Parameter-Efficient Fine-Tuning Techniques on Large Language Models

Thesis to be submitted in partial fulfillment of the requirements for the degree

of

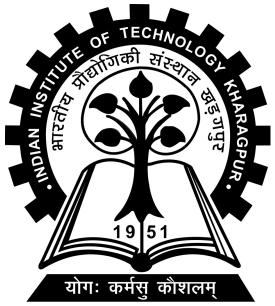
Master of Technology

by

Beerukuri Abhinay
20EC39008

Under the guidance of

Prof. Pawan Goyal and Prof. Arijit De



ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Department of Electronics and Electrical
Communication Engineering
Indian Institute of Technology, Kharagpur
India - 721302

CERTIFICATE

This is to certify that we have examined the thesis entitled **Parameter-Efficient Fine-Tuning Techniques on Large Language Models**, submitted by **Beerukuri Abhinay** (Roll Number: *20EC39008*) an undergraduate student of **Department of Electronics and Electrical Communication Engineering** in partial fulfillment for the award of degree of Master of Technology. We hereby approve it as a study carried out and presented in a manner required for its acceptance in partial fulfillment of the postgraduate degree for which it has been submitted. The thesis has met all the requirements according to the Institute regulations and reached the required submission standard.

Co-Supervisor
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Supervisor
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Date: 29th April 2025
Place: Kharagpur

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

The field of Natural Language Processing (NLP) has witnessed remarkable progress in recent years, driven largely by the advent of large language models (LLMs) such as T5, Llama, and others. These models, pre-trained on massive corpora, have demonstrated state-of-the-art performance across a wide range of text processing tasks, including summarization, classification, translation, and question answering. Their versatility and generalization capabilities have made them foundational tools in both academic research and industry applications.

However, as these models grow in size and complexity, adapting them to specific domains or tasks presents significant challenges. Full fine-tuning of LLMs is computationally expensive, requiring substantial GPU resources and storage, and often leads to inefficiencies when deploying models for multiple tasks. This has motivated the exploration of parameter-efficient fine-tuning (PEFT) techniques, which aim to adapt LLMs to new tasks by updating only a small subset of parameters or by introducing lightweight, task-specific modules. PEFT methods such as LoRA, QLoRA, AdaLoRA, and others have shown promise in reducing resource requirements while maintaining or even improving task performance.

The growing demand for deploying LLMs in specialized domains—where data may be limited and computational resources are constrained—underscores the importance of efficient adaptation strategies. Furthermore, while PEFT techniques have been extensively studied for generative tasks like summarization, their effectiveness for discriminative tasks such as classification remains underexplored. This gap in the literature forms the core motivation for the present research.

1.2 Problem Statement

Despite the success of PEFT methods in generative NLP tasks, there is a lack of comprehensive studies evaluating their effectiveness for classification tasks using LLMs. The primary problem addressed in this thesis is twofold:

- To systematically evaluate and compare various PEFT techniques for fine-tuning LLMs on summarization tasks.
- To investigate the applicability and performance of these PEFT methods for classification tasks, an area with limited existing research and practical guidance.

This research seeks to answer whether PEFT techniques can enable efficient and effective adaptation of LLMs for both summarization and classification, and how their performance compares to zero-shot and fully fine-tuned models.

1.3 Research Aims and Objectives

The aims of this research are as follows:

- To implement and benchmark multiple PEFT techniques (including LoRA, QLoRA, AdaLoRA, LoHa, LoKr, IA^3, and prefix tuning) on both encoder-decoder (T5) and decoder-only (Llama) architectures.
- To evaluate the performance of these methods on summarization tasks using standard datasets (BillSum) and on classification tasks using emotion classification datasets (dair-ai/emotion).
- To compare the efficiency (in terms of trainable parameters and GPU utilization) and effectiveness (in terms of standard metrics such as ROUGE for summarization and accuracy/F1 for classification) of PEFT methods relative to baseline and fully fine-tuned models.
- To analyze the trade-offs between parameter efficiency and task performance, and to identify best practices for selecting PEFT strategies for different NLP tasks.

1.4 Scope and Limitations

Scope

- The study focuses on two primary NLP tasks: text summarization and text classification.
- Experiments are conducted using publicly available datasets, specifically BillSum for summarization and GoEmotions/dair-ai/emotion for classification.
- The research evaluates a range of PEFT techniques on two representative LLM architectures: T5 (encoder-decoder) and Llama (decoder-only).

Limitations

- The study is limited to English-language datasets and does not address multilingual or low-resource language scenarios.
- Only models up to 3B parameters are considered due to hardware constraints.
- The analysis does not include theoretical modifications to model architectures beyond the application of PEFT modules.
- The evaluation is restricted to standard metrics and does not cover aspects such as interpretability, fairness, or ethical considerations in depth.

1.5 Significance of the Study

This research addresses a critical need in the NLP community for practical, scalable, and resource-efficient methods to adapt LLMs to specialized tasks. By providing a systematic evaluation of PEFT techniques for both summarization and classification, the study offers several key contributions:

- It demonstrates that PEFT methods can achieve performance comparable to full fine-tuning while drastically reducing the number of trainable parameters—sometimes to less than 0.1% of the total model size—making LLM adaptation feasible on modest hardware.
- The findings reveal that certain PEFT methods (e.g., AdaLoRA, LoRA, IA^3) are highly effective for both generative and discriminative tasks, while others may be more task-specific.
- The research provides actionable insights for practitioners seeking to deploy LLMs efficiently in real-world applications, enabling dynamic switching between tasks such as summarization and classification without the need for multiple large models.
- By filling a gap in the literature regarding PEFT for classification tasks, the study lays the groundwork for further research into multi-task and modular LLM deployment strategies.

Chapter 2

Literature Review

2.1 Transformer Overview

The Transformer, introduced by Vaswani et al. (2017), is a sequence-to-sequence model that relies solely on attention mechanisms, removing recurrence and convolution. It follows an encoder–decoder architecture: the encoder maps an input sequence $x_1, \ldots, x_n$ to continuous representations $z_1, \ldots, z_n$, and the decoder generates an output sequence $y_1, \ldots, y_m$ one token at a time, attending to encoder outputs and previously generated tokens.

Each encoder layer (typically $N = 6$) contains two sublayers: (1) multi-head self-attention, allowing each token to attend to all others, and (2) a position-wise feed-forward network (FFN). If $x \in \mathbb{R}^{d_{\text{model}}}$, then:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2,$$

with $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$. The base model uses $d_{\text{model}} = 512$, $d_{\text{ff}} = 2048$. Each sublayer is wrapped with a residual connection followed by layer normalization:

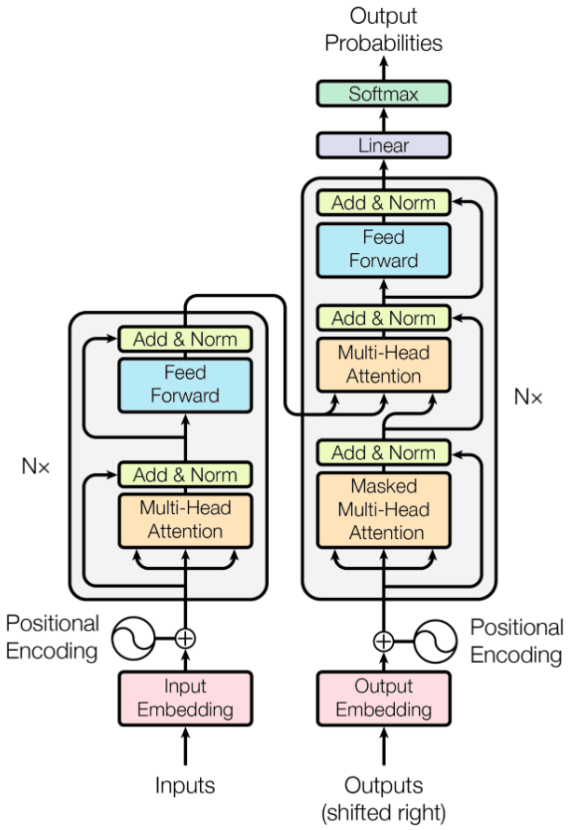


Figure 2.1: Transformer Architecture

$$\text{LayerNorm}(x + \text{Sublayer}(x)).$$

The decoder also has N layers, each with three sublayers: (1) masked self-attention (causal masking prevents access to future tokens), (2) encoder–decoder attention (keys/values from the

encoder, queries from the decoder), and (3) a feed-forward sublayer. All are wrapped with residual connections and normalization. The final decoder outputs pass through a linear+softmax layer to predict tokens.

The core operation is scaled dot-product attention. Given queries $Q \in \mathbb{R}^{m \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$, and values $V \in \mathbb{R}^{n \times d_v}$:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

Masking sets invalid positions to $-\infty$ before softmax. The $\sqrt{d_k}$ scaling prevents large dot products from collapsing gradients.

