2022) and traditional adapters (Houlsby et al., 2019; Pfeiffer et al., 2020), mitigate these issues by minimizing the number of parameters updated, thereby curbing overfitting. However, even in PEFT, deciding on the adapter location (attention vs MLP) and rank remains pivotal. A substantial portion of the few?shot parameter?efficient fine?tuning (PEFT) literature continues to report results on high?resource NLU suites, most prominently GLUE and SuperGLUE, rather than in settings with only a handful of labeled examples per task. For instance, Ahead-of-Time P-Tuning (Gavrilov, 2023) evaluates exclusively on GLUE and SuperGLUE, and P-Tuning v2 (Liu et al., 2021) likewise benchmarks on SuperGLUE, exemplifying the community's reliance on these comparatively well?resourced datasets for empirical validation. While such evaluations are informative, they do not target the "true fewshot" regime as CLUES (Mukherjee et al., 2021) argues, mainstream benchmarks like GLUE/SuperGLUE "give models access to relatively large amounts of labeled data," and there has historically been a lack of standardized few?shot evaluation, a gap that leaves very small?data ($<$ 200 examples) scenarios comparatively under?studied. Prior work explores LoRA in few-shot instruction settings but relies on synthetic data, thereby overlooking computational challenges. Therefore, prior work has explored only limited token budgets while adjusting the adapter location and rank in real-world laptop-scale tuning settings.

2.4 Code Generation Benchmarks and Evaluation

Code generation tasks evaluate a model's ability to generate correct, functional programs from natural-language descriptions. The Mostly Basic Python Problems (MBPP) benchmark (Austin et al., 2021b) consists of 974 crowd-sourced Python problems focused on loops, functions, and if-else statements, each with sample solutions and automated tests. For our study, we use a subset of MBPP with 200 training and 100 test samples, as it is well-suited for evaluating performance in ultra-low-resource settings. In contrast, the HumanEval dataset (Chen et al., 2021a) contains 164 more complex algorithmic tasks, making it less suitable for experiments with very limited data.

We evaluate code-generation performance using pass@k, the HumanEval metric introduced by Chen et al. (Chen et al., 2021b), which estimates the probability that a task is solved when up to k independent samples are attempted; consequently, in one-shot or consistency-focused settings, we report pass@1 as the primary score. Following Chen et al. (Chen et al., 2021b), our protocol is execution-based: model completions are run against unit tests to determine functional correctness, using the HumanEval harness. Because executing untrusted code can yield inconsistent outcomes, we adopt the EvalPlus safeguards of Liu et al. (Liu et al., 2023), tight timeouts, I/O suppression, and syscall/subprocess restrictions, together with expanded test suites (HumanEval+) to reduce flakiness and prevent risky behaviors. To make sure that the assessments are reliable in a low-data tuning setting, we introduce a hardened evaluation protocol:

- Consistent prompts enforcing fixed-function names (like def solution(..):)
- AST-based code extraction method to remove the unnecessary add-ons like notes, comments, or explanations

- Import-safe execution in a locked space blocking the modules like os, sys, and sub-processes.

We also track execution success rates alongside pass@1 to identify issues such as syntax errors, import errors, runtime errors, or logic failures.

2.5 Mathematical Reasoning and Cross-Domain Transfer

Mathematical reasoning tasks assess a model's ability to perform step-by-step calculations from natural-language word problems. The GSM8K dataset (Cobbe et al., 2021b) contains 8,500 grade-8 level math problems, each accompanied by detailed step-by-step solutions. For our low-resource experiments, we use a subset of GSM8K comprising 200 training and 100 test samples, aligning with our code-generation experiments and enabling consistent evaluation of fine-tuned models.

For evaluation, we use Exact Match (EM) on the final numerical answers, which requires both the correct result and proper formatting. Previous PEFT research on the GSM8K dataset has typically used over 1000 samples (Liu et al., 2022b), focusing on prompt engineering or full model fine-tuning rather than systematically exploring adapter placement or rank adjustment under limited compute resources.

2.6 Hardware-Aware PEFT Configuration

Existing PEFT research focuses primarily on task accuracy, with hardware considerations treated as secondary constraints rather than first-class design variables. Practitioners typically select LoRA configurations through trial-and-error, risking out-of-memory failures or suboptimal resource utilization (Zhang et al., 2024). While QLoRA (Dettmers et al., 2023) addresses memory efficiency through quantization, and AdaLoRA (Zhang et al., 2023) dynamically adjusts rank during training, no prior work provides upfront profiling that maps configuration choices (placement, rank) to concrete hardware requirements (GPU memory, training time) before training begins.

This gap is particularly acute in ultra-low-resource settings, where failed training runs due to memory constraints represent significant wasted effort. Our work addresses this by systematically profiling all configurations, enabling practitioners to select feasible and near-optimal settings based on their hardware constraints without exhaustive search.

3 Methodology

3.1 Experimental Setup

We conduct all experiments using **Qwen2.5-0.5B-Instruct** (Team, 2024), a 0.5 billion parameter instruction-tuned language model. We have selected this model for three primary reasons:

i) performs well on the coding and math tasks despite its compact size.

ii) it is compatible with consumer GPUs (such as the T4 with 16GB RAM using `bfloat16`), enabling laptop-scale fine-tuning.

iii) its weights are open and use flexible licensing that allows anyone to reproduce our work efficiently.

