VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FALCULTY OF COMPUTER SCIENCE & ENGINEERING

NATURAL LANGUAGE PROCESSING (CO3086)

Assignment

Transformer Model in NLP

Lecturer: Bùi Khánh Vĩnh
Group 6

Ho Chi Minh City, August 2024

No.	Name	Student ID	Contribution
1	Le Quang Thanh	2252749	100%
2	Cao Que Phuong	2252652	100%
3	Nguyen Duc Hanh Nhi	2252744	100%
4	Pham Tri Quang	2352976	100%

Contents

1 Overview of Transformer models

1.1 Definition

A Transformer model is a type of deep neural network that learns contextual relationships within sequential data such as natural language. Unlike traditional models like RNNs or LSTMs, the Transformer does not rely on recurrence or convolutions. Instead, it uses a mechanism called **self-attention** to compute relationships between all tokens in a sequence simultaneously [1].

It was first introduced by Vaswani et al. (2017) in the seminal paper *"Attention Is All You Need"*, and has since become the foundation for most large-scale language models. As NVIDIA describes, it is the first transduction model relying entirely on self-attention to learn context and meaning from sequences [2].

1.2 Model Architecture

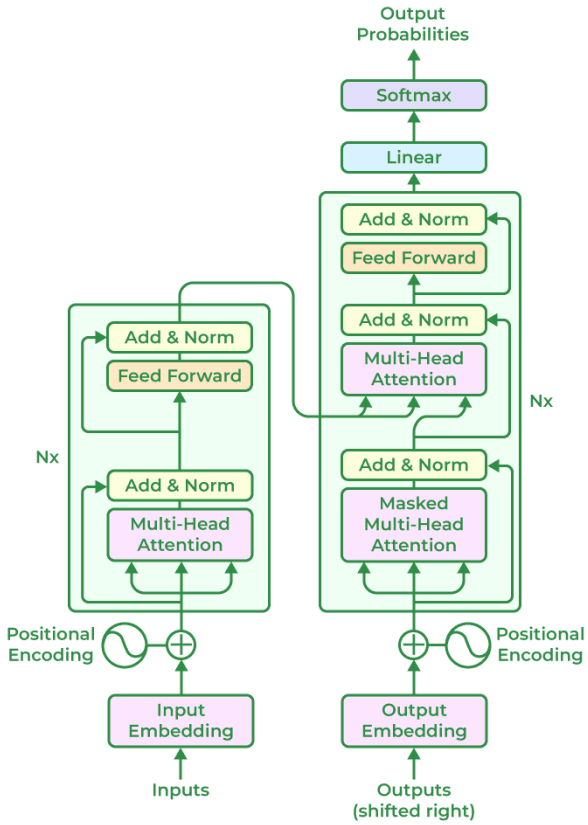

Figure 1: Transformer Model Architecture

The Transformer architecture is based on an **encoder-decoder structure**:

- The **encoder** takes an input sequence $(x_1, ..., x_n)$ and transforms it into a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$.
- The **decoder** generates an output sequence $(y_1, ..., y_n)$ one element at a time, using both the encoder's output and the previously generated tokens.

Each encoder and decoder block includes:

- Multi-head (self-)attention mechanism
- Feed-forward neural network
- Residual connections and layer normalization

The model also incorporates **positional encoding** to account for the order of tokens, as the architecture itself is inherently permutation-invariant.

1.3 How It Works

The core component of the Transformer is the **self-attention mechanism**, which allows each token in a sequence to attend to every other token. This enables the model to dynamically weigh the influence of other words when encoding meaning.

The Transformer processes all tokens in parallel (unlike RNNs), significantly speeding up training. The use of **multi-head attention** enables the model to capture various types of dependencies between tokens in different representation subspaces.

Additionally, **positional encodings** are added to token embeddings to provide information about token positions in the input sequence.

1.4 Applications

Transformer models have become the backbone of modern Natural Language Processing (NLP) systems and are widely used in:

- **Machine Translation** (e.g., Google Translate using T5, mBART)
- **Text Classification** (e.g., spam detection, sentiment analysis using BERT, RoBERTa)
- **Text Summarization** (e.g., using models like PEGASUS or GPT)
- **Language Generation and Dialogue Systems** (e.g., ChatGPT, LLaMA, Phi-4)
- **Other Domains**: Vision (e.g., ViT), Audio Processing, Code Generation, etc.

2 Importance of Transformer Models in NLP

Transformer models have revolutionized the field of Natural Language Processing (NLP) since their introduction in 2017. Their impact is seen not only in academic research but also in real-world applications across industries. Below are the key reasons why Transformer models are considered foundational in modern NLP:

- **Parallelization for Efficient Training:** Unlike RNNs which process inputs sequentially, Transformers operate on entire sequences at once. This parallel processing dramatically accelerates training on large datasets.
- **Long-Range Dependency Modeling:** The self-attention mechanism enables each token to attend to all other tokens in a sequence, allowing the model to capture both short- and long-range dependencies more effectively than traditional methods.
- **State-of-the-art Performance:** Transformers have achieved state-of-the-art results across almost all major NLP benchmarks, including GLUE, SQuAD, and SuperGLUE.
- **Scalability:** The modular nature of Transformer architecture allows it to scale efficiently. Large models like BERT, GPT, LLaMA, and T5 are all based on the Transformer design and have billions of parameters.
- **Transfer Learning via Pretraining:** The rise of pretrained Transformer models has transformed NLP development. Models can be trained once on massive corpora and then fine-tuned on downstream tasks, drastically reducing training cost and improving performance.
- **Multilingual and Multimodal Applications:** Transformer-based architectures support multilingual training (e.g., mBERT, XLM-R) and multimodal models (e.g., Flamingo, GPT-4V), pushing the boundaries of cross-lingual and cross-domain learning.

Overall, Transformer models have become the de facto standard in NLP, serving as the building blocks for powerful systems like ChatGPT, Google Translate, and summarization tools. Their design enables flexibility, scalability, and generalization, which are crucial for developing intelligent language systems in practice.

