

https://dspace.org/

Parameter-Efficient Fine-Tuning of Pre-trained Large Language Models for Financial Text Analysis

Langa, Kelly Kiba

2024-07
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/47000
Langa, Kelly Kiba. (2024). Parameter-Efficient Fine-Tuning of Pre-trained Large Language Models for Financial Text Analysis. [Master's dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace. https://hdl.handle.net/10539/47000

Downloaded from a DSpace repository. https://dspace.org/

Parameter-Efficient Fine-Tuning of Pre-trained Large Language Models for Financial Text Analysis

---

Kelly Langa

Supervisors:
Dr. Hairong Bau
Dr. Olaperi Okuboyejo



A research report submitted in partial fulfillment of the requirements for the degree of Master of Science in the field of e-Science

in the

School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

1 July 2024

Declaration

I, Kelly Langa, hereby affirm that this report represents my original and independent work. It is being submitted in fulfillment of the requirements for the degree of Master of Science in the field of e-Science at the University of the Witwatersrand, Johannesburg. This report has not been submitted for any other degree or examination at any other university.



Kelly Langa

1 July 2024

Abstract

The recent advancements in natural language processing (NLP) have been largely fueled by the emergence of large language models (LLMs), which excel in capturing the complex semantic and syntactic structures of natural language. These models have revolutionized NLP tasks by leveraging transfer learning, where pre-trained LLMs are fine-tuned on domain-specific datasets. Financial sentiment analysis poses unique challenges due to the intricate nature of financial language, often necessitating more sophisticated approaches beyond what traditional sentiment analysis methods offer. Fine-tuning LLMs holds potential for improving modeling performance within the financial domain, but the computational expense of the standard full fine-tuning poses a challenge. This study investigates the efficacy of Parameter-Efficient Fine-Tuning (PEFT) methods for fine-tuning LLMs to specific tasks, with a focus on sentiment analysis in the financial domain. Through extensive analysis of PEFT methods, including Low-Rank Adaptation (LoRA), prompt tuning, prefix tuning, and adapters, several critical insights have emerged. The results demonstrate that by employing PEFT methods, performance levels that match or surpass those of full fine-tuning can be achieved. Particularly, adapting the Open Pre-trained Transformers (OPT) model with LoRA achieved the highest modeling performance, with an accuracy of 89%, while utilizing 0.19% of the model's total parameters. This highlights the high modularity of PEFT methods, necessitating minimal storage sizes for trainable parameters, ranging from 0.1MB to 7MB for the OPT model. Despite slower convergence rates than full fine-tuning, PEFT methods resulted in substantial reductions in Graphics Processing Unit (GPU) memory consumption, with savings of up to 80%. Small-scale fine-tuned LLMs outperformed large-scale general-purpose LLMs such as ChatGPT, emphasizing the importance of domain-specific fine-tuning. Model head fine-tuning fell short compared to PEFT methods, suggesting additional benefits from training more layers. Compared to state-of-the-art non-LLM-based deep learning models, Long Short-Term Memory (LSTM), LLMs demonstrated superiority achieving a 17% increase in accuracy, thereby validating their higher implementation costs.

Acknowledgements

I would like to extend my heartfelt gratitude and appreciation to Dr. Hairong Bau and Dr. Olaperi Okuboyejo for their invaluable guidance, advice, and unwavering support throughout this study. I am also deeply appreciative of the DST-CSIR National e-Science Postgraduate Teaching and Training Platform (NEPTTP) for their generous financial support, which has made my studies possible. Their investment in my education is a testament to their commitment to nurturing the next generation of scholars in scientific research.

Contents

Declaration i

Abstract ii

Acknowledgements iii

List of Figures vii

List of Tables ix

1 Introduction 1

1.1 Background . . . 3
1.2 Problem Statement . . . 5
1.3 Research Aim and Objectives . . . 5
1.3.1 Aim . . . 6
1.3.2 Objectives . . . 6
1.4 Research Questions . . . 6
1.5 Significance and Motivation . . . 7
1.6 Main Findings of the Study . . . 9
1.7 Contributions of the Study . . . 10
1.8 Delineation of the Study . . . 10
1.9 Research Report Outline . . . 11

2 Literature Review 12

2.1 Introduction . . . 12
2.2 Large Language Models . . . 12
2.2.1 Pre-training of Large Language Models . . . 13
2.2.2 Generative Artificial Intelligence . . . 15
2.3 The Transformer Model Architecture . . . 16

2.3.1 Encoder-only Transformer Architecture . . . 16
2.3.2 Decoder-only Transformer Architecture . . . 18
2.3.3 Encoder-Decoder Transformer Architecture . . . 19
2.4 Transfer Learning . . . 20
2.5 Fine-Tuning . . . 20
2.6 Parameter-Efficient Fine-Tuning . . . 22
2.6.1 Prompt tuning . . . 22
2.6.2 Prefix tuning . . . 24
2.6.3 Adapter tuning . . . 26
2.6.4 Low-rank adaptation . . . 27

3 Research Methodology **29**
3.1 Research Design . . . 29
3.2 Data and Data Preprocessing . . . 30
3.2.1 Data Description . . . 30
3.2.2 Data Preprocessing . . . 31
3.2.2.1 Tokenization and Encoding . . . 31
3.2.2.2 Batch processing and Padding . . . 31
3.2.2.3 Attention Mask . . . 32
3.2.2.4 Sentiment Label Encoding . . . 32
3.3 Methods . . . 32
3.3.1 Training, Validation and Test Split . . . 32
3.3.2 Pre-trained Models . . . 33
3.3.3 Incorporating PEFT Techniques in Fine-Tuning LLMs . . . 35
3.3.4 Model Training . . . 39
3.3.4.1 Hyperparameter Tuning . . . 39
3.3.5 Hardware, Libraries and Software . . . 40
3.4 Analysis . . . 41
3.4.1 Benchmark . . . 41
3.4.2 Evaluation Metric . . . 42
3.5 Limitations . . . 42

4 Results and Discussion **44**
4.1 Knowledge Transfer Efficacy of LLMs through PEFT techniques . . . 44
4.2 Comparing the Modeling Performance of PEFT Methods . . . 46

4.3 Number of Trainable Parameters 49
4.4 Portability and Modularity of PEFT Methods 51
4.5 Computational Efficiency 52
4.6 Convergence Rate 53
4.7 Training Speed 55
4.8 Inference Latency 56
4.9 Scaling LLMs and Zero-shot Inference 57
4.10 Model Head Fine-tuning 58
4.11 Comparison of LLMs with Traditional NLP Algorithms 59
4.12 Summary 60

5 Conclusion and Future Work **62**
5.1 Conclusion 62
5.2 Limitations of the Study 63
5.3 Future Work 64

A Effectiveness of Adaptation Methods **65**
A.1 Fine-tuning results 65

B Hyperparameters **68**
B.1 Hyperparameters for Fine-tune LLMs using PEFT Methods 68

Bibliography **71**

List of Figures

1.1 Performance of proprietary and open-source LLMs on the Massive Multitask Language Understanding (MMLU) benchmark [66] 8

2.1 A timeline of recent LLMs with +10 billion parameters [80]. 15
2.2 The transformer model architecture [70] 17
2.3 Pre-trained model adaptation methods [36] 21
2.4 Modelling performance vs training efficiency of adaptation mechanisms. 21
2.5 Parameter-efficient fine-tuning methods taxonomy [43] 23
2.6 Prompt tuning incorporated into the transformer architecture [48] . . 24
2.7 Structure of prefix tuning [56] 25
2.8 Adapters incorporated in the transformer architecture using the Houlsby adapters configuration [26] 27
2.9 Low-rank adaptation incorporated in the transformer architecture using the self-attention layer LoRA configuration [71] 28

3.1 Sentiment polarity distribution 31
3.2 Sequence length distribution 31
3.3 Parallel adapters configuration [56] 38
3.4 Hyperparameter search using HyperOpt algorithm with ASHA scheduler 40

4.1 Modelling performance of PEFT methods and their various configurations with full fine-tuning as the baseline 45
4.2 Confusion matrix generated for each PEFT method when applied to the OPT model 48
4.3 Performance of PEFT methods with number of trainable parameters 50
4.4 Peak GPU memory consumed during fine-tuning 53

4.5 Convergence rate of PEFT methods with full fine-tuning as the baseline 54
4.6 Inference latency using various fine-tuning methods 56
4.7 Model head fine-tuning 59

A.1 Modelling performance of PEFT methods and their various configurations with full fine-tuning as the baseline 65

List of Tables

3.1 Comparative overview of hyperparameters utilized in the pre-training of RoBERTa-large, FLAN-T5-base and OPT-350 models 35
3.2 PEFT methods and their configurational variants 36
3.3 Hyperparameters used for fine-tuning OPT model with LoRA 39
3.4 Benchmark methods 42
3.5 Modelling performance and training efficiency evaluation metrics. . 42

4.1 Comparison of trainable parameters and training time for different fine-tuning methods 49
4.2 Modelling performance of large-scale LLMs in zero-shot inference [42] 57
4.3 Modelling performance of small-scale LLMs in zero-shot inference . 58
4.4 The modelling performance of LSTM models and OPT model fine-tuned using LoRA 59

A.1 Results of fine-tuning RoBERTa, FLAN-T5 and OPT models using PEFT methods 66
A.2 Trainable parameters using PEFT methods to fine-tune RoBERTa, FLAN-T5 and OPT models 67

B.1 Training parameters of RoBERTa, FLAN-T5 and OPT models using LoRA 68
B.2 Training parameters of RoBERTa, FLAN-T5 and OPT models using prompt tuning 69
B.3 Training parameters of RoBERTa, FLAN-T5 and OPT models using prefix tuning 69
B.4 Training parameters of RoBERTa, FLAN-T5 and OPT models using adapter tuning 70

Chapter 1

Introduction

The field of natural language processing (NLP) has experienced remarkable advancements in recent years, largely attributed to the emergence of large language models (LLMs) that have attained cutting-edge performance on a wide range of language-related tasks. LLMs possess the ability to capture the intricate semantic and syntactic complexities of natural language, making them highly suitable for tasks that require language understanding and text generation [14]. Given their centrality in language tasks, there has been a fundamental shift in NLP methodologies, moving away from specialized custom models trained from scratch towards building on top of general purpose base models. With the increasing availability of open-source pre-trained general purpose base LLMs, transfer learning has emerged as a dominant paradigm to specialize (fine-tune) pre-trained models for specific downstream tasks using a domain-specific dataset.

The current state-of-the-art LLMs such as GPT-4 and Gemini Ultra are generalists in that they have been pre-trained on a diverse large corpus of text. These LLMs perform poorly out-of-the-box (zero-shot inference) on knowledge-intensive tasks as these tasks require domain specific expertise and go beyond syntax analysis or simple pattern recognition. One such task is financial sentiment analysis, which involves predicting the sentiment polarity of finance textual data, such as earnings calls and social media posts. Financial sentiment analysis poses unique challenges due to the complex and domain-specific nature of financial data, characterized by technical jargon and acronyms [51]. Traditional approaches to sentiment analysis using rule-based or machine learning techniques have limitations in accurately capturing the nuances of financial language, resulting in suboptimal performance [17].

Fine-tuning base LLMs for financial sentiment analysis holds promise for improving modelling accuracy in this domain. Fine-tuned LLMs have shown promising results in financial sentiment analysis due to their ability to learn comprehensive language representations that capture the semantic and syntactic relationships within words and phrases [77]. Full parameter fine-tuning, which encompasses fine-tuning all the layers of a pre-trained LLM, remains the most effective method for adapting a model to a new domain and task in terms of modelling performance [37]. However, full fine-tuning is computationally expensive and time-consuming for training large-scale language models, making it almost infeasible in acade-mia and industry. In recent years, researchers have been actively exploring light-weight adaptation methods that allow fine-tuning using a small subset of parameters while maintaining high modelling performance [20]. Parameter-efficient fine-tuning (PEFT) techniques are lightweight adaptation methods which involve adding a small set of weights to a pre-trained LLM and only fine-tuning the newly added weights, while keeping the pre-trained LLM parameters fixed.

Base LLMs acquire powerful general representations during pre-training, embedding compressed knowledge into the neural network's weights. Through transfer learning, we can harness all the linguistic knowledge encoded in a language model. PEFT endeavors to make the transfer learning training process less compute-intensive. This approach offers several advantages, including reduced computational costs, shorter training times, adaptability to lower hardware specifications such as smaller GPUs and limited memory, potential improvement in modeling performance by mitigating overfitting, and decreased storage requirements due to weight sharing across various tasks [12]. PEFT methods including prompt tuning, low-rank adaptation and adapter, which modify a small subset of newly added weights while keeping the pre-trained model parameters fixed, have demonstrated comparable performance to full fine-tuning in certain NLP tasks while greatly decreasing computational costs and time required for training [20].

This research project aims to explore the potential of adapting LLMs, considering

both modelling performance and training efficiency, for sentiment analysis of financial data using PEFT methods. By harnessing the capabilities of LLMs and optimizing the fine-tuning process, the project seeks to improve the efficiency and accuracy of sentiment analysis in the financial domain, enabling the democratization of artificial intelligence and better data-driven decision-making.

1.1 Background

Financial modelling has evolved significantly over the years, due to advancements in NLP and machine learning techniques. Historically, financial modelling was purely reliant on quantitative financial data. Subsequent research found that financial performance is more than just a function of quantitative data [51]. Finance has embraced the integration of textual analysis alongside its traditional quantitative methods, bridging the gap between numerical data and subjective factors that influence financial markets [51]. By combining quantitative and qualitative factors, investors gain a more holistic view of market dynamics [47]. Some common sources of data used in financial text analysis include company fillings and reports, financial news services, social media platforms, and press releases and announcements. In recent times, sentiment analysis has become pivotal in algorithmic trading and investment strategies, gaining traction as a commercial application in NLP [47].

Sentiment analysis is a challenging task as financial language is complex and highly specialized, requiring a deep understanding of financial concepts, instruments, and industry specific terms. The methods used in financial sentiment analysis has evolved significantly. In the early years, lexicon-based methods were used in determining the sentiment polarity in financial text [55]. Sentiment polarity was determined by analysing the presence and frequency of positive and negative words from specially curated financial sentiment lexicons, which included comprehensive lists of both positive and negative words [51]. Around the same period, rule-based systems emerged, incorporating linguistic rules and patterns specific to financial sentiment analysis [55]. Manually designed rules were used to identify sentiment bearing

words and phrases related to financial concepts. These approaches lacked context and struggled to capture the nuances of sentiment in financial language [55].

Machine learning and deep learning models have transformed NLP in finance. The growth in labelled datasets saw researchers beginning to use supervised machine learning to train models on financial textual data [51]. The greatest success in sentiment analysis was achieved through deep learning by the utilization of recurrent neural networks (RNNs) [3]. This algorithm learns contextual representations and significantly improved the accuracy of sentiment analysis [3]. However, RNNs don't perform well with long sequences and encounter challenges with vanishing and exploding gradient issues. In mitigating the long-term dependency problem, gated RNNs were developed which are designed to selectively retain information and capture long term dependencies [74]. There are two variants of gated RNNs, namely, long short-term memory (LSTM) and gated recurrent unit (GRU). In selectively retaining information, LSTMs have a cell state, forget gate, input gate and output gate while GRUs have a reset gate and update gate. GRUs provide comparable performance to LSTM while offering reduced complexity due to having fewer weights. The vanishing gradient problem persists in LSTMs and GRUs which are designed to mitigate this issue. Furthermore, sequential nature of processing makes them slow in their computation. Despite these limitations, RNNs provided a foundation for more advanced approaches.

The transformer architecture serves as the cornerstone of recent advancements in NLP. Transformer-based models have attained state-of-the-art results by utilizing an attention mechanism to capture the interdependencies among different elements of a sequence [70]. The attention mechanism allows transformers to learn long-range dependencies in sequences [70]. The rapidly growing ecosystem of LLMs has led to a paradigm shift in NLP. Rather than building a model from the ground up, a pre-trained model is utilised and adapted to a specific task. Transfer learning enabled sentiment analysis models generalise better, even with limited labelled financial data [20]. Researchers today are concerned with finding adaptation mechanisms that are computationally efficient without sacrificing modelling performance.

1.2 Problem Statement

Textual analysis has emerged as a valuable addition to the methodological toolkit in finance. Sentiment analysis of financial data is a daunting task due to the complexity and inherent ambiguity of the language used in financial documents. Traditional approaches to sentiment analysis using rule-based or machine learning techniques have limitations in accurately capturing the nuances of financial language, leading to suboptimal results [55, 47]. Furthermore, building a successful customized sentiment analysis model from scratch is time-consuming and requires massive amounts of labelled data [8, 36]. With the increasing availability of LLMs with permissive licenses that allow for commercial use, transfer learning has gained prominence as a promising approach for fine-tuning pre-trained models on domain specific data to perform sentiment analysis tasks [80, 16]. The challenge, however, is that fine-tuning LLMs for a specific task on a custom domain-specific dataset is notoriously difficult due to their immense size [8]. The gold standard of full fine-tuning LLMs is very expensive in terms of computational resources and training time [20, 19]. Moreover, the expenses associated with storing and deploying each full fine-tuned model can quickly escalate. Dedicated research efforts in recent years have resulted in the emergence of lightweight adaptation mechanisms. Parameter-efficient fine-tuning techniques significantly reduces the costs associated with model adaptation, both in terms of data volume and model optimization [19, 45, 12].

1.3 Research Aim and Objectives

The broader aim of this research was to contribute to the democratization of AI. This research seeks to make LLMs more accessible, inclusive, and equitable by developing techniques that empower individuals and organizations to harness the power of LLMs.

1.3.1 Aim

The aim of this study was to leverage parameter-efficient fine-tuning methods to develop a resource-efficient predictive model. Specifically, the study focused on utilizing a pre-trained large language model to accurately determine the sentiment polarity of financial data.

1.3.2 Objectives

The objectives of this study are:

- To fine-tune pre-trained LLMs (RoBERTa, FLAN-T5 and OPT) using full parameter fine-tuning, model head (last layer) tuning and zero-shot learning (no fine-tuning) for performing sentiment analysis on financial data.
- To utilize parameter-efficient fine-tuning methods (prompt tuning, prefix tuning, adapters and LoRA) to fine-tune pre-trained LLMs (RoBERTa, FLAN-T5 and OPT) for sentiment analysis on financial data.
- To utilize various configurations of parameter-efficient fine-tuning methods to conduct sentiment analysis on financial data.
- To evaluate the performance of parameter-efficient fine-tuned LLMs in terms of modelling performance and training efficiency.
- To compare the performance of parameter-efficient fine-tuned LLMs to full fine-tuned LLMs, model head (last layer) tuning and zero-shot learning (no fine-tuning) in terms of modelling performance and training efficiency.
- To compare the performance of parameter-efficient fine-tuned LLMs to traditional NLP methods (LSTMs)

1.4 Research Questions

- Which of the selected LLMs are more effective in transfer learning using the various PEFT methods for the task of financial sentiment analysis?

- How does the performance of the PEFT methods compare in terms of modelling performance and training efficiency in fine-tuning the selected LLMs to a financial sentiment analysis task?
- What is the optimal configuration of the various PEFT methods in terms of modelling performance and training efficiency?
- What is the modelling performance and training efficiency gap between PEFT methods and full parameter fine-tuning, model head (last layer) fine-tuning and zero-shot learning (no fine-tuning)?
- How does the modelling performance of LLMs fine-tuned using PEFT methods compare to the selected state-of-the-art traditional NLP method?

1.5 Significance and Motivation

The recent proliferation of LLMs has validated their remarkable potential across various applications and provides developers with additional choices for building AI applications. To date, there has been widespread acceptance of LLMs as a great consumer tool e.g. ChatGPT. One trend which is still under appreciated is LLMs as a developer tool. Pre-trained LLMs can enable application developers to achieve previously unseen feats in AI. Businesses can integrate these LLMs into their core business operations, services and products. Previously building and deploying a commercial grade machine learning system typically required a timeline of 6 to 12 months [57]. However, with LLMs, many applications that were once challenging to develop can now be built quickly and effortlessly.