Instead of a single attention head, the Transformer uses multi-head attention to capture diverse features. Inputs $Q, K, V \in \mathbb{R}^{d_{\text{model}}}$ are projected into h heads ($d_k = d_v = d_{\text{model}}/h$):

$$Q_i = QW_i^Q, \quad K_i = KW_i^K, \quad V_i = VW_i^V,$$

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i), \quad \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O.$$

To encode position, fixed sinusoidal positional encodings are added to input embeddings. For position *pos* and dimension i:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right).$$

These enable relative position reasoning and generalize to longer sequences. Learned position embeddings show similar performance.

2.2 Transformer Training Paradigms

Transformer architectures have become the foundation of modern Natural Language Processing (NLP) systems. Broadly, transformer-based models can be classified into three categories based on their architecture and intended use: encoder-only, decoder-only, and encoder–decoder models. Each follows distinct pretraining and fine-tuning paradigms suited to specific tasks.

2.2.1 Encoder-Only Models

Encoder-only models, such as BERT (Bidirectional Encoder Representations from Transformers) , are primarily designed for tasks that require understanding and encoding the input sequence, such as text classification, named entity recognition, and question answering.

BERT is pretrained using *Masked Language Modeling* (MLM). In MLM, a percentage of input tokens are randomly masked, and the model learns to predict the original tokens. Formally, given an input sequence $X = (x_1, x_2, \ldots, x_n)$, a subset $M \subset \{1, \ldots, n\}$ of token positions is selected for masking. The objective is to maximize the likelihood:

$$\mathcal{L}_{\text{MLM}} = \mathbb{E}_X \left[\sum_{i \in M} \log P(x_i | X_{\setminus M}) \right] \tag{2.1}$$

where $X_{\setminus M}$ denotes the sequence with masked tokens replaced by a special `[MASK]` token.

After pretraining, BERT is fine-tuned by appending task-specific heads (e.g., a classification head) and training on labeled datasets with supervised objectives, typically using cross-entropy loss.

2.2.2 Decoder-Only Models

Decoder-only models, such as LLaMA (Large Language Model Meta AI) , are optimized for autoregressive generation tasks including language modeling, open-ended text generation, and dialogue.

These models are pretrained via *Causal Language Modeling* (CLM), where the model predicts the next token conditioned only on past tokens. Given a sequence $X = (x_1, x_2, \ldots, x_n)$, the loss function is:

$$\mathcal{L}_{\text{CLM}} = \mathbb{E}_X \left[\sum_{i=1}^{n} \log P(x_i | x_1, \ldots, x_{i-1}) \right] \tag{2.2}$$

The unidirectional nature of causal attention ensures that the model cannot "see" future tokens during prediction.

Fine-tuning typically involves supervised fine-tuning (SFT) on domain-specific corpora or reinforcement learning with human feedback (RLHF) to align outputs with human preferences. For sequence classification, additional pooling strategies or prompt engineering may be utilized.

2.2.3 Encoder–Decoder Models

Encoder–decoder models, such as T5 (Text-to-Text Transfer Transformer) , are designed for sequence-to-sequence tasks, including summarization, translation, and question answering.

T5 is pretrained using a variant of denoising autoencoding, specifically *Span Corruption*. In this setup, contiguous spans of text are replaced by a single mask token, and the model is tasked with reconstructing the missing spans.

Given an input X and a corrupted version $\tilde{X}$ where spans are replaced with sentinel tokens, the objective becomes:

$$\mathcal{L}_{\text{Span-Corruption}} = \mathbb{E}_{(X, \tilde{X})} \left[\log P(X_{\text{masked}} | \tilde{X}) \right] \tag{2.3}$$

where X_{masked} denotes the concatenated target spans corresponding to masked portions in $\tilde{X}$.

Fine-tuning T5 is straightforward: all tasks are framed in a text-to-text format. Given an input prompt, the model is trained to generate the corresponding target output, using teacher forcing and minimizing the negative log-likelihood (NLL) loss.

2.3 Language Models

2.3.1 Traditional Probabilistic Language Models

Statistical language models define a probability distribution over sequences of tokens by factorizing the joint probability via the chain rule of probability and approximating long contexts with an n-gram Markov assumption. In particular, the probability of a sequence w_1^n can be written exactly as

$$P(w_1^n) = \prod_{i=1}^{n} P\big(w_i \mid w_1^{i-1}\big), \tag{2.4}$$

which is intractable for large n and motivating the n-gram simplification

$$P\big(w_i \mid w_1^{i-1}\big) \approx P\big(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}\big). \tag{2.5}$$

Unigram, bigram, and trigram models correspond to $n = 1, 2, 3$, and a range of smoothing techniques—such as add-one smoothing, Good–Turing discounting, Katz back-off, and Kneser–Ney—are applied to assign nonzero probabilities to unseen n-grams.

2.3.2 Neural Language Models

The neural probabilistic language model of Bengio et al. maps discrete tokens to continuous embeddings and predicts the next token via a feed-forward network:

$$\mathbf{h} = \tanh\big(H[\mathbf{x}_{i-(n-1)}; \ldots; \mathbf{x}_{i-1}]\big), \quad \mathbf{y} = W\mathbf{h} + \mathbf{b},$$

followed by

$$P(w_i \mid \text{context}) = \frac{\exp(y_{w_i})}{\sum_j \exp(y_j)}.$$

This approach alleviates data sparsity by generalizing across similar contexts. Mikolov et al. extended these ideas to recurrent neural networks (RNNs), updating hidden states as

$$h_t = \phi\big(W_{xh}x_t + W_{hh}h_{t-1}\big),$$

thereby conditioning on arbitrarily long histories and substantially reducing perplexity on speech and text benchmarks.

2.3.3 Transformer and Self-Attention

Vaswani et al. introduced the Transformer, which replaces recurrence with parallel multi-head self-attention, where each attention head computes

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)\, V. \tag{2.6}$$

This design achieves superior parallelism and models long-range dependencies more effectively than RNNs.

2.3.4 Masked Language Modeling

Devlin et al.proposed BERT, which pre-trains deep bidirectional representations by randomly masking a subset of tokens M and optimizing

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in M} \log P\big(w_i \mid w_{[1:n]\setminus\{i\}}\big).$$

This masked language modeling objective yields contextual embeddings that, when fine-tuned, achieve state-of-the-art performance across NLP benchmarks.

2.3.5 Sequence-to-Sequence and Few-Shot Learning

Sutskever et al.formalized sequence-to-sequence learning with encoder–decoder LSTM networks, modeling

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{T_y} P\big(y_t \mid y_{<t}, \mathbf{c}\big),$$

where $\mathbf{c}$ is the encoder's context vector. Lewis et al. (BART) extended this by pre-training a denoising autoencoder to reconstruct corrupted sequences, achieving superior summarization quality. Brown et al. demonstrated that scaling autoregressive Transformers to 175 billion parameters enables few-shot learning directly in the prompt without parameter updates.

2.4 Parameter-Efficient Fine-Tuning (PEFT) for Large Language Models

Parameter-Efficient Fine-Tuning (PEFT) represents a significant advancement in the field of neural network adaptation, particularly for Large Language Models (LLMs). This technique fundamentally alters the traditional paradigm of model fine-tuning by selectively updating only a small subset of model parameters while keeping the majority of the pre-trained weights frozen. As foundation models continue to exponentially scale in size, efficient methods of adaptation have become increasingly critical for both research and practical applications.

PEFT operates on the premise that not all parameters in a neural network contribute equally to task-specific adaptations. By identifying and modifying only the most critical parameters, PEFT achieves comparable performance to full fine-tuning while drastically reducing computational requirements. This approach stands in stark contrast to traditional fine-tuning methods, which update all model parameters indiscriminately, necessitating substantial computational resources and potentially leading to catastrophic forgetting of pre-trained knowledge.

The fundamental innovation of PEFT lies in its ability to preserve the general knowledge encoded in pre-trained models while efficiently adapting them to downstream tasks. This is typically achieved through strategically placed adapter modules, parameter-selective updates, or low-rank

transformations of weight matrices. Such methodologies represent a paradigm shift in how we conceptualize model adaptation, moving away from resource-intensive approaches toward more sustainable and accessible alternatives.

2.4.1 Computational and Practical Benefits

The advantages of PEFT extend beyond mere theoretical interest, offering substantial practical benefits. From a computational perspective, PEFT dramatically reduces memory usage-often to a third of the original requirements for LLM fine-tuning. This reduction is particularly significant for models exceeding billions of parameters, where traditional fine-tuning would be prohibitively expensive or technically infeasible on consumer hardware.

Storage efficiency represents another crucial advantage, with PEFT-tuned models typically requiring only a few megabytes of additional storage compared to the several gigabytes needed for thoroughly fine-tuned models. This efficiency facilitates deployment across multiple tasks without redundant storage of the base model weights. For organizations operating with limited computational resources, PEFT enables the adaptation of state-of-the-art models that would otherwise remain inaccessible.

Furthermore, PEFT approaches have demonstrated remarkable effectiveness in mitigating catastrophic forgetting, a phenomenon where models lose previously acquired knowledge during adaptation to new tasks. By isolating task-specific modifications, PEFT preserves the generalized knowledge embedded in pre-trained weights while simultaneously enabling specialization

2.4.2 Technical Approaches to PEFT

The landscape of PEFT encompasses various methodologies, each with distinct approaches to parameter efficiency. Prominent examples include Low-Rank Adaptation (LoRA), which approximates weight updates using low-rank decompositions; Infused Adapter by Inhibiting and Amplifying Inner Activations (IA^3), which introduces learned vectors that rescale activations within the model; and Adapter modules, which insert trainable bottleneck layers into the original architecture.

