LD COLLEGE OF ENGINEERING AHMEDABAD

A Report On

Parameter-Efficient Fine Tuning (PEFT) Techniques

Under the subject of

Machine Learning Using Python

B.E. Semester-V

(AI-ML BRANCH)

Submitted by

SR . NO	**Name Of The Students**	**Enrollment No.**
1	Makadiya Deep Dineshbhai	230280152034
2	Dholakiya Mayur Ashokbhai	230280152018
3	Balar Maulik Ghanshyambhai	230280152005
4	Gajera Kuldeep Vipulbhai	230280152020

Guided By
Prof. MAITRIK SHAH

1. Abstract

In recent years, fine-tuning large pre-trained language models such as BERT has become a standard approach for various natural language understanding tasks. However, traditional full fine-tuning involves updating all model parameters, resulting in significant computational costs, high memory usage, and longer training times. To overcome these limitations, Parameter-Efficient Fine-Tuning (PEFT) techniques like LoRA (Low-Rank Adaptation) QLora (Quantized Low-Rank Adaptation) and Adapters have been introduced. These methods enable efficient model adaptation by updating only a small subset of parameters, thereby reducing training overhead without compromising performance.

This project presents a comparative analysis of LoRA, AdapterFusion, QLora and Full Fine-Tuning using BERT-based models on benchmark text classification datasets such as SST-2. The experiments are implemented using the HuggingFace Transformers and PEFT libraries, with performance tracking and visualization through Weights & Biases (wandb). Each method is evaluated on key performance metrics, including accuracy, training time, and GPU utilization.

The outcomes are illustrated through bar charts comparing accuracy, training time, and parameter count, highlighting the trade-offs among the different fine-tuning approaches. The results indicate that PEFT techniques such as LoRA, QLora and AdapterFusion achieve comparable accuracy to full fine-tuning while significantly reducing resource consumption, making them highly suitable for deployment in resource-constrained environments.

2. Introduction

The rapid growth of large-scale pre-trained language models such as BERT, GPT, and RoBERTa has revolutionized the field of Natural Language Processing (NLP). These models achieve state-of-the-art performance across a wide range of tasks, including text classification, sentiment analysis, and question answering. However, the traditional full fine-tuning approach—where all model parameters are updated during training—poses significant challenges in terms of computational cost, memory requirements, and scalability, especially when dealing with billions of parameters.

To address these limitations, Parameter-Efficient Fine-Tuning (PEFT) techniques have gained prominence. PEFT methods such as LoRA (Low-Rank Adaptation), AdapterFusion, and QLoRA (Quantized LoRA) enable efficient model adaptation by updating only a small subset of parameters or introducing lightweight trainable modules. These approaches drastically reduce the number of trainable parameters while maintaining competitive model performance.

LoRA works by injecting low-rank decomposition matrices into the transformer architecture, allowing efficient weight adaptation. AdapterFusion builds upon adapter layers by combining multiple pre-trained adapters for improved task performance. Meanwhile, QLoRA extends LoRA by performing fine-tuning on quantized models (e.g., 4-bit precision), achieving substantial reductions in GPU memory usage without sacrificing accuracy.

In this project, BERT-based models are fine-tuned on benchmark datasets such as SST-2 and TREC-6 using Full Fine-Tuning, LoRA, AdapterFusion, and QLoRA. The experiments are implemented using the HuggingFace Transformers and PEFT libraries, with experiment tracking and performance visualization through Weights & Biases (wandb). Each method is evaluated based on accuracy, training time, and GPU utilization.

This comparative study aims to analyze the trade-offs between efficiency and performance among different fine-tuning approaches. The results demonstrate how PEFT methods particularly QLoRA enable scalable and resource-efficient fine-tuning of large models while achieving near state-of-the-art performance.

3. Related Work / Literature Survey

The evolution of fine-tuning strategies for large language models has been a central focus of recent NLP research, as traditional full fine-tuning methods have become computationally expensive with the increasing size of pre-trained models such as BERT, GPT-3, and T5. This section reviews key developments related to Parameter-Efficient Fine-Tuning (PEFT) methods, including LoRA, AdapterFusion, and QLoRA, which form the foundation of this project.

Early work on transfer learning in NLP was popularized by Devlin et al. (2018) with the introduction of BERT (Bidirectional Encoder Representations from Transformers), demonstrating that pre-trained models could be fine-tuned on downstream tasks with high performance. However, full fine-tuning of large models proved resource-intensive, prompting research into more efficient alternatives.

Adapters, introduced by Houlsby et al. (2019), were among the first PEFT methods. Adapters add small trainable layers between transformer blocks while keeping the original model parameters frozen. This significantly reduces training cost and allows for modular transfer across multiple tasks.

AdapterFusion (Pfeiffer et al., 2021) extended this concept by combining multiple pre-trained adapters, enabling knowledge integration from different tasks without retraining the entire model.

LoRA (Low-Rank Adaptation), proposed by Hu et al. (2021), further improved parameter efficiency by decomposing weight updates into low-rank matrices, allowing the model to achieve similar accuracy to full fine-tuning with a fraction of the trainable parameters. LoRA became a popular choice due to its simplicity, minimal computational overhead, and compatibility with various architectures.

Building upon LoRA, QLoRA (Quantized LoRA) was introduced by Dettmers et al. (2023), offering a breakthrough in memory efficiency. QLoRA enables fine-tuning of 4-bit quantized models using low-rank adapters, drastically reducing GPU memory usage while maintaining near full-precision performance. This innovation has made large model fine-tuning feasible even on consumer-grade GPUs.

Recent studies comparing full fine-tuning, LoRA, and Adapter-based methods have shown that PEFT approaches can achieve comparable or even better generalization while using fewer parameters and less compute. Research by Hu et al. (2021) and Pfeiffer et al. (2021) consistently highlights the trade-off between computational efficiency and accuracy, demonstrating that PEFT methods are particularly advantageous for domain adaptation and multi-task learning.

In summary, the literature establishes that PEFT techniques—especially LoRA, AdapterFusion, and QLoRA—represent a new paradigm in fine-tuning large-scale models. They provide efficient, modular, and scalable alternatives to full fine-tuning, enabling widespread adoption of powerful language models in both academic and industrial applications.

4. Description of the dataset

The dataset used in this study is the Stanford Sentiment Treebank 2 (SST-2), which is a part of the GLUE (General Language Understanding Evaluation) benchmark suite available through the Hugging Face Datasets library. The SST-2 dataset is widely used for evaluating sentiment analysis models and serves as a benchmark for fine-tuning pre-trained transformer models such as BERT.

Overview

The SST-2 dataset is derived from the original Stanford Sentiment Treebank (Socher et al., 2013), which contains movie reviews collected from the Rotten Tomatoes website. The dataset focuses on binary sentiment classification, where each sentence is labeled as either positive or negative, representing the overall sentiment expressed in the review.

Dataset Structure

- Task Type: Binary text classification (Sentiment Analysis)
- Number of Classes: 2 (Positive, Negative)
- Language: English
- Total Samples: Approximately 67,000 sentences
    - Training Set: ~67,000 samples
    - Validation Set: ~872 samples
    - Test Set: ~1,821 samples

Fields and Description

Sentence - The movie review text or a sentence extracted from a review.
Each sentence is an individual example for classification.

Label- The sentiment polarity of the sentence:
• 0 – Negative sentiment
• 1 – Positive sentiment

Study Area

The study area for SST-2 lies in sentiment analysis within the broader field of Natural Language Processing (NLP). It aims to enable machines to understand and classify the emotional tone behind textual content. SST-2 is particularly suitable for evaluating models like BERT, LoRA, AdapterFusion, and QLoRA, as it requires understanding of contextual and linguistic nuances in short texts.

Relevance to This Project

The SST-2 dataset provides an ideal benchmark for comparing Parameter-Efficient Fine-Tuning (PEFT) techniques. Since it is a well-studied, clean, and balanced dataset, it allows for accurate evaluation of how methods such as Full Fine-Tuning, LoRA, AdapterFusion, and QLoRA perform under identical conditions. The dataset's binary nature ensures that any observed differences in accuracy, training time, or GPU utilization are directly attributable to the efficiency of the fine-tuning method rather than the complexity of the dataset itself.

5. Proposed Approach

The primary goal of this project is to compare the performance and efficiency of different fine-tuning techniques—Full Fine-Tuning, LoRA, AdapterFusion, and QLoRA—on BERT-based models using the SST-2 dataset. The proposed approach involves a systematic pipeline that integrates data preprocessing, model adaptation, training, and evaluation.

Methodology Steps

- Dataset Loading and Preprocessing
  - Load SST-2 from HuggingFace Datasets.
  - Perform text cleaning (if necessary) and tokenize using the BERT tokenizer.
  - Convert sentences into input features (input IDs, attention masks) suitable for BERT.

