# Towards a Cost-Effective Domain Adaptation for NLP

**EmpiriNLP Team**

## Abstract

This study investigates cost-effective fine-tuning strategies for large language models (LLMs) within the financial question answering (QA) domain. A set of parameter-efficient fine-tuning (PEFT) approaches- prefix tuning, low-rank adaptation (LoRA), adapters, and programmatic prompts- were evaluated with the FinQA dataset as the benchmark and computation cost measured in floating point operations (FLOs). Of the evaluated methods, prefix tuning achieved the best trade-off between performance and efficiency, while programmatic prompting demonstrated marked accuracy improvements without incurring training costs. This project represents preliminary steps towards a cost-conscious and accessible system of delivering financial knowledge.

## 1 Introduction

Recent advances in machine learning have led to the development of large-scale models that achieve state-of-the-art performance across many tasks. However, these gains come with high computational costs, especially during training (Cottier et al., 2024), which constitute both economic and environmental challenges (Sánchez-Mompó et al., 2025). As model sizes continue to grow, balancing performance with computational efficiency becomes increasingly important.

To address this, a range of parameter-efficient fine-tuning (PEFT) methods have been proposed. These aim to preserve model performance while significantly reducing the compute required for fine-tuning. This is especially valuable in specialized domains like financial QA, where domain-specific fine-tuning is essential, but the cost of doing so repeatedly or at scale can be prohibitive (Phogat et al., 2024).

In this work, we evaluate several PEFT techniques: Low-Rank Adaptation (LoRA), adapters, prefix tuning, and programmatic prompting on a financial QA task. Our goal is to compare their ability to achieve strong task performance and minimize computational cost (Han et al., 2024), measured in floating point operations (FLOs). We use FLOs as our primary cost metric because they are hardware-agnostic, objective, and reproducible, unlike wall-clock time or energy consumption, which can vary across different systems (Sevilla et al., 2022).

As of the time of writing, no prior study has applied a FLOs-based analysis to fine-tuning methods in the financial QA domain. This project aims to support the development of more efficient, cost-effective approaches to model adaptation in specialized settings.

## 2 Related Work

PEFT methods such as adapters (Houlsby et al., 2019), prefix tuning (Li and Liang, 2021), and LoRA (Hu et al., 2022) reduce trainable parameters to under 1% of a model, offering a scalable alternative to full-model fine-tuning in domain-specific or compute-constrained settings. While widely applied in general NLP, their use in financial QA remains limited.

Most prior work in financial NLP has focused on classification or sentiment analysis, with fewer studies addressing numerical QA. FinQA (Chen et al., 2021), the primary dataset used in this work, targets multi-step numerical reasoning over tabular and textual financial data. Other datasets, TAT-QA (Zhu et al., 2021), ConvFinQA (Chen et al., 2022), FinTextQA (Chen et al., 2024), and FinanceBench (Islam et al., 2023), expand the task space but differ in focus, b FinQA remains central for evaluating discrete, multi-hop reasoning in finance.

Recent methods such as QLoRA (Dettmers et al., 2023) and LLaMA-Adapter (Zhang et al., 2023) combine low-rank adaptation with quanti-

zation or modular components. (Phogat et al., 2024) show that small models, when fine-tuned with reasoning examples, can approach larger models' performance on financial QA, emphasizing the potential of efficient fine-tuning strategies. This work focuses on training efficiency and cost, with floating-point operations (FLOs) as a primary proxy (Hoffmann et al., 2022). FLOs, in contrast with other cost metrics such as runtime or memory usage, provide a platform-neutral basis for comparing PEFT methods. Applied to LoRA, prefix tuning, and adapters on FinQA, FLOs present reproducible analysis of performance-compute trade-offs in financial reasoning.

