A Study on PEFT Techniques for Optimizing Content Generation and Textual Comprehension

A Thesis Submitted
In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

in
Artificial Intelligence

by

ROHIT NEGI
(Roll No. 2K22/AFI/17)

Under the Supervision of
Prof. RAHUL KATARYA
(Professor, Dept of Computer Science & Engineering)

To the
Department of Computer Science and Engineering

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-110042. India

May, 2024

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

ACKNOWLEDGEMENTS

I have taken efforts in this thesis. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to **Prof. Rahul Katarya** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing this review paper. I would like to express my gratitude towards the **Head of the Department (Computer Science and Engineering, Delhi Technological University)** for their kind cooperation and encouragement which helped me in the completion of this research survey. I would like to express my special gratitude and thanks to all the Computer Science and Engineering staff for giving me such attention and time.

My thanks and appreciation also go to my colleague in writing the survey paper and the people who have willingly helped me out with their abilities.

Rohit Negi

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CANDIDATE'S DECLARATION

I, **Rohit Negi**, Roll No. 2K22/AFI/17 student of M.Tech (Artificial Intelligence), hereby certify that the work which is being presented in this thesis entitled "**A Study on PEFT Techniques for Optimizing Content Generation and Textual Comprehension**" in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Artificial Intelligence in the Department of Computer Science and Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from August 2022 to Jun 2024 under the supervision of Prof Rahul Katarya, Asst Prof, Dept of Computer Science and Engineering. The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Place: Delhi **Candidate's Signature**

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CERTIFICATE

Certified that **Rohit Negi** (Roll No. 2K22/AFI/17) has carried out the research work presented in the thesis titled "**A Study on PEFT Techniques for Optimizing Content Generation and Textual Comprehension**", for the award of Degree of Master of Technology from Department of Computer Science and Engineering, Delhi Technological University, Delhi under my supervision. The thesis embodies result of original work and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree for the candidate or submit else from the any other University /Institution.

Prof. Rahul Katarya
(Supervisor)
Department of CSE
Date: Delhi Technological University

LIST OF PUBLICATIONS

1. R. Negi and R. Katarya, "Emerging Trends in Chatbot Development : A Recent Survey of Design, Development and Deployment," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-6, doi: 10.1109/ICCCNT56998.2023.10307280.

2. Paper title: ‘Experimental Study of Hyperparameter Tuning in Text Generation Chatbot on Deep Learning Techniques’ accepted in ICAAIML2024.

Table of Contents

List of Tables

List of Figures

A Study on PEFT Techniques for Optimizing Content Generation and Textual Comprehension

Rohit Negi

ABSTRACT

Recent progress towards large language models (LLMs) have brought significant improvents towards natural language processing (NLP), making it possible to perform tasks such as translating languages, generating text, and classifying information. To help these LLMs work efficiently with limited computational resources for specific tasks, parameter-efficient fine-tuning (PEFT) techniques have plays a major role. In our thesis we explore the history, methods, experiments, and real-world applications of PEFT techniques in detail, with a specific focus on optimizing them for creating content and understanding text better. It also covers the ethical and social consequences of using these techniques, along with strategies for adapting and collaborating on models. We analyze different aspects of PEFT techniques & compare different words to understand the changes it brings in differnet conditions so others can get help in their work while using these methods.

CHAPTER 1 - INTRODUCTION

1.1 Overview

Language modelling is the process of teaching computers to understand & generate text that resembles humans. It plays a major role in NLP, as it includes developing models & algorithms that can accurately predict the next words in a sentence or paragraph. This helps in generating new text, finishing sentences, and figuring out the likelihood of different word combos.

Back in the day, early language models were pretty simple. They used basic stats to guess what words would come next based on how often they appeared together. But then came deep learning, tons of public data, and super powerful computers. That's when Large Language Models (LLMs) stepped onto the scene. They use fancy deep learning techniques, especially transformer architectures, to really dig into language patterns. One cool thing about LLMs is they can handle massive amounts of data, even stuff like pictures and sounds. They're basically language wizards, understanding and generating human-like text better than ever before.

Today's LLMs are can ace all sorts of tasks, thanks to their massive size and computational prowess, but that also means they're pretty demanding. For example GPT-175B, a model with a mind-blowing 175 billion parameters, requires a small army of GPUs just to function properly. To deal with this issue, there's something called model compression. It's like squeezing a large model into a smaller, more manageable version making it easier to run on low coputation devices like smartphone.

LLMs should also raise some serious questions about the environment and ethics. They also consume a large amount of energy, which is not good for the planet. Morever in some areas it become challenging to obtain the necessary hardware needed for it. One way to address these concerns is through PEFT techniques which can help in assisting this issue by reducing the resources & energy required without any significant downfall in performance. Following these ways can help in coserving power and environment and increase its access to more people.

Here in our thesis we will explore various Parameter Efficient Fine Tuning (PEFT) Techniques to understand its working in different scenarios.

1.2 History

1.2.1 History of LLMs

Origin of our present LLMs goes way back, when researchers first started exploring language models and neural networks. Imagine this: it all began with what we call statistical language models. Back then, researchers were all about using probabilities to predict what words would come next in a sentence. That time methods like n-grams & Hidden Markov Models were pretty basic compared to what we have now, but they paved the way for understanding natural language, capapble of helping with simple things like generating text and guessing the next word, but were not good with complex stuff.

Further ahead, researchers started looking at big chunks of text and using machine learning to find patterns. Support Vector Machines were a big deal during this phase as they brought a different understanding to language and gave us cool things like spam detectors and sentiment analyzers.

Moving in forward direction, things improved with techniques like deep learning with RNNs and LSTM netoworks which were like detectives, diving deep into text and uncovering all the little details and long-distance connections which bought a big change as machines could now understand context better, opening doors for translating languages and recognizing speech. But it was not all perfect.

The big change happened with the Transformer model having fancy new architecture based on self-attention allowing models like GPT and BERT to truly shine. They could finally look at a whole sentence or even an entire document all at once, giving them a real grasp of context. Thanks to these models, we now have smarter chatbots, better text summaries, and smoother translations.

Since then, language learning models (LLMs) have been on a constant journey of growth and improvement. Models like GPT-1, GPT-2, and GPT-3 just kept getting bigger and smarter. Not only them but other models like ALBERT and RoBERTa are continuing in pushing the boundaries too, not just limiting to general language tasks anymore and now we even have various models specialized for fields like medicine, science, and even coding. And as we continue moving forward, we're ensuring that these models are ethical, fair, and understandable, so that everyone can benefit from them responsibly.

1.2.2 History of PEFT Techniques

Origin of our PEFT techniques for LLMs starts with the challenges of traditional fine-tuning methods. These methods involved updating all parameters of a pre-trained model, which was a real headache. It required a lot of resources and was impractical for very large models, making it difficult for many researchers and practitioners to access. But it changed in 2019 as adapter modules were introduced which were like small, trainable layers that we add to each layer of a pre-trained model & during fine-tuning only these modules are updated, which means we can reduce the number of parameters that need training. It was a real game-changer, making the process more efficient and accessible. One of the example of this is the use of adapter modules with BERT.

In 2021, further innovations emerged with Low-Rank Adaptation (LoRA), which introduced low rank decomposition of the weight matrices to the model. This technique reduced the trainable number of parameters by focusing on most important aspects of the weight updates, as seen in the application of LoRA to GPT-3. Another notable method from 2021 was BitFit. It finetuned only bias terms of the model, drastically cutting down the number of parameters while maintaining much of the model performance. Layer-wise fine-tuning also gained popularity, involving the selective fine-tuning of only certain layers, typically the final ones, while freezing the rest. BERT was an example of a model that benefited from this approach.

The year 2021 also saw the rise of prompt tuning and prefix tuning. Prompt tuning involved learning a set of prompt tokens that guide the model to produce desired outputs, with GPT-3 being a prominent example. Prefix tuning, on the other hand, involved prepending learned tokens (prefix) to the input embeddings, as seen in models like BART. P-Tuning, introduced in 2021, combined prompt tuning with other optimization strategies improving both efficiency & performance of the fine tuning process.

Recent innovations in parameter-efficient fine-tuning have continued to evolve. In 2022, sparse fine-tuning focused on only updating a sparse subset of the model parameters, determined by their importance during fine-tuning, as applied to models like Sparse GPT-3. Efficient Prompt Learning (EPL), introduced in 2023, leveraged efficient prompts to guide model behavior without extensive parameter updates, significantly reducing training overhead while maintaining high performance. Compacter, also from 2023, combined adapters with low-rank updates to further compress the fine-tuning process, making it highly efficient for very large models such as GPT-4. HyperNetworks, another 2023 innovation, used small networks to dynamically generate weights for larger networks, reducing the need for extensive parameter updates and enhancing efficiency.

The most recent advancements include Parameter-Efficient Multi-Task Learning (PEMTL) in 2024, which enables fine-tuning across multiple tasks simultaneously by optimizing shared parameters to enhance performance and efficiency across tasks.

We use these methods in multi-task learning models and it shows significant improvements in both efficiency and performance.

Way of fine-tuning parameters have completely changed with context to adapting large language models as now, even with limited computing power, we can fine-tune huge models making it possible for more researchers and practitioners to customize these advanced NLP models for specific tasks. Because of this, a noticeable increase in innovation and new applications across various fields can be seen.

1.3 Motivation

The rapid advancement of large language models (LLMs) has completely transformed the fields of natural language processing (NLP) and artificial intelligence (AI). Models like GPT-3 and BERT have demonstrated impressive abilities in generating human-like text, understanding complex language structures, and handling a wide variety of NLP tasks. However, there's a downside: these models are enormous and need a lot of computing power, which makes them somewhat challenging to work with.

That's where PEFT techniques come by offering a promising solution to the problems posed by these massive models by reducing the number of trainable parameters, making it possible to customize large models for specific tasks without incurring high computing costs. It may sound great but we need more research to understand how effective these PEFT techniques are for generating content and understanding text.

Here we seek to address the gaps by exploring and comparing various PEFT methods making our goal to find best ways to make your models more efficient and boost their performance. Get ready: we are about to dive deep into the world of PEFT techniques and their impact on LLMs.

1.4 Objectives

1. Literature Review:
   - Explore the various PEFT techniques available, such as adapter modules, LoRA, prompt tuning, prefix tuning, and the latest advancements.
   - Delve into the strengths, limitations, and particular applications of each technique, particularly in content generation and text comprehension.

2. Methodology Development:
   - Create a solid experimental framework to evaluate PEFT techniques. This means choosing the right datasets, defining performance metrics, and setting up baseline models.

- Design experiments that systematically test how different PEFT methods affect model performance, computational efficiency, and resource usage.

3. Experimental Evaluation:
- Apply and refine the chosen PEFT techniques on large language models.
- Experiment with these techniques across different tasks, like text generation and summarization, to assess their effectiveness.

4. Ethical and Societal Implications:
- Dive into the ethical considerations that come with using PEFT techniques, including things like bias, transparency, and accessibility.
- Offer recommendations and best practices for ethically and responsibly utilizing PEFT-optimized models in various applications.

By achieving these goals, this thesis aims to contribute to the field of NLP by providing a comprehensive understanding of PEFT techniques and how they can be optimized for content generation and text comprehension. The findings will offer valuable insights for researchers and practitioners who want to use advanced LLMs effectively and responsibly.

CHAPTER 2 – LITERATURE SURVEY

2.1 Large Language Models

The development of Deep Learning (DL) techniques, the presence of super powerful computers, and enough training data have all led to the rise of LLMs. These models can learn complex patterns, understand subtle nuances in language, and make connections between words and ideas because they are trained on huge amounts of online text. When these models are fine-tuned for specific tasks, they have shown impressive results, achieving cutting-edge performance across various benchmarks.

Table 2.1 Comparison of Traditional ML, Deep Learning, and LLMs [1]

Comparison	Traditional ML	Deep Learning	LLMs
Training Data Size	Large	Large	Very large
Feature Engineering	Manual	Automatic	Automatic
Model Complexity	Limited	Complex	Very Complex
Interpretability	Good	Poor	Poorer
Performance	Moderate	High	Highest
Hardware Requirements	Low	High	Very High

Table 2.2 Overview of different LLM models

Model	Release Date	Parameters	Pretrain Data	Hardware Trained On	Training Time
BigGAN	2018-09	488 million	ImageNet	TPU v3	-
BERT	2018-11	340 million	16GB	16 TPU v2	4 days
BERT-Large	2018-11	340 million	16GB	16 TPU v2	4 days
GPT-2	2019-02	1.5 billion	40GB	256 V100 GPU	1 month
VQ-VAE-2	2019-05	85 million	ImageNet	128 TPU v3	-
XLNet	2019-06	340 million	32GB	512 V100 GPU	-
RoBERTa	2019-07	355 million	160GB	1024 V100 GPU	-
DistilBERT	2019-10	66 million	16GB	8 V100 GPU	-
T5	2019-10	11 billion	750GB	1024 TPU v3	2 weeks
BART	2019-10	400 million	160GB	512 V100 GPU	-
StyleGAN2	2019-12	23.1 million	Flickr-Faces-HQ	32 V100 GPU	-
ALBERT	2019-12	223 million	16GB	64 TPU v3	1 week
Turing-NLG	2020-02	17 billion	-	256 V100 GPU	-
GPT-3	2020-06	175 billion	570GB	DGX SuperPOD (NVIDIA V100, 10,000)	1 month
EleutherAI GPT-Neo	2021-03	2.7 billion	Pile dataset (825GB)	192 V100 GPU	1 month
LaMDA	2021-05	137 billion	1.56TB	1024 TPU v4	-

EleutherAI GPT-J	2021-06	6 billion	Pile dataset (825GB)	256 V100 GPU	1 month
Jurassic-1	2021-08	178 billion	-	256 V100 GPU	-
OpenAI Codex	2021-08	12 billion	Multiple sources	DGX SuperPOD (NVIDIA V100, 10,000)	-
FLAN	2021-09	137 billion	-	128 TPU v3	60 hours
Gopher	2021-12	280 billion	300 billion tokens	1024 TPU v4	-
ERNIE 3.0	2021-12	10 billion	-	256 V100 GPU	-
GLaM	2021-12	1.2 trillion	280 billion tokens	2048 TPU v4	-
Megatron-Turing NLG	2022-01	530 billion	270TB	4480 NVIDIA A100	2 months
Chinchilla	2022-03	70 billion	-	1024 TPU v4	-
PaLM	2022-04	540 billion	780GB	6144 TPU v4	-
M6-T	2022-04	100 billion	-	512 TPU v4	-
OPT	2022-05	175 billion	180 billion tokens	992 NVIDIA A100	-
BLOOM	2022-07	176 billion	350 billion tokens	Jean Zay supercomputer (NVIDIA A100, 384)	3.5 months
UL2	2022-10	20 billion	1 trillion tokens	1024 TPU v4	-
Flan-T5	2022-10	11 billion	-	512 TPU v4	-
Galactica	2022-11	120 billion	106 billion tokens	1024 TPU v4	-
LLaMA	2023-02	65 billion	1.4 trillion tokens	2048 NVIDIA A100	-
GPT-4	2023-03		-		-
PanGu-Σ	2023-03	1.1 trillion	329 billion tokens	512 Ascend 910	100 days
Pythia	2023-04	12 billion	300billion tokens	256 40G A100	-
PaLM2	2023-05	16 billion	100billion tokens	-	-
CodeGen2	2023-05	16 billion	400billion tokens	-	-
StarCoder	2023-05	15.5 billion	1 trillion tokens	512 40G A100	-
LLaMA2	2023-07	70 billion	2 trillion tokens	2000 80G A100	-

From 2018 to 2023, large language models (LLMs) have gotten a lot bigger and more complex. First, we had models like BERT and GPT-2 that set the foundation. But now we have even larger models like GPT-3 and GLaM, which have hundreds of billions of parameters. To handle these behemoth models, we need advanced hardware like TPU v4 and NVIDIA A100 GPUs that can handle the heavy lifting. And the amount of data used for pretraining these models has also increased significantly, showing us how important it is to have diverse and extensive datasets. Additionally, there's a shift towards using multimodal and specialized models. As we move forward, we need to find ways to tackle the computational costs, reduce the environmental impact, and address ethical concerns to ensure sustainable development of AI.

