International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Parameter-Efficient Fine-Tuning for Generative AI: Implementations, Best Practices, and Trade-Offs

Chandan Singh Troughia[1], Sriraman Suryanarayanan[2]

[1]Staff Machine Learning Engineer, CVS Health
ORCID: 0009-0004-2921-6355

[2]Senior Manager, Data Engineering, CVS Health
ORCID: 0009-0002-9564 – 466

Abstract: ***Large language models (LLMs) have revolutionized natural language processing (NLP), yet full fine-tuning remains computationally expensive. Parameter-efficient fine-tuning (PEFT) techniques offer a solution by modifying only a small number of parameters. This paper provides a comparative analysis of key PEFT strategies, including Adapters, Low-Rank Adaptation (LoRA), Prompt Tuning, and Prefix Tuning, discussing their best practices, trade-offs, and empirical evaluations. We present a real-world implementation demonstrating the efficacy of these methods on a text classification task.***

Keywords: Generative AI, Parameter-efficient fine-tuning (PEFT), Adapters, Low-Rank Adaptation (LoRA), Prompt Tuning, Prefix Tuning, Natural Language Processing (NLP)

1. Introduction

Large language models (LLMs), such as GPT-4, LLaMA, GPT-NeoX, GPT-4.5 Orion, Claude 3.7 Sonnet, Grok-3, and Google's Gemini series, have significantly advanced artificial intelligence applications, improving capabilities in text generation, language translation, and question answering. Despite these advancements, adapting these increasingly largescale models to specific downstream tasks through traditional fine-tuning methods presents considerable challenges. Traditional fine-tuning typically involves retraining or updating all parameters within the pre-trained model, leading to substantial computational demands and extensive memory requirements, often exceeding readily available hardware capabilities.

As LLMs continue to scale, concerns regarding their carbon footprint and sustainability have also gained attention. Researchers are exploring alternative fine-tuning methods that balance accuracy with environmental efficiency. Additionally, the rising cost of training and maintaining these models has pushed enterprises to seek cost-effective solutions. PEFT methods present a viable way forward, offering substantial improvements in computational efficiency without sacrificing model adaptability.

Parameter-efficient fine-tuning (PEFT) offers a solution by selectively modifying a subset of model parameters, reducing computational resources while preserving pre-trained knowledge. This paper provides a comparative analysis of prominent PEFT techniques: Adapters, Low-Rank Adaptation (LoRA), Prompt Tuning, and Prefix Tuning, discussing their best practices, trade-offs, and empirical evaluations. We present a real-world implementation demonstrating the efficacy of these methods on a text classification task.

The contributions of this paper include a structured comparative analysis of key PEFT strategies, an evaluation of tradeoffs, and practical insights from a real-world implementation.

2. PEFT Techniques

a) Adapters

Adapters are small, trainable neural networks inserted between the layers of a pre-trained model [1]. The original model parameters remain frozen, and only the adapter layers are updated. This modular approach facilitates multi-task learning and domain adaptation, allowing for easy integration of multiple tasks by swapping adapters. Research has shown that adapters can achieve performance comparable to full fine-tuning with significantly fewer trainable parameters [2].

Architecture:

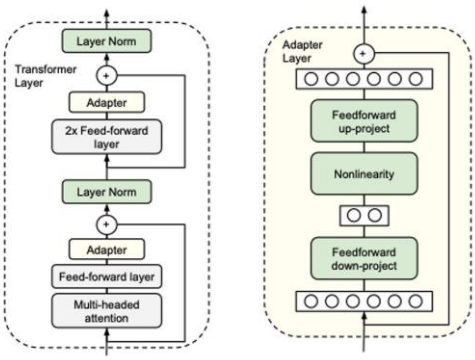

Figure 1: Diagram of the Adapter Module. Adapted From [2]

In this illustration, two adapter modules are added to each Transformer layer:

- One after the multi-headed attention block
- Another after the feed-forward block

Each adapter module is designed as a bottleneck: it reduces the dimension of the hidden state (down-projection), applies a nonlinear transformation, and then restores it to the original dimension (up-projection), with a skip connection to stabilize training. Only these adapter modules (and a few related parameters, like layer normalization) are trainable during fine-tuning, while most of the Transformer parameters remain frozen [2].

By training just these smaller adapter modules, we significantly decrease the total number of trainable parameters, making the fine-tuning process more efficient than traditional full fine-tuning.

Benefits of Adapters:

- Parameter Efficiency: Adapters drastically reduce the number of trainable parameters, allowing multiple tasks to share the same backbone with minimal overhead.
- Modular Design: Different adapters can be swapped in and out for different tasks, facilitating multi-task learning and continuous integration of new tasks.
- Stable Training: The bottleneck architecture helps maintain stability and preserve core model knowledge.
- Use cases of Adapters:
- Multi-Task Learning: Where one wants to maintain a single base model and switch adapter modules for each task.
- Domain Adaptation: Easy to add domain-specific adapters without re-training the entire model.
- Resource-Constrained Environments: Ideal where memory and compute are limited.

Best Practices: Use Adapter Fusion for multi-task learning to combine knowledge from different adapters.

Trade-offs: Requires additional model parameters, increasing the overall model size.

b) Low-Rank Adaptation (LoRA)

LoRA (Low-Rank Adaptation) is a parameter-efficient technique introduced by Hu et al. [4] that factorizes weight updates into smaller matrices, drastically reducing trainable parameters during fine-tuning. Unlike full fine-tuning-which retrains all parameters—LoRA freezes the original pre-trained weights and injects trainable low-rank matrices alongside them, making fine-tuning far more memory-and compute-friendly.