# 3 Method

## 3.1 Dataset

To evaluate model performance and efficiency in the financial QA domain, we utilize the FinQA dataset introduced by Chen et al. (2021). Designed to support complex numerical reasoning over both structured (tables) and unstructured (text) financial data, FinQA comprises approximately 2,800 financial reports and 8,000 question-answer pairs, of which 6,251 are training pairs, 883 are validation pairs, and 1,147 are test pairs. It has become a standard benchmark for financial language modeling, with a public leaderboard on Hugging Face enabling consistent evaluation and direct model comparison. The dataset presents natural language questions paired with financial tables, requiring multi-step reasoning and numerical computation. It includes distinct training and test splits for fine-tuning and evaluation. To address issues of data incompleteness, particularly in the test set, preprocessing was conducted to remove entries with missing questions, answers, or type inconsistencies.

## 3.2 Baseline

For our experiments, we selected LLaMA-3.1-8B-Instruct as the baseline model and evaluated its zero-shot performance. This model was chosen due to its strong results on public leaderboards, suitability for fine-tuning, accessibility for reproducible research, and alignment with our resource constraints. It strikes a useful balance between model size and usability, making it effective for assessing fine-tuning efficiency. The model weights were obtained through Hugging Face.

Given hardware memory limitations, a set of efficiency optimization strategies was implemented during training and evaluation, including: model quantization, reduced batch size and sequence length, and gradient checkpointing to minimize memory usage during backpropagation.

## 3.3 Fine-tuning methods

Several PEFT methods were evaluated for their trade-off between model performance and computational costs. Each method, including their motivation and application to the central task, is described in detail below.

### 3.3.1 Prefix Tuning

Prefix tuning is a PEFT method that optimizes a small set of task-specific vectors called prefixes, which are prepended to the model's input at each transformer layer. These prefix embeddings are learned while keeping the original model parameters frozen. Thus, only a minimal number of additional parameters are trained, making prefix tuning effective for reducing both memory usage and FLOs during training (Li and Liang, 2021).

For our experiments, we used the LLaMA-3.1-8B-Instruct model and implemented prefix tuning using Hugging Face Transformers with the PEFT library. The setup involved learning 100 prefix tokens, with the rest of the model remaining unchanged. Training was performed over 3 epochs with a batch size of 1.

### 3.3.2 LoRA (Low-Rank Adaptation)

LoRA (Low-Rank Adaptation) is a PEFT method that reduces computational overhead by injecting trainable low-rank matrices into the weights of a pre-trained model. By decomposing weight updates into two smaller matrices rather than modifying the full weight matrix, LoRA enables effective task adaptation with a minimal number of trainable parameters, thereby reducing FLOs and memory consumption during training (Hu et al., 2022).

We applied LoRA using the PEFT library on a LLaMA-3.1-8B-Instruct model loaded from Hugging Face Transformers. The dataset was locally cleaned by removing entries with missing answers, and a custom PyTorch Dataset was implemented for preprocessing and efficient loading. Prompts included pre- and post-text, table content, and task-specific instructions to guide model reasoning during training and inference.

#### 3.3.3 Adapters

Adapters are a PEFT technique that introduce small, trainable dense layers in bottleneck style into a frozen pre-trained model, enabling task-specific adaptation with minimal additional parameters (Houlsby et al., 2019). In contrast to methods like LoRA or Prefix Tuning, which modify attention mechanisms or input prefixes respectively, adapters operate by inserting lightweight neural modules between transformer layers.

As adapter-style fine-tuning is somewhat old-fashioned, few libraries support bottleneck-only implementation, without LoRA or other hybrid methods. `PEFT` and `Unsloth` libraries were discarded, as only `adapter` (AdapterHub) has adapter-only framework.

Llama-3.1-8B-Instruct was used in fine-tuning. The model is trained on a subset of FinQA training set for maximum of 15 epochs. At end of each epoch and 10 steps, validation is done on dev set using default loss provided in `adapter`. Customized token-level accuracy was considered but discarded due to complexity and generalisability. Early stopping is enabled from the framework to stop at early epochs if the loss patience is reached. Finally, the best checkpoint is selected at the end of run to inference and report accuracy on test set.