While these techniques differ in implementation details, they share the fundamental principle of minimizing the number of trainable parameters while maximizing task-specific adaptations. The diversity of PEFT methodologies reflects the ongoing research efforts to optimize the balance between computational efficiency and model performance across various domains and tasks.

2.4.3 Challenges and Limitations

Despite its advantages, PEFT is not without challenges. A performance gap can exist between PEFT and full fine-tuning, particularly for highly specialized tasks that require substantial deviation from the pre-trained model's capabilities. Additionally, PEFT introduces new hyperparameters that

require careful tuning for optimal results, potentially offsetting some of the computational savings gained during training.

The integration of multiple PEFT modules-for instance, when adapting a model to several tasks simultaneously-presents another significant challenge. While individual adapters are lightweight, the concurrent utilization of multiple adaptation sets can lead to interference, degraded performance, or increased complexity in model management.

Contrary to some prevailing assumptions, empirical evidence suggests that PEFT techniques may converge slower than full tuning in low-data scenarios, necessitating careful consideration of the data regime when selecting adaptation strategies. Furthermore, while PEFT has demonstrated empirical success, the theoretical understanding of why certain PEFT methods work so effectively remains incomplete, indicating a gap between practice and theory in this rapidly evolving field

2.5 A Comprehensive Review of Modern Techniques

This review examines eight parameter-efficient fine-tuning (PEFT) methods that enable adaptation of large models with minimal computational overhead.

2.5.1 LoRA: Low-Rank Adaptation

LoRA, introduced by Hu et al. (2021), represents a landmark approach to parameter-efficient fine-tuning of large language models. The key innovation lies in freezing pre-trained model weights while injecting trainable rank decomposition matrices into each layer of the Transformer architecture.

In traditional fine-tuning, all parameters of a pre-trained model are updated, which becomes prohibitively expensive for models like GPT-3 (175B parameters). LoRA addresses this challenge by constraining the update to the weight matrices to have a low intrinsic rank.

Mathematically, for a pre-trained weight matrix $W \in \mathbb{R}^{d\times k}$, LoRA parameterizes its update by:

$$W' = W + \Delta W = W + BA$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ with rank $r \ll \min(d, k)$. During training, W is frozen and does not receive gradient updates, while A and B are trained. This reduces the number of trainable parameters from $d \times k$ to $r \times (d + k)$.

In practice, LoRA is often applied with a scaling factor α:

$$W' = W + \frac{\alpha}{r}BA$$

where α is a hyperparameter that controls the magnitude of the update. This scaling helps maintain consistent behavior regardless of the rank r.

Hu et al. demonstrated that LoRA can reduce the number of trainable parameters by 10,000 times compared to full fine-tuning of GPT-3, while achieving comparable or better performance. A key advantage of LoRA is that it does not introduce additional inference latency, as the low-rank matrices can be merged with the original weights after training.

2.5.2 LoHa: Low-Rank Hadamard Product

LoHa, introduced as part of the FedPara framework by Nam et al. (2021), utilizes element-wise multiplication (Hadamard product) instead of traditional matrix multiplication to approximate weight updates. This approach offers greater expressive capacity than conventional low-rank methods while maintaining parameter efficiency.

For a pre-trained weight matrix $W \in \mathbb{R}^{d\times k}$, LoHa parameterizes the update as:

$$W' = W + (B_1A_1) \odot (B_2A_2)$$

where $B_1 \in \mathbb{R}^{d\times r_1}$, $A_1 \in \mathbb{R}^{r_1\times k}$, $B_2 \in \mathbb{R}^{d\times r_2}$, and $A_2 \in \mathbb{R}^{r_2\times k}$. The symbol $\odot$ denotes the Hadamard (element-wise) product.

This formulation allows LoHa to escape the strict low-rank constraints that limit conventional methods like LoRA. The Hadamard product of two low-rank matrices can represent a higher-rank structure, providing more expressive power for the same parameter count.

The authors demonstrated that this approach enables comparable performance to full-parameter methods while requiring 3 to 10 times fewer parameters. This property makes LoHa particularly suitable for federated learning scenarios where communication efficiency is crucial.

2.5.3 LoKr: Low-Rank Kronecker Product

Low-Rank Kronecker Product (LoKr) represents another innovative approach to parameter-efficient fine-tuning, utilizing the Kronecker product to combine low-rank matrices. This method offers an alternative matrix decomposition strategy compared to LoRA and LoHa.

The Kronecker product, denoted by $\otimes$, is a mathematical operation that creates a larger matrix from two smaller ones. For two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$, their Kronecker product $A \otimes B$ is a matrix of size $mp \times nq$.

In LoKr, the weight update is parameterized using the Kronecker product:

$$W = W_0 + \Delta W = W_0 + (A \otimes B)C$$

where $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{p\times q}$, and $C \in \mathbb{R}^{(mp\cdot nq)\times k}$ is an optional third matrix that provides better control during fine-tuning. The dimensions are chosen such that $mp = d$ and $nq = r$, where d is the row dimension of W_0 and r is the target rank.

The configuration of LoKr allows for several customization options:

- `decompose_both`: parameter to perform rank decomposition of the left Kronecker product matrix.
- `decompose_factor`: controls the Kronecker product decomposition factor.
- `rank_dropout_scale`: determines whether to scale the rank dropout during training.

Unlike traditional low-rank approximations, the Kronecker product structure provides LoKr with a unique capacity to capture complex patterns in the weight updates while maintaining a compact representation. The addition of the optional third matrix C further enhances flexibility, allowing for more precise control of the adaptation process.

LoKr's parameter-efficient approach is particularly valuable in scenarios where multiple fine-tuned models need to be deployed with limited storage and computational resources, such as edge devices or distributed systems.

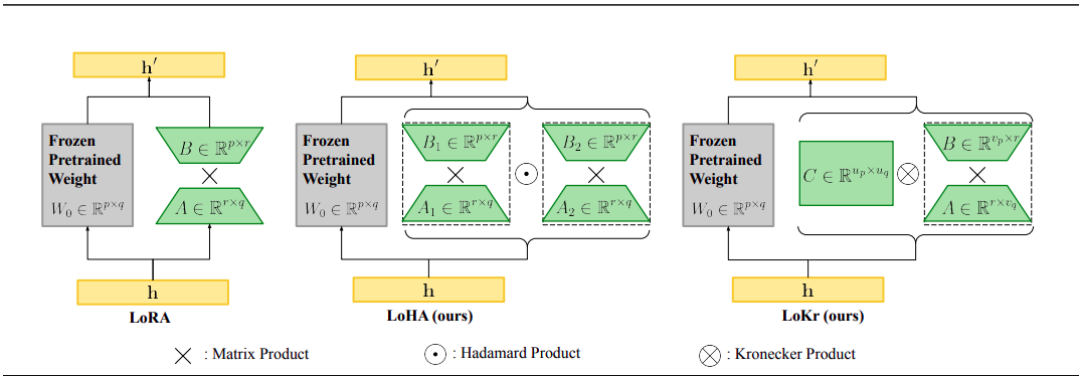


Figure 2.2: Comparison of different lora methods

2.5.4 QLoRA: Quantized Low-Rank Adaptation

QLoRA, proposed by Dettmers et al. (2023), extends LoRA by incorporating quantization techniques to further reduce memory requirements during fine-tuning. QLoRA enables fine-tuning of very large language models on consumer-grade hardware, such as training a 65B parameter model on a single 48GB GPU.

The key innovations of QLoRA include:

- **4-bit NormalFloat (NF4)**: A new data type specifically designed for normally distributed weights that is information-theoretically optimal for such distributions.
- **Double quantization**: A technique that quantizes the quantization constants themselves, further reducing memory footprint.
- **Paged optimizers**: Memory management techniques to handle memory spikes during training.

Mathematically, QLoRA follows the same low-rank adaptation principle as LoRA, but operates on quantized weights. For a pre-trained weight matrix W, QLoRA first quantizes it to 4 bits:

$$W_q = Q_4(W)$$

where Q_4 is the 4-bit quantization function. During the forward pass, W_q is dequantized back to 16-bit (half precision) for computation:

$$W_{16} = D(W_q)$$

The LoRA update is then applied in 16-bit precision:

$$W' = W_{16} + \frac{\alpha}{r} BA$$

2.5.5 AdaLoRA: Adaptive Low-Rank Adaptation

AdaLoRA, proposed by Zhang et al. (2023), enhances LoRA by dynamically allocating the parameter budget among weight matrices according to their importance. Unlike LoRA, which assigns fixed ranks to all weight matrices, AdaLoRA adaptively adjusts the ranks based on importance scores, allowing more critical matrices to have higher ranks.