The core idea of LoRA is represented in Fig.2, which illustrates how the technique decomposes weight updates. For a pre-trained weight matrix $W \in R^{d\times d}$, LoRA constrains its update by representing the change ΔW as a product of two low-rank matrices: $\Delta W = BA$, where $B \in R^{d\times r}$, $A \in R^{r\times d}$, and the rank $r \ll min\ (d, k)$. During the forward pass, both the frozen pre-trained weights W and the low-rank adaptation $\Delta W = BA$ are applied to the same input x, with their outputs being summed:

$$h = Wx + \Delta Wx = Wx + BAx \quad (1)$$

This approach allows the model to learn task-specific adaptations while keeping the original parameters frozen. Studies have shown that LoRA can achieve results comparable to full fine-tuning while updating only a fraction of the parameters [3], with GPT-3 experiments demonstrating up to a 10, 000× reduction in trainable parameters.

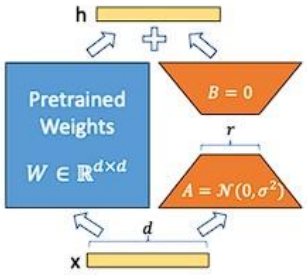

Figure 2: LoRA Reparameterization (Adapted From [4]): The figure shows how LoRA keeps the pre-trained weights W frozen while adding trainable low-rank matrices A and B. The output h is computed as the sum of the original pathway and the low-rank adaptation pathway.

Key Implementation Steps:

- Freeze the original model weights W
- Initialize A with random Gaussian weights and B with zeros, so $\Delta W = BA$ is zero at the beginning of training
- Choose an appropriate rank r (typically 4–32) that determines the inner dimension of matrices A and B
- Scale ΔW by α/r where α is a constant, helping to stabilize training across different rank values
- Apply LoRA selectively to specific weight matrices (often query and value matrices in attention blocks)
- Fine-tune only A and B, capturing the task-specific adaptations in a low-dimensional subspace
- After training, the low-rank update can be merged with the original weights ($W + BA$) for inference, introducing no additional latency

The rank r is a critical hyperparameter that balances parameter efficiency and model expressiveness. Lower ranks offer greater efficiency but potentially less expressivity, while higher ranks can capture more complex adaptations but require more parameters. Empirical studies suggest that surprisingly low ranks (even $r = 1$ or $r = 2$) can be sufficient for many downstream tasks, indicating that the weight updates during adaptation have an inherently low "intrinsic rank."

Benefits of LoRA:

- Efficiency: LoRA limits updates to small, low-rank matrices, substantially reducing trainable parameters and optimizer states.
- No Extra Latency: Once merged, the final weights look like any fully fine-tuned model.
- Flexible: Can be applied to different parts of the Transformer depending on parameter budget and performance needs.
- Scalable: Applicable across various model scales with consistent memory savings.
- Use Cases for LoRA:
- Transformer Attention Focus: When fine-tuning attention blocks suffices for high performance (e. g., classification, QA).
- Large Language Models: Where full fine-tuning is costly or infeasible.
- Resource-Constrained GPUs: Minimizes memory usage during training, suitable for smaller GPU setups.

Best Practices and Trade-offs: Optimize the rank r and scaling factor α to balance performance and resource usage. Target specific transformer components based on the task. Performance depends heavily on rank selection; an overly small rank can degrade accuracy, while a large rank reduces efficiency gains.

c) Prompt Tuning

Prompt Tuning is a parameter-efficient approach proposed by Lester et al. [5] that learns soft prompts—trainable embedding vectors—while keeping the large language model's main parameters frozen. Unlike traditional prompting (where discrete tokens are manually crafted), Prompt Tuning leverages backpropagation to update these soft prompt embeddings directly, conditioning the model on downstream tasks. This is particularly advantageous because it allows the model to retain its general language understanding, focusing adaptation on a small set of learned prompt parameters.

Simply put, instead of fine-tuning the entire model, you only learn a set of additional (or "virtual") tokens—a prompt—that is prepended (or appended) to the actual input. During training, only these prompt embeddings are adjusted, while the original model weights remain frozen. This approach can be particularly useful when you want to reduce compute, avoid overfitting on small datasets, or quickly adapt a large model to a new task.

Architecture:

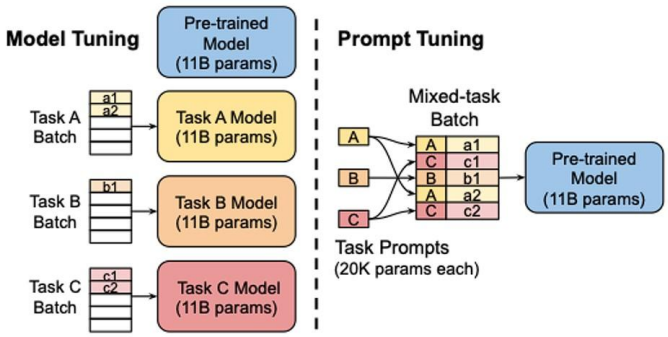

Figure 3: Prompt Tuning (Adapted From [5])

With Model Tuning, each new task duplicates the large pre-trained model's parameters, leading to huge storage and memory demands. By contrast, Prompt Tuning retains a single frozen backbone and learns only a small prompt embedding (20K parameters), reducing the trainable parameter count by several orders of magnitude. This also supports mixed-task inference, where multiple prompts can be batched together on a single forward pass.

Key Insights from Lester et al. [5]:

- Scalability: As model size increases (e. g., T5-XXL), Prompt Tuning's performance can approach full finetuning.
- Minimal Memory Footprint: Only a small prompt embedding is updated, lowering compute/storage costs for each downstream task.
- Domain Transfer & Robustness: Freezing the main model can reduce overfitting, improving zero-shot transfer to out-of-domain data.
- Prompt Ensembling: Multiple prompts can be combined in a single forward pass for further performance gains, without storing multiple large models.

Benefits of Prompt Tuning

- Minimal Memory Footprint: Only the prompt embeddings are trained, leaving the core model untouched.
- Adaptation: Quick to fine-tune and easy to swap prompts for different tasks or domains.
- Avoids Overfitting: Freezing the backbone helps preserve general language understanding, improving domain transfer.
- Scalable to Large Models: Performance gap to full finetuning narrows as model size grows.
- Use Cases for Prompt Tuning
- Rapid Prototyping: Quickly adapt a large LLM (e. g., GPT-3, T5-XXL) to new tasks.
- Low-Resource Settings: Effective when labeled data are limited, avoiding the overhead of full fine-tuning.
- Efficient Multi-Task Serving: Batching different prompts reduces latency.