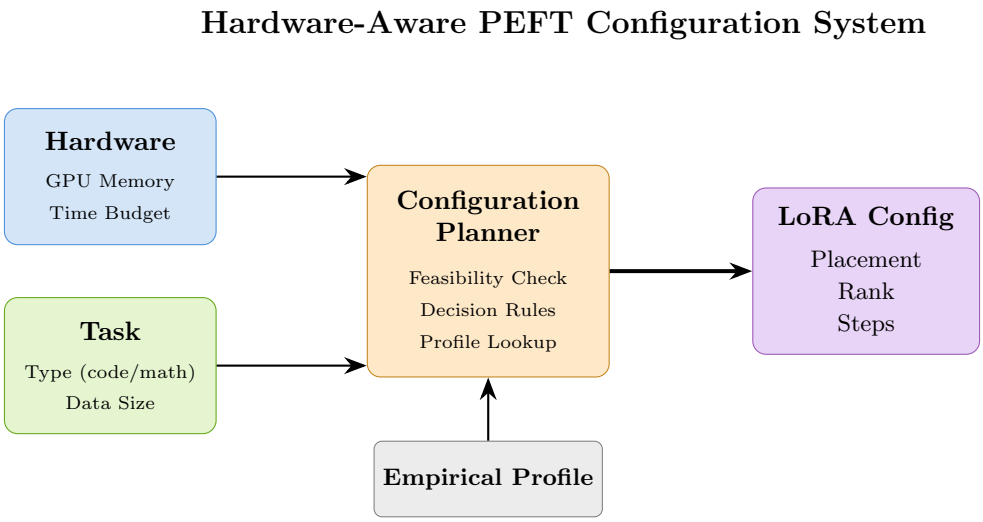

Figure 2: Hardware-aware PEFT configuration system. The planner takes hardware constraints (GPU memory, time budget) and task characteristics (type, data size) as input, consults the empirical performance profile (Table 1), and outputs an optimal LoRA configuration *before training begins*. It enables practitioners to select feasible configurations without trial-and-error.

Datasets

To compare across various areas, we test on two small benchmarks with the same 200 training and 100 test samples.

- MBPP-mini (Austin et al., 2021b;a): a 200/100 train/test subset of the Mostly Basic Python Problems benchmark covering basic control flow, loops, and if-else logic.
- GSM8K-mini (Cobbe et al., 2021b;a; Contributors, 2024): a 200/100 train/test subset of GSM8K containing grade-school math problems requiring multi-step arithmetic.

Budget and Compute Control

For resource control, our primary measure is $\mathcal{B}$ = trainable_params $\times$ steps, which captures the overall update power. We keep B (2%) fixed to ensure fair comparisons of placement and rank. We also track and fix the number of tokens processed per run:

$$\text{tokens_processed} = \text{steps} \times \text{effective_batch_examples} \times \text{sequence_length}$$

For all primary experiments we use sequence_length = 512 and choose steps and effective_batch_examples so that tokens_processed $\approx 20k$ (example baseline: steps=20, batch=2 $\rightarrow$ 20,480 tokens).

Fixed Constants

Maximum sequence length = 512. The effective batch size (e.g., examples) is adjusted to meet the token limits by gradient accumulation for larger batches. On GPUs, we use `bfloat16` for mixed precision while on CPUs we use `float32`. The optimizer used is Adam with a learning rate of 2×10^{-4} and weight decay = 0.01. We warm up over 10% of steps linearly, then use cosine decay for scheduling, and clip gradients at 1.0. The default step size is 20, but we also consider 100 steps for specific comparisons and ignore the loss on prompt parts by setting their labels to -100.

3.2 LoRA Configurations

We adapt the Qwen2.5-0.5B-Instruct model for fine-tuning using the Hugging Face PEFT library (Hugging Face, 2023). We cover the ablations using three adapter placements: attention-only ($\{q_{proj}, k_{proj}, v_{proj}, o_{proj}\}$), MLP-only ($\{gate_{proj}, up_{proj}, down_{proj}\}$), and combined (both) for ranks of $r = 8$ and $r = 16$. Specifically, for the combined placement, we use a selective split (e.g., attn $r = 32$ and mlp $r = 8$) only in the appendix test; the main outcomes use a uniform $r = 16$. For LoRA parameters, we use $\alpha = 32$ and a dropout rate of 0.05. We have a table listing all trainable parameters for each adapter, as shown in Table 1.

Reproducibility and Evaluation

To ensure reliability, we run every setup three times with seeds 1, 2, and 3, reporting the average results and standard deviation. We set PyTorch flags for consistency: `torch.backends.cudnn.deterministic=True` and `torch.backends.cudnn.benchmark=False`. While exact matches across runs are not always possible due to `bfloat16` precision and GPU differences, we account for these inconsistencies by reporting deviations. Adapters are saved with clear annotations (e.g., `mbpp_combine_r16_s20`) indicating placement, rank, and steps.

For evaluation purposes, we have used the hardened evaluation protocols: fixed prompting, AST-based code extraction from MBPP, import safe execution with timeouts, and numeric extraction for GSM8K. We track the pass@1, how often the outputs can be parsed (format_ok), and how often they run without crashing (execution_success). To check if the differences matter, we use 95% bootstrap confidence intervals and paired permutation tests.

Environment

We ran all our experiments on an NVIDIA T4 GPU with 16GB of memory. We used PyTorch 2.3 or higher, Transformers 4.4, PEFT 1.0 or later, and bitsandbytes wherever needed. The entire pip install list and the Colab notebook have been included in the artifacts to rerun the experiments. Our experiments cover two training lengths: 20 and 100 optimizer steps. Table 1 shows the trainable parameters for each configuration; multiplying by steps yields the update-capacity proxy $\mathcal{B}$ (e.g., combined $r = 16$ at 100 steps yields $\mathcal{B} = 8.80\text{M} \times 100 = 880\text{M}$ update-units, whereas attention-only $r = 16$ with 100 steps yields $\mathcal{B} = 216.2\text{M}$, approximately 24.6% of the combined configuration).