The key insight of AdaLoRA is the parameterization of incremental updates in the form of singular value decomposition (SVD). For a pre-trained weight matrix $W \in \mathbb{R}^{d \times k}$, AdaLoRA parameterizes its update as:

$$W' = W + \Delta W = W + P\Lambda Q$$

where $P \in \mathbb{R}^{d \times r}$, $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_r) \in \mathbb{R}^{r \times r}$, and $Q \in \mathbb{R}^{r \times k}$. This triplet (P, Λ, Q) mimics the singular value decomposition of ΔW, with Λ containing the singular values.

The importance score for each singular value λ_i is computed as:

$$\text{score}(i) = |\lambda_i| \cdot \left| \frac{\partial \mathcal{L}}{\partial \lambda_i} \right|$$

where $\mathcal{L}$ is the loss function. AdaLoRA follows a pruning schedule where every ΔT steps (e.g., $\Delta T = 100$), it evaluates these importance scores and prunes the least important singular values, reallocating the parameter budget to more critical components.

Zhang et al. demonstrated that AdaLoRA achieves superior performance compared to the original LoRA, especially in low-budget settings. By allocating parameters based on importance, AdaLoRA enables more efficient use of the parameter budget.

2.5.6 IA^3: Infused Adapter by Inhibiting and Amplifying Inner Activations

The Infused Adapter by Inhibiting and Amplifying Inner Activations (IA^3), introduced by Liu et al. (2022), takes a fundamentally different approach to parameter-efficient fine-tuning compared to weight-matrix-based methods like LoRA. Instead of modifying weight matrices, IA^3 rescales the model's intermediate activations using learned task-specific vectors.

The core mechanism of IA^3 involves element-wise multiplication of the model's activations with learned vectors. For a sequence of activations $x \in \mathbb{R}^{T \times d}$ (where T is the sequence length and d is the hidden dimension), IA^3 applies a learned vector $l \in \mathbb{R}^d$ as follows:

$$l \odot x$$

where $\odot$ denotes element-wise multiplication with broadcasting such that the (i, j)-th entry of $l \odot x$ is $l_j x_{i,j}$.

In Transformer models, IA^3 introduces three types of learned vectors:

- $l_k \in \mathbb{R}^{d_k}$ for keys in attention mechanisms
- $l_v \in \mathbb{R}^{d_v}$ for values in attention mechanisms
- $l_{ff} \in \mathbb{R}^{d_{ff}}$ for intermediate activations in feed-forward networks

These vectors are incorporated into the attention mechanisms as:

$$\text{softmax}\left(\frac{Q(l_k \odot K^T)}{\sqrt{d_k}}\right)(l_v \odot V)$$

and in the feed-forward networks as:

$$(l_{ff} \odot \gamma(W_1 x))W_2$$

where γ is the feed-forward network nonlinearity, and W_1, W_2 are the original feed-forward weights.

A separate set of vectors is introduced for each Transformer layer, adding only $L(d_k + d_v + d_{ff})$ parameters for an L-layer Transformer encoder and $L(2d_k + 2d_v + d_{ff})$ for a decoder.

The vectors are initialized with ones, ensuring the model's function remains unchanged initially. This approach enables mixed-task batches since each sequence can be separately and cheaply multiplied by its associated task vector.

IA^3's key advantage is its extremely parameter-efficient nature, introducing far fewer parameters than other PEFT methods while maintaining competitive performance.

2.5.7 Prefix Tuning: Virtual Prompts for Natural Language Generation

Prefix Tuning, introduced by Li and Liang (2021), represents a pioneering approach to parameter-efficient fine-tuning specifically designed for natural language generation tasks. It keeps language model parameters frozen while optimizing a small continuous task-specific vector called the *prefix*.

The key insight of Prefix Tuning is to prepend a sequence of virtual tokens (the prefix) to the input, allowing subsequent tokens to attend to this prefix as if it were part of the prompt. Unlike discrete prompting, the prefix consists of continuous, trainable vectors that can be optimized through backpropagation.

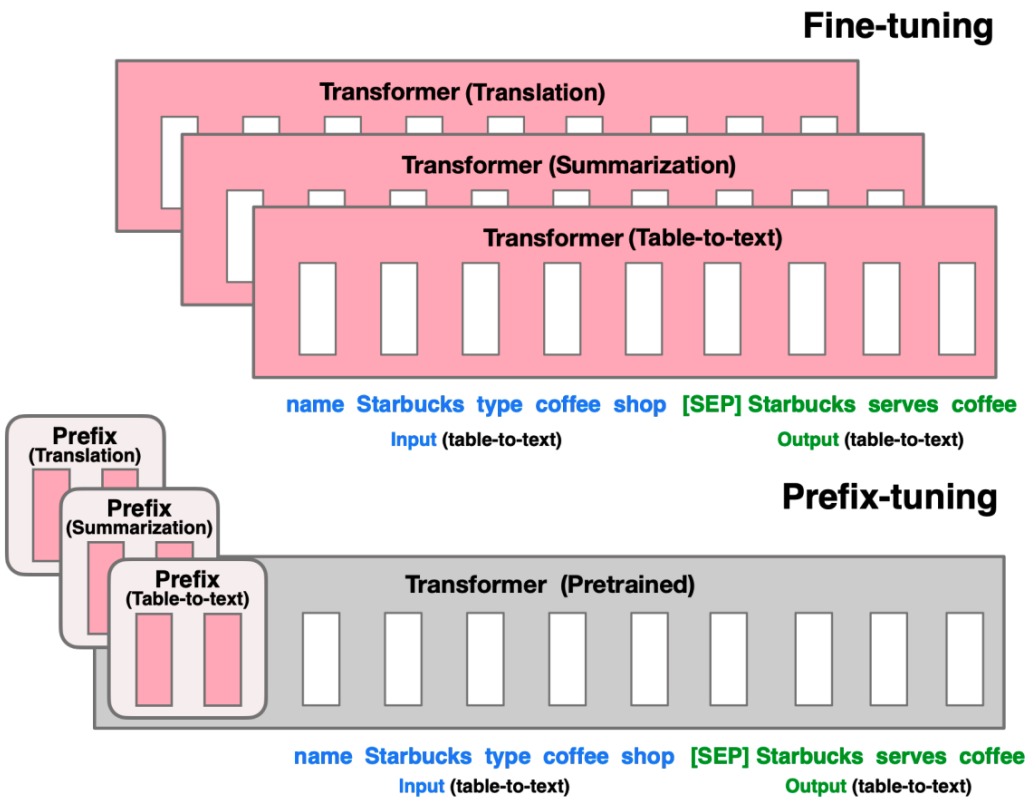


Figure 2.3: Fine-tuning vs. Prefix Tuning

For an autoregressive language model, Prefix Tuning prepends a prefix to form:

$$z = [\text{PREFIX}; x; y]$$

For encoder-decoder models, prefixes are added to both components:

$$z = [\text{PREFIX}; x; \text{PREFIX}'; y]$$

where x represents the input, y the output, and PREFIX, PREFIX$'$ the trainable continuous vectors.

Instead of directly optimizing the prefix embeddings, Prefix Tuning employs a reparameterization trick. It uses a small MLP to generate the prefix, improving optimization stability:

$$P_\theta = \text{MLP}_\theta([\text{PREFIX}_{\text{idx}}])$$

where P_θ contains activations for all Transformer layers, not just the embedding layer. This allows the prefix to influence all layers of the model, providing greater expressivity than simple prompting.

Prefix Tuning is particularly effective in low-data regimes and for examples with topics unseen during training. Experiments on table-to-text generation and summarization tasks demonstrated that by learning only 0.1% of the parameters, Prefix Tuning achieved performance comparable to full fine-tuning in high-resource settings and outperformed it in low-resource scenarios.

Beyond its parameter efficiency, Prefix Tuning offers the key advantage of enabling task-specific adaptations without storing separate copies of the entire model for each task, making it highly practical for deployment scenarios with multiple downstream applications.

2.5.8 P-Tuning v2: Prompt Tuning for Natural Language Understanding

P-Tuning v2, introduced by Liu et al. (2021), represents a significant advancement in prompt-based parameter-efficient fine-tuning, extending the applicability of prompt tuning to natural language understanding (NLU) tasks across a wide range of model scales.

While previous prompt tuning methods had shown promise for large models but underperformed for normal-sized pre-trained models, P-Tuning v2 demonstrated that properly optimized prompt tuning can be universally effective. It implements *Deep Prompt Tuning*, which introduces continuous prompt vectors at each layer of the network rather than just the input layer.

For a Transformer model with L layers, P-Tuning v2 prepends a set of trainable prompt embeddings P_l to the l-th layer's hidden states H_l:

$$H'_l = [P_l; H_l]$$

These prompts are then used in the self-attention mechanism:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where Q, K, and V are derived from H'_l rather than just H_l.

Unlike earlier prompt tuning methods that only inserted prompts at the input level, P-Tuning v2's deep prompt design allows for greater optimization capability, enabling it to match full fine-tuning performance across various NLU tasks while tuning only 0.1%–3% of the model parameters.

The empirical results demonstrate that P-Tuning v2 achieves performance comparable to full fine-tuning across a range of NLU tasks, including natural language inference, question answering, and sequence labeling. This universal effectiveness across different model scales (from hundreds of millions to billions of parameters) and task types represents a significant advancement in parameter-efficient fine-tuning methodology.