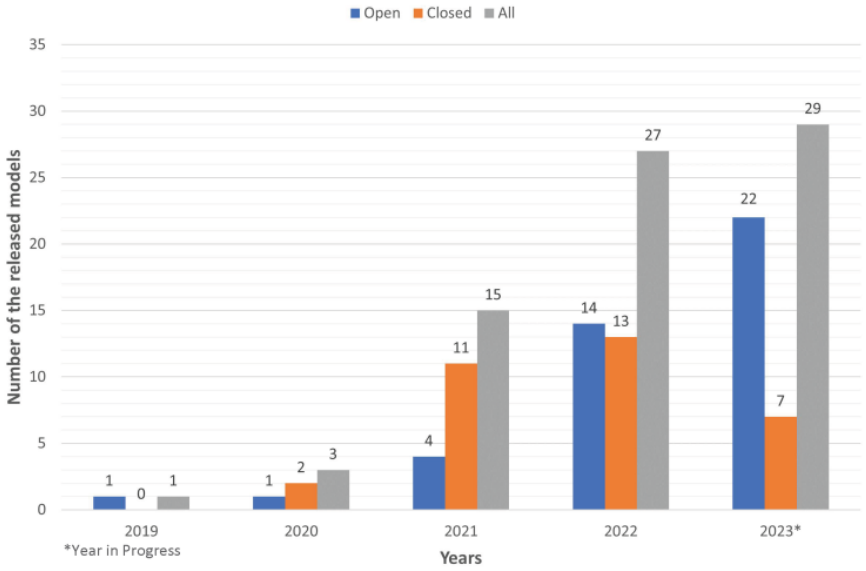


Fig. 2.1: No. of LLMs released over the years [2]

2.1.1 Capabilities of LLMs

1. Question-answering (QA):
- LLMs are great at giving direct answers to questions in everyday language.
- They can be trained on big text collections and fine-tuned on datasets with labeled QA information to improve their performance.
- LLM-based QA systems are excellent at handling tricky questions and putting together answers from different sources.

2. Text Generation:
- LLMs can create top-notch content for various purposes like articles, blogs, and social media posts.
- They understand and produce natural language, ensuring that the generated content is accurate and makes sense.

3. Language Translation:
- LLMs can accurately and fluently translate text between different languages, helping people communicate across language barriers.
- Their precision and fluency make global collaboration and access to information much easier.

4. Text Classification:
- LLMs are experts when it comes to analyzing and categorizing text, like figuring out the sentiment of a message or detecting spam.

- They handle large amounts of text data efficiently, making data management a breeze.

5. Summarization:
- LLMs can create short and coherent summaries of long texts, saving time and effort in content creation.
- They make content creation more efficient by capturing the most important information accurately.

6. Virtual Assistance:
- In virtual assistants and chatbots, LLMs understand what users are asking, provide the required information, and have natural human-like conversations.
- They improve user-experience and operation efficiency by offering personalized help and automating routine tasks.

7. Information Extraction (IE):
- LLMs accurately extract important details and relationships from unstructured text, making it easier to create organized knowledge graphs.
- They make information extraction processes more efficient and accurate.

8. Dialog Systems:
- LLMs make open-domain chatbots sound more natural and coherent, creating more engaging conversations.
- Their use in dialog systems changes how people interact with technology, making conversations more enjoyable.

9. Semantic Search:
- LLMs understand what users are searching for and provide search results beyond just matching keywords.
- They improve search accuracy and relevance, helping finding information more easier.

10. Speech Recognition:
- LLMs trained on huge amounts of data improve accuracies of automated speech transcription.
- Their contextual knowledge and ability to keep learning make them perfect for handling speech signals effectively.

2.1.2 Languange Model Types

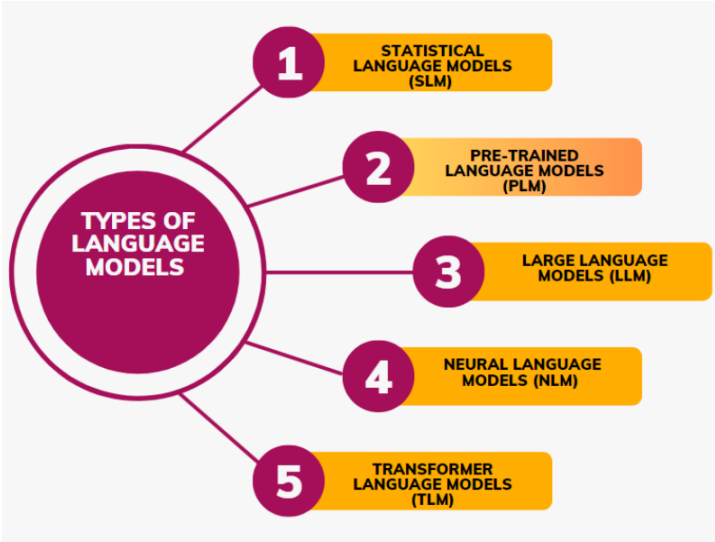


Fig. 2.2: Distribution of language models [3]

1. Statistical Language Models (SLMs):
   - Use statistical methods to assign probability values to word sequences.
   - Based on the idea that word frequencies and patterns in large text corpora can predict word occurrence likelihood.
   - Examples: N-gram models, Hidden Markov Models.

2. Pre-trained Language Models (PLMs):
   - Those Language models that are trained in unsupervised manner on large corpora before being fine-tuned for specific tasks.
   - Learn representations of general language by extracting structures & patterns from huge amounts of text data.
   - Examples: BERT, RoBERTa, GPT.

3. Large Language Models (LLMs):
   - These Language models have many parameters and significant processing power.
   - Examples: GPT-3, BERT, etc.

4. Neural Language Models (NLMs):
   - Word sequence probability distribution can be modelled using neural network topologies.
   - Designed to capture intricate relationships between words and generate contextually appropriate text.
   - Examples: RNNs, LSTM networks.

5. Transformer Language Models (TLMs):
   - Language models that specifically utilize the Transformer architecture.
   - Examples: GPT series, BERT.

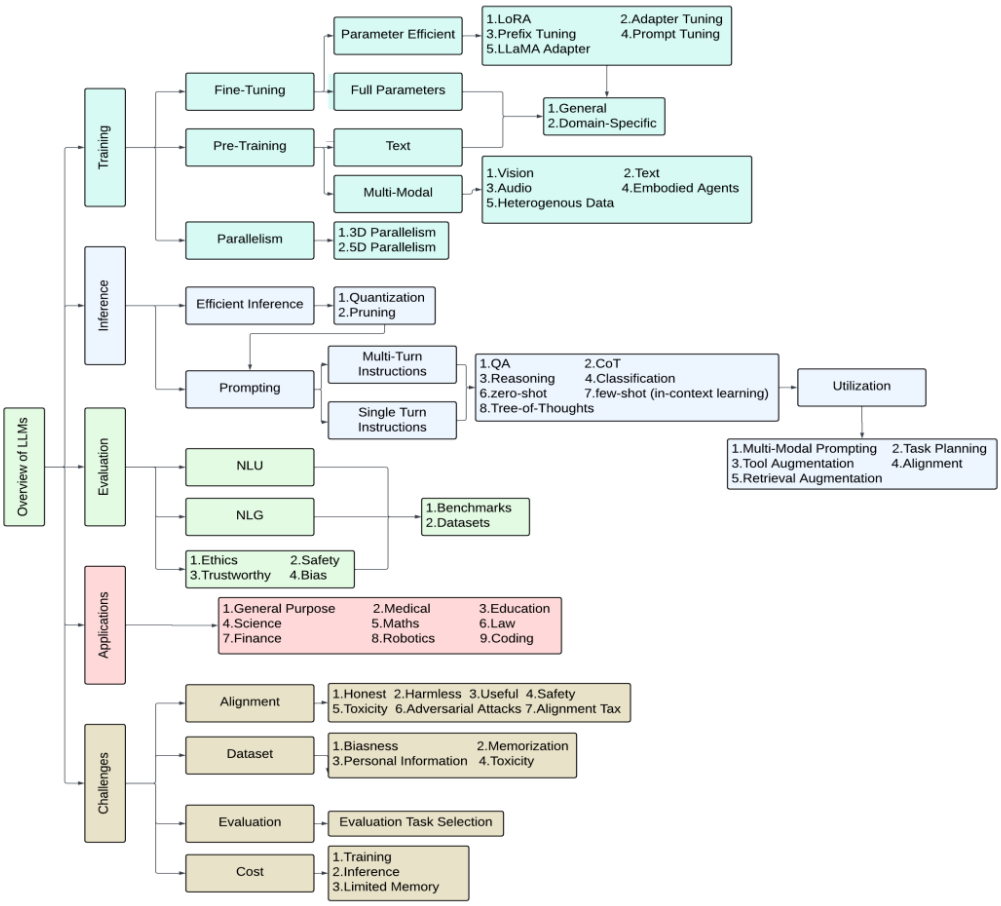


Fig. 2.3: A broader overview of LLMs [2]

2.2 PEFT Techniques

As large language models gain traction, effective deployment and training become essential prerequisites for facilitating broad adoption. A completely distinct set of weights is needed for every job on which an LLM is refined. When models reach hundreds of billions of parameters, it becomes prohibitively slow to reload all the weights for every task and highly inefficient to host a separate set of weights for every model. By adjusting a very tiny percentage of weights in relation to the entire model size while maintaining the model's frozen state, PEFT algorithms seek to address this issue. Several versions of the same model can be served simultaneously at inference time by quickly switching small submodules rather than all of the weights.

PEFT techniques, aims to maximise the process of fine-tuning LLMs. PEFT fine-tunes the model by updating a subset of parameters, in contrast to standard approaches that update the complete model. This is especially helpful because fine-tuning LLMs typically calls for a large volume of task-specific data, which may be costly and time-consuming to collect. It is computationally costly to update all parameters using the conventional method, particularly when working with LLMs. By providing a more effective use of computing power, data, and time, PEFT tackles these issues. This method works great when it's tough or expensive to gather a lot of task-specific data. It's also useful when you need a faster and more resource-efficient way to fine-tune the process. From a technical standpoint, PEFT methods involve building layers, adding more tokens, and breaking down weight gradients into specific matrices. This helps learn a minimal set of parameters for the job. [4]

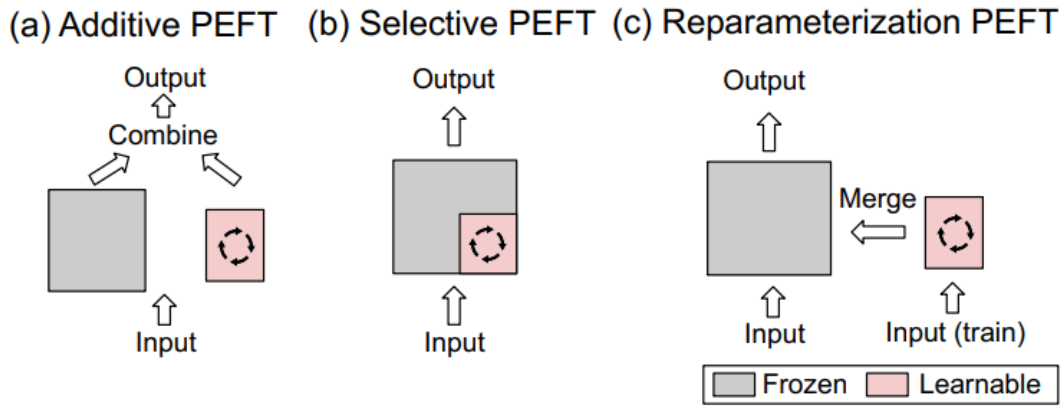


Fig. 2.4: Different types of PEFT algorithms. [5]

PEFT algorithms are grouped into four main types: additive, selective, reparameterized, and hybrid fine-tuning. Additive fine-tuning introduces new adjustable modules or parameters using methods like Adapter, Soft Prompt, and other

variations. Selective fine-tuning doesn't add new parameters but instead makes a subset of the existing model parameters adjustable. It uses techniques like unstructural and structural masking. Trainable low-rank parameters are incorporated into reparameterized fine-tuning during training and subsequently reintegrated into the original model for inference. Techniques like low-rank decomposition and LoRA derivatives are used to achieve this. By combining components from several PEFT techniques, hybrid fine-tuning builds a single, cohesive model that capitalises on their combined advantages. These approaches aim to optimize fine-tuning by modifying the model architecture, selectively tuning parameters, or reparametrization models in a low-dimensional space.

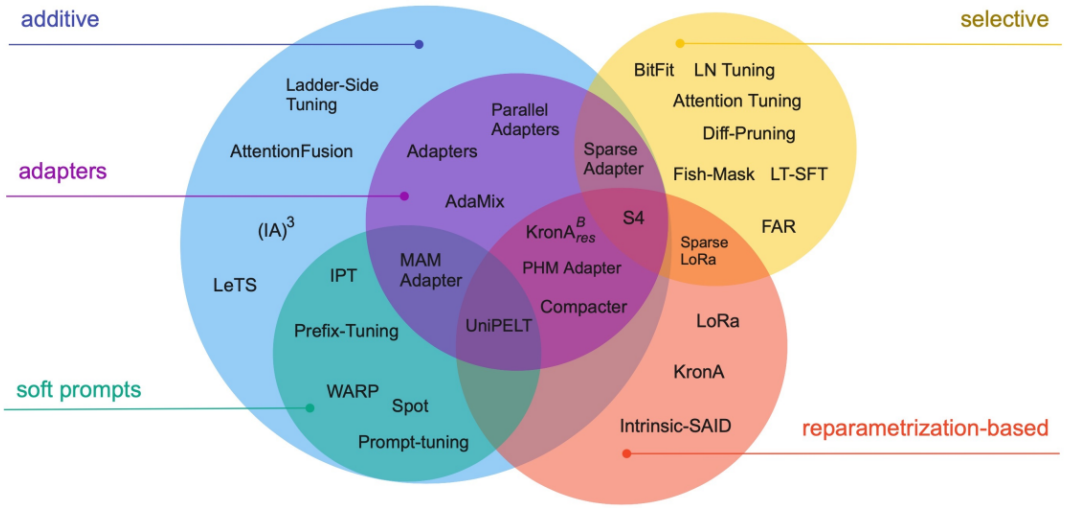


Fig. 2.5: Parameter-efficient fine-tuning methods taxonomy. [6]

Below here discussed are some common PEFT Techniques.