Best Practices:

- Experiment with prompt initialization strategies (random, vocabulary-based, class-label-based).
- Use longer prompts in smaller models; short prompts can suffice in larger LLMs.
- LM-adaptive pre-training for encoder-decoder models (like T5) helps them better handle natural text prompts.
- Evaluate prompt ensembling for potential performance boosts.

Trade-offs:

- May struggle with complex tasks or large domain gaps if only the prompt is trainable.
- Performance is sensitive to prompt length and initialization; hyperparameter tuning may be required.
- Small models can have a bigger gap from fully fine-tuned performance; larger LLMs see greater relative benefit from prompt tuning.

d) Prefix Tuning

Prefix Tuning is a parameter-efficient technique introduced by Li and Liang [6] that adds a sequence of trainable, task-specific vectors (prefixes) to the hidden states of each Transformer layer, rather than just prepending tokens at the input embedding level. Unlike Prompt Tuning—which primarily modifies the initial input embeddings—Prefix Tuning influences multiple layers of the model, potentially leading to richer adaptations for complex tasks.

How Prefix Tuning Works

- Continuous Prefixes: Like Prompt Tuning, Prefix Tuning adds virtual tokens, but instead of appending them only at the input layer, these prefixes are applied across the hidden states in all Transformer blocks [7]. The large majority of the model's parameters remain frozen, with only these prefix vectors being updated during training.
- Layer-wise Modifications: At each layer, the Transformer can attend to these trainable prefixes, allowing for more expressive control. This yields an approach that is more flexible than Prompt Tuning, which primarily operates at the input embedding level.

- Sparse Updates: By freezing most of the model's weights and only tuning the prefix vectors, Prefix Tuning reduces computational overhead and avoids the need to maintain multiple fine-tuned copies of a large model.

Architecture:

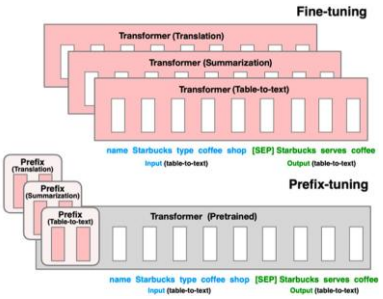

Figure 4: Prefix Tuning (Adapted from Li and Liang [6])

- Prefix vectors are inserted as learnable parameters at each Transformer layer.
- The main model parameters remain frozen.
- Only the prefix vectors are optimized to align the model with downstream tasks.

Benefits of Prefix Tuning:

- More Expressive than Prompt Tuning: Because it modifies activations across multiple Transformer layers, Prefix Tuning can capture more complex adaptations, potentially leading to better performance on tasks that demand deeper model adjustments.
- Combines Well with LoRA: The prefix approach can be used alongside LoRA, leveraging low-rank updates in attention while simultaneously controlling intermediate representations.
- Efficient Parameter Usage: Retains the majority of the pre-trained model's knowledge while updating only a small fraction of parameters.
- Strong for NLG Tasks: By influencing hidden states, Prefix Tuning is especially useful in natural language generation scenarios where controlling generation at multiple stages is advantageous.

Use Cases:

- Natural Language Generation: Tasks like summarization, story generation, or table-to-text often benefit from the richer control that Prefix Tuning provides.
- Resource-Constrained Scenarios: Similar to Prompt Tuning, Prefix Tuning can achieve strong performance with fewer trainable parameters than full fine-tuning.
- Complex Adaptations: Where small shifts in embedding-level prompts are insufficient, layer-wise prefix modifications offer more fine-grained control.

Best Practices:

- Optimize Prefix Length: Longer prefixes can offer more expressive capacity but risk overfitting or slowing training.
- Careful Initialization: Initializing prefix vectors to meaningful embeddings or random values can affect stability.
- Empirically, using domain-relevant tokens often speeds convergence.
- Combine with LoRA: For tasks requiring minimal memory usage, applying LoRA to specific attention weights while employing Prefix Tuning for layer control can deliver strong results.

Trade-offs:

- Reduced Sequence Length: Because the prefix tokens occupy space in the model's hidden states, the effective sequence length for original tokens can be lower.
- Potential Overhead in Multi-Layer Insertion: Maintaining prefixes at every layer can be slightly more complex compared to input-level prompt tuning.
- Hyperparameter Sensitivity: Finding the right prefix length and initialization strategy may require tuning, especially for more specialized domains.

3. GAPS in Existing Research

Although parameter-efficient fine-tuning (PEFT) has led to remarkable progress in adapting large language models (LLMs) with fewer resources, several challenges and open questions remain:

1) Inconsistent Reporting of Parameter Counts and Efficiency:

A recurring issue in PEFT studies is the inconsistency in reporting parameter counts. Some methods report only trainable parameters, whereas others cite changed parameters or low-rank dimensions. This inconsistency makes it difficult to compare methods fairly. Moreover, the relationship between the number of parameters and actual resource usage (e. g., GPU memory) is often non-linear. For instance, a method that introduces many new parameters in a reparameterization may still have low memory usage, whereas a sparse, selective method could introduce minimal changed parameters but require large gradients or masks for full model layers.

Recommended Direction: Researchers should explicitly report both trainable and changed parameters and detail memory usage, training time, and inference overhead, adhering to a standardized approach. Consistent metrics and clear definitions would help practitioners make better-informed decisions.

2) Lack of Standardized Metrics Beyond Accuracy:

While downstream performance is crucial, PEFT methods need to be evaluated along multiple axes, including training speed, peak memory usage, inference latency, and storage overhead for adapter modules or soft prompts. Many recent papers focus almost exclusively on accuracy (e. g., GLUE, SuperGLUE) but do not provide details about GPU memory consumption, training throughput, or inference speed, which can vary significantly across different parameter-efficient approaches.

Recommended Direction: A common benchmark that includes efficiency metrics—such as training time in tokens/second, inference speed, maximum VRAM usage, and final model storage size—would provide a more holistic view of method trade-offs.