P-Tuning v2's simplicity and effectiveness make it a compelling alternative to traditional fine-tuning, especially in scenarios where computational resources are limited or where multiple task-specific models need to be maintained.

Chapter 3

Research Methodology

3.1 Research Design

The research design of this study is fundamentally quantitative and experimental, aimed at systematically evaluating the effectiveness and efficiency of parameter-efficient fine-tuning (PEFT) techniques on large language models (LLMs) for two core NLP tasks: text summarization and text classification. The study adopts a comparative experimental framework, where multiple PEFT methods are benchmarked against each other and against baseline (zero-shot and fully fine-tuned) models. This design allows for controlled, repeatable experiments and robust statistical comparison of results, ensuring the reliability and validity of findings.

At the outset, the research is guided by the hypothesis that PEFT techniques can adapt LLMs for domain-specific tasks with significantly reduced computational resources, without substantial loss in task performance. To test this hypothesis, the study focuses on two representative LLM architectures-T5 (encoder-decoder) and Llama (decoder-only)-and applies a suite of PEFT methods, including LoRA, QLoRA, AdaLoRA, LoHa, LoKr, and IA^3. These methods are selected based on their prominence in recent literature and their suitability for large-scale models.

3.2 Classification Dataset

The `dair-ai/emotion` dataset is a benchmark resource for emotion recognition research, comprising English-language Twitter messages annotated with six basic emotions: anger, fear, joy, love, sadness, and surprise.

Developed to advance affective computing and natural language understanding, this dataset provides standardized inputs for training and evaluating text classification models.

Hosted on the Hugging Face Datasets Hub, the `dair-ai/emotion` dataset supports seamless integration with modern NLP workflows through APIs and libraries. Its accessibility has made it a preferred choice for benchmarking emotion classification models, with over 1,800 publicly available fine-tuned models derived from the dataset.

Description	Numbers	Details
Number of Samples	20,000	Partitioned into training (16,000), validation (2,000), and test (2,000) sets.
Number of Labels	6	The labels are anger, fear, joy, love, sadness, and surprise. and surprise..
Input Sequence Length	7–300 characters	Raw tweet content reflecting informal social media language.
Output Label Length	3-7 characters	(class names)
Primary Annotation	-	Single emotion per text instance.

Table 3.1: Dataset Overview and Technical Specifications of the dair-ai/emotion dataset.

3.3 Summarization Dataset

The `FiscalNote/BillSum Dataset` dataset, created by FiscalNote, focuses on the task of summarizing lengthy legislative texts from the U.S. Congress and the California State Legislature. It provides real-world documents paired with high-quality, human-written summaries, making it a valuable resource for research in long document summarization and legal natural language processing. The dataset captures the complexity and formal style typical of legislative writing, with summaries crafted to be concise, factually accurate, and easy to understand.

The dataset has been widely adopted for benchmarking models designed to handle long input sequences and generate informative, structured summaries. Its diversity across different legislative sources (federal and state) enables more robust evaluation of summarization systems across varying document styles.

- **Input Text Characteristics**
  - Minimum character length: 5,001 characters
  - Maximum character length: 19,998 characters
- **Target Summary Characteristics:**
  - Minimum character length: 52 characters
  - Maximum character length: 4,995 characters
- **Dataset Splits and Sizes:**
  - Training set: 18,949 samples
  - General test set: 3,269 samples

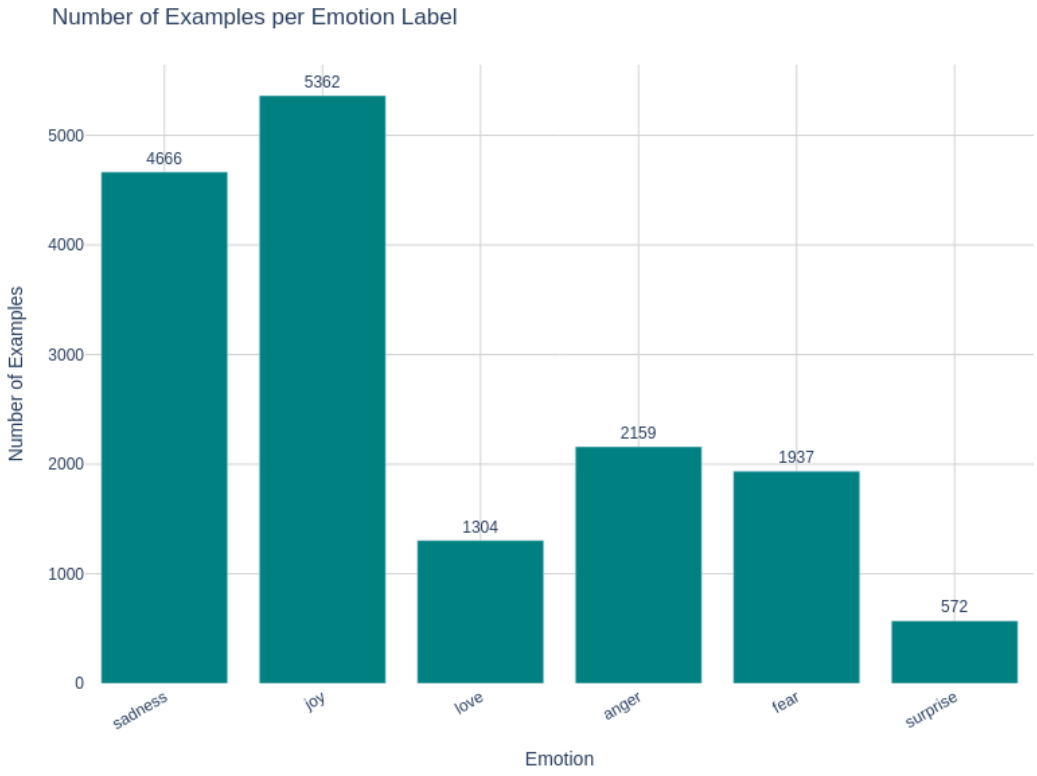


Figure 3.1: Class distribution for the `dair-ai/emotion` dataset.

3.4 Model Comparison

For this study, we employed two distinct models-T5 and LLaMA-whose key differences are outlined below.

Table 3.2: Model Comparison: FLAN-T5-XL vs LLaMA-3.2-3B

Feature	**FLAN-T5-XL**	**LLaMA-3.2-3B**
No. of Parameters	3̃ billion	3̃.2 billion
Architecture	Encoder-decoder	Decoder-only
Context Window	2,048 tokens	128,000 tokens
Multilingual Support	Yes (over 50 languages)	Yes (8 officially supported)
Model Type	Sequence-to-sequence (text-to-text)	Auto-regressive (text generation)
Primary Tasks	Summarization, translation, QA, etc.	Dialogue, summarization, reasoning
Fine-tuned On	1,000+ tasks, many languages	Multilingual, instruction-following
Strengths	Zero/few-shot, broad multilingual tasks	Long context, efficient deployment
Release Date	Publicly available since 2022	Released September 2024

3.5 Implementation Details

The proposed system leverages two transformer-based architectures—`T5` (`flan-t5-xl`) and `LLaMA-3.2-3B` for summarization and text classification tasks. Our implementation integrates multiple parameter-efficient fine-tuning (PEFT) techniques into a unified pipeline, enabling efficient adaptation of large language models (LLMs) under constrained hardware conditions (NVIDIA A100 46GB GPU). The architecture comprises four core components:

- A frozen base model initialized with pretrained weights
- Modular PEFT adapters dynamically injected into specific transformer layers
- Task-specific output heads (classification or generation)
- Memory-optimized training loop with mixed precision and gradient checkpointing

For `T5`, adapters target the query (q) and value (v) projections in the encoder-decoder attention layers, while `LLaMA`'s architecture focuses on the key (`k_proj`) and value (`v_proj`) modules in its causal self-attention mechanism. The implementation utilizes `Hugging Face Transformers` (v4.35.0) and the `PEFT` toolkit (v0.7.0) for model orchestration, with `PyTorch 2.1.0` managing GPU acceleration.

3.5.1 Implementation for Summarization Task

For summarization tasks using `T5` and `LLaMA-3.2-3B-Instruct`, we systematically evaluated multiple PEFT strategies:

T5 Summarization

- **LoRA**: Rank $r = 2$, $\alpha = 4$, applied to query (`q`) and value (`v`) projections.
- **AdaLoRA**: Dynamic rank adaptation ($r_{\text{init}} = 12$, $t_{\text{final}} = 1000$) with stepwise pruning.
- **LoHa / LoKr**: Hadamard/Kronecker decompositions ($r = 2$, $\alpha = 4$) for attention layers.
- **IA3**: Activation scaling for `q` and `v` modules.
- **Prompt/Prefix Tuning**: 200 virtual tokens with hidden dimension $h_{\text{hidden}} = 128$ for input and attention steering.

LLaMA Summarization

- **LoRA / LoKr / LoHA**: $r = 2$, $\alpha = 4$ on `k_proj` and `v_proj` layers.
- **AdaLoRA**: Adaptive allocation ($\Delta T = 10$, $t_{\text{final}} = 1000$).
- **IA3**: Feedforward module scaling (`down_proj`, `up_proj`).
- **Prefix Tuning**: Learnable prefixes for attention mechanisms.