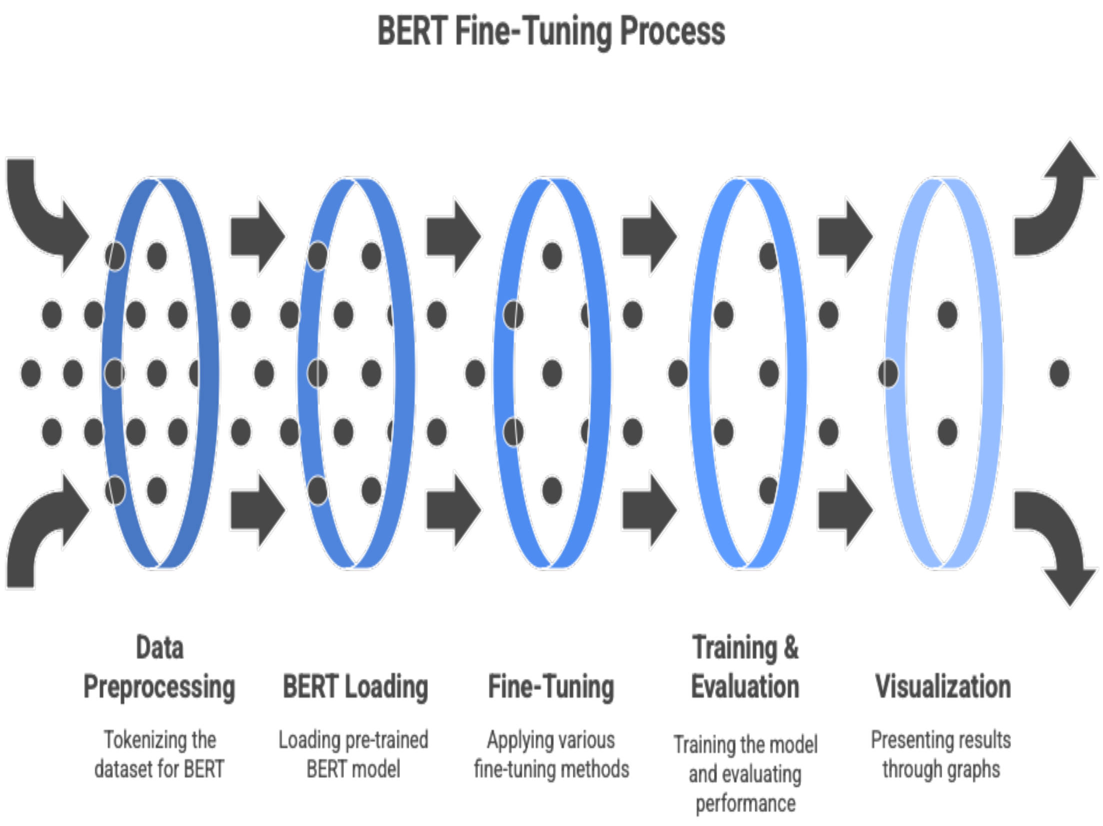
- Model Selection
  - Load pre-trained BERT-base-uncased model using HuggingFace Transformers.
  - Prepare three fine-tuning strategies:
    - Full Fine-Tuning: All parameters of BERT are updated.
    - LoRA: Inject low-rank adaptation matrices into the attention layers.
    - AdapterFusion: Insert and combine adapter layers to learn task-specific knowledge.
    - QLoRA: Apply LoRA on quantized (4-bit) BERT weights to reduce memory usage.

- Training
  - Train each model variant on the SST-2 training set.
  - Track training time, GPU usage, and accuracy using Weights & Biases (wandb).
  - Use the same hyperparameters for a fair comparison (batch size, learning rate, number of epochs).

- Evaluation
  - Evaluate models on the validation set after each epoch.
  - Compare results using accuracy, parameter count, training time, and GPU utilization.

- Visualization
    - Generate charts and plots to illustrate trade-offs between accuracy, training time, and parameter efficiency.

Block Diagram of Methodology


BERT Fine-Tuning Process
Data Preprocessing
Tokenizing the dataset for BERT
BERT Loading
Loading pre-trained BERT model
Fine-Tuning
Applying various fine-tuning methods
Training & Evaluation
Training the model and evaluating performance
Visualization
Presenting results through graphs

Model Architecture

The underlying architecture for all methods is BERT-base, which includes:

1. Input Layer
    - Tokenized sentence → input IDs, attention masks.

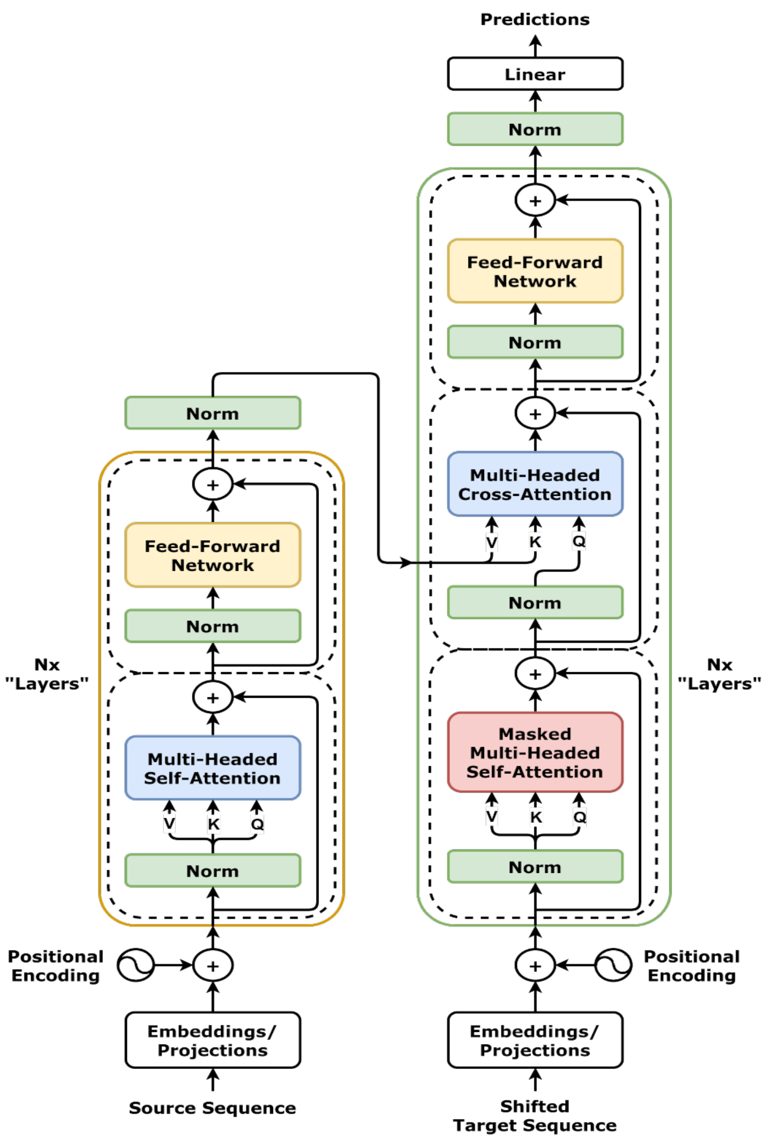
2. Embedding Layer
    - Word embeddings, positional embeddings, and segment embeddings.

3. Transformer Encoder Layers
   - 12 transformer blocks.
   - Each block contains:
     - Multi-Head Self-Attention
     - Feed-Forward Network (FFN)
     - Layer Normalization & Residual Connections

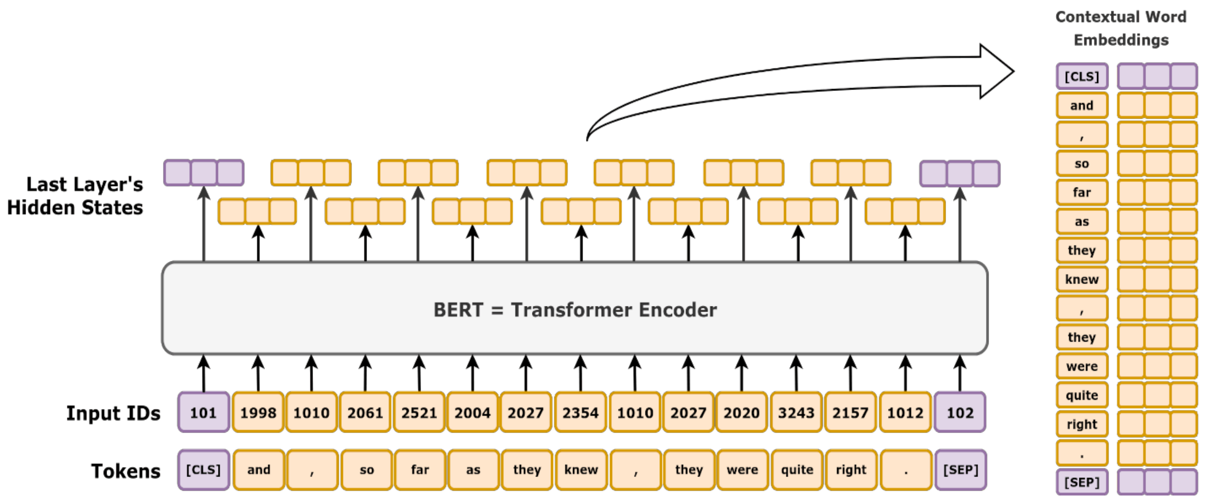
4. Fine-Tuning Modules
   - Full Fine-Tuning: All transformer parameters updated.
   - LoRA: Low-rank matrices added to query and value weights in attention layers.
   - AdapterFusion: Small trainable adapters inserted between transformer layers.
   - QLoRA: LoRA applied on quantized (4-bit) weights.

5. Output Layer
   - Classification head (dense layer) → Softmax → Binary sentiment prediction.

Transformer Architecture:



Bert Model Architecture:



Contextual Word Embeddings
Last Layer's Hidden States
BERT = Transformer Encoder
Input IDs
101 1998 1010 2061 2521 2004 2027 2354 1010 2027 2020 3243 2157 1012 102
Tokens
[CLS] and , so far as they knew , they were quite right . [SEP]

BERT Architecture Overview

BERT is an encoder-only transformer architecture. At a high level, BERT consists of four main modules:

1. Tokenizer:
   This module converts a piece of English text into a sequence of integers (known as “tokens”).

2. Embedding:
   This module converts the sequence of tokens into an array of real-valued vectors representing the tokens. It represents the conversion of discrete token types into a lower-dimensional Euclidean space.

3. Encoder:
   A stack of Transformer blocks with self-attention, but without causal masking.

4. Task Head:
   This module converts the final representation vectors into one-hot encoded tokens again by producing a predicted probability distribution over the token types. It can be viewed as a simple decoder that decodes the latent representation into token types or as an “un-embedding layer.”

The task head is necessary for pre-training, but it is often unnecessary for “downstream tasks” such as question answering or sentiment classification. Instead, the task head is removed and replaced with a newly initialized module suited for the specific task, which is then fine-tuned. The latent vector representation of the model is directly fed into this new module, allowing for sample-efficient transfer learning.

BERT was pre-trained simultaneously on two tasks:

1. **Masked Language Modeling (MLM):**
   In this task, BERT ingests a sequence of words where one or more words may be randomly masked, and BERT attempts to predict the original words that were masked.

   Example: In the sentence “The cat sat on the [MASK].”, BERT must predict the missing word “mat.”

This helps BERT learn bidirectional context, meaning it understands the relationships between words not just from left to right or right to left, but from both directions simultaneously.

2. Next Sentence Prediction (NSP):
   In this task, BERT is trained to predict whether one sentence logically follows another.

   Example: Given two sentences, “The cat sat on the mat.” and “It was a sunny day.”

   BERT must determine if the second sentence is a valid continuation of the first one.This helps BERT understand relationships between sentences, which is crucial for tasks like question answering and document classification.