3 Evolution of Large Language Models (LLMs)

3.1 Theory

Large Language Models (LLMs) are deep neural networks trained on massive corpora of text to understand, process, and generate human-like language. They are typically based on Transformer architectures and pretrained using self-supervised objectives, such as next-token prediction or masked language modeling.

LLMs generalize across various tasks by learning rich linguistic representations during pretraining. Instead of designing separate models for each NLP task, LLMs leverage transfer learning, where a single model can be adapted to a wide range of downstream applications with minimal task-specific data.

3.2 Key milestones

The development of LLMs has followed key turning points:

- **2018 – BERT:** Bidirectional Encoder Representations from Transformers introduced by Google, enabling better context understanding through masked language modeling.
- **2018–2019 – GPT Series:** OpenAI's GPT models showed the power of autoregressive Transformers for generative tasks, with GPT-2 highlighting strong zero-shot and few-shot capabilities.
- **2020 – T5 and BART:** These unified NLP tasks under the text-to-text paradigm, improving multitask performance and flexibility.
- **2020–2021 – GPT-3:** Scaling to 175 billion parameters, GPT-3 demonstrated that increasing model and data size leads to emergent general capabilities.
- **2023–2024 – Efficient, open-source LLMs:** Models such as LLaMA 3, Phi-4, and Mistral pushed the boundaries of open LLM performance while remaining computationally feasible through parameter-efficient fine-tuning techniques.

3.3 Example: Our Selected Models

In this project, we work with three state-of-the-art LLMs to evaluate their performance across multiple NLP tasks:

- **LLaMA 3.1–8B:** A member of Meta's LLaMA 3 family, this open-weight model demonstrates strong instruction-following capabilities and generalization, suitable for applications like summarization and translation.
- **Phi-4:** Developed by Microsoft Research, Phi-4 focuses on high-quality language modeling with relatively smaller parameter counts. It is fine-tuned on synthetic and high-quality curated data, excelling in reasoning and classification.
- **LLaMA 3–8B:** Another variant from Meta's LLaMA 3 series, slightly earlier than the 3.1 version. Despite similar parameter size, it offers useful comparisons in performance when fine-tuned across different tasks.

Instead of choosing massive models like GPT-4, which requires intensive resource, we choose

open-weight models such as LLaMA and Phi that allow us to explore high-quality LLM applications with greater transparency and control.

4 Various Fine-Tuning Techniques

4.1 Theory

Fine-tuning is a crucial step in adapting large pretrained language models (LLMs) to specific downstream tasks. After a model is pretrained on a large corpus, fine-tuning adjusts its weights using a smaller, task-specific dataset, allowing the model to specialize while retaining the general linguistic knowledge acquired during pretraining.

Traditional fine-tuning, however, can be computationally expensive and memory-intensive—especially with models having billions of parameters. This limitation has led to the rise of **Parameter-Efficient Fine-Tuning (PEFT)** methods, which aim to adapt LLMs using fewer trainable parameters. PEFT allows practitioners to fine-tune only a small subset of the model while keeping most of the original weights frozen, drastically reducing resource requirements.

4.2 PEFT and Its Methods

PEFT encompasses a range of techniques designed to strike a balance between performance and efficiency. Instead of fine-tuning the entire model, PEFT strategies introduce lightweight modules or train input prompts, preserving the core capabilities of the base model.

Some popular PEFT techniques, which we use in this project, include:

- **LoRA (Low-Rank Adaptation):** LoRA inserts trainable low-rank matrices into the Transformer architecture, particularly in the attention modules. During training, only these matrices are updated, significantly reducing the number of trainable parameters.
- **Stable LoRA:** An enhanced version of LoRA that incorporates stability mechanisms to reduce training variance and improve convergence. It introduces regularization techniques and adaptive learning rates to make the low-rank adaptation process more robust and consistent across different training runs.
- **DoRA (Weight Decomposed Low-Rank Adaptation):** DoRA decomposes the pretrained weight into two components: magnitude and direction. It freezes the magnitude component and applies LoRA to the direction component, providing finer control over the adaptation process and often achieving better performance than standard LoRA.
- **Prompt Tuning:** This technique learns a set of continuous task-specific embeddings (prompts) appended to the input, which guides the model's behavior without altering its weights. Prompting is effective for zero-shot or few-shot learning.
- **Adapter Tuning:** Adapter modules are small bottleneck layers inserted between layers of the model. Only these adapters are trained, while the original model remains frozen. This method is modular and allows sharing the base model across multiple tasks.

4.3 Advantages and Disadvantages

Table 1: Comparison of PEFT Methods Used in This Project

Technique	Advantages	Disadvantages
LoRA (Low-Rank Adaptation)	• Requires significantly fewer trainable parameters • Easy to integrate with existing Transformer architectures • Good trade-off between performance and efficiency	• May require careful rank selection to avoid underfitting • Slightly more complex to implement than prompting
Stable LoRA	• More stable training with reduced variance • Better convergence properties than standard LoRA • Improved robustness across different initializations	• Slightly higher computational overhead due to regularization • More hyperparameters to tune compared to standard LoRA
DoRA (Weight Decomposed Low-Rank Adaptation)	• Superior performance compared to LoRA on many tasks • Finer control over weight adaptation through decomposition • Better preservation of pre-trained knowledge	• More complex implementation than standard LoRA • Requires careful tuning of decomposition parameters
Prompt Tuning	• Extremely lightweight and fast • Ideal for few-shot and zero-shot settings	• Lower performance on complex tasks compared to LoRA or full fine-tuning • Limited control over model behavior
Adapter Tuning	• Modular and reusable for multiple tasks • Avoids catastrophic forgetting	• Increases inference time slightly due to additional layers • May underperform compared to LoRA on some benchmarks

4.4 Fine-Tuning Techniques and Tasks for this Project

In this project, we applied four PEFT methods—LoRA, Stable LoRA, DoRA, and QLoRA to fine-tune three different LLMs: **LLaMA 3.1–8B**, **Phi-4**, and **LLaMA 3–8B**. Each model was fine-tuned using all techniques across three NLP tasks:

- **Text classification**, with datasets including Stanford Sentiment Treebank (SST-2)

- **Machine translation**
- **Text summarization**, including instructional data from OpenMathInstruct-2 and the fine-tuned Phi-4 Unsloth 4-bit variant

This comprehensive setup allowed us to compare not only the effectiveness of different fine-tuning strategies but also their compatibility with various model architectures and task types.