#### 3.3.4 Programmatic Prompt

To investigate programmatic reasoning, we modified the baseline prompts to request that the model produced executable Python code, adopting a Chain-of-Thought (CoT) approach rather than generating the answer directly. The concept builds on ideas discussed in (Reynolds and McDonell, 2021), which explores structured and programmatic approaches to interacting with large language models (LLMs). The generated code was executed, and its output was evaluated using the same normalization procedure. Failures in running the generated code were treated as empty answers. This setup allowed us to compare the models' performances across both reasoning strategies- direct answer generation and programmatic reasoning.

To evaluate the impact of programmatic prompting, we compared it to non-programmatic baselines where models are prompted to generate only final answers. We conducted experiments using Phi-3.5, LLaMA-3.1-8B-Instruct, and GPT-4o, evaluating each model under both prompting strategies.

## 4 Experiments

This section outlines the empirical setup used to evaluate the models and fine-tuning approaches. All experiments were logged and tracked on GitHub for reproducibility and transparency. Git Large File Storage (git-lfs) was used to manage file versioning and efficient storage due to the large size of some model checkpoints and intermediate outputs. The full experimental repository is available at: `https://github.com/EmpiriNLP/FrugalML/`.

### 4.1 Experimental setup

Experiments were conducted on lab computers equipped with either NVIDIA RTX 4060 Ti GPUs (16GB RAM) or NVIDIA RTX 3090 GPUs (24GB RAM). After cleaning the test set of samples with missing or incomplete information to ensure consistency and fairness, training and evaluation were carried out using the train.json and test.json datasets provided by the FinQA benchmark.

#### 4.1.1 Evaluation metrics

The model's performance on the FinQA dataset was evaluated using the thresholded-similarity accuracy. The implementation follows the format suggested by EleutherAI's lm-evaluation-harness and the prompt format used by the Aiera Finance leaderboard. Ensuring accurate comparison with other models on the leaderboard.

To accurately quantify computational cost, FLOs is used as a hardware-agnostic metrics. Additional metrics such as CUDA time, CPU time, memory consumption are also captured to supplement FLOs. These collectively provide insights real-world execution efficiency in aspects like parallelization and memory access patterns. A tracker is implemented to monitor these metrics by wrapping a decorator pattern around training functions. Measurements taken are during trainings steps are: FLOs (computing operations), GPU time, CPU time, and memory usage.

### 4.2 Baseline

To evaluate the initial performance of LLMs on FinQA without task-specific adaptation, a zero-shot baseline was established using the meta-llama/Llama-3.1-8B-Instruct model. The model was loaded with NormalFloat 4-bit double quantization, at bfloat16 precision, float16 precision at

inference, via the Hugging Face Transformers library. Inference was run with a maximum generation length of 20 tokens, using default decoding parameters (temperature=0.6, no top-k or top-p sampling).

Prompt inputs were constructed by concatenating the serialized table context, accompanying textual information, and the target question, followed by a minimal instruction asking for a numerical answer. Tokenization was handled with automatic padding and truncation. All generations were performed without gradient updates or additional training.

Predicted outputs were post-processed to extract numerical content and normalize formatting. This included removing percent signs, currency symbols, and other surface-level variations. Reference answers were cleaned using the same logic to ensure consistency during evaluation. Performance was measured using two metrics: exact match accuracy, calculated over normalized string; and fuzzy string similarity based on a Levenshtein ratio, threshold = 85, to allow for minor formatting mismatching.

This baseline provides a consistent point of comparison for evaluating the effectiveness and efficiency of subsequent fine-tuning methods.

### 4.3 Prefix Tuning

Prefix tuning experiments implemented via Hugging Face Transformers and the PEFT library were conducted using the LLaMA-3.1-8B-Instruct model. The model was trained with 100 prefix tokens for 3 epochs using a batch size of 1, with all original model parameters kept frozen. During training, an initial run with 10 prefix tokens achieved approximately 48% accuracy. Increasing the number of prefix tokens to 100 improved training accuracy to around 56.47%, with an average fuzzy similarity of roughly 72.83%. On the held-out test set, the model achieved 57.19% accuracy and an average fuzzy similarity of about 71.92%. These results are discussed in depth in the Results section.

