SEND-IT: Sentiment Analysis on Parameter-Efficient Neural Distillate with Iterative Fine Tuning

A Georgia Tech (CS 7650) Natural Language Processing final class project

Collaborators: Priya Tamilselvan, Jack Henderson, Saaliha Allaudin, Thanasis Taprantzis

About:

This project is the culmination of the work of four computer science master's students at Georgia Tech over the course of a semester in the Natural Language Processing graduate course taught by Dr. Alan Ritter, as an extension of some of the topics covered there.

A short summary:

The aims of this project, which are refined and expressed more eloquently through the publication this work aided us in composing, are to examine the performative differences between two primary parameter-efficient fine-tuning techniques used in natural language model refinement, namely BitFit and LoRA, and evaluate these and other techniques (such as model distillation) in a binary sentiment analysis task on the Sentiment140 dataset, which amasses 1.6m Tweets flagged as positive or negative in sentiment, all made possible by open-use of Meta AI's Open Pre-trained Transformer 125m parameter model (OPT-125m), which is the target of the noted model distillation.

```
"""We begin our process by installing packages such as pytorch, which is used ext
transformers and datasets packages, which are used to run the OPT transformer mod

!pip install torch transformers datasets
```

```
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pacl
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (36
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.9 MB/s eta 0:00:
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 78.2 MB/s eta 0:00:0
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 55.1 MB/s eta 0:00:0
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 42.0 MB/s eta 0:00
```

```
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.2 MB/s eta 0:00:0
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 4.9 MB/s eta 0:00:0
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (5
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 20.0 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (1
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.6 MB/s eta 0:00:0
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.7 MB/s eta 0:00:0
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 73.2 MB/s eta 0:00:00
Downloading datasets-3.5.0-py3-none-any.whl (491 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.2/491.2 kB 42.9 MB/s eta 0:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 11.8 MB/s eta 0:00
Downloading fsspec-2024.12.0-py3-none-any.whl (183 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 183.9/183.9 kB 19.2 MB/s eta 0:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 14.5 MB/s eta 0:00
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 18.9 MB/s eta 0:00
Installing collected packages: xxhash, nvidia-nvjitlink-cu12, nvidia-curand-cu
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
```

```
"""This step configures the credentials of the active user to seemlessly enable p

!git config --global credential.helper store


"""We next import the installed packages, namely the OPT model (and ALBERT and GP

import torch
from torch.utils.data import DataLoader
from datasets import load_dataset
from transformers import AutoTokenizer, OPTForSequenceClassification
from transformers import AutoTokenizer, AlbertForSequenceClassification
from transformers import AutoTokenizer, GPT2ForSequenceClassification
import time
from sklearn.metrics import classification_report


""" We next instantiate (load) our temporary dataset, calling to our sentiment140

dataset = load_dataset("./sentiment140.py", name="sentiment140")
full_train = dataset["train"]

print("Train size:", len(dataset["train"]))
print("Test size:", len(dataset["test"]))
```

```
The repository for sentiment140 contains custom code which must be executed tc
You can avoid this prompt in future by passing the argument `trust_remote_code

Do you wish to run the custom code? [y/N] Y
```

Downloading data: 100% 81.4M/81.4M [00:55<00:00, 2.65MB/s]

Generating train split: 1600000/0 [01:13<00:00, 27176.38 examples/s]

