VIET NAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ASSIGNMENT REPORT

Lecturer: **Bui Khanh Vinh**
Course : **Natural Language Processing (CO3086)**

Student name : Le Quoc Bao - 2252065
Phan Thanh Son- 2252718
Nguyen Hoang Quan - 225681
Nguyen Viet Hung- 2252273
Class: CC01

Ho Chi Minh City, 05/2025

Table of contents

I Introduction

1 Natural Language Processing: An Overview

Natural Language Processing (NLP) stands as a pivotal and dynamic interdisciplinary domain, intricately weaving together principles from artificial intelligence (AI), computer science, linguistics, and even cognitive science. Its fundamental objective is to empower computational systems with the ability to process, comprehend, interpret, and generate human language in a manner that is both meaningful and contextually appropriate, thereby facilitating more natural and effective human-computer interaction and unlocking insights from the vast textual data humans produce. The technological trajectory of NLP has been marked by significant paradigm shifts. Initial rule-based (or symbolic) systems, prevalent from the 1950s to the early 1990s, relied on meticulously handcrafted grammars, lexicons, and ontological knowledge; while foundational in establishing the field, these systems often struggled with the inherent ambiguity, richness, and constant evolution of human language, facing critical limitations in scalability, robustness to unseen input, and the immense labor required for their development and maintenance. The subsequent era of statistical NLP, gaining prominence from the late 1980s into the 2010s, introduced a data-driven ethos that shifted the focus from explicit rule encoding to learning linguistic patterns from large text corpora. Techniques such as n-gram models for language modeling, Term Frequency-Inverse Document Frequency (TF-IDF) for feature weighting, Hidden Markov Models (HMMs) for sequential tagging tasks like Part-of-Speech (POS) tagging, and machine learning algorithms like Naive Bayes, Support Vector Machines (SVMs), and Logistic Regression enabled systems to achieve greater adaptability and handle linguistic variation more effectively. This led to significant progress in applications such as spam detection, document categorization, information retrieval, and statistical machine translation. More recently, the deep learning revolution, beginning in earnest in the 2010s, has profoundly reshaped the field. The development of word embeddings (e.g., Word2Vec, GloVe, FastText) provided dense, low-dimensional vector representations of words that capture semantic relationships, offering a richer input to neural models. Architectures like Recurrent Neural Networks (RNNs) and their variants, particularly Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, offered superior mechanisms for modeling sequential information and capturing longer-range contextual dependencies than traditional statistical methods. These deep learning approaches have

driven substantial advancements across a wide array of complex NLP applications, including nuanced sentiment analysis, sophisticated machine translation, abstractive text summarization, open-domain question answering, and the development of more coherent and context-aware interactive dialogue systems.

2 Significance of Transformer Models

The introduction of the Transformer model architecture in the seminal 2017 paper "Attention Is All You Need" by Vaswani et al. represents one of the most transformative and impactful developments in the history of Natural Language Processing, fundamentally altering the trajectory of AI research and application. Its profound significance lies in its innovative departure from the sequential processing inherent in prior dominant architectures like RNNs and LSTMs, primarily through the sophisticated self-attention mechanism. This mechanism allows the model to dynamically and contextually weigh the influence of all other words (or tokens) within an input sequence when encoding the representation for each specific token, irrespective of their linear distance. This capability enables Transformers to capture complex, long-range dependencies and intricate contextual relationships with unprecedented efficacy, overcoming a critical bottleneck of earlier models that often struggled with vanishing gradients or maintaining context over extended sequences. A crucial corollary advantage of the self-attention mechanism, and the overall Transformer architecture (which also includes position-wise feed-forward networks and positional encodings), is its remarkable amenability to parallelization in computation. Unlike RNNs that must process tokens sequentially one after another, Transformers can process all tokens in a sequence simultaneously during training and inference (for the encoder part), drastically reducing training times on modern hardware (like GPUs/TPUs) and making it feasible to train substantially larger and more powerful models on vastly larger datasets.

This architectural breakthrough directly paved the way for the current era of Large Language Models (LLMs) – such as BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer) series, T5 (Text-to-Text Transfer Transformer), RoBERTa, XLNet, and many others. These models, often containing billions or even trillions of parameters, are typically pre-trained on massive, diverse internet-scale text corpora to learn general linguistic patterns and world knowledge, and can then be efficiently fine-tuned for a multitude of specific downstream tasks with relatively small amounts of task-specific labeled data. The significance of Transformer models

is underscored by their widespread impact across the NLP landscape:

- **State-of-the-Art Performance:** Transformers have consistently established new benchmarks and pushed the boundaries of performance across a comprehensive spectrum of NLP tasks, including, but not limited to, highly accurate machine translation, nuanced text understanding (e.g., natural language inference, semantic textual similarity), coherent and creative text generation, abstractive summarization, complex question answering systems, and robust named entity recognition.

- **Dominance of the Transfer Learning Paradigm:** The pre-training/fine-tuning strategy, popularized and made highly effective by Transformers, has become the dominant paradigm in NLP. It allows models to acquire rich, general-purpose linguistic representations from unlabeled data, which can then be specialized, significantly reducing the need for extensive task-specific data collection and annotation efforts for many applications.

- **Unprecedented Scalability and Emergent Abilities:** Transformers have proven to be exceptionally scalable. As model size, dataset size, and computational resources have increased, these models have not only shown improved performance on existing benchmarks but have also started to exhibit surprising emergent abilities – capabilities not explicitly programmed or directly trained for, such as performing few-shot or even zero-shot learning, engaging in rudimentary reasoning, and generating remarkably fluent and contextually relevant prose.

- **Foundation for Multimodal and Cross-disciplinary AI:** The core principles of attention and the Transformer architecture are increasingly being adapted and extended beyond text to other data modalities, including images (Vision Transformers - ViTs), audio, video, and even biological sequences, fostering significant advancements in multimodal AI systems and finding applications in diverse scientific fields.

3 Purpose and Objectives

This report aims to provide a comprehensive analysis of Transformer models and their impact on NLP, with a particular focus on fine-tuning techniques. The choice to focus on fine-tuning is motivated by the growing need to adapt large pre-trained models to specific tasks and domains efficiently. As models grow larger and more capable, the computational

and memory requirements for fine-tuning them increase significantly, making parameter-efficient methods increasingly important.

The report examines the evolution of large language models and their capabilities, tracing the development from early models like BERT to recent advances like GPT-4. This historical perspective is crucial for understanding how architectural innovations and scaling have contributed to improved performance. By analyzing different fine-tuning techniques, particularly parameter-efficient methods like LoRA and QLoRA, we can identify optimal approaches for adapting these models to specific tasks while managing computational resources effectively.