2.2.1 Pruning

To make the model smaller by removing unnecessary parts, pruning is a powerful optimization strategy. This technique allows you to directly prune certain parameters without significantly affecting the effectiveness of the model, as these parameters don't contribute much to its performance. Pruning offers benefits such as memory optimization, improved computational efficiency, and better storage efficiency. There are two basic forms of pruning: Unstructured Pruning, which removes specific parameters, and Structured Pruning, which preserves the overall network structure while removing connections or hierarchical structures based on predefined rules. Researchers have been working on combining pruning methods with Large Language Models (LLMs) to address the large size and computational costs associated with

these models. They have classified their work into organized and unstructured pruning strategies. [7]

2.2.2 Knowledge Distillation

It is an effective machine learning method for improving model performance and generalization. Knowledge is transferred from a more sophisticated instructor model to a more straightforward student model. The primary goal is to transform the teacher model's vast body of knowledge into a more useful representation, thereby improving and streamlining it. There are two categories of knowledge distillation: White-box KD, where the teacher's parameters are accessible, and Black-box KD, where only the teacher's predictions are available. [8]

2.2.3 Quantization

Quantization is a commonly used technique in model compression to reduce the computational and storage overhead of deep learning models. Conventional representation uses floating-point numbers; however, quantization reduces computer complexity and storage needs by converting them to integers or other discrete forms. Though quantization entails inherent precision loss, effective implementation can lead to significant model compression with negligible accuracy damage. Quantization can be generally classified into two primary approaches: quantization-aware training (QAT) and post-training quantization (PTQ). When quantization is used to compress the model is when the main differences are found. Quantization is used by QAT when the model is being trained or fine-tuned, whereas PTQ quantizes a model after training. [9]

2.2.4 Low-Rank Factorization (LoRA)

The goal of Low-Rank Factorization, a model compression technique, is to approximate a weight matrix by splitting it them to smaller matrices that have dimensions that are noticeably less. A big weight matrix W must be factorised into two matrices, U and V, so that $W \approx UV$. U is a $m \times k$ matrix, and V is a $k \times n$ matrix, with k being significantly smaller than m and n. This is the basic idea behind the process. The number of parameters and processing overhead are significantly decreased because the product of U and V almost equates to original weight matrix. [10]

2.2.5 Prompt Tuning (PT)

By using an embedding layer (intra-connectivity) to create token-like embeddings, prompt tuning (PT) entails concatenating these embeddings to the input embeddings of the Pre-trained Language Model (PLM) in the workspace. To make these token-like embeddings more in line with the job goal, fine-tuning is applied. PT inserts parameters precisely at the embedding layer with respect to modular qualities and cooperation with the PLM. An inter-connectivity of fixed density is established

between the PLM and PT by concatenating token-like embeddings with input embeddings. [11]

2.2.6 Prefix Tuning (PF)

This method uses two linear layers with a Softmax activation in between, as opposed to Prompt Tuning. This extends the workspace to incorporate the input embeddings and the keys and values of the Attention in all Transformer levels, with respect to modular qualities and cooperation with the Pre-trained Language Model (PLM). These matrices are concatenated with the PF token-like embeddings produced by this method (integration form). Through the PLM, PF creates a fixed, dense interconnectivity for all of its data. [12]

2.2.7 Adapters

Adapters apply intra-connectivity via a feedforward layer, bottlenecking data through two linear layers with ReLU activation positioned in between that project data down and then up. With the use of FNN and Attention blocks, adapters create hidden representations that are inserted sequentially. Adapters combine their findings with their workspace (Attention and FNN blocks) by direct addition ($h + \Delta h$), which facilitates cooperation with the Pre-trained Language Model (PLM) and has modular qualities. Although there are variations such as AdapterDrop, Compacters, and Tiny-Attention Adapters that can change the amount of insertions or internal connectivity, they are all integrated using direct addition. Connectors establish fixed, dense interconnectivity by sending all of their data to their workspace. [13]

2.2.8 Tiny-Attention Adapters

By adding a tiny Attention layer, Tiny-Attention Adapters, a variation of Adapters, alter intra-connectivity. Tiny-Attention Adapters are comparable to Adapters in terms of modular qualities and their ability to work with the Pre-trained Language Model (PLM). They are added one after the other, operate together directly with their workspace, and take in inputs in the form of concealed representations. Their insertion into the workspace happens after the Attention block, though, and they create dynamic interconnectivity by sending information to the workspace only when it is needed. [14]

2.2.9 Compacters

A type of adapter known as a compacter has a different reparameterization of the layer W. The representation of W in the conventional Adapter layer is $W \in Rk \times dW \in Rk \times d$. On the other hand, Compacters reparameterize W by summing Kronecker products, where k and d are divisible by a hyperparameter nn that is defined by the user. In particular, $W = \sum i=1nAi \otimes BiW = \sum i=1nAi \otimes Bi$ = nn Kronecker products, where $Ai \in Rn \times nAi \in Rn \times n$ and $Bi \in Rnk \times ndBi \in Rnk \times nd$. By distributing the weights of AI among its layers, compacters improve parameter efficiency even more. Compacters are similar to Adapters in terms of their modular characteristics and

ability to work with the Pre-trained Language Model (PLM). They function together with the PLM, utilise direct addition for integration, adhere to a sequential insertion form, and have the same workspace. [15]

2.2.10 (IA)3

A (IA)3 module is made up of three vectors that resize the Transformer layer's Attention (keys, values) and FeedForward Network (FFN) blocks. In order to prevent the module from influencing the Pre-trained Language Model's (PLM) functioning before it is led by the task's objective gradient, these vectors are initialised to one during the tuning phase. In order to work with the PLM and include modular qualities into its workspace, which comprises intermediate FFN , keys & values across all Transformer levels, (IA)3 employs learned vector rescaling. Sequential insertion allows it to transmit all of its data to its workspace, creating fixed, dense interconnectivity. [16]

2.2.11 MHM

Researchers looking at adapter insertion form discovered that sequential adapters did not perform as well in testing as parallel adapters (PA). On the basis of their study, they dissected the architecture of the previously stated PEFT techniques and presented a brand-new approach termed MHM, which combines parallel adapters with prefix tuning. By parallel insertion of adapters in the FeedForward Network (FFN) modules in MHM, parallel adapters obtain the same FFN moduls input. On the other hand, the self-attention layers and FFN modules come before the standard adapters. PA is used as one of the experimental PEFT approaches as a point of comparison. [17]

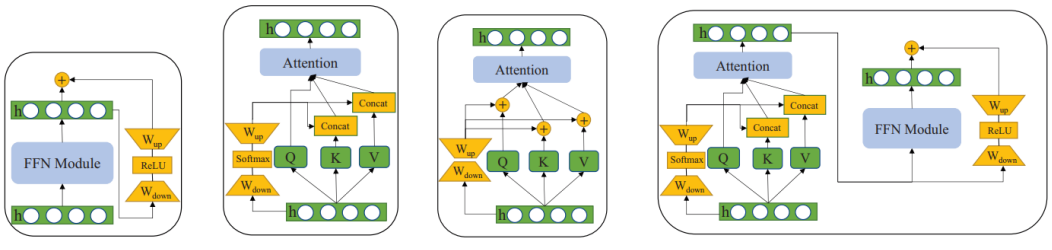


Fig. 2.6: General structure of some common PEFT models. [18]

Table 2.3 Structural properties of different PEFT modules [19]

PEFT technique	Intra-connectivity	Inter-connectivity	Parameters adapted	Parameter Sharing	Input type	Insertion form	#Insertions	Integration form	Workspace
Prompt tuning	dense:embedding	fixed:dense	Addition	none	Weights	Parallel	1 layer	concatenation	embedding layer
Prefix tuning	dense:non-linear MLP	fixed:dense	Addition	none	Weights	Parallel	all layers	gated addition	embedding layer; att. layer:keys/values
LoRA	dense:linear MLP	fixed:dense	Reparametr.	none	Data	Parallel	all layers	scaled addition	att. layer:queries/values
Adapters	dense:non-linear MLP	fixed:dense	Addition	none	Hidden	Sequential	all layers	direct addition	FFN layer; attention layer
Tiny-Att. Ad.	dense:self-attention	dynamic	Addition	none	Hidden	Sequential	all layers	direct addition	attention layer
Compacters	dense:non-linear MLP	fixed:dense	Addition	shared	Hidden	Sequential	all layers	direct addition	FFN layer; attention layer
$(IA)^3$	none:parameter vector	fixed:dense	Addition	none	Weights	Sequential	all layers	rescaling	FFN layer:intermed. repres.; attention layer:keys/values

Table 2.4 Efficiency of the different PEFT techniques [19]

PEFT	Module complexity	Number of parameters per Transformer layer
Prompt Tuning	$\mathcal{O}(1)$	$n\ d_m$
Prefix Tuning	$\mathcal{O}(kd)$	$n\ d_m + {d_m}^2 + 2d_h\ d_m$
LoRA	$\mathcal{O}(rd)$	$2 \times (2d_h\ d_m\ )$
Tiny-Attention Adapters	$\mathcal{O}(T)$	$4 \times d_m$
Adapters	$\mathcal{O}(kd)$	$2 \times (2\ d_h\ d_m\ )$
Compacters	$\mathcal{O}(\frac{kd}{N})$	$2 \times (2\ (d_h + d_m)\ )$
$(IA)^3$	$\mathcal{O}(1)$	$6 \times d_m$

where, for each of the following: n = number of tokens for prompt and prefix tuning; d = input/output dimension of PEFT module; r = rank for LoRA; k = bottleneck dimension for Adapters; and dm = model dimension. d = dm. T = Input embeddings. N = Kronecker-products Reduced dimension.

2.3 Related Work

The paper [20] investigates the optimal balance of model-size and training-tokens for LLMs within a computing budget. It finds that current practices focus too much on increasing model size without proportionally increasing training data, resulting in under-trained models. According to the study, equal scaling of model size and training tokens is necessary for optical computing training. Empirical evidence from training over 400 models shows that many large models, like Gopher and GPT-3, are not optimally trained. The authors introduced the Chinchilla model, which, with 70 billion parameters and 1.4 trillion tokens, outperformed larger models like Gopher and GPT-3, achieving a state of the art accuracy on MMLU benchmark. This indicates that smaller models trained with more data are more efficient and effective, reducing compute costs for fine-tuning and inference. The paper advocates for revisiting training practices to adopt these findings for more efficient use of computational resources and better-performing models.

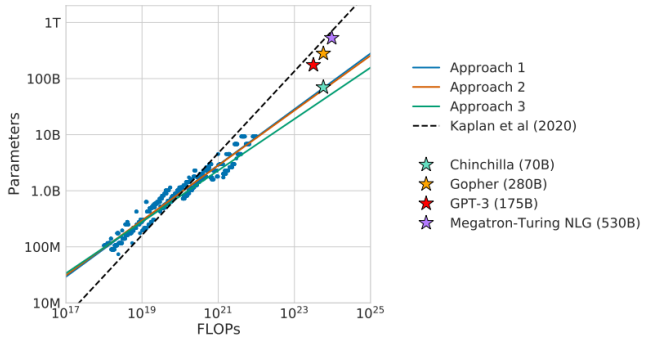


Fig. 2.7: Overlaid predictions. [20]

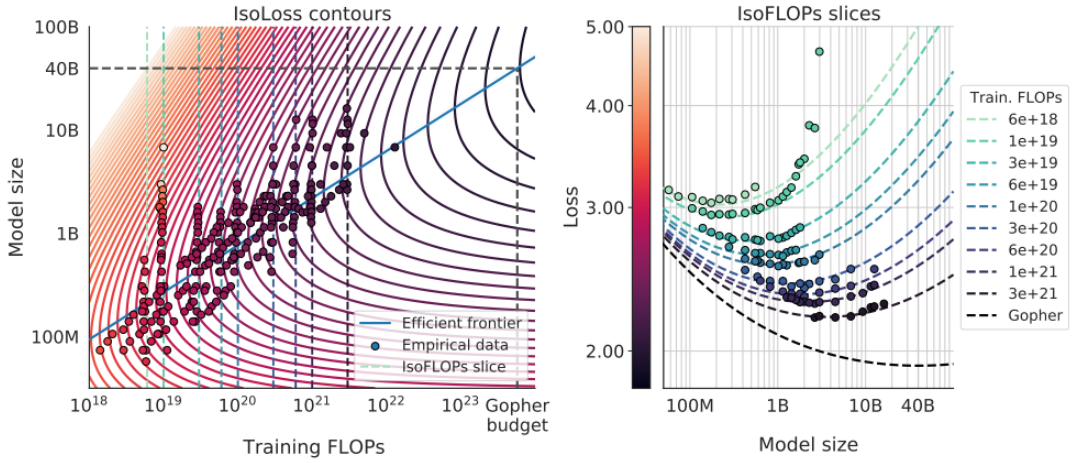


Fig. 2.8: Parametric fit. [20]

Parametric modelling of the loss *Lˆ„N*, *D*” and display contour (left) and isoFLOP slices (right).

Below here are some related work which efficiently uses different PEFT Techniques.

Integrating pruning techniques can significantly enhance the efficiency of PEFT methods. AdapterDrop [21] examines taking adapters out of lower transformer layers and using multitasking adapters in AdapterFusion [22]. This study finds that pruning can boost both training & efficiency with only a minor performance impact. SparseAdapter [23] looks at a variety of pruning techniques and finds that typical adapters can be outperformed when a high sparsity ratio, like 80%, is used. The model's capacity and performance are significantly improved by the Large-Sparse configuration, which doubles the bottleneck dimension while keeping the same parameter budget (for example, 50% sparsity). SPLoRA [24] uses channel based pruning on the LoRA weights, affecting both the source weights and the LoRA parameters. Similarly, LoRAPruning [25] applies structured pruning to both the pre-trained model-weights and the LoRA weights. LoRAPruning makes weight merging easier than unstructured pruning of LoRA, which sparsifies the model-weights while maintaining the density of LoRA weights. Also, utilising the gradients of LoRA, it presents a new standard to approximate the pretrained weights' gradients, helping to estimate the importance of weights. ProPETL [26] creates a single-shared prototype, like an LoRA, prefix or adapter, that is used across different layers & tasks. This method also develops binary masks pruning various sub-networks for different layers & tasks, significantly improving parameter efficiency by allowing parameters to be reused across layers and tasks.

Quantization is a popular technique to boost computational efficiency and cut down on memory usage. For example, BI-Adapter [27] takes advantage of adapters' resilience to noise in the parameter space and introduces a clustering-based quantization method. This approach achieves impressive results by quantizing adapters to just 1-bit precision, which drastically reduces storage needs while still delivering top-notch performance compared to other precision settings. PEQA [27] uses a two-stage process for fine-tuning that is both parameter efficient and & aware of quantization. The first step involves quantizing the FFN weight matrix that has been pre-trained. In the second stage, only specific parameters are fine-tuned, which keeps both memory and parameter use in check. QLoRA [28] unveils a number of innovative methods, including as 4-bit Paged Optimizers, Double Quantization & NormalFloat, that enable the optimisation of a large 65B language model with 4-bit quantization on a single 48GB GPU. This method maintains performance on par with full 16-bit fine-tuning. LoftQ [29] offers a framework for better initializing quantized backbone weights and LoRA weights, tackling quantization issues through optimization during network initialization. LQ-LoRA [30] employs an iterative approach to apply integer linear programming for dynamic quantization settings and breaks down weights, drawing inspiration from robust principal components analysis. QA-LoRA [31] builds on QLoRA by allowing quantization during the inference stage with INT4 quantization and group-wise operators, which boosts both efficiency and accuracy. Finally, BitDelta [32] presents a quantization method of 1 bit post training that focuses on weight differences between finetuned & pretrained models. This method simplifies deployment by using a single full precision base model along with 1-bit deltas that are efficiently batched.