### 4.4 LoRA (Low-Rank Adaptation)

LoRA-based fine-tuning was applied to the LLaMA-3.1-8B Instruct model and evaluated on FinQA. Before training, the base model was evaluated in a zero-shot setting to establish a baseline. It was loaded from Hugging Face with 4-bit quantization, bfloat16 precision, and FlashAttention-2 enabled. Fine-tuning was applied only to the query, key, value, and output projection layers in the attention blocks, comprising approximately 0.16% of the model's total parameters. The LoRA configuration used a rank of 16, scaling factor (alpha) of 32, and a dropout rate of 0.2, which was increased from the default to mitigate early divergence between training and validation losses.

The optimizer used was adamw_torch, with a learning rate of 2e-4, weight decay of 0.01, and 100 warm-up steps. The training was conducted over 3 epochs, and we tested batch sizes of 1 and 2 with gradient accumulation set to 1. The loss function was kept as the default from the Hugging Face Trainer. To reduce unnecessary computation and memory usage, dynamic padding and truncation were applied during batching.

Training was conducted over three epochs using batch sizes of 1 & 2 in separate experiments. Gradient checkpointing was enabled, and the output layer was cast to bfloat16 to prevent precision mismatches. The default trainer loss function was used, and the optimizer was configured in accordance with the previously described LoRA setup.

Inference was run on the full test set with a 128-token generation limit, and the post-processing pipeline was extended to extract numerical predictions, normalize formatting, and compute exact-match accuracy and fuzzy string similarity. The same tracker utility discussed earlier was used to measure the utilization of different computational resources during model training.

### 4.5 Adapters

Adapter experiments conducted grid search on the following hyperparameters space: batch sizes = {1, 2, 4}; training samples = {50, 200, 1000}; and learning rates = $\{3\times10e^{-5}, 1\times10e^{-4}, 1\times10e^{-3}\}$, following literature suggestions (Houlsby et al., 2019). The default adamw_torch optimizer was used.

Integrating adapters into the LLaMA-3.1-8B-Instruct model required significant reconstruction of the training pipeline. AdapterHub does not attach a causal language modeling head by default, so a custom output layer was manually added to match the model's architecture and data type requirements. Label formatting also required additional care; to ensure loss was computed only over the predicted answer tokens, the training labels were restructured as tensors with non-target tokens

masked to -100. Mismatches between model components, particularly across adapter layers and the newly added output head, also required explicit dtype and device alignment to avoid runtime errors, for both training and inference.

On more technical details, dynamic padding and truncation were enabled with max length set to 4096 to avoid out-of-memory errors. Also, only 100 validation samples were used to prevent loss becoming unstable and turning into `nan` or `inf`. The FLOs for each experiment were recorded natively by `adapter` library, so a customized tracker was unnecessary.

### 4.6 Programmatic Prompt

Each model was evaluated on 1,133 samples in both non-programmatic and programmatic settings. In the baseline setting, models generated final answers based on direct prompts with fine-tuning applied to improve performance. In the programmatic setting, models generated Python code, which was executed in a sandbox to compute the final answer. Output was normalized to remove superficial formatting mismatches.

Accuracy and average similarity for each model are shown in Table 3.

## 5 Results and Discussion

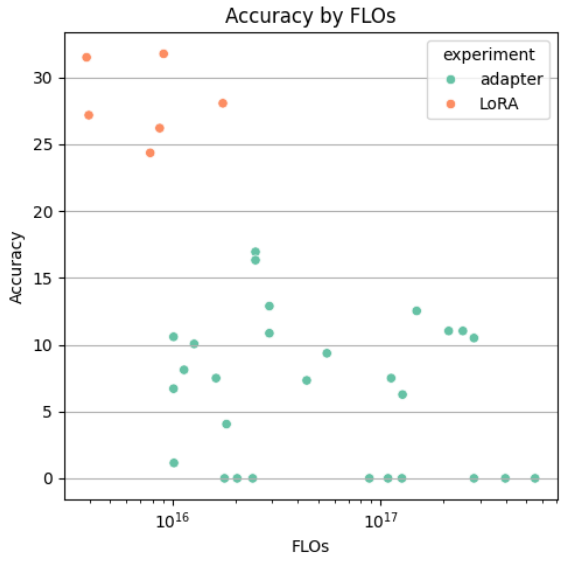


Figure 1: Accuracy vs FLOs for different fine-tuning methods.