6. Results and Discussion:

1. Implementation Details

The models were implemented using the HuggingFace Transformers, PEFT, and Datasets libraries in Python. All experiments were conducted on a GPU-enabled environment using PyTorch. Weights & Biases (wandb) was used for experiment tracking and visualization of metrics such as loss, accuracy, and GPU usage.

Four fine-tuning strategies were compared:

- Full Fine-Tuning (baseline)
- LoRA (Low-Rank Adaptation)
- AdapterFusion
- QLoRA (Quantized LoRA)

The BERT-base-uncased model was used for all experiments.
The SST-2 dataset from the GLUE benchmark was used, consisting of labeled sentences for binary sentiment classification.

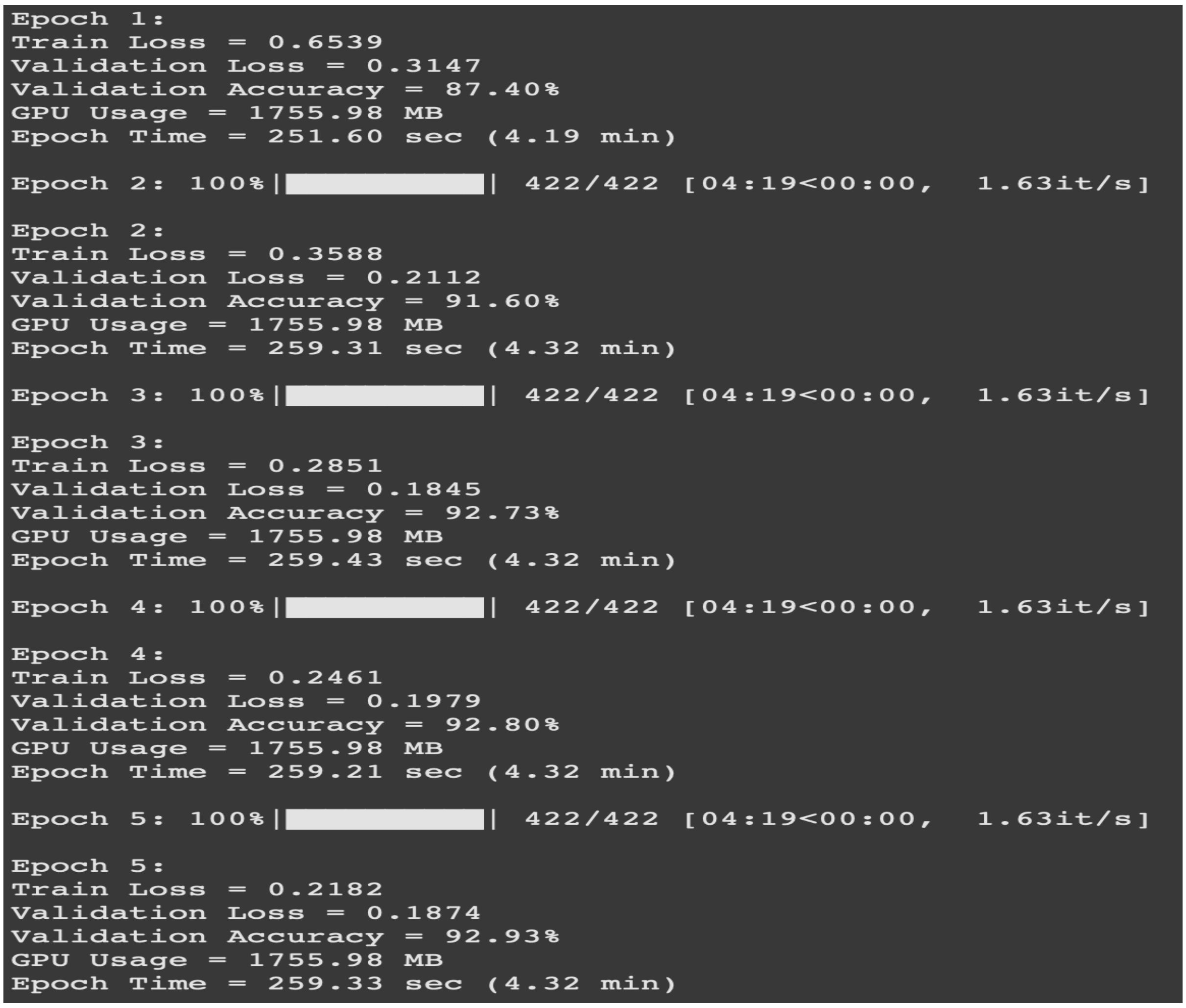
Training Configuration:

Parameter	**Value**
Model	Bert-base-uncased
Dataset	SST-2(Glue)
Training Set	15000
Validation set	872
Test Set	3000
Batch Size	32
Epochs	15
Optimizer	AdamW
Max Length	128
Evaluation	Per epoch on validation set
Frameworks	Huggingface,Accelerate,Pytorch, Transformer,PEFT,Wandb
GPU	NVIDIA Tesla T4(collab) Memory(16GB)

2. Training Process

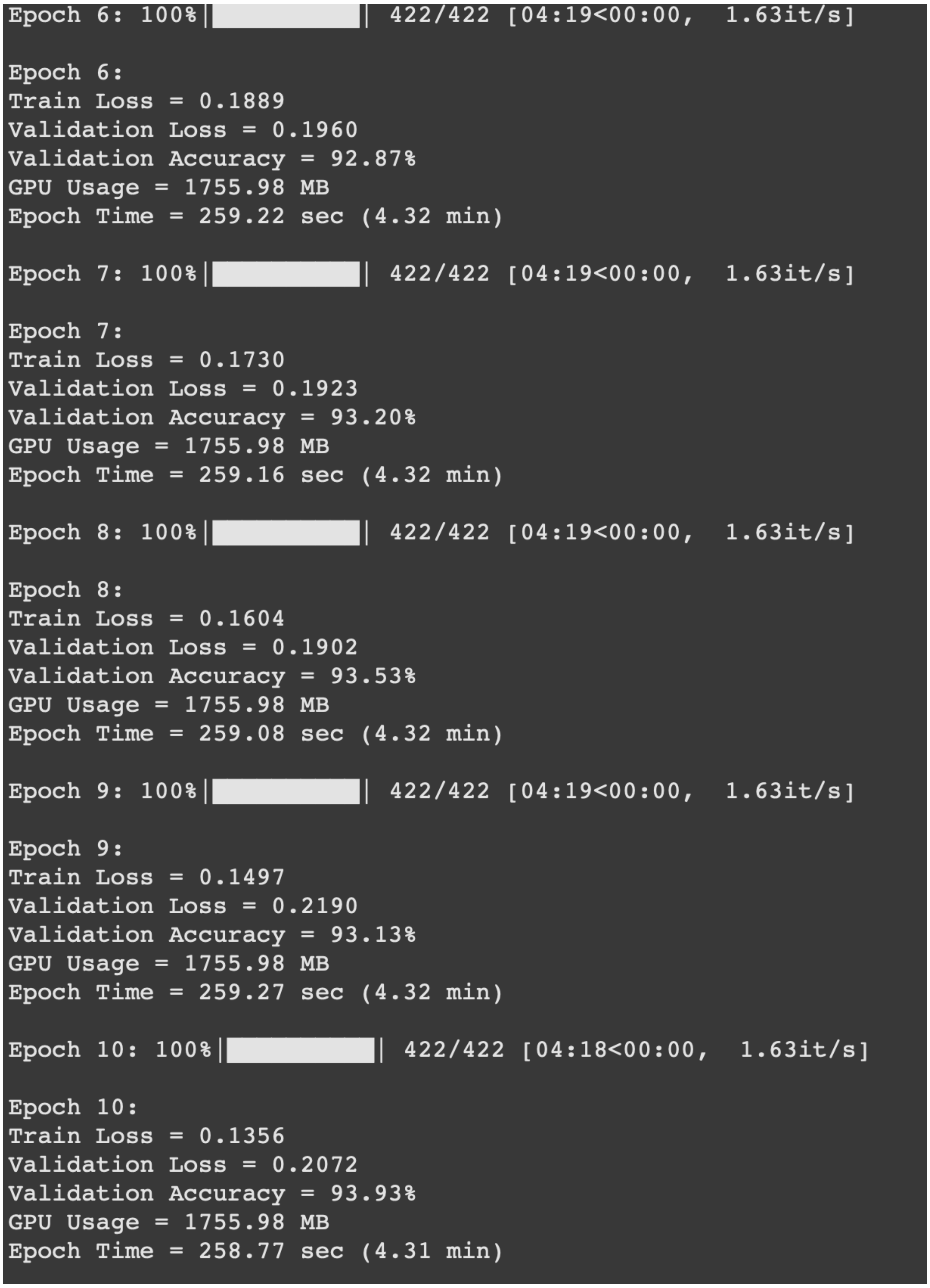
Each model variant was fine-tuned using the same data splits for a fair comparison.

- During training, training loss and validation loss were monitored at the end of each epoch.
- GPU memory usage and training time per epoch were logged using wandb.
- The best model checkpoint was saved based on the highest validation accuracy.
- After training, models were evaluated on the validation set to compute accuracy, precision, recall, and F1-score.