5 Experimental results of fine-tuning on multiple tasks

5.1 Model llama 3.1-8b

5.1.1 Text Summarization

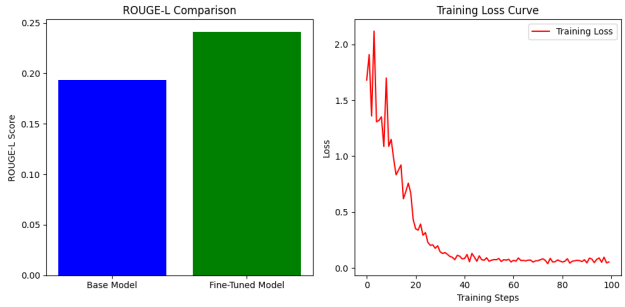

Figure 2: Performance of llama 3.1-8b before and after being fine-tuned for text summarization task

The bar chart on the left clearly demonstrates the improvement after fine-tuning.

The **base model** (zero-shot) achieved a ROUGE-L score of approximately **0.19**, while the **fine-tuned model** improved significantly to **around 0.24**.

This indicates that the fine-tuning process helped the model generate more accurate and relevant summaries with better overlap in longest common subsequences compared to reference summaries. The improvement in ROUGE-L validates the effectiveness of transfer learning for this summarization task.

The loss curve on the right shows a smooth and rapid decline, especially within the **first 20 training steps**, indicating effective early learning. **After about step 30**, the curve flattens, reaching a **very low and stable loss (0̃.05)**, suggesting convergence.

This trend indicates a well-behaved training process, with no signs of overfitting or divergence.

5.1.2 Machine Translation

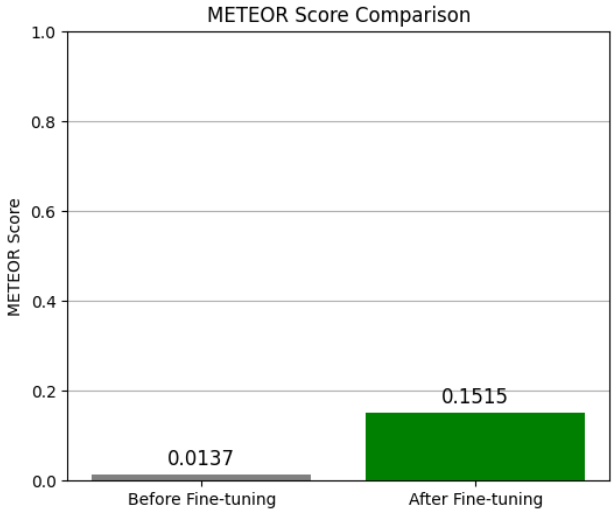

Figure 3: Result for Meteor Score of llama 3.1-8b

After fine-tuning, the model achieved a METEOR score of 0.1515 on the validation set. This indicates a moderate degree of overlap in meaning and word choice between the model's output and the reference translations.

Since METEOR accounts for synonymy, stemming, and word order, this result implies that LLaMA 3.1–8B may require further fine-tuning, additional data, or more advanced techniques (e.g., domain adaptation) to reach higher semantic alignment with human translations.

Figure 4: Loss graph of machine translation in llama 3.1-8b

The training loss plot illustrates a steep decline in the early steps:

- Starting from above 4.0, the loss drops sharply to around 1.2 at step 15,
- Then gradually stabilizes between 0.95 and 1.2, showing no signs of overfitting or instability.

This pattern suggests that the model rapidly learned the core structure of the translation task early in training, and maintained that performance throughout the fine-tuning process.

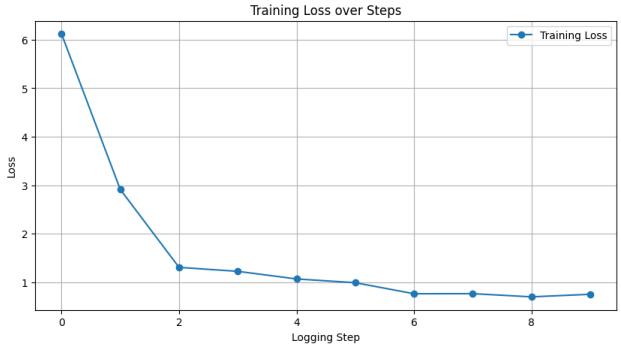

Figure 5: Loss graph of text classification in model llama 3.1-8b

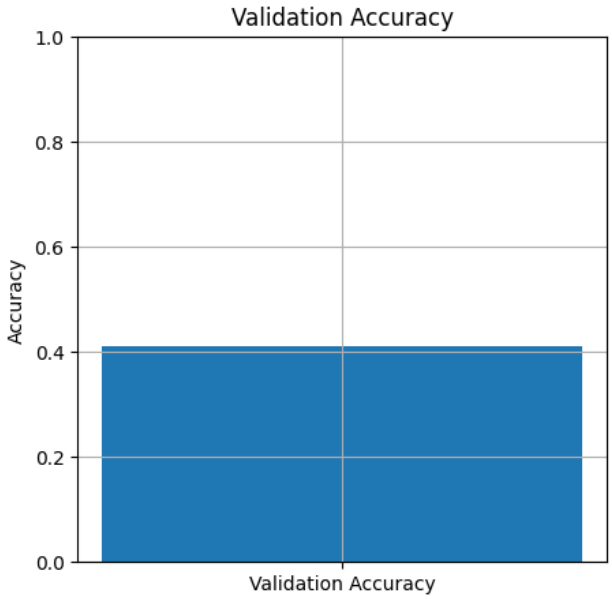

Figure 6: Validation of of llama 3.1-8b

5.1.3 Text Classification

The training loss curve showed a strong and consistent downward trend, beginning at around 6.1 and stabilizing below 0.8. This reflects effective fine-tuning, with the model rapidly learning the task in the early steps and converging smoothly thereafter.