Several methods have been developed to reduce memory utilisation during fine-tuning of large language models (LLMs) by minimising the requirement for caching gradients across the LLM. For example, Side-Tuning [33] & Ladder-Side Tuning (LST) [34] present a network branch that learns & operates in tandem with the primary model. These techniques greatly reduce the amount of memory needed during training by channelling backpropagation only through this parallel branch, removing the need to maintain gradient information for the weights of the main model. In a similar way, Res-Tuning [35] separates PEFT tuners (such as prompt tuning and adapters) from the main model. Expanding on this, Res-Tuning-Bypass creates a parallel bypass network that removes the data-flow induced in the decoupled tuners to main model, thus eliminating the need of gradient caching during backpropagation in the main model. MEFT [36] takes inspiration from reversible models [37], where intermediate activations aren't cached during the forward pass but are recalculated during backpropagation. Without more pretraining, MEFT investigates how to transform an LLM into its reversible counterpart. In order to guarantee that the fine-tuning performance is comparable to complete fine-tuning methods, a crucial step in this procedure is meticulously initialising newly added parameters in the pretrained models to preserve the original starting point. Three techniques are presented by MEFT to drastically lower the amount of memory required to store activations. LoRA-FA [38] tackles the memory overhead issue in LoRA fine-tuning, where high activation memory consumption is required during backpropagation. To solve this, LoRA-FA updates only the projection-up weights, leaving pretrained weights & projection-down weights frozen. This approach removes the need to store input activations, as the intermediate activation is sufficient for gradient computation, thus greatly reducing memory requirements. In order to further reduce memory consumption during fine-tuning, certain techniques completely prevent backpropagation inside LLMs. HyperTuning [39] uses a HyperModel to create PEFT parameters from a small number of examples, yielding outcomes comparable to full model fine-tuning. PEFT Plug-in [40] first trains PEFT modules on smaller language models, which is more memory-efficient, and then integrates these trained modules into LLMs during inference. This technique saves a significant amount of memory by avoiding the requirement for gradient-based optimisation on larger models. It's crucial to remember that HyperModel and PEFT Plug-in both need further model training, which has its own expenses. MeZO [41] presents a memory efficient zeroth-order (ZO) optimizer for training LLMs, bypassing conventional backpropagation for gradient computation and instead using only forward passes. In order to minimise memory consumption during inference, MeZO implements an in-place solution for the traditional ZO gradient estimator and uses it to compute gradients. With this method, LLMs with up to 30 billion parameters may be fine-tuned efficiently on a single 80GB GPU, all while keeping performance close to that of backpropagation based finetuning techniques. Additionally, it substantially reduces storage demands compared to methods like LoRA and Adapter.

Some more experimental works are stated below.

Table 2.5 Comparing FLAN-T5 across data splits, gauging ROUGE-L for E2E NLG/SAMSum and AG News/CoLA accuracy in measurements. [42]

PEFT/Dataset	AG News	CoLA	E2E NLG	SAMSum
Full Tuning (low)	0.8588	0.699	0.46464	0.3876
$(IA)^3$ (low)	0.67	0.6973	0.29508	0.32924
LoRA (low)	0.8612	**0.76436**	0.48257	0.41197
BitFit (low)	**0.8808**	0.7203	**0.48825**	**0.41914**
Prompt Tune (low)	0.60684	0.68199	0.0258	0.00472
Full Tuning (med.)	**0.9212**	0.79119	0.46523	0.40908
$(IA)^3$ (med.)	0.9208	0.78161	0.48483	0.43183
LoRA (med.)	0.9148	**0.81418**	0.48364	**0.43283**
BitFit (med.)	0.9156	0.78736	**0.48539**	0.42822
Prompt Tune (med.)	0.7312	0.76245	0.44173	0.3792
Full Tuning (high)	**0.934**	0.81417	**0.48051**	0.43356
$(IA)^3$ (high)	0.9252	0.80842	0.4789	**0.43998**
LoRA (high)	0.9264	**0.83333**	0.4756	0.43485
BitFit (high)	0.9192	0.8295	0.47973	0.43098
Prompt Tune (high)	0.872	0.80567	0.45942	0.3914

Interestingly, most Parameter Efficient Fine Tuning (PEFT) algorithms show faster convergence with increased data. According to the hypothesis, PEFT approaches learn unstably at low data amounts, while full-tuning learns and overfits quickly to the small datasets. On the other hand, PEFT techniques show more stability and an improved capacity to understand the underlying data structure at larger data sizes.
The benchmarking results here are inconsistent, with no particular PEFT approach emerging as the top performer. On the other hand, there exists a clear distinction between PEFT & full-tuning based on the amount of accessible data.

Table 2.6 Different PEFT performance for SER. [43]

	Whisper Tiny	Whisper Base	Whisper Small	Wav2vec 2.0 Base	WavLM Base+
Downstream Tuning	64.21 ± 7.92	65.19 ± 7.51	$\mathbf{66.53 \pm 6.91}$	62.27 ± 6.32	65.28 ± 7.78
Adapter Tuning	57.80 ± 7.26	60.90 ± 9.03	64.36 ± 7.96	63.07 ± 6.49	$\mathbf{67.09 \pm 9.35}$
Paralle Adapter Tuning	55.18 ± 6.36	61.14 ± 6.30	61.62 ± 7.91	62.94 ± 6.68	$\mathbf{65.87 \pm 9.30}$
Embedding Prompt	57.42 ± 6.93	57.72 ± 6.24	58.40 ± 6.56	60.87 ± 6.61	$\mathbf{66.66 \pm 8.79}$
LoRa	63.42 ± 7.02	66.41 ± 8.10	66.54 ± 9.50	63.30 ± 6.44	$\mathbf{67.28 \pm 8.79}$

Here, Adapter Tuning, Parallel Adapter Tuning, & Embedding Prompt methods perform worse compared to direct downstream classification models with the Whisper model, with parallel adapter being the least effective. However, these methods improve emotion recognition for Wav2Vec 2.0 Base & WavLM Base+. LoRa enhances SER performance for WavLM Base+ & Wav2Vec 2.0 Base but may underperform with Whisper models, likely due to positional embeddings. LoRa here achieves best finetuning performance for WavLM Base+ with an average UAR of 67.3% across four datasets.

Table 2.7 RoBERTa-large (RoBL) and RoBERTa-base (RoBB) model optimisation using the GLUE benchmark, providing the Pearson/Spearman correlation for STS-B, accuracy/F1 score for MRPC and QQP, averaged matched accuracy for MNLI, and accuracy for other NLU tasks, as well as the Matthews correlation for COLA. [44]

Model	PEFT Method	#TPs	CoLA	SST2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Avg.
RoB_B	FT	124.6M	59.07	92.89	88.24/91.58	90.87/90.61	90.81/87.72	86.27	91.07	72.20	84.00/84.00
	Adapter^S	7.41M	63.32	94.31	**90.44/93.18**	91.25/90.94	90.81/86.55	87.33	92.06	73.56	85.39/85.16
	Prompt-tuning	0.61M	49.37	92.09	70.83/81.72	82.44/83.11	82.99/78.35	80.57	80.03	58.12	74.56/75.42
	Prefix-tuning	0.96M	59.31	93.81	87.25/91.03	88.48/88.32	87.75/84.09	85.21	90.77	54.51	80.89/80.88
	$(\text{IA})^3$	0.66M	59.58	93.92	87.00/90.52	90.30/90.32	87.99/84.10	83.95	90.88	71.12	83.09/83.05
	BitFit	0.69M	61.32	94.72	89.22/92.41	90.34/90.27	88.12/84.11	84.64	91.09	**77.98**	84.68/84.57
	Child-Tuning_D	-	60.33	93.58	89.22/92.20	91.14/90.93	**90.98/88.04**	**87.40**	92.20	77.62	85.31/85.29
	LoRA	0.89M	62.09	94.04	87.50/90.68	90.66/90.83	88.83/85.21	86.54	92.02	72.92	84.33/84.29
	AdaLoRA	1.03M	59.82	93.92	87.99/91.33	90.83/90.73	88.58/84.98	86.26	91.43	70.04	83.61/83.56
	MAM Adapter	46.78M	61.42	**94.87**	89.31/92.21	90.74/90.42	88.31/83.20	86.63	90.19	72.62	84.26/83.95
	$\text{ProPELT}_{\text{Adapter}}$	1.87M	**66.33**	93.85	87.25/90.82	**91.33/91.04**	89.22/85.79	86.49	**92.56**	75.54	**85.32/85.30**
	$\text{ProPELT}_{\text{Prefix}}$	10.49M	61.79	94.30	88.73/91.98	90.30/90.19	88.54/85.05	86.22	91.51	63.31	83.08/83.04
	$\text{ProPELT}_{\text{LoRA}}$	1.77M	60.38	94.11	87.42/90.87	90.76/90.55	88.90/85.55	86.84	92.04	67.39	83.48/83.47
RoB_L	FT	355.3M	65.78	95.54	89.22/92.28	91.74/91.76	89.30/86.68	89.42	93.61	81.23	86.98/87.04
	Adapter^S	19.77M	67.03	96.37	89.94/92.54	**92.58/92.42**	**92.19/88.50**	91.00	94.31	85.25	88.58/88.43
	Prompt-tuning	1.07M	61.13	94.61	73.04/81.29	78.51/78.99	80.74/75.16	68.15	89.13	60.29	75.70/76.09
	Prefix-tuning	2.03M	59.01	95.76	88.24/91.37	90.92/91.07	88.88/85.45	89.30	93.32	74.01	84.93/84.91
	$(\text{IA})^3$	1.22M	61.15	94.61	86.52/90.33	92.22/92.03	89.45/86.25	88.63	94.25	81.23	86.00/86.06
	BitFit	1.32M	**68.01**	96.10	**90.93/93.38**	91.93/91.77	89.48/86.43	89.98	94.47	87.73	88.57/88.47
	Child-Tuning_D	-	63.08	95.07	90.69/93.43	92.36/92.18	91.52/88.75	35.45	93.15	86.25	80.95/80.92
	LoRA	1.84M	64.47	**96.67**	87.50/91.19	91.66/91.44	90.15/86.91	90.76	95.00	79.78	87.00/87.03
	AdaLoRA	2.23M	65.85	94.95	89.46/92.34	92.05/91.80	89.60/86.30	90.36	94.62	77.98	86.86/86.78
	MAM Adapter	122.2M	67.39	95.81	90.12/92.77	92.44/92.18	90.87/86.65	90.62	94.31	86.62	88.52/88.29
	$\text{ProPELT}_{\text{Adapter}}$	5.40M	65.55	96.27	89.71/92.54	91.92/91.67	90.67/87.74	**91.37**	**95.20**	**88.89**	**88.70/88.65**
	$\text{ProPELT}_{\text{Prefix}}$	26.85M	62.24	96.17	90.04/92.92	90.70/90.49	89.30/86.30	90.33	94.73	79.71	86.65/86.61
	$\text{ProPELT}_{\text{LoRA}}$	4.19M	61.90	95.93	89.06/92.19	91.66/91.38	90.93/88.05	90.53	94.93	83.57	87.31/87.31

Above PEFT methods effectively reduce trainable parameters while often matching or exceeding full finetuning performance on GLUE benchmark. And for RoBERTa-base, methods like BitFit, sequential adapter, Child-TuningD, MAM adapter, LoRA, and ProPELTAdapter outperform full fine-tuning, whereas others like prompt-tuning and prefix-tuning generally underperform. A similar trend is observed for RoBERTa-large, with ProPELTLoRA also performing well. ProPELTAdapter, using only 1.50% of the trainable parameters with AdapterFusion, achieves the best overall performance. MAM Adapter combines parallel adapters & prefix-tuning for better results but requires more parameters. Sequential adapter, while parameter-intensive, performs exceptionally well. Prompt-tuning, despite using the fewest parameters, delivers the poorest performance. Child-TuningD performs well with RoBERTa-base but struggles with RoBERTa-large on specific tasks, likely due to learning rate issues.

Table 2.8 PEFT techniques on RoBERTa-large. [45]

Method	#Params	SST-2 Acc.	SST-5 Acc.	MR Acc.	CR Acc.	MPQA Acc.	Subj Acc.	TREC Acc.	Avg
Single-Sentence									
Prompt-based zero shot [4] †	0%	83.6	35	80.8	79.5	67.6	51.4	32.0	61.4
"GPT-3" in-context learning [4] †	0%	84.8(1.3)	30.6(0.9)	80.5(1.7)	87.4(0.8)	63.8(2.1)	53.6(1.0)	26.2(2.4)	70.0(1.5)
Fine-tuning [13] †	100%	81.4(3.8)	43.9(2.0)	76.9(5.9)	75.8(3.2)	72.0(3.8)	90.8(1.8)	88.8(2.1)	75.7(3.2)
LM-BFF [4] †	100%	92.3(1.0)	49.2(1.6)	85.5(2.8)	89.0(1.4)	**85.8(1.9)**	**91.2(1.1)**	88.2(2.0)	83.0(1.7)
Adapter tuning [16]	3%	92.2(1.8)	50.4(3.2)	87.7(2.6)	90.4(3.9)	73.8(5.8)	90.8(1.6)	88.6(6.4)	82.0(3.6)
Bitfit [17]	0.09%	92.8(1.6)	48.6(3.4)	87.7(3.7)	90.5(1.2)	64.3(9.4)	88.8(3.8)	85.7(10.3)	79.8(4.8)
LoRA [18]	0.1%	**93.0(2.4)**	49.2(2.3)	87.2(2.0)	90.5(2.7)	68.7(7.8)	90.1(2.4)	88.8(5.8)	81.1(3.6)
Prefix tuning [8]	0.2%-1%	91.9(1.5)	49.6(2.0)	87.4(2.3)	90.6(2.2)	74.2(3.1)	88.4(2.3)	88.5(3.7)	81.5(2.4)
IDPT [12]	0.08%	91.5(1.6)	49.0(2.5)	86.4(3.2)	90.9(2.0)	67.3(7.4)	89.7(1.2)	86.4(3.4)	80.2(3.0)
Fixed-UPL	0.1%	92.5(1.2)	**50.8(1.9)**	87.9(2.2)	91.6(1.8)	76.4(4.0)	89.3(1.7)	89.6(3.0)	82.6(2.3)
Dynamic-UPL	0.12%	92.6(1.4)	50.6(1.3)	**88.3(1.6)**	**91.6(1.4)**	78.1(2.3)	90.5(1.4)	**90.4(3.6)**	**83.2(1.9)**