Generating test split: 498/0 [00:00<00:00, 11815.68 examples/s]

```
Train size: 1600000
Test size: 498
```

```
""" We next import a few packages for randomization of our sampling, re for text

import random
import re
from datasets import Dataset
```

```
""" With the entire 1.6m entry dataset loaded in as full_train above, we next fil
though there seemed not to be any such instances), and we define negative and pos
We lastly overwrite our dataset with just the 50k class-balanced records from pre
our resource-efficiency computationally-constrained focus."""

all_data = [x for x in full_train if x["sentiment"] in [0,4]]

negative = [x for x in all_data if x["sentiment"] == 0]
positive = [x for x in all_data if x["sentiment"] == 4]

random.seed(42)

negative_sample = random.sample(negative, 25000)
positive_sample = random.sample(positive, 25000)

sampled_data = negative_sample + positive_sample

random.shuffle(sampled_data)

dataset = Dataset.from_list(sampled_data)


""" Next, with our 50k sentiment dataset, we perform pre-processing for standardi
noise in the form of mentions (e.g. @gatech), URLs (e.g. https://...), hashtags (
non-alphanumeric characters (e.g. emojis, capitalization, punctuation)"""

def clean_text(text):
  text = text.lower()
  text = re.sub(r"http\S+", "", text)
  text = re.sub(r"@\w+", "", text)
  text = re.sub(r"#\w+", "", text)
  text = re.sub(r"[^a-z0-9\s]", "", text)
  return text.strip()

dataset = dataset.map(lambda x: {"text": clean_text(x["text"])})
```

Map: 100% 50000/50000 [00:04<00:00, 9268.59 examples/

```
""" Class label remapping (in lieu of errant 'neutral' class label in the origina
negative was 0, is kept 0...positive was 4, is now 1"""

def relabel(datum):
  datum["sentiment"] = 0 if datum["sentiment"] == 0 else 1
  return datum

dataset = dataset.map(relabel)
```

Map: 100% 50000/50000 [00:03<00:00, 17307.13 examples/

```
""" We next import our tokenizer and model from HuggingFace with a user-unique lo

from huggingface_hub import login
token = "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"


model_name = "facebook/opt-125m"
num_labels = 2
tokenizer = AutoTokenizer.from_pretrained(model_name, token=token)
model = OPTForSequenceClassification.from_pretrained(model_name, num_labels=num_l
```

tokenizer_config.json: 100% 685/685 [00:00<00:00, 73.3kB/s]

config.json: 100% 651/651 [00:00<00:00, 47.3kB/s]

vocab.json: 100% 899k/899k [00:00<00:00, 31.3MB/s]

merges.txt: 100% 456k/456k [00:00<00:00, 25.7MB/s]

special_tokens_map.json: 100% 441/441 [00:00<00:00, 48.4kB/s]

```
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installe
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, bu
```

pytorch_model.bin: 100% 251M/251M [00:00<00:00, 356MB/s]

```
Some weights of OPTForSequenceClassification were not initialized from the mod
You should probably TRAIN this model on a down-stream task to be able to use i
```

```
""" We apply mapping from the seemingly arbitrary 0,4 scale to 0,1 for standard b

def map_labels(example):
    example["sentiment"] = 0 if example["sentiment"] == 0 else 1
    return example

dataset = dataset.map(map_labels)

print("Dataset size:", len(dataset))

from collections import Counter
print("Class distribution:", Counter(dataset["sentiment"]))
```

Map: 100% 50000/50000 [00:05<00:00, 18207.07 examples/s]

```
Dataset size: 50000
```