Table 1 presents a representative summary of the best-performing, most cost-efficient experiments. Although batch size of 2 yielded the highest accuracy for LoRA, the accuracy improvement is negligible compared to the associated increased in cost of training. Of the methods included in the study, only prefix tuning and programmatic prompts outperformed the baseline zero-shot inference in LLama-3.1-8B-Instruct. The best epoch is similar across fine-tuning techniques- training settings are not applicable to programmatic prompts, which do not need to be trained. The training and inference times largely vary across the experiments, with prefix tuning being the fastest in both.

Heavier fine-tuning methods such as LoRA and Adapter yielded lower accuracy. This may be due to the sharp threshold between underfitting and overfitting, if the model does not learn reasoning as training progresses. These methods may also disrupt the fragile stability of model weights and architecture, leading to divergence in training. The results also indicate that the programmatic prompt method requires by far the most inference time to generate more tokens but saw a marked increase in performance, with accuracy improvement of 15.27% with no training FLOs cost incurred.

This is a clear example of compromising inference time, a stand-in for cost, for accuracy. Finally, most of the methods reached their best performance at the lowest batch size 1 and the best epoch quite early (e.g. 2 or 3), suggesting the model readily overfits on the fine-tuning dataset.

To better visualise cost-effectiveness of the fine-tuning methods, Figure 1 displays the accuracy vs FLOs for the experiments. LoRA achieved higher accuracy using lower FLOs, which aligns with standard findings elsewhere, whereas adapter experiments failed to perform well across a wide range of metrics.

### 5.1 Prefix Tuning

Prefix tuning led to consistent improvements over the zero-shot baseline, while maintaining a low computational footprint. Early experiments with shorter prefix lengths (10 tokens) achieved lower performance (48% accuracy), confirming the benefit of a longer prefix sequence. Final experiments used 100 prefix tokens trained over three epochs with a batch size of one, achieving 57.19% accuracy and 71.92% average fuzzy similarity on the FinQA test set (n = 1,147). Notably, despite the strong numerical metrics, several incorrect predictions still showed high similarity scores, suggest-

| Experiment | Training samples | Batch size | Best Epoch | Training FLOPs | Avg. inference time (s) | Accuracy | Avg. similarity |
|---|---|---|---|---|---|---|---|
| Baseline | N/A | N/A | N/A | N/A | 0.8692 | 39.1% | 70.69% |
| Prefix Tuning | 6251 | 1 | 3 | 1.4e+14 | 0.027 | 57.19% | 71.92% |
| LoRA | 6203 | 1 | 2 | 7.81e+15 | 3.600 | 31.51% | 65.57% |
| Adapter | 200 | 1 | 3 | 2.50e+16 | 1.0752 | 16.95% | 58.5% |
| Programmatic Prompt | N/A | N/A | N/A | N/A | 11.7215 | 54.37% | 70.24% |

Table 1: Summary of most ideal results from each fine-tuning experiments.

| Batch Size | Epoch | Epoch Accuracy | Epoch Similarity |
|---|---|---|---|
| 1 | 1 | 27.18% | 64.02% |
| | 2 | 31.51% | 65.57% |
| | 3 | 24.36% | 62.34% |
| 2 | 1 | 26.21% | 63.42% |
| | 2 | 31.77% | 67.22% |
| | 3 | 28.07% | 64.48% |

Table 2: LoRA Accuracy and Similarity for different batch sizes.