Method	#Params	MNLI Acc.	MNLI-mm Acc.	SNLI Acc.	QNLI Acc.	RTE Acc.	MRPC F1.	QQP F1.	Avg
Sentence-Pair									
Prompt-based zero shot [4] †	0%	50.8	51.7	49.5	50.8	51.3	61.9	49.7	52.2
"GPT-3" in-context learning [4] †	0%	52.0(0.7)	53.4(0.6)	47.1(0.6)	53.8(0.4)	60.4(1.4)	45.7(6.0)	36.1(5.2)	49.8(2.1)
Fine-tuning [13] †	100%	45.8(6.4)	47.8(6.8)	48.4(4.8)	60.2(6.5)	54.4(3.9)	76.6(2.5)	60.7(4.3)	56.3(5.0)
LM-BFF [4] †	100%	68.3(2.5)	**70.1(2.6)**	**77.1(2.1)**	68.3(7.4)	**73.9(2.2)**	76.2(2.3)	67.0(3.0)	71.6(3.2)
Adapter tuning [16]	3%	67.6(4.3)	67.5(4.1)	74.0(4.6)	64.3(4.9)	68.8(7.4)	81.8(2.9)	67.5(4.6)	70.2(4.7)
Bitfit [17]	0.09%	67.2(5.7)	67.6(4.6)	74.4(3.7)	64.7(3.7)	66.6(10.7)	76.6(10.4)	66.8(3.7)	69.1(6.1)
LoRA [18]	0.1%	68.3(4.2)	67.4(2.7)	75.9(2.1)	**68.6(3.9)**	69.6(6.7)	81.0(4.8)	**68.1(2.6)**	71.3(3.3)
Prefix tuning [8]	0.2%-1%	68.1(3.5)	68.3(2.4)	75.4(1.8)	67.5(3.5)	70.2(5.1)	81.4(2.8)	66.8(6.5)	71.1(3.7)
IDPT [12]	0.08%	66.3(3.6)	66.0(2.1)	73.5(2.3)	66.6(3.7)	70.3(4.3)	80.6(3.4)	66.4(4.6)	70.0(3.4)
Fixed-UPL	0.1%	68.9(1.5)	68.7(1.9)	76.2(1.1)	68.2(1.4)	71.2(5.3)	81.8(1.3)	66.1(2.8)	71.6(2.2)
Dynamic-UPL	0.12%	**69.3(2.4)**	68.8(2.2)	75.9(2.1)	68.3(2.4)	72.8(3.9)	**82.1(2.1)**	66.6(3.2)	**72.0(2.6)**

The results on RoBERTa-large across 13 NLU tasks with 5 randomly sampled splits reveal Dynamic-UPL outperforming all other methods, achieving new state of the art results for few-shot learning. Dynamic-UPL improves performance compared to Prefix tuning and IDPT, especially on SST-5, TREC and RTE datasets, with fewer model parameters tuned. It also surpasses Adapter, LoRA, and BitFit across most tasks, requiring only about 0.1% of parameters. Compared to LM-BFF, a prompt based finetuning method, Dynamic-UPL achieves similar or better results on 7 datasets with significantly fewer parameters. These findings highlight Dynamic-UPL's superior generalization and efficiency in few-shot learning settings.

Table 2.9 PEFT Techniques performance on datasets [46]

	CoLA	STS-B	MRPC	RTE	CB	COPA	WSC	AVG
FullTuning	58.36±1.74	89.80±0.52	**89.55**$_{[1]}$±0.81	76.03±2.14	88.93$_{[2]}$±2.37$_{[2]}$	67.70±4.41	53.10±6.18	74.78±2.60
Random	58.35±1.05$_{[2]}$	89.81±**0.11**$_{[1]}$	88.73±0.80	72.71±3.23	**90.54**$_{[1]}$±3.39	68.80±2.64	52.88±5.97	74.55±2.46
MixOut	58.66±1.96	90.15$_{[3]}$±0.17	88.69±0.60$_{[3]}$	**77.55**$_{[1]}$±**1.64**$_{[1]}$	86.51±4.13	71.30±4.84	52.98±6.78	75.12$_{[3]}$±2.88
Bitfit	56.67±1.45	90.12±0.14$_{[3]}$	87.35±0.58$_{[2]}$	72.74±2.47	86.96±3.20	71.20±3.79	55.10±5.39	74.31±2.43
MagPruning	56.57±2.47	90.30$_{[2]}$±0.14$_{[3]}$	88.09±0.79	73.53±1.84$_{[3]}$	81.25±3.50	71.50$_{[3]}$±2.46$_{[2]}$	55.67±**2.73**$_{[1]}$	73.85±1.99$_{[2]}$
Adapter	**62.11**$_{[1]}$±1.22$_{[3]}$	90.05±0.13$_{[2]}$	89.29$_{[3]}$±0.60$_{[3]}$	76.93$_{[3]}$±2.05	87.32±4.62	69.50±2.54$_{[3]}$	57.02$_{[2]}$±5.27	76.03$_{[2]}$±2.35
LoRA	60.88$_{[3]}$±1.48	87.19±0.51	89.53$_{[2]}$±0.62	76.97$_{[2]}$±1.92	84.64±3.76	69.70±2.83	56.84$_{[3]}$±4.52	75.11±2.24$_{[3]}$
DiffPruning	58.53±1.49	89.59±0.34	78.79±6.09	69.93±7.87	86.25±2.65$_{[3]}$	72.10$_{[2]}$±2.91	53.37±3.60$_{[3]}$	72.65±3.57
ChildPruning	60.00±1.29	89.97±1.51	87.19±3.86	75.76±4.38	86.61±3.22	69.40±4.00	55.59±3.81	74.93±3.15
SAM	60.89$_{[2]}$±**0.96**$_{[1]}$	**90.59**$_{[1]}$±0.14$_{[3]}$	88.84±**0.49**$_{[1]}$	76.79±1.72$_{[2]}$	88.93$_{[2]}$±**1.75**$_{[1]}$	**74.30**$_{[1]}$±**2.45**$_{[1]}$	**59.52**$_{[1]}$±3.08$_{[2]}$	**77.12**$_{[1]}$±**1.51**$_{[1]}$

The study finds that parameter-efficient models generally outperform the FullTuning model, aligning with prior research and supporting theoretical analysis on their superior generalization capabilities. These models also demonstrate greater stability compared to the FullTuning model, corroborating theoretical stability analysis. Interestingly, even the Random model surpasses the FullTuning model, indicating that sparsity enhances performance. Additionally, the proposed SAM model outperforms several baseline models across various tasks, ranking among the top three in most evaluations.

Table 2.10 Training time (in hour) analysis [46]

	CoLA	MRPC	STS-B	RTE	CB	COPA	WSC	AVG
FullTuning	0.74	1.04	1.90	2.63	3.18	0.66	1.55	1.67
ChildPruning	3.91	1.36	2.12	3.77	3.13	0.91	2.34	2.51
Adapter	0.89	1.24	3.43	3.51	1.16	0.41	0.47	1.59
LoRA	1.39	1.23	2.12	4.87	2.15	0.60	1.43	1.97
Bitfit	2.30	1.70	6.70	7.02	2.18	1.20	0.87	3.14
DiffPruning	0.60	1.07	6.61	5.30	2.86	1.21	1.06	2.67
Random	0.41	3.36	4.34	6.70	1.60	1.27	0.69	2.62
SAM	2.31	1.68	2.78	3.15	2.70	0.72	0.82	2.02

Adapter and LoRA models outperform the FullTuning model by tuning only a few new parameters, while other parameter-efficient models take more time due to the use of masks. The SAM model, despite needing masks, outperforms most models except Adapter and LoRA due to faster convergence. These parameter-efficient models, though slower to train, require less storage space, which is advantageous for handling multiple downstream tasks.

Table 2.11 SURE benchmark on pre-trained Wav2Vec 2.0. [47]

Method	#Parameters	SER (acc % / w-f1) ↑		SR (acc %) ↑		ASR (wer) ↓		KS (acc %) ↑
		ESD	MELD	ESD	VCTK	ESD	FLEURS	Speech Command
Fine Tuning	315,703,947	**96.53**	42.93	99.00	92.36	0.2295	**0.135**	99.08
Adapter	25,467,915 (8.08%)	94.07	41.58	98.87	96.32	0.2290	0.214	**99.19**
Prefix Tuning	1,739,787 (**0.55%**)	90.00	44.21	**99.73**	**98.49**	**0.2255**	0.166	98.86
LoRA	3,804,171 (1.20%)	90.00	**47.05**	99.00	97.61	0.2428	0.149	98.28
ConvAdapter	2,952,539 (0.94%)	91.87	46.30	99.60	97.61	0.2456	0.206	98.99

Here, Parameter-efficient approaches in speech processing tasks perform on par with or better than thorough fine-tuning, particularly excelling on the Emotional Speech Dataset (ESD) and demonstrating effectiveness in speaker recognition (SR) tasks. Notably, the proposed ConvAdapter achieves significant parameter reduction, utilizing only 0.94% of trainable parameters of fine-tuning. While slightly trailing other adapters in performance, it presents a promising alternative for efficient transfer learning. Combining ConvAdapter with other methods like prefix fine-tuning or LoRA could further enhance results, particularly in challenging datasets like MELD.

Table 2.12 Performance of different methods in the SUPERB benchmark. [48]

Method	Params	ASR	PR	SD	SID	SF	IC	KS
FT	94.7M	6.35	**2.45**	9.32	66.48	84.87	99.10	95.87
Baseline	0	7.09	7.74	7.05	64.78	86.25	96.39	95.32
Houlsby	0.60M	5.88	3.00	**4.00**	**87.71**	85.87	**99.60**	97.17
AdapterBias	0.02M	**5.54**	4.19	5.48	77.38	86.60	99.50	97.30
BitFit	0.10M	9.34	4.23	5.13	83.68	**87.40**	99.50	**97.33**
LoRA	0.29M	6.94	8.74	7.39	62.90	86.25	96.57	96.59
Prefix	0.10M	6.56	4.18	8.17	71.87	85.85	99.31	97.05
Weighted-sum	12	6.42	5.41	5.88	81.42	88.53	98.34	96.30

Efficient methods generally outshine Baseline and full fine-tuning (FT), particularly in recognition and speaker tasks like ASR, PR, SD, and SID. Houlsby notably improves SID accuracy by 23% compared to Baseline and maintains consistently high performance due to its larger parameter size. However, LoRA performs poorly, even worse than Baseline in some tasks. Weighted-sum stands out for its effectiveness, achieving comparable performance to more complex methods with minimal additional parameters, as shown in the SUPERB benchmark for speech tasks.

Table 2.13 Project-specific code summary task on the Bleu-4 Scores. [18]

Task	Project-specific Code Summarization				
Dataset	Spring-boot	Spring-framework	Spring-security	Apache Flink	Apache Kafka
CodeT5$_{w/o-FT}$	1.60	1.57	0.76	0.95	1.80
CodeT5$_{Full(100\%)}$	**38.45**	32.67	36.61	**31.38**	**26.26**
CodeT5$_{Adapter(0.5\%,r=30)}$	37.89	31.90	35.84	28.36	25.87
CodeT5$_{LoRA(0.5\%,r=10)}$	37.29	31.13	**36.71**	30.84	26.18
CodeT5$_{PA(0.5\%,r=30)}$	36.83	31.07	34.57	29.62	25.76
CodeT5$_{MHM(0.5\%,r=15,l=16,m=350)}$	38.15	**33.28**	35.37	31.11	26.02

In this case, CodeT5 models were trained on one project dataset using the full finetuning and PEFT approaches, and their performance was assessed on a different project dataset. Full fine-tuning frequently performed better than PEFT techniques in project-specific code summarization tasks; optimal performance was attained when models were fine-tuned and assessed on the same dataset. However, PEFT methods showed better results in cross-project scenarios, with relative improvements when fine-tuning on one dataset and testing on others. This suggests that while preserving outcomes that are comparable to comprehensive fine-tuning, PEFT approaches improve the model's transferability between projects. The potential of PEFT approaches to improve transfer learning capacity by utilising frozen pre-trained parameters may account for their higher performance in certain settings. Without any fine-tuning, direct inference on the pre-trained CodeT5 model could not produce sufficient evaluation performance, indicating the necessity of adjusting in some way.

CHAPTER 3 - METHODOLOGY

3.1 Pre-Training Objectives

The pre-training goals of LLMs are explained in this section.

1. Full Language Modeling:
   - An autoregressive language modelling goal in which the model's job is to forecast tokens in the future based on tokens in the past.

2. Prefix Language Modeling:
   - A non-causal training goal in which the loss is determined solely by using the target tokens that are still there after a randomly selected prefix.

3. Masked Language Modeling:
   - Tokens or spans (a series of tokens) are randomly masked in this training objective, and the model's job is to forecast masked tokens based on the context of the past and future.

4. Unified Language Modeling:
   - It combines non-causal, masked, & causal language-training goals. In the context of masked language modelling, Instead of being bilateral, attention is either focused from left to right or from right to left.

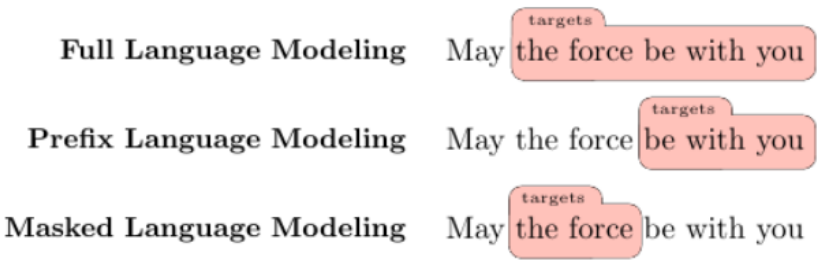


Fig. 3.1: An example of language model training objectives. [49]

3.2 Distributed LLM Training

1. Data Parallelism:
 - This technique duplicates the model across several devices, dividing a batch of data across them. All of the devices' weights are synchronised at the conclusion of each training cycle.

2. Tensor Parallelism:
 - Tensor computations are split up among several devices using tensor parallelism. It is sometimes referred to as intra-layer model parallelism or horizontal parallelism.

3. Pipeline Parallelism:
 - This technique divides up model layers among various devices. Another name for this is vertical parallelism.

4. Model Parallelism:
 - It combines elements of pipeline and tensor parallelism.

5. 3D Parallelism:
 - It is a combination of tensor, model, and data parallelism.

6. Optimizer Parallelism:
 - It lowers communication costs and memory use by gradient partitioning, parameter partitioning, and optimizer state partitioning among devices. It is also referred to as zero redundancy optimizer.

3.3 Model Adaptation

This section gives a summary of the important phases involved in adapting LLMs, from pretraining going to finetuning for specific tasks, and how they are utilized. When we mention alignment-tuning, we're referring to aligning with human preferences, although the term can have different meanings in other contexts.

1. Pre-Training:

Initially, the model undergoes self-supervised training on a vast dataset to predict subsequent tokens based on the given input. LLMs come in different architectures, including encoder decoder & decoder only models, with various building blocks & loss functions.

2. Fine-Tuning:

Fine-tuning LLMs can be approached in different ways:

- Transfer Learning: While pretrained LLMs perform well on many tasks, fine-tuning with task-specific data can further enhance performance for a specific downstream task.