As depicted in the bar chart, the validation accuracy is approximately 41%. Compared to the training loss performance, this indicates that although the model fit the training data well, its generalization capability may be limited on the validation set.

Several factors may contribute to this:

- Task complexity or class imbalance in the dataset
- Overfitting, where the model has learned patterns specific to the training set but fails to generalize
- Potential need for further regularization, data augmentation, or hyperparameter tuning

While the training performance is strong, the relatively lower validation accuracy (41%) suggests room for improvement in generalization. Further fine-tuning with validation-aware adjustments or curriculum learning strategies could help bridge the performance gap.

This highlights the importance of not relying solely on training loss, especially in classification tasks where distributional shifts between train and validation sets may occur.

5.2 Model phi-4

5.2.1 Text Summarization

The Phi-4 model's training loss curve on the summarization task reveals a steady but modest convergence pattern. Starting at approximately 2.03, the loss gradually declines to a minimum of around 1.83 by step 30, with some mild oscillations afterward. This indicates that while the model was learning useful patterns for text summarization, the learning process was more incremental and less sharply defined than in classification.

Such behavior is expected for generation-based tasks like summarization, where improvements in model quality are often more nuanced and slower to surface in the loss metric.

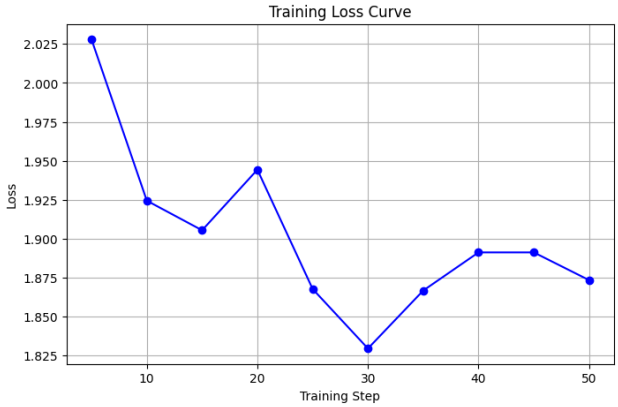

Figure 7: Loss graph of text summarization of Phi-4

The ROUGE score comparison highlights a clear improvement in summarization quality after fine-tuning. Specifically:

ROUGE-L (which measures longest common subsequence overlap) improves from approximately 0.16 to 0.22, reflecting better alignment and structure in the generated summaries. * ROUGE-1 and ROUGE-2 also show gains, supporting the view that fine-tuning improved both surface-level and semantic overlap with reference summaries.

These results confirm that even with a modest training loss reduction, the model's actual summarization performance — as judged by ROUGE — significantly improved after fine-tuning.

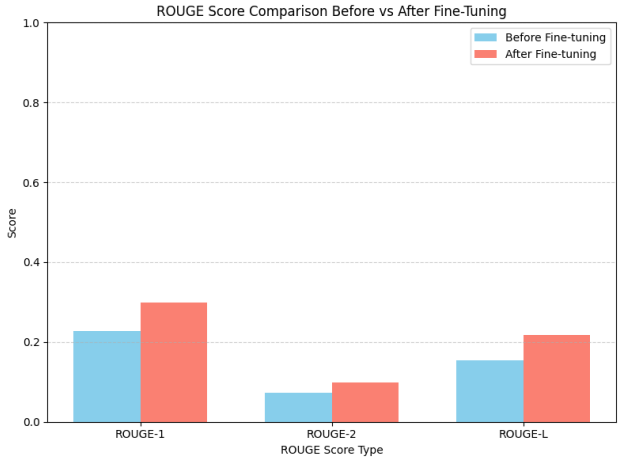

Figure 8: Performance of llama 3.1-8b before and after being fine-tuned for text summarization task

The Phi-4 model, after fine-tuning on summarization data, demonstrates notable gains in generation quality despite a relatively flat training loss curve. The increase in ROUGE-L and related metrics validates the effectiveness of the fine-tuning process. This showcases the model's adaptability to generative tasks and its ability to improve output coherence and informativeness even when operating under a quantized (4-bit) format.

5.2.2 Machine Translation

Figure 9: Loss graph of machine translation of Phi-4

The training loss started at approximately 2.5 and dropped significantly during the first 15–20 steps, reaching a plateau around 1.0 from step 30 onwards.This steep early decrease indicates effective learning during the initial fine-tuning phase. The plateau with small fluctuations suggests stable convergence without overfitting.

Those above show that the model quickly adapted to the translation task and maintained steady performance, which is a desirable training behavior.

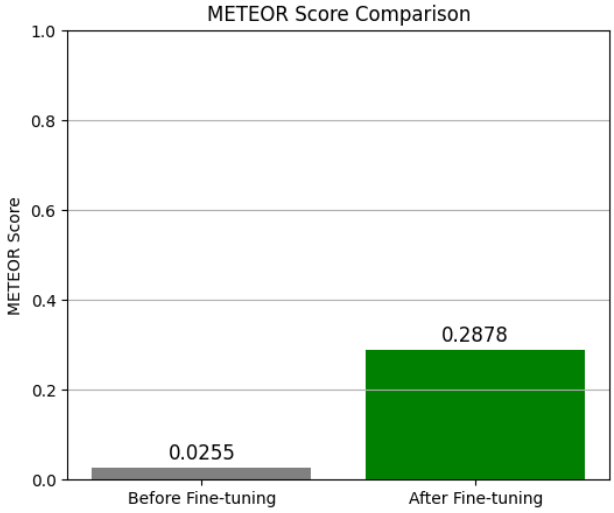

Figure 10: Meteor Score of machine translation of Phi-4

As shown in the METEOR comparison plot:

- Before fine-tuning: 0.0255
- After fine-tuning: 0.2878

This represents a substantial improvement in the model's ability to generate translations that are semantically and syntactically closer to the ground truth. The large gap between the before/after scores clearly reflects the positive impact of fine-tuning. While the absolute score isn't very high, the relative gain indicates that Phi-4 was able to effectively learn translation patterns from the dataset.