3) Hyperparameter Sensitivity and Limited Tuning Protocols:

Several PEFT methods (e. g., Prefix Tuning, Prompt Tuning, hybrid approaches) often exhibit high sensitivity to hyperparameters like the prompt length, learning rate, and initialization scheme. Many impressive results rely on heavily tuned settings, which might not transfer well to real-world constraints (limited compute budgets, new domains). This gap is exacerbated in large-scale LLM fine-tuning, where hyperparameter sweeps can be cost-prohibitive.

Recommended Direction: Systematic investigations on hyperparameter robustness could yield heuristics or meta-tuning strategies for practitioners. Auto-tuning or adaptively choosing hyperparameters based on limited pilot runs (similar to Hyperband or Bayesian optimization) might also mitigate sensitivity concerns.

4) Model Scale and Generalization to Billion-Parameter+ Regimes:
Despite being labeled "large, " many PEFT experiments are performed on moderate-size Transformer backbones (e. g., BERT-base, T5-base). Yet, these methods often behave differently once the model surpasses the billion-parameter scale. For instance, some methods become more parameter-efficient as model size grows, while others run into memory bottlenecks or instability. This discrepancy indicates a need for studying PEFT at truly large scales (e. g., 20B, 175B) to capture real-world usage patterns.

Recommended Direction: Evaluate new PEFT methods on diverse model sizes (e. g., smaller than 1B, 3B–20B, larger than 20B) and analyze scalability and robustness. Results at smaller scales should be clearly distinguished from those at billion-parameter scales.

5) Comparison Across Hybrid or Combined Methods:
Although hybrid PEFT approaches (e. g., combining LoRA with Adapters or Prompt Tuning) can leverage complementary strengths, there is limited empirical exploration of which combinations work best under different constraints. Some studies do propose multicomponent solutions (e. g., MAM Adapters, UniPELT), but they can be memory-intensive or hyperparameter-heavy to the point that real-world adoption is hindered.

Recommended Direction: More thorough head-to-head comparisons and ablation studies of multi-component PEFT recipes. Clarifying how each sub-component influences memory and compute overhead could guide modular adoption in production scenarios.

6) Reporting Challenges and Reproducibility:
In some open-source PEFT implementations, users must install custom forks of Transformers libraries, with minimal documentation on changes to code paths. This complicates reproducibility and discourages consistent baselines. As a result, many "new" PEFT techniques are not directly comparable because they were tested in distinct frameworks or rely on non-standard data preprocessing.

Recommended Direction: Encourage shared code repositories and well-documented integration into popular libraries (e. g., HuggingFace PEFT, AdapterHub). A multitask PEFT benchmark suite, accompanied by scripts for standard data splits and metric logging, would foster transparency and direct comparisons.

7) Broader Ethical and Long-Term Implications:
Efficiency-focused methods could have unintended consequences. For instance, constant freezing of large backbones might inhibit necessary updates for bias reduction or domain fairness. Similarly, minimal adapters or prompts might not rectify harmful stereotypes ingrained in the pre-trained model. Empirical work on catastrophic forgetting often lacks rigorous analysis of potential fairness or safety considerations for real-world deployment.

Recommended Direction: Investigate bias, fairness, and safety under the lens of PEFT, ensuring that parameter efficiency does not come at the cost of ethical compromises. Future research should measure and mitigate bias shifts that can arise from partial or modular fine-tuning.

In summary, while PEFT methods—such as Adapters, LoRA, Prompt/Prefix Tuning, and hybrids—offer promising avenues to reduce cost and memory footprints, standardization in reporting, robust hyperparameter protocols, and scalable evaluations are critical next steps. Addressing these gaps will enable a more reliable comparison of techniques and help practitioners choose the most suitable approach for their domain and resource constraints.

4. Implementation and Demonstration of PEFT Methods

A. Experimental Setup

We implemented and compared four popular parameter-efficient fine-tuning (PEFT) techniques on a biomedical question-answering classification task using the PubMedQA dataset. This section details our experimental setup, implementation specifics, and execution environment.

Dataset: The PubMedQA dataset's labeled subset (pqa_labeled) contains 1, 000 expert-annotated biomedical question-answer pairs. Each sample includes a natural language question, a supporting PubMed abstract as context, and an expert-labeled answer (yes, no, or maybe). We split this dataset into training 80% and validation 20% sets.

Model Architecture: We utilized a pre-trained BERT-base-uncased model (110M parameters) as our foundation, adding a classification head for the 3-class classification task. The model was implemented using the Hugging Face Transformers library.

Computing Environment: Experiments were executed on a cloud instance with the following specifications:

- GPU: 1× NVIDIA RTX A5000
- vCPUs: 12
- System Memory: 25 GB RAM
- Storage: 20 GB
- OS: Ubuntu 22.04 with CUDA 11.8.0
- Framework: PyTorch 2.1.0

Implementation Framework: We utilized Hugging Face's Transformers (v4.48.0) and PEFT (v0.7.1) libraries for model implementation, with the following additional libraries:

- Accelerate (v0.26.0) for training optimization

- Datasets (v2.14.5) for data processing
- Evaluate (v0.4.1) for metrics calculation
- Weights & Biases for experiment tracking

Common Training Configuration: For consistent comparison, all methods used identical training parameters:

- Maximum sequence length: 512 tokens (adjusted as needed for prompt-based methods)
- Batch size: 8
- Learning rate: 5e-5
- Weight decay: 0.01
- Training epochs: 3
- Optimization: AdamW optimizer
- Mixed precision (FP16): Enabled
- Random seed: 42

B. PEFT Method Implementations

1) Adapters: Adapters insert small trainable bottleneck layers between the existing layers of a pre-trained model. We implemented adapters using the Pfeiffer configuration:

```
adapter_config = ADAPTER_CONFIG_MAP ["pfeiffer"]
model. add_adapter ("pubmedqa_adapter",
config=adapter_config)
model. train_adapter ("pubmedqa_adapter")
model. set_active_adapters ("pubmedqa_adapter")
```

The Pfeiffer adapter architecture adds bottleneck layers with a reduction factor of 16 after the attention and feedforward blocks in each transformer layer. This resulted in 2.1M trainable parameters (1.89% of total parameters).