- Instruction-Tuning: This involves finetuning the pretrained model using data formatted as instructions paired with input-output examples. These instructions typically cover multiple natural language tasks, guiding model response appropriately to different prompts. Task performance and zero-shot generalisation are enhanced by this approach.

- Alignment-tuning: To mitigate issues like generating false, biased, or harmful content, LLMs are aligned using human feedback. This involves updating the model to avoid undesirable responses, ensuring it remains helpful, honest, and harmless (HHH). Techniques like reinforcement learning with human feedback (RLHF) are employed, where a model initially fine-tuned on demonstrations is further trained using reward modeling and reinforcement learning.

- Parameter-Efficient Tuning: Given the substantial memory and computing requirements for training LLMs, researchers have developed techniques to fine-tune models more efficiently by updating only a few parameters or adding new ones.

3. Prompting/Utilization:

One method of asking trained LLMs to produce answers is called prompting. Different prompt styles can be used to prompt LLMs; sometimes they will adapt to instructions without requiring additional fine-tuning, and other times they will require fine-tuning based on data comprising multiple prompt styles.

- Zero-Shot Prompting: This method enables LLMs to answer questions they have never seen before, without requiring examples in the prompt.

- In-context Learning: Often called "few-shot learning," this technique entails displaying multiple input-output pairings to the model in order to elicit the intended response.

- Reasoning in LLMs: LLMs can handle task planning, logical problems, and critical thinking through reasoning, which can be enhanced by various prompting styles or training on reasoning datasets.

- Chain-of-Thought (CoT): In this method, prompts include reasoning steps along with inputs-outputs, enabling the model to produce results with step-by-step reasoning.

- Self-Consistency: This improves CoT via generating multiple-response and selecting most frequent one.

- Tree-of-Thought (ToT): This approach explores multiple-reasoning paths, allowing for forward-looking and backtracking in problem-solving.

- Single-Turn Instructions: Here, LLMs are questioned once with all relevant data, and they can produce answers in a few-shot or zero-shot scenario by comprehending the context.

- Multi-Turn Instructions: For complex tasks requiring multiple interactions, feedback & responses using other tools as inputs for subsequent rounds with the LLM, commonly used in autonomous agents.

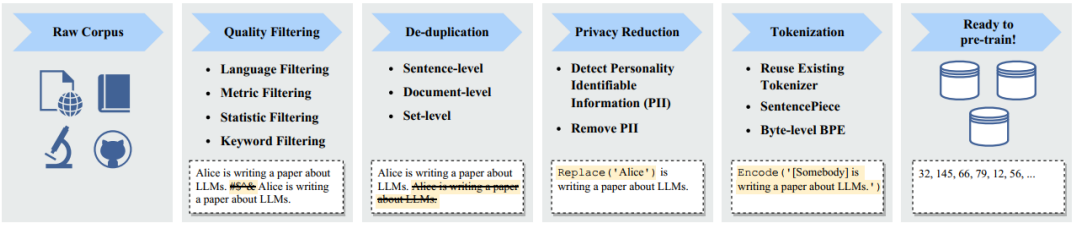


Fig. 3.2: Standard data pre-processing workflow for LLMs pre-training. [50]

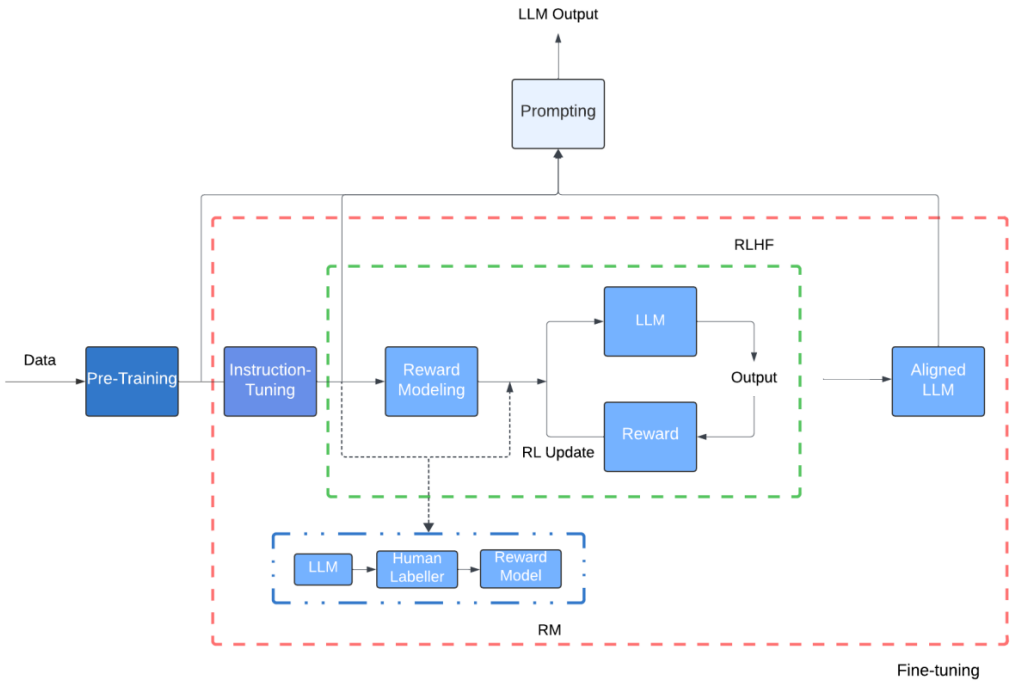


Fig. 3.3: Simple flowchart showcasing several phases of LLMs. [2]

3.4 Model selection and deployment guidelines

If you want to make the most of Language Models (LLMs) for NLP tasks and improve your applications and systems, just follow these steps:

1. Figure out the Task:
- Decide which specific NLP task you want the LLM to handle first. Anything from text categorization and sentiment analysis to text production or question-answering could be included.

2. Pick the Right Model:
- Select a pretrained LLM based on the needs of your assignment. Think about models such as RoBERTa, BERT, or GPT-3, each with pros and cons of their own.

3. Customize the Model:
- It's now time to adapt the chosen pre-trained model to your particular assignment. Adjust it using your personal dataset. To achieve the best results, change variables like epochs, batch size, & learning rate.

4. Evaluate the Model:
- Test performance on fine-tuned model on a separate dataset. Use metrics like precision, recall, accuracy, and F1 score to see how well it's doing. This helps ensure effective task performance and lets you identifying improvement areas.

5. Get the Model in Action:
- Include the optimised model in your system or application. Make it available via a user interface or an API. When it's in production, don't forget to use logging and monitoring to track its performance.

6. Keep an Eye on the Model:
- Keep a close watch on how the model performs in production. If necessary, retrain it based on its performance. Regularly evaluate its performance and adjust parameters or train it on new data if its performance starts to decline.

7. Keep Improving the Model:
- Gather feedback from users to enhance the model's performance. Keep the model updated with fresh data to ensure it stays relevant and aligned with the ever-evolving needs and trends.

3.5 Ethical guidelines

To make sure we develop and use Language and Learning Models (LLMs) responsibly, with a focus on user privacy, fairness, ethics, transparency, competition, collaboration, and environmental impact, here are some guidelines. By adhering to these guidelines, LLMs can minimise any possible negative impacts while still making a good contribution to society.

1. Protect User Privacy:
 - Keep user privacy a priority by implementing practices like minimizing data collection, anonymizing user content, and using encryption to secure user-generated data.

2. Reduce Bias:
 - Actively seek out and reduce biases in LLMs through the use of inclusive and varied training data, methods for detecting bias, and comprehensive evaluation metrics.

3. Address Ethical Considerations:
 - Take into account the potential for LLMs to be used in harmful ways and focus on developing models that benefit society. Build models with accountability, transparency, and responsibility in mind.

4. Enhance Transparency:
 - Improve the transparency and explainability of LLMs by using techniques like attention mechanisms & model interpretation tools. This will help users trust the system.

5. Encourage Competition:
 - Encourage cooperation between government, business, and academics to prevent monopolisation of LLM development and implementation. This encourages creativity and sensible use.

6. Foster Collaboration:
 - By sharing research findings and best practices, and by opening-sourcing models, you can foster collaboration between researchers, developers, and industry.

7. Minimize Environmental Impact:
 - In order to lessen the environmental impact of LLMs, try to develop more energy-efficient models and investigate alternate training strategies.

8. Consider Optimization Implications:
 - Be aware that optimizing LLMs could perpetuate inequalities & introduce new exploitation forms. Carefully consider these ethical implications during development and deployment.

3.6 Metrics

When it comes to evaluating how well LLMs (Language Models) perform, there are various metrics that we can use. These metrics give us different insights into their efficiency. To get a comprehensive understanding of an LLM, we often look at metrics like accuracy and zero-shot ability alongside others.

1. Number of Parameters:
   - The total number of variables or learnable weights that the LLM must optimise during training is indicated by this measure. Parameters are like the weights in the connections between neurons or attention layers. Generally, if there are more parameters, the LLM becomes more expressive. But keep in mind, having more parameters also means you'll need more computational resources and memory for both training and inference.

2. Model Size:
   - This is the amount of memory or disc space needed by the LLM to hold all of its components, including weights and biases. The model size is closely related to the number of parameters. Usually, more parameters mean a larger model size. However, the type of data used to represent parameters and the model architecture can also affect the overall size.

3. Compression Ratio:
   - The ratio of the uncompressed LLM's initial size to its compressed size is provided by this measure. A higher compression ratio means that the LLM has been compressed more efficiently. In other words, it has been significantly reduced in size while still maintaining its functionality and performance.

4. Inference Time:
   - This is a measurement of how long it takes the LLM to analyse and provide replies for input data during inference; it is also referred to as latency. Inference time is crucial for real-world applications where the LLM needs to respond to user queries or process large amounts of data in real-time.

5. Floating Point Operations (FLOPs):
   - This metric tells us the number of arithmetic operations involving floating-point numbers (usually 32-bit or 16-bit) that the LLM performs when processing input data. FLOPs give us an idea of the computational requirements of an LLM and allow us to compare the efficiency between different LLMs or compression techniques.

3.7 Best Practices

To improve your work it is best to follow these practices:

1. Explicit Reporting of Parameter Counts:
   - Authors should clearly specify the type of parameter count (trainable, changed, rank) in their papers, ideally reporting all of them. This will enhance understanding and enable more accurate comparisons between methods.

2. Evaluation with Different Model Sizes:
   - Methods should be assessed using various model sizes providing comprehensive understanding to their strengths and limitations. This is especially important as many studies focus primarily on BERT.

3. Comparisons to Similar Methods:
   - In addition to comparing new methods with popular approaches (e.g., LoRa, BitFit, Adapters), it is crucial to compare them with other conceptually and architecturally similar techniques. This will offer a more thorough understanding of a method's performance and its relative strengths.

4. Standardized PEFT Benchmarks and Competitions:
   - Developing standardized benchmarks and competitions will allow participants to compete under the same conditions, facilitating direct comparisons. These benchmarks should provide standardized data and models at different scales and include a standardized way to evaluate GPU memory consumption.

5. Emphasis on Code Clarity and Minimal Implementations:
   - The community should prioritize easy-to-understand code with simple, reusable implementations. Such clarity not only aids understanding but also increases the likelihood of methods being adopted and cited by other researchers.

CHAPTER 4 – EXPERIMENTAL ANALYSIS

4.1 Need for Model Compression Methods

```
In [10]:  1 trainer.train()
executed in 3.21s, finished 11:16:12 2023-12-13
   1530     result = None

File ~\anaconda3\envs\test\lib\site-packages\transformers\models\t5\modeling_t5.py:561, in T5Attention.forward(self, hidden_s
tates, mask, key_value_states, position_bias, past_key_value, layer_head_mask, query_length, use_cache, output_attentions)
    558     position_bias_masked = position_bias
    560 scores += position_bias_masked
--> 561 attn_weights = nn.functional.softmax(scores.float(), dim=-1).type_as(
    562     scores
    563 )  # (batch_size, n_heads, seq_length, key_length)
    564 attn_weights = nn.functional.dropout(
    565     attn_weights, p=self.dropout, training=self.training
    566 )  # (batch_size, n_heads, seq_length, key_length)
    568 # Mask heads if we want to

OutOfMemoryError: CUDA out of memory. Tried to allocate 96.00 MiB. GPU 0 has a total capacty of 4.00 GiB of which 0 bytes is
free. Of the allocated memory 3.42 GiB is allocated by PyTorch, and 26.46 MiB is reserved by PyTorch but unallocated. If rese
rved but unallocated memory is large try setting max_split_size_mb to avoid fragmentation.  See documentation for Memory Mana
gement and PYTORCH_CUDA_ALLOC_CONF
```

As you can see that when training the model it runs out of memory as the model is too big for the hardware to support.

So we use can use different PEFT methods to train it.

```
In [47]:
from peft import LoraConfig, get_peft_model, TaskType

lora_config = LoraConfig(
    r=16, # Rank
    lora_alpha=16,
    target_modules=["q", "v"],
    lora_dropout=0.1,
    bias="none",
    task_type=TaskType.SEQ_2_SEQ_LM # FLAN-T5
)
executed in 20ms, finished 11:20:57 2023-12-13

In [48]:
peft_model = get_peft_model(original_model, lora_config)

print(print_number_of_trainable_model_parameters(peft_model))
executed in 140ms, finished 11:20:58 2023-12-13

trainable model parameters: 1769472
all model parameters: 249347328
percentage of trainable model parameters: 0.71%
```

e.g. above we have used LORA for model compression.
It brings down the trainable parameter down to only 0.71% which drastically reduces the memory needed to train the model.

4.2 Text Generation

Below are the snapshots of the training steps & the result.
As a model was trained on a very small subset of data, due to low resources thus it does not gives a good enough convergence rate & result but is enough for generating text.

In [12]:
```
peft_trainer.train()

peft_model_path="./peft-dialogue-summary-checkpoint-local"

peft_trainer.model.save_pretrained(peft_model_path)
tokenizer.save_pretrained(peft_model_path)
```

executed in 25m 1s, finished 21:10:04 2023-12-13

[6120/6120 24:59, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum
1	3.649400	3.372850	0.145343	0.021676	0.116550	0.128481
2	3.555800	3.362702	0.154233	0.022780	0.125848	0.136337
3	3.517700	3.360362	0.152215	0.022886	0.119871	0.134752
4	3.487600	3.356078	0.144497	0.022359	0.112126	0.128062
5	3.436500	3.365880	0.151877	0.023762	0.120840	0.134484
6	3.398900	3.366742	0.155215	0.023910	0.122680	0.138239
7	3.402300	3.371326	0.159276	0.023752	0.125692	0.139608
8	3.398200	3.371223	0.159361	0.023635	0.126308	0.141137
9	3.369100	3.372497	0.160902	0.023677	0.125164	0.140897
10	3.349300	3.373973	0.159481	0.023826	0.125129	0.139010

In [21]:
```
peft_model.to('cuda')

my_question = "What do you think about the benefit of Artificial Intelligence?"
inputs = "Please answer to this question: " + my_question

inputs = tokenizer(inputs, return_tensors="pt").to('cuda')

outputs = peft_model.generate(**inputs, max_new_tokens=100)

answer = tokenizer.decode(outputs[0])

print("Answer:", answer)
```

executed in 712ms, finished 21:21:29 2023-12-13

```
Answer: <pad> I think that artificial intelligence is a great way to improve our lives. It is a great way to make our lives bet
ter.</s>
```

4.3 Text Summarization

Below are the snapshots of the training steps & the result.