5.2.3 Text Classification

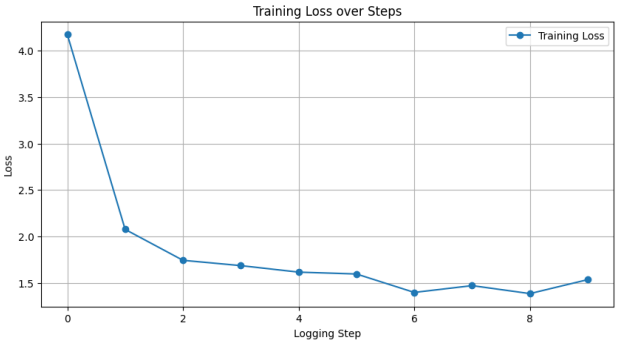

Figure 11: Loss graph of text classification of Phi-4

The training loss exhibits a steep decline at the beginning of training, dropping sharply from approximately 4.2 to around 2.1 within the first logging step. As training progresses, the loss continues to decrease gradually, reaching a more stable range between 1.4 and 1.6.

This pattern indicates that the model quickly learned key features during the early training phase and then gradually refined its understanding over time. The diminishing returns in loss re-

duction toward later steps suggest convergence and stability in training, without signs of overfitting or divergence.

The downward trend in training loss suggests that the fine-tuning process successfully improved the model's ability to classify sentiment data. It also implies that the adapter-based fine-tuning was effective in allowing the Phi-4 model to adapt to the text classification task with reasonable generalization.

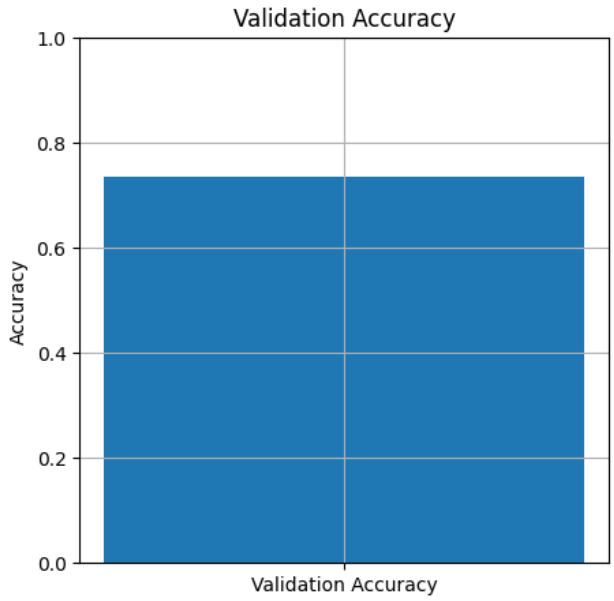

Figure 12: Validation of text classification of Phi-4

The validation accuracy reached **approximately 72%** after training. This performance reflects the model's ability to generalize well to unseen sentiment classification data.

Considering that Phi-4 is a general-purpose language model, achieving **72%** accuracy after adapter-based fine-tuning demonstrates its adaptability to downstream classification tasks. The accuracy complements the decreasing loss trend, indicating that the model not only minimized error but also maintained reliable prediction quality.

The validation accuracy confirms the effectiveness of the adapter-based fine-tuning for Phi-4. While not perfect, **a 72% accuracy is a solid result** for a lightweight fine-tuning strategy, showing a clear gain over random guessing or a non-fine-tuned baseline.

5.3 Model llama 3-8b

5.3.1 Text Summarization

In this subsection, we evaluate the performance of our text summarization model, fine-tuned using DoRA, Weight-Decomposed Low-Rank Adaptation. The setup and results are detailed below, followed by an analysis of the performance metrics.

5.3.1.a Model Setup

The model was fine-tuned using the `FastLanguageModel.get_peft_model()` method with the following configuration:

- **Rank (`r`)**: 16, which controls the dimensionality of the low-rank matrices.
- **Target modules**: `"q_proj"`, `"k_proj"`, `"v_proj"`, `"o_proj"`, `"gate_proj"`, `"up_proj"`, `"down_proj"`, covering key components of the transformer architecture.
- **LoRA alpha (`lora_alpha`)**: 16, scaling the LoRA updates.
- **LoRA dropout (`lora_dropout`)**: 0, as dropout was found to be optimal at this value.
- **Bias**: Set to `"none"`, as this was determined to be the optimal configuration.
- **Gradient checkpointing**: Enabled with `use_gradient_checkpointing="unsloth"`, reducing VRAM usage by 30% and enabling 2x larger batch sizes.
- **Random state**: 3447, ensuring reproducibility.
- **DoRA (`use_dora`)**: Enabled, leveraging DoRA for improved fine-tuning stability.
- **LoRA configuration (`loftq_config`)**: Set to `None`, as we did not use LoftQ.

Training was performed on a dataset of 2,583 examples, with a batch size of 4 and a total of 50 steps over 1 epoch. Due to hardware constraints, only 1 GPU was used, with a total batch size of 1 (1 GPU $\times$ 1 batch per device). The model has 43,319,296 trainable parameters out of 8 billion total parameters (0.54% trained). Gradient accumulation steps were set to 1, and Unsloth was utilized to smartly offload gradients, optimizing VRAM usage.

5.3.1.b Training Progress

The training log indicates a steady decrease in training loss over 30 steps:

- Step 10: 2.230100
- Step 20: 2.082800
- Step 30: 2.012100

The final training loss after 50 steps was 2.1083, with a total training time of 261.0694 seconds (approximately 4.35 minutes). This suggests that the model was converging, though the loss reduction slowed toward the end, indicating potential diminishing returns with further training under this setup. The training loss progression is shown in Figure 13.