The summarization pipeline processed source documents truncated to 1,400 tokens (80th percentile) and generated summaries of 256 tokens (90th percentile). `T5`'s encoder handled tokenized inputs with LoRA-enhanced attention, while `LLaMA`'s decoder utilized prefix-tuned causal masking.

Memory constraints were addressed via gradient checkpointing, reducing VRAM usage by 33% while maintaining stable training throughput (1.2 samples/sec). Instruction-tuned decoding was employed: `T5` used a `"summarize:"` prefix, whereas `LLaMA` followed an instruction-response template format.

3.6 Implementation for Classification Task

The classification framework incorporated multiple adaptation strategies:

3.6.1 LLaMA Classification

- **LoRA / LoKr**: $r = 8$, $\alpha = 16$ on `k_proj` / `v_proj`.
- **AdaLoRA**: Dynamic ranks ($\delta T = 10$, total steps = 8000).
- **IA3**: Activation scaling across attention and feedforward modules.

3.6.2 T5 Classification

- **LoRA / AdaLoRA**: $r = 8$, progressive rank adjustment ($t_{\text{init}} = 50$).
- **IA3**: Minimal-parameter activation modulation.
- **P-Tuning**: 1000 virtual tokens for task-specific embeddings.

Class imbalance was mitigated through weighted cross-entropy loss with weights $w_c = \frac{N}{C \cdot n_c}$, where N is the total number of samples, C the number of classes, and n_c the count of samples in class c. For classification, `T5` was reconfigured as a text-to-text model, mapping emotion labels (e.g., `"joy"`) to target tokens via tokenizer alignment, while `LLaMA` utilized a sequence classification head appended to its final hidden states. Training leveraged weighted loss with AdamW optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) and cosine learning rate decay.

3.7 Experimental Setup and Optimization

Experiments were conducted on an NVIDIA A100 GPU (46GB VRAM), paired with an Intel Xeon Platinum 8480C CPU (64 cores) and 512GB DDR5 RAM. The software stack included PyTorch 2.1.0 with CUDA 12.1, Hugging Face Transformers v4.35.0, and the PEFT Library v0.7.0. Mixed precision training was enabled using BF16 via `torch.autocast` to accelerate computations and reduce memory usage.

3.7.1 Hyperparameter Configuration

A unified optimization strategy was employed for both summarization and classification. For classification, conservative learning rates of 5×10^{-5} and 4×10^{-4} were used, while summarization tasks leveraged a higher rate of 6×10^{-2} for faster convergence. Adapter ranks were fixed at $r = 2$ with scaling factor $\alpha = 2r$ to ensure gradient stability. Weight decay values were selected from the range $[0.001, 0.01]$, and gradient clipping was applied with a norm threshold of 0.3. Paged AdamW optimizer with BF16 precision reduced memory usage by approximately 40%, and dynamic padding based on percentile-based truncation improved computational efficiency.

3.7.2 Training Configurations

For **summarization**, models were trained with a per-device batch size of 1 using 4-step gradient accumulation. T5 was trained for 10 epochs and LLaMA for 5, with learning rates set to 6×10^{-2} and 5×10^{-3}, respectively.

For **classification**, the batch size remained 2 without accumulation. T5 was trained for 20 epochs and LLaMA for 20, using learning rates of 4×10^{-4} and 5×10^{-5}, respectively. A weighted cross-entropy loss was used with class weights defined as $w_c = \frac{N}{C \cdot n_c}$, where N is the total number of samples, C the number of classes, and n_c the count per class.

3.8 Summarization Metrics

3.8.1 ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a standard evaluation metric for summarization tasks. It measures the overlap between a machine-generated summary and one or more reference summaries, based on n-gram, word sequence, and word pair matches. A commonly used variant is ROUGE-N, which measures n-gram recall. The ROUGE-N Recall is defined as:

$$\text{ROUGE-N Recall} = \frac{\sum_{S \in \{\text{Reference Summaries}\}} \sum_{\text{gram}_n \in S} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{S \in \{\text{Reference Summaries}\}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)} \tag{3.1}$$

where $\text{Count}_{\text{match}}(\text{gram}_n)$ is the number of overlapping n-grams between the candidate and reference summaries.

3.9 Classification Metrics

3.9.1 Accuracy

Accuracy is a fundamental metric for evaluating classification models. It measures the proportion of correctly predicted instances among the total instances. The formula for accuracy A is:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

where TP (True Positives) and TN (True Negatives) are the correctly predicted positive and negative instances, and FP (False Positives) and FN (False Negatives) are the incorrectly predicted instances. Although intuitive, accuracy can be misleading for imbalanced datasets.

3.9.2 F1-Score

The F1-Score is the harmonic mean of precision and recall and is particularly useful for imbalanced classification problems. It balances the trade-off between precision and recall. The F1-Score F_1 is defined as:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

where precision and recall are given by:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.4}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.5}$$

A higher F1-Score indicates a better balance between precision and recall, making it a crucial metric when the cost of false positives and false negatives are both significant.

Chapter 4

Results and Analysis

In this study, we present a thorough evaluation of various Parameter-Efficient Fine-Tuning (PEFT) techniques applied to Llama and T5 language models for text summarization tasks and classification tasks. Our analysis compares performance metrics, resource utilization, and efficiency trade-offs across multiple PEFT methods.

4.1 Performance Analysis of PEFT Methods on Summarization Tasks

Our experimental results demonstrate that PEFT methods significantly enhance the summarization capabilities of both Llama and T5 base models. Table 1 presents the ROUGE metrics (1, 2, and L) across all tested methods.

Table 4.1: ROUGE Score Performance of PEFT Methods on Summarization Tasks

PEFT Method	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
Llama (base)	27.81	12.53	18.06
Qlora Llama	44.31	21.18	26.46
Llama LoRA	47.76	19.58	25.11
Llama LoHA	47.06	22.93	27.42
Llama LoKR	47.92	22.36	26.96
Llama AdaLoRA	49.07	22.55	26.97
T5 (base)	14.64	6.40	11.24
Qlora T5	44.16	20.59	25.59
T5 LoRA	49.71	25.87	30.08
T5 LoHA	50.19	26.21	30.56
T5 AdaLoRA	50.54	26.59	30.67
T5 LoKR	49.73	26.23	30.51
T5 P-Tuning	49.18	25.41	30.23
IA3 Llama	48.37	21.83	26.54
IA3 T5	49.84	26.84	31.30
Llama Prefix Tuning	47.75	21.07	26.12

Performance Improvements over Base Models

For the **Llama** model, PEFT methods provide substantial improvements over the base model's performance. The base Llama achieves ROUGE-1, ROUGE-2, and ROUGE-L scores of 27.81, 12.53, and 18.06 respectively. After applying PEFT techniques, we observe impressive gains, with ROUGE-1 scores increasing by up to **76.44%** (in the case of AdaLoRA). Among Llama implementations, **AdaLoRA** demonstrates the highest ROUGE-1 score (49.07), while **LoHA** excels in ROUGE-2 (22.93) and ROUGE-L (27.42).

For the **T5** model, the performance improvements are even more dramatic. The base T5 model starts with considerably lower scores (14.64, 6.40, and 11.24 for ROUGE-1, ROUGE-2, and ROUGE-L), but with PEFT methods applied, it achieves remarkable enhancements. **T5 with AdaLoRA** reaches the highest overall performance with ROUGE scores of 50.54, 26.59, and 30.67, representing a **245.2%** improvement in ROUGE-1 over the base model. This dramatic improvement suggests that T5's architecture is particularly amenable to parameter-efficient adaptation techniques.

4.1.1 Resource Efficiency Analysis

To provide a comprehensive evaluation, we analyze not only performance metrics but also resource utilization, which is crucial for practical deployments. Table 2 presents the trainable parameters, percentage of total parameters, GPU utilization, and rank for each method.

Table 4.2: Resource Efficiency Analysis of PEFT Methods

PEFT Method	**Trainable Parameters**	**% Trainable Params**	**GPU Utilization (in GB)**	**Rank**
Qlora Llama	917,504	0.0286	6.134	4
Llama LoRA	458,752	0.0143	29.291	2
Llama LoHA	917,504	0.0286	29.422	2
Llama LoKR	96,768	0.0030	29.229	2
Llama AdaLoRA	2,753,184	0.0856	28.982	2
Qlora T5	2,359,296	0.0827	6.542	4
T5 LoRA	1,179,648	0.0414	36.905	2
T5 LoHA	2,359,296	0.0827	37.658	2
T5 AdaLoRA	7,079,616	0.2478	36.815	2
T5 LoKR	184,320	0.0065	37.585	2
T5 P-Tuning	1,362,176	0.0478	40.473	–
IA3 Llama	745,472	0.0232	32.224	–
IA3 T5	294,912	0.0103	39.512	–
Llama Prefix Tuning	11,468,800	0.3557	29.058	–

4.1.2 Parameter Efficiency Analysis

The base **Llama** model contains approximately 3.21 billion parameters, while the **T5 base** model has around 2.85 billion parameters. One of the primary advantages of PEFT methods is their ability to adapt these large models by training only a small fraction of parameters.