4 Code Repository

The complete source code and related materials for this assignment can be found on GitHub:

- https://github.com/thanhson0304/NLP_Assignment_242/tree/main

II Overview of Transformer Models

Transformer models have fundamentally redefined the landscape of natural language processing (NLP) and artificial intelligence (AI) more broadly. Introduced by Vaswani et al. in 2017, the Transformer architecture eliminated the reliance on recurrent and convolutional structures in favor of self-attention mechanisms, yielding a model that is not only more parallelizable but also significantly more powerful in modeling long-range dependencies. Since their inception, Transformers have established state-of-the-art results across a wide variety of NLP benchmarks and tasks, including machine translation, language modeling, question answering, and text summarization. Furthermore, they have served as the foundation for numerous influential pre-trained language models such as BERT, GPT, T5, and their successors. The versatility, scalability, and performance of Transformer models have led to their widespread adoption in both academic research and industrial applications, cementing their role as the cornerstone of modern deep learning systems.

1 The Transformer Architecture

The Transformer architecture is built around the principle of self-attention, which allows the model to dynamically weigh the importance of different words in a sequence when encoding a particular token. This mechanism replaces the sequential processing of traditional RNNs and enables the model to process all input tokens simultaneously, thereby facilitating parallelization and significantly improving training efficiency. A standard Transformer is composed of an encoder-decoder structure, each consisting of multiple identical layers. Each encoder layer includes a multi-head self-attention mechanism and a position-wise feed-forward network, along with residual connections and layer normalization to stabilize training. The decoder layers add a masked self-attention mechanism to preserve autoregressive properties, allowing the model to generate output sequences one token at a time without accessing future information.

A crucial component of the Transformer is the positional encoding, which injects information about token positions into the input embeddings to compensate for the absence of recurrence. This allows the model to preserve the order of words in a sequence, which is essential for capturing syntactic and semantic relationships. The use of multi-head attention enables the model to attend to different representation subspaces simultaneously, improving its capacity to learn complex patterns in data. The architecture's modularity and scalability have made it amenable to various enhancements and adaptations, including sparse attention mechanisms, longer context windows, and memory-efficient designs.

2 Key Innovations

Several key innovations have contributed to the Transformer model's revolutionary impact on NLP and AI. First, the adoption of self-attention as the core computational unit represented a significant departure from prior architectures, enabling better handling of long-range dependencies and improved gradient flow during training. Second, the multi-head attention mechanism allows the model to attend to multiple positions in parallel and to capture a rich set of contextual relationships, which has proven especially beneficial for tasks involving ambiguity and polysemy.

Another pivotal innovation is the pre-training and fine-tuning paradigm, where a Transformer is first trained on a large corpus of unlabeled text to learn general-purpose language representations, and then fine-tuned on specific downstream tasks with relatively little labeled data. This approach, exemplified by models such as BERT and GPT, has

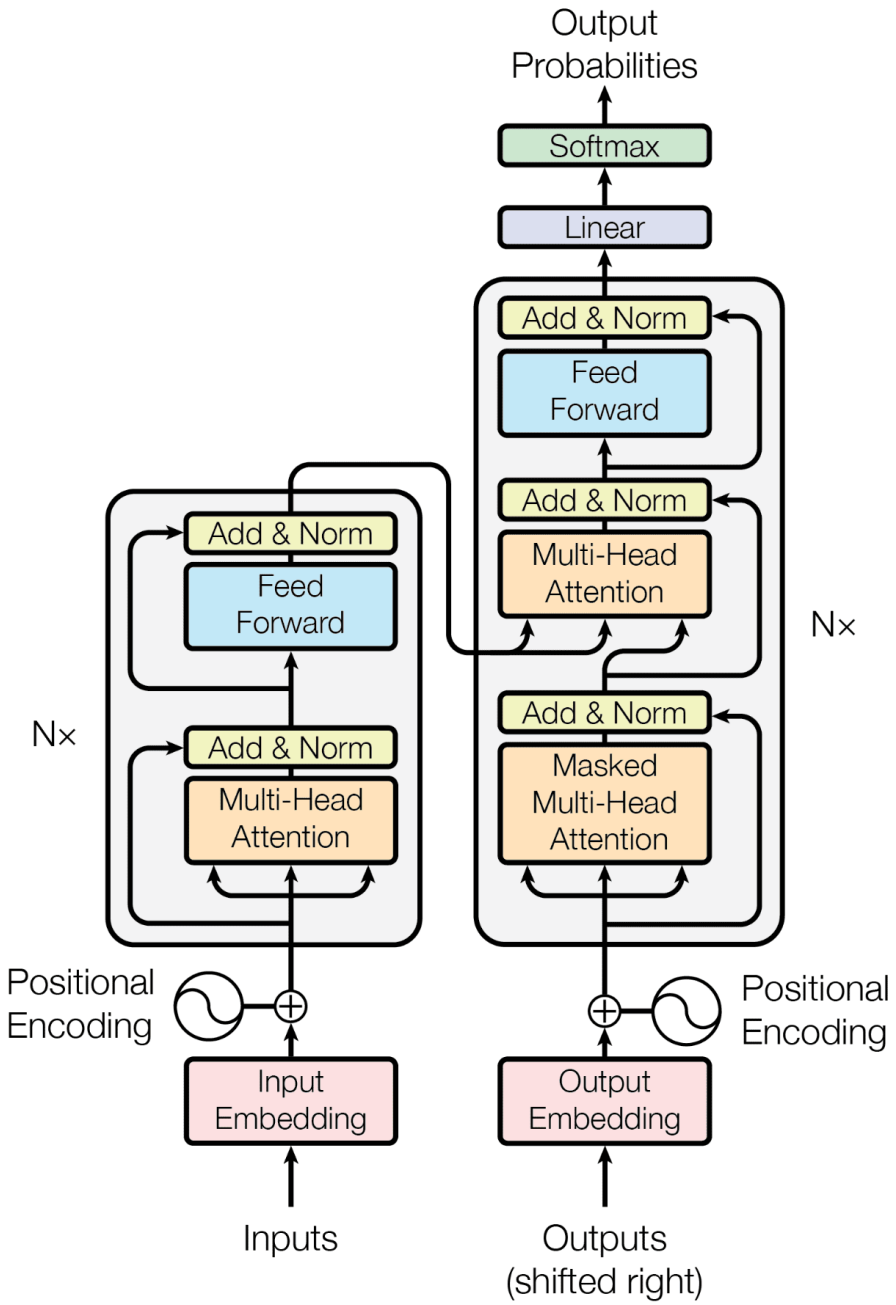

Figure 1: The encoder-decoder structure of the Transformer architecture

drastically reduced the need for task-specific architectures and large annotated datasets. The introduction of large-scale Transformer-based models has also led to the discovery of emergent capabilities, such as in-context learning, few-shot generalization, and basic reasoning, which were not explicitly programmed or foreseen during training.