2) Low-Rank Adaptation (LoRA): LoRA applies low-rank decomposition to weight matrices in the model, particularly in the attention mechanism:

```
lora_config = LoraConfig (
 r=8, # Rank of update matrices
 lora_alpha=16, # Scaling factor
 target_modules= ["query", "value"], #
 Apply to specific matrices
 lora_dropout=0.1, # Regularization
 bias="none",
 task_type=TaskType. SEQ_CLS
)
 model = get_peft_model (model,
 lora_config)
```

Our implementation used a rank of 8 and targeted the query and value matrices in the attention layers, resulting in 0.297M trainable parameters (0.27% of total parameters).

3) Prompt Tuning: Prompt Tuning prepends trainable "soft prompts" to the input while keeping the model frozen:

```
prompt_config = PromptTuningConfig (
 task_type=TaskType. SEQ_CLS,
 num_virtual_tokens=10,
 tokenizer_name_or_path="bert-base-
 uncased",
 prompt_tuning_init="RANDOM"
model = get_peft_model (model,
 prompt_config)
```

We implemented prompt tuning with 10 virtual tokens, resulting in just 9, 987 trainable parameters (0.009% of total parameters). To accommodate these virtual tokens, we reduced the maximum sequence length for actual tokens to 502.

Note that we increased the training epochs to 10 for Prompt Tuning (compared to 3 epochs for other methods) to give this lightweight approach more opportunity to learn. Despite the additional training time, Prompt Tuning still converged to predicting only the majority class, achieving the same 51.5% accuracy as LoRA and Prefix Tuning.

4) Prefix Tuning: Prefix Tuning adds trainable vectors to the hidden states at each layer of the model:

```
prefix_config = PrefixTuningConfig (
 task_type=TaskType. SEQ_CLS,
 num_virtual_tokens=20,
 prefix_projection=True,
 encoder_hidden_size=768
)
model = get_peft_model (model,
 prefix_config)
```

Our implementation used 20 virtual tokens with a projection layer, resulting in 14.8M trainable parameters (11.9% of total parameters). We reduced the maximum sequence length to 492 to accommodate the prefix tokens.

5. Result and Analysis

A. Performance Comparison

Table I summarizes the performance metrics, parameter efficiency, and training characteristics of the four PEFT methods:

Table I: Comparison of PEFT Methods on PUBMEDQA

Method	Accuracy	F1 Score	Trainable Param-eters	Parameter Efficiency	Class Discriminat-ion
Adapter (Pfeiffer)	79.0%	0.79	2.1M	1.89%	Strong
LoRA	51.5%	0.23	0.297M	0.27%	Poor
Prompt Tuning	51.5%	0.23	9, 987	0.009%	None
Prefix Tuning	51.5%	0.23	14.8M	11.9%	None

B. Detailed Analysis

Performance Across Methods: The adapter approach significantly outperformed other PEFT methods, achieving 79% accuracy with good discrimination between classes. In contrast, LoRA, Prompt Tuning, and Prefix Tuning all converged to predicting only the majority class ("yes"), resulting in identical performance metrics (51.5% accuracy and 0.23 F1 score).

Classification Behavior: The classification reports revealed distinct behavior patterns:

Adapters: Showed balanced precision and recall across all classes

	Precision	Recall	F1-Score	Support
Yes	0.82	0.87	0.84	113
No	0.77	0.70	0.73	67
Maybe	0.65	0.65	0.65	20

LoRA, Prompt Tuning, Prefix Tuning: All displayed the same pattern of predicting only the majority class

	Precision	Recall	F1-Score	Support
Yes	0.52	1.00	0.68	103
No	0.00	0.00	0.00	72
Maybe	0.00	0.00	0.00	25

Parameter Efficiency: Prompt Tuning achieved the highest parameter efficiency, requiring only 0.009% of the model's parameters, while Prefix Tuning unexpectedly had the lowest efficiency at 11.9%. LoRA demonstrated excellent efficiency at 0.3% while Adapters used a moderate 1.89% of parameters. Training Dynamics: All methods showed unusual training loss patterns, with relatively flat losses across epochs. This is particularly notable for LoRA, Prompt Tuning, and Prefix Tuning, which all converged to predicting only the majority class despite having different architectures and parameter counts.

The poor performance of LoRA, Prompt Tuning, and Prefix Tuning on the PubMedQA task likely stems from their limited architectural modifications. While these methods are parameter-efficient, they may lack the computational capacity needed for complex biomedical question-answering. Adapters, by adding new neural network layers with non-linear transformations, create additional processing pathways that can learn task-specific patterns. In contrast, LoRA's low-rank updates, Prompt Tuning's input-only modifications, and Prefix Tuning's attention-only adjustments might be too constrained to capture the nuanced relationships between medical questions and scientific abstracts, causing them to default to the simplest solution: majority class prediction.

C. Discussion

Our results highlight several important insights about PEFT methods when applied to biomedical classification tasks:

- Method Selection Criticality: The choice of PEFT method dramatically impacts performance, with Adapters clearly superior for this particular task despite using more parameters than some alternatives.
- Task Compatibility: Not all PEFT methods are equally suited to all tasks. The poor performance of prompt-based methods suggests they may require task-specific optimization or may simply be less effective for classification with BERT-like models.
- Efficiency-Performance Trade-off: The most parameter-efficient method (Prompt Tuning) performed poorly, while the moderately efficient Adapter approach achieved the best results. This suggests that some minimum level of parameter flexibility may be necessary for effective learning.
- Implementation Challenges: Both Prompt Tuning and Prefix Tuning required special handling of sequence lengths to accommodate virtual tokens, adding implementation complexity.
- Domain-Specific Considerations: The biomedical domain's specialized vocabulary and complex reasoning requirements may favor adapter-based approaches that retain more expressive power at key layers in the network.