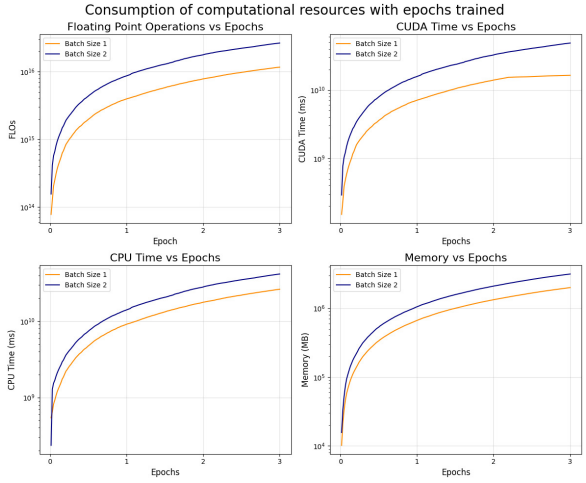


Figure 2: Consumption of computational resources for LoRA with epoch trained

ing surface-level alignment rather than true reasoning.

To systematically measure the computational efficiency of prefix tuning, a lightweight profiler that wraps Huggingface *Trainer.train* method without modifying its core logic. This tracker captures floating point operations (FLOs), GPU and CPU runtime, memory consumption and training loss at configurable speed intervals. Internally, it uses *torch.profiler* with *with_flops=True* to estimate total FLOs per training step, together with GPU memory execution time on both CPU and CUDA devices. The tracker logs results to a structured CSV file, enabling post-hoc analysis and visualization of performance trends over time. This tool was instrumental in quantifying the compute-efficiency of prefix tuning, confirming its low-cost profile (1.4e+14 FLOs, 0.027s inference latency). It also aided debugging during early runs, helping to identify memory bottlenecks and optimize training stability. The same profiler was reused across all subsequent fine-tuning experiments (e.g., LoRA, Adapters), ensuring consistency in metric reporting.

All source code and logs for this tracker are available in the *prefixTuning* branch of the project repository.

### 5.2 LoRA

Table 2 displays the performance of LoRA fine-tuning on FinQA dataset for different epochs and batch sizes. The LLaMA-3.1-8B-Instruct model, when trained for up to 3 epochs, yielded best performance at epoch 2: best validation loss, and best accuracy (31.51% and 31.77% for batch sizes 1 and 2 respectively) and similarity (65.57% and 67.22% for batch sizes 1 and 2 respectively). The model exhibits underfitting for epoch 1 and overfitting for epoch 3, even with a higher dropout rate. An early stopping was thus performed to obtain fine-tuned model weights at the second epoch, highlighting that training for more epochs may not improve model learning and generalizability. Nonetheless, none of the epoch performances surpassed the zero-shot baseline performance, which achieved 39.35% accuracy and 70.21% average similarity.

The model was trained with batch sizes of 1 & 2, with the latter yielding only marginally better performance; however, this cannot be deemed significant evidence of improvement, as it may be due

to chance. On the other hand, the training operation with batch size 1 consumed fewer computational resources, as evident from Fig. 2. The telemetry data for FLOs, GPU Time, CPU Time, and Memory was computed, with the resource utilization appearing to increase linearly with number of epochs (y-axis is on log scale), and the rate of increase higher for batch size 2. At epoch 2, the training FLOs were $7.81e{+}15$ for batch size 1 and $1.77e{+}16$ for batch size 2. These observations demonstrate that increased resource consumption for batch size 2 does not proportionally translate to improvement in performance over batch size 1. All relevant files and outputs for this experiment are hosted in the tanpatil_lora_peft branch of the repository.

### 5.3 Adapters

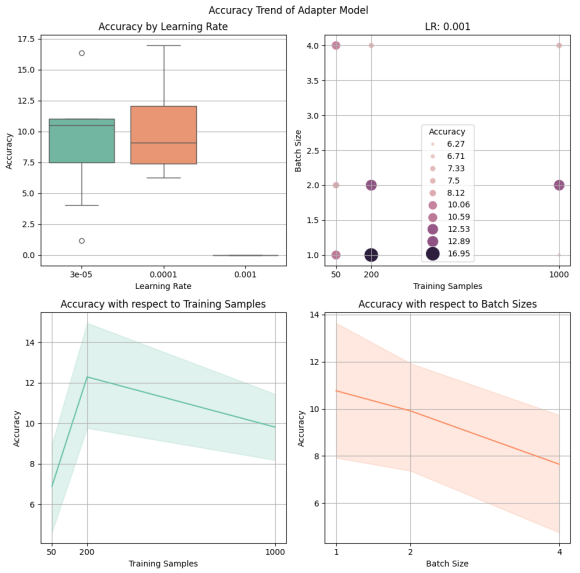


Figure 3: Accuracy trend for adapter experiments at different hyperparameters.