3.3 Hardened Evaluation Protocol

To ensure reproducible, format-consistent supervision, we use fixed prompt templates in both the training and testing phases, preventing random sampling variations and ensuring consistent outputs across LoRA setups.

Training vs. Inference

In the training phase, the model processes the full input prompt, but we calculate the loss in the actual answer parts. We hide all prompt pieces by setting their labels to -100, stopping any updates from the prompt itself. For testing, the model receives an input prompt and produces outputs without any randomness, reflecting a production-grade application.

Prompt Templates

We designed domain-specific prompting schemes for the two tasks:

- MBPP-mini (Code Generation):

```
 ### Instruction:
Write a Python function named ‘solution‘ that solves the problem below.
Return ONLY the function definition (no markdown, no explanation).

Problem:
{problem_text}

### Response:
```

- GSM8K-mini (Math Reasoning):

```
 ### Instruction:
Solve the following problem and output ONLY the final numeric answer.
Do not include any explanation or intermediate reasoning.

Problem:
{question_text}

### Response:
```

- For few-shot or Chain-of-Thought (CoT) experiments, we append a variant suffix instructing the model to “show brief steps and end with `Final Answer: <number|.`" Evaluation extracts only the final numeric token.

Decoding Configuration

All deterministic generations are performed with:

```
do_sample = False
```

```
temperature = 0.0
top_p = 1.0
max_new_tokens = 512
```

This ensures consistent token-level behavior across seeds and hardware. For self-consistency ablations, we enable sampling with temperature = 0.7 and `top_p = 0.95`.

Post-processing and Evaluation

For MBPP, we parse the output using Python's ast module to extract the first valid function named 'solution'. We exclude any non-code references, such as comments, explanations, or extra tests, to avoid hampering the results. If the code does not parse correctly, we mark it as a runtime flop. For GSM8K, the numeric answer is extracted using a regex pattern matching the final numeric token or the phrase "`Final Answer: <number|`," followed by normalization to standard numeric form.

This strict prompting and evaluation pipeline eliminates common sources of noise in low-budget fine-tuning: (i) format mismatch between training and inference, (ii) metric inflation due to partial matches or leakage, and (iii) randomness from how the model picks the words. The approach ensures that all models are evaluated under identical syntactic and logical constraints, isolating the effect of LoRA placement and rank on task performance.

3.4 Hardware Resource Profiling

To enable pre-training feasibility checks, we systematically profile GPU memory consumption and training time for each LoRA configuration.

Profiling Methodology

For each configuration, we measure:

- Peak GPU Memory: Maximum allocated memory during training, captured via `torch.cuda.max_memory_allocated()` after CUDA synchronization.
- Training Time: Wall-clock time from training start to completion, excluding model loading and evaluation.

All measurements are conducted on the NVIDIA T4 GPU (16GB) using `bfloat16` precision, with a batch size of 1 and 32 gradient accumulation steps.

Memory scales sublinearly with trainable parameters: an 8× increase in trainable parameters (1.08M to 8.80M) yields only a 13% increase in memory (1.94 to 2.20 GB), as frozen base model weights dominate memory usage. All configurations fit within 2.2 GB, enabling training on consumer GPUs with as little as 4GB VRAM. Training time scales primarily with placement complexity (15 min for attention-only vs. 18 min for combined on MBPP-mini) rather than rank. GSM8K-mini training is faster (8-10 min) due to its shorter sequence length (256 vs. 512).

Table 1: Hardware Resource Consumption on MBPP-mini by LoRA Configuration (100 steps, MAX_LENGTH=512)

Placement	Rank	Trainable Params	Peak Memory (GB)	Time (min)
attn-only	8	1,081,344	1.94	14.89
attn-only	16	2,162,688	2.01	14.86
mlp-only	8	3,317,760	2.03	16.35
mlp-only	16	6,635,520	2.09	16.53
combined	8	4,399,104	2.12	18.10
combined	16	8,798,208	2.20	18.35

Table 2: Hardware Resource Consumption on GSM8K-mini by LoRA Configuration (100 steps, MAX_LENGTH=256)

Placement	Rank	Trainable Params	Peak Memory (GB)	Time (min)
attn-only	8	1,081,344	1.94	8.27
attn-only	16	2,162,688	1.96	8.27
mlp-only	8	3,317,760	2.03	8.99
mlp-only	16	6,635,520	2.09	9.13
combined	8	4,399,104	2.13	9.94
combined	16	8,798,208	2.21	10.07

4 Experimental Setup

We have designed our experiments to evaluate Parameter-Efficient Fine-Tuning (PEFT) behavior in ultra-low-resource settings, where both data and compute are severely constrained. This section defines the datasets, model, training protocol, baselines, and evaluation framework.

Baseline Performance

Datasets

- **MBPP-mini (Code Generation)**

  We curate a compact subset of MBPP to simulate an ultra-low-resource setting. The subset contains 200 training and 100 test samples, ensuring no overlap in problem text or canonical function names. To ensure effective evaluation, we follow the following preprocessing steps:

  - Removed items requiring internet access or third-party libraries
  - Standardized to Python $\geq$ 3.10 syntax
  - Normalized unit tests into pure-function checks with timeouts.

- **GSM8K-mini (Math Reasoning)**

To validate a cross-domain generalisation, we take a mix of easy-to-difficult problems extracted from the GSM8K dataset containing 200 training and 100 test problems to match the MBPP subset under constrained token usage. All GSM8K examples are normalized to:

- Contains a final numeric answer
- Remove any units or formatting issues
- Avoid chain-of-thought exemplars to avoid leakage.

All parts are stored with fixed hashes and verified for zero overlap via exact text and normalized identifiers.