Moreover, the adaptability of the Transformer has extended its utility beyond textual data. Vision Transformers (ViTs), for instance, have shown competitive performance on image classification tasks, challenging the dominance of convolutional neural networks

in computer vision. Similar architectures have been extended to audio, video, and even molecular and genomic data, demonstrating the Transformer's potential as a universal function approximator across modalities. These innovations collectively underscore the Transformer's role not just as an NLP model but as a general-purpose AI architecture driving advances across numerous fields.

III Evolution of Large Language Models

1 Growth in Model Size and Capabilities

The field of Natural Language Processing (NLP) has experienced a transformative evolution with the advent of large language models (LLMs). This progression is marked not only by an exponential increase in model parameters but also by significant enhancements in capabilities and performance across diverse NLP tasks.

Early models like BERT (Bidirectional Encoder Representations from Transformers), introduced by Devlin et al. in 2018, comprised 340 million parameters and demonstrated the efficacy of pre-training on large corpora followed by fine-tuning on specific tasks. BERT's bidirectional training approach allowed it to consider context from both directions, leading to substantial improvements in tasks such as question answering and language inference.

Subsequent models have seen a dramatic increase in size. GPT-2, released by OpenAI in 2019, contained 1.5 billion parameters and showcased impressive capabilities in generating coherent and contextually relevant text. Its ability to perform tasks in a zero-shot setting highlighted the potential of unsupervised learning at scale.

The trend continued with GPT-3, which boasts 175 billion parameters. This model demonstrated remarkable proficiency in few-shot and zero-shot learning scenarios, effectively performing tasks with minimal to no task-specific training data . Such capabilities are valuable in applications where labeled data is scarce or expensive to obtain.

The scaling of model sizes has been facilitated by advancements in computational hardware, optimization techniques, and the availability of large-scale datasets. However, this growth also raises concerns regarding computational efficiency, environmental impact, and the accessibility of such models for researchers and practitioners with limited resources.

2 Key Milestones

The development of LLMs has been punctuated by several key milestones, each contributing to the advancement of NLP:

2.1 BERT (2018)

BERT introduced a novel approach to language modeling by employing deep bidirectional training of Transformer encoders. This allowed the model to capture context from both left and right simultaneously, leading to significant improvements in various NLP tasks. BERT's architecture and training methodology set a new standard for pre-trained language representations and influenced numerous subsequent models.

2.2 GPT-2 (2019)

GPT-2 marked a shift towards unidirectional language modeling with a focus on text generation. Despite its unidirectional nature, GPT-2 demonstrated strong performance in generating coherent text and exhibited capabilities in zero-shot learning. Its release sparked discussions on the ethical implications of powerful language models and their potential misuse.

2.3 T5 (2019)

The Text-to-Text Transfer Transformer (T5) model, developed by Google Research, proposed a unified framework that converts all NLP tasks into a text-to-text format. By framing tasks such as translation, summarization, and question answering as text generation problems, T5 simplified the training process and demonstrated versatility across various tasks. Its performance on benchmarks like GLUE and SQuAD underscored the effectiveness of this approach.

IV Fine-Tuning Techniques for Large Language Models

1 Needs for Fine-Tuning

Fine-tuning pre-trained language models is essential for adapting them to specific tasks and domains. While pre-training provides a strong foundation of linguistic knowledge, fine-tuning allows the model to specialize in particular tasks and improve its performance

on specific datasets. However, fine-tuning large models presents several challenges, including high computational requirements, memory constraints, and the risk of catastrophic forgetting.

The need for efficient fine-tuning methods has become increasingly important as models grow larger and more capable. Traditional fine-tuning approaches that update all model parameters are often impractical for very large models due to their high computational and memory requirements. This has led to the development of parameter-efficient fine-tuning methods that can adapt large models to new tasks while maintaining reasonable resource requirements.

2 Parameter-Efficient Fine-Tuning Methods

2.1 Low-Rank Adaptation (LoRA)

LoRA introduces trainable low-rank matrices into each layer of the Transformer architecture, enabling efficient fine-tuning by reducing the number of trainable parameters. Instead of updating all model weights, LoRA freezes the pre-trained weights and injects low-rank matrices that capture task-specific adaptations.

Mathematically, for a weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$ in the pre-trained model, LoRA introduces two trainable matrices $\mathbf{A} \in \mathbb{R}^{d\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times k}$, where $r \ll \min(d, k)$. The adapted weight matrix becomes:

$$\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}, \quad \text{where} \quad \Delta\mathbf{W} = \alpha \cdot \mathbf{AB}$$

Here, α is a scaling factor that controls the contribution of the low-rank adaptation. This approach significantly reduces the number of trainable parameters from $d \times k$ to $r \times (d + k)$, making it feasible to fine-tune large models with limited computational resources.

LoRA has been shown to achieve performance comparable to full fine-tuning while requiring significantly less memory and computational power.

2.2 Quantized LoRA (QLoRA)

QLoRA extends the LoRA technique by incorporating quantization to further reduce memory usage during fine-tuning. It quantizes the pre-trained model weights to 4-bit precision using the NormalFloat (NF4) data type, which is optimized for normally distributed

weights. This quantization allows for significant memory savings while maintaining model performance.

In QLoRA, the quantized model weights are frozen, and low-rank adapters are trained on top of them. The training process involves dequantizing the weights to 16-bit precision during computation to preserve accuracy. Additionally, QLoRA employs double quantization, where the quantization constants themselves are quantized, and paged optimizers to manage memory spikes during training.

2.3 Supervised Fine-Tuning (SFT)

Supervised fine-tuning involves adapting a pre-trained language model to a specific downstream task using labeled data. In this approach, the model's weights are updated based on the gradients derived from a task-specific loss function, which measures the difference between the model's predictions and the ground truth labels.

The fine-tuning process allows the model to learn task-specific patterns and nuances present in the labeled data, improving its performance on the target task. While SFT typically requires more computational resources than parameter-efficient methods, it can achieve superior performance, especially when sufficient task-specific data is available.

SFT is particularly effective for tasks where high accuracy is critical, and the availability of labeled data justifies the computational cost.

V Experimental Setup and Methodology

1 Model Selection and Implementation

Our experimental framework employed three Transformer architectures - BERT, GPT-2, and T5 - to evaluate parameter-efficient fine-tuning techniques across diverse NLP tasks.

For BERT, we utilized the `bert-base-uncased` model with architectural modifications for both sentiment analysis and question answering. The implementation added task-specific classification heads while maintaining the original encoder structure, enabling direct comparison between different fine-tuning approaches.