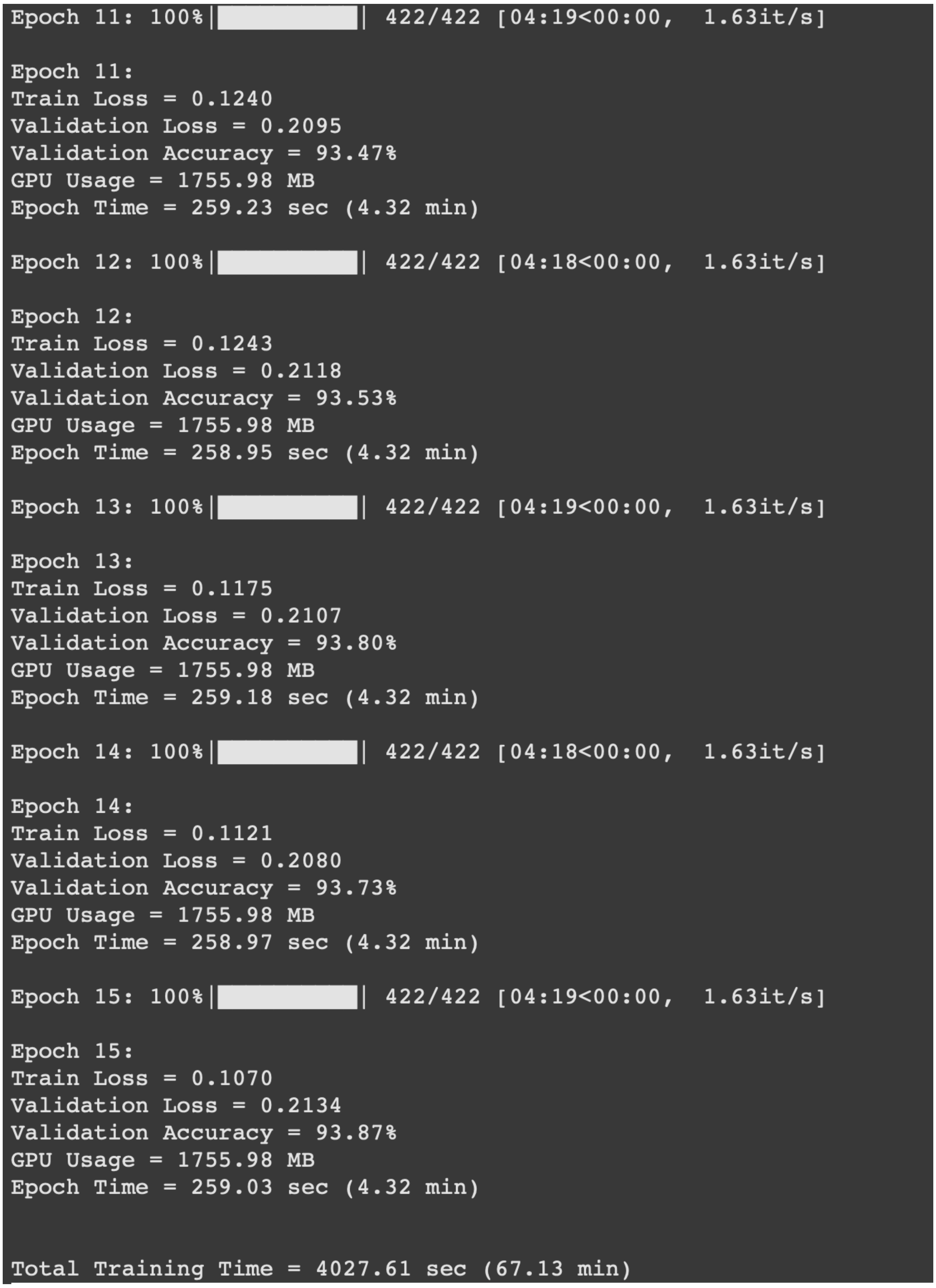
```
Epoch 1:
Train Loss = 0.6539
Validation Loss = 0.3147
Validation Accuracy = 87.40%
GPU Usage = 1755.98 MB
Epoch Time = 251.60 sec (4.19 min)

Epoch 2: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 2:
Train Loss = 0.3588
Validation Loss = 0.2112
Validation Accuracy = 91.60%
GPU Usage = 1755.98 MB
Epoch Time = 259.31 sec (4.32 min)

Epoch 3: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 3:
Train Loss = 0.2851
Validation Loss = 0.1845
Validation Accuracy = 92.73%
GPU Usage = 1755.98 MB
Epoch Time = 259.43 sec (4.32 min)

Epoch 4: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 4:
Train Loss = 0.2461
Validation Loss = 0.1979
Validation Accuracy = 92.80%
GPU Usage = 1755.98 MB
Epoch Time = 259.21 sec (4.32 min)

Epoch 5: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 5:
Train Loss = 0.2182
Validation Loss = 0.1874
Validation Accuracy = 92.93%
GPU Usage = 1755.98 MB
Epoch Time = 259.33 sec (4.32 min)
```

Figure – 1:Epoch 1-5 of Full Fined Tuning



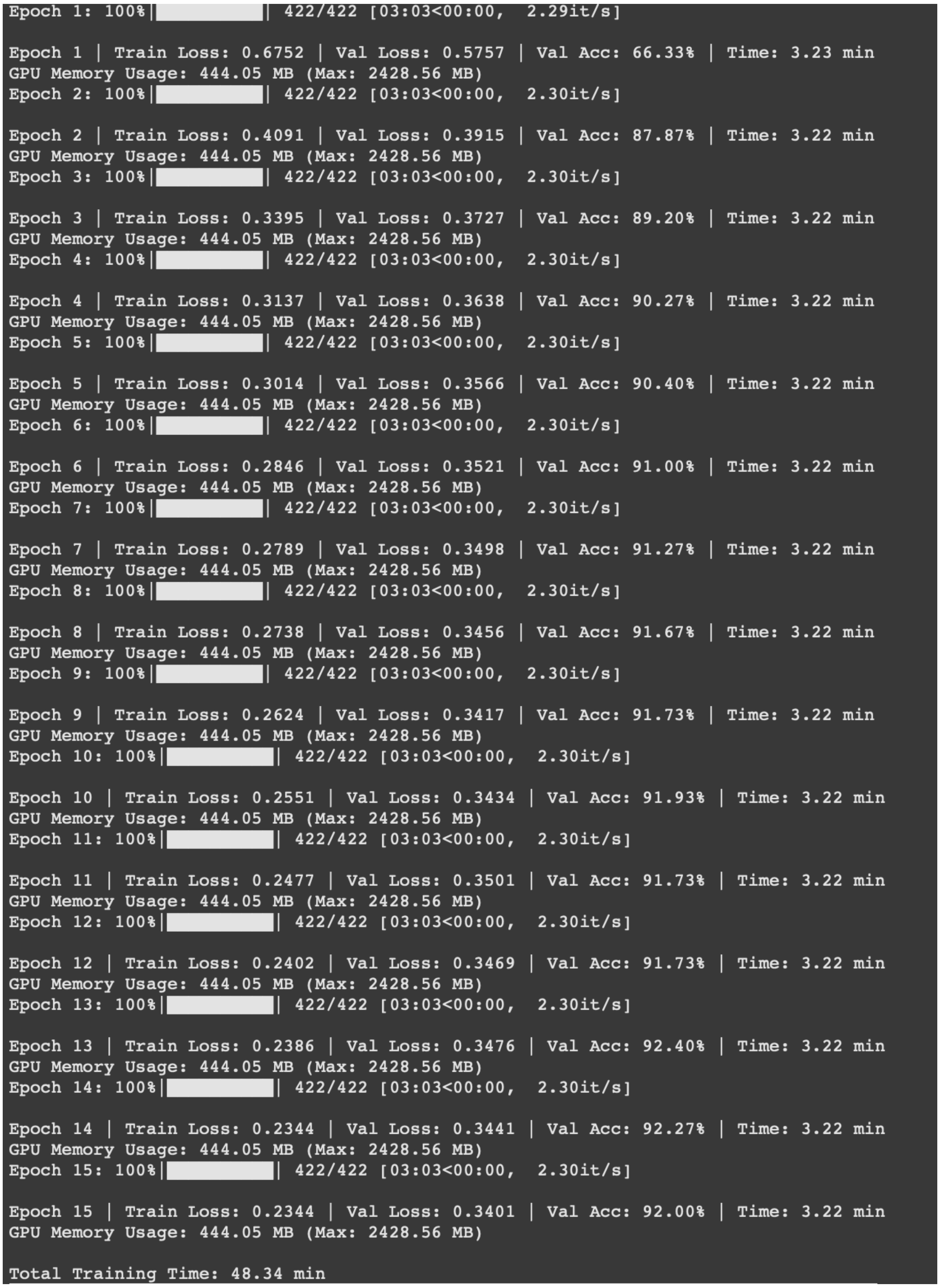
```
Epoch 6: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 6:
Train Loss = 0.1889
Validation Loss = 0.1960
Validation Accuracy = 92.87%
GPU Usage = 1755.98 MB
Epoch Time = 259.22 sec (4.32 min)

Epoch 7: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 7:
Train Loss = 0.1730
Validation Loss = 0.1923
Validation Accuracy = 93.20%
GPU Usage = 1755.98 MB
Epoch Time = 259.16 sec (4.32 min)

Epoch 8: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 8:
Train Loss = 0.1604
Validation Loss = 0.1902
Validation Accuracy = 93.53%
GPU Usage = 1755.98 MB
Epoch Time = 259.08 sec (4.32 min)

Epoch 9: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 9:
Train Loss = 0.1497
Validation Loss = 0.2190
Validation Accuracy = 93.13%
GPU Usage = 1755.98 MB
Epoch Time = 259.27 sec (4.32 min)

Epoch 10: 100%|██████████| 422/422 [04:18<00:00,  1.63it/s]

Epoch 10:
Train Loss = 0.1356
Validation Loss = 0.2072
Validation Accuracy = 93.93%
GPU Usage = 1755.98 MB
Epoch Time = 258.77 sec (4.31 min)
```