DOWNSAMPLING FOR TRAINING

```
""" We next perform downsampling and stratified splitting of the data 80/10/10 Tr

from sklearn.model_selection import train_test_split
from datasets import Dataset, DatasetDict
from collections import Counter
import pandas as pd

df = dataset.to_pandas()
df = df.dropna(subset=["text", "sentiment"])

print("Initial class distribution:", Counter(df["sentiment"]))

df_train, df_val_test = train_test_split(
    df,
    stratify=df["sentiment"],
    test_size=0.2,
    random_state=42
)

df_val, df_test = train_test_split(
    df_val_test,
    stratify=df_val_test["sentiment"],
    test_size=0.5,
```

```
    random_state=42
)

print("Train size:", len(df_train))
print("Val size:", len(df_val))
print("Test size:", len(df_test))
print("Train class distribution:", Counter(df_train["sentiment"]))
print("Val class distribution:", Counter(df_val["sentiment"]))
print("Test class distribution:", Counter(df_test["sentiment"]))

train_dataset = Dataset.from_pandas(df_train).remove_columns(["__index_level_0__"
val_dataset   = Dataset.from_pandas(df_val).remove_columns(["__index_level_0__"])
test_dataset  = Dataset.from_pandas(df_test).remove_columns(["__index_level_0__"]

def tokenize(example):
    return tokenizer(example["text"], truncation=True, padding="max_length", max_

train_dataset = train_dataset.map(tokenize, batched=True)
val_dataset   = val_dataset.map(tokenize, batched=True)
test_dataset  = test_dataset.map(tokenize, batched=True)

train_dataset.set_format("torch", columns=["input_ids", "attention_mask", "sentim
val_dataset.set_format("torch", columns=["input_ids", "attention_mask", "sentimen
test_dataset.set_format("torch", columns=["input_ids", "attention_mask", "sentime

tokenized_dataset = DatasetDict({
    "train": train_dataset,
    "validation": val_dataset,
    "test": test_dataset
})
```

```
Initial class distribution: Counter({1: 25000, 0: 25000})
Train size: 40000
Val size: 5000
Test size: 5000
Train class distribution: Counter({1: 20000, 0: 20000})
Val class distribution: Counter({0: 2500, 1: 2500})
Test class distribution: Counter({0: 2500, 1: 2500})
```

Map: 100% 40000/40000 [00:04<00:00, 10293.33 examples/s]

Map: 100% 5000/5000 [00:00<00:00, 10199.76 examples/s]

```
""" We print the head of each of the train/test/val sets to visualize our cleaned

print("\nSample training examples:")
display(df_train.head(5))

print("\nSample validation examples:")
display(df_val.head(5))

print("\nSample test examples:")
display(df_test.head(5))
```

Sample training examples:

	text	date	user	sentiment	query
47782	finally set up wireless internet huzzah for tw...	Sat May 30 23:43:31 PDT 2009	alexwilliamson	1	NO_QUERY
20407	lebron james please dont leave usfor the love ...	Sat May 30 20:58:04 PDT 2009	ryangetty	0	NO_QUERY
42997	i broke our site	Sat May 30 23:21:22 PDT 2009	DjDATZ	0	NO_QUERY
19678	ugh idk if thats going to be possible my frie...	Fri May 22 08:28:11 PDT 2009	JulieAnnCook	0	NO_QUERY
13754	wrong place at the wrong time always sigh	Wed Jun 17 08:20:28 PDT 2009	Nadiahazman	0	NO_QUERY

Sample validation examples:

	text	date	user	sentiment	query
42591	so am i	Tue Apr 21 01:41:39 PDT 2009	joelted	0	NO_QUERY
23532	had a change of mind i wont give email addr i ...	Sat May 30 20:57:14 PDT 2009	brianwelburn	0	NO_QUERY
49522	grrtrr up early	Mon May 11 23:22:36 PDT	StephanieLedigo	0	NO_QUERY

	text	date	user	sentiment	query
		2009			
48516	cleaning the house and packing for my journey ...	Sat May 30 12:38:45 PDT 2009	LoveandYoga	1	NO_QUERY
23853	my dad came up to our cabin yaaay we r up here...	Sat Jun 20 07:06:08 PDT 2009	ptwentzplswtch1	0	NO_QUERY

```
Sample test examples:
```

	text	date	user	sentiment	query
46754	ouch	Tue Apr 07 02:36:39 PDT 2009	paulrjmellors	0	NO_QUERY
	3 day long weekend	Thu May 28			

```
""" We initialize our dataloader for each of the sets, fix their batch sizes
and randomize their order"""

train_loader = DataLoader(tokenized_dataset["train"], batch_size=16, shuffle=True
val_loader   = DataLoader(tokenized_dataset["validation"], batch_size=16, shuffle
test_loader  = DataLoader(tokenized_dataset["test"], batch_size=16)
```