The downside of employing LLMs as a developer tool is that the current state-of-the-art LLMs are proprietary and can only be accessed through paid application programming interface (API) services. Access to these proprietary LLMs is cost prohibitive, and full access to the model weights is unavailable, rendering it impossible to fine-tune them with your private data. However, the open-source AI community is making significant strides and the speed of innovation and research in open-source LLMs is remarkable. Open-source LLMs are poised to surpass proprietary ones, as depicted in Figure 1.1. A majority of these open-source LLMs are released

under the Apache 2.0 license, permitting commercial use. The lingering challenge lies in the fine-tuning and practical deployment of these extensive LLMs, which poise substantial cost and efficiency hurdles. Due to the exorbitant cost of accessing cloud GPUs and the current scarcity and expense of purchasing GPUs, there is a concerted effort to minimize the utilization of multiple GPUs in model inferencing and fine-tuning. Lightweight adaptation techniques like PEFT methods show potential for addressing these challenges.

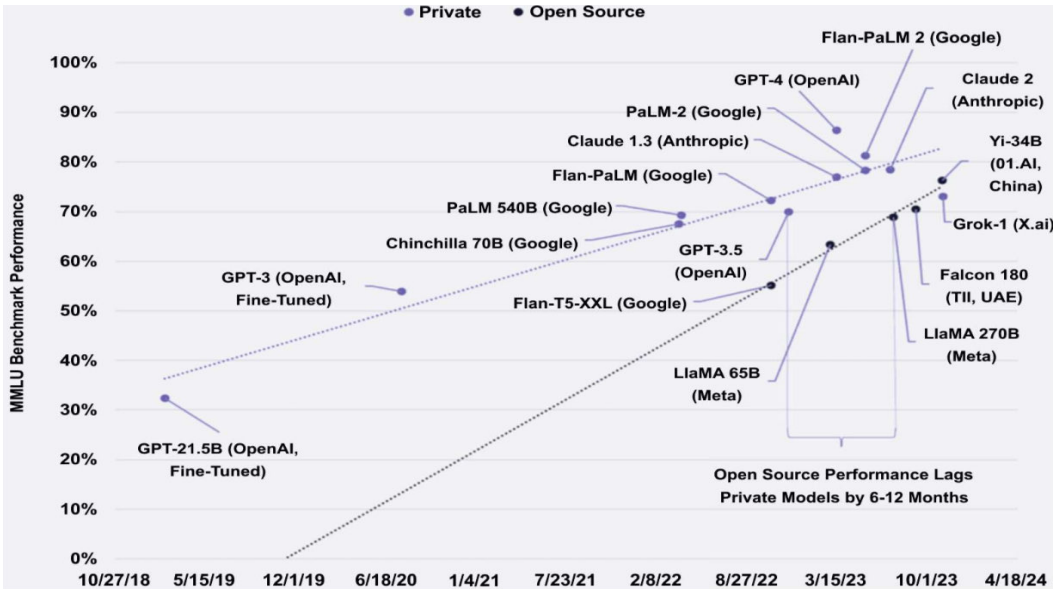


FIGURE 1.1: Performance of proprietary and open-source LLMs on the Massive Multitask Language Understanding (MMLU) benchmark [66]

The research and development of LLMs is currently predominantly led by big technology companies with substantial financial resources. The findings of this study could democratize access to LLMs and foster broader innovation by empowering individuals and organisations with average consumer-grade hardware to leverage the capabilities of LLMs for their specific use cases. By reducing the computational requirements for fine-tuning LLMs, we hope to make LLMs more accessible and inclusive to a wider audience, bringing more voices in studying these LLMs. LLMs are still in their early phases of development and continue to face difficulties such as hallucination and catastrophic forgetting, human alignment, AI existential risk and regulation, bias, robustness, interpretability and explainability, jailbreaks and copyright [8, 79]. The magnitude of transformation brought about by this technology to

our way of life is too significant to be defined solely by a few AI labs with substantial resources. The potential dangers associated with AI monopolies are significant. The entire AI community, including industry, academic researchers, policymakers and civil society must collaborate to shape and collectively drive the field forward.

1.6 Main Findings of the Study

The main findings of this study are the following:

- We developed resource-efficient models leveraging LLMs and PEFT methods to achieve state-of-the-art performance in financial sentiment analysis on the financial phrase bank dataset.
- We found that PEFT methods led to significant savings in peak GPU memory consumption, conserving up to 80% of GPU memory.
- Contrary to most literature, we observed that generative models can outperform descriminative models in natural language understanding tasks, emphasizing the influence of model size and pre-training dataset size on LLM capabilities.
- We found that PEFT methods demonstrate high modularity by utilizing merely a fraction of the model's total parameters, allowing for the creation of compact checkpoints compared to full fine-tuning.
- We discovered that within the transformer architecture, lower layers capture general features shared among tasks, while later layers specialize in task-specific features, making them more beneficial for adapting LLMs to a specific task.
- We observed that surpassing a specific threshold of trainable parameters resulted in a notable decrease in performance, attributed to an excessive dependency on newly introduced parameters.
- We found that PEFT methods typically demonstrated slower convergence rates compared to full fine-tuning. However, strategic adjustments in batch sizes

can unleash the full potential of PEFT techniques, speeding up model adaptation while ensuring computational feasibility.

- We found that the convergence rate of PEFT methods is less influenced by the number of trainable parameters and more affected by the structural characteristics of the PEFT methods.
- We found that domain-specific fine-tuning of small-scale LLMs outperform large-scale general-purpose LLMs out-of-the-box.
- We discovered that the notion that fine-tuning LLMs necessitates a large dataset is inaccurate. Our findings indicate that domain-adapted LLMs outperform LSTMs even with smaller datasets.
- We found that training more layers beyond just the last layer yielded benefits in modelling performance.

1.7 Contributions of the Study

The main contribution of this study lies in conducting a comprehensive comparative study into the fine-tuning of different open-source LLMs employing different PEFT methods and their various configurations. This study sheds light on the optimal configuration and placement of different PEFT methods, highlighting the weaknesses and strengths in utilizing pre-trained LLMs to perform financial sentiment analysis.

1.8 Delineation of the Study

This study is delineated by its focus on financial sentiment analysis using pre-trained LLMs and PEFT techniques. It does not extend to other applications of LLMs and other domains outside of finance. The research is confined to the evaluation of specific PEFT methods, namely prompt tuning, prefix tuning, adapter tuning, and LoRA, and does not explore other fine-tuning techniques. Additionally, while the study benchmarks the performance of these methods against traditional NLP methods and full parameter fine-tuning, it does not delve into developing new

PEFT methods or significantly modifying existing ones. By clearly defining these boundaries, the study aims to provide a focused and rigorous examination of the effectiveness and efficiency of PEFT methods for financial sentiment analysis, while acknowledging the limitations imposed by the scope.

1.9 Research Report Outline

The following chapters of this research report are organized into four chapters. Chapter 2, the literature review, provides an in-depth exploration of the relevant topics, including large language models, the transformer model architecture, transfer learning, model adaptation, and lightweight PEFT methods. Chapter 3, the research methodology, outlines the research design, data collection and preprocessing methods, and details the training and evaluation processes. Chapter 4 presents the results of the study and provides a detailed discussion of the findings. Finally, chapter 5 provides a comprehensive conclusion, summarizing the key findings and providing insights into future research directions.

Chapter 2

Literature Review

2.1 Introduction

The emergence of large-scale pre-trained language models has revolutionized natural language processing tasks. However, tailoring these models to particular domains or tasks through fine-tuning can be computationally expensive and memory-intensive. This literature review explores existing studies on lightweight adaptation methods of LLMs. It provides an overview of approaches, methodologies, and findings focused on achieving efficient model adaptation while maintaining high performance. The review discusses methods such as prompt tuning, prefix tuning, adapter, transformer architecture modification, and knowledge transfer. The findings highlight the potential of parameter-efficient fine-tuning for reducing computational requirements and memory footprint without significant degradation in model performance.

2.2 Large Language Models

Large scale pre-trained language models have become a transformative development in the realm of NLP. LLMs are deep learning algorithms that have undergone training on a large unstructured text corpus of unlabelled data through a self-supervised learning process [8]. LLaMA (Large Language Model Meta AI), the latest LLM From Meta was trained on 1.4 trillion tokens and has 65 billion parameters [69]. The extensive size of LLMs results in a substantial amount of knowledge and information encoded within the model. LLMs exhibit profound

knowledge of various linguistic aspects, including semantics, sentiment, phrasal structure, co-occurrence patterns, and syntax, thanks to their extensive pre-training and exposure to vast amounts of text data [76, 49, 62]. The term "foundational models" was introduced by researchers from Stanford University [8] to describe a class of models that have been trained on diverse data and can be customized for different downstream tasks. Base LLMs are classified as foundational models due to their foundational linguistic knowledge required for various NLP tasks. The term foundational models emphasize the fundamental role the models play in NLP, serving as an initial foundation for subsequent fine-tuning and adaption to specific downstream tasks [49]. Language models are either pre-trained in an autoregressive fashion, where they take a sequence of tokens as input and recursively predict the next token to generate text, or in a masked fashion where masked words are predicted in a sequence taking into account the context surrounding them [68].

2.2.1 Pre-training of Large Language Models

Pre-training LLMs involves teaching these computational giants to understand and generate language by exposing them to vast amounts of text data. This foundational stage is crucial because it enables LLMs to cultivate an extensive grasp of language nuances, structure, and contexts, essentially encoding a condensed version of the knowledge found in the datasets they're trained on [62]. Utilizing self-supervised learning techniques, pre-training allows these models to learn general language representations, which can then be fine-tuned with a smaller set of labeled data for specific tasks. This approach has proven remarkably effective, with pre-training serving as a pivotal process that embeds wide-ranging knowledge into the model's parameters [11].

The tasks employed for pre-training LLMs are masked language modelling (MLM) and autoregressive language modelling (ALM). Autoregressive language modelling, for instance, is a key objective used to pre-train decoder-only models like GPT-3 and PaLM. This task involves predicting the subsequent tokens in a sequence based on the preceding ones, leveraging the vast, unlabeled datasets. Such a methodology

not only aids in learning the statistical correlations within the text but also, as models excel in next-step prediction, pushes them towards uncovering the "underlying truth" of the data they are trained on [60]. This notion of "understanding" the data goes beyond mere memorization, suggesting a deeper grasp of the content that's akin to a fundamental form of comprehension.

The scale and diversity of the pre-training data play a critical role in the development of these capabilities [64]. A dataset that is rich and varied ensures that the model has a broad base of knowledge to draw from, enhancing its ability to understand complex language structures, context, and semantics. However, the process of pre-training LLMs is fraught with challenges, including the immense computational demands and the need for high-quality training data [50]. These hurdles underscore the importance of developing more efficient and systematic pre-training methods that balance model effectiveness with computational efficiency and training stability.

An intriguing outcome of this rigorous pre-training is the emergence of unexpected abilities in LLMs, aptly termed "emergent capabilities" [39]. These capabilities tend to emerge unpredictably when language models reach a critical size and go beyond traditional language understanding and generation tasks [73, 2, 39]. These capabilities range from language translation to creative writing, and are not explicitly taught but rather evolve as a byproduct of the model's proficiency in predicting what comes next in the data. Such phenomena highlight the profound impact of pre-training, as it not only equips LLMs with a broad understanding of language but also enables them to adapt this knowledge creatively to new tasks, often with minimal additional input [72].

In essence, pre-training sets the stage for the remarkable abilities exhibited by LLMs, embedding within them a deep, albeit abstract, understanding of language that forms the foundation for their advanced computational abilities. As the field progresses, optimizing the pre-training process remains a pivotal area of research, promising even greater advances in the capabilities of these linguistic giants. Figure 2.1 illustrates a chronological representation of LLMs with a model size surpassing 10 billion parameters and developed post 2022. This figure provides a

comprehensive overview of the recent progress made in the development of LLMs. As highlighted in Figure 2.1, many of these LLMs have been released under open-source licensing, fostering the democratization of AI.

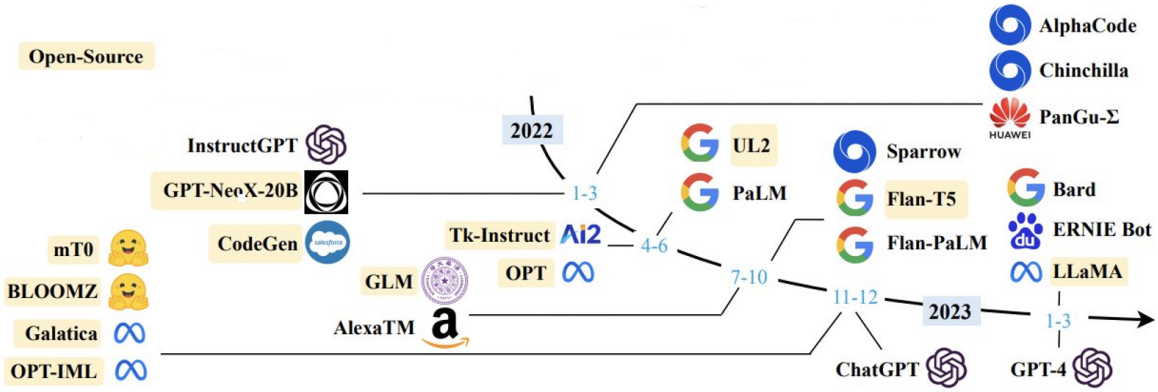


FIGURE 2.1: A timeline of recent LLMs with +10 billion parameters [80].

2.2.2 Generative Artificial Intelligence

The generative capability of LLMs, which allows them to predict the subsequent word based on preceding words positions them in the field of AI known as generative AI [76]. Generative AI focuses on developing models and algorithms that can generate new content. At its core, LLMs are primarily trained to perform generation tasks [12]. However, with the introduction of a small labelled dataset, LLMs can be adapted to specialise in a range of NLP tasks, including sentiment analysis [77]. The general-purpose nature of LLMs has made them widely adopted in various industries and applications. One of the key advantages of foundational models like LLMs is their flexibility in adaptation to various specialized tasks [14]. Once these models have undergone training, they possess the capability to be modified and fine-tuned repeatedly, allowing them to effectively address a variety range of specific tasks and domains [14].

2.3 The Transformer Model Architecture

The substantial success achieved in generative AI can largely be attributed to the utilization of transformer-based architectures [76]. The transformer architecture lies at the heart of nearly all recent advancements in NLP. This innovative architecture has revolutionized the field of generative AI by addressing several key challenges in sequence modelling. Traditional recurrent neural networks faced limitations in capturing long-term dependencies and parallel processing due to their sequential nature [68]. The transformer model is an innovative type of encoder-decoder model that leverages self-attention mechanism to model the connection among words in a sequence [70, 43, 55]. The attention mechanism empowers the model to assign varying degrees of importance to individual words based on their contextual relevance [70]. Transformer-based models can be classified into three categories based on their architecture: encoder-only, decoder-only, and encoder-decoder models. Each of these architectures serves different purposes and excels in distinct areas of application. The orginal transformer model utilised an encoder-decoder architecture as seen in Figure 2.2.

2.3.1 Encoder-only Transformer Architecture

The encoder consists of two components: a multi-head self-attention mechanism and a dense feed-forward layer. The input words are converted to vectors of a specified dimension using an embedding algorithm prior to being fed to the encoder [70]. The input word embeddings are also positionally encoded by adding a vector to each input embedding. Positional encoding is used to account for the order of the input words. The input word representation is channeled through the multi-head self-attention mechanism, enabling it to consider the contextual information from other words in the input sequence to determine how much attention it should pay to other words in the sequence [70]. This is essential in generating better encoding for that particular word vector. To calculate the self-attention score, three vectors are derived from each input vector of the encoder. The three representational vectors of the input are query (Q), key (K) and value (V). The self-attention score is calculated as $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$, where d_k

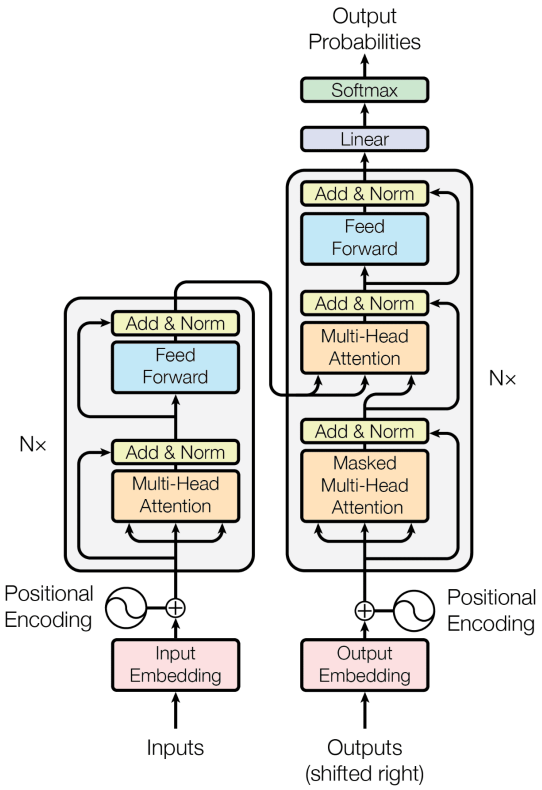


FIGURE 2.2: The transformer model architecture [70]

represents the dimension of the new vectors.

The self-attention layer in the encoder is called multi-head self-attention because the self-attention score is calculated multiple times with different weight matrices to improve the performance of the attention layer [45, 21]. This implies that each head acquires distinct information regarding a given sequence, and this knowledge is consolidated at the final stage. Subsequently, following the training phase, each head is employed to map the input embedding into a distinct representation subspace [70]. Equation 2.1 is the equation for multi-head self-attention where W^0 is the weight matrix.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^0 \tag{2.1}$$
$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

The fully-connected feed-forward sub-layer in the encoder processes the output

from the self-attention layer (attention vectors), and transforms them to a format that is acceptable to the next decoder layer [21]. The feed-forward layer consists of two linear transformers with a ReLU activation in between [45]. The feed-forward network can be represented as $FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$ where W and b are the weights and biases respectively. Every sub-layer in the encoder is followed by a normalization layer. The output from each sub-layer is normalized to effectively restrain the extent of the value variation within a specific layer which results in enhanced convergence speed of the model [70].

Encoder-only models, exemplified by Bidirectional Encoder Representations from Transformers (BERT), leverage bidirectional self-attention, enabling them to understand the context of each word in a sentence fully [50]. This architecture excels in discriminative tasks where the goal is to classify or identify elements within the input text rather than generate new text [77]. The success of encoder-only models in natural language understanding (NLU) tasks can be attributed to their training approach, notably the Masked Language Model objective. This pre-training method, where random words in a sentence are masked and the model predicts them based on context, helps in developing a profound understanding of language nuances. Despite their strengths in NLU, encoder-only models are inherently limited in text generation tasks due to their bidirectional nature, which focuses on understanding rather than producing text [77].

2.3.2 Decoder-only Transformer Architecture

In the transformer model structure, the decoder resembles the encoder but includes an additional masked multi-head self-attention component, which makes a total of three sub-layers in the decoder. Masked multi-head self-attention is a variant of the self-attention mechanism where certain elements of the attention matrix are masked to limit the model's access to certain information during training [45]. This masking is applied to enforce causality, ensuring that the model attends only to previous positions and avoids information leakage from future positions [70, 21].

In contrast, decoder-only models, such as GPT, adopt a unidirectional, autoregressive approach where each subsequent word is predicted based on the preceding context [79]. This architecture is highly effective for natural language generation (NLG), allowing for the creation of coherent and contextually relevant text sequences [79]. The unidirectional characteristic ensures a flow that mimics human language production, making it particularly suited for tasks like content creation, storytelling, and conversational AI. However, the generative prowess of decoder-only models comes at the cost of the deep contextual understanding provided by bidirectional encoder models [79].