In [50]:
```
peft_trainer.train()

peft_model_path="./peft-dialogue-summary-checkpoint-local"

peft_trainer.model.save_pretrained(peft_model_path)
tokenizer.save_pretrained(peft_model_path)
```

executed in 21.4s, finished 11:21:19 2023-12-13

[100/100 00:18, Epoch 0/1]

Step	Training Loss
10	41.400000
20	23.987500
30	9.021900
40	4.528100
50	4.050000
60	3.489100
70	2.751600
80	2.139100
90	1.843000
100	1.642200

In [52]:
```
dialogues = dataset['test'][0:10]['dialogue']
human_baseline_summaries = dataset['test'][0:10]['summary']

original_model_summaries = []
peft_model_summaries = []

for idx, dialogue in enumerate(dialogues):
    prompt = f"""
Summarize the following conversation.

{dialogue}

Summary: """

    input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to('cuda')

    human_baseline_text_output = human_baseline_summaries[idx]

    original_model_outputs = original_model.generate(input_ids=input_ids, generation_config=GenerationConfig(max_new_tokens=200))
    original_model_text_output = tokenizer.decode(original_model_outputs[0], skip_special_tokens=True)

    peft_model_outputs = peft_model.generate(input_ids=input_ids, generation_config=GenerationConfig(max_new_tokens=200))
    peft_model_text_output = tokenizer.decode(peft_model_outputs[0], skip_special_tokens=True)

    original_model_summaries.append(original_model_text_output)
    peft_model_summaries.append(peft_model_text_output)

zipped_summaries = list(zip(human_baseline_summaries, original_model_summaries, peft_model_summaries))

df = pd.DataFrame(zipped_summaries, columns = ['human_baseline_summaries', 'original_model_summaries', 'peft_model_summaries'])
df
```

executed in 6.05s, finished 11:21:26 2023-12-13

Out[52]:

	human_baseline_summaries	original_model_summaries	peft_model_summaries
0	Ms. Dawson helps #Person1# to write a memo to ...	Memo to be distributed to all employees by 4 pm.	Staff will be required to take a decision on t...
1	In order to prevent employees from wasting tim...	This memo is for internal and external communi...	Memo distributed to all employees by 4 pm.
2	Ms. Dawson takes a dictation for #Person1# abo...	Employees are restricted to email corresponden...	Request a dictation from #Person1# to all empl...
3	#Person2# arrives late because of traffic jam....	You're finally here! What took so long?	You're finally here.
4	#Person2# decides to follow #Person1#'s sugges...	@Person1#: I got stuck in traffic again. There...	#Person1# is a little stressed about the conge...
5	#Person2# complains to #Person1# about the tra...	You might be stuck in traffic again during the...	The public transport system is good.
6	#Person1# tells Kate that Masha and Hero get d...	Masha and Hero are getting divorced.	@Person1#: Well, I always thought they are wel...
7	#Person1# tells Kate that Masha and Hero are g...	Masha and Hero are getting divorced.	@Por1 : Masha and Hero are getting divorced.
8	#Person1# and Kate talk about the divorce betw...	Masha and Hero are getting a divorce.	#Person1#: #Person#: Masha and Hero are gettin...
9	#Person1# and Brian are at the birthday party ...	Brian's birthday party.	The birthday party is on.

4.4 Analysis & Result

As shown in above code snapshots, all three components are properly working.

In [54]:
```
print("Absolute percentage improvement of PEFT MODEL over ORIGINAL MODEL")

improvement = (np.array(list(peft_model_results.values())) - np.array(list(original_model_results.values())))
for key, value in zip(peft_model_results.keys(), improvement):
    print(f'{key}: {value*100:.2f}%')
```

executed in 12ms, finished 11:21:28 2023-12-13

```
Absolute percentage improvement of PEFT MODEL over ORIGINAL MODEL
rouge1: 3.83%
rouge2: 2.06%
rougeL: 6.71%
rougeLsum: 7.00%
```

The result here shows how PEFT model performed better than the original model with minimal computation resources.

4.5 Hardware & Libraries Used

- Graphic Card : NVIDIA GeForce RTX 3050 4GB VRAM
- Libraries : tensorflow, tflearn, torch, nltk, transformers, evaluate.
- LLM : flan-t5-base
- PEFT Technique : LoRA

CHAPTER 5 - CONCLUSION

5.1 Challenges for LLMs

LLMs like GPT-4 have made great strides in natural language processing, but they also come with their fair share of challenges. Let's take a closer look:

1. Computational Cost: Training LLMs requires a massive amount of computational resources, which can drive up costs and raise concerns about the environmental impact.

2. Bias and Fairness: Due to LLMs' ability to detect and even magnify biases in their training data, there are ethical concerns that must be addressed.

3. Overfitting: LLMs can sometimes get too caught up in the specific examples they were trained on, resulting in illogical responses. Striking the right balance between memorization and generalization is crucial.

4. Economic and Research Inequality: The high costs associated with LLM development can create a situation where only well-funded organizations can participate, exacerbating existing inequalities.

5. Reasoning and Planning: LLMs struggle when it comes to tasks that require reasoning and planning, often falling short in common-sense scenarios.

6. Hallucinations: LLMs have the tendency to generate responses that may sound plausible but are actually incorrect or inconsistent.

7. Prompt Engineering: Designing effective prompts is essential as it greatly influences the quality of the LLM's outputs.

8. Limited Knowledge: Pre-trained information can become outdated, and retraining can be quite costly. Augmenting retrieval techniques can help, but it requires some adaptation.

9. Safety and Controllability: Ensuring that LLMs do not produce harmful or inappropriate content is a significant concern that needs to be addressed.

10. Multi-Modality: Integrating diverse data like text, images, and videos presents challenges in aligning the data and increasing the computational demands.

11. Catastrophic Forgetting: Fine-tuning LLMs can sometimes cause them to forget previously learned information, which can be problematic.

12. Adversarial Robustness: LLMs are vulnerable to adversarial attacks, which highlights the need for robust evaluation tools, especially in safety-critical applications.

13. Interpretability and Explainability: It might be challenging to comprehend LLMs' decision-making processes since they frequently function like "black boxes." Trust and acceptance may be impacted by this lack of transparency.

14. Privacy Concerns: Using LLMs raises concerns about data privacy and the potential for extracting sensitive information from the models.

15. Real-Time Processing: The high computational demands of LLMs can hinder their ability to process information in real-time, particularly in mobile and edge computing environments.

16. Long-Term Dependencies: LLMs struggle with maintaining context during long or multi-turn conversations, which can affect the overall coherence of their responses.

17. Hardware Acceleration: The increasing size of LLMs is surpassing the capabilities of existing hardware, making model inference costly. We need advancements in hardware and model quantization to address this issue.

18. Regulatory and Ethical Frameworks: To control the social and ethical ramifications of LLMs, regulatory monitoring and ethical frameworks must be established.

5.2 Recommendations for Optimal Performance of LLMs

LLMs, such as GPT-4, are super effective at a bunch of different tasks because they know a ton and can learn like champs. To get the most out of these models, there are a few strategies you should keep in mind:

- Go for the Fancy Architecture: One of the most sophisticated language models available at the moment is GPT-4. It's really good at generating content that actually makes sense, so it's a top choice for all sorts of tasks.

- Give Detailed Prompts with Task Context and Relevant Info: How well LLMs perform depends a lot on how clear and specific your input prompts are. If you provide lots of details about the task and relevant info, it'll help the model understand what it needs to do and give more accurate responses.

- Include Relevant Info in Your Prompts: If you want the model to give precise and focused responses, throw in some extra relevant info in your prompts. In particular,

with tasks requiring specialised expertise, such as medical or coding, the output can be improved by include pertinent information and instructions in the prompt.

- Try Different Prompt Techniques: Given the complexity and unpredictability of LLMs, it is worthwhile to experiment with various prompt strategies in order to improve performance. You may try asking leading questions, providing more detailed instructions, "double-quoting keywords," or presenting the information in a different way. Who knows, maybe you'll achieve even greater outcomes!

5.3 PEFT Technique Selection

Because prompt tuning makes use of the embedding layer—which has sufficient contextual information after navigating the frozen language model layers—it is perfect for applications like Named Entity Recognition. This indicates that the assignment can be completed by concentrating only on the embeddings. In addition, Prompt Tuning is an effective choice that works well even with a small computation budget because it requires very few parameters and has a straightforward layer structure O(1) complexity.

LoRA works well for activities involving answering questions. It assists the model in determining the connections between words and phrases in the inquiry and the response by working on the attention queries and values. The efficacy of LoRA is supported by its impressive results in multiple-choice quality assurance assignments. The model utilises critical information more effectively thanks to the configurable scaling integration. Subsequent to the Transformer attention block, Tiny-Attention Adapters have the potential to enhance attention even more, as well as the calibre of hidden representations.

For tasks like Data-to-Text and Summarization, both LoRA and Prefix Tuning can be effective. Research by [12],[16],[51],[52] shows that these techniques offer similar performance, but the choice depends on your computational resources. LoRA, with its fewer parameters and better layer efficiency, tends to be more efficient. These findings are supported by their characteristics in PEFT-Ref. Although Adapters perform well in generation tasks, research by [51] suggests they have lower faithfulness scores comparing to full finetuning and Prefix Tuning. This is because Adapters use both the feed-forward and attention blocks. [53] discovered that there is a lot of redundancy in the feed-forward block, and altering it can make generating jobs less faithful.

In summary, the ideal PEFT technique depends on complexity of your task. For tasks requiring complex reasoning [54], techniques using attention modules are recommended. For tasks involving the addition of new concepts, feed-forward modules can be used to store new knowledge [55] (Dai et al., 2022). For simpler tasks, adding task-specific information through the embedding layer is sufficient.

5.4 Conclusion

Although recent advancements have made it possible to train larger models, we must stress the necessity to scale datasets responsibly while putting an emphasis on high-quality data. Only high-quality data that has been carefully collected and managed can benefit from scaling to larger datasets. Accurate language modelling and subsequent tasks depend on a correct train-test set separation. Furthermore, because massive web-scraped datasets may contain harmful language, prejudices, and private information, training on billions of tokens presents ethical and privacy issues. Comprehensive dataset introspection becomes more crucial as datasets get larger in order to solve these problems.

Furthermore, we see that, on our baselines, PEFT approaches outperform fully supervised fine-tuning in general at low- to medium-resource levels, but they converge more slowly. Furthermore, we find that downstream performance is significantly impacted by changes in attention levels and the choice of later layers.

By this study we hope that it can help in the better understanding & choice of these techniques to help in your work. To further enhance performance and tackle the difficulties associated with large-scale data management, future studies should investigate the integration of several PEFT techniques.

REFERENCES

[1].....“A Survey on Evaluation of Large Language Models | ACM Transactions on Intelligent Systems and Technology.” Accessed: May 26, 2024. [Online]. Available: https://dl.acm.org/doi/full/10.1145/3641289?casa_token=_vcfJs0OIJIAAAAA%3AtAyFrOuSVhHe9LoK7kR5qBLsIetEAoyweibBjHMPMgSN47C_5mVLTatMKDuqIQvkyPPR9SsRucHb_qo
[2].....H. Naveed *et al.*, “A Comprehensive Overview of Large Language Models.” arXiv, Apr. 09, 2024. doi: 10.48550/arXiv.2307.06435.
[3].....S. Pahune and M. Chandrasekharan, “Several categories of Large Language Models (LLMs): A Short Survey,” *IJRASET*, vol. 11, no. 7, pp. 615–633, Jul. 2023, doi: 10.22214/ijraset.2023.54677.
[4].....M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. Sahraoui, “Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models.” arXiv, Jan. 18, 2024. doi: 10.48550/arXiv.2308.10462.
[5].....Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, “Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey.” arXiv, Apr. 29, 2024. doi: 10.48550/arXiv.2403.14608.
[6] V. Lialin, V. Deshpande, and A. Rumshisky, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning.” arXiv, Mar. 27, 2023. doi: 10.48550/arXiv.2303.15647.
[7]…Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, “AMC: AutoML for Model Compression and Acceleration on Mobile Devices,” presented at the Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 784–800. Accessed: May 25, 2024. [Online]. Available: https://openaccess.thecvf.com/content_ECCV_2018/html/Yihui_He_AMC_Automated_Model_ECCV_2018_paper.html
[8]. .F. Tung and G. Mori, “Similarity-Preserving Knowledge Distillation,” presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1365–1374. Accessed: May 25, 2024. [Online]. Available: https://openaccess.thecvf.com/content_ICCV_2019/html/Tung_Similarity-Preserving_Knowledge_Distillation_ICCV_2019_paper.html
[9].....A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A Survey of Quantization Methods for Efficient Neural Network Inference,” in *Low-Power Computer Vision*, Chapman and Hall/CRC, 2022.
[10] D. Yin, Y. Yang, Z. Wang, H. Yu, K. Wei, and X. Sun, “1% VS 100%: Parameter-Efficient Low Rank Adapter for Dense Predictions,” presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 20116–20126. Accessed: May 25, 2024. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2023/html/Yin_1_VS_100_Parameter-Efficient_Low_Rank_Adapter_for_Dense_Predictions_CVPR_2023_paper.html
[11].....B. Lester, R. Al-Rfou, and N. Constant, “The Power of Scale for Parameter-Efficient Prompt Tuning.” arXiv, Sep. 02, 2021. doi: 10.48550/arXiv.2104.08691.
[12].....X. L. Li and P. Liang, “Prefix-Tuning: Optimizing Continuous Prompts for Generation.” arXiv, Jan. 01, 2021. doi: 10.48550/arXiv.2101.00190.