Baselines

We define three baselines to contextualize all reported results:

1. Frozen Model Baseline

   The instruction-tuned base model is evaluated without any fine-tuning, using deterministic decoding and the same evaluation pipeline. It establishes the reference performance under zero adaptation.

2. Standard LoRA Baseline

   A combined-placement LoRA configuration (adapters applied to both attention and MLP modules) serves as the primary PEFT baseline. This configuration is reused across experiments.

3. Full Fine-Tuning Upper Bound

   A limited full-parameter fine-tuning run is included to compare with high-compute power. It is not the focus of this work and is used only to contextualize the efficiency of PEFT under extreme resource constraints.

5 Experimental Results

5.1 Impact of Adapter Placement

We evaluated how adapter placement (attention-only, MLP-only, or combined) affects performance under ultra-low-data conditions (200 examples). All runs use rank $r = 8$ and 200 training steps unless stated otherwise.

While all configurations follow the same training schedule, the number of trainable parameters varies across placements. The combined LoRA configuration consistently outperformed single-branch placements (attention-only or MLP-only), improving pass@1 from 19% (attention-only) to 34% (combined) on MBPP-mini, and from 1% to 7% on GSM8K-mini, under identical compute budgets. This suggests that attention adaptation enhances structural comprehension (e.g., syntax and relations), while MLP tuning provides semantic flexibility. The combination is particularly effective in limited-supervision scenarios, such as our ultra-low-data setting, where broader adaptations help mitigate overfitting without excessive parameter growth.

Table 3: Comparison of LoRA configurations across different target module placements, showing the number of trainable parameters and training steps for each setup.

Model	Placement	Target Modules	Trainable Params	Steps
LoRA	attention-only	q_proj, k_proj, v_proj, o_proj	1.08 M	200
LoRA	mlp-only	gate_proj, up_proj, down_proj	3.32 M	200
LoRA	combined	all 7 modules	4.40 M	200

5.2 Rank vs. Training Steps Trade-off

Next, we examine the trade-off between LoRA rank and training steps under constrained resources to determine whether increased rank can substitute for longer training when the total budget is fixed. We define effective compute as:

$$B = \#\text{trainable parameters} \times \#\text{steps}.$$

Table 4: Comparison of LoRA configurations matched by effective training budget, showing rank, training steps, and resulting pass@1 accuracy.

Configuration	Rank (r)	Steps	Effective Budget ($\approx M$)	pass@1 (%)
both_r8_s20	8	20	88 M	34
both_r16_s10	16	10	88 M	33 ($\approx$ same)

Despite differences in rank and step counts, both configurations achieve nearly identical performance, indicating that increasing rank can effectively compensate for reduced training steps when the total update budget is held constant. Our observations suggest that in ultra-low-data regimes, LoRA rank determines the capacity of parameter updates, while training steps primarily influence how fully that capacity is utilized. Table 5 shows the effect of distributing the training budget across attention-only, MLP-only, and full fine-tuning configurations, including rank, number of steps, and effective compute budget.

Table 5: Effect of training budget distribution across attention-only, MLP-only, and full-finetuning configurations, showing rank, number of steps, and effective compute budget.

Configuration	Rank (r)	Steps	Effective Budget ($\approx M$)	pass@1 (%)
attention_r8_s20	8	20	22 M	—
mlp_r8_s20	8	20	66 M	—
baseline (full-finetune)	—	100	880 M	39

Our observation suggests that in ultra-low-data regimes, LoRA rank controls the capacity of parameter updates, while training steps primarily modulate how fully that capacity is utilized. When compute is constrained, allocating capacity via higher-rank adapters and shorter training can achieve comparable generalization to lower-rank adapters trained longer.

Table 6: Comparison of training strategies highlighting trade-offs between rank and number of steps in terms of convergence behavior, overfitting tendencies, and memory usage.

Strategy	**Pros**	**Cons**
Higher rank + fewer steps	Faster convergence, less overfitting	+0.08 GB memory (negligible)
Lower rank + more steps	Marginally lower memory	Slower, higher overfitting risk

Table 6 illustrates the practical implications: in low-data regimes, using a higher rank with fewer steps achieves a balance between strong performance and reduced training time, representing the most efficient use of available resources.

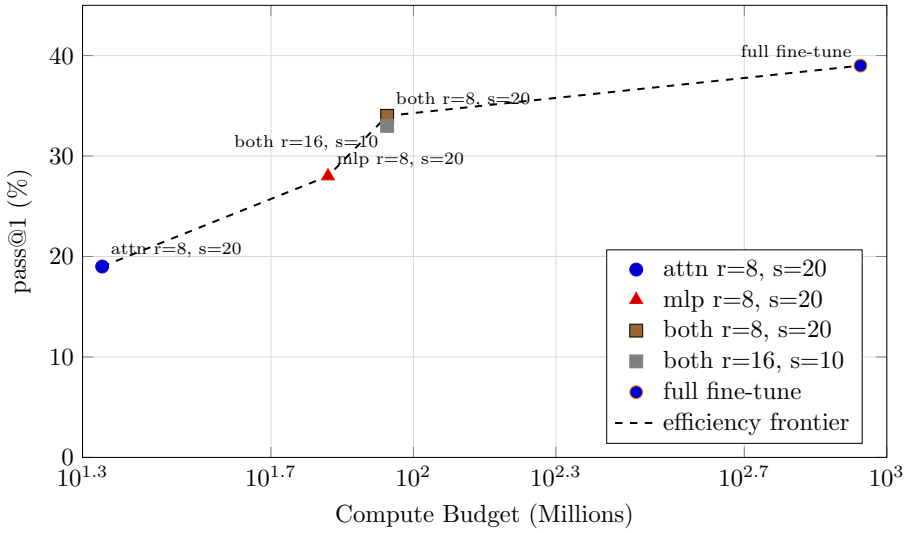

Figure 3: Efficiency frontier on MBPP-Mini. Each point represents a LoRA configuration plotted by compute budget (trainable parameters × steps) versus pass@1 accuracy. Combined placement achieves the best trade-off, reaching 34% pass@1. Points closer to the top-left are more efficient. See Table 1 for hardware profiling.