Beyond just the number of trainable parameters, it's crucial for practitioners to consider the broader computational overhead. The reduced number of trainable parameters in PEFT methods, as shown in Table I, directly translates to lower GPU memory consumption during training compared to traditional full fine-tuning. This is a significant advantage for environments with limited hardware resources. Furthermore, fewer trainable parameters generally mean less computation during the backpropagation step, which can lead to faster training times per epoch, even if the total training time might vary based on convergence speed. For inference, the impact varies. Methods like LoRA offer the significant advantage of being mergeable into the original model weights after training, resulting in no additional inference latency. Adapter-based methods, on the other hand, introduce small additional layers, which might add a minor overhead during inference, although this is often negligible compared to the efficiency gains during training. Prompt and Prefix Tuning operate by modifying input or hidden states, and while they avoid modifying the core model weights, their impact on inference latency can depend on the specific implementation and how the prefixes/prompts are handled. When selecting a PEFT method, practitioners should evaluate not just parameter count but also measure training speed and inference latency in their target deployment environment.

D. Limitations and Future Work

Several limitations should be noted:

- Dataset Size: The PubMedQA labeled subset contains only 1, 000 examples, which may be insufficient for prompt-based methods that typically benefit from larger datasets.
- Hyperparameter Optimization: We used consistent hyperparameters across methods to ensure fair comparison, but method-specific tuning might improve performance, particularly for the underperforming methods.
- Additional PEFT Methods: Newer methods such as IA^3, BitFit, and UniPELT were not evaluated and could potentially offer different performance characteristics.
- Future work could explore:
- Combining methods (e. g., LoRA + Adapters) for potentially better performance.
- Extending evaluations to larger biomedical datasets.
- Investigating the impact of model scale on PEFT method effectiveness.
- Developing specialized PEFT approaches for biomedical classification tasks.

The implementation code for all four PEFT methods used in this study is available at [https: //github. com/chandantroughia/PEFT], providing researchers with a practical starting point for their own experiments with parameter-efficient fine-tuning methods.

6. Real World Applications

Parameter-efficient fine-tuning methods have rapidly gained adoption across multiple industries, enabling organizations to customize large language models for domain-specific applications while managing computational resources effectively. Several key sectors are leading the implementation of these techniques:

A. Healthcare and Biomedical Applications

The healthcare industry has embraced PEFT methods to adapt language models for specialized medical tasks:

Clinical Decision Support: Research by Christophe et al. (2024) demonstrated that while full-parameter fine-tuning achieved better performance, parameter-efficient methods like LoRA yielded remarkably close results when adapting large language models for medical reasoning and diagnostic tasks, including on the USMLE medical licensing exam where their Med42 model achieved 72% accuracy [8].

Biomedical Information Extraction: Dagdelen et al. (2024) showed how large language models could be fine-tuned to extract structured information from scientific texts, with applications in materials chemistry demonstrating effective extraction of complex scientific knowledge while maintaining efficiency in computational resources [9].

Hospital Discharge Summarization: Parameter-efficient fine-tuning using QLoRA (Quantized Low-Rank Adaptation) has been applied to summarize hospital discharge papers, enabling accurate interpretation of medical terminologies and contexts while significantly reducing memory usage compared to full fine-tuning, as demonstrated by Prajapati et al. (2024) [10].

B. Financial Services

The financial sector leverages PEFT methods to enhance analytical capabilities:

Financial Sentiment Analysis: Research by Zhang et al. (2023) demonstrated that instruction-tuned models using parameter-efficient fine-tuning outperformed state-of-the-art supervised sentiment analysis models on financial text, particularly when handling numerical values and financial contexts crucial for market analysis [11].

Financial Documentation Processing: A study on FinLoRA by Wang et al. (2025) showed that parameter-efficient fine-tuning with quantized low-rank adaptation (QLoRA) significantly reduced GPU memory requirements and improved training efficiency while maintaining high model performance for processing financial regulatory documentation and extracting relevant information [12].

Portfolio Management: Research by Konstantinidis et al. (2024) demonstrated that LoRA-based fine-tuning applied to financial large language models reduced model training time significantly while maintaining comparable performance on financial sentiment analysis tasks, using only 0.06% of the total trainable parameters (4.2 million parameters) to enhance portfolio management and risk assessment capabilities [13].

C. Customer Service and Enterprise Applications

Organizations across industries are deploying PEFT methods to enhance customer interactions with significant measurable benefits. The implementation of these techniques is transforming how businesses handle customer engagement while substantially reducing computational costs.

Domain-Specific Chatbots: Enterprise implementations of PEFT methods in customer service have shown remarkable efficiency gains. A 2024 study documented that AI-powered chatbots can handle up to 70% of routine customer inquiries, allowing human agents to focus on complex issues [14]. This reduction in manual workload translates to lower operational costs as businesses can manage their customer service operations with fewer resources.

Multilingual Support: Parameter-efficient fine-tuning enables organizations to efficiently adapt foundation models for multilingual customer support applications. Research has shown that PEFT methods like Semantic Knowledge Tuning (SK-Tuning) can achieve competitive or superior performance compared to full fine-tuning while utilizing significantly fewer parameters [15]. By reducing the number of trainable parameters from hundreds of millions to just a few million, these approaches allow companies to deploy specialized models for different languages without the computational overhead of maintaining fully fine-tuned models for each language, making multilingual support more accessible and cost-effective.

Enterprise Knowledge Management: A 2024 Together. ai case study on Multi-LoRA demonstrated how enterprise teams can deploy task-specific adapters for functions ranging from customer service automation to documentation management—all using a shared base model [16]. This approach eliminated the need for separate model deployments while improving information retrieval quality and allowing for more efficient resource utilization.

D. Emerging Applications

Beyond established use cases, PEFT methods are enabling innovation in emerging areas:

Resource-Constrained Devices: Recent research by Ding et al. (2024) demonstrates that LoRA approaches can be optimized for IoT and edge devices with limited computational resources, enabling efficient model adaptation without compromising performance [17].