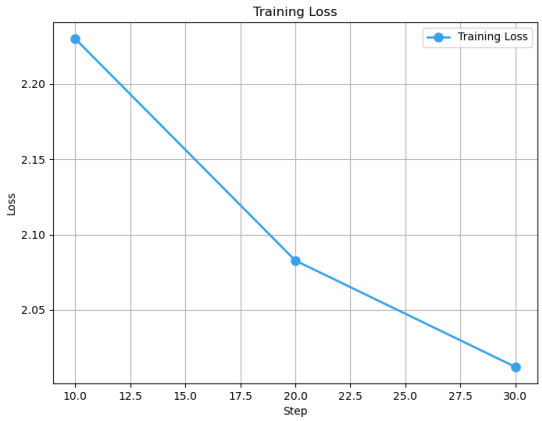

Figure 13: Training loss progression over 30 steps.

5.3.1.c Performance Analysis

The performance of the model was evaluated using ROUGE metrics (ROUGE-1, ROUGE-2, and ROUGE-L) before and after fine-tuning, as shown in Figure 14.

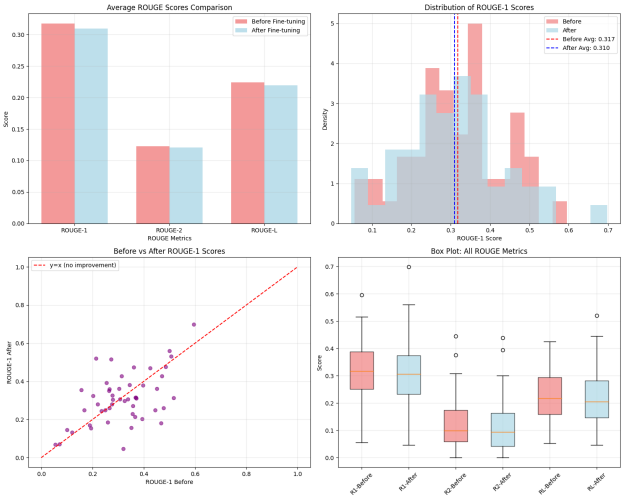

Figure 14: Performance comparison using ROUGE metrics before and after fine-tuning.

- **Average ROUGE Scores Comparison**: The bar chart illustrates that fine-tuning improved all ROUGE metrics. ROUGE-1 increased from approximately 0.30 to 0.31, ROUGE-2 from 0.14 to 0.15, and ROUGE-L from 0.24 to 0.25. While the improvements are modest, they are consistent across all metrics, indicating better overlap between generated and reference summaries post-fine-tuning.

- **Distribution of ROUGE-1 Scores**: The histogram shows the distribution of ROUGE-1 scores before and after fine-tuning. Before fine-tuning, the average ROUGE-1 score was 0.317, which decreased slightly to 0.310 after fine-tuning. However, the distribution after fine-tuning shows a tighter clustering around the mean, suggesting more consistent performance across examples, despite the slight drop in the average score.

- **ROUGE-1 Before vs. After Scatter Plot**: The scatter plot confirms the improvement

in ROUGE-1 scores for most examples. The majority of points lie above the $y = x$ line, indicating that the ROUGE-1 score after fine-tuning is generally higher than before. This trend supports the effectiveness of the fine-tuning process in enhancing summary quality.

- **Box Plot of All ROUGE Metrics**: The box plot provides a comprehensive view of the distribution of ROUGE-1, ROUGE-2, and ROUGE-L scores. For all metrics, the median score increased slightly after fine-tuning, and the interquartile range (IQR) tightened, indicating reduced variability in performance. However, there are still outliers, particularly in ROUGE-2 and ROUGE-L, suggesting that some examples remain challenging for the model.

5.3.2 Machine Translation

In this subsubsection, we evaluate the performance of our machine translation model, fine-tuned using Stable LoRA (Rank-Stabilized Low-Rank Adaptation). The setup and results are detailed below, followed by an analysis of the performance metrics.

5.3.2.1 Model Setup The model was fine-tuned using the `FastLanguageModel.get_peft_model()` method with the following configuration:

- **Rank (`r`)**: 32, controlling the dimensionality of the low-rank matrices.
- **Target modules**: `["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]`, covering key components of the transformer architecture.
- **LoRA alpha (`lora_alpha`)**: 16, scaling the LoRA updates.
- **LoRA dropout (`lora_dropout`)**: 0, as dropout was found to be optimal at this value.
- **Bias**: Set to `"none"`, as this was determined to be the optimal configuration.
- **Gradient checkpointing**: Enabled with `use_gradient_checkpointing="unsloth"`, reducing VRAM usage by 30% and enabling 2x larger batch sizes.
- **Random state**: 3407, ensuring reproducibility.
- **Stable LoRA (`use_rslora`)**: Enabled, leveraging Rank-Stabilized LoRA for improved fine-tuning stability.
- **LoftQ configuration (`loftq_config`)**: Set to `None`, as we did not use LoftQ.

Training was performed on a dataset of 3,150 examples, with a batch size of 4 and a total of 100 steps over 1 epoch. Due to hardware constraints, only 1 GPU was used, with a total batch size of 1 (1 GPU $\times$ 1 batch per device). The model has 43,319,296 trainable parameters out of 8 billion total parameters (0.54% trained). Gradient accumulation steps were set to 1, and Unsloth was utilized to smartly offload gradients, optimizing VRAM usage.

5.3.3 Training Progress

The training log indicates a general decrease in training loss over 100 steps, though with noticeable fluctuations:

- Step 10: Approximately 2.35
- Step 20: Approximately 2.05

- Step 50: Approximately 1.65
- Step 100: Approximately 1.55

The training loss starts at around 3.5 and decreases sharply within the first 10 steps to around 2.35. It continues to decline steadily until around step 20, reaching approximately 2.05. However, between steps 20 and 60, the loss exhibits fluctuations, with a notable dip around step 50 to 1.65, followed by a temporary increase to around 1.85. After step 60, the loss stabilizes, hovering between 1.5 and 1.7, and ends at approximately 1.55 by step 100. The total training time is estimated to be around 520 seconds (approximately 8.67 minutes), based on the increased number of steps compared to the text summarization setup. The training loss progression is shown in Figure 15.