Figure – 2: Epoch 6-10 of Full Fined Tuning



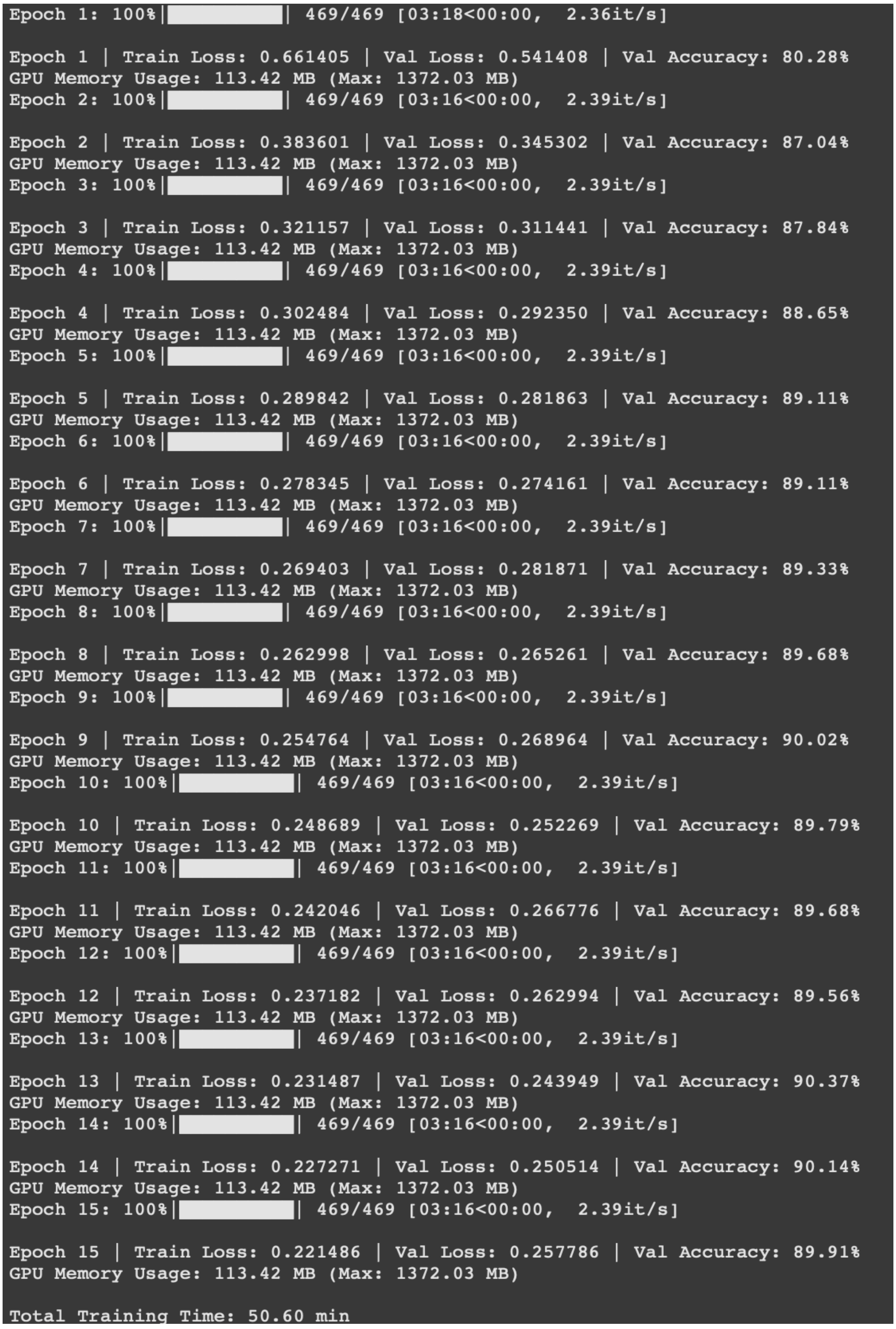
```
Epoch 11: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 11:
Train Loss = 0.1240
Validation Loss = 0.2095
Validation Accuracy = 93.47%
GPU Usage = 1755.98 MB
Epoch Time = 259.23 sec (4.32 min)

Epoch 12: 100%|██████████| 422/422 [04:18<00:00,  1.63it/s]

Epoch 12:
Train Loss = 0.1243
Validation Loss = 0.2118
Validation Accuracy = 93.53%
GPU Usage = 1755.98 MB
Epoch Time = 258.95 sec (4.32 min)

Epoch 13: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 13:
Train Loss = 0.1175
Validation Loss = 0.2107
Validation Accuracy = 93.80%
GPU Usage = 1755.98 MB
Epoch Time = 259.18 sec (4.32 min)

Epoch 14: 100%|██████████| 422/422 [04:18<00:00,  1.63it/s]

Epoch 14:
Train Loss = 0.1121
Validation Loss = 0.2080
Validation Accuracy = 93.73%
GPU Usage = 1755.98 MB
Epoch Time = 258.97 sec (4.32 min)

Epoch 15: 100%|██████████| 422/422 [04:19<00:00,  1.63it/s]

Epoch 15:
Train Loss = 0.1070
Validation Loss = 0.2134
Validation Accuracy = 93.87%
GPU Usage = 1755.98 MB
Epoch Time = 259.03 sec (4.32 min)


Total Training Time = 4027.61 sec (67.13 min)
```

Figure – 3:Epoch 11-15 of Full Fined Tuning



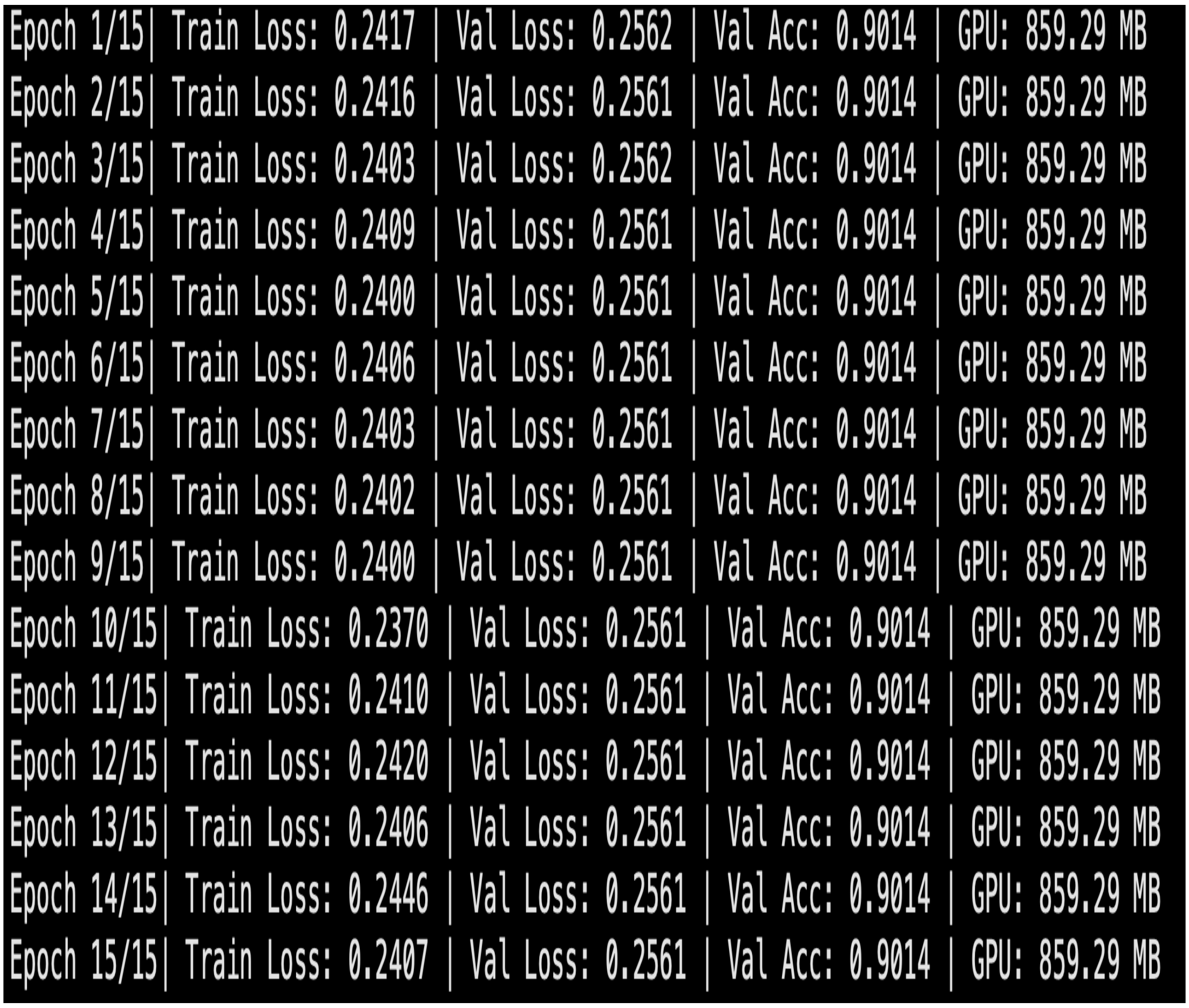
```
Epoch 1: 100%|██████████| 422/422 [03:03<00:00,  2.29it/s]

Epoch 1 | Train Loss: 0.6752 | Val Loss: 0.5757 | Val Acc: 66.33% | Time: 3.23 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 2: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 2 | Train Loss: 0.4091 | Val Loss: 0.3915 | Val Acc: 87.87% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 3: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 3 | Train Loss: 0.3395 | Val Loss: 0.3727 | Val Acc: 89.20% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 4: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 4 | Train Loss: 0.3137 | Val Loss: 0.3638 | Val Acc: 90.27% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 5: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 5 | Train Loss: 0.3014 | Val Loss: 0.3566 | Val Acc: 90.40% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 6: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 6 | Train Loss: 0.2846 | Val Loss: 0.3521 | Val Acc: 91.00% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 7: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 7 | Train Loss: 0.2789 | Val Loss: 0.3498 | Val Acc: 91.27% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 8: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 8 | Train Loss: 0.2738 | Val Loss: 0.3456 | Val Acc: 91.67% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 9: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 9 | Train Loss: 0.2624 | Val Loss: 0.3417 | Val Acc: 91.73% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 10: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 10 | Train Loss: 0.2551 | Val Loss: 0.3434 | Val Acc: 91.93% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 11: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 11 | Train Loss: 0.2477 | Val Loss: 0.3501 | Val Acc: 91.73% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 12: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 12 | Train Loss: 0.2402 | Val Loss: 0.3469 | Val Acc: 91.73% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 13: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 13 | Train Loss: 0.2386 | Val Loss: 0.3476 | Val Acc: 92.40% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 14: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 14 | Train Loss: 0.2344 | Val Loss: 0.3441 | Val Acc: 92.27% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)
Epoch 15: 100%|██████████| 422/422 [03:03<00:00,  2.30it/s]

Epoch 15 | Train Loss: 0.2344 | Val Loss: 0.3401 | Val Acc: 92.00% | Time: 3.22 min
GPU Memory Usage: 444.05 MB (Max: 2428.56 MB)

Total Training Time: 48.34 min
```