2.3.3 Encoder-Decoder Transformer Architecture

Encoder-decoder models combine the strengths of both worlds, using an encoder to understand the input text and a decoder to generate output text. This architecture is ideal for sequence-to-sequence tasks such as translation and summarization, where the relationship between the input and output text is complex and requires both deep understanding and coherent generation capabilities [15]. Models like T5 and its FLAN-T5 variant have pushed the boundaries of what's possible with encoder-decoder architectures, adopting innovative training strategies such as sequence to sequence MLM.

Interestingly, the versatility of encoder-decoder models has also been demonstrated in non-autoregressive applications, where they outperform other architectures in certain NLP tasks by generating entire sequences in parallel rather than one token at a time [19]. This approach combines the efficiency of encoder models with the generative flexibility of decoder models, offering an effective tool for a broad spectrum of applications [64, 15].

The transformer architecture stands out as an efficient neural network architecture that offers remarkable scalability and easy parallelization primarily due to its unique self-attention mechanism. This unique design allows for seamless expansion and improved computational efficiency in processing large-scale datasets. Transformers have demonstrated exceptional effectiveness, positioning them as a

general-purpose architecture widely utilized in multi-modal applications spanning text, images, audio, video and even cross-modal domains [62, 77].

2.4 Transfer Learning

Transfer learning is among the most impactful concepts in deep learning and refers to the process of leveraging knowledge acquired from pre-training on a generic large-scale dataset to initialise a model for a target task [77]. In the realm of NLP, pre-trained LLMs capture general language patterns and semantic representation that possess transferability across diverse downstream tasks [47]. Leveraging a pre-trained model weights as a starting point to building a customized model has proven to be superior in terms of modelling performance and modelling efficiency as compared to building a model from scratch [14, 47]. There is no criteria to determine how well transfer learning will work, however, in general the target task and target domain must be related to the data and task the pre-trained model was trained on [64].

2.5 Fine-Tuning

Fine-tuning is employed to facilitate the efficient transfer of knowledge from pre-trained models to specific tasks. It involves taking a pre-trained model and adapting it to a specific task by further training it on a task-specific dataset. During fine-tuning, the model's parameters are updated using a smaller labelled dataset related to the target task. This process allows the model to refine its representation and learn task-specific patterns [54]. When deploying pre-trained LLMs in specialized fields such as law and finance, the model performs poorly and is therefore worthwhile to fine-tune the model on a domain-specific dataset [77].

Figure 2.3 illustrates the three most popular model adaptation methods. In the feature-based approach, the pre-trained model weights are kept frozen and used

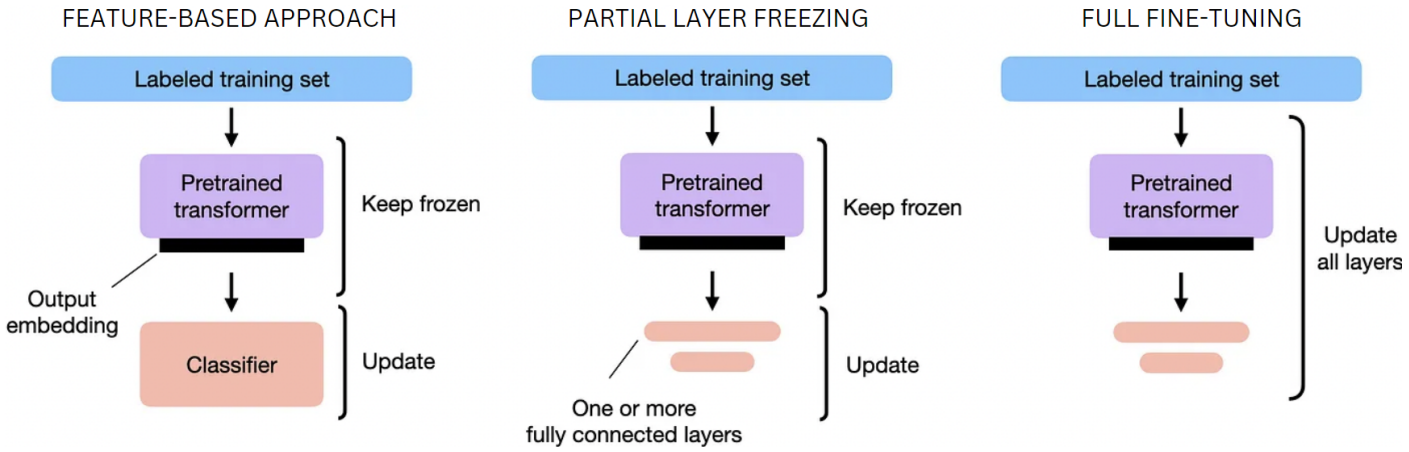


FIGURE 2.3: Pre-trained model adaptation methods [36]

as input features to train a classification model. In the partial layer freezing approach, the final layer of the pre-trained model, known as the model head, is substituted with one or more task-specific layers or adaptation modules are inserted within the layers of the transformer, while the weights of the base model remain fixed. With full parameter fine-tuning, all layers of the model are updated. Full fine-tuning of a pre-trained LLM is the gold standard approach for adapting a model to a new target task [36, 37]. However, it is computationally expensive and time consuming to update all weights of LLMs as illustrated in Figure 2.4. Figure 2.4 aligns with the common understanding that fine-tuning more layers typically leads to improved performance, but it comes with increased costs [37].

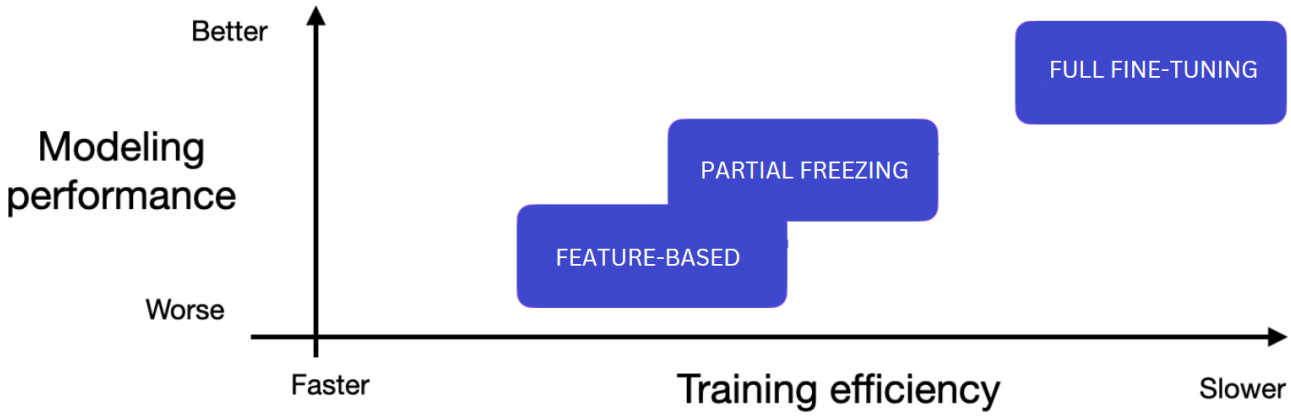


FIGURE 2.4: Modelling performance vs training efficiency of adaptation mechanisms.

2.6 Parameter-Efficient Fine-Tuning

In the past few years, substantial advancements have been made by researchers in the development of techniques for fine-tuning LLMs in a computationally efficient manner without sacrificing on modelling performance. These techniques are known as PEFT. PEFT methods enable efficient adaption of pre-trained LLMs to downstream tasks by adding a small set of weights to the pretrained LLM and only fine-tune the newly added weights while the weights of the pre-trained LLM remain frozen [20, 36]. This approach results in major savings in computational and storage costs [12]. According to Ding et al. [20], PEFT techniques can achieve performance levels that are comparable to those achieved through full fine-tuning.

Figure 2.5 shows the PEFT techniques taxonomy. PEFT techniques can be categorized into three main classes: addition-based, selection-based and reparameterization-based. Under the addition-based methods, we can further categorize them into two major groups: adaptor-like methods and soft prompts. PEFT techniques were developed based on the observation that earlier layers in pre-trained LLMs learn generic linguistic patterns that are applicable across various tasks while later layers tend to capture more task-specific patterns [43]. The later layers therefore undergo significant changes while earlier layers remain relatively stable during full fine-tuning [60]. PEFT techniques have produced state-of-the-art results, enabling fine-tuning of LLMs on devices with limited computational power [20]. The most notable ones are prompt tuning, prefix tuning, adapter tuning and low-rank adaptation (LoRA). Further elaboration on these techniques is provided in the subsequent subsections.

2.6.1 Prompt tuning

Prompt tuning represents an innovative approach for fine-tuning LLMs for specific tasks, leveraging the concept of soft prompts which are learnable tensors that

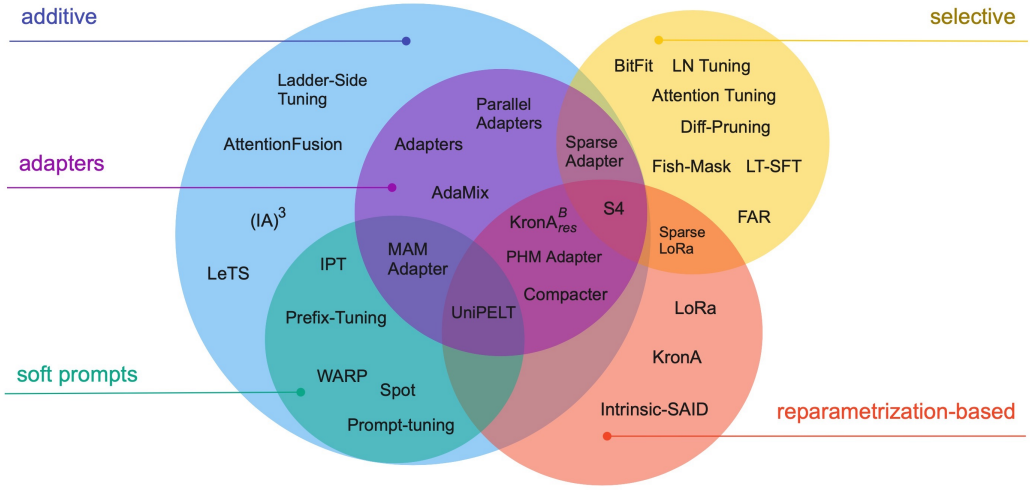


FIGURE 2.5: Parameter-efficient fine-tuning methods taxonomy [43]

are concatenated with the input embeddings. Prompt tuning inserts trainable vectors at the beginning of the input sequence only at the embedding layer, without altering the model's deeper layers as shown in Figure 2.6 [20]. Unlike traditional prompts that involve prepending text with specific instructions or examples, soft prompts are not directly interpretable by humans since they do not correspond to real-world tokens but rather serve as virtual tokens optimized during training [39]. This method preserves the original model parameters, focusing updates exclusively on the soft prompt embeddings through backpropagation, which makes it a more parameter-efficient alternative to full model fine-tuning.

A crucial aspect of prompt tuning is its initialization strategy, which significantly affects performance, especially in low-data scenarios. Initializing these soft prompts with the activations of real words, rather than randomly, has shown to enhance the model's ability to generate more relevant outputs [48]. This strategy aligns with the objective of maintaining the integrity of the pre-trained model's knowledge while enabling it to adapt to new tasks. The scalability of prompt tuning, as observed by recent studies, suggests that its performance can rival that of traditional fine-tuning methods, particularly in models with over 10 billion parameters [20].

Despite its efficiency and the reduced risk of overfitting, prompt tuning is not without challenges. It tends to converge slower than other tuning methods, which can make optimization more difficult in practice [39]. However, its ability to improve language model performance on natural language understanding tasks, combined with the computational and memory efficiencies it offers, makes it a valuable tool in the evolving landscape of model adaptation.

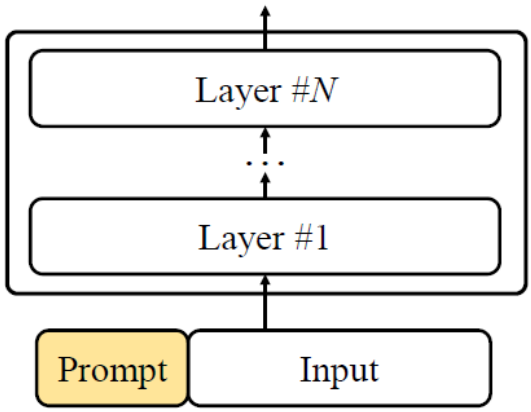


FIGURE 2.6: Prompt tuning incorporated into the transformer architecture [48]

2.6.2 Prefix tuning

Introduced by Li and Liang in 2021 [41], this technique innovates by appending task-specific trainable vectors, referred to as prefixes, not only to the input layer but across all layers of the transformer model [41]. These prefixes are integrated into the key and value matrices of the model's self-attention mechanism as in Figure 2.7, providing a task-relevant context that guides the model towards generating desired outputs [41]. Distinctively, prefix tuning diverges from direct manipulation of soft prompts. It employs a separate bottleneck feed-forward neural network (FFN) to optimize these prefix parameters, thereby circumventing the training instabilities and performance degradation associated with direct training through the reparameterizion of the prefix vectors [20]. After the optimization process, this FFN can be discarded, leaving the optimized prefixes to influence the model's behavior.

By keeping the bulk of the LLMs parameters frozen, prefix tuning introduces a minuscule fraction of new, trainable parameters into the model. Despite learning only about 0.1% of the total parameters, this method achieves performance comparable to full model fine-tuning in comprehensive data settings, and notably surpasses it in scenarios characterized by data scarcity or when extrapolating to unobserved topics [20]. Such efficiency is attributed to the method's ability to finely adjust the model's focus and processing through the nuanced steering provided by the prefix parameters. The length of the prefix plays a critical role in the tuning process, with empirical evidence suggesting that there exists an optimal prefix length beyond which performance may plateau or slightly decline [41]. This balance underscores the nuanced relationship between the amount of introduced task-specific information and model performance.

Despite its advantages, prefix tuning is not devoid of challenges. Its optimization process can be complex, necessitating experimentation with various parameters to achieve the best results [56]. Moreover, the introduction of prefixes reduces the available sequence length for actual input data, potentially limiting the model's capacity to handle longer inputs.

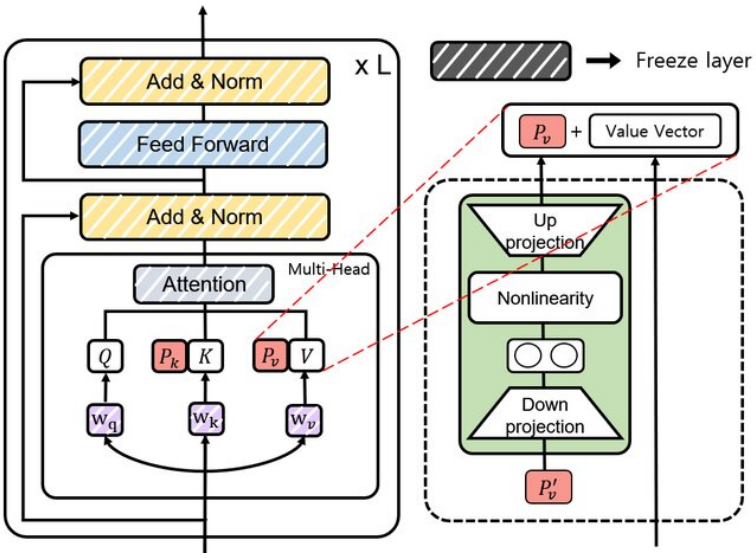


FIGURE 2.7: Structure of prefix tuning [56]

2.6.3 Adapter tuning

Adapter tuning has emerged as an innovative and lightweight approach to customizing pre-trained networks for specific downstream tasks without the need for comprehensive retraining of the entire model. By integrating small, fully connected networks known as adapters between the sub-layers of transformer models as shown in Figure 2.8, this method achieves notable performance enhancements while tuning fewer than 5% of the total model weights [30]. The essence of adapter tuning lies in its ability to introduce architectural modifications that repurpose a pre-trained network, focusing solely on the new parameters introduced by the adapters while keeping the original LLM parameters frozen [26]. This strategy not only preserves the model's acquired knowledge but also ensures a minimal addition of parameters, making it a highly memory-efficient solution.

The core architecture of these adapters incorporates a bottleneck design, which significantly reduces the dimensionality of the input features before applying a nonlinearity and projecting them back to their original dimension. This process, coupled with an internal skip-connection within each adapter module, allows for an effective balance between performance and parameter efficiency [56]. The skip-connection plays a crucial role, ensuring that the adapters can function as an approximate identity when their parameters are near-zero at initialization, thus preserving the integrity of the original input while introducing task-specific adaptations [30]. Originating from the pioneering work of Houlsby et al. [26], the adapter tuning method has been refined to include not just the bottleneck structure but also specific layer normalization parameters trained for each task. The Houlsby adapter configuration places two adapters in the sublayers of the transformer as shown in Figure 2.8.

Despite introducing additional parameters, the adapter method stands out for its computational efficiency, training can be up to 60% faster than traditional fine-tuning methods, with only a marginal slowdown during inference [24]. This efficiency, combined with the method's parameter and memory efficiency, positions adapter tuning as a formidable approach for customizing pre-trained models in a resource-constrained environment, allowing for rapid adaptation to new tasks

while leveraging the foundational knowledge embedded within large-scale pre-trained networks.

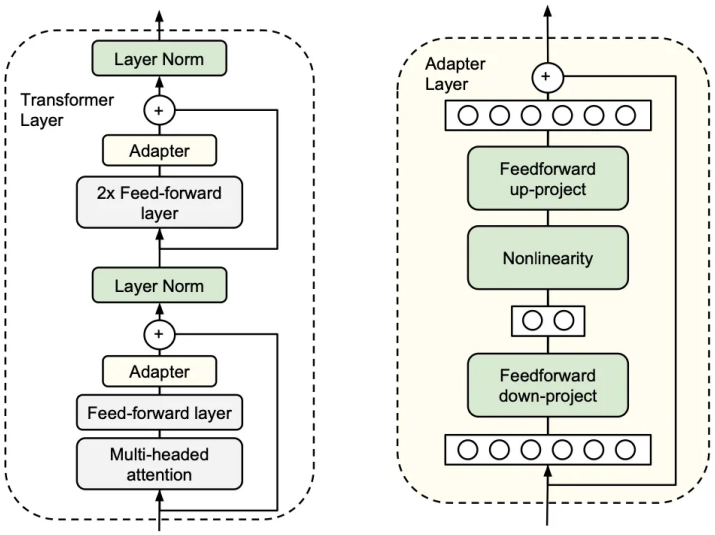


FIGURE 2.8: Adapters incorporated in the transformer architecture using the Houlsby adapters configuration [26]

2.6.4 Low-rank adaptation

Developed by Hu et al., LoRA fundamentally changes the approach to model adaptation by injecting trainable low-rank decomposition matrices into the layers of a pre-trained model [1]. Figure 2.9 illustrates a schematic representation of LoRA, showcasing the self-attention layer LoRA configuration which adds LoRA matrices to the self-attention sublayer. This method is predicated on the insight that pre-trained models, despite their size, operate within a low intrinsic dimension, meaning that their complexity can be effectively captured using matrices of much lower rank than initially presumed [1]. At the heart of LoRA is the concept of matrix rank, which signifies the maximum number of linearly independent rows or columns in a matrix. This concept is crucial because it informs us about the amount of unique information or the dimensionality of the space spanned by the matrix [83]. LoRA leverages this by approximating the weight adjustments needed during fine-tuning as the product of two smaller matrices (W_A and W_B), significantly

reducing the number of parameters required for effective fine-tuning. The rank r becomes a critical hyperparameter in this setup, dictating the degree of approximation and the balance between model complexity and training efficiency [18].

