University *of Ljubljana*
Faculty *of Computer and Information Science*

# Parameter-Efficient Fine-Tuning of Large Language Models

Ondřej Komín, Andrej Sušnik, Eileen Vu

**Abstract**

This is the abstract

**Keywords**
Keyword1, Keyword2, Keyword3 ...

*Advisors: Slavko Žitnik*

## Introduction

Since the introduction of attention [1], using large language models (LLMs) such as Google's BERT [2] and OpenAI's GPT [3] has become inevitable to various applications across natural language processing (NLP) domains. These models have demonstrated remarkable capabilities in understanding and generating human-like text. However, to achieve optimal performance in specific tasks, fine-tuning these pre-trained models on task-specific data is often necessary.

This research paper focuses on presenting and comparing various parameter-efficient fine tuning (PEFT) methods in the context of optimizing natural language processing tasks. Through empirical experiments, we aim at exploring the effectiveness of different PEFT approaches in achieving desirable trade-offs between model complexity, computational efficiency, and task performance. Similarly to the work done in [4], we begin our research by reviewing and categorizing popular PEFT methods. We continue by presenting and discussing the theoretical foundations of three specific methodologies and applying these to five different datasets that cover various natural language understanding skills. Lastly, the fine-tuned models will be evaluated based on appropriate performance metrics, computational resources required, and ease of adaptation to different tasks. Through this exploration, we aim at facilitating more resource-conscious approaches to model optimization, accelerating progress in the field of NLP and its applications.

## Related Work

Fine-tuning LLMs plays a crucial role in adapting these models to domain-specific tasks. As LLMs are pre-trained on vast amounts of text data, they capture linguistic patterns and semantic information. However, for tasks with specific requirements, such as sentiment analysis or named entity recognition, fine-tuning allows these models to tailor their representations to better suit the task at hand.

Traditional methods of full fine-tuning involve updating all parameters of the pre-trained LLM. The popular GPT-4 model released in early 2024 contains 1.76 trillion weights [5]. Fine-tuning (and storing) this amount of parameters whenever one wants to apply the model to a specific use case however not only requires significant computational resources but also poses a risk of overfitting, especially in scenarios with limited task-specific data.

In order to avoid these problems, PEFT methods have emerged as a solution to the drawbacks of full fine-tuning. These methods aim to optimize neural networks with fewer parameters while maintaining comparable performance to traditional fine-tuning approaches. By reducing the number of parameters updated during fine-tuning, PEFT not only mitigates computational costs but also helps alleviate overfitting concerns. As described in [4], PEFT techniques can be divided into five main categories: additive fine-tuning, partial fine-tuning, reparameterized fine-tuning, hybrid fine-tuning and lastly unified fine-tuning. While some of these methods aim at introducing new trainable parameters for use-case-specific fine-tuning, others reduce the number of trainable parameters by transforming the weights into lower dimensions.

In this paper, we focus on presenting and comparing three PEFT methods in the context of different NLP tasks. Specifically, we investigate the performance of the following methodologies: low rank adaptation (LoRA), soft prompt-based fine-tuning, and partial fine-tuning. **LoRA** [6] has become very popular in the last years due to its ability to reduce the number of parameters without introducing additional latency, unlike for example adapter methods. This lowers training computational requirements while improving performance for specific

NLP tasks. **Soft-prompting** [7] is a machine learning technique that offers subtle guidance to models during training, aiding in learning without the need for explicit labels. This approach is valuable as it allows for more flexible decision-making while still achieving desired outcomes, especially in scenarios where labeled data may be scarce or costly to obtain. Lastly, we will investigate the partial fine-tuning method **BitFit**[8]. This method only fine-tunes the bias term of the layers while freezing the rest of the network. This technique, which trains less than 0.1% of the total number of weights, was proven to achieve comparable performance than full fine-tuning.

| Benchmark | NLP Task |
|---|---|
| CommonsenseQA [9] | Commonsense Reasoning |
| CoNLL-2012 [10] | Coreference Resolution |
| XSum [11] | Text Summarization |
| SST5 [12] | Sentiment Analysis |
| Slovene SuperGLUE [13] | Slovene BoolQ (Boolean Questions) |

**Table 1.** Chosen benchmarks for performance evaluation

We will provide an empirical comparison of these three methodologies based on five different NLP tasks. The benchmarks we have chosen for this each represent distinct natural language understanding skills allowing us to provide a comprehensive overview of the advantages and disadvantages of all techniques.