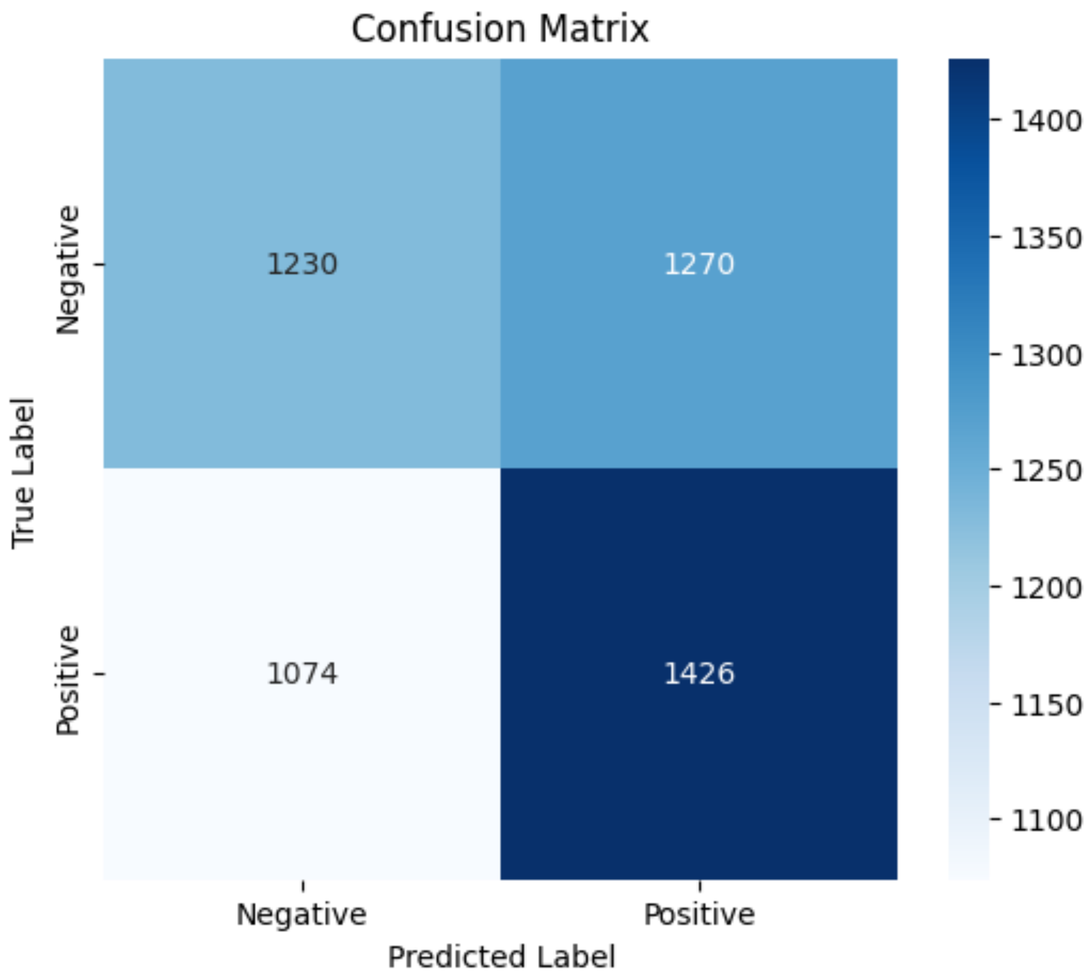
```
""" We define our optimizer using Adam and set a conservative learning rate and
weight decay (though later hyperparameter search will overwrite)"""

optimizer = torch.optim.AdamW(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=5e-5,
    weight_decay=0.01
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
OPTForSequenceClassification(
  (model): OPTModel(
    (decoder): OPTDecoder(
      (embed_tokens): Embedding(50272, 768, padding_idx=1)
      (embed_positions): OPTLearnedPositionalEmbedding(2050, 768)
      (final_layer_norm): LayerNorm((768,), eps=1e-05,
elementwise_affine=True)
      (layers): ModuleList(
        (0-11): 12 x OPTDecoderLayer(
          (self_attn): OPTSdpaAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
          (activation_fn): ReLU()
          (self_attn_layer_norm): LayerNorm((768,), eps=1e-05,
elementwise_affine=True)
          (fc1): Linear(in_features=768, out_features=3072, bias=True)
          (fc2): Linear(in_features=3072, out_features=768, bias=True)
          (final_layer_norm): LayerNorm((768,), eps=1e-05,
elementwise_affine=True)
        )
      )
    )
  )
  (score): Linear(in_features=768, out_features=2, bias=False)
)
```

```
""" Baseline inference for binary sentiment analysis task run on OPT-125m
without PEFT (i.e. without BitFit and/or LoRA)"""

import time
import torch
from sklearn.metrics import classification_report, confusion_matrix, f1_score
import seaborn as sns
```

```
import matplotlib.pyplot as plt

inference_start = time.time()

model.eval()
total_correct = 0
total_samples = 0
all_preds = []
all_labels = []

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["sentiment"].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)

        all_preds.extend(predictions.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

        total_correct += (predictions == labels).sum().item()
        total_samples += labels.size(0)

accuracy = total_correct / total_samples
f1_macro = f1_score(all_labels, all_preds, average="macro")
f1_weighted = f1_score(all_labels, all_preds, average="weighted")
inference_time = time.time() - inference_start

print(f"\nTest Accuracy   : {accuracy:.4f}")
print(f"F1 Score (macro): {f1_macro:.4f}")
print(f"F1 Score (weighted): {f1_weighted:.4f}")
print(f"Inference Time  : {inference_time:.2f}s")
print("\nClassification Report:")
print(classification_report(all_labels, all_preds, target_names=["Negative", "Posit

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative", "Positi
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Test Accuracy    : 0.5312
F1 Score (macro): 0.5305
F1 Score (weighted): 0.5305
Inference Time   : 27.51s

Classification Report:
              precision    recall  f1-score   support

    Negative       0.53      0.49      0.51      2500
    Positive       0.53      0.57      0.55      2500

    accuracy                           0.53      5000
   macro avg       0.53      0.53      0.53      5000
weighted avg       0.53      0.53      0.53      5000
```