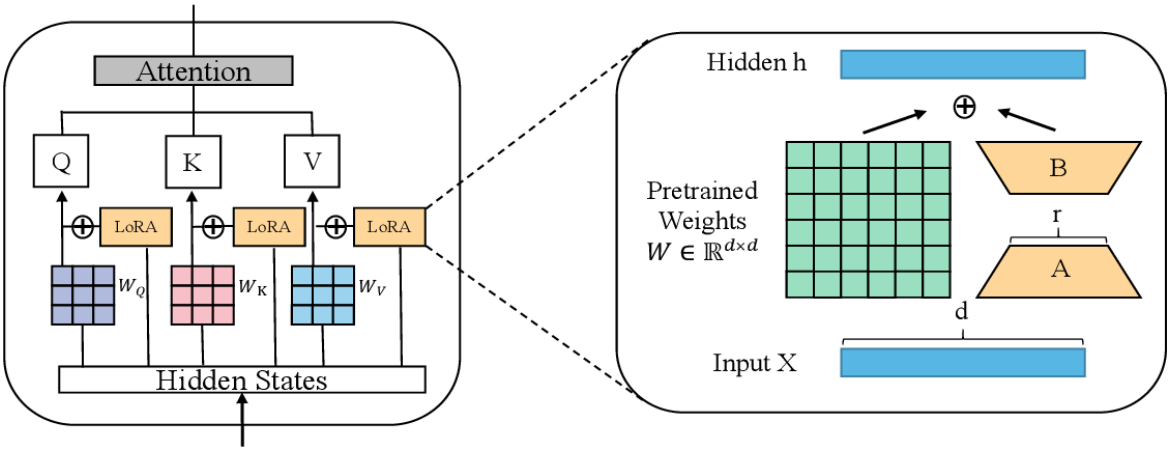


FIGURE 2.9: Low-rank adaptation incorporated in the transformer architecture using the self-attention layer LoRA configuration [71]

LoRA does not merely offer a theoretical advantage but has practical implications for model performance and efficiency. It allows for the fine-tuning of LLMs with minimal increases in parameter count, ensuring that the fine-tuned models are not only effective but also efficient and adaptable. This is particularly beneficial when deploying models in resource-constrained environments or when fine-tuning across multiple tasks, as LoRA's approach significantly lowers the hardware requirements without sacrificing model accuracy [19]. Moreover, LoRA's design is such that it introduces no additional inference latency [18]. The low-rank matrices can be seamlessly integrated with the pre-trained model's existing weights, ensuring that the enhanced model can be deployed without any compromise on performance speed [27]. This aspect is critical for applications requiring real-time processing and responsiveness.

Chapter 3

Research Methodology

This Chapter presents the research approach used in conducting the study. It outlines the step-by-step approach taken to address the research objectives and answer the research questions. Through a systematic and rigorous process, this research methodology aims to ensure the reliability, validity, and reproducibility of the findings. Each aspect of the methodology is described in detail, shedding light on the rationale behind the chosen techniques and tools. Moreover, potential limitations are also addressed to uphold the integrity of the study.

3.1 Research Design

In this section, we provide an outline detailing the implementation process for all the models under consideration. The steps taken are as follows:

1. Dataset preprocessing: Preprocess the dataset by tokenizing and encoding the phrases, and padding, adding attention masks and converting the sentiment labels into numerical values.

2. Dataset splitting: Split the preprocessed dataset into training, validation, and test sets.

3. LLM fine-tuning using PEFT techniques: Initialize the LLMs with the pre-trained weights and incorporate PEFT techniques for model adaptation.

4. Benchmarking using established adaptation methods: LLMs fine-tuning using full parameter fine-tuning, model head (last layer) fine-tuning and zero-shot learning (no fine-tuning).

5. Evaluation: Evaluate the efficiency of the fine-tuning process, and the model performance using evaluation metrics.

6. Analysis and Iteration: Analyse the results and performance of the fine-tuned LLMs. Iterate and further fine-tune the models, performing hyperparameter tuning.

3.2 Data and Data Preprocessing

3.2.1 Data Description

Knowledge-intensive tasks heavily rely on general real-world knowledge and domain-specific expertise and necessitate proficient memorization and effective utilization of domain knowledge. LLMs were fine-tuned using the financial phrasebank dataset for a sentiment analysis task, aiming to thoroughly assess the encoded knowledge within the LLMs. This dataset consists of sentences from financial news categorised by sentiment. The dataset serves as a benchmark to assess the effectiveness of sentiment models specifically designed for economic texts. The financial phrasebank dataset [52] is publicly available and was retrieved from the HuggingFace Hub . The dataset consists of 4846 sentences labelled as either negative, positive or neutral sentiment. The class distribution of the labels is shown in Figure 3.1. The percentage for neutral, negative and positive labels are 59%, 28% and 13% respectively. The data exhibits an imbalance in the distribution of the sentiment polarity. Stratified sampling was performed when splitting the dataset, ensuring an adequate representation of each sentiment type. Figure 3.2 shows the distribution of sequence length of sentences in the dataset. The maximum sequence length is 81 while the minimum sequence length is 2.

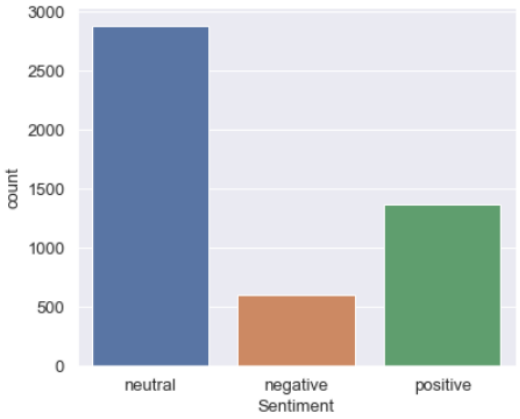


FIGURE 3.1: Sentiment polarity distribution

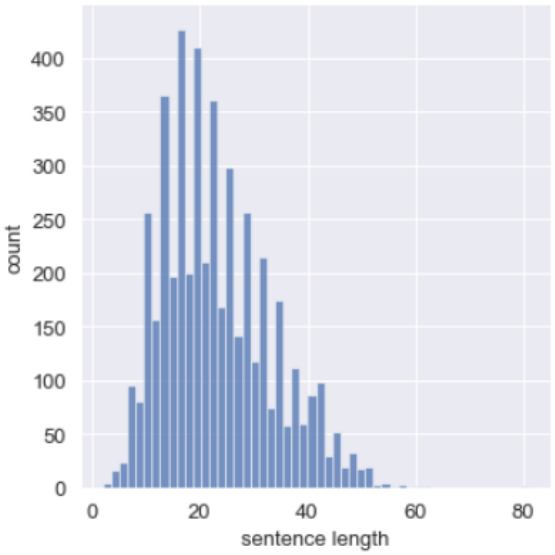


FIGURE 3.2: Sequence length distribution

3.2.2 Data Preprocessing

3.2.2.1 Tokenization and Encoding

The text in the dataset was tokenized and encoded into numerical representations that can be understood by the model. Pre-trained LLMs have their own built-in-tokenizers specifically designed to work with the model's vocabulary and architecture [23]. The tokenizers of the employed LLMs are based on the byte-pair encoding (BPE) and SentencePiece tokenizers. The BPE method breaks down texts into tokens by iteratively merging the most frequent occurring pairs of characters. SentencePiece method breaks down texts into smaller units by taking into account the position of words in a sentence and the meaning of the words [15, 79]. These tokens were then encoded into unique integers based on the model's vocabulary.

3.2.2.2 Batch processing and Padding

The sequence length of the sentences in the dataset varies as shown in Figure 3.2. The input sequences need to be of the same length to allow for parallel processing and optimised memory usage during training [76]. The self-attention mechanism in transformers is quadratically expensive in terms of the sequence length of the

inputs [43]. However, the sequence length should not be too small as the model will only be able to see a shorter sequence of tokens, which can limit its ability to learn long-range dependencies [1]. To optimize memory usage and computational efficiency, batch processing and padding were employed. A batch size of 8 was utilized, with dynamic padding employed for padding purposes. With dynamic padding the max_length hyperparameter in a batch is determined by the longest sequence in the batch. If the length of a sequence subceeds the max_length, it will be padded with special tokens to reach the max_length.

3.2.2.3 Attention Mask

Attention masks were added to the padded sequences to guide the model's attention mechanism. Attention masks are binary masks that indicate which tokens should be attended to and which tokens should be ignored. Given that we have input sentences with different sequence length, attention masks help identify actual content of each sequence while distinguishing it from the padding tokens used to equalize the sequence lengths.

3.2.2.4 Sentiment Label Encoding

The financial phrasebank dataset has categorical sentiment labels. The categorical labels were converted into unique integers in line with the vocabulary of the model.

3.3 Methods

3.3.1 Training, Validation and Test Split

The dataset was split into 70% training set, 15% validation set, and 15% testing set. The aforementioned split ratio is commonly used for evaluating model performance and efficiency. The training set was used for fine-tuning the pre-trained model, the validation set was used for hyperparameter tuning and early stopping, and the test set was used to evaluate the final performance. Hyperparameter tuning aims to optimise the model's generalization and improve its performance on unseen data. The early stopping technique safeguards against overfitting of the

model to the training data by halting the training process when the model no longer demonstrates improvement on the validation data.

3.3.2 Pre-trained Models

The rapid expansion of the LLM ecosystem provides us with an abundance of options when seeking to adapt an LLM to a downstream task. The choice of pre-trained LLM significantly impact the performance of parameter-efficient adaptation methods [80]. This can make it difficult to find the best settings for your dataset and task. The suitability of an LLM to perform a particular task is determined by various factors, including its model architecture, pre-training objective and data, and model size [76]. To effectively adapt LLMs for downstream tasks it is essential to comprehend their capabilities and limitations across various NLP tasks. While many LLMs are constructed using the decoder-only transformer architecture, there is currently no formal study that provides theoretical evidence of its superiority over other transformer architectures. Considering architectural diversity is crucial as it helps in understanding how different model designs interact with parameter efficient fine-tuning methods and how these interactions impact the model's performance when adapted to a specific task [80].

Decoder-only transformer-based LLMs are generative LLMs as they are pre-trained using the autoregressive language model (ALM) approach, in which the models are trained by predicting the subsequent word in a sequence based on the preceding words. Generative LLMs are designed to understand the overall data distribution and perform well at natural language generation (NLG) tasks. These models learn the joint probability distribution of both the input data and the target output, making decoder-only transformer models simple but powerful [79].

While decoder-only transformer-based architectures have their benefits, encoder-only LLMs have also found success in many NLP tasks. Unlike decoder-only LLMs which are generative models, encoder-only models are discriminative models pre-trained using the masked language model (MLM) approach where masked tokens

in a sequence are predicted while considering the surrounding context [23]. Discriminative LLMs are designed to learn the conditional probability distribution of the output variables given the input variables [22]. Encoder-only LLMs are not text-to-text models like decoder-only models, and therefore can only produce a vector representation capturing the meaning and context for each word in a given text. The MLM training paradigm allows the model to gain a deeper understanding of the connections between words and their contextual usage, making them ideal for natural language understanding (NLU) tasks such as text classification [80]. Yang et al. [76] observed that models trained using the masked language model training excel on natural NLU tasks as compared to most autoregressive language models of equivalent scale.

The encoder-decoder transformer-based models are mainly pre-trained using the MLM objective. They are trained to learn both joint and conditional distributions over the input and output text as they are composed of both the encoder and decoder [64]. What sets the encoder-decoder transformer apart from encoder-only and decoder-only models is its capability to learn both joint and conditional distributions over training inputs, contributing significantly to its remarkable power. The encoder-decoder LLMs are therefore both a NLU and NLG model. Raffel et al. [64] found that employing a encoder-decoder transformer model architecture yielded good results in both NLU and NLG tasks.

Due to the limited existing literature that compares the different transformer-based LLM architectures and their pre-training strategies for parameter-efficient adaptation to specific tasks, we conducted a comparative analysis using LLMs built upon the three distinct transformer architectures. The selection of LLMs was based on several criteria, including performance on benchmarks such as the Massive Multitask Language Understanding (MMLU) and General Language Understanding Evaluation (GLUE), open-source licensing allowing commercial use, fine-tunability, model size, and out-of-the-box quality. The LLMs employed are Robustly Optimized BERT approach (RoBERTa) [50], Fine-tuned LAnguage Net Text-To-Text Transfer Transformer (FLAN-T5) [15] and Open Pre-trained Transformers (OPT) [79]. Motivated by the study conducted by Ding et al. [20], that found that applying lightweight adaptation methods to smaller scale LLMs can result in performance

levels comparable or even superior to that of large scale LLMs. We used smaller versions of the LLMs also taking into account our limited computational resources. Table 3.1 gives a summary of the LLMs that were employed, showing their model size, architecture, training objective, vocabulary size, context length and dataset size used for pre-training. The capacity of an LLM is a dynamic interplay between these factors. The models fall within a similar scale range, and their pre-training utilised a diverse generic dataset consisting of billions of tokens making them general-purpose models.

TABLE 3.1: Comparative overview of hyperparameters utilized in the pre-training of RoBERTa-large, FLAN-T5-base and OPT-350 models

	RoBERTa-large	**FLAN-T5-base**	**OPT-350**
Model size	355M	248M	350M
Model Architecture	encoder-only	encoder-decoder	decoder-only
Pre-training strategy	MLM	MLM	ALM
Pre-training data size	160B tokens	34B tokens	180B tokens
Vocabulary size	50K	32K	50K
Context length	512	512	2048

3.3.3 Incorporating PEFT Techniques in Fine-Tuning LLMs

The rapid development of LLMs has led to extensive research of efficient adaptation strategies for transformer-based language models. Numerous lightweight adaptation methods have been proposed as shown in Figure 2.5. To date, there still exists a void in comprehensive investigation regarding the impact of various efficient tuning methods on LLMs pre-trained under diverse settings. Although PEFT methods have demonstrated effectiveness, the essential factors for achieving success and the relationship between different methods remain poorly comprehended, and the interconnections among these methods remain ambiguous. Lee et al.[37] found that the performance of PEFT methods varies across different distribution shifts between the source and the target domains. Additionally, the study [37] revealed that an increase in the count of trainable parameters does

not necessarily lead to improved performance. Given the aforementioned, we employed four PEFT methods, each introducing distinct modifications to the transformer architecture. The PEFT methods are prompt tuning, prefix tuning, adapter tuning and low-rank adaptation (LoRA). We further explored various configurations of these methods as summarized in Table 3.2.

TABLE 3.2: PEFT methods and their configurational variants

PEFT Method	**Configuration**
Prompt Tuning	Initializing prompt with own text Random prompt initialization Initializing prompt with LLM tokens
Prefix Tuning	No reparameterization Reparameterization
Adapter	Pfeiffer adapter [61] Houlsby adapter [26] Parallel adapter [30]
LoRA	Self-attention layer LoRA Feed-forward layer intermediate weights LoRA Feed-forward layer output weights LoRA

The selected methods fall under the categories of prompt-based (prompt tuning and prefix tuning), adapter-based (adapter tuning), and reparameterization-based (LoRA) PEFT methods. Although these techniques introduce unique designs for the structure and placement of trainable parameters in pre-trained LLMs, during adaptation, only a small added subset of trainable parameters are tuned.

Prompt-based tuning methods are relatively new, inspired by the success of prompt engineering where input prompts are designed and formulated for a language model in order to guide its generation of desired responses [39]. Prompt tuning and prefix tuning add tunable prompts to the input layer and hidden layers respectively, and exclusively fine-tune these tunable prompts for downstream tasks. In contrast to prefix tuning, which appends trainable prefixes to every intermediate transformer layer, prompt tuning does not modify the internal structure of the transformer [78].

Instead, prompt tuning appends external prompt text to the input sequence to provide additional context to guide the language model's behaviour for the specific task [41].

Prompt tuning was utilized with three distinct configurations, each varying in how the appended prompt to the input sequence is initialized. First approach we initialized the prompt using our own text, we experimented with different prompts that provides context to the model about the type of task it is expected to perform. In the second approach, we used random prompt initialization. In the third approach, the prompt was initialized using the LLM's tokens, tokens are selected from the model's vocabulary in sequential order, starting from the first word in the vocabulary.

In our implementation of prefix tuning, trainable prefix vectors were added to both the Key matrix and the Value matrix in the transformer's self-attention layer. The prefix vectors were optimized either directly (no reparameterization) or indirectly (reparameterization) using a separate feed-forward network with a bottleneck architecture. The feed-forward network was discarded after updating the prefix vectors.

Adapter-based methods introduce small-scale neural modules called "adapters" into the transformer layers and solely fine-tune these adapters for model adaptation [43]. For implementing the adapter module, a bottleneck architecture was employed. This architecture initially reduces the dimension of the original feature vector and subsequently restores it to its original dimension. This enables their insertion into the network without requiring any extra alterations to the structure or parameters of the pre-trained model. The placement of adapters is an active area of research, for this work we implemented the Pfeiffer, Houlsby and Parallel adapter configurations. The Houlsby adapter configuration places an adapter in each of the core parts of the transformer architecture, one after the multi-head self-attention sublayer and the other after the feed-forward network sublayer as shown in Figure 2.8. This configuration is inspired by the work of Housby et al. [26]. The Pfeiffer adapter configuration, as proposed by Pfeiffer et al. [61], is similar to the Houlsby adapter configuration but integrates an adapter layer solely after the feed-forward

layer within each transformer sublayer. The Parallel adapter configuration, as proposed by Hu et al. [30], places two adapter layers in parallel to the multi-head self-attention sublayer and feed-forward network sublayer as illustrated in Figure 3.3. In our implementation, we sought to minimize the number of added adaptation parameters by including only one adapter, which runs parallel to the feed-forward network sublayer.

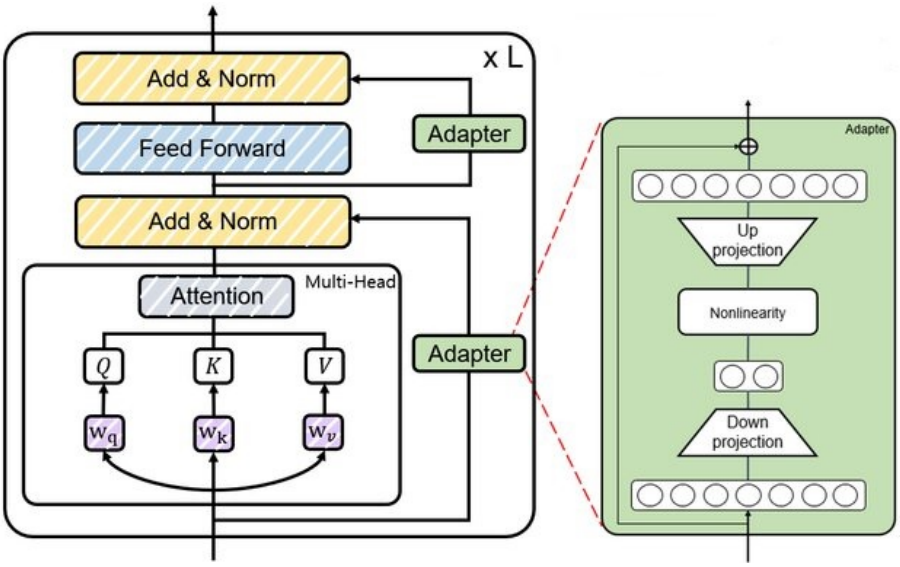


FIGURE 3.3: Parallel adapters configuration [56]

LoRA is a reparameterization-based method that enables efficient adaptation by decomposing the original weights of the pre-trained LLM into low-rank tunable matrices [27]. Trainable low-rank decomposition matrices are incorporated into the existing layers of pre-trained LLMs, while the original LLM parameters remain unchanged. Three variations of LoRA were implemented, each differing in the parameters of the transformer that are decomposed into lower ranks. In the first approach, the "Self-Attention Layer LoRA", the query and value matrices within the self-attention sublayer were decomposed, as depicted in Figure 2.9. The second implementation, "Feed-Forward Layer Intermediate Weights LoRA", involved decomposing the intermediate weights of the feedforward sublayer. In the third approach, "Feed-Forward Layer Output Weights LoRA", the output weights of the feed-forward sublayer were decomposed.