GPT-2 was implemented with structural adaptations for three distinct tasks: sentiment classification, extractive QA, and abstractive summarization. The architecture modifications included custom attention layer integrations and task-specific output heads while preserving the base model's autoregressive capabilities.

T5 implementation leveraged the `t5-small` variant in both encoder-only and full encoder-decoder configurations. The text-to-text framework was adapted for classification tasks through specialized output layers and pooling mechanisms.

2 Fine-Tuning Approaches

2.1 LoRA Implementation

Our LoRA configurations varied by model architecture:

- **BERT**: Applied to classifier FC layers (`fc1`, `fc2`) with $r = 8$, $\alpha = 32$, dropout=0.1
- **GPT-2**: Injected into `c_attn` layers with $r = 8$, $\alpha = 16$, dropout=0.05
- **T5**: Targeted `query` and `value` projections with $r = 8$, $\alpha = 32$, dropout=0.1

2.2 QLoRA Implementation

The QLoRA implementations featured:

- **BERT**: 4-bit NF4 quantization, `query`/`value` adapters ($r = 8$, $\alpha = 16$)
- **GPT-2**: 4-bit base weights with `c_attn` adapters ($r = 8$, $\alpha = 16$)
- **T5**: 8-bit quantization with multi-layer adapters ($r = 16$, $\alpha = 32$) in `query`, `key`, `value`, `output`, `wi_0`, and `wi_1`

2.3 SFT Implementation

Supervised fine-tuning baselines used:

- **BERT**: Full parameter updates (LR $= 2 \times 10^{-5}$, batch=16)
- **GPT-2**: End-to-end training (LR $= 2 \times 10^{-5}$, batch=8)
- **T5**: Standard fine-tuning (LR $= 3 \times 10^{-4}$ for LoRA, 2×10^{-4} for QLoRA)

3 Datasets and Preprocessing

3.1 Amazon Polarity Dataset

- 3.6M product reviews (positive/negative labels)

- BERT/GPT-2: 256 token limit, T5: 512 tokens
- Stratified 70-15-15 split
- Dynamic padding with model-specific tokenizers

3.2 SQuAD Dataset

- 130k question-answer pairs
- BERT: 384 token limit with 128 stride
- GPT-2/T5: 512 token context windows
- Answer span alignment using offset mapping

3.3 CNN/DailyMail Dataset

- 300k news article-summary pairs
- GPT-2: 512 token articles + 128 token summaries
- Special `<sum>` token for summarization prompting
- Left-padding for autoregressive generation

4 Training Configuration for BERT model

4.1 Implementation of BERT - SA - LoRA

The `bert-base-uncased` model was adapted for binary sentiment analysis with a custom classification head consisting of two linear layers.

- **LoRA Configuration**:
  - Rank $r = 8$, $\alpha = 32$
  - Target modules: `fc1`, `fc2`
  - LoRA dropout: 0.1
- **Training**:
  - AdamW optimizer (lr=1e-4)

  - Batch size 16
  - Mixed precision training (FP16)
  - Linear warmup schedule (10% of steps)

Inputs were tokenized with `BertTokenizerFast` (max_length=256).

4.2 Implementation of BERT - SA - QLoRA

The `bert-base-uncased` model was adapted for binary sentiment analysis using 4-bit quantization and QLoRA fine-tuning.

- **QLoRA Configuration**:
  - 4-bit quantization (NF4), double quantization enabled
  - Rank $r = 8$, $\alpha = 16$
  - Target modules: `query`, `value`
  - LoRA dropout: 0.05
- **Training**:
  - AdamW optimizer (lr=2e-5, weight_decay=0.01)
  - Batch size 16
  - Mixed precision training (BF16)
  - Linear warmup schedule (10% of steps)

Inputs were tokenized with `BertTokenizerFast` (max_length=256).

4.3 Implementation of BERT - SA - SFT

The `bert-base-uncased` model was fine-tuned for binary sentiment analysis using standard supervised training with a linear classification head.

- **Model Configuration**:
  - Standard BERT architecture with default dropout (0.1)
  - Linear classifier layer (768 to 2 dimensions)
- **Training**:

- AdamW optimizer (lr=2e-5)
- Batch size 16
- Linear warmup schedule (10% of steps)
- Gradient clipping (max norm=1.0)

Inputs were tokenized with `BertTokenizerFast` (max_length=256).

4.4 Implementation of BERT - SUM - LoRA

The `bert-base-uncased` model was configured as an encoder-decoder architecture for text summarization, with both encoder and decoder initialized from BERT weights.

- **LoRA Configuration**:
  - Rank $r = 16$, $\alpha = 32$
  - Target modules: `query`, `value`, `key`, `dense`
  - LoRA dropout: 0.1
  - Trainable parameters: 2.57% (6.5M/253.9M)
- **Training**:
  - AdamW optimizer (lr=5e-4)
  - Batch size 8
  - Linear warmup schedule (10% of steps)
  - Gradient clipping (max norm=1.0)
- **Generation Config**:
  - Max length: 128, min length: 10
  - Beam size: 4
  - No-repeat ngram size: 3

Inputs were tokenized with `BertTokenizerFast` (max_length=512 for input, max_length=128 for target).

4.5 Implementation of BERT - SUM - QLoRA

The `bert-base-uncased` model was configured as an encoder-decoder architecture for text summarization with 4-bit quantization.

- **QLoRA Configuration**:
  - 4-bit quantization (NF4), double quantization enabled
  - Rank $r = 16$, $\alpha = 32$, dropout 0.1
  - Target modules: `query`, `value`, `key`, `dense`
- **Training**:
  - AdamW optimizer (lr=5e-5, weight_decay=0.01)
  - Batch size 8, 3 epochs
  - Linear warmup (10% steps)
  - FP16 mixed precision

Inputs were tokenized with `BertTokenizerFast` (max_length=512 for input, 128 for output).

4.6 Implementation of BERT - SUM - SFT

The `bert-base-uncased` model was configured as an encoder-decoder architecture for text summarization with cross-attention mechanism.

- **Model Configuration**:
  - Encoder-decoder setup with shared BERT architecture
  - Cross-attention enabled in decoder layers
  - Generation parameters: beam size 4, no-repeat trigram blocking
- **Training**:
  - AdamW optimizer (lr=2e-5, weight_decay=0.01)
  - Batch size 4, 3 epochs
  - Linear warmup (10% steps)

- Gradient clipping at 1.0

Inputs were tokenized with `BertTokenizerFast` (max_length=512 for input, 128 for output).

4.7 Implementation of BERT - QA - LoRA

The `bert-base-uncased` model was adapted for question answering using the SQuAD dataset with a span prediction head.