## Methods

PEFT methods can be grouped into five main categories. Additive fine-tuning introduces new trainable parameters for task-specific adaptation, including adapter-based fine-tuning, soft prompt-based fine-tuning, and others. Partial fine-tuning reduces the number of fine-tuned parameters by focusing on critical pre-trained parameters, with methods like bias update, pretrained weight masking, and delta weight masking. Reparameterized fine-tuning utilizes low-rank transformation to decrease trainable parameters, through techniques like low-rank decomposition and LoRA derivatives. Hybrid fine-tuning combines multiple PEFT approaches to leverage strengths and mitigate weaknesses, either manually or automatically. Unified fine-tuning provides streamlined frameworks for incorporating diverse fine-tuning methods into cohesive architectures, emphasizing consistency and efficiency across model adaptation without combining multiple methods.

In this analysis, we will focus on three of these methods:

**BitFit** summarization of bitfit

**LoRA** LoRA [6] is a method designed for fine-tuning large models. It operates by fixing the weights of the original model and introducing a trainable low-rank decomposition matrix into the LLM architecture. This modified architecture involves fixing the original weights $W_0$ while introducing additional trainable weights $\Delta W$, which can be decomposed into matrices $BA$, where the rank of both $B$ and $A$ is much smaller than that of $W_0$. Consequently, the resulting weights are formulated as $W0 + BA$, significantly reducing the number of parameters that need to be trained.

Empirical results indicate that LoRA performs comparably or even better than other methods, while requiring a comparable or lower number of trainable parameters. Notably, LoRA drastically reduces the number of trainable parameters, such as in the case of fine-tuning the GPT-3 model, where the parameter count was reduced by 10000 times, accompanied by a threefold reduction in GPU memory requirement. Additionally, LoRA exhibits several benefits, including the ability to train specialized models without introducing latency, as seen in adapter methods.

Moreover, it enables the deployment of multiple specialized models simultaneously by reducing memory and computational footprint, achieved through maintaining fixed weights for the base model while having several trained decomposition matrices for each specialized task.

LoRa model is trained with configuration showed in Listing 1.

**Listing 1.** LoRa parameters

```
lora_config = LoraConfig(
        r=16,
        lora_alpha=32,
        lora_dropout=0.1,
        bias="lora_only",
        task_type="SEQ_CLS"
)
```

**Prompt Tuning** summarization of prompt tuning

## Training Pipeline

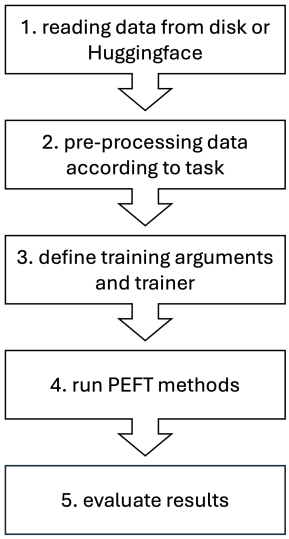


**Figure 1.** Training Pipeline

The training pipeline consists of five main steps, each

applicable for the successful fine-tuning of the models for specific tasks.

**Step 1** Initially, we imports necessary libraries and set up the environment. Using custom dataset handlers, we load the datasets based on the specified task.

**Step 2** After loading the dataset, the we apply pre-processing functions tailored to the BERT tokenizer. This step involves tokenizing and encoding the text data, which is essential for input to the BERT model. The pre-processed data is split into train, validation, and test sets, and formatted as PyTorch tensors for compatibility with the BERT model.

**Step 3** Next, we configure the training arguments, including parameters such as output directory, evaluation strategy, learning rate, batch size, and number of epochs. It also initializes the BERT model for the task and defines the trainers for the different methods. Each trainer is associated with its specific model path and training configurations.

**Listing 2.** Training Arguments

```
args = TrainingArguments(
    output_dir=model_path,
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
    auto_find_batch_size=True,
    num_train_epochs=20,
    weight_decay=0.01,
)
```

**Step 4** With the trainers defined, we execute the training process for all PEFT methods. This step involves fine-tuning the BERT model on the data using the specified training arguments and data.

**Step 5** Finally, we evaluate the trained models using the evaluation datasets. Performance metrics such as accuracy, precision, F1-score, and recall are computed for the different methods. The results are compared to assess the effectiveness of each fine-tuning approach on specific tast.