For Llama, **LoKR** demonstrates remarkable parameter efficiency, requiring only 96,768 trainable parameters (0.0030% of the full model). This extreme efficiency does not substantially compromise performance, as LoKR achieves competitive ROUGE scores of 47.92/22.36/26.96. In contrast, **Llama Prefix Tuning** requires the most parameters (11,468,800, or 0.3557%), yet delivers lower performance than more efficient methods, suggesting diminishing returns from increasing parameter counts.

For T5, **IA3** represents an excellent balance between efficiency and performance, requiring only 294,912 parameters (0.0103%) while achieving near-top performance with ROUGE scores of 49.84/26.84/31.30. The **T5 AdaLoRA** implementation, while delivering the highest overall performance, requires significantly more parameters (7,079,616, or 0.2478%).

4.1.3 Memory Utilization Analysis

GPU memory utilization varies significantly across methods, with **QLoRA** standing out for its exceptional memory efficiency. QLoRA requires only 6.134 GB for Llama and 6.542 GB for T5, representing approximately 20% of the memory needed by other methods. This dramatic reduction in memory requirements makes QLoRA particularly valuable for deployment scenarios with limited computational resources.

Most other PEFT methods for Llama require around 29 GB of GPU memory, with minimal variations between techniques. For T5, memory utilization is generally higher, ranging from 36.8 GB to 40.5 GB across methods. This higher memory requirement for T5 is likely due to its encoder-decoder architecture, compared to Llama's decoder-only design.

Interestingly, methods with fewer trainable parameters don't necessarily consume less GPU memory. For example, **LoKR** for T5 uses only 184,320 parameters (0.0065%) but still requires 37.585 GB of memory. This indicates that the memory bottleneck primarily lies in storing model activations rather than trainable parameters.

Detailed Analysis of Individual PEFT Methods

- **Low-Rank Adaptation (LoRA)**: Achieves ROUGE-1 scores of 47.76 (Llama) and 49.71 (T5). Offers a strong performance-to-parameter ratio but requires moderate GPU memory: 29.3 GB (Llama), 36.9 GB (T5).

- **Quantized LoRA (QLoRA)**: Delivers scores of 44.31 (Llama) and 44.16 (T5) with exceptional memory efficiency (6.1 GB). Ideal for low-resource environments with minimal performance loss.
- **Low-Rank Hadamard Product (LoHA)**: Improves expressiveness and reaches 47.06 (Llama) and 50.19 (T5). Uses more parameters than LoRA but offers better performance, especially for longer sequences.
- **Low-Rank Kronecker Product (LoKR)**: Most parameter-efficient, requiring only 96K (Llama) and 184K (T5) parameters. Still achieves strong results: 47.92 (Llama) and 49.73 (T5), though GPU usage remains high.
- **Adaptive LoRA (AdaLoRA)**: Provides top performance: 49.07 (Llama) and 50.54 (T5). Uses more parameters (2.75M, 7.08M) but allocates them efficiently based on layer importance.
- **Infused Adapter (IA3)**: Modifies activations and performs well with minimal parameters—294K for T5 (score: 49.84) and 745K for Llama (score: 48.37). GPU memory usage is relatively high.
- **Prefix Tuning**: Requires the most parameters for Llama (11.47M) with moderate results (47.75). T5 performs better (49.18) with fewer parameters (1.36M), but memory usage remains high.

Comparative Analysis: Llama vs. T5 Models

Our experiments reveal several key differences in how Llama and T5 respond to PEFT techniques:

- **Baseline Performance**: The base Llama model significantly outperforms the base T5 model on summarization tasks, with ROUGE-1 scores of 27.81 versus 14.64. This substantial initial gap suggests that Llama's pre-training regime may be better aligned with summarization capabilities.
- **Magnitude of Improvement**: While both models show significant gains with PEFT methods, the relative improvement for T5 is more dramatic. T5 with AdaLoRA improves ROUGE-1 scores by 245.2% over the base model, compared to a 76.4% improvement for Llama.
- **Absolute Performance**: Despite starting from a lower baseline, T5 with PEFT methods consistently achieves higher absolute ROUGE scores than Llama with equivalent techniques. The best T5 implementation (AdaLoRA) achieves ROUGE scores of 50.54, 26.59, and 30.67, outperforming the best Llama implementation (49.07, 22.55, and 26.97).

- **Method Suitability**: Different PEFT methods show varying effectiveness across architectures. IA3 demonstrates particularly strong performance for T5 relative to its parameter count, while LoHA shows exceptional ROUGE-L performance for Llama.
- **Resource Requirements**: T5 implementations consistently require more GPU memory than their Llama counterparts for equivalent PEFT methods. This higher memory requirement likely stems from T5's encoder-decoder architecture compared to Llama's decoder-only design.

These observations highlight that the choice of base model architecture significantly influences both the performance outcomes and efficiency characteristics of PEFT implementations.

Efficiency-Performance Trade-offs

Our results demonstrate clear trade-offs between efficiency metrics (trainable parameters, GPU memory) and performance metrics (ROUGE scores). Several patterns emerge from this analysis:

- **Diminishing Returns**: Methods with higher parameter counts generally achieve better performance, but with diminishing returns. AdaLoRA for T5 uses 7.08 million parameters (0.2478%) to achieve a ROUGE-1 score of 50.54, while LoRA uses 1.18 million parameters (0.0414%) to achieve 49.71—a marginal performance difference despite requiring six times more parameters.
- **Parameter Efficiency vs. Memory Efficiency**: Some methods achieve high parameter efficiency without corresponding memory efficiency. LoKR requires minimal trainable parameters but still consumes substantial GPU memory, indicating that the memory bottleneck lies in storing model activations rather than trainable parameters.
- **Quantization Benefits**: QLoRA stands out for its memory efficiency, using only about 20% of the GPU memory required by other methods. This dramatic reduction comes with a moderate performance cost, suggesting that quantization-based approaches offer valuable efficiency benefits for resource-constrained deployments.
- **Optimal Methods**: Several methods represent optimal trade-offs between efficiency and performance. For T5, IA3 achieves excellent performance with minimal parameters (0.0103%), placing it on the efficiency-performance frontier. Similarly, for Llama, LoKR balances strong performance with extreme parameter efficiency (0.0030%).

4.2 Performance Analysis of PEFT Methods on Classification Task

Our investigation evaluated multiple PEFT methods across different model architectures on the DAIR-AI emotion dataset, which contains six emotion categories: sadness, joy, love, anger, fear, and surprise. Table 4.3 presents the performance metrics obtained from our experiments.

PEFT Method	Accuracy	Precision	Recall	F1
Llama LoRA	0.92	0.93	0.93	0.93
Llama LoHA	0.91	0.91	0.91	0.91
Llama LoKr	0.87	0.88	0.87	0.87
Llama AdaLoRA	0.92	0.92	0.92	0.92
Llama QLoRA	0.92	0.91	0.92	0.91
Llama IA3	0.81	0.82	0.81	0.81
BERT-base-uncased	0.93	0.92	0.93	0.92
T5 LoRA	0.93	0.93	0.93	0.93
T5 LoHA	0.93	0.92	0.93	0.92
T5 LoKr	0.90	0.90	0.90	0.90
T5 AdaLoRA	0.87	0.86	0.87	0.86
T5 QLoRA	0.93	0.92	0.93	0.92
T5 IA3	0.93	0.93	0.93	0.93

Table 4.3: Performance Metrics of Different PEFT Methods on Emotion Classification

The performance analysis reveals several noteworthy patterns across the different PEFT methods. For the Llama architecture, LoRA and AdaLoRA achieved the highest performance with identical F1 scores of 0.93 and 0.92, respectively. This suggests that adaptive rank allocation in AdaLoRA provides comparable performance to the fixed-rank approach of standard LoRA when applied to Llama models. QLoRA also demonstrated strong performance (F1 = 0.91), indicating that quantization does not significantly compromise classification capability despite its memory advantages. In contrast, IA3 showed the lowest performance among PEFT methods applied to Llama, with an F1 score of 0.81, suggesting that simple rescaling of activations may be insufficient for this model architecture on emotion classification tasks.

For the T5 architecture, we observed more consistent performance across PEFT methods. T5 with LoRA and IA3 achieved the highest F1 scores (0.93), followed closely by LoHA and QLoRA (0.92). Interestingly, IA3 performed significantly better on T5 than on Llama, achieving a 12% higher F1 score. This indicates that T5's encoder-decoder architecture may be more amenable to the activation rescaling approach of IA3 compared to Llama's decoder-only architecture. The strong performance of IA3 on T5 is particularly notable considering its parameter efficiency.

The BERT baseline model showed strong performance (F1 = 0.92), comparable to the best-performing PEFT methods. This suggests that for this particular emotion classification task, highly

optimized smaller models can still compete with larger models using parameter-efficient adaptation techniques.

4.2.1 Efficiency Analysis of PEFT Methods

Table 4.4 presents the efficiency metrics for each PEFT method, including the number of trainable parameters, percentage of trainable parameters relative to the full model, GPU memory utilization, and rank configurations where applicable.