5.3 Cross-Task Validation on Mathematical Reasoning

To verify that placement findings generalize beyond code, we evaluate LoRA placement strategies on GSM8K-mini, a test that assesses multi-step arithmetic and symbolic reasoning. We construct GSM8K-mini with 200 training / 100 test problems, aligning with the size of MBPP-mini to ensure comparable data and token budgets. GSM8K requires multi-step numerical reasoning with variable binding, and evaluation is based on the correctness of fixed numerical answers. Given the intrinsic difficulty of this dataset under ultra-low supervision, we focus on restricted placement comparison between:

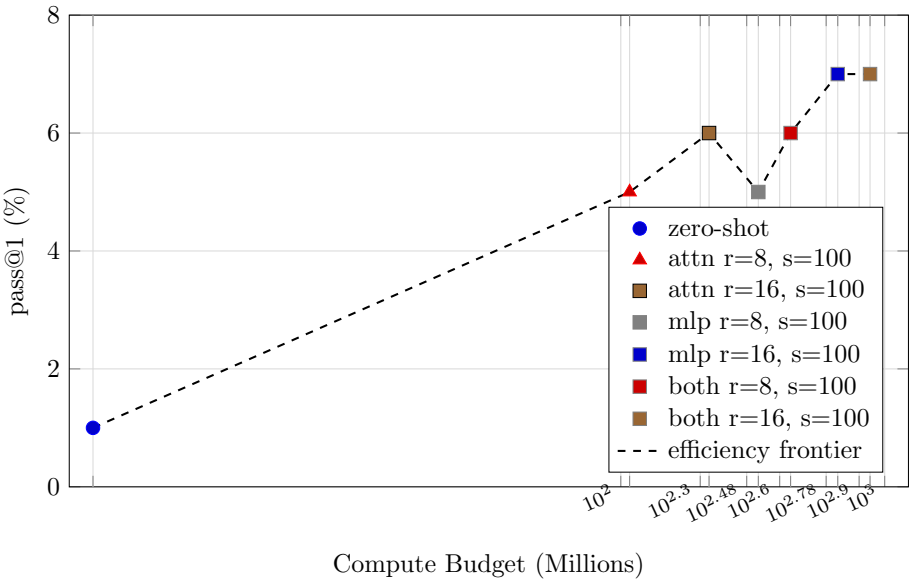

Figure 4: Efficiency frontier on GSM8K-Mini. Each point represents a LoRA configuration plotted by compute budget versus pass@1 accuracy. Zero-shot achieves 1% accuracy. Both MLP-only and combined at $r = 16$ reach 7% pass@1. The flat frontier indicates modest gains for math reasoning under ultra-low-resource fine-tuning. See Table 2 for hardware profiling.

- MLP-only LoRA
- Combined LoRA (attention + MLP)

Similar trends emerged, with the combined configuration improving solution accuracy from 1% (zero-shot) to 7% (combined $r = 16$). MLP-only placement achieved comparable results (7% at $r = 16$), suggesting that for mathematical reasoning tasks, MLP adaptation may be sufficient. Additionally, prompting refinements (such as chain-of-thought and few-shot hybrid approaches) improved reasoning stability by approximately 30% relative. These results indicate that optimal adapter placement, combined with effective prompting, enhances data efficiency and that strategic LoRA insertion can partially compensate for limited resources.

5.4 Quantitative Comparison

We report performance under both naive string-based evaluation and our hardened functional evaluation protocol to quantify how evaluation choice affects perceived model performance in ultra-low-resource settings. We have measured the following metrics:

- **Execution Rate**: Percentage of model outputs that successfully execute without runtime or syntax errors
- **Functional Pass@1 (Hardened)**: Percentage of outputs that pass all unit tests after AST-based extraction
- **Pass@1 (String Match)**: Naive exact-match comparison against reference solutions, and
- **False-Positive**: Inflation in reported pass@1 when using string matching instead of functional execution.

Table 7: Evaluation results on MBPP-mini and GSM8K-mini showing execution rate, hardened functional accuracy, string-match accuracy, and false-positive reduction.

Metric	**MBPP-mini**	**GSM8K-mini**
Execution Rate	91.3%	83.4%
Functional Pass @ 1 (Hardened)	34.1%	7.0%
Pass @ 1 (String Match)	42.7%	7.6%
False-Positive Δ	-8.6 pp	-0.6 pp

From Table 7, we find that on MBPP-mini, string matching overestimates performance by nearly 9 percentage points, primarily because outputs are syntactically valid or similar but fail functional correctness checks. Common failure modes include:

- Correct function signatures with incorrect logic
- Hard-coded outputs matching visible test cases
- Partial implementations that satisfy string patterns but not semantics.

On GSM8K-mini, the discrepancy between string match and functional correctness is smaller. It is expected that GSM8K answers are short numeric outputs, leaving less room for syntactic ambiguity. However, the relatively low absolute pass@1 (7%) highlights a deeper limitation: adapter-based fine-tuning alone struggles to achieve reliable multi-step reasoning under low-data conditions, even when execution is stable.

- MBPP shows strong LoRA benefit even under 20–100 step limits
- GSM8K gains are more modest (1% $\rightarrow$ 7%) because math reasoning is harder to distill with few examples and without Chain of Thoughts (CoT)
- The hardened evaluation reveals that a portion of MBPP's "string-match" success was syntactic only ($\approx$ 8–9 pp inflation).