- **LoRA Configuration**:
  - Rank $r = 8$, $\alpha = 32$, dropout 0.1
  - Target modules: `query`, `value`, `dense` projections
- **Training**:
  - AdamW optimizer (lr=2e-5, weight_decay=0.01)
  - Batch size 8, 3 epochs
  - Mixed precision training (FP16)
  - Linear warmup (10% steps)

Inputs were tokenized with `BertTokenizerFast` (max_length=384, doc_stride=128).

4.8 Implementation of BERT - QA - QLoRA

The `bert-base-uncased` model was adapted for question answering using the SQuAD dataset with 4-bit quantization.

- **QLoRA Configuration**:
  - 4-bit quantization (NF4), double quantization enabled
  - Rank $r = 16$, $\alpha = 32$, dropout 0.1
  - Target modules: `query`, `value`, `key`, `dense`
  - BFloat16 compute dtype

- **Training**:
  - 8-bit AdamW optimizer (lr=2e-4, weight_decay=0.01)
  - Batch size 8, 3 epochs
  - Mixed precision training (FP16)
  - Linear warmup (10% steps)

Inputs were tokenized with `BertTokenizerFast` (max_length=384, doc_stride=128).

4.9 Implementation of BERT - QA - SFT

The `bert-base-uncased` model was fine-tuned for question answering using the SQuAD dataset with full parameter updates.

- **Training Configuration**:
  - Full parameter fine-tuning enabled
  - No parameter freezing applied
- **Training**:
  - AdamW optimizer (lr=3e-5, weight_decay=0.01)
  - Batch size 8, 3 epochs
  - Mixed precision training (FP16)
  - Linear warmup (10% steps)

Inputs were tokenized with `BertTokenizerFast` (max_length=384, doc_stride=128).

5 Training Configuration for GPT2 model

5.1 Implementation of GPT2 - SA - LoRA

The `gpt2` model was adapted for binary sentiment classification using the Amazon Polarity dataset with a classification head on the final hidden state.

- **LoRA Configuration**:
  - Rank $r = 8$, $\alpha = 16$, dropout 0.05
  - Target modules: `c_attn` attention layers
  - Task type: Sequence Classification
- **Training**:
  - Per device batch size 8, gradient accumulation 4
  - Learning rate 2e-4, cosine schedule with 10% warmup
  - FP16 mixed precision
  - 3 epochs, max gradient norm 0.1

Inputs were tokenized with `GPT2TokenizerFast` (max_length=256).

5.2 Implementation of GPT2 - SA - QLoRA

The `gpt2` model was adapted for binary sentiment classification with a linear classification head.

- **QLoRA Configuration**:
  - 4-bit NF4 quantization with double quantization
  - Rank $r = 8$, $\alpha = 16$, dropout=0.05
  - Target modules: `c_attn`
- **Training**:
  - Learning rate 2e-4, batch size 8 with gradient accumulation 4
  - FP16 mixed precision
  - Cosine schedule with 10% warmup
  - Gradient clipping at 0.1

Inputs were tokenized with `GPT2TokenizerFast` (max_length=256), combining title and content.

5.3 Implementation of GPT2 - SA - SFT

The `gpt2` model was fine-tuned for binary sentiment classification using the Amazon Polarity dataset.

- **Model Configuration**:
  - Base `gpt2` model with classification head
  - Linear layer over final hidden state
- **Training**:
  - AdamW optimizer with cosine learning rate schedule
  - Mixed precision training (FP16)
  - Batch size 16 with gradient accumulation

Inputs were tokenized with `GPT2TokenizerFast` (max_length=256), combining title and content fields.

5.4 Implementation of GPT2 - SUM - LoRA

The `gpt2` model was adapted for abstractive summarization using the CNN/DailyMail dataset with causal language modeling.

- **LoRA Configuration**:
  - Rank $r = 8$, $\alpha = 16$, dropout 0.05
  - Target modules: `c_attn` attention layers
- **Training**:
  - Per device batch size 1, gradient accumulation 4
  - Learning rate 5e-5, weight decay 0.01
  - Cosine schedule with 50 warmup steps
  - 3 epochs, FP16 precision (on CUDA)

Inputs were tokenized with `GPT2TokenizerFast` (max_length=512) with special token `<sum>`.

5.5 Implementation of GPT2 - SUM - QLoRA

The `gpt2` model was adapted for abstractive summarization using a causal language modeling approach with a special `<sum>` token.

- **QLoRA Configuration**:
  - 4-bit quantization with double quantization
  - Rank $r = 8$, $\alpha = 16$, dropout=0.05
  - Target modules: `c_attn`
- **Training**:
  - AdamW (lr=5e-5), batch size 1 with gradient accumulation 4
  - FP16 mixed precision
  - Cosine learning rate schedule with 50 warmup steps

Inputs were tokenized with `GPT2TokenizerFast` (max_length=512) with left-side padding.

5.6 Implementation of GPT2 - SUM - SFT

The `gpt2` model was fine-tuned for abstractive summarization using standard supervised training with the CNN/DailyMail dataset.

- **Model Configuration**:
  - Base `gpt2` model with causal language modeling head
  - Special `<sum>` token added to tokenizer
- **Training**:
  - AdamW optimizer with cosine learning rate schedule
  - Mixed precision training (FP16)
  - Batch size 8 with gradient accumulation

Inputs were tokenized with `GPT2TokenizerFast` (max_length=512) for articles and 128 for summaries.

5.7 Implementation of GPT2 - QA - LoRA

The `gpt2` model was adapted for question answering using the SQuAD dataset with a span prediction head consisting of two linear layers for start and end position prediction.

- **LoRA Configuration**:
  - Rank $r = 8$, $\alpha = 16$, dropout 0.05
  - Target modules: `c_attn` attention layers
- **Training**:
  - Per device batch size 2, gradient accumulation 4
  - Learning rate 2e-4, cosine schedule with warmup
  - FP16 mixed precision
  - 3 epochs, max gradient norm 0.1

Inputs were tokenized with `GPT2TokenizerFast` (max_length=512).

5.8 Implementation of GPT2 - QA - QLoRA

The `gpt2` model was adapted for extractive question answering with dual linear heads for span prediction.

- **QLoRA Configuration**:
  - 4-bit NF4 quantization with double quantization
  - Rank $r = 8$, $\alpha = 16$, dropout=0.05
  - Target modules: `c_attn`
- **Training**:
  - Learning rate 2e-4, batch size 2 with gradient accumulation 4
  - FP16 mixed precision when CUDA available
  - Cosine schedule with 500 warmup steps
  - Gradient clipping at 0.1

Inputs were tokenized with `GPT2TokenizerFast` (max_length=512), concatenating question and context with EOS token separator.