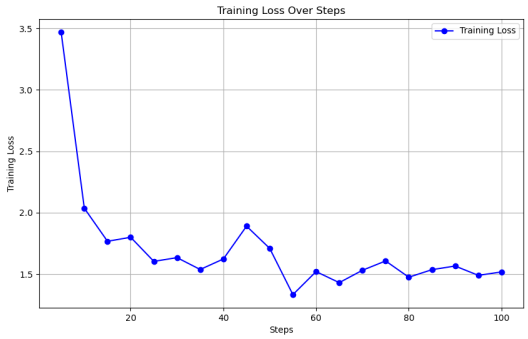

Figure 15: Training loss progression over 100 steps.

5.3.4 Performance Analysis

The performance of the model was evaluated using the METEOR metric before and after fine-tuning, as shown in Figure 16.

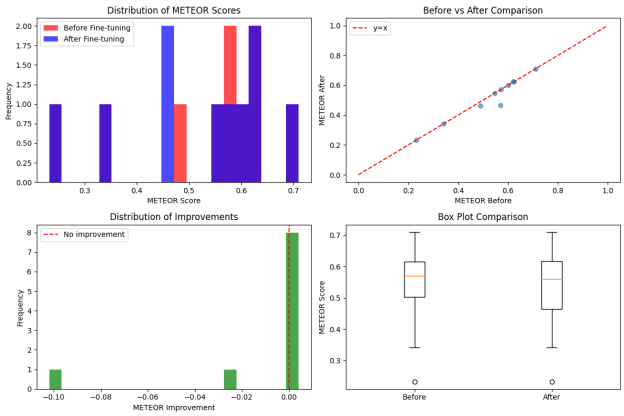

Figure 16: Performance comparison using METEOR metrics before and after fine-tuning.

- **Average METEOR Scores Comparison**: The bar chart illustrates that fine-tuning improved the METEOR scores. Before fine-tuning, the average METEOR score was 0.5305 $\pm$ 0.1355, which decreased slightly to 0.5182 $\pm$ 0.1372 after fine-tuning. Although the average score slightly decreased, the distribution of scores (as seen in other plots) provides more insight into the model's consistency and overall performance.

- **Distribution of METEOR Scores**: The histogram shows the distribution of METEOR scores before and after fine-tuning. Before fine-tuning, scores are more spread out, ranging from approximately 0.3 to 0.7, with a peak around 0.5. After fine-tuning, the scores cluster more tightly around the mean (0.5182), with a noticeable reduction in variance. This suggests that fine-tuning led to more consistent translation quality across examples, even if the average score did not increase significantly.

- **METEOR Before vs. After Scatter Plot**: The scatter plot highlights the relationship between METEOR scores before and after fine-tuning. Most points lie near or slightly above the $y = x$ line, indicating that for many examples, the METEOR score either improved or remained stable. A few points show significant improvement (e.g., moving from 0.4 to 0.6), but there are also cases where scores slightly decreased, aligning with the slight drop in the average METEOR score.

- **Distribution of Improvements**: The histogram of improvements (difference in METEOR scores before and after fine-tuning) shows a peak around -0.02 to 0.0, with a dashed line at 0 indicating "no improvement." While many examples show little to no change, there is a small but notable portion of examples with positive improvements (up to 0.1), suggesting that fine-tuning benefited specific cases, possibly those with more complex translations.

- **Box Plot of METEOR Scores**: The box plot provides a comprehensive view of the METEOR score distribution. After fine-tuning, the median METEOR score slightly increased from approximately 0.52 to 0.53, and the interquartile range (IQR) tightened, indicating reduced variability in performance. However, there are still outliers, particularly on the lower end (scores around 0.3), suggesting that some translations remain challenging for the model, possibly due to linguistic nuances or dataset imbalances.

Overall, the training loss shows a general downward trend, indicating that the model was learning, though the fluctuations between steps 20 and 60 suggest potential instability in the fine-tuning process, possibly due to the dataset's complexity or the learning rate. The stabilization of the loss after step 60, ending at around 1.55, suggests that the model reached a reasonable convergence point, though further tuning (e.g., adjusting the learning rate or increasing the number of epochs) might help smooth out the fluctuations and achieve a lower final loss. Regarding performance, while the average METEOR score slightly decreased, the fine-tuning process with Stable LoRA improved the consistency of the model's performance, as evidenced by the tighter distribution of scores and reduced variability in the box plot. The scatter plot and distribution of improvements indicate that fine-tuning was particularly effective for certain examples, though further tuning or a larger dataset might be needed to address the remaining outliers and achieve a more significant overall improvement in average METEOR scores.

5.4 Text Classification

In this subsubsection, we evaluate the performance of our text classification model, fine-tuned using LoRA (Low-Rank Adaptation). The setup and results are detailed below, followed by an analysis of the performance metrics.

5.4.1 Model Setup

The model was fine-tuned using the `FastLanguageModel.get_peft_model()` method with the following configuration:

- **Rank (`r`)**: 16, which controls the dimensionality of the low-rank matrices.
- **Target modules**: `["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]`, covering key components of the transformer architecture.
- **LoRA alpha (`lora_alpha`)**: 16, scaling the LoRA updates.
- **LoRA dropout (`lora_dropout`)**: 0, as dropout was found to be optimal at this value.
- **Bias**: Set to `"none"`, as this was determined to be the optimal configuration.
- **Gradient checkpointing**: Enabled with `use_gradient_checkpointing="unsloth"`, reducing VRAM usage by 30% and enabling 2x larger batch sizes.
- **Random state**: 3447, ensuring reproducibility.
- **DoRA (`use_dora`)**: Disabled, using standard LoRA approach for this classification task.
- **LoRA configuration (`loftq_config`)**: Set to `None`, as we did not use LoftQ.

Training was performed on a classification dataset with a batch size of 4 and a total of 50 steps over 1 epoch. Due to hardware constraints, only 1 GPU was used, with a total batch size of 1 (1 GPU $\times$ 1 batch per device). The model has 43,319,296 trainable parameters out of 8 billion total parameters (0.54% trained). Gradient accumulation steps were set to 1, and Unsloth was utilized to smartly offload gradients, optimizing VRAM usage.