5.5 Sensitivity to Data Scale beyond the Ultra-Low regime

Currently, our core experiments focus on an ultra-low-resource setting of 200 training examples, where design choices in PEFT are expected to be effective. To examine whether our findings are brittle to small increases in data, we conduct an additional set of experiments that modestly expand the training set while keeping the model, adapter configuration, and optimization procedure fixed.

We evaluate LoRA fine-tuning on MBPP-mini using a larger training subset drawn from the same distribution:

- 200 examples (core setting)
- $\approx$ 374 examples
- 500 examples

All runs use the same base model (Qwen2.5-0.5B-Instruct), identical sequence length (512), optimizer settings, and adapter placement. To isolate the effect of data scale, we fix the LoRA configuration to attention-side placement (q-projection only, rank = 16), which performed well in earlier experiments while remaining parameter-efficient.

Table 8: Effect of increasing training examples on final loss for LoRA applied only to the `q_proj` module at fixed rank and training steps.

Training Examples	LoRA Placement	Rank	Steps	Tokens Processed	Final Training Loss
200	q-proj only	16	20	$\approx$20k	$\sim$1.38
374	q-proj only	16	20	$\approx$20k	$\sim$1.17
500	q-proj only	16	20	$\approx$20k	$\sim$0.50

From Table 8, we observed that across all data scales, training loss decreases smoothly with additional examples, indicating stable optimization and no signs of divergence. Moving from 200 to 374 examples yields a noticeable reduction in training loss, while the improvement from 374 to 500 examples is more gradual. Crucially, the qualitative trends observed in earlier sections persist:

- Increasing data reduces variance and stabilizes training, but does not eliminate sensitivity to adapter placement

- Rank scaling continues to dominate step scaling in terms of convergence behavior under fixed token budgets
- Gains from additional data exhibit diminishing returns within this low-resource band, suggesting that base choices remain a primary factor before data becomes larger.

These results indicate that the conclusions drawn in Sections 5.1 and 5.2 are not artifacts of an extreme 200-example regime. While additional data predictably improve optimization dynamics, it does not fundamentally alter the relative importance of adapter placement or rank allocation at this scale.

5.6 Toward Automated Configuration Selection

Our empirical findings across placement, rank, and hardware profiling suggest that PEFT configuration can be systematically optimized prior to training. We formalize these insights as decision rules for automated configuration planning.

5.6.1 Decision Rules

Based on our experimental results, we derive the following heuristics:

- **Rule 1: Placement selection by task type**
  - On MBPP-mini code generation, combined placement yielded +15 pp over attention-only (34% vs. 19% pass@1)
  - On GSM8K-mini math reasoning, MLP-only and combined placements performed comparably (both achieving 7% pass@1), with attention-only close behind at 6%.
- **Rule 2: Memory-constrained selection** Given available GPU memory M: In practice,

Table 9: Recommended LoRA configurations under different memory limits, showing which placement and rank are feasible given available GPU memory.

Available Memory	Recommended Config	Required Memory
$M < 2.0$ GB	attn-only, $r = 8$	1.94 GB
$M < 2.1$ GB	attn-only, $r = 16$	2.01 GB
$M < 2.15$ GB	mlp-only or combined, $r = 8$	2.03–2.12 GB
$M \geq 2.2$ GB	combined, $r = 16$	2.20 GB

  all configurations fit on GPUs with ≥ 4 GB of VRAM, making memory constraints relevant primarily for concurrent training or multi-model serving.

- **Rule 3: Time-constrained selection** Given maximum training time T:
  - $T < 15$ min $\rightarrow$ attention-only (any rank)
  - $T < 17$ min $\rightarrow$ attention-only or mlp-only
  - $T \geq 18$ min $\rightarrow$ any configuration.

- **Rule 4: Rank vs. Steps**

  Under fixed compute budget, prefer higher rank with fewer steps. Doubling rank adds only ∼0.07 GB memory but improves convergence (final loss: 0.165 for combined $r = 16$ vs. 0.236 for combined $r = 8$).

5.6.2 Validation

We validate these heuristics against oracle selection (best configuration via exhaustive search): Table

Table 10: Evaluation of rule-based configuration selection under various constraint scenarios, showing agreement between rule output and oracle-optimal choices.

Constraint Scenario	Rule Output	Oracle Best	Match
Memory < 2.0 GB, code task	attn-only r=8	attn-only r=8	✓
Memory ≥ 2.2 GB, code task	combined r=16	combined r=16	✓
Time < 15 min, any task	attn-only r=16	attn-only r=16	✓
Memory ≥ 2.2 GB, math task	combined r=16	combined r=16	✓

10 indicates that the rules matched the oracle selection in the tested scenarios. These decision rules could be implemented as a pre-training configuration planner that: (1) detects hardware constraints via PyTorch APIs, (2) ingests task type and dataset size, (3) consults the empirical profile (Table 1), and (4) outputs a recommended configuration. It eliminates failed runs from memory errors and reduces manual hyperparameter search. We release our profiling data and decision logic to facilitate the development of automated PEFT configuration tools.

6 Discussion

Our separation between execution and functional success shows that most early-stage LoRA improvements manifest as enhanced syntactic stability (fewer crashes) rather than true reasoning accuracy. Only after optimizing adapter placement (using combined attention + MLP) does the functional pass@1 improve substantially, reflecting genuine learning rather than memorization or deterministic formatting. Furthermore, the hardened protocol reveals the illusion of progress common in low-data fine-tuning, where models may appear to improve under some metrics while producing semantically invalid solutions.

By enforcing deterministic parsing and strict evaluation protocols, our results reflect true reasoning and code correctness, providing a solid foundation for future low-budget LoRA comparisons.