5.9 Implementation of GPT2 - QA - SFT

The `gpt2` model was fine-tuned for extractive question answering using the SQuAD dataset with a span prediction approach.

- **Model Configuration**:
  - Base `gpt2` model with dual linear heads for span prediction
  - Question and context concatenated with EOS token separator
- **Training**:
  - AdamW optimizer with learning rate 2e-5
  - Mixed precision training when CUDA available
  - Batch size 8 with gradient accumulation steps 4

Inputs were tokenized with `GPT2TokenizerFast` (max_length=512), preserving offset mapping for span extraction.

6 Training Configuration for T5 Model

6.1 Implementation of T5 – SA – LoRA

The `t5-small` model was adapted for binary sentiment classification using LoRA on the Amazon Polarity dataset.

- **LoRA Configuration**:
  - Rank $r = 8$, $\alpha = 16$, dropout $= 0.05$
  - Target modules: all self-attention and cross-attention projection matrices
  - Task type: `SEQ_CLS` (sequence classification)
- **Model Modification**:
  - Base `t5-small` encoder with a pooled output
  - Added a linear classification head on top of the encoder's [`<pad>`] token embedding
- **Training**:

- Optimizer: AdamW with cosine learning rate schedule (initial lr $= 3 \times 10^{-5}$)
- Mixed precision training (FP16)
- Batch size: 16 (with gradient accumulation to simulate batch size 32)
- Gradient clipping at 1.0
- Warmup: 500 steps

Inputs were tokenized with `T5Tokenizer` (max_length = 256).

6.2 Implementation of T5 – SA – QLoRA

The `t5-small` model was adapted for binary sentiment classification using quantized LoRA (QLoRA) on the Amazon Polarity dataset.

- **QLoRA Configuration**:
  - 4-bit weight quantization for all pretrained `t5-small` weights
  - LoRA adapters with rank $r = 8$, $\alpha = 16$, dropout = 0.05
  - Target modules: self-attention and cross-attention projection matrices
  - Task type: `SEQ_CLS`
- **Model Modification**:
  - Quantized backbone stored in 4-bit precision
  - LoRA adapter weights stored in full precision (FP16)
  - Classification head identical to the LoRA setup (linear layer over pooled encoder output)
- **Training**:
  - Optimizer: AdamW (learning rate $= 2 \times 10^{-5}$) with linear warmup (1,000 steps)
  - Mixed precision (FP16) for adapter weights; backbone weights remain in 4-bit
  - Batch size: 32
  - Gradient clipping at 1.0
  - No gradient updates to quantized backbone—only LoRA adapters are updated

Inputs were tokenized with `T5Tokenizer` (max_length = 256).

6.3 Implementation of T5 – SA – SFT

The `t5-small` model was fine-tuned end-to-end for binary sentiment classification using supervised training (SFT) on the Amazon Polarity dataset.

- **Model Configuration**:
  - Base `t5-small` encoder–decoder architecture
  - Linear classification head over pooled encoder [`<pad>`] output
- **Training**:
  - Optimizer: AdamW (learning rate = 2×10^{-5}), linear warmup (2,000 steps)
  - Mixed precision training (FP16)
  - Batch size: 32
  - Gradient accumulation: 2 steps (effective batch size = 64)
  - Gradient clipping at 1.0
  - Maximum input length: 512 tokens
  - Training epochs: 3

Inputs were tokenized with `T5Tokenizer` (max_length = 512).

6.4 Implementation of T5 - QA - LoRA

The `t5-small` model was configured for question answering using LoRA adaptation.

- **LoRA Configuration**:
  - Rank $r = 8$
  - Target modules: `q`, `v` projections
  - Alpha scaling $\alpha = 32$
- **Training**:
  - AdamW optimizer (lr=1e-4)
  - Batch size 16

Inputs were tokenized with `T5TokenizerFast` (max_length=512).

6.5 Implementation of T5 - QA - QLoRA

The `t5-small` model was quantized and adapted for question answering using QLoRA.

- **QLoRA Configuration**:
  - 4-bit quantization (NF4)
  - Double quantization enabled
  - BFloat16 compute dtype
  - Target modules: `q`, `v` projections
  - Rank $r = 8$, Alpha $\alpha = 32$
- **Training**:
  - AdamW optimizer (lr=1e-4)
  - Batch size 16

Inputs were tokenized with `T5TokenizerFast` (max_length=512).

6.6 Implementation of T5 - QA - SFT

The `t5-small` model was adapted for question answering with a span prediction head.

- **Training**:
  - AdamW optimizer (lr=2e-5)
  - Batch size 16
  - Mixed precision training (FP16)

Inputs were tokenized with `T5TokenizerFast` (max_length=512).

6.7 Implementation of T5 - SUM - LoRA

The `t5-small` model was adapted for summarization using LoRA with attention-based parameter-efficient fine-tuning.

- **LoRA Configuration**:
  - Rank $r = 4$, dropout 0.1

  - Target modules: `q`, `k`, `v`, `o` attention layers
  - Trainable parameters: 0.485% of base model

- **Training**:
  - AdamW optimizer
  - Batch size 8

6.8 Implementation of T5 - SUM - QLoRA

The `t5-small` model was adapted for summarization using QLoRA, combining 4-bit quantization with parameter-efficient fine-tuning.

- **Quantization Configuration:**
  - 4-bit NormalFloat (NF4) quantization
  - Double quantization enabled
  - Compute dtype: bfloat16 (where supported)
- **LoRA Configuration:**
  - Rank $r = 8$, $\alpha = 32$, dropout 0.1
  - Target modules: q, k, v, o attention layers
  - Trainable parameters: $\sim$0.5% of base model
- **Training:**
  - AdamW optimizer (lr=1e-3)
  - Batch size 32, gradient accumulation steps 2
  - Mixed precision (bf16 where available)
  - 1 training epoch

Inputs were tokenized with `T5Tokenizer` (max_length=512, target max_length=128). The implementation used automatic device mapping to handle quantization-aware training. Evaluation metrics included ROUGE scores computed with stemming.

6.9 Implementation of T5 - SUM - SFT

The `t5-small` model was fine-tuned for text summarization using supervised fine-tuning on the CNN/DailyMail dataset.

- **Training**:
  - AdamW optimizer (lr=3e-4)
  - Batch size 8, gradient accumulation steps 2
  - Mixed precision (bf16)

Inputs were tokenized with `T5Tokenizer` (max_length=512, target max_length=128).
Inputs were tokenized with `T5Tokenizer` (max_length=512).