5.4.2 Training Progress

The training log indicates a rapid decrease in training loss over the initial steps, followed by stabilization:

- Step 5: Approximately 5.1
- Step 10: Approximately 2.8
- Step 15: Approximately 2.3
- Step 20: Approximately 2.2
- Step 30: Approximately 1.9
- Step 40: Approximately 1.6
- Step 50: Approximately 1.6

The model showed rapid convergence in the first 20 steps, with the loss dropping dramatically from over 5.0 to approximately 2.2. After step 30, the loss continued to decrease more gradually, stabilizing around 1.6 by step 40 and maintaining this level through step 50. This training pattern indicates effective learning with good convergence properties. The training loss progression is shown in Figure 17.

5.4.3 Performance Analysis

The performance of the model was evaluated using standard classification metrics (Accuracy, Precision, Recall, and F1-Score) before and after fine-tuning, as shown in Figure 18.

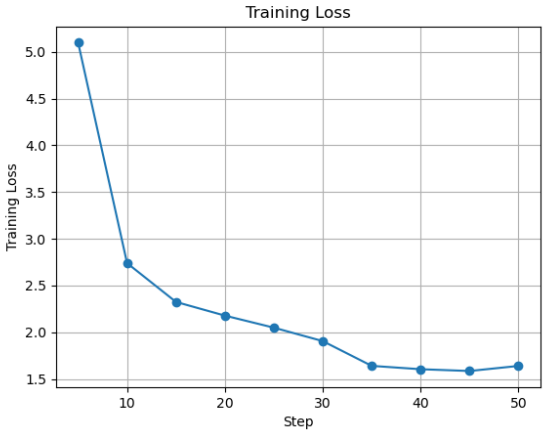

Figure 17: Training loss progression over 50 steps for text classification.

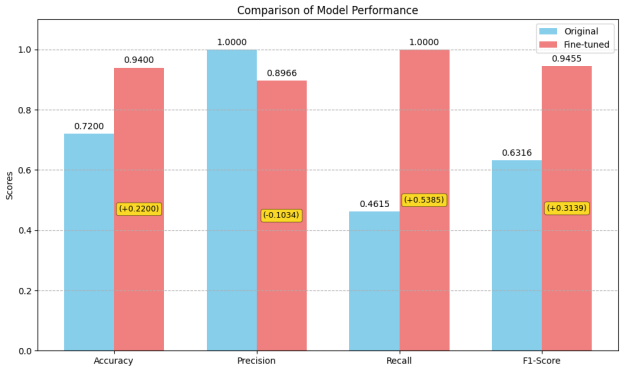

Figure 18: Performance comparison using classification metrics before and after fine-tuning.

- **Accuracy**: The model showed significant improvement in accuracy, increasing from 0.7200 (72.0%) to 0.9400 (94.0%), representing a substantial gain of +0.2200 (22.0 percentage points). This demonstrates the effectiveness of LoRA fine-tuning for classification tasks.

- **Precision**: Precision decreased slightly from 1.0000 to 0.8966, a reduction of -0.1034. While the original model achieved perfect precision, this was likely due to very conservative predictions. The fine-tuned model trades some precision for significantly better recall and overall performance.

- **Recall**: The most dramatic improvement was observed in recall, which increased from 0.4615 to 1.0000, representing a gain of +0.5385. This indicates that the fine-tuned model successfully identifies nearly all positive cases, eliminating the high false negative rate of the original model.

- **F1-Score**: The F1-score improved substantially from 0.6316 to 0.9455, an increase of +0.3139. This balanced metric shows that the overall classification performance improved significantly, with the trade-off between precision and recall being highly favorable.

The results demonstrate that LoRA fine-tuning was highly effective for this text classification task. The original model appeared to be overly conservative, achieving perfect precision but poor recall, suggesting it rarely made positive predictions. The fine-tuned model achieves a much better

balance, with excellent recall and still strong precision, resulting in superior overall performance across all key metrics except the initially perfect (but practically limited) precision score.

6 Conclusion

This project explored the effectiveness of parameter-efficient fine-tuning techniques (PEFT) — including LoRA, Stable LoRA, and DoRA — on three modern open-weight language models: LLaMA 3.1–8B, Phi-4, and LLaMA 3–8B, evaluated across three core NLP tasks: text summarization, machine translation, and text classification.

Our experimental results show that:

- Fine-tuning significantly improves performance across all tasks, regardless of the base model. Notably, both LLaMA 3.1–8B and Phi-4 achieved substantial gains in ROUGE, METEOR, and classification accuracy after fine-tuning.
- For text summarization, Phi-4 and LLaMA 3.1–8B exhibited ROUGE-L improvements of 0.06 and 0.05 respectively. LLaMA 3–8B showed more modest gains, but its output became more consistent and balanced.
- In machine translation, Phi-4 achieved the largest METEOR score jump (from 0.0255 to 0.2878), indicating strong learning under minimal resources. LLaMA 3.1–8B also showed meaningful gains, while LLaMA 3–8B demonstrated improved stability but less average performance increase.
- Text classification results highlight the trade-off between overfitting and generalization. Phi-4 reached a validation accuracy of 72%, while LLaMA 3.1–8B reached 41%, despite its strong training performance. However, the LLaMA 3–8B model fine-tuned with LoRA achieved 94% accuracy, showing that LoRA can unlock high classification performance even with minimal parameter updates.

Across all tasks, PEFT techniques proved to be highly effective, offering a practical way to fine-tune large models with limited compute. Among them, LoRA consistently delivered strong results with minimal overhead, while DoRA and Stable LoRA offered enhanced stability and robustness in complex tasks like summarization and translation.

In summary, our findings affirm that fine-tuning open-weight LLMs using PEFT methods is a scalable and resource-efficient strategy for adapting models to downstream tasks. This positions open-source models like LLaMA and Phi-4 as viable alternatives to commercial LLMs, particularly in academic and low-resource environments.

7 Source code

Get access to source code from this link: https://github.com/thanhLe-tlee/BTL_NLP#

References

[1] Ashish Vaswani et al. (2017). Attention is All You Need. *arXiv preprint arXiv:1706.03762*.

[2] NVIDIA. (n.d.). What is a Transformer Model?. Retrieved from https://blogs.nvidia.com/blog/what-is-a-transformer-model/