3.3.4 Model Training

Parameter-efficient fine-tuning techniques (prompt tuning, prefix tuning, adapter tuning and LoRA) were employed to fine-tune the LLMs. The models were fine-tuned by adjusting the extra added PEFT parameters using backpropagation and gradient decent while the base model was kept frozen. During training, the gradients flow through the network, and the extra added parameters get updated iteratively to minimize the loss function. Each model was trained once.

TABLE 3.3: Hyperparameters used for fine-tuning OPT model with LoRA

Hyperparameter	OPT model		
	Low-rank adaptation		
	Attention weights	FFN intermediate weights	FFN output weights
LoRA rank	18	5	41
LoRA dropout	0.1	0.1	0.1
LoRA alpha	32	32	32
Batch size	8	8	8
Epochs	3	3	4
Learning rate (LR)	0.0001227	0.00008069	0.0001569
LR Scheduler	warmup LR	warmup LR	warmup LR
Warmup ratio	0.03560	0.03778	0.08630
Optimizer	AdamW	AdamW	AdamW
Loss	cross-entropy	cross-entropy	cross-entropy
Dropout	0.1	0.1	0.1

3.3.4.1 Hyperparameter Tuning

Table 3.3 summarises the hyperparameters that were utilized to fine-tune the OPT model with LoRA. The hyperparameters for the remaining PEFT methods are available in Appendix B. Prompt tuning and prefix tuning utilize the number of virtual tokens hyperparameter, adapters employ the bottleneck dimension hyperparameter, and LoRA utilizes the LoRA rank, LoRA alpha, and LoRA dropout hyperparameters. The LoRA alpha and dropout were set at 32 and 0.1, respectively, in line

with the recommendations from the original LoRA paper [27]. To identify the optimal hyperparameter configurations for training the LLMs, we utilized the HyperOpt Tree of Parzen Estimators (TPE) search algorithm in conjunction with an Asynchronous Successive Halving (ASHA) trial scheduler. The HyperOpt algorithm systematically explores the hyperparameter space, searching for the best combinations by examining more hyperparameters and wider ranges. Meanwhile, the ASHA scheduler has the capability to terminate, pause, or adjust the hyperparameters of ongoing trials as needed. Figure 3.4 illustrates a virtual representation of the hyperparameter search process for fine-tuning the OPT model using LoRA. Fractional GPU utilization was employed to concurrently execute multiple trials on a single GPU, thereby conserving resources.

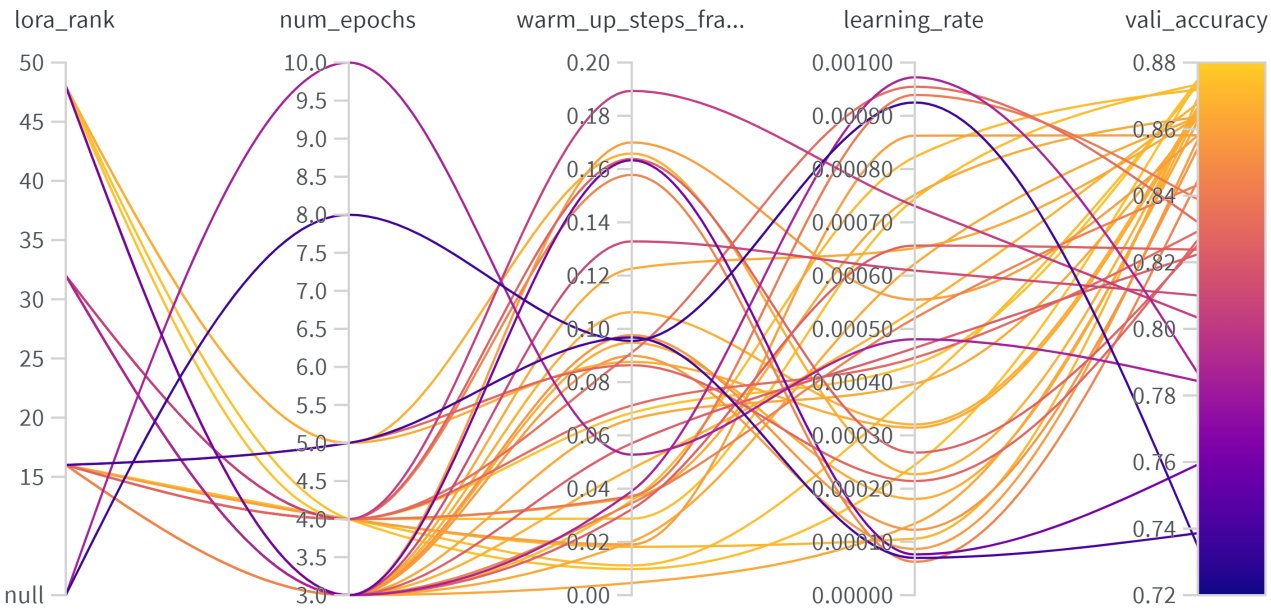


FIGURE 3.4: Hyperparameter search using HyperOpt algorithm with ASHA scheduler

3.3.5 Hardware, Libraries and Software

The model training and analysis were conducted in Python 3, utilizing Google Colab powered by an Intel Xeon CPU (13 GB RAM) and a NVIDIA Tesla K80 GPU (12 GB VRAM). All models were implemented using open-source transformers and

PyTorch libraries. The transformers library facilitated dataset loading, pre-trained model loading, and data tokenization. PyTorch was employed for model training. Model performance evaluation utilized scikit-learn, while pandas was utilized for data manipulation. Visualizations were created using seaborn and matplotlib. The various PEFT methods were implemented using Peft, opendelta, and adapter-transformers libraries. Hyperparameter tuning was performed using the Ray Tune library, while artifacts were logged using the weights and biases library.

3.4 Analysis

3.4.1 Benchmark

To assess the effectiveness of fine-tuning LLMs using PEFT techniques, a benchmark for comparison was established. We conducted tests using established fine-tuning methods and traditional NLP algorithms also presenting outcomes obtained from previous studies. Table 3.4 outlines the benchmark methods utilized. Full fine-tuning represents the current standard for LLM adaptation, and we leveraged this approach as the baseline method. Additionally, we employed model head fine-tuning, focusing on adapting only the last layer of the model. This method is known for its efficiency, as it updates only a few parameters of the model. Furthermore, we conducted out-of-the-box usage of the LLM by performing inference without any fine-tuning in a zero-shot setting, also referencing existing literature for comparison. This approach provides insight into the extent to which pre-training enhances the performance of our model in the baseline configuration. Lastly we presented results from prior studies that employed traditional NLP algorithms (LSTM), aiming to ascertain whether it is justifiable to utilize LLMs considering the significant costs associated with fine-tuning them. Insights from prior research were instrumental in evaluating and contextualizing the results obtained in this study.

Benchmark Methods
Full fine-tuning
Model head (last layer) fine-tuning
No fine-tuning (zero-shot learning)
Traditional NLP algorithms (LSTM)

TABLE 3.4: Benchmark methods

3.4.2 Evaluation Metric

In parameter-efficient fine-tuning of LLMs, there is a trade-off between efficiency and modelling performance. Achieving a balance between these factors ensures that the adapted models maintain efficiency without compromising on modelling performance. In evaluating the modelling performance of the models, accuracy and F1-scores were computed. In evaluating the computational efficiency of training the models, the following evaluation metrics were used: number of trainable parameters, storage required to save the adaptation parameters, training time and convergence rate, inference latency and peak GPU memory usage. Table 3.5 summarizes the evaluation metrics employed.

TABLE 3.5: Modelling performance and training efficiency evaluation metrics.

Modelling Performance	Training Efficiency
Accuracy	Number of trainable parameters
F1-score	Storage size of trainable parameters
	Training time
	Inference latency
	Convergence rate
	Peak GPU memory usage

3.5 Limitations

The research methodology employed in this study is subject to certain limitations. It is worth noting that the scale of the model can significantly impact the performance of PEFT techniques. Given the limited resources, the implementation of

larger and more computationally intensive LLMs is not feasible. As a result, the study primarily focused on utilizing smaller variants of the models, using smaller batch sizes and maximum sequence length to overcome hardware limitations and ensure efficient experimentation. Small-scale LLMs still offer valuable insights and can demonstrate the general capabilities of the models, it is important to acknowledge that their performance might not fully represent the potential of large-scale LLMs. It is crucial to consider the limitations arising from hardware constraints when interpreting and generalizing the findings of this study.

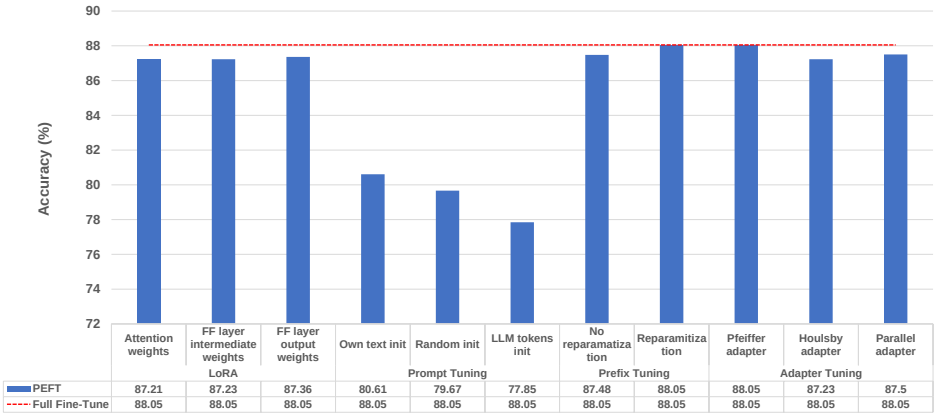
Chapter 4

Results and Discussion

This chapter presents the results of our investigation into PEFT methods. We present the results of various aspects of PEFT methods, including modeling performance, training speed, optimal trainable parameters and their storage size, convergence rate, inference latency, and GPU memory consumption, providing a detailed discussion of their performance characteristics. Additionally, the chapter presents results from literature of the scalability of LLMs and their emergent capabilities for zero-shot learning, comparing the performance of small-scale fine-tuned LLMs with large-scale models in specialized domains. Furthermore, it investigates the effectiveness of model head fine-tuning as a simplified adaptation method in NLP tasks. Lastly we compared the performance of traditional NLP algorithms with LLMs. We conclude the chapter with a concise summary of the attained results.

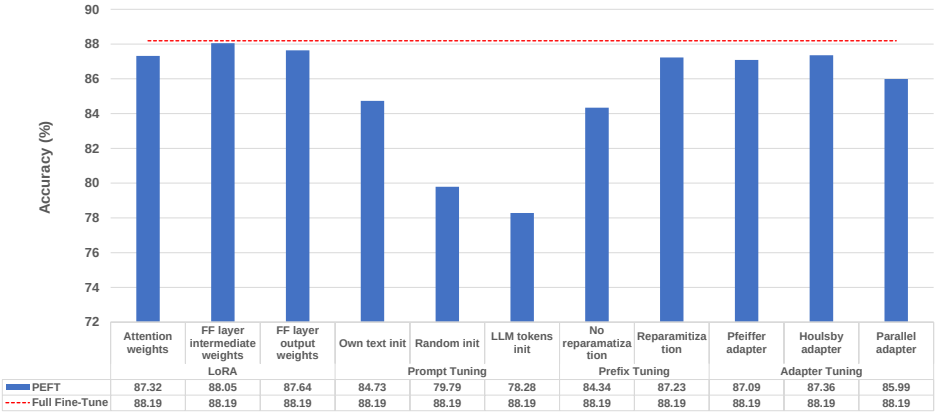
4.1 Knowledge Transfer Efficacy of LLMs through PEFT techniques

Figure 4.1 illustrates the modeling performance of the employed PEFT methods and their various configurations in terms of accuracy across three different models, with full fine-tuning serving as the baseline for comparison. Evaluation of each PEFT method's performance on RoBERTa, FLAN-T5 and OPT models revealed consistent performance, ranging from a minimum accuracy of 78% to a maximum of 89%. This suggests the effectiveness of PEFT methods in transferring knowledge during the fine-tuning process of LLMs pre-trained using different strategies in terms of transformer architectures. Overall, the OPT model showed the highest average accuracy when fine-tuned using the different PEFT methods, followed by



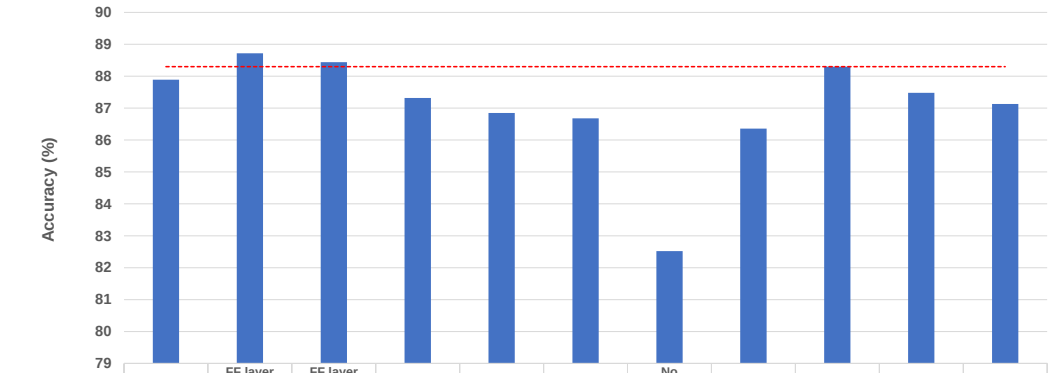
	Attention weights	FF layer intermediate weights	FF layer output weights	Own text init	Random init	LLM tokens init	No reparamatization	Reparamitization	Pfeiffer adapter	Houlsby adapter	Parallel adapter
	LoRA			Prompt Tuning			Prefix Tuning		Adapter Tuning		
PEFT	87.21	87.23	87.36	80.61	79.67	77.85	87.48	88.05	88.05	87.23	87.5
Full Fine-Tune	88.05	88.05	88.05	88.05	88.05	88.05	88.05	88.05	88.05	88.05	88.05

(A) RoBERTa model



	Attention weights	FF layer intermediate weights	FF layer output weights	Own text init	Random init	LLM tokens init	No reparamatization	Reparamitization	Pfeiffer adapter	Houlsby adapter	Parallel adapter
	LoRA			Prompt Tuning			Prefix Tuning		Adapter Tuning		
PEFT	87.32	88.05	87.64	84.73	79.79	78.28	84.34	87.23	87.09	87.36	85.99
Full Fine-Tune	88.19	88.19	88.19	88.19	88.19	88.19	88.19	88.19	88.19	88.19	88.19

(B) FLAN-T5 model



	Attention weights	FF layer intermediate weights	FF layer output weights	Own text init	Random init	LLM tokens init	No reparamatization	Reparamitization	Pfeiffer adapter	Houlsby adapter	Parallel adapter
	LoRA			Prompt Tuning			Prefix Tuning		Adapter Tuning		
PEFT	87.89	88.72	88.44	87.32	86.85	86.68	82.52	86.36	88.3	87.48	87.13
Full Fine-Tune	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3	88.3

(C) OPT model

FIGURE 4.1: Modelling performance of PEFT methods and their various configurations with full fine-tuning as the baseline

RoBERTa and then FLAN-T5, in line with the combined model size and pre-training dataset size of each respective model. The results suggest that, contrary to a widely held belief, discriminative models do not always outperform generative models in natural language understanding tasks. The observed trend suggests that the LLM's capacity and its ability to transfer knowledge are primarily determined by its model size and the scale of the pre-training data, with the pre-training strategy having a less significant impact in terms of architecture. However, it is important to note that this conclusion is based on the results of only three model architectures, and further research is needed to generalize these observations.

4.2 Comparing the Modeling Performance of PEFT Methods

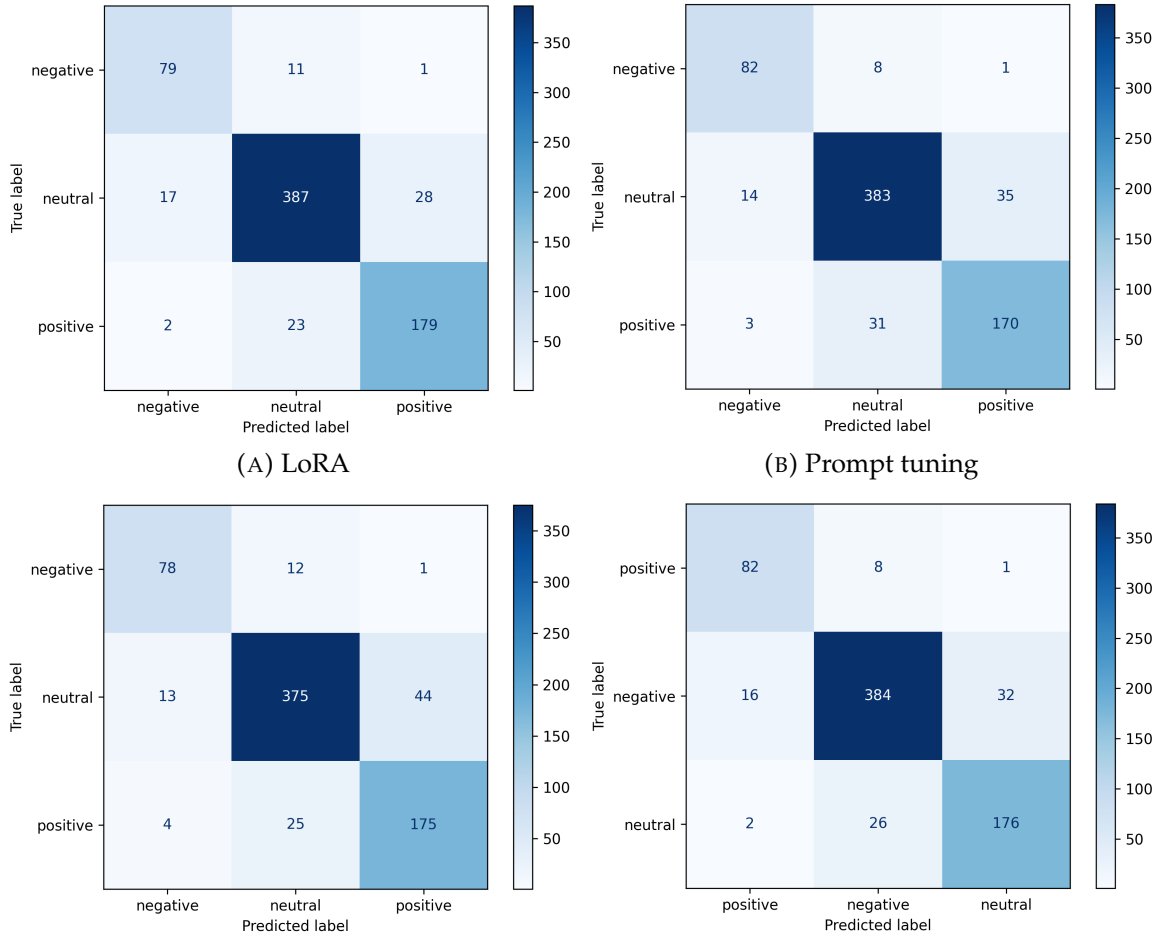
LoRA emerged as the top-performing PEFT method in terms of modelling accuracy, surpassing full fine-tuning for both feed-forward layer intermediate weights and feed forward layer output weights configurations when applied to the OPT model. Drawing from our review of existing literature, LoRA is almost always applied using the self-attention weights LoRA configuration which was proposed in the original LoRA paper [27]. However, our findings strongly advocate for its utilization using the feed-forward layer intermediate weights. This deviation from the conventional approach aligns with the common understanding in deep learning that early layers capture generic linguistic patterns, while later layers specialize in task-specific features. Indeed, both feed-forward layer intermediate weights and feed-forward layer output weights LoRA configurations outperform the self-attention weights configuration. Both these methods inject LoRA modules in the later layers to the model head.