Personalized AI Experiences: Tan et al. (2024) introduced "One PEFT Per User" (OPPU), a personalization framework that equips each user with their own parameter-efficient fine-tuning module. This approach significantly outperformed existing prompt-based methods across diverse tasks while requiring only 0.5% of the trainable parameters compared to full model fine-tuning [18].

Rapid Domain Adaptation: The Hugging Face PEFT library (2023) provides implementations of LoRA and other parameter-efficient techniques that enable practitioners to adapt models to new domains in hours rather than days, while maintaining over 95% of the performance of full fine-tuning approaches [19].

These real-world applications demonstrate that PEFT methods are not merely theoretical innovations but practical solutions addressing genuine industry needs for efficient, domain-specific AI deployment.

7. Best Practices and Trade-Offs

Successfully applying Parameter-Efficient Fine-Tuning methods in practice often requires careful consideration and tuning of hyperparameters. While PEFT significantly reduces the search space compared to full fine-tuning, parameters specific to each PEFT technique (such as LoRA rank, prompt length, or adapter reduction factor) can significantly influence performance. The best practices and trade-offs discussed below provide a valuable starting point, but practitioners should anticipate the need for some level of hyperparameter experimentation to achieve optimal results for their specific task and dataset.

PEFT Methods	Best Practices	Trade-offs
Adapters	• Use reduction factor of 16 for balanced efficiency and implement Adapter Fusion for multi-task learning • Place adapters strategically in transformer layers	• Introduces architectural complexity and higher inference latency compared to the base model • Requires additional model-parameters, increasing model size
LoRA	• Optimize rank parameter (r=8 recommended starting point) and target specific matrices (query and value for attention) • Consider applying to different layers based on task	• Performance depends heavily on rank selection and requires weight merging for deployment • Lower rank may significantly degrade results
Prompt Tuning	• Experiment with prompt initialization strategies, use longer prompts for complex tasks, and works best with larger models (larger than 1B parameters)	• May struggle with complex tasks, performance highly dependent on model size, and limited effectiveness with smaller models
Prefix Tuning	• Optimize prefix length based on task complexity and use reparameterization for better stability	• Consider combining with LoRA for better results • Reduces usable sequence length, can be surprisingly parameter-intensive, and introduces implementation complexity with attention masks

8. Future Research Directions

While PEFT methods have advanced significantly, several promising research directions could further enhance their effectiveness and applicability. Based on our findings and identified gaps in current research, we propose the following areas for future investigation:

A. Hybrid PEFT Approaches

Current research predominantly evaluates individual PEFT methods in isolation, yet preliminary evidence suggests that combining techniques may yield superior results. Future research should systematically investigate:

- Integration of complementary methods (e. g., LoRA with Prefix Tuning) to leverage their respective strengths while minimizing weaknesses [3].
- Development of adaptive frameworks that dynamically select optimal PEFT techniques based on task characteristics and resource constraints, expanding on work by Li et al. [6].
- Quantification of efficiency-performance trade-offs in hybrid approaches across diverse model architectures and sizes [3, 6].

B. Cross-Modal and Multi-Domain Applications

Most PEFT research has focused on natural language processing, leaving significant opportunities to explore:

- Adaptation of PEFT techniques for vision transformers, audio models, and multi-modal architectures, building on preliminary work by Wang et al. [3].
- Domain-specific optimizations for healthcare, finance, legal, and scientific applications [8, 11].
- Transfer learning capabilities across diverse domains with minimal parameter updates [1, 4].
- Evaluation of PEFT methods in robotics and embodied AI, where resource constraints are often stringent.

C. Ethical Considerations and Bias Mitigation

As PEFT methods become more widely adopted, research must address:

- Impact of parameter-efficient adaptation on inherited biases from pre-trained models.
- Development of PEFT variations specifically designed to reduce harmful biases while maintaining task performance.
- Comparative analysis of how different PEFT approaches affect model fairness and safety.
- Transparency frameworks for documenting adaptations and their potential effects on model behavior.

D. Resource-Constrained Environments

Further research is needed to optimize PEFT methods for deployment in settings with limited computational resources:

- Quantization-aware PEFT techniques that maintain compatibility with low-precision inference.
- Auto-scaling approaches that adjust parameter efficiency based on available resources.
- Edge-specific adaptations that minimize memory footprint and power consumption.
- Differential updating strategies that prioritize parameter modifications based on task-specific importance.

E. Standardized Evaluation Frameworks

To address inconsistencies in reporting and enable fair comparisons:

- Development of comprehensive benchmarks evaluating accuracy, memory usage, training speed, and inference latency.

- Standardized protocols for hyperparameter selection and optimization.
- Long-term studies on stability and catastrophic forgetting in PEFT-adapted models.
- Metrics for quantifying the robustness of PEFT methods across different domains and tasks [3, 5].

These research directions address current limitations in PEFT methodologies while expanding their applicability to new domains and deployment scenarios. As large language models continue to grow in size and capability, parameter-efficient fine-tuning approaches will become increasingly critical for democratizing access to state-of-the-art AI technologies.

9. Conclusion

Parameter-efficient fine-tuning (PEFT) techniques represent a significant advancement in the adaptation of large language models, offering computationally viable alternatives to traditional full fine-tuning approaches. This paper has provided a comprehensive analysis of four prominent PEFT strategies—Adapters, Low-Rank Adaptation (LoRA), Prompt Tuning, and Prefix Tuning—examining their architectural designs, implementation details, and empirical performance on a biomedical classification task.

Our comparative evaluation revealed considerable differences in parameter efficiency and effectiveness across methods. While Adapter-based approaches demonstrated superior performance for our classification task, achieving 79% accuracy with only 1.89% of trainable parameters, other methods showed limitations when applied to smaller models and specialized domains. These findings underscore the importance of method selection based on specific use cases, available computational resources, and task requirements.
The best practices and trade-offs we identified highlight that PEFT methods are not universally interchangeable—each offers distinct advantages for particular applications. Adapters provide robust performance with modular design benefits, LoRA offers excellent parameter efficiency with minimal inference overhead, while prompt-based methods potentially excel with larger models despite struggling in our experiments with a smaller model.