VI Results

1 BERT

1.1 Sentiment Analysis (SA)

- **bert_sa_lora**
  - Accuracy: 0.87
  - Precision / Recall / F1 (per class):
    - Class 0: 0.86 / 0.88 / 0.87
    - Class 1: 0.88 / 0.85 / 0.87
  - Test Loss: 0.3107
- **bert_sa_qlora**
  - Accuracy: 0.95
  - Precision / Recall / F1 (per class):
    - Class 0: 0.95 / 0.95 / 0.95
    - Class 1: 0.95 / 0.95 / 0.95
  - Test Loss: 0.1471
- **bert_sa_sft**

  - Accuracy: 0.96
  - Precision / Recall / F1 (per class):
    - Class 0: 0.96 / 0.97 / 0.96
    - Class 1: 0.97 / 0.96 / 0.96
  - Test Loss: 0.1456

1.2 Summarization

- **bert_sum_lora**
  - Test Loss: 5.2265
  - ROUGE-1: 0.2688, ROUGE-2: 0.1269, ROUGE-L: 0.2452
  - BERTScore F1 / Precision / Recall: 0.5623 / 0.5514 / 0.5568
- **bert_sum_qlora**
  - Test Loss: 6.0231
  - ROUGE-1: 0.2549, ROUGE-2: 0.1366, ROUGE-L: 0.1971
  - BERTScore F1 / Precision / Recall: 0.4122 / 0.3744 / 0.4006
- **bert_sum_sft**
  - Test Loss: 5.2538
  - ROUGE-1: 0.2743, ROUGE-2: 0.2371, ROUGE-L: 0.2552
  - BERTScore F1 / Precision / Recall: 0.5342 / 0.5106 / 0.5281

1.3 Question Answering (QA)

- **bert_qa_lora**
  - Test Loss: 0.8653
  - Exact Match (EM): 62.11, F1: 62.58
- **bert_qa_qlora**
  - Test Loss: 0.7714
  - Exact Match (EM): 63.37, F1: 66.75

- **bert_qa_sft**
  - Test Loss: 0.8653
  - Exact Match (EM): 62.11, F1: 62.58

Basic comparison: QLoRA (bert_sa_qlora, bert_qa_qlora) generally outperforms LoRA and SFT variants in both accuracy and loss, with SFT closely matching or slightly exceeding QLoRA for some SA and summarization metrics.

2 GPT-2

2.1 Sentiment Analysis (SA)

- **gpt2_sa_lora**
  - Accuracy: 0.9488, F1: 0.9498
  - Precision / Recall / F1:
    - Negative: 0.95 / 0.95 / 0.95
    - Positive: 0.95 / 0.95 / 0.95
- **gpt2_sa_qlora**
  - Accuracy: 0.9496, F1: 0.9506
  - Precision / Recall / F1:
    - Negative: 0.95 / 0.95 / 0.95
    - Positive: 0.95 / 0.95 / 0.95
- **gpt2_sa_sft**
  - Accuracy: 0.95
  - Precision / Recall / F1:
    - Negative: 0.94 / 0.95 / 0.95
    - Positive: 0.95 / 0.94 / 0.95

2.2 Question Answering (QA)

- **gpt2_qa_lora**
  - EM: 59.31, F1: 69.95
- **gpt2_qa_qlora**
  - EM: 57.81, F1: 68.53
- **gpt2_qa_sft**
  - EM: 58.74, F1: 68.03

2.3 Summarization

- **gpt2_sum_lora**
  - ROUGE-1: 0.1864, ROUGE-2: 0.0431, ROUGE-L: 0.1324, ROUGE-Lsum: 0.1656
- **gpt2_sum_qlora**
  - ROUGE-1: 0.1597, ROUGE-2: 0.0363, ROUGE-L: 0.1212, ROUGE-Lsum: 0.1448
- **gpt2_sum_sft**
  - ROUGE-1: 0.2995, ROUGE-2: 0.1016, ROUGE-L: 0.2086, ROUGE-Lsum: 0.2780

Interpretation: For GPT-2, LoRA and SFT yield similar SA performance; QLoRA does not significantly improve QA metrics. SFT outperforms LoRA and QLoRA in summarization by a substantial margin.

3 T5

3.1 Sentiment Analysis (SA)

- **t5_sa_lora**
  - Accuracy: 0.93

  - Precision / Recall / F1:
    - Negative: 0.92 / 0.94 / 0.93
    - Positive: 0.94 / 0.93 / 0.93

- **t5_sa_qlora**
  - Accuracy: 0.93
  - Precision / Recall / F1:
    - Negative: 0.92 / 0.93 / 0.93
    - Positive: 0.93 / 0.92 / 0.93

- **t5_sa_sft**
  - Accuracy: 0.94
  - Precision / Recall / F1:
    - Negative: 0.93 / 0.94 / 0.94
    - Positive: 0.95 / 0.93 / 0.94

3.2 Question Answering (QA)

- **t5_qa_lora**
  - Exact Match: 71.01, F1: 80.32

- **t5_qa_qlora**
  - Exact Match: 69.21, F1: 79.20

- **t5_qa_sft**
  - Exact Match: 73.40, F1: 82.00

3.3 Summarization

- **t5_sum_lora**
  - Validation ROUGE-1: 25.12, ROUGE-2: 11.95, ROUGE-L: 20.57, ROUGE-Lsum: 23.65

  - Test ROUGE-1: 25.40, ROUGE-2: 12.06, ROUGE-L: 20.84, ROUGE-Lsum: 23.89

- **t5_sum_qlora**
  - Validation ROUGE-1: 25.05, ROUGE-2: 11.95, ROUGE-L: 20.55, ROUGE-Lsum: 23.60
  - Test ROUGE-1: 25.38, ROUGE-2: 12.05, ROUGE-L: 20.81, ROUGE-Lsum: 23.88

- **t5_sum_sft**
  - Validation ROUGE-1: 25.12, ROUGE-2: 11.95, ROUGE-L: 20.57, ROUGE-Lsum: 23.65
  - Test ROUGE-1: 25.40, ROUGE-2: 12.06, ROUGE-L: 20.84, ROUGE-Lsum: 23.89

Interpretation: For T5, SFT yields the highest QA performance, while LoRA and QLoRA achieve nearly identical summarization scores. In SA, SFT slightly outperforms the other adaptation methods.

VII Discussion

The adaptation of three backbone architectures (BERT, GPT-2, and T5) through LoRA, QLoRA, and SFT reveals several consistent patterns. Across all three tasks—sentiment analysis (SA), summarization, and question answering (QA)—the choice of fine-tuning strategy influences both performance and stability. In general, QLoRA often accelerates convergence and reduces loss, while SFT combines the advantages of larger parameter updates with targeted supervision, leading to marginally better or comparable downstream metrics.