LORA

```
""" Install Parameter Efficient Finetuning Packages (e.g. LoRA and BitFit)"""

!pip install peft
```

```
Requirement already satisfied: peft in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: accelerate>=0.21.0 in /usr/local/lib/python3.1
Requirement already satisfied: safetensors in /usr/local/lib/python3.11/dist-
Requirement already satisfied: huggingface-hub>=0.25.0 in /usr/local/lib/pyth
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/p
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/loc
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/l
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/l
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/li
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/pyt
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/p
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.1
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/pytho
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
```

```
""" Establish LoRA configuration """

from peft import get_peft_model, LoraConfig, TaskType

lora_config = LoraConfig(
    r=16,                             # LoRA rank (8 or 16 are common)
    lora_alpha=32,                    # Scaling factor
    lora_dropout=0.1,                 # Dropout for LoRA layers
    bias="none",                      # Don't train bias terms
    task_type=TaskType.SEQ_CLS,       # Task type: sequence classification
    target_modules=["q_proj", "v_proj"]  # or ["c_attn"] for GPT-2
)


model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

```
trainable params: 591,360 || all params: 125,832,192 || trainable%: 0.4700
```

```
""" Perform training of OPT-125m model with LoRA PEFT and save results to .csv ""

from itertools import product
from tqdm import tqdm
import time
import psutil
from sklearn.metrics import accuracy_score, f1_score

learning_rates = [5e-5, 1e-4]
batch_sizes = [8, 16]
num_epochs = 3

results = []
epoch_logs_all = []
best_overall_state = None
best_overall_config = {}

for lr, bs in product(learning_rates, batch_sizes):
    print(f"\n=== Training with learning rate = {lr}, batch size = {bs} ===")

    base_model = OPTForSequenceClassification.from_pretrained(model_name, num_lab
    model = get_peft_model(base_model, lora_config)
    model.print_trainable_parameters()

    optimizer = torch.optim.AdamW(filter(lambda p: p.requires_grad, model.paramet
```

```
train_loader = DataLoader(tokenized_dataset["train"