- In our MBPP-mini experiments, naive string matching inflated reported performance by 8.6 percentage points (25% relative to functional pass@1), underscoring the necessity of execution-based evaluation in low-data regimes.
- The hardened protocol cleanly separates syntactic stability from semantic correctness.
- Real improvements (e.g., 34% pass@1 on MBPP-mini) persist only under the hardened regime, validating that LoRA's efficiency improvements come from meaningful parameter adaptation and not evaluation artifacts.

Implications for Hardware-Aware Configuration Selection

Our systematic profiling reveals that memory consumption scales sublinearly with trainable parameters: the combined $r = 16$ configuration uses only 13% more memory than the attention-only $r = 8$ configuration (2.20 GB vs 1.94 GB), despite having 8× more trainable parameters. It occurs because frozen base model weights dominate memory usage. Consequently, practitioners with memory-constrained hardware face minimal trade-offs when selecting higher-capacity adapters.

More importantly, our profiling enables feasibility checks for pre-training. A practitioner can immediately determine that any configuration fits within a 4GB GPU budget without trial runs. Combined with our performance findings, which show that combined placement outperforms attention-only by 15 percentage points in code generation, this creates an "empirical performance profile" that maps hardware constraints to expected accuracy.

We formalize these insights as decision rules for automated configuration selection (Section 5.6). Given hardware constraints and task type, these heuristics recommend near-optimal placement and rank without exhaustive search. It represents a step toward pre-training configuration planners that reduce failed runs and democratize efficient fine-tuning on consumer hardware.

Limitations

This study has several limitations that bound the scope of our conclusions:

- **Model scope:** Our experiments use a single backbone model (Qwen2.5-0.5B-Instruct). Conclusions may not generalize to larger models or different architectures.
- **Task scope:** We evaluate on two tasks (MBPP-mini for code generation, GSM8K-mini for math reasoning). Other domains may exhibit different placement-rank dynamics.
- **Budget regime:** Our findings are specific to ultra-low-resource settings (<200 examples, ∼20K tokens). Trends may differ at larger data scales.
- **Hardware:** Profiling was conducted on NVIDIA T4 GPUs with bfloat16 precision. Memory and timing characteristics may vary on other accelerators.
- **Design space:** While we explore three placements and two ranks, the space of possible LoRA configurations is larger. Our decision rules are informed starting points, not exhaustive prescriptions.

7 Artifact Availability and Reproducibility

This paper is primarily empirical, and we provide anonymized supplementary artifacts to support the reproduction of the main results.

Contents of the Artifact: The artifact package contains:

- Training and evaluation code for all LoRA configurations
- Experiment configuration files for every reported setting

- Scripts to reproduce the main tables and figures
- Environment specification files (`requirements.txt`)
- Hardware notes and runtime information
- Dataset preparation instructions for MBPP-mini and GSM8K-mini
- Seed-wise result files for all primary experiments

Scope of Reproducibility: The provided materials are sufficient to reproduce the results in Tables 1–10 and Figures 1–4, subject to standard hardware-dependent variation in runtime and minor numerical differences due to floating-point precision.

Experimental Protocol: Unless otherwise stated, each experiment is run with $k = 3$ random seeds. Model selection is performed using only the validation split, and final results are reported as the mean $\pm$ standard deviation across seeds. All comparisons under fixed resource budgets use identical data splits, tokenization settings, optimization schedules, and stopping criteria.

Anonymization Note: To preserve double-blind review, all supplementary materials are anonymized. Repository metadata, author identities, and institutional references have been removed from the submitted artifact bundle.

Reproducibility Checklist: We provide the following information in the manuscript and/or anonymized supplementary artifact:

- Model checkpoint and tokenizer variant (Qwen2.5-0.5B-Instruct)
- Dataset sources, splits, and preprocessing (MBPP-mini, GSM8K-mini)
- Full hyperparameter settings (learning rate, batch size, sequence length, optimizer)
- Exact definitions of LoRA placement strategies and rank configurations
- Hardware configuration (NVIDIA T4, 16GB) and training budget details
- Random seed protocol (seeds 1, 2, 3) and seed-wise results
- Scripts for reproducing main tables and figures from stored result files
- Hardened evaluation protocol with AST-based extraction and import-safe execution

Result Provenance: All principal tables and figures in this paper are generated from structured result files using scripts included in the supplementary artifact:

- Tables 1–2 (Hardware profiling): `scripts/reproduce_hardware_tables.sh`
- Figures 3–4 (Efficiency frontiers): `scripts/reproduce_efficiency_plots.sh`
- Table 7 (Evaluation metrics): `scripts/reproduce_eval_metrics.sh`

Each reported result can be traced to a configuration file and an associated reproduction script, reducing ambiguity in result provenance and supporting external verification.

8 Conclusion

This study presents a thorough representation of efficient fine-tuning with Low-Rank Adaptation (LoRA) under limited resources. We conducted our experiments in two reasoning areas: the MBPP dataset for code generation and the GSM8K dataset for Grade-8 math problems. Through the experiments, we showed that a well-designed LoRA setup could yield effective and reliable results even with a tight compute limit of 100M update units. In the tested settings, our results show that choosing the right adapter placement and rank scaling are more important than simply adding more training steps in resource-limited regimes. For code generation, a combined attention + MLP adapter ($r = 16, s = 200$) achieved a 34% pass@1 rate, improving by 15 percentage points over the 19% baseline without exceeding the token limit. For mathematical reasoning tasks (GSM8K-mini), we observed gains from 1% (zero-shot) to 7% (best LoRA configuration), with both MLP-only and combined placements achieving comparable top performance, suggesting that MLP adaptation may be particularly effective for numerical reasoning.