## Environment and reproducibility

We ran the experiments on the Arnes cluster, the compute node that we used included AMD EPYC™ 9124 CPU, 256GB of RAM and NVIDIA H100 GPU. The cluster uses SLURM middleware for it's job managment and we used sbatch command to run our jobs. To have reproducible environment we used containerization with Apptainer. Apptainer is a containerization solution mainly used in high performance computing. We used Python version 3.12 and we have gathered all the requirements into requirements.txt file and specified the versions of the packages. The source code is available in the repository[1]. Results can be reproduced by logging into the supercomputer cluster of your choice, cloning the repository and executing following commands.

**Listing 3.** Commands for running the training and evaluation

```
cd src
sbatch run.sh
```

Sbatch command will put job in the SLURM queue. After the job is accepted Apptainer container with the environment will be built. When the environment is built, the main script will be executed inside the containerized environment, firstly all the models will be trained and the evaluated.

## Results

**Slovene SuperGLUE** While most of the research in the field of NLP has been conducted for the English language, we also aim at applying the PEFT methods to multilingual text reasoning. The benchmark used is the Slovene SuperGLUE [13], which is the Slovene version of the English SuperGLUE. The benchmark consists of eight tasks encompassing general NLP tasks. Let us begin with the task BoolQ: a dataset with questions and binary answers [2]. Each instance consists of three components: a question, a passage, and an answer, with the possibility of including the page title as supplementary context. This configuration mirrors the structure of typical natural language inference tasks focused on text pairs.

We employed the BERTić [14] model for fine-tuning, which is a transformer language model specifically designed for Bosnian, Croatian, Montenegrin, and Serbian. BERTić has been trained on large corpora in these languages and has shown effectiveness in various NLP tasks in the region.

| Metric | FFT | LoRA | Unit |
|---|---|---|---|
| Training Time | 3967.46 | 3276.88 | seconds |
| RAM Usage | 760.78 | 458.01 | MB |
| Num. Parameters | 110,618,882 | 1,200,386 | - |
| Accuracy | 0.74 | 0.65 | - |
| Precision | 0.78 | 0.71 | - |
| F1-Score | 0.79 | 0.73 | - |
| Recall | 0.81 | 0.75 | - |

**Table 2.** Comparison of Full Fine Tuning (FFT) vs. LoRA

When comparing the full fine-tuning (FFT) method with the LoRA method on the Slovene SuperGLUE task, several important observations can be made. Firstly, LoRA exhibits notable computational advantages in terms of training time and memory usage, which suggests its potential for efficient model training, especially in resource-constrained environments. However, the reduction in trainable parameters with LoRA might imply a trade-off in model complexity and potentially performance. The higher number of parameters in FFT indicates a more complex model, which can capture finer

[1] https://github.com/UL-FRI-NLP-2023-2024/ul-fri-nlp-course-project-naturallylazypeople

[2] https://huggingface.co/datasets/google/boolq

nuances in the data, potentially leading to better performance metrics such as accuracy, precision, F1-score, and recall. The discrepancy in performance between FFT and LoRA underscores the importance of considering not only computational efficiency but also model capacity and task-specific requirements when selecting a fine-tuning method.

While LoRA may offer computational advantages, FFT's superior performance on this particular Slovene task suggests that a balance between computational efficiency and model complexity is crucial for achieving optimal results in NLP tasks.

## Future directions, ideas

In the future we want to focus on other PEFT method, *BitFit* and *Prompt Tuning*.

Other direction of our work will be to broaden selection of benchmarks on which we test the methods.

Lastly, we may do more in depth hyperparameter tuning regarding each model, e.g. with LoRa try different matrix ranks, etc.

## Discussion

## References


[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training, 2018.

[4] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023.

[5] OpenAI. Gpt-4 technical report, 2024.

[6] 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, 2021.

[7] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[8] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2022.

[9] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge, 2019.

[10] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint conference on EMNLP and CoNLL-shared task*, pages 1–40, 2012.

[11] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

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

[13] Aleš Žagar and Marko Robnik-Šikonja. Slovene SuperGLUE benchmark: Translation and evaluation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2058–2065, Marseille, France, June 2022. European Language Resources Association.

[14] Nikola Ljubešić and Davor Lauc. BERTić - the transformer language model for Bosnian, Croatian, Montenegrin and Serbian. In *Proceedings of the 8th Workshop on Balto-Slavic Natural Language Processing*, pages 37–42, Kiyv, Ukraine, April 2021. Association for Computational Linguistics.