The results of main adapter experiments are shown in Figure 3 and 4. No adapter implementation of the LLaMA-3.1-8B-Instruct model surpassed the zero-shot baseline. Regardless, visualising trends in performance and computation costs could provide insight for further study.

Attempts to run the adapter method on the full dataset were limited by the duration of a single run ( 24hrs), so only only a few representative runs were performed. A run over the full training sample set (n = 6023) with a learning rate of 0.001 and a batch size of 1 yielded an accuracy of 29.83%, outperforming other adapter experiments.

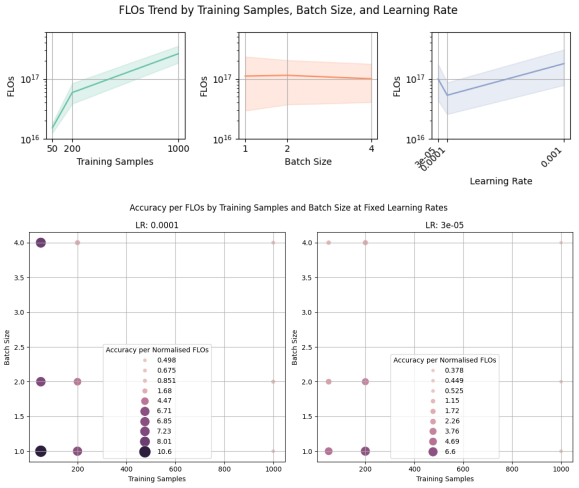


Figure 4: FLOs and Accuracy per Normalized FLOs trend for adapter experiments at different hyperparameters.

Learning rate has a significant impact in adapter fine-tuning. Figure 3 shows that all experiments with learning rate 0.001- the high mode learning rate- yielded 0% accuracy. This suggests excessive learning may overly fine-tune the model, thereby supporting more delicate fine-tuning and smoother updating. Performance peaked at a sample size of 200 compared with 50 or 1000, and the figure also demonstrates smaller batch size correlating with better accuracy.

On FLOs and efficiency, Figure 4 displays increase in computational costs as training samples increase. Batch size did not significantly affect FLOs, which, however, fluctuate with learning rate. Accuracy per normalized FLOs, although an arbitrary metric as an absolute value, is a useful metric in visualising trends for cost effectiveness. Smaller hyperparameter values relate to higher efficiency, peaking at 50 training samples, a batch size of 1, and a learning rate of 0.001.

Regarding accuracy, learning rate and training sample appear to benefit from a balance-oriented approach rather than prioritizing minimization or maximization of values, although lower learning rates provide more stable solutions even as other hyper parameters vary. Smaller batch sizes are associated with both improved empirical accuracy and less resource consumption. FLOs and cost efficiency plots may further emphasize on smaller hyperparameters- with two caveats: that more FLOs cannot be enforced to achieve increase accuracy even if apparently suggested by plots; and that a performance threshold is set so the model

| Experiment | Accuracy | Similarity |
|---|---|---|
| llama-non-programmatic | 39.72% | 70.46% |
| llama-programmatic | 54.37% | 70.24% |
| gpt-non-programmatic | 63.28% | 76.25% |
| gpt-programmatic | 70.17% | 80.37% |
| Phi-3-mini-4k-non-programmatic | 13.5% | 50.12% |
| Phi-3-mini-4k-programmatic | 50.57% | 66.11% |

Table 3: Non-Programmatic and Programmatic Prompt Accuracy and Similarity for different LLMs

minimally satisfies the need. Only past that threshold may the most efficient setting be selecting, e.g. rejecting models trained with 50 samples despite being most efficient if they do not meet an established 10% accuracy threshold.