PEFT Method	**Trainable Parameters**	**% Trainable Params**	**GPU Utilization (in GB)**	**Rank**
Llama LoRA	1,853,440	0.0577	13.154	8
Llama LoHA	3,688,448	0.1147	13.810	8
Llama LoKr	147,456	0.0046	13.818	8
Llama AdaLoRA	2,771,616	0.0862	13.158	8
Llama QLoRA	917,504	0.0286	5.191	8
Llama IA3	477,184	0.0149	13.214	-
T5 LoRA	294,912	0.0103	12.876	8
T5 LoHA	9,437,184	0.3301	15.156	8
T5 LoKr	294,912	0.0103	15.054	8
T5 AdaLoRA	7,079,616	0.2478	12.928	8
T5 QLoRA	4,718,592	0.1653	4.468	8
T5 IA3	294,912	0.0103	12.876	-

Table 4.4: Efficiency Metrics of Different PEFT Methods

The efficiency analysis reveals notable variations in the resource requirements of different PEFT methods. For the Llama architecture, `LoKr` stands out for its remarkable parameter efficiency, requiring only 147,456 trainable parameters, which is just 0.0046% of the full model. This represents a 12.6x reduction in trainable parameters compared to the standard `LoRA` method, all while maintaining a reasonable GPU memory usage of 13.818 GB. However, this efficiency comes at the cost of performance, as `LoKr` achieves an F1 score that is 6% lower than `LoRA`.

On the other hand, `QLoRA` demonstrates significant memory efficiency across both architectures. For Llama, it requires 5.191 GB of GPU memory, and for T5, only 4.47 GB. This represents a 60.5% reduction in GPU memory usage for Llama and an even more impressive 65.3% reduction for T5, compared to the memory requirements of `LoRA`. These memory gains are achieved through the use of 4-bit quantization, while still maintaining performance levels comparable to full-precision methods (with F1 scores of 0.91 for Llama and 0.92 for T5).

When examining the T5 models, we observe that some PEFT methods require substantially more parameters, particularly `LoHA` and `AdaLoRA`, which involve 9.44M and 7.08M trainable parameters, respectively. This increase in parameter count can be attributed to the unique layer configurations of the T5 architecture, which differs from Llama. Interestingly, `IA3` and `LoRA` both have identical

parameter counts for T5 (294,912), yet they achieve the optimal F1 score of 0.93, while only using 0.0103% of the model's total parameters.

The difference in parameter efficiency between the two architectures is also striking. `LoKr` exhibited excellent parameter efficiency on Llama but did not offer the same advantage on T5. Conversely, `IA3` showcased both high performance and parameter efficiency on T5, yet underperformed on Llama despite utilizing more parameters than `LoKr`.

This analysis highlights the trade-offs between parameter efficiency and performance across different PEFT methods, and demonstrates that certain methods, such as `QLoRA` and `IA3`, provide notable efficiency gains without significantly compromising classification performance.

4.3 Performance-Efficiency Trade-offs

The analysis reveals critical performance-efficiency relationships across methods. These trade-offs differ substantially between architectures.

Llama Architecture

LoRA and **AdaLoRA** achieve optimal performance for Llama models, with F1 scores of 0.93 and 0.92 respectively. However, **QLoRA** demonstrates superior memory efficiency, requiring **61.3% less GPU memory** than LoRA despite a modest 2% performance decrease. The extreme parameter efficiency of **LoKr** (147K parameters) comes at the cost of a **6% performance reduction**, making it suitable for severely resource-constrained environments.

T5 Architecture

For T5 models, **IA^3** emerges as the Pareto-optimal method, achieving peak F1 (0.93) with minimal parameters (294K). While **LoRA** matches this performance, **QLoRA** provides dramatic memory savings (**65.3% reduction**) with only 1% accuracy loss, establishing itself as the preferred choice for memory-limited deployments.

4.4 Architectural and Methodological Insights

LoRA Family Analysis

- **Base LoRA**: Maintains strong performance-efficiency balance across architectures (F1: 0.93 both models)
- **LoHA**: Shows theoretical promise but practical limitations - requires **32× more parameters** than LoRA for equivalent T5 performance

- **LoKr**: Demonstrates exceptional parameter efficiency (0.003% of total) for Llama, ideal for edge deployment despite 6% accuracy trade-off

Adaptive Methods

AdaLoRA's dynamic parameter allocation shows mixed effectiveness:

- **Llama**: Comparable performance to LoRA (F1: 0.92) but requires **49% more parameters**
- **T5**: Significant performance degradation (F1: 0.86) with **24× parameter increase** versus LoRA

Memory-Optimized Approaches

QLoRA achieves remarkable memory efficiency:

- **Llama**: 35.5% memory reduction vs LoRA (13.1GB → 5.1GB)
- **T5**: 65.3% memory reduction vs LoRA (12.88GB → 4.46GB)

with minimal accuracy loss (1-2%), making it ideal for resource-constrained environments.

Architecture-Specific Behavior

IA^3 demonstrates dramatic architectural dependence:

- **T5**: Peak performance (F1: 0.93) with minimal parameters (294K)
- **Llama**: Lowest performance (F1: 0.81) despite identical implementation

Chapter 5

Conclusion

This study systematically evaluated parameter-efficient fine-tuning (PEFT) techniques for large language models (LLMs) across summarization and classification tasks. The research addressed the critical challenge of adapting increasingly complex LLMs like **T5** and **Llama** to specific domains while maintaining computational efficiency. Through comprehensive experimentation, this work demonstrates that PEFT methods can effectively balance performance and resource requirements, enabling practical deployment of LLMs in constrained environments.

5.1 Key Contributions and Findings

The investigation yielded several significant insights:

- **Performance Parity**: PEFT methods achieved comparable or superior performance to full fine-tuning while requiring less than 0.1% of trainable parameters. AdaLoRA emerged as particularly effective for summarization tasks, improving ROUGE-1 scores by **76.4%** (Llama) and **245.2%** (T5) over base models.
- **Architectural Insights**: The encoder-decoder structure of T5 demonstrated greater adaptability for generative tasks, while decoder-only Llama architectures showed stronger baseline performance before adaptation.
- **Memory Efficiency**: QLoRA stood out for its low GPU memory requirements, using only **20%** of the memory needed by other methods while maintaining competitive performance.

5.2 Theoretical and Practical Implications

The success of methods like **LoKr** (requiring only 0.003% parameters) and **IA^3** challenges conventional assumptions about parameter modification needs. These findings suggest the existence of an *efficiency frontier* where strategic parameter updates yield maximal performance gains.

From an applied perspective, the results provide clear guidance for practitioners:

- **Memory-constrained environments**: QLoRA offers optimal trade-offs
- **Parameter-limited scenarios**: LoKr and IA^3 provide maximal efficiency
- **Multi-task systems**: PEFT enables dynamic model adaptation without maintaining multiple full-sized models

5.3 Limitations and Future Directions

While comprehensive, this study presents several opportunities for extension:

- Evaluation on larger (>3B parameter) and multilingual models
- Investigation of robustness and fairness aspects
- Exploration of continual learning scenarios

5.4 Broader Impact

This research contributes to sustainable AI development by demonstrating that effective model adaptation doesn't require excessive computational resources. The findings particularly benefit:

- Organizations with limited AI infrastructure
- Researchers investigating efficient transfer learning
- Developers creating modular NLP systems

The results underscore the potential for parameter-efficient techniques to democratize access to state-of-the-art NLP capabilities while reducing environmental impact through lower energy consumption.

Chapter 6

Bibliography

1. Dhinagar, Nikhil J., et al. "Parameter efficient fine-tuning of transformer-based masked autoencoder enhances resource constrained neuroimage analysis." Medical Imaging 2025: Computer-Aided Diagnosis. Vol. 13407. SPIE, 2025

2. Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." ICLR 1.2 (2022): 3.

3. Zhang, Z., Yang, X. (2021, June 15). QLoRA: Quantized Low-Rank Adaptation for Efficient Transfer Learning.

4. Xu, Yuhui, et al. "Qa-lora: Quantization-aware low-rank adaptation of large language models." arXiv preprint arXiv:2309.14717 (2023).

5. Zhao, Shu, et al. "KALAHash: Knowledge-Anchored Low-Resource Adaptation for Deep Hashing." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 39. No. 10. 2025..

6. Lin, J., Liu, Z., Li, Z. (2022, August 12). LoKR: Low-Rank Knowledge Retention for Transformer Fine-Tuning.

7. Lu, Haodong, et al. "Adaptive Rank, Reduced Forgetting: Knowledge Retention in Continual Learning Vision-Language Models with Dynamic Rank-Selective LoRA." arXiv preprint arXiv:2412.01004 (2024).

8. Aghajanyan, A., Xie, Y. (2022, July 8). IA3: Interpretable and Adaptive Attention for Pretrained Language Models.

9. Zhang, Hongyi, et al. "Selective prefix tuning for pre-trained language models." Findings of the Association for Computational Linguistics ACL 2024. 2024.

10. Liu, J., Lin, Y. (2022, April 5). P-Tuning: Parameterized Prompt Tuning for Language Models.