As large language models continue to grow in size and capability, the significance of parameter-efficient adaptation will only increase. Future research should focus on developing hybrid approaches that combine the strengths of multiple PEFT methods, expanding applications beyond NLP to other domains, addressing ethical considerations including bias mitigation, and optimizing for resource-constrained environments. Standardized evaluation frameworks will be crucial for fair comparison across the rapidly evolving landscape of parameter-efficient fine-tuning techniques.
The insights and implementation guidelines presented in this paper provide practitioners with practical knowledge for deploying PEFT methods effectively in real-world scenarios, contributing to more accessible, efficient, and sustainable adaptation of large language models across diverse applications and domains.

References

[1] https: //medium. com/[at]ysfckmk/parameter-efficient fine-tuning-peft-for-large-language-models-making-ai more-adaptable-and-52fb0395af2a
[2] N. Houlsby, et al., “Parameter-Efficient Transfer Learning for NLP, ” ICML, 2019. [Online]. Available: https: //arxiv. org/abs/1902. 00751
[3] L. Wang, S. Chen, L. Jiang, S. Pan, R. Cai, S. Yang, F. Yang, “Parameter-Efficient Fine-Tuning in Large Models: A Survey of Methodologies, ” arXiv preprint arXiv: 2410.19878 (v2), 2024. [Online]. Available: https: //doi. org/10. 48550/arXiv. 2410. 19878
[4] E. J. Hu, et al., “LoRA: Low-Rank Adaptation of Large Language Models, ” ICLR, 2022. [Online]. Available: https: //arxiv. org/abs/2106. 09685.
[5] Bs. Lester, et al., “The Power of Scale for Parameter Efficient Prompt Tuning, ” EMNLP, 2021. [Online]. Available: https: //arxiv. org/abs/2104. 08691.
[6] X. Li and P. Liang, “Prefix-Tuning: Optimizing Continuous Prompts for Generation, ” ACL, 2021. [Online]. Available: https: //arxiv. org/abs/2101. 00190
[7] Parameter Efficient Fine Tuning. Adapters; LoRA; QLora-Medium. Accessed February 10, 2025. [Online]. Available: https: //medium. com/aimonks/parameterefficient-fine-tuning-075954d1db51
[8] C. Christophe, P. K. Kanithi, P. Munjal, et al., “Med42 – Evaluating Fine-Tuning Strategies for Medical LLMs: Full-Parameter vs. Parameter-Efficient Approaches, ” arXiv preprint arXiv: 2404.14779, 2024. [Online]. Available: https: //arxiv. org/abs/2404. 14779
[9] J. Dagdelen, A. Dunn, S. Lee, et al., “Structured information extraction from scientific text with large language models, ” Nature Communications, vol.15, no.1, p.1418, 2024. [Online]. Available: https: // www. nature. com/articles/s41467-024-45563-x
[10] K. K. Prajapati, A. Saha, A. K. Saha, and J. Goswami, “Parameter-efficient fine-tuning large lan-guage model approach for hospital discharge paper summarization, ” Applied Soft Computing, 2024. [Online]. Available: https: //www. sciencedirect. com/science/ article/abs/pii/S1568494624003053
[11] B. Zhang, H. Yang, and X. Liu, “Instruct-FinGPT: Financial Sentiment Analysis by Instruction Tuning of General-Purpose Large Language Models, ” arXiv preprint arXiv: 2306.12659, 2023. [Online]. Available: https: //arxiv. org/abs/2306. 12659
[12] D. Wang, et al., “FinLoRA: Fine-tuning Quantized Financial Large Language Models Using Low-Rank Adaptation, ” arXiv preprint arXiv: 2412.11378, 2025. [Online]. Available: https: //arxiv. org/abs/2412. 11378
[13] T. Konstantinidis, G. Iacovides, M. Xu, T. G. Constantinides, and D. Mandic, “FinLlama: Financial Sentiment Classification for Algorithmic Trading Applications, ” arXiv preprint arXiv: 2403.12285, 2024. [Online]. Available: https: //arxiv. org/abs/2403. 12285
[14] ResearchGate. (2024). “Leveraging AI-Powered chatbots to enhance customer service efficiency and future opportunities in automated support.” ResearchGate, 385230161. [Online]. Available: https: //www. researchgate. net/publication/385230161 Leveraging AI-Powered chatbots to enhance _ customer service efficiency and future opportunities in automated support

[15] Scientific Reports. (2024). "Parameter-efficient finetuning of large language models using semantic knowledge tuning. " Nature. [Online]. Available: https: // www. nature. com/articles/s41598-024-75599-4
[16] Together. ai. (2024). "Announcing Serverless MultiLoRA: Fine-tune and deploy hundreds of adapters for model customization at scale. " Together. ai Blog. [Online]. Available: https: //www. together. ai/blog/ serverless-multi-lora-fine-tune-and-deploy-hundredsof-adapters-for-model-customization-at-scale
[17] C. Ding, Y. Wang, T. Zhang, X. Hei, Y. Jin, and X. Yang, "LoRA-C: Parameter-Efficient Fine-Tuning of Robust CNN for IoT Devices, " arXiv preprint arXiv: 2410.16954, 2024. [Online]. Available: https: // arxiv. org/abs/2410. 16954
[18] Z. Tan, Q. Zeng, Y. Tian, Z. Liu, B. Yin, and M. Jiang, "Democratizing Large Language Models via Personalized Parameter-Efficient Fine-tuning, " arXiv preprint arXiv: 2402.04401, 2024. [Online]. Available: https: // arxiv. org/abs/2402. 04401
[19] Hugging Face, "PEFT: Parameter-Efficient Fine-Tuning of Large Language Models, " GitHub Repository, 2023. [Online]. Available: https: //github. com/huggingface/peft
[20] Adapter documentation and implementation guidelines from HuggingFace PEFT library. [Online]. Available: https: //huggingface. co/docs/peft/main/en/ conceptual guides/adapter