### 5.4 Programmatic Prompt

Instructing the model to generate a Python program to derive the answer improves accuracy from 39.7% to 54.37%, but increases inference time from 1.2s to 11.7s per sample. This accuracy gain is likely due to the program-generation instruction functioning similarly to a chain-of-thought prompt (Reynolds and McDonell, 2021)), effectively allocating more computation toward answer construction. Similarity scores remain largely unchanged. This effect is consistent across all models tested (Llama-3.1-8B, GPT-4o, Phi-3-mini-4k). The largest accuracy gain is observed in Phi-3-mini-4k (13.5% to 50.12%), and the smallest in GPT-4o (63.28% to 76.25%), suggesting a potential interaction with model size.

### 5.5 Failure Mode Analysis

Also analysing the failure mode of low accuracies in general, it could be due to naive but overdosed fine-tuning. This is further visualized with examples in Appendix A where repetitive nonsense pattern or total inhibition is observed in fine-tuned Adapters. This suggests need of more delicate model weights and architecture updating in the future.

Other possible reasons could due to the need of alignment after fine-tuning, to better follow instructions, like separate small supervised fine-tuning (SFT) or reinforcement learning from human feedback. It could also be the data leakage, i.e. the model has already been pretrained on FinQA, and repeated text has been shown to impair performance (Raffel et al., 2020).

## 6 Conclusion

This study explored lightweight fine-tuning techniques for adapting the QA task to financial domain, considering on both their performance and computational demand. Among the methods explored, prefix tuning stood out as the most efficient fine-tuning method in terms of FLOs and accuracy. On the other hand, programmatic prompting shows that structured reasoning prompts can boost performance significantly, but with increased inference costs. Lastly, while the ideas behind LoRA and adapter-based methods are promising, these methods tend to result in overfitting and are less effective when resources are limited.

This analysis underscores the importance of carefully balancing model accuracy with resource consumption, particularly in high-stakes domains like finance. Future work may explore increase in inference time such as using longer output tokens. Few-shot learning strategies may also lead to better performance by examples. Self-consistency sampling could improve quality of results reducing hallucination. Alignment techniques such as SFT or RLHF could alternatively further enhance model robustness. Expanding beyond FinQA to include other datasets like TAT-QA and ConvFinQA would also help generalize these findings. Ultimately, this study moves toward building more accessible and sustainable NLP solutions for specialized domains.

## References

Jian Chen, Peilin Zhou, Yining Hua, Yingxin Loh, Kehui Chen, Ziyuan Li, Bing Zhu, and Junwei Liang. 2024. Fintextqa: A dataset for long-form financial question answering. *arXiv preprint arXiv:2405.09980*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, et al. 2021. Finqa: A dataset of numerical reasoning over financial data. *arXiv preprint arXiv:2109.00122*.

Zhiyu Chen, Shiyang Li, Charese Smiley,

Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. *arXiv preprint arXiv:2210.03849*.

Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. 2024. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms, 2023. *URL https://arxiv. org/abs/2305.14314*, 2.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

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, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. 2023. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*.

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

Karmvir Singh Phogat, Sai Akhil Puranam, Sridhar Dasaratha, Chetan Harsha, and Shashishekar Ramakrishna. 2024. Fine-tuning smaller language models for question answering over financial documents. *arXiv preprint arXiv:2408.12337*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7.

Adrián Sánchez-Mompó, Ioannis Mavromatis, Peizheng Li, Konstantinos Katsaros, and Aftab Khan. 2025. Green mlops to green genops: An empirical study of energy consumption in discriminative and generative ai operations. *arXiv preprint arXiv:2503.23934*.

Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. 2022. Compute trends across three eras of machine learning. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. 2023. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*.

## A Example Adapter Answers at High Learning Rate

Question 1 Expected Answer: 94

```
Training Samples 50, Batch Size 4, Learning Rate 0.001:
  “”
Training Samples 200, Batch Size 2, Learning Rate 0.001:
  "%%%.%%.....%.%..%%%”
```

Question 2 Expected Answer: 14%

```
Training Samples 50, Batch Size 4, Learning Rate 0.001:
  “”
Training Samples 200, Batch Size 2, Learning Rate 0.001:
  "%%.%.%%%..%..%%.%%%.”
```