Adapter tuning emerged as the second-best performing method, matching the performance of full fine-tuning when utilizing the Pfeiffer adapter configuration on both the RoBERTa and OPT models. Notably, the Pfeiffer adapter configuration outperforms the pioneering Houlsby adapter configuration, which originally introduced the concept of adapters. Unlike the Houlsby adapter, which inserts an adapter module after both the self-attention layer and the feed-forward layer, the

Pfeiffer adapter inserts the module solely after the feed-forward layer. This strategic placement of the adapter module exclusively at the top layers proves more beneficial, as observed with LoRA as well. Injecting adapter modules solely at the higher layers yields superior results compared to distributing them across both early and top layers, which can potentially lead to diminished performance. The results further emphasize the intuition that lower layers capture lower general features that are shared among tasks, while the later layers develop task-specific features. The parallel adapter configuration falls short of both the Houlsby and Pfeiffer adapter configurations in terms of modelling performance, highlighting the advantage of applying adapters in a serial manner.

Prefix tuning secured the third position, achieving the same performance as full fine-tuning when employing the reparameterization prefix tuning configuration on the RoBERTa model. Prefix-tuning prepends task-specific trainable prefix vectors to the key matrix and value matrix in the self-attention layer. Considering the self-attention layer as the foundational element of the transformer architecture, which provides it with the capability to capture intricate relationships and nuances within data, the incorporation of prefix modules in the self-attention layer's matrix weights proves to be an efficient strategy for enhancing LLMs. It is more effective to carry out the optimization of prefix parameters through an independent feed-forward network (reparameterization), rather than directly adjusting the soft prompts, to avoid instability and performance degradation. Once the prompts are updated, the feed-forward network is then discarded.

The performance of prompt tuning fell behind that of the other PEFT methods. Notably, initializing the prompt with a custom-designed prompt yielded better results compared to other prompt tuning configurations. The subpar performance of prompt tuning was expected, as this method introduces no alterations to the internal structure of the transformer but merely appends tokens to the input sequence. These additional tokens solely guide the model in generating the output. Consequently, prompt tuning with random prompt initialization and initialization using tokens from the model's vocabulary proved ineffective in steering the model towards generating the desired output. Both approaches seemed to cause confusion within the model, resulting in poorer performance.



(A) LoRA

(B) Prompt tuning

(C) Prefix tuning

(D) Adapter tuning

FIGURE 4.2: Confusion matrix generated for each PEFT method when applied to the OPT model

Appendix A presents the modeling performance of our PEFT methods when applied to LLMs, focusing on the F1-score metric. The observed trend mirrors that of modeling accuracy, underlining the consistency and reliability of our findings. Notably, in handling imbalance inherent in our dataset, our models demonstrate remarkable capability in classifying each class effectively without exhibiting bias towards any specific class. This equitable performance is further elucidated through the detailed examination provided by the confusion matrix showcased in Figure 4.2. In the realm of finance, where precision and recall play pivotal roles in decision-making processes concerning investments, striking a balance between these two

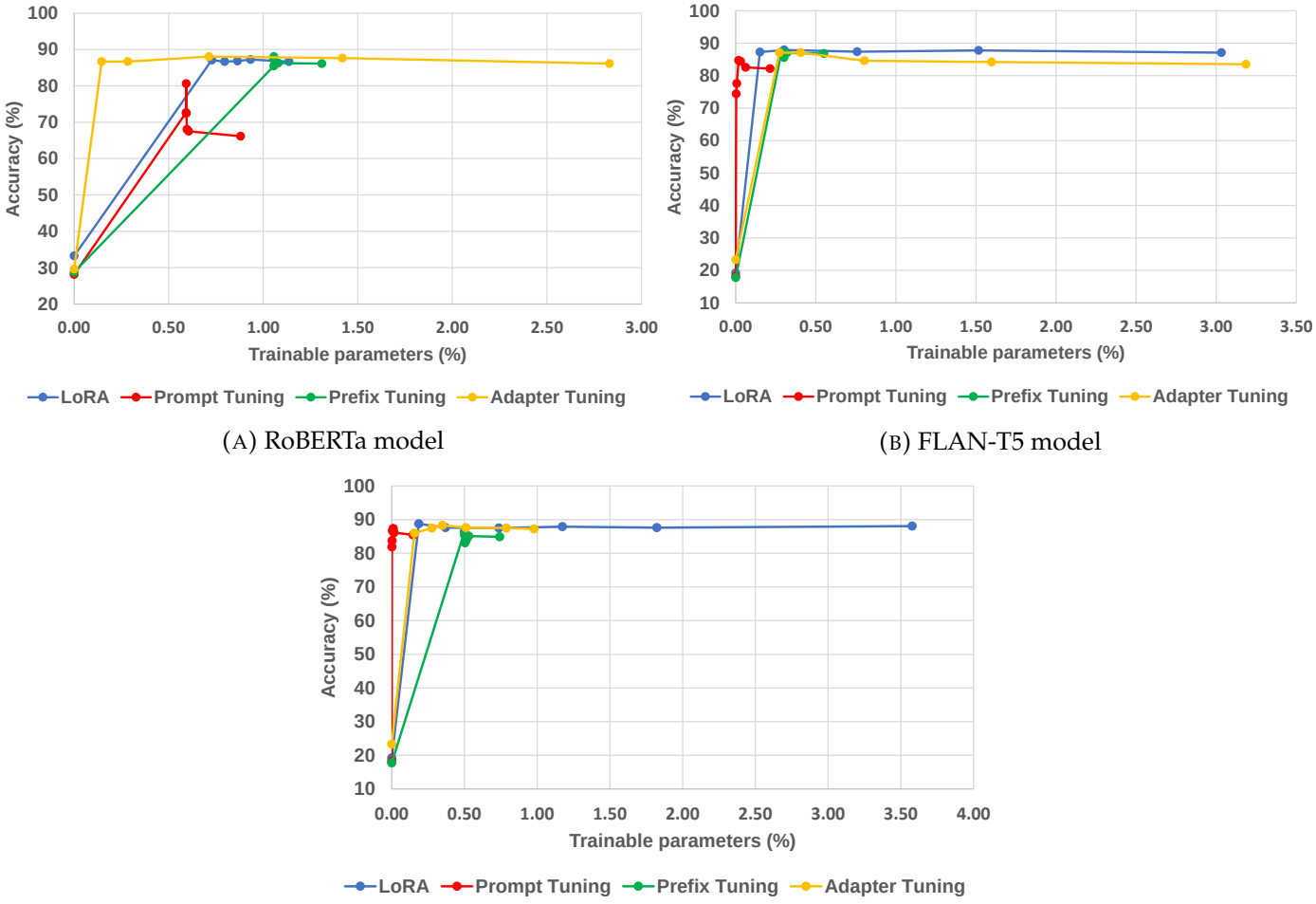
metrics is paramount. This emphasis on balanced precision and recall underscores the practical relevance and applicability of our findings in real-world financial scenarios, where accurate classification of diverse financial assets is essential for informed investment strategies.

4.3 Number of Trainable Parameters

TABLE 4.1: Comparison of trainable parameters and training time for different fine-tuning methods

Model	Method	Trainable parameters	Trainable parameters size (MB)	Training time (s)
RoBERTa	Full fine-tune	355.36M (100%)	1421.45 (100%)	458
	LoRA (FFN intermediate weights)	3.33M (0.93%)	13.34 (0.94%)	303
	Prompt tuning (Own text init)	2.11M (0.59%)	8.44 (0.59%)	779
	Prefix tuning (Reparamitization)	3.78M (1.06%)	15.12 (1.06%)	363
	Adapter tuning (Pfeiffer)	3.59M (1.00%)	14.34 (1.01%)	229
Flan-T5	Full fine-tune	296.93M (100%)	990.31 (100%)	289
	LoRA (FFN intermediate weights)	67.58M (0.30%)	101.4 (10.24%)	470
	Prompt tuning (Own text init)	46.08K (0.01%)	0.18 (0.02%)	207
	Prefix tuning (Reparamitization)	0.64M (0.002%)	101.24 (10.22%)	340
	Adapter tuning (Pfeiffer)	0.90M (0.41%)	102.31 (10.33%)	389
OPT	Full fine-tune	331.20M (100%)	1324.79 (100%)	477
	LoRA (FFN intermediate weights)	0.62K (0.19%)	2.46 (0.19%)	206
	Prompt tuning (Own text init)	32.26K (0.01%)	0.13 (0.01%)	1886
	Prefix tuning (Reparamitization)	1.66M (0.50%)	6.65 (0.50%)	600
	Adapter tuning (Pfeiffer)	1.16M (0.35%)	4.63 (0.35%)	291

Table 4.1 illustrates the characteristics of each PEFT method, including trainable parameters, storage size, and training time, focusing on the best performing configuration for each PEFT method, with full fine-tuning serving as the baseline. Across the board, the trainable parameters introduced by our PEFT methods represent a fraction ranging from 0.01% to 1.06% of the model's total parameters. Notably, LoRA applied to the OPT model excels, outperforming full fine-tuning while utilizing 0.19% of the model's total parameters. Interestingly, the relationship between the performance of PEFT methods and the number of trainable parameters is not linear. Figure 4.3 elucidates how performance tends to improve with an increase



(A) RoBERTa model

(B) FLAN-T5 model

(C) OPT model

FIGURE 4.3: Performance of PEFT methods with number of trainable parameters

in trainable parameters up to a certain threshold, after which a slight decline is observed. This phenomenon underscores the critical role of the PEFT method's architecture and the placement of PEFT modules in determining its efficacy. Notably, some PEFT methods outperform full fine-tuning due to their ability to mitigate catastrophic forgetting, a phenomenon observed during full fine-tuning of LLMs, where the model inadvertently unlearns previously acquired knowledge from pre-training, leading to overfitting on small fine-tuning datasets. In contrast, PEFT methods circumvent this issue by training only a subset of parameters while preserving the base model's integrity. To alleviate the risk of catastrophic forgetting

and prevent feature distortion during the training of our models using full fine-tuning, we restricted the number of epochs to 3.

Table 4.1 and Figure 4.3 provides insights into the relationship between the number of trainable parameters and the performance of PEFT methods. Notably, FLAN-T5 exhibited a remarkable efficiency in achieving optimal accuracy, demanding the least number of trainable parameters, with a maximum of 0.41%. In contrast, the OPT model required slightly more parameters, reaching a maximum of 0.50%,while RoBERTa necessitated the highest proportion, reaching a maximum of 1.06%. Despite FLAN-T5 utilizing fewer parameters, its performance levels were slightly below those of RoBERTa and OPT. Notably, in Figure 4.3, it is apparent that prompt tuning demands the fewest trainable parameters, yet an excess of parameters leads to model confusion and subsequent performance degradation. Conversely, prefix tuning necessitated the highest number of trainable parameters to match the performance of other PEFT methods, indicating its sensitivity to number of trainable parameters. Both LoRA and adapter tuning managed to achieve performance levels that matched or even exceeded those of full fine-tuning with a relatively modest number of trainable parameters. However, our investigation revealed that exceeding a threshold of 5% trainable parameters resulted in a significant performance decline. This phenomenon is expected as an overabundance of parameters causes the model to overly rely on the newly introduced parameters, which are trained on a smaller dataset compared to the extensive pre-training undergone by the base model, encompassing billions of tokens. This delicate trade-off between parameter quantity and performance underscores the necessity for a nuanced approach in determining the optimal configuration for PEFT methods.

4.4 Portability and Modularity of PEFT Methods

Given the minimal number of trainable parameters, PEFT methods offer notable benefits in terms of portability and modularity. They enable the creation of compact checkpoints, typically just a few megabytes in size, compared to the bulky checkpoints generated through full fine-tuning. This characteristic ensures that the trained weights from PEFT approaches are highly portable and easily deployable across various tasks without necessitating the replacement of the entire model. As

seen in Table 4.1, while full fine-tuning of the RoBERTa model yields a hefty 1.4GB checkpoint for our downstream task, PEFT methods generate only a fraction of that size for each task while maintaining comparable performance. For instance, storing the adapter module requires just 14MB disk storage, accounting for only 1.01% of the storage size needed for full fine-tuning of the RoBERTa model. This modular approach streamlines deployment, particularly when handling multiple fine-tuned models, as the same base model can be loaded once and shared across different fine-tuned modules. In contrast, traditional full fine-tuning tends to result in cumbersome, task-specific models lacking in modularity. Overall, these advantages position PEFT methods as an efficient solution for achieving modular, resource-conscious fine-tuning across diverse tasks.

4.5 Computational Efficiency

Figure 4.4 depicts the peak GPU memory consumption during the fine-tuning of LLMs using both PEFT methods and full fine-tuning. Notably, PEFT methods demonstrate significant savings, conserving up to 80% of GPU memory when fine-tuning the RoBERTa model. Similarly, savings of up to 60% and 62% are observed for the FLAN-T5 and OPT models, respectively. Among the PEFT methods, slight variations in memory efficiency were observed, with prompt tuning often demonstrating the lowest memory consumption. This consistent reduction across various models and methods underscores the broad applicability and effectiveness of PEFT approaches, indicating their critical role in optimizing computational resources while maintaining or even enhancing model performance. These savings are primarily achieved by reducing the need for gradient computation for most parameters, as they are held fixed during fine-tuning. In light of the current scarcity of GPUs and the high costs associated with accessing cloud-based GPU resources, PEFT methods play a crucial role in ensuring the computational feasibility of the fine-tuning process. The substantial reductions in GPU memory usage enable fine-tuning of large-scale LLMs even on consumer-grade hardware, thereby democratizing access to cutting-edge models, underscoring the practical importance of PEFT methods in addressing computational constraints.

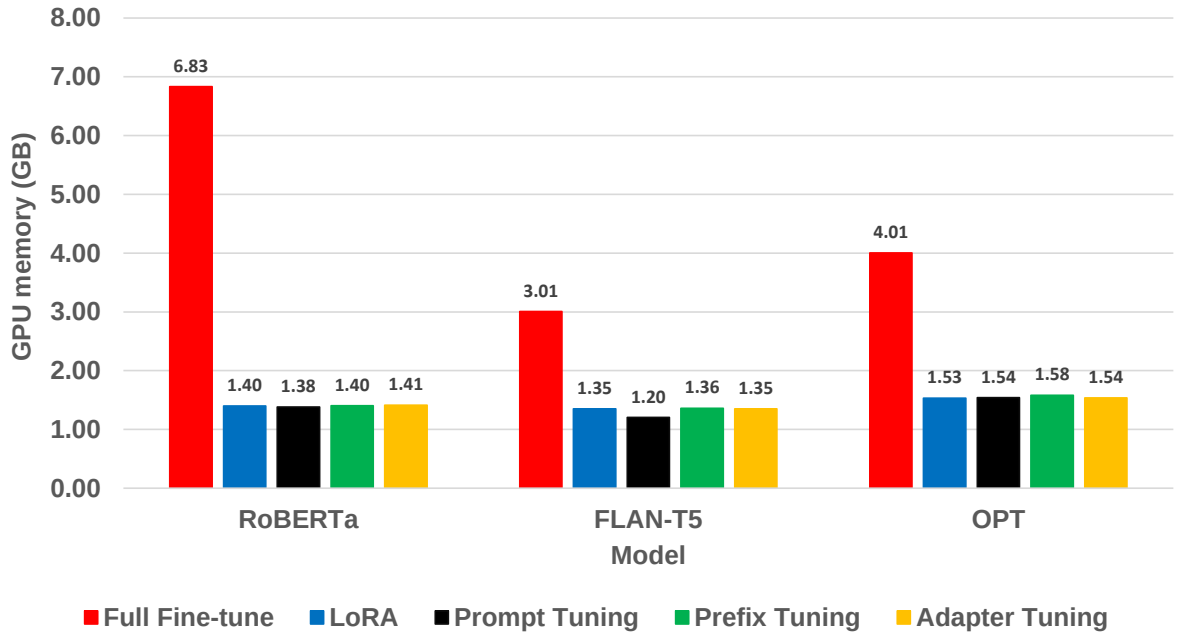
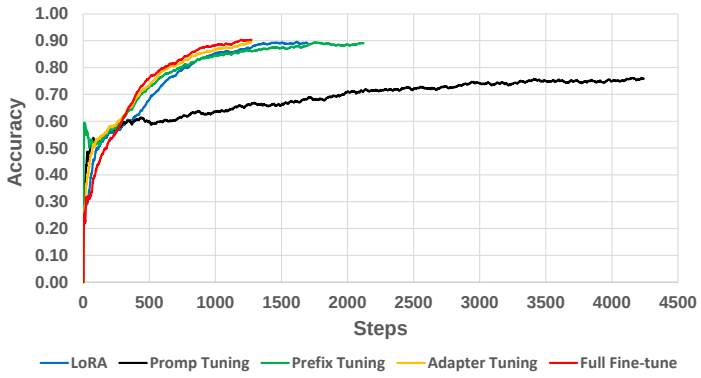


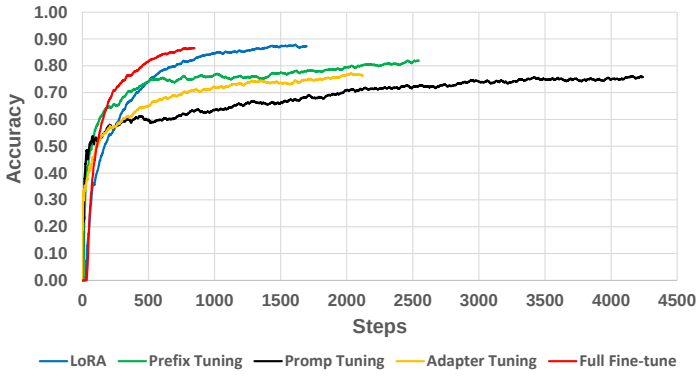
FIGURE 4.4: Peak GPU memory consumed during fine-tuning

4.6 Convergence Rate

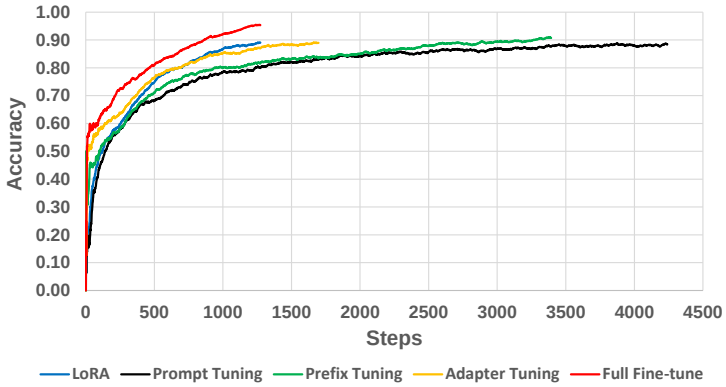
Figure 4.5 depicts the performance of various PEFT methods in comparison to full fine-tuning across different training steps, providing insights into their convergence rates. Notably, PEFT methods generally exhibit slower convergence rates compared to full fine-tuning. These methods face optimization challenges, often necessitating more training iterations to reach convergence. This could be attributed to the optimization dynamics of the PEFT methods, as these methods aim to learn downstream tasks with significantly fewer free parameters. With fewer free parameters, the dimensionality of the optimization problem is much lower, potentially limiting the directions available to decrease the loss. Additionally, this optimization difficulty may pose challenges in maintaining the robustness of PEFT methods. Among the PEFT methods, adapter tuning demonstrates the highest convergence rate, followed by LoRA, prefix tuning, and prompt tuning. Interestingly, the convergence rate appears to be less influenced by the number of tuneable parameters and more affected by the structural characteristics of the PEFT methods. The superior convergence rate observed in adapter tuning can be attributed to the internal skip-connections within adapter modules, facilitating direct information flow between



(A) RoBERTa model



(B) FLAN-T5 model



(C) OPT model

FIGURE 4.5: Convergence rate of PEFT methods with full fine-tuning as the baseline

early layers and later layers promoting gradient preservation. Similarly, the presence of a scaling factor in adapter tuning modulates the impact of adapter modules on output generation, enhancing gradient propagation and network training. Excessive emphasis on the LoRA module can slow down convergence. The stability of convergence in prefix tuning heavily relies on reparameterization, with configurations lacking this element performing inadequately. Prompt tuning lags behind other methods in terms of convergence. Despite the potential of prompt tuning, optimizing its performance remains challenging, as tokens in the input sequence often converge slower compared to other PEFT methods. Increasing the size of LLMs has been identified as a strategy to enhance the convergence rate of prompt tuning [20]. The efficiency of PEFT methods in terms of memory usage come at the expense of slower convergence rates. Therefore, it's crucial to carefully consider the trade-offs between memory optimization and convergence rates when employing PEFT

methods.