[13].....N. Houlsby *et al.*, “Parameter-Efficient Transfer Learning for NLP,” in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, May 2019, pp. 2790–2799. Accessed: May 25, 2024. [Online]. Available: https://proceedings.mlr.press/v97/houlsby19a.html
[14].....H. Zhao, H. Tan, and H. Mei, “Tiny-Attention Adapter: Contexts Are More Important Than the Number of Parameters.” arXiv, Oct. 18, 2022. doi: 10.48550/arXiv.2211.01979.
[15].…R. Karimi Mahabadi, J. Henderson, and S. Ruder, “Compacter: Efficient Low-Rank Hypercomplex Adapter Layers,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2021, pp. 1022–1035. Accessed: May 26, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/081be9fdff07f3bc808f935906ef70c0-Abstract.html
[16]. H. Liu *et al.*, “Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, Dec. 2022.
[17].J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, “Towards a Unified View of Parameter-Efficient Transfer Learning.” arXiv, Feb. 02, 2022. doi: 10.48550/arXiv.2110.04366.
[18].…J. Liu, C. Sha, and X. Peng, “An Empirical Study of Parameter-Efficient Fine-Tuning Methods for Pre-Trained Code Models,” in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2023, pp. 397–408. doi: 10.1109/ASE56229.2023.00125.
[19].....M. Sabry and A. Belz, “PEFT-Ref: A Modular Reference Architecture and Typology for Parameter-Efficient Finetuning Techniques.” arXiv, Oct. 19, 2023. doi: 10.48550/arXiv.2304.12410.
[20]J. Hoffmann *et al.*, “Training Compute-Optimal Large Language Models.” arXiv, Mar. 29, 2022. doi: 10.48550/arXiv.2203.15556.
[21].A. Rücklé *et al.*, “AdapterDrop: On the Efficiency of Adapters in Transformers.” arXiv, Oct. 05, 2021. doi: 10.48550/arXiv.2010.11918.
[22].....J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, “AdapterFusion: Non-Destructive Task Composition for Transfer Learning.” arXiv, Jan. 26, 2021. doi: 10.48550/arXiv.2005.00247.
[23].....S. He, L. Ding, D. Dong, M. Zhang, and D. Tao, “SparseAdapter: An Easy Approach for Improving the Parameter-Efficiency of Adapters.” arXiv, Nov. 10, 2022. doi: 10.48550/arXiv.2210.04284.
[24]..L. Hedegaard, A. Alok, J. Jose, and A. Iosifidis, “Structured Pruning Adapters.” arXiv, Feb. 02, 2023. doi: 10.48550/arXiv.2211.10155.
[25].....M. Zhang *et al.*, “LoRAPrune: Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning.” arXiv, Oct. 03, 2023. doi: 10.48550/arXiv.2305.18403.
[26].....G. Zeng, P. Zhang, and W. Lu, “One Network, Many Masks: Towards More Parameter-Efficient Transfer Learning.” arXiv, Jun. 11, 2023. doi: 10.48550/arXiv.2305.17682.
[27].....S. Jie, H. Wang, and Z.-H. Deng, “Revisiting the Parameter Efficiency of Adapters from the Perspective of Precision Redundancy,” presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 17217–17226. Accessed: May 27, 2024. [Online]. Available:

https://openaccess.thecvf.com/content/ICCV2023/html/Jie_Revisiting_the_Parameter_Efficiency_of_Adapters_from_the_Perspective_of_ICCV_2023_paper.html
[28]...T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “QLoRA: Efficient Finetuning of Quantized LLMs,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 10088–10115, Dec. 2023.
[29] Y. Li *et al.*, “LoftQ: LoRA-Fine-Tuning-Aware Quantization for Large Language Models.” arXiv, Nov. 28, 2023. doi: 10.48550/arXiv.2310.08659.
[30].....H. Guo, P. Greengard, E. P. Xing, and Y. Kim, “LQ-LoRA: Low-rank Plus Quantized Matrix Decomposition for Efficient Language Model Finetuning.” arXiv, Jan. 17, 2024. doi: 10.48550/arXiv.2311.12023.
[31].…Y. Xu *et al.*, “QA-LoRA: Quantization-Aware Low-Rank Adaptation of Large Language Models.” arXiv, Oct. 09, 2023. doi: 10.48550/arXiv.2309.14717.
[32].….J. Liu *et al.*, “BitDelta: Your Fine-Tune May Only Be Worth One Bit.” arXiv, Feb. 27, 2024. doi: 10.48550/arXiv.2402.10193.
[33]J. O. Zhang, A. Sax, A. Zamir, L. Guibas, and J. Malik, “Side-Tuning: A Baseline for Network Adaptation via Additive Side Networks,” in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 698–714. doi: 10.1007/978-3-030-58580-8_41.
[34] Y.-L. Sung, J. Cho, and M. Bansal, “LST: Ladder Side-Tuning for Parameter and Memory Efficient Transfer Learning,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 12991–13005, Dec. 2022.
[35].....Z. Jiang *et al.*, “Res-Tuning: A Flexible and Efficient Tuning Paradigm via Unbinding Tuner from Backbone,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 42689–42716, Dec. 2023.
[36].....B. Liao, S. Tan, and C. Monz, “Make Pre-trained Model Reversible: From Parameter to Memory Efficient Fine-Tuning.” arXiv, Oct. 19, 2023. doi: 10.48550/arXiv.2306.00477.
[37]..A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse, “The Reversible Residual Network: Backpropagation Without Storing Activations,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: May 27, 2024. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f9be311e65d81a9ad8150a60844bb94c-Abstract.html
[38].....L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li, “LoRA-FA: Memory-efficient Low-rank Adaptation for Large Language Models Fine-tuning.” arXiv, Aug. 07, 2023. doi: 10.48550/arXiv.2308.03303.
[39]…J. Phang, Y. Mao, P. He, and W. Chen, “HyperTuning: Toward Adapting Large Language Models without Back-propagation,” in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 27854–27875. Accessed: May 27, 2024. [Online]. Available: https://proceedings.mlr.press/v202/phang23a.html
[40]…F. Jin, J. Zhang, and C. Zong, “Parameter-efficient Tuning for Large Language Model without Calculating Its Gradients,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 321–330. doi: 10.18653/v1/2023.emnlp-main.22.

[41].....S. Malladi *et al.*, “Fine-Tuning Language Models with Just Forward Passes,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 53038–53075, Dec. 2023.
[42]…G. Pu, A. Jain, J. Yin, and R. Kaplan, “Empirical Analysis of the Strengths and Weaknesses of PEFT Techniques for LLMs.” arXiv, Apr. 28, 2023. doi: 10.48550/arXiv.2304.14999.
[43].....T. Feng and S. Narayanan, “PEFT-SER: On the Use of Parameter Efficient Transfer Learning Approaches For Speech Emotion Recognition Using Pre-trained Speech Models,” in *2023 11th International Conference on Affective Computing and Intelligent Interaction (ACII)*, Sep. 2023, pp. 1–8. doi: 10.1109/ACII59096.2023.10388152.
[44]....L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, “Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment.” arXiv, Dec. 19, 2023. doi: 10.48550/arXiv.2312.12148.
[45].....F. Jin, J. Lu, and J. Zhang, “Unified Prompt Learning Makes Pre-Trained Language Models Better Few-Shot Learners,” in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2023, pp. 1–5. doi: 10.1109/ICASSP49357.2023.10095738.
[46]....“On the Effectiveness of Parameter-Efficient Fine-Tuning | Proceedings of the AAAI Conference on Artificial Intelligence.” Accessed: May 27, 2024. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26505
[47].....“Evaluating Parameter-Efficient Transfer Learning Approaches on SURE Benchmark for Speech Understanding | IEEE Conference Publication | IEEE Xplore.” Accessed: May 27, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10095656?casa_token=AsbGRX5MYF0AAAAA:Ad9jqW2AX6z6e0S71Ri-vMOGh28CebmdoOJ9PQlWDeOa3hA-zVuLZbp8rCWd16i5sihMJyusb74
[48].“Exploring Efficient-Tuning Methods in Self-Supervised Speech Models | IEEE Conference Publication | IEEE Xplore.” Accessed: May 27, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10023274?casa_token=aZwG97qCgsQAAAAA:P85HzlBexqBcvMwzlsJgN-E2A29DNKldqHFoeXwhj50t0udkmW4YSdFETLLGiVtJLB0SIJKQXvQ
[49]…T. Wang *et al.*, “What Language Model Architecture and Pretraining Objective Works Best for Zero-Shot Generalization?,” in *Proceedings of the 39th International Conference on Machine Learning*, PMLR, Jun. 2022, pp. 22964–22984. Accessed: May 26, 2024. [Online]. Available: https://proceedings.mlr.press/v162/wang22u.html
[50] W. X. Zhao *et al.*, “A Survey of Large Language Models.” arXiv, Nov. 24, 2023. doi: 10.48550/arXiv.2303.18223.
[51].…P. Xu *et al.*, “Evaluating Parameter Efficient Learning for Generation.” arXiv, Oct. 24, 2022. doi: 10.48550/arXiv.2210.13673.
[52].....N. Ding *et al.*, “Parameter-efficient fine-tuning of large-scale pre-trained language models,” *Nat Mach Intell*, vol. 5, no. 3, pp. 220–235, Mar. 2023, doi: 10.1038/s42256-023-00626-4.
[53].Z. Zhang, Y. Lin, Z. Liu, P. Li, M. Sun, and J. Zhou, “MoEfication: Transformer Feed-forward Layers are Mixtures of Experts.” arXiv, Apr. 05, 2022. doi: 10.48550/arXiv.2110.01786.
[54].....S. Chen, M. Jiang, J. Yang, and Q. Zhao, “Attention in Reasoning: Dataset, Analysis, and Modeling,” *IEEE Transactions on Pattern Analysis and Machine*

Intelligence, vol. 44, no. 11, pp. 7310–7326, Nov. 2022, doi: 10.1109/TPAMI.2021.3114582.
[55]. D. Dai, L. Dong, Y. Hao, Z. Sui, B. Chang, and F. Wei, “Knowledge Neurons in Pretrained Transformers.” arXiv, Mar. 09, 2022. doi: 10.48550/arXiv.2104.08696.

List of Publications

1st Paper

Link: https://ieeexplore.ieee.org/document/10307280

Citation :

R. Negi and R. Katarya, "Emerging Trends in Chatbot Development : A Recent Survey of Design, Development and Deployment," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-6, doi: 10.1109/ICCCNT56998.2023.10307280.

IEEE ELECTRONICS PACKAGING SOCIETY

IEEE PRO COMM Professional Communication Society

IAS IEEE INDUSTRY APPLICATIONS SOCIETY

IEEE

THE FOURTEENTH INTERNATIONAL CONFERENCE ON COMPUTING, COMMUNICATION AND NETWORKING TECHNOLOGIES (ICCCNT)

July 6th - 8th, 2023, IIT - Delhi, Delhi, India.

This is to certify that Prof./Dr./Mr./Ms Rohit Negi

of

Delhi Technological University

has presented a paper on

Emerging Trends in Chatbot Development : A Recent Survey of Design, Development & Deployment.

in the Fourteenth International Conference on computing, Communication and Networking Technologies, held at IIT-Delhi, in association with IEEE Electronics Packaging Society, during July 6th-8th, 2023.

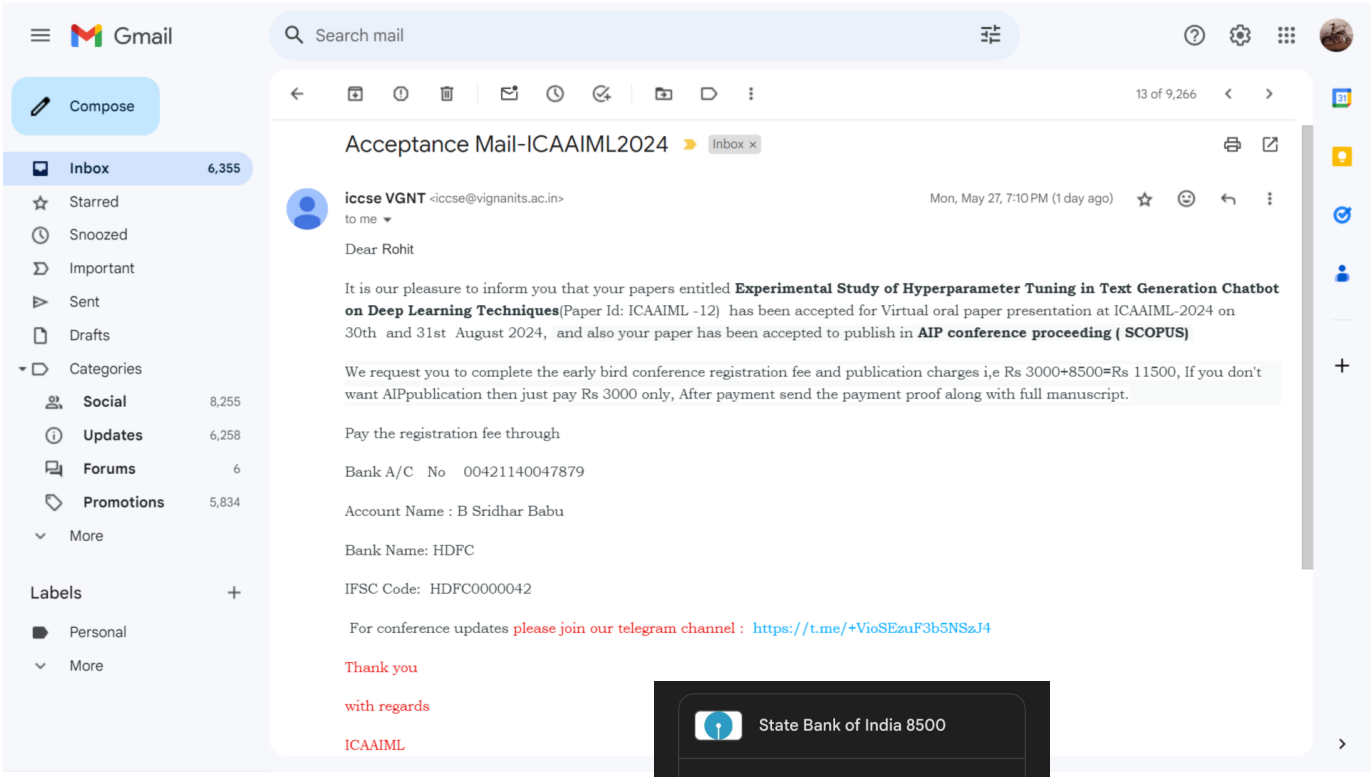
Conference Chair / Co-chair

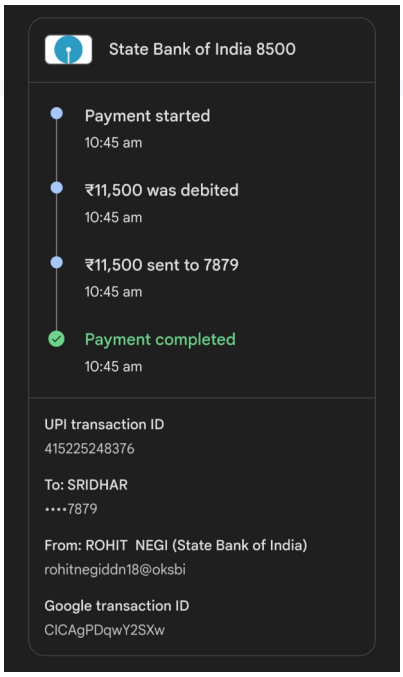
2nd Paper

Title:

"Experimental Study of Hyperparameter Tuning in Text Generation Chatbot on Deep Learning Techniques"

Status: Accepted in ICAAIML2024





DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

Plagiarism Verification

Title of the Thesis : **A Study on PEFT Techniques for Optimizing Content Generation and Textual Comprehension**

Total Pages : **48**

Name of the Scholar : **Rohit Negi**

Supervisor : **Prof. Rahul Katarya**

Department : **Dept. of Computer Science & Engineering**

This is to report that the above thesis was scanned for similarity detection. Process and outcome are given below:

Software used: **Turnitin** Similarity Index: **3%** Total Word Count: **11969**

Place: Delhi
Date:

Candidate's Signature **Signature of Supervisor**

PAPER NAME

Thesis_Rohit_New.pdf

WORD COUNT

11969 Words

CHARACTER COUNT

69610 Characters

PAGE COUNT

48 Pages

FILE SIZE

2.6MB

SUBMISSION DATE

May 29, 2024 1:49 AM GMT+5:30

REPORT DATE

May 29, 2024 1:50 AM GMT+5:30

3% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 2% Internet database
- 1% Publications database
- Crossref database
- Crossref Posted Content database
- 1% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 10 words)