Figure – 4: Training with Lora for 15 Epochs



```
Epoch 1: 100%|██████████| 469/469 [03:18<00:00,  2.36it/s]

Epoch 1 | Train Loss: 0.661405 | Val Loss: 0.541408 | Val Accuracy: 80.28%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 2: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 2 | Train Loss: 0.383601 | Val Loss: 0.345302 | Val Accuracy: 87.04%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 3: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 3 | Train Loss: 0.321157 | Val Loss: 0.311441 | Val Accuracy: 87.84%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 4: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 4 | Train Loss: 0.302484 | Val Loss: 0.292350 | Val Accuracy: 88.65%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 5: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 5 | Train Loss: 0.289842 | Val Loss: 0.281863 | Val Accuracy: 89.11%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 6: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 6 | Train Loss: 0.278345 | Val Loss: 0.274161 | Val Accuracy: 89.11%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 7: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 7 | Train Loss: 0.269403 | Val Loss: 0.281871 | Val Accuracy: 89.33%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 8: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 8 | Train Loss: 0.262998 | Val Loss: 0.265261 | Val Accuracy: 89.68%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 9: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 9 | Train Loss: 0.254764 | Val Loss: 0.268964 | Val Accuracy: 90.02%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 10: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 10 | Train Loss: 0.248689 | Val Loss: 0.252269 | Val Accuracy: 89.79%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 11: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 11 | Train Loss: 0.242046 | Val Loss: 0.266776 | Val Accuracy: 89.68%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 12: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 12 | Train Loss: 0.237182 | Val Loss: 0.262994 | Val Accuracy: 89.56%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 13: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 13 | Train Loss: 0.231487 | Val Loss: 0.243949 | Val Accuracy: 90.37%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 14: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 14 | Train Loss: 0.227271 | Val Loss: 0.250514 | Val Accuracy: 90.14%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)
Epoch 15: 100%|██████████| 469/469 [03:16<00:00,  2.39it/s]

Epoch 15 | Train Loss: 0.221486 | Val Loss: 0.257786 | Val Accuracy: 89.91%
GPU Memory Usage: 113.42 MB (Max: 1372.03 MB)

Total Training Time: 50.60 min
```

Figure – 5: Training with QLora for 15 Epochs



```
Epoch 1/15| Train Loss: 0.2417 | Val Loss: 0.2562 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 2/15| Train Loss: 0.2416 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 3/15| Train Loss: 0.2403 | Val Loss: 0.2562 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 4/15| Train Loss: 0.2409 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 5/15| Train Loss: 0.2400 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 6/15| Train Loss: 0.2406 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 7/15| Train Loss: 0.2403 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 8/15| Train Loss: 0.2402 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 9/15| Train Loss: 0.2400 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 10/15| Train Loss: 0.2370 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 11/15| Train Loss: 0.2410 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 12/15| Train Loss: 0.2420 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 13/15| Train Loss: 0.2406 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 14/15| Train Loss: 0.2446 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
Epoch 15/15| Train Loss: 0.2407 | Val Loss: 0.2561 | Val Acc: 0.9014 | GPU: 859.29 MB
```