1 Sentiment Analysis

For BERT, QLoRA (bert_sa_qlora) achieved a significantly higher accuracy (0.95) than LoRA (0.87), confirming that quantized low-rank adaptations maintain representational capacity while reducing memory overhead. The SFT variant (bert_sa_sft) further

improved accuracy to 0.96, indicating that supervised fine-tuning better aligns the pretrained embeddings with the sentiment classification objective. A similar trend holds for GPT-2: both LoRA and QLoRA reached approximately 0.95 accuracy, with SFT matching at 0.95. In T5, the SFT strategy (0.94) slightly outperformed LoRA/QLoRA (0.93) despite all three variants achieving high precision and recall for both classes. These results suggest that the added supervision from SFT helps capture subtle nuances in sentiment data, especially when the backbone architecture already possesses strong language representations.

2 Summarization

Summarization performance varied more noticeably among adaptation methods. For BERT-based summarization, bert_sum_sft outperformed both LoRA and QLoRA in terms of ROUGE scores (ROUGE-2: 0.2371) and BERTScore (F1: 0.5342), despite having a test loss comparable to LoRA. This indicates that SFT can better leverage pretrained language modeling knowledge for coherent, informative summaries. Interestingly, QLoRA's higher test loss (6.0231) and lower BERTScore (F1: 0.4122) suggest that quantization noise may be more detrimental when generating longer outputs. In the GPT-2 experiments, gpt2_sum_sft achieved a ROUGE-1 of 0.2995—substantially higher than both LoRA (0.1864) and QLoRA (0.1597). This gap underscores that parameter-efficient methods alone may not fully capture abstractive summarization subtleties. For T5, however, all three methods (t5_sum_lora, t5_sum_qlora, t5_sum_sft) yielded nearly identical ROUGE-1 scores around 25.4. This suggests that T5's sequence-to-sequence pretraining is inherently well suited for summarization, making it more robust to the choice of adaptation technique.

3 Question Answering

In QA, T5 consistently led the leaderboard: t5_qa_sft achieved 73.40 EM and 82.00 F1, surpassing t5_qa_lora (71.01 EM, 80.32 F1) and t5_qa_qlora (69.21 EM, 79.20 F1). This demonstrates that supervised fine-tuning on extractive QA datasets most effectively aligns T5's generative capabilities with span extraction. For BERT, bert_qa_qlora outperformed both LoRA and SFT variants (63.37 EM, 66.75 F1), indicating that quantized adaptation retains sufficient expressivity for precise answer extraction. It is noteworthy that bert_qa_lora and bert_qa_sft yielded identical QA metrics, both at 62.11 EM and 62.58

F1, suggesting that LoRA and SFT provide similar fine-grained adjustments for BERT in the QA setting. GPT-2's QA results were generally lower (EM around 58–59, F1 around 68–70) regardless of adaptation method, reflecting that a causal language model is less aligned with span-selection tasks than encoder–decoder or bidirectional architectures.

4 Methodological Implications

Overall, QLoRA strikes a balance between memory efficiency and performance: it matches or slightly underperforms SFT on most classification tasks but often outperforms vanilla LoRA in terms of loss reduction. When model size and GPU memory are limiting factors, QLoRA can be a practical choice: it reduces the precision of adapter weights without substantial sacrifices in task accuracy. However, SFT retains an edge in tasks requiring nuanced generation (summarization, QA) because it allows the model to adjust a broader subset of parameters under direct supervision.

From another perspective, the SFT approach may involve longer training times and larger checkpoints, but it reliably yields strong downstream behavior. In contrast, LoRA and QLoRA can be iterated more quickly when computational resources are constrained. Notably, T5's performance exhibited minimal variance between LoRA, QLoRA, and SFT for summarization, suggesting that the encoder–decoder paradigm can absorb low-rank or quantized updates more gracefully than decoder-only architectures.

5 Limitations and Future Directions

While the current study focuses on offline test metrics, future work could investigate robustness under distribution shifts or in adversarial settings. Additionally, integrating a lightweight calibration phase (e.g., through knowledge distillation or parameter interpolation between LoRA/QLoRA and SFT checkpoints) might further narrow performance gaps. Finally, extending these comparisons to larger backbone variants (e.g., BERT-large, GPT-2-medium, T5-large) would clarify how scaling interacts with adaptation strategies.

In conclusion, all three adaptation methods demonstrate value depending on the task requirements and resource constraints. SFT generally yields the best end-task scores but at higher computational cost, QLoRA offers a competitive trade-off, and LoRA provides a baseline low-rank alternative with minimal overhead.

VIII Conclusions and Future Work

1 Key Findings

Our experiments demonstrated that parameter-efficient fine-tuning methods can be a practical alternative to full fine-tuning in many scenarios. LoRA and QLoRA achieved competitive performance while significantly reducing resource requirements, making them valuable tools for adapting large language models to specific tasks.

The effectiveness of different methods varied across models and tasks, highlighting the importance of choosing the appropriate method for each specific case. The performance of parameter-efficient methods was particularly strong for tasks where the model's pre-trained knowledge was highly relevant, suggesting that these methods can be especially valuable for domain adaptation.

2 Practical Implications

The findings of this study have several practical implications for the development and deployment of NLP systems. Parameter-efficient fine-tuning methods enable the adaptation of large language models to specific tasks with limited computational resources, making advanced NLP capabilities more accessible to researchers and developers.

The ability to fine-tune large models efficiently also has implications for the development of specialized models for specific domains or tasks. By reducing the resource requirements for fine-tuning, these methods enable more rapid experimentation and iteration, potentially leading to faster progress in the field.

3 Limitations and Future Work

While our work provided valuable insights into the effectiveness of different fine-tuning methods, there are several areas for future research. One important direction is the investigation of optimal hyperparameters for LoRA and QLoRA, which could further improve their performance and efficiency.

The extension of these methods to more complex tasks and larger models is also an important direction for future work. As models continue to grow in size and capability, the development of efficient fine-tuning methods will become increasingly important.

IX References

1. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

2. Hugging Face, "T5," [Online]. Available: https://huggingface.co/docs/transformers/en/model_doc/t5

3. Hugging Face, "BERT," [Online]. Available: https://huggingface.co/docs/transformers/en/model_doc/bert

4. Hugging Face, "GPT-2," [Online]. Available: https://huggingface.co/openai-community/gpt2

5. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," OpenAI, 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

6. C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," 2019. [Online]. Available: https://arxiv.org/abs/1910.10683

7. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

8. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

9. A. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," 2023. [Online]. Available: https://arxiv.org/abs/2305.14314

10. Y. Liu, Y. Liu, and J. Lin, "LoRA: Low-Rank Adaptation of Large Language Models," 2024. [Online]. Available: https://arxiv.org/abs/2401.01335