4.7 Training Speed

When comparing the training duration of PEFT methods with those of full fine-tuning, several factors come into play. At first glance, PEFT methods, designed to refine only a fraction of the model's parameters, might seem to promise shorter training periods owing to their diminished parameter space. Despite the introduction of new parameters and operations during the forward pass, the subsequent backpropagation step required to update PEFT weights proves considerably faster than updating the entire set of model parameters. This accelerated pace stems from the significantly smaller size of PEFT modules, which in turn leads to reduced gradient computation on GPUs. However, as discussed earlier, PEFT methods demonstrate slower convergence rates compared to full fine-tuning, which can extend the overall training time. As shown in Table 4.1, some PEFT methods exhibit longer training times in adapting certain LLMs compared to full fine-tuning. This is exemplified in the case of the RoBERTa model, where prompt tuning results in a lengthier training duration than full fine-tuning.

Despite the influence of slow convergence on the duration of the optimization process, we observed that the batch size significantly affects the training speed of PEFT methods. We found that PEFT methods don't necessarily offer a significant speed advantage over full fine-tuning if the reduced memory footprint isn't fully utilized by increasing the batch size. With the substantial memory savings of up to 80% as observed with PEFT methods, there's ample opportunity to increase the batch size while remaining within memory constraints, thus enhancing training throughput. Our findings underscore the significance of harnessing the diminished memory footprint achievable through augmented batch sizes to fully exploit the training speed benefits provided by PEFT methods compared to full fine-tuning. By scaling up the batch size, we can efficiently optimize the training process. This approach not only improves the efficiency of PEFT methods but also maximizes the training speed, allowing for faster convergence and reduced training times. Thus, by strategically leveraging batch size adjustments, we can unlock the full potential of PEFT

techniques in accelerating model adaptation while maintaining computational feasibility.

4.8 Inference Latency

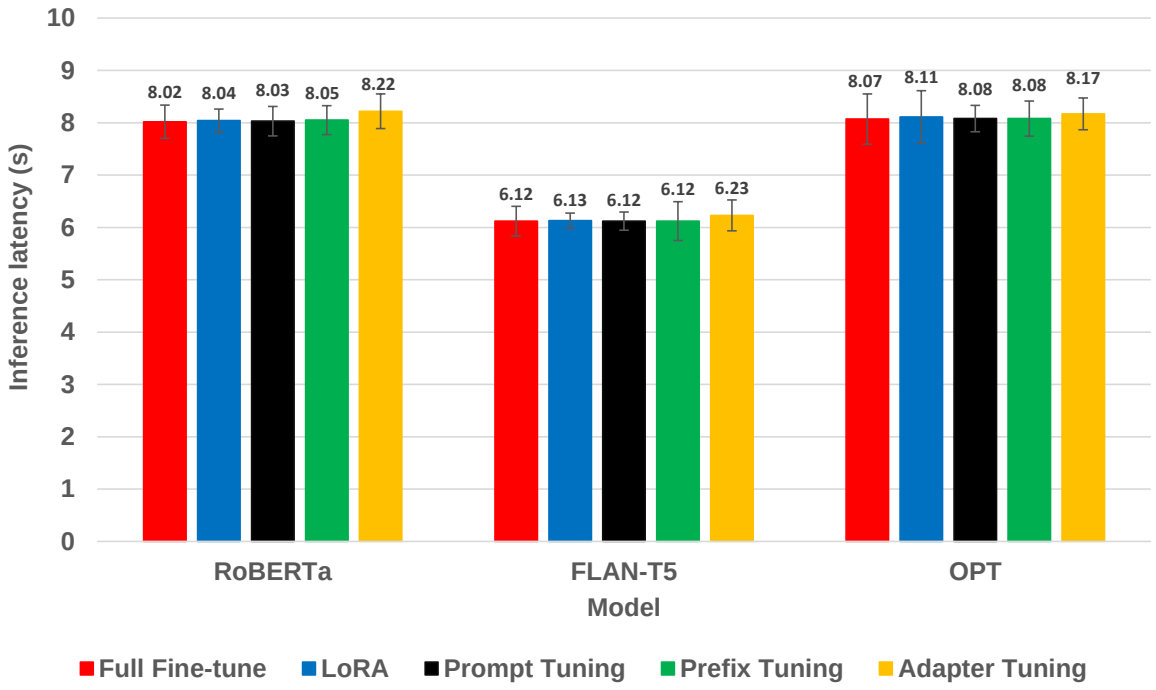


FIGURE 4.6: Inference latency using various fine-tuning methods

Network latency is a crucial consideration when deploying deep networks in real-world scenarios, where rapid inference times are paramount, often ranging from milliseconds to seconds. In assessing network latency, the focus is primarily on measuring the feed-forward operation of the network. Figure 4.6 illustrates the inference latency of the PEFT methods for the utilized LLMs, with full fine-tuning serving as the baseline. To account for variability in latency measurements, we conducted 10 trials and calculated the average runtime to improve measurement reliability. Error bars were plotted using a 95% confidence interval. PEFT methods introduce additional trainable parameters to the base model, potentially resulting in inference latency due to increased data flow paths. In industries like finance, where decisions are made within milliseconds, speed is a critical constraint for deploying

NLP systems.However, the results indicate that there is no statistically significant difference in the inference latency between the various techniques.
Notably, LoRA exhibits no increase in inference time as the LoRA modules are seamlessly merged with the base model, allowing for effective standalone usage. The adapted weights are directly used in every forward pass without passing activations through an additional module. Adapter tuning incurs a modest 1.24% to 2.49% increase in inference time, attributed to the sequential addition of adapter modules in transformer-based LLMs. Conversely, prefix tuning minimally impacts inference speed, as self-attention computation over the entire prefix is parallelized on GPUs. Prompt tuning does not introduce additional inference latency, as it does not alter the internal structure of the transformer architecture or adjust the context length of the model. Given their negligible impact on inference latency, PEFT methods present an appealing option for deployment in sectors like finance.

4.9 Scaling LLMs and Zero-shot Inference

TABLE 4.2: Modelling performance of large-scale LLMs in zero-shot inference [42]

Model	Model size	Accuracy (%)
ChatGPT [42]	175B	78
GPT-4 [42]	$\sim$ 1.2T	76

The performance of LLMs adheres to LLM scaling laws, indicating that both model size and pre-training data volume contribute to improved performance. Moreover, as LLMs scale up, they exhibit emergent capabilities for zero-shot learning. This study was confined to small-scale LLMs due to limitations in computing power. Based on the scaling laws, we expect our results to enhance when evaluated on large-scale LLMs. Considering the emergent zero-shot learning abilities of large-scale LLMs, a pertinent question arises: whether specialized models crafted through fine-tuning are necessary, or if superior performance can be achieved leveraging general-purpose large-scale LLMs like ChatGPT. To address this query, we refer to Table 4.2, showcasing results from a study conducted by Xianzhi Li et al. [42], who performed sentiment analysis using GPT-4 and ChatGPT on the financial phrase

bank dataset in a zero-shot setup. ChatGPT and GPT-4, reigning as the state-of-the-art general-purpose models, boast model sizes of 175 billion and an estimated 1.2 trillion parameters, respectively. In zero-shot inference, GPT-4 and ChatGPT achieve modeling performances of 76% and 78%, respectively. Table 4.3 outlines the performance of our small-scale LLMs in zero-shot inference, revealing that they fall short of large-scale LLMs in this regard. Scaling LLMs significantly enhances task-agnostic few-shot performance. However, upon comparing our fine-tuned models utilizing PEFT methods to current state-of-the-art large-scale LLMs, we observed that small-scale fine-tuned LLMs outperform GPT-4 and ChatGPT. Remarkably, fine-tuning less than 1% of OPT's 365 million model parameters using LoRA surpasses ChatGPT with 175 billion parameters. Therefore, in specialized domains, it's preferable to fine-tune a small-scale domain-specific LLM rather than relying on a general-purpose state-of-the-art LLM out-of-the-box. Additionally, state-of-the-art LLMs are proprietary and can only be accessed through expensive paid APIs, limiting their accessibility. Consequently, only a few organizations can afford the deployment and experimental validation costs associated with large-scale LLMs. Nevertheless, large-scale LLMs enable zero-shot transfer learning in domains where acquiring large datasets is either prohibitively expensive or the data is proprietary.

TABLE 4.3: Modelling performance of small-scale LLMs in zero-shot inference

Model	Model size	Accuracy (%)
RoBERTa	355M	28
FLAN-T5	248M	0
OPT	350M	19

4.10 Model Head Fine-tuning

Adding a task-specific head to a base model and fine-tuning only the head (last layer) is a well-established adaptation method in computer vision, known for its computational efficiency and successful application. Similarly, PEFT methods involve updating merely a handful of newly introduced parameters, mirroring this

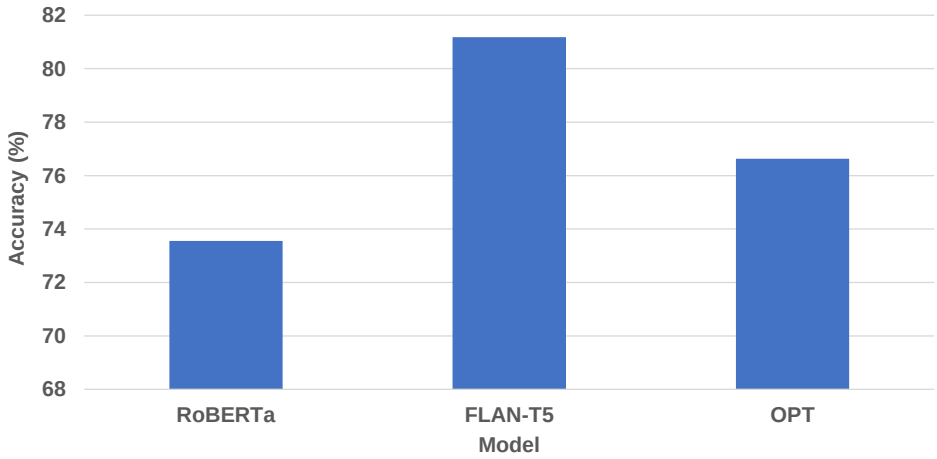


FIGURE 4.7: Model head fine-tuning

strategy. Motivated by the efficacy of this method in computer vision, we investigated whether altering the internal structure of LLMs by adding PEFT modules is necessary. Figure 4.7 illustrates the performance of our LLMs after undergoing model head fine-tuning. Notably, FLAN-T5 emerges as the top-performing model with an accuracy of 81%. While model head fine-tuning outperforms zero-shot learning, it falls short compared to full fine-tuning and PEFT methods. These results suggest that training more layers beyond just the last layer yields benefits. Overall, model head fine-tuning demonstrates promise in NLP tasks and warrants further investigation due to its simplicity and training efficiency.

4.11 Comparison of LLMs with Traditional NLP Algorithms

TABLE 4.4: The modelling performance of LSTM models and OPT model fine-tuned using LoRA

Model	Accuracy (%)	F1-score
LSTM [5]	71	0.64
LSTM [58]	72	0.66
OPT+LoRA	89	0.89

Researchers in computational linguistics have raised concerns about whether LLMs are overly complex and offer insufficient improvements to justify their heightened implementation costs when compared to simpler, traditional state-of-the-art NLP algorithms [31]. It remains uncertain whether and to what extent LLMs, fine-tuned for financial text using lightweight adaptation methods, outperform traditional NLP models in finance related tasks. Table 4.4 presents the results of sentiment analysis using the RNN-based LSTM model, which is the conventional choice for NLP tasks. By employing the OPT model fine-tuned using LoRA, we observed a 17% increase in accuracy over the LSTM model. This superior performance implies that self-supervised pre-training of LLMs, though costly, offers substantial benefits. Domain-adapted LLMs leverage pre-trained language representations to effectively capture complex patterns and nuances in financial text data. These results challenge the common belief that fine-tuning LLMs requires a large dataset, LSTMs require more data to achieve optimal performance compared to domain-adapted LLMs. PEFT methods enable efficient knowledge transfer, achieving state-of-the-art performance without distorting LLM features, even with a small training dataset of only 3392 samples. These findings hold significant practical value as the manual labeling of training samples is widely acknowledged as one of the most costliest tasks in supervised machine learning. Analysis of the F1-score reveals that LLMs exhibit robust performance in classifying financial sentiment, achieving high recall and precision, while LSTMs face challenges in capturing the nuanced interplay between positive, negative, and neutral sentiments crucial for informed investment decisions in finance.

4.12 Summary

In this chapter, we presented the results obtained and provided further insights into the findings. By employing PEFT methods for adapting a base LLM to perform sentiment analysis in the financial domain, we achieved performance levels that matched or even exceeded the gold standard of full fine-tuning. Notably, adapting the OPT model using LoRA yielded the best modeling performance with an accuracy of 89%. Remarkably, these high performance levels were attained using only a fraction of the model's total parameters, less than 1.06%. Moreover, the

storage size of the trainable parameters was minimal, ranging from just 0.1MB to 7MB for the OPT model, which underscores the high modularity of PEFT methods. PEFT modules introduced negligible inference overhead. Despite these advantages, it is worth noting that PEFT methods exhibited slower convergence rates compared to full fine-tuning, resulting in longer training times. However, the most significant benefit was observed in GPU memory consumption, where savings of up to 80% were realized, highlighting the computational efficiency of PEFT methods. The substantial saving in GPU memory usage enables us to augment batch sizes, thereby mitigating the prolonged training duration. Small-scale fine-tuned LLMs outperform state-of-the-art large-scale LLMs in specialized domains, highlighting the importance of domain-specific fine-tuning. Model head fine-tuning demonstrates promise but falls short compared to full fine-tuning and PEFT methods, suggesting that training more layers beyond just the last layer yields benefits. LLMs demonstrated superiority over conventional LSTM models by achieving a 17% increase in accuracy, thereby validating their higher implementation costs.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This report investigates the efficacy of parameter-efficient fine-tuning methods for adapting large language models for sentiment analysis of financial data, considering both modelling performance and training efficiency. Through a comprehensive analysis of different PEFT methods, including LoRA, prompt tuning, prefix tuning, and adapter, alongside traditional full fine-tuning, several key insights have emerged. Firstly, we achieved performance levels that matched or even exceeded the gold standard of full fine-tuning. Notably, adapting the OPT model using LoRA yielded the best modeling performance with an accuracy of 89%. Remarkably, these high performance levels were attained using only a small fraction of the model's total parameters, less than 1.06%. Moreover, the storage size of the trainable parameters was minimal, ranging from just 0.1MB to 7MB for the OPT model, which underscores the high modularity of PEFT methods. PEFT modules introduced negligible inference overhead. Despite these advantages, it is worth noting that PEFT methods exhibited slower convergence rates compared to full fine-tuning, resulting in longer training times. However, the most significant benefit was observed in the peak GPU memory consumption, where savings of up to 80% were realized, highlighting the computational efficiency of PEFT methods. The substantial saving in GPU memory usage enables us to augment batch sizes, thereby mitigating the prolonged training duration. Small-scale fine-tuned LLMs outperform state-of-the-art large-scale LLMs in specialized domains, highlighting the importance of domain-specific fine-tuning. Model head fine-tuning demonstrates promise but falls short compared to full fine-tuning and PEFT methods, suggesting that training more layers beyond just the last layer yields benefits. LLMs demonstrated superiority over conventional

LSTM models by achieving a 17% increase in accuracy, thereby validating their higher implementation costs.

These findings highlight the promise of PEFT methods for real-world deployment across diverse domains and underscore the need for further research to refine PEFT methodologies and explore their applicability to other tasks and domains in NLP. The advancements stemming from this research could significantly contribute to making AI more accessible and inclusive, thereby allowing both individuals and organizations with constrained computational resources to leverage AI models. Moreover, in an era increasingly attentive to environmental sustainability, PEFT contributes to minimizing the energy requirements and carbon emissions associated with training expansive AI models, aligning technological advancement with ecological responsibility.

5.2 Limitations of the Study

While this research report aimed to investigate the capabilities of fine-tuning LLMs for sentiment analysis of financial data using parameter-efficient fine-tuning techniques, there are some limitations to consider. LLMs are trained on extensive datasets and can unintentionally acquire and reproduce biases existing in the training data [46]. They can perpetuate stereotypes, reinforce discrimination, or generate offensive or harmful content [81, 8]. LLMs are regarded as opaque models because of the challenges in understanding their decision-making mechanisms. Understanding how and why LLMs generate certain outputs can be difficult, raising concerns about their trustworthiness and accountability. Spurious biases can arise during the fine-tuning process when the models learn associations between certain features and sentiment that are not relevant to the financial domain. These spurious biases can lead to misleading predictions and undermine the accuracy and reliability of the sentiment analysis. Model hallucination occurs when fine-tuned models generate outputs that are not supported by the input data [80]. LLMs, including those fine-tuned for sentiment analysis, have been known to generate plausible yet false information. These limitations highlight the need for ongoing research and vigilance to ensure that LLMs make predictions that are accurate, fair, and reliable.

Acknowledging these limitations helped ensure a comprehensive and realistic approach to this work and its outcomes.

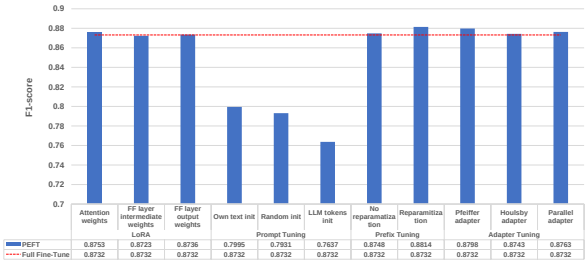
5.3 Future Work

The utilization of LLMs as a developer tool for building LLM-powered applications is still in its infancy stages, representing a rapidly evolving field. There are no established best practices, necessitating the formulation of a more structured process for building production applications with LLMs. The following are possible future research avenues to build on this work. Firstly, exploring the composability of PEFT methods by stacking, fusing, or mixing different PEFT modules to harness their combined knowledge effectively. Additionally, leveraging lower-precision floating-point representations (e.g., FP16) during training with PEFT methods has the potential to accelerate training speed, while introducing slight noise, it improves generalization and reduces overfitting. Further enhancement can be achieved through quantization techniques, converting model weights to lower-bit integer representations (e.g., 4-bit), thereby reducing memory requirements and enhancing computational efficiency. Future investigations into lightweight adaptation methods tailored specifically for LLMs hold significant promise for further improving their efficiency and capabilities in real-world applications.