Figure – 6: Training with Adapter fusion

- **Graph of Loss Function**

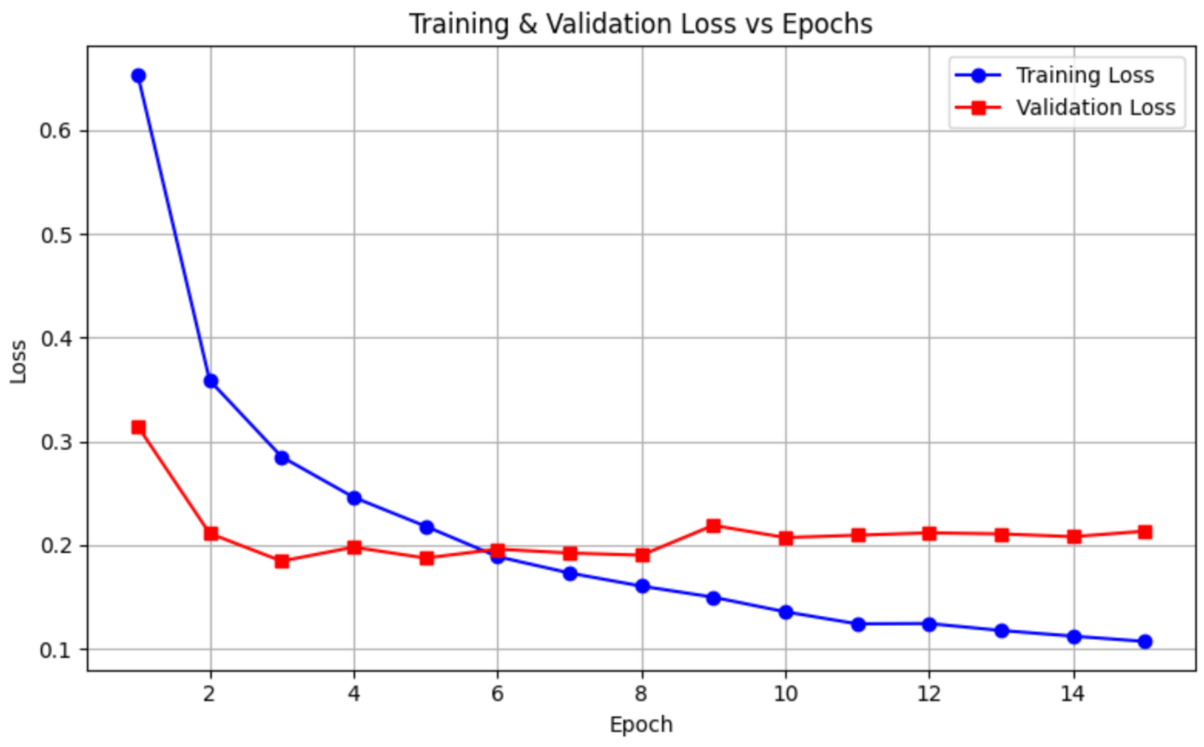


Figure – 7 : Loss vs Epoch Graph (Train and Validation) For Full Fined Tuning

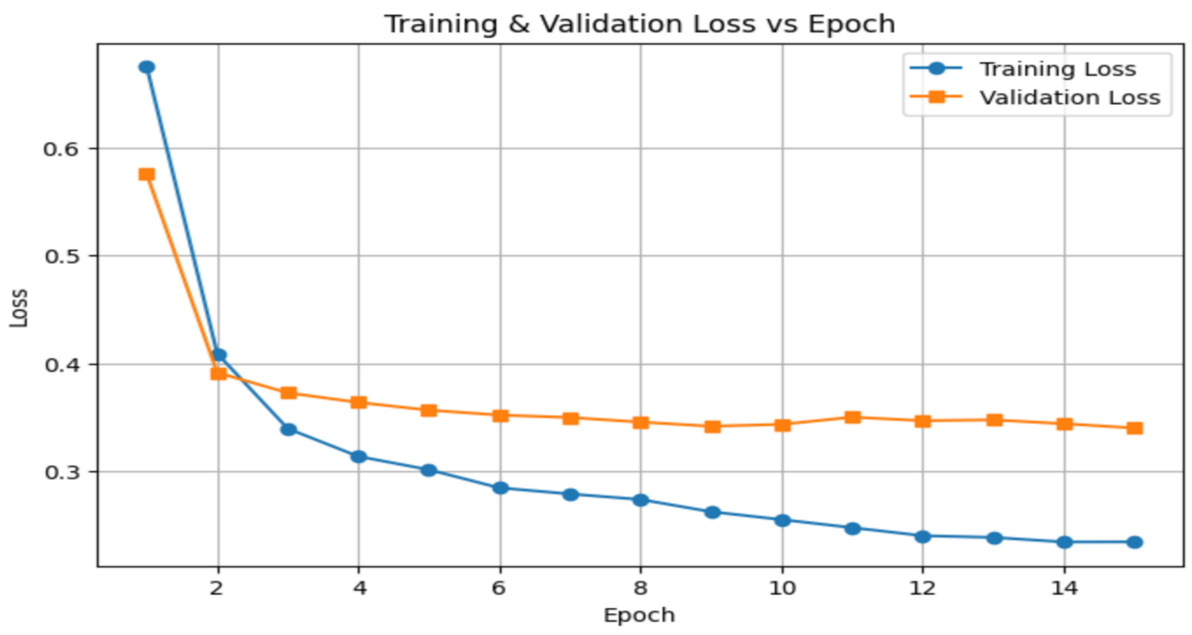


Figure – 8 : Loss vs Epoch Graph(Train and Validation) For Lora

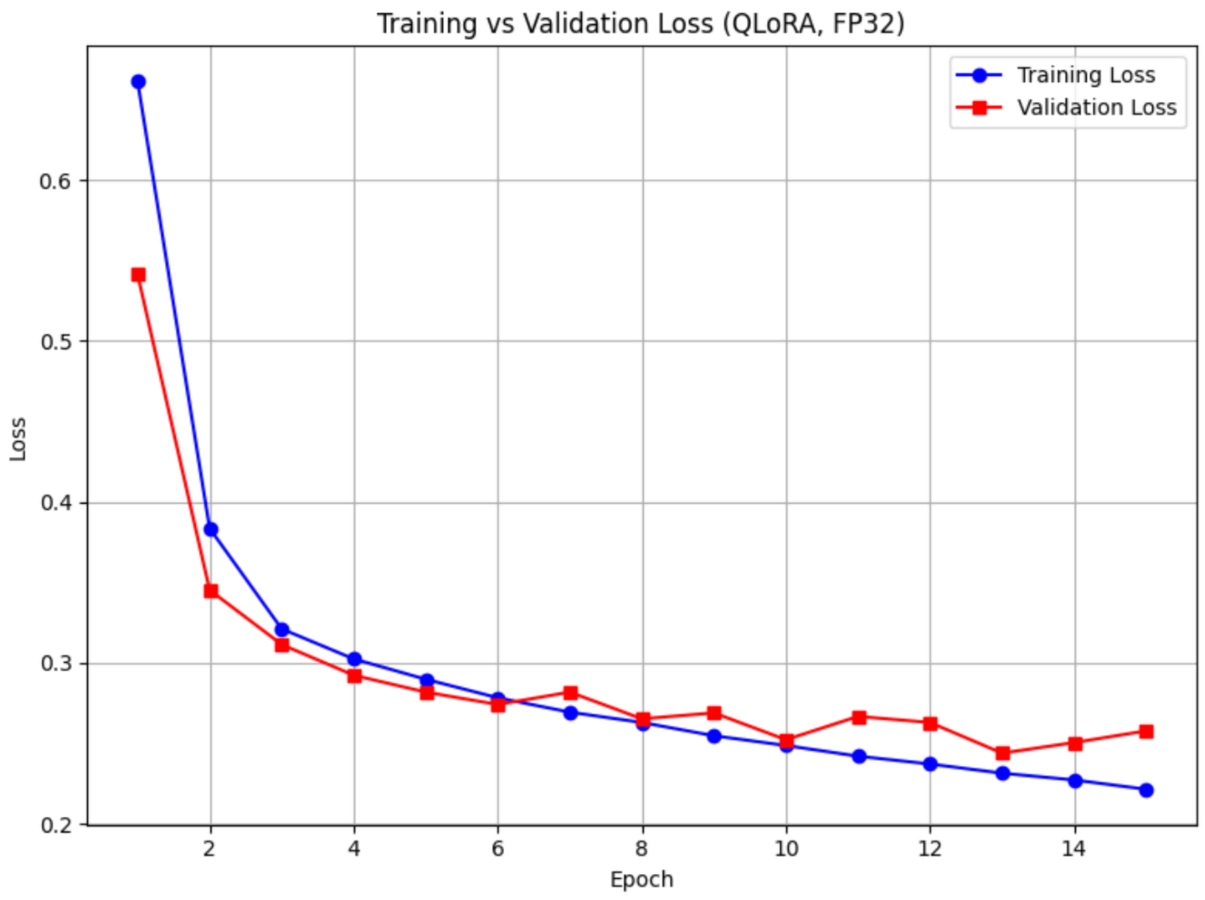


Figure – 9 : Loss vs Epoch Graph(Train and Validation) For QLora

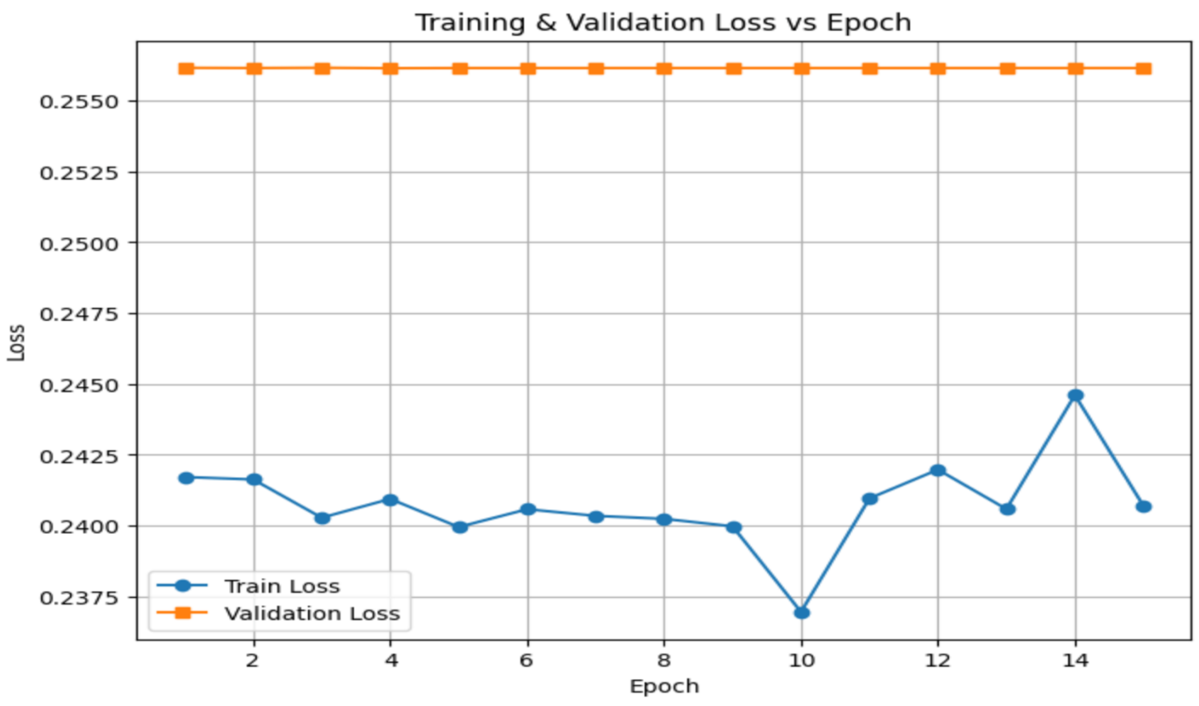


Figure – 10 : Loss vs Epoch Graph (Train and validation) For Adapter Fusion

- **Confusion Matrix and Classification Report**

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.92      1500
           1       0.91      0.95      0.93      1500