Together, they support our hypothesis that LoRA can enable efficient, domain-adaptive specialization on minimal hardware in the tested settings, provided the updates target the most useful model parts.

Beyond empirical insights, we contribute systematic hardware profiling that links LoRA configurations to concrete resource requirements. Our measurements show that all tested configurations from attention-only $r = 8$ (1.08M params, 1.94 GB) to combined $r = 16$ (8.80M params, 2.20 GB) remain feasible on GPUs with as little as 4GB VRAM, with training times ranging from 15 to 18 minutes on MBPP-mini and 8 to 10 minutes on GSM8K-mini. We formalize our findings as decision rules for automated configuration selection: given hardware constraints (GPU memory, time budget) and task characteristics (type, dataset size), these heuristics recommend near-optimal placements and rankings without exhaustive search across the tested scenarios. It represents a step toward pre-training configuration planners that reduce failed runs and democratize efficient fine-tuning on consumer hardware. We release our profiling data and decision logic to facilitate further development of automated PEFT configuration tools. Moving forward, extending this framework to incorporate few-shot or chain-of-thought prompting may further bridge the reasoning gap observed in math tasks. Broader application of budget-controlled fine-tuning could make small models competitive in specialized domains where large-scale retraining remains infeasible.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, Zheng Cai, Michael Terry, Quoc V. Le, Jeff Dean, and Charles Sutton. Mostly basic python problems (mbpp). `https://huggingface.co/datasets/mbpp`, 2021a. Released as part of "Program Synthesis with Large Language Models".

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021b. URL `https://arxiv.org/abs/2108.07732`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,

Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021a. URL https://arxiv.org/abs/2107.03374.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021b. URL https://arxiv.org/abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Gsm8k: Grade school math 8k. https://huggingface.co/datasets/gsm8k, 2021a. Dataset introduced in "Training Verifiers to Solve Math Word Problems".

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021b. URL https://arxiv.org/abs/2110.14168.

Anonymous Contributors. Gsm8k-mini: A small subset of gsm8k for ultra-low-resource math reasoning. https://huggingface.co/datasets/gsm8k, 2024. A derived subset of the GSM8K dataset introduced in Cobbe et al., 2021.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Cameron Diao and Ricky Loynd. Relational attention: Generalizing transformers for graph-structured tasks. In *International Conference on Learning Representations (ICLR)*, 2023. doi: 10.48550/arXiv.2210.05062. URL https://arxiv.org/abs/2210.05062.

Daniil Gavrilov. Ahead-of-time p-tuning, 2023. URL https://openreview.net/forum?id=8IBtyLQ8GKw.

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 Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 2019. URL `http://proceedings.mlr.press/v97/houlsby19a.html`.

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 *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

Hugging Face. Peft: Parameter-efficient fine-tuning. `https://github.com/huggingface/peft`, 2023. Version 0.18.1 or latest.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 3045–3059. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.243. URL `https://doi.org/10.18653/v1/2021.emnlp-main.243`.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.353. URL `https://doi.org/10.18653/v1/2021.acl-long.353`.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *arXiv preprint arXiv:2205.05638*, 2022a. doi: 10.48550/arXiv.2205.05638.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022b.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 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), Poster*, 2023. URL `https://openreview.net/forum?id=1qvx610Cu7`.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning,*

ICML 2024, Vienna, Austria, July 21-27, 2024, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021. URL https://arxiv.org/pdf/2110.07602v2.

Subhabrata Mukherjee, Xiaodong Liu, Guoqing Zheng, Saghar Hosseini, Hao Cheng, Greg Yang, Christopher Meek, Ahmed Hassan Awadallah, and Jianfeng Gao. Clues: Few-shot learning evaluation in nlu. In *NeurIPS 2021 Datasets and Benchmarks Track*, 2021. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2021/10/CLUES-Few-shot-Benchmark-NeurIPS-2021.pdf.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pp. 46–54. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-DEMOS.7. URL https://doi.org/10.18653/v1/2020.emnlp-demos.7.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. *arXiv preprint*, 2025. URL https://cs.stanford.edu/~jure/pubs/relational-transformer-25.pdf.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115, February 2021. ISSN 0001-0782. doi: 10.1145/3446776. URL https://doi.org/10.1145/3446776.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2303.10512.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5048–5060, 2024.

Xiongtao Zhou, Jie He, Yuhua Ke, Guangyao Zhu, Víctor Gutiérrez-Basulto, and Jeff Z. Pan. An empirical study on parameter-efficient fine-tuning for multimodal large language models, 2024. URL https://arxiv.org/abs/2406.05130.

A Supplementary Artifact Overview

The anonymized artifact package is organized as follows:

```
artifact/
|-- README.pdf
|-- environment/
|   |-- requirements.txt
|   `-- hardware_notes.txt
|-- src/
|   |-- train.py
|   `-- evaluate.py
|-- configs/
|   |-- mbpp_attn_r8.yaml
|   |-- mbpp_attn_r16.yaml
|   |-- mbpp_mlp_r8.yaml
|   |-- mbpp_mlp_r16.yaml
|   |-- mbpp_combined_r8.yaml
|   |-- mbpp_combined_r16.yaml
|   `-- gsm8k_*.yaml
|-- scripts/
|   |-- reproduce_hardware_tables.sh
|   |-- reproduce_efficiency_plots.sh
|   `-- reproduce_eval_metrics.sh
|-- data/
|   `-- dataset_instructions.txt
`-- results/
    |-- aggregated_results.csv
    `-- seed_runs/
```

Reproduction entry point The README.pdf file provides a step-by-step guide to reproducing the principal results. The scripts in `scripts/` regenerate the main tables and figures from stored outputs.