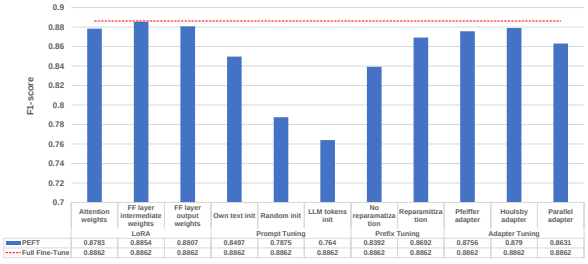
Appendix A

Effectiveness of Adaptation Methods

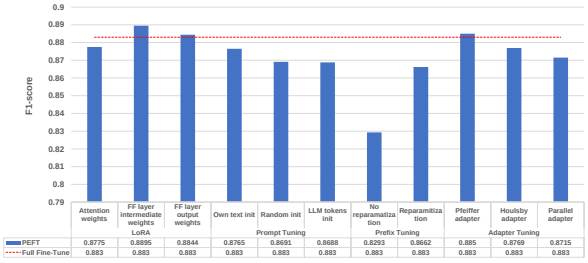
A.1 Fine-tuning results



(A) RoBERTa model



(B) FLAN-T5 model



(C) OPT model

FIGURE A.1: Modelling performance of PEFT methods and their various configurations with full fine-tuning as the baseline

TABLE A.1: Results of fine-tuning RoBERTa, FLAN-T5 and OPT models using PEFT methods

Model	Metric	Benchmark			LoRA			Prompt Tuning			Prefix Tuning		Adapter Tuning		
		No adaptation	Model head	Full fine-tune	Attention weights	FF layer intermediate weights	FF layer output weights	Own text init	Random init	LLM tokens init	No reparamatization	Reparamitization	Pfeiffer adapter	Houlsby adapter	Parallel adapter
RoBERTa	Accuracy	28.08	73.55	88.05	87.21	87.23	87.36	80.61	79.67	77.85	87.48	88.05	88.05	87.23	87.5
	F1 score	0.1441	0.7063	0.8732	0.8753	0.8723	0.8736	0.7995	0.7931	0.7637	0.8748	0.8814	0.8798	0.8743	0.8763
	Inference time	9	8	8	8	8	8	8	8	9	8	8	8	8	8
	Training time	0	590	458	306	303	323	779	682	919	379	363	229	310	212
	Trainable parameters size	0	4.21	1421.45	11.57	13.34	18.25	8.44	8.43	4.26	18.25	15.12	14.34	20.14	8.24
	Peak GPU memory	0	1434.88	6831.25	1392.93	1397.99	1412.06	1383.99	1383.97	1380.05	1438.83	1403.54	1412.93	1429.55	1395.68
Flan-T5	Accuracy	0	81.18	88.19	87.32	88.05	87.64	84.73	79.79	78.28	84.34	87.23	87.09	87.36	85.99
	F1 score	0	0.8159	0.8862	0.8783	0.8854	0.8807	0.8497	0.7875	0.764	0.8392	0.8692	0.8756	0.879	0.8631
	Inference time	11	6	6	6	6	6	6	6	7	7	6	6	6	6
	Trainable parameters size	0	98.7	990.31	103.12	101.4	104.1	0.18	0.6	0.02	0.15	101.24	102.31	113.01	102.31
	Training time	0	92	289	470	318	207	354	331	447	254	340	389	436	392
	Peak GPU memory	0	1334.16	3006.14	1351.65	1351.7	1327.05	1202.75	1173.57	1171.07	1165.51	1359.02	1352.1	1366.78	1336.5
OPT	Accuracy	19.01	76.63	88.3	87.89	88.72	88.44	87.32	86.85	86.68	82.52	86.36	88.3	87.48	87.13
	F1 score	0.1546	0.7614	0.883	0.8775	0.8895	0.8844	0.8765	0.8691	0.8688	0.8293	0.8662	0.885	0.8769	0.8715
	Inference time	9	8	8	8	8	8	17	8	14	8	8	8	8	8
	Training time	0	255	477	209	206	281	1886	359	1300	658	600	291	334	212
	Trainable parameters size	0	0.01	1324.79	7.08	2.46	20.16	0.13	0.02	0.11	11.8	6.65	4.63	8.86	8.24
	Peak GPU memory	0	1572.51	4005.12	1546.29	1531.24	1583.57	1543	1548.12	1685.89	1753.78	1580.49	1539.03	1550.4	1509.2

TABLE A.2: Trainable parameters using PEFT methods to fine-tune RoBERTa, FLAN-T5 and OPT models

Model	Metric	Benchmark			LoRA			Prompt Tuning			Prefix Tuning		Adapter Tuning		
		No adaptation	Model head	Full fine-tune	Attention weights	FF layer intermediate weights	FF layer output weights	Own text init	Random init	LLM tokens init	No reparamatization	Reparamitization	Pfeiffer adapter	Houlsby adapter	Parallel adapter
RoBERTa	Trainable params	0	1052675	355362819	2,891,782	3,334,150	4,562,950	2,110,470	2,108,422	1,064,963	4,562,950	3,780,646	3,585,227	5,035,907	2,060,771
	Non-trainable params	355362819	354310144	0	354,310,144	354,310,144	354,310,144	354,310,144	354,310,144	354,310,144	354,310,144	354,310,144	355,359,744	355,359,744	355,359,744
	Total params	355362819	355362819	355362819	357,201,926	357,644,294	358,873,094	356,420,614	356,418,566	355,375,107	358,873,094	358,090,790	358,944,971	360,395,651	357,420,515
	trainable%	0.0000	0.2962	100.0000	0.8096	0.9323	1.2715	0.5921	0.5916	0.2997	1.2715	1.0558	0.9988	1.3973	0.5766
Flan-T5	Trainable params	0	24,674,304	296,926,464	1105920	675840	1351680	46,080	15,360	6144	36,864	636704	903744	3578112	903744
	Non-trainable params	296926464	272,252,160	0	222,903,552	222903552	222903552	222903552	222903552	222903552	222903552	222903552	222903552	222903552	222903552
	Total params	296,926,464	296,926,464	296,926,464	224009472	223579392	224255232	247,623,936	247,593,216	247584000	247,614,720	223540256	223807296	226481664	223807296
	trainable%	0.0000	8.3099	100.0000	0.4961	0.3032	0.6064	0.0186	0.0062	0.0025	0.0149	0.2856	0.4054	1.6052	0.4054
OPT	Trainable params	0	1,536	331,197,952	1,771,008	615,936	5,039,616	32,256	4,096	27,136	2,950,656	1,661,472	1,157,112	2,214,384	1009632
	Non-trainable params	331197952	331,196,416	0	331,197,952	331,197,952	331,197,952	331,197,952	331,197,952	331,196,416	331,197,952	331,197,952	331,098,112	331,098,112	331,098,112
	Total params	331,197,952	331,197,952	331,197,952	332,968,960	331,813,888	336,237,568	331,230,208	331,202,048	331,223,552	334,148,608	332,859,424	332,255,224	333,312,496	332206048
	trainable%	0	0.0005	100.0000	0.5319	0.1856	1.4988	0.0097	0.0012	0.0082	0.8830	0.4992	0.3483	0.6644	0.3039

Appendix B

Hyperparameters

B.1 Hyperparameters for Fine-tune LLMs using PEFT Methods

TABLE B.1: Training parameters of RoBERTa, FLAN-T5 and OPT models using LoRA

Hyperparameter	LoRA								
	RoBERTa			FLAN-T5			OPT		
	Attention weights	FFN intermediate weights	FFN output weights	Attention weights	FF layer intermediate weights	FF layer output weights	Attention weights	FF layer intermediate weights	FF layer output weights
LoRA rank	8	10	20	10	10	20	18	5	41
LoRA dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
LoRA alpha	32	32	32	32	32	32	32	32	32
Batch size	8	8	8	8	8	8	8	8	8
Epochs	4	4	4	4	4	3	3	3	4
Learning rate (LR)	0.00005	0.00005	0.00005	0.00117	0.001	0.080003	0.000122721	0.00008069	0.000156912
LR Scheduler	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR
Warmup ratio	0.05	0.05	0.05	0.02	0.02	0.02	0.0355968	0.03777972	0.08630454
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Loss	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy
Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

TABLE B.2: Training parameters of RoBERTa, FLAN-T5 and OPT models using prompt tuning

Hyperparameter	Promt Tuning								
	RoBERTa			FLAN-T5			OPT		
	Own text init	Random init	LLM tokens init	Own text init	Random init	LLM tokens init	Own text init	Random init	LLM tokens init
Num virtual tokens	5	3	12	30	10	8	60	5	50
Batch size	8	8	8	8	8	8	8	8	8
Epochs	10	9	10	5	5	7	10	5	10
Learning rate (LR)	0.001	0.001	0.001	0.05	0.1509	0.10514659	0.000480524	0.000518052	0.000480524
LR Scheduler	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR
Warmup ratio	0.02	0.02	0.02	0.06	0.031504	0.08089	0.0527406	0.052740631	0.0527406
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Loss	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy
Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

TABLE B.3: Training parameters of RoBERTa, FLAN-T5 and OPT models using prefix tuning

Hyperparameter	Prefix Tuning					
	RoBERTa		FLAN-T5		OPT	
	No reparamatization	Reparamitization	No reparamatization	Reparamitization	No reparamatization	Reparamitization
Num virtual tokens	50	20	2	5	60	5
Batch size	8	8	8	8	8	8
Epochs	5	5	6	6	8	8
Learning rate (LR)	0.005	0.0005	0.023221869	0.023221869	0.000771015	0.000371015
LR Scheduler	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR
Warmup ratio	0	0	0.03415876	0.034158757	0.04638139	0.04638139
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Loss	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy
Dropout	0.1	0.1	0.1	0.1	0.1	0.1

TABLE B.4: Training parameters of RoBERTa, FLAN-T5 and OPT models using adapter tuning

Hyperparameter	Adapter Tuning								
	RoBERTa			FLAN-T5			OPT		
	Pfeiffer config	Houlsby config	Parallel config	Pfeiffer config	Houlsby config	Parallel config	Pfeiffer config	Houlsby config	Parallel config
Bottleneck dim	51	41	20	32	64	32	21	21	21
Batch size	8	8	8	8	8	8	8	8	8
Epochs	3	4	3	5	5	5	4	4	4
Learning rate (LR)	0.0001	0.0001	0.0001	0.000508	0.0005294	0.000409194	0.00015	0.0001	0.00012
LR Scheduler	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR	warmup LR
Warmup ratio	0.06	0.06	0.06	0.094946	0.0697265	0.04741224	0	0	0
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Loss	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy	Cross-entropy
Dropout	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Bibliography

[1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. "Intrinsic dimensionality explains the effectiveness of language model fine-tuning". In: *arXiv preprint arXiv:2012.13255* (2020).

[2] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. "What learning algorithm is in-context learning? investigations with linear models". In: *arXiv preprint arXiv:2211.15661* (2022).

[3] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S. Awwal, and Vijayan K. Asari. *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. 2018. arXiv: 1803.01164 [cs.CV].

[4] Dogu Araci. "Finbert: Financial sentiment analysis with pre-trained language models". In: *arXiv preprint arXiv:1908.10063* (2019).

[5] Dogu Araci. *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models*. 2019. arXiv: 1908.10063 [cs.CL].

[6] Sudhandar Balakrishnan, Yihao Fang, and Xioadan Zhu. "Exploring Robustness of Prefix Tuning in Noisy Data: A Case Study in Financial Sentiment Analysis". In: *arXiv preprint arXiv:2211.05584* (2022).

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems* 24 (2011).

[8] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258* (2021).

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[10] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. "Large language models as tool makers". In: *arXiv preprint arXiv:2305.17126* (2023).

[11] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. *A Survey on Evaluation of Large Language Models*. 2023. arXiv: 2307.03109 [cs.CL].

[12] Lingjiao Chen, Matei Zaharia, and James Zou. "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance". In: *arXiv preprint arXiv:2305.05176* (2023).

[13] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. "Palm: Scaling language modeling with pathways". In: *arXiv preprint arXiv:2204.02311* (2022).

[14] Alexandra Chronopoulou, Christos Baziotis, and Alexandros Potamianos. "An embarrassingly simple approach for transfer learning from pretrained language models". In: *arXiv preprint arXiv:1902.10547* (2019).

[15] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. "Scaling instruction-finetuned language models". In: *arXiv preprint arXiv:2210.11416* (2022).

[16] Leonardo Colacicchi. "Comparison and fine-tuning of methods for Financial Sentiment Analysis". In: (2022).

[17] Naeem Khatieb Dalika and Yudhvir Seetharam. "Sentiment and returns: Analysis of investor sentiment in the south African market". PhD thesis. University of the Witwatersrand, Faculty of Commerce, Law and Management, 2014.

[18] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. "Qlora: Efficient finetuning of quantized llms". In: *arXiv preprint arXiv:2305.14314* (2023).

[19] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models". In: *arXiv preprint arXiv:2203.06904* (2022).

[20] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. "Parameter-efficient fine-tuning of large-scale pre-trained language models". In: *Nature Machine Intelligence* 5.3 (2023), pp. 220–235.

[21] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. "Towards a unified view of parameter-efficient transfer learning". In: *arXiv preprint arXiv:2110.04366* (2021).

[22] Pengcheng He, Jianfeng Gao, and Weizhu Chen. *DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing*. 2021. arXiv: 2111.09543 [cs.CL].

[23] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. "DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=XPZIaotutsD.

[24] Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. *On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation*. 2021. arXiv: 2106.03164 [cs.CL].

[25] 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. "Training compute-optimal large language models". In: *arXiv preprint arXiv:2203.15556* (2022).

[26] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. "Parameter-efficient transfer learning for NLP". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2790–2799.

[27] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685* (2021).

[28] Shengding Hu, Ning Ding, Weilin Zhao, Xingtai Lv, Zhen Zhang, Zhiyuan Liu, and Maosong Sun. "OpenDelta: A Plug-and-play Library for Parameter-efficient Adaptation of Pre-trained Models". In: *arXiv preprint arXiv:2307.03084* (2023).

[29] Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. "LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models". In: *arXiv preprint arXiv:2304.01933* (2023).

[30] Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, Xing Xu, and Soujanya Poria. *LLM-Adapters: An Adapter Family for Parameter-Efficient Fine-Tuning of Large Language Models*. 2023. arXiv: 2304.01933 [cs.CL].

[31] Allen H Huang, Hui Wang, and Yi Yang. "FinBERT: A large language model for extracting information from financial text". In: *Contemporary Accounting Research* 40.2 (2023), pp. 806–841.

[32] Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, et al. *Camels in a Changing Climate: Enhancing LM Adaptation with Tulu 2*. 2023. arXiv: 2311.10702 [cs.CL].

[33] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[34] Yova Kementchedjhieva and Ilias Chalkidis. "An Exploration of Encoder-Decoder Approaches to Multi-Label Classification for Legal and Biomedical Text". In: *arXiv preprint arXiv:2305.05627* (2023).

[35] Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. "General-purpose in-context learning by meta-learning transformers". In: *arXiv preprint arXiv:2212.04458* (2022).

[36] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. "Fine-tuning can distort pretrained features and underperform out-of-distribution". In: *arXiv preprint arXiv:2202.10054* (2022).

[37] Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. "Surgical fine-tuning improves adaptation to distribution shifts". In: *arXiv preprint arXiv:2210.11466* (2022).

[38] Matteo Lengkeek, Finn van der Knaap, and Flavius Frasincar. "Leveraging hierarchical language models for aspect-based sentiment analysis on financial data". In: *Information Processing & Management* 60.5 (2023), p. 103435.

[39] Brian Lester, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning". In: *arXiv preprint arXiv:2104.08691* (2021).

[40] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[41] Xiang Lisa Li and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation". In: *arXiv preprint arXiv:2101.00190* (2021).

[42] Xianzhi Li, Xiaodan Zhu, Zhiqiang Ma, Xiaomo Liu, and Sameena Shah. "Are ChatGPT and GPT-4 General-Purpose Solvers for Financial Text Analytics? An Examination on Several Typical Tasks". In: *arXiv preprint arXiv:2305.05862* (2023).

[43] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. "Scaling down to scale up: A guide to parameter-efficient fine-tuning". In: *arXiv preprint arXiv:2303.15647* (2023).

[44] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv preprint arXiv:1807.05118* (2018).

[45] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 1950–1965.

[46] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. "Understanding the difficulty of training transformers". In: *arXiv preprint arXiv:2004.08249* (2020).

[47] Ruijun Liu, Yuqian Shi, Changjiang Ji, and Ming Jia. "A survey of sentiment analysis based on transfer learning". In: *IEEE access* 7 (2019), pp. 85401–85412.

[48] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks". In: *arXiv preprint arXiv:2110.07602* (2021).

[49] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. "GPT understands, too". In: *arXiv preprint arXiv:2103.10385* (2021).

[50] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[51] Tim Loughran and Bill McDonald. "Textual analysis in finance". In: *Annual Review of Financial Economics* 12 (2020), pp. 357–375.

[52] P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala. "Good debt or bad debt: Detecting semantic orientations in economic texts". In: *Journal of the Association for Information Science and Technology* 65 (2014).

[53] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul. *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*. https://github.com/huggingface/peft. 2022.

[54] Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney. "What happens to bert embeddings during fine-tuning?" In: *arXiv preprint arXiv:2004.14448* (2020).

[55] Kostadin Mishev, Ana Gjorgjevikj, Irena Vodenska, Lubomir T. Chitkushev, and Dimitar Trajanov. "Evaluation of Sentiment Analysis in Finance: From Lexicons to Transformers". In: *IEEE Access* 8 (2020), pp. 131662–131682. DOI: 10.1109/ACCESS.2020.3009626.

[56] Yunho Mo, Joon Yoo, and Sangwoo Kang. "Parameter-Efficient Fine-Tuning Method for Task-Oriented Dialogue Systems". In: *Mathematics* 11.14 (2023), p. 3048.

[57] Andrew Ng. *Andrew Ng: Opportunities in AI - 2023*. Stanford Online. 2023. URL: https://www.youtube.com/watch?v=5p248yoa3oE.

[58] Moldir Omarkhan, Gulnur Kissymova, and Iskander Akhmetov. "Handling data imbalance using CNN and LSTM in financial news sentiment analysis". In: *2021 16th international conference on electronics computer and computation (ICECCO)*. IEEE. 2021, pp. 1–8.

[59] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[60] Matthew E Peters, Sebastian Ruder, and Noah A Smith. "To tune or not to tune? adapting pretrained representations to diverse tasks". In: *arXiv preprint arXiv:1903.05987* (2019).

[61] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. "AdapterHub: A Framework for Adapting Transformers". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 2020, pp. 46–54.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[63] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. "Direct preference optimization: Your language model is secretly a reward model". In: *arXiv preprint arXiv:2305.18290* (2023).

[64] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: 1910.10683 [cs.LG].

[65] Shashank Shekhar, Adesh Bansode, and Asif Salim. "A comparative study of hyper-parameter optimization tools". In: *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. IEEE. 2021, pp. 1–6.

[66] Jozef Soja. *Open-Source Models Are Gaining In Performance Relative To Private Competitors*. ARK Investment Management LLC. 2023. URL: https://ark-invest.com/.

[67] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[68] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

[69] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[71] Chenglong Wang, Jiangyan Yi, Xiaohui Zhang, Jianhua Tao, Le Xu, and Ruibo Fu. *Low-rank Adaptation Method for Wav2vec2-based Fake Audio Detection*. 2023. arXiv: 2306.05617 [cs.SD].

[72] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. "Finetuned language models are zero-shot learners". In: *arXiv preprint arXiv:2109.01652* (2021).

[73] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[74] Jianqiong Xiao and Zhiyong Zhou. "Research Progress of RNN Language Model". In: *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. 2020, pp. 1285–1288. DOI: 10.1109/ICAICA50127.2020.9182390.

[75] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. "On layer normalization in the transformer architecture". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10524–10533.

[76] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. "Harnessing the power of llms in practice: A survey on chatgpt and beyond". In: *arXiv preprint arXiv:2304.13712* (2023).

[77] Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long. "Logme: Practical assessment of pre-trained models for transfer learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12133–12143.

[78] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. "Adaptive budget allocation for parameter-efficient fine-tuning". In: *arXiv preprint arXiv:2303.10512* (2023).

[79] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. *OPT: Open Pre-trained Transformer Language Models*. 2022. arXiv: 2205.01068 [cs.CL].

[80] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. *A Survey of Large Language Models*. 2023. arXiv: 2303.18223 [cs.CL].

[81] Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, et al. "Secrets of rlhf in large language models part i: Ppo". In: *arXiv preprint arXiv:2307.04964* (2023).

[82] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. "Lima: Less is more for alignment". In: *arXiv preprint arXiv:2305.11206* (2023).

[83] Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. "Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices". In: *arXiv preprint arXiv:2309.02411* (2023).