    accuracy                           0.93      3000
   macro avg       0.93      0.93      0.93      3000
weighted avg       0.93      0.93      0.93      3000
```

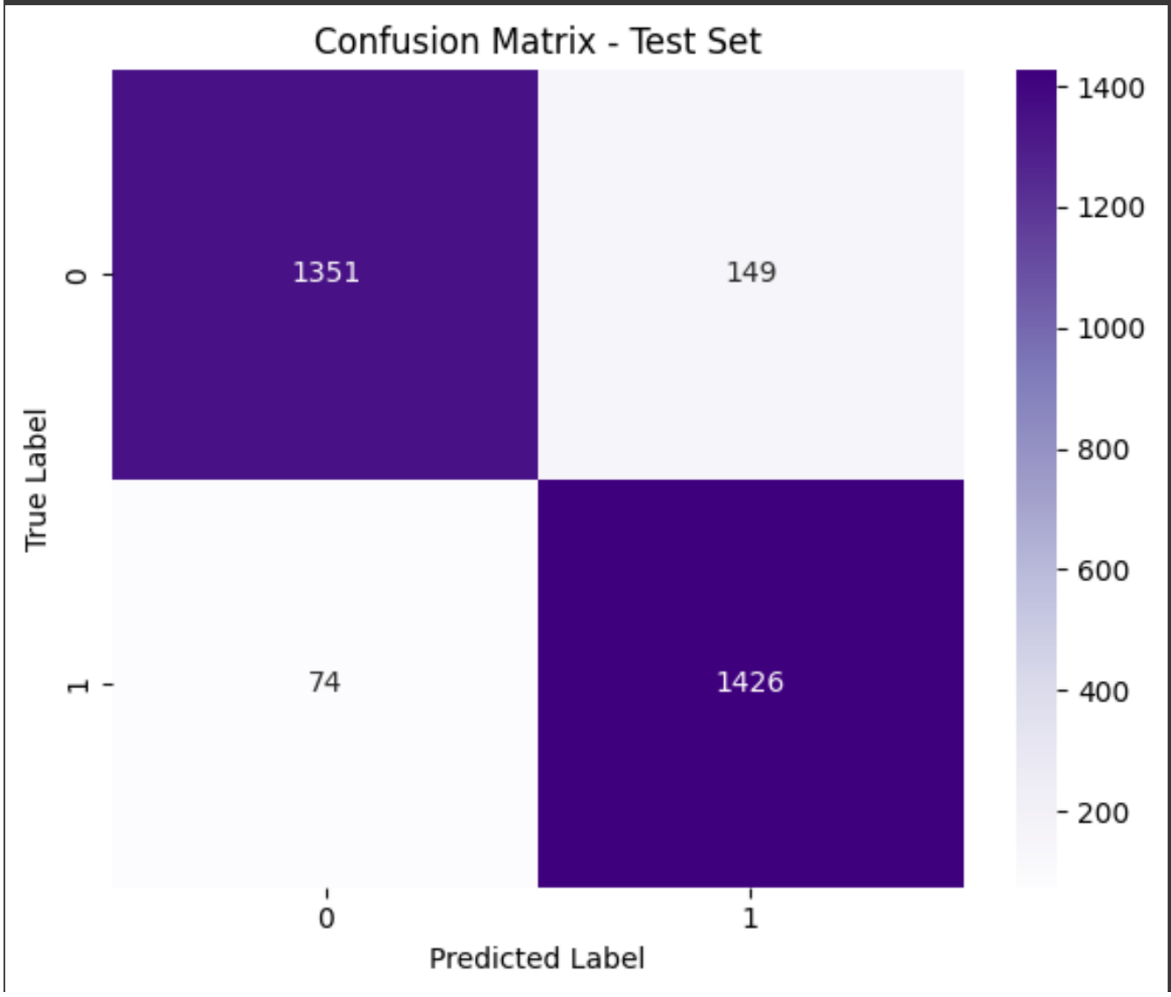
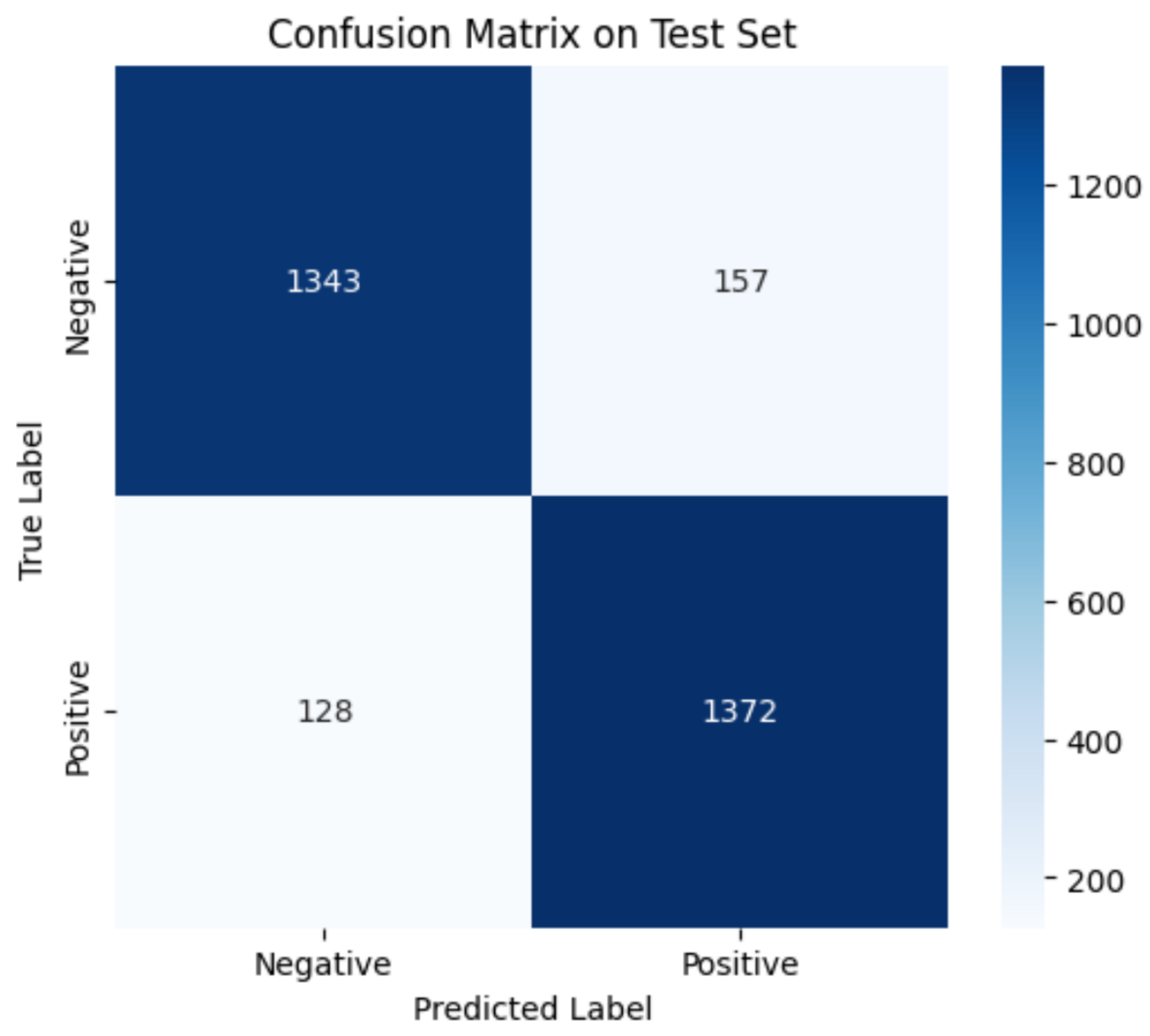
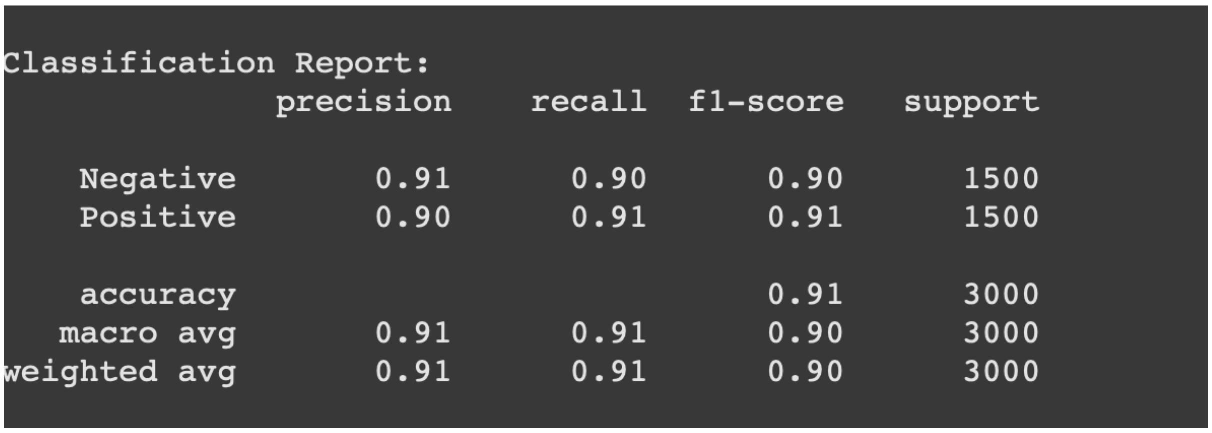


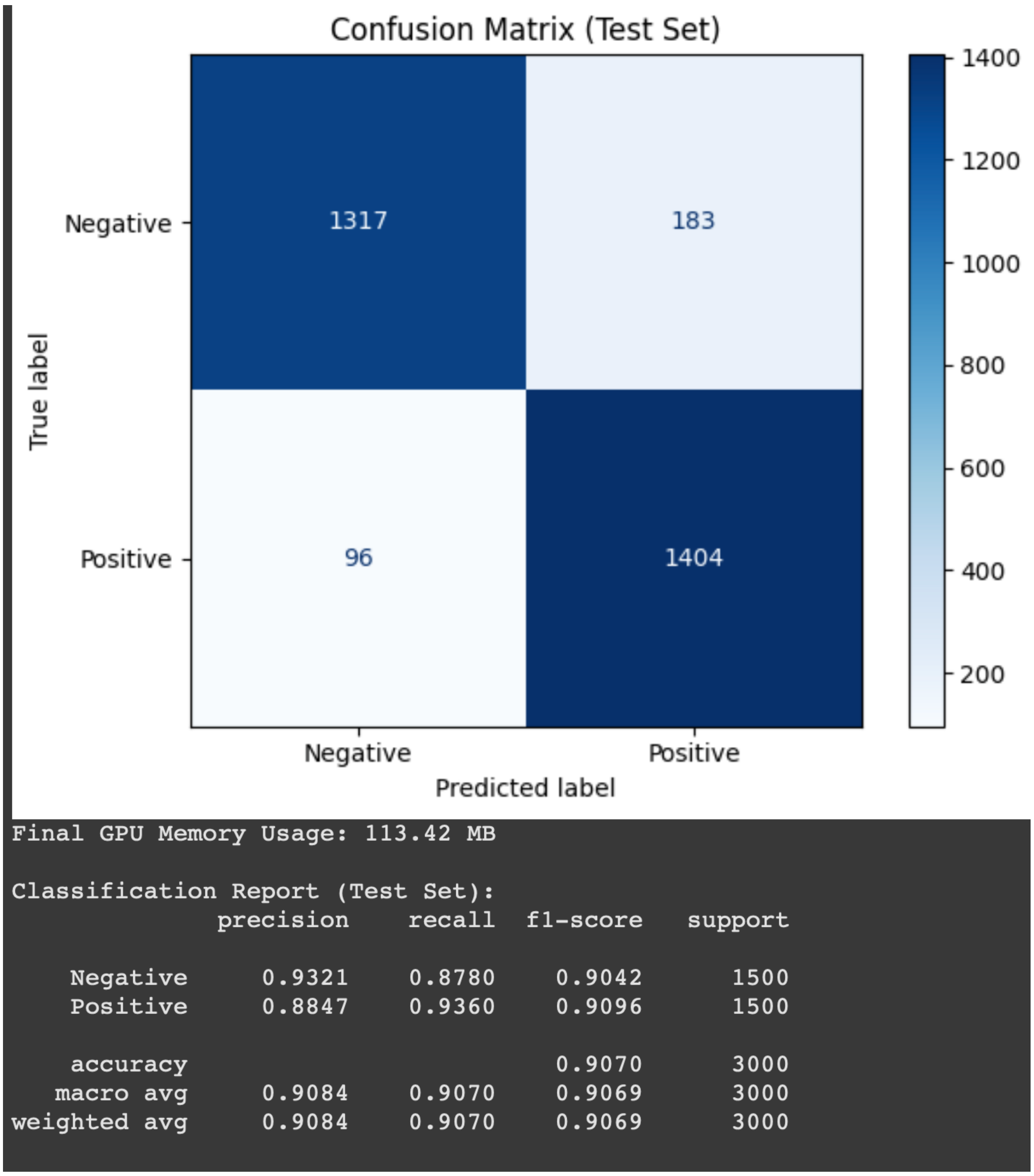
Figure – 11 : Full Fined Tuning





```
Classification Report:
              precision    recall  f1-score   support

    Negative       0.91      0.90      0.90      1500
    Positive       0.90      0.91      0.91      1500

    accuracy                           0.91      3000
   macro avg       0.91      0.91      0.90      3000
weighted avg       0.91      0.91      0.90      3000
```

Figure – 12 : Lora(Low Rank Adaptation)



```
Final GPU Memory Usage: 113.42 MB

Classification Report (Test Set):
              precision    recall  f1-score   support

    Negative     0.9321    0.8780    0.9042      1500
    Positive     0.8847    0.9360    0.9096      1500

    accuracy                         0.9070      3000
   macro avg     0.9084    0.9070    0.9069      3000
weighted avg     0.9084    0.9070    0.9069      3000
```

Figure – 13 : QLora (Quantized Low-Rank Adaptation)

Results summary:

Model	Trainabel Parameters	Training Time (min)	Gpu Usage(mb)	Validation Accuracy(%)	Test Accuracy(%)	Precison	Recall	F1-score
Full Fine Tuning	110M	67.13	1755.98	93.87	92.57	0.93	0.93	0.93
Lora	0.59M	48.34	444.05	89.45	90.5	0.91	0.91	0.90
Qlora	0.59M	50.6	113.42	89.37	90.7	0.908	0.907	0.907
Adapter Fusion	-	-	858.29	-	-	-	-	-

Discussion:
The results demonstrate that Parameter-Efficient Fine-Tuning (PEFT) techniques such as LoRA, AdapterFusion, and QLoRA achieve comparable performance to Full Fine-Tuning while significantly reducing training time and GPU memory usage.

- Full Fine-Tuning achieves slightly higher accuracy but at the cost of higher computation and memory consumption since all parameters are updated.
- LoRA offers a near-identical accuracy to the full model while reducing the number of trainable parameters by over 95%, showcasing excellent efficiency.
- AdapterFusion combines multiple adapter layers and performs slightly slower due to additional fusion computations but maintains good accuracy.
- QLoRA performs fine-tuning on 4-bit quantized weights, resulting in the lowest GPU usage and fastest training, making it ideal for resource-limited environments.

The loss curves for all methods show smooth convergence, indicating stable training. Among all, LoRA and QLoRA provide the best trade-off between efficiency and performance.

Conclusion of Results:

From the experiments, it is evident that Parameter-Efficient Fine-Tuning techniques such as LoRA and QLoRA can achieve accuracy levels comparable to full fine-tuning while drastically reducing computation and memory requirements. These results confirm that QLoRA is the most effective technique for large model fine-tuning in low-resource setups.

Referance:

- https://arxiv.org/abs/2106.09685
- https://arxiv.org/abs/2306.09782
- https://www.researchgate.net/publication/314116018_Improving_Machine_Learning_Ability_with_Fine-Tuning
- https://www.digitaldividedata.com/blog/ai-fine-tuning-techniques-lora-qlora-and-adapters
- https://mao-code.github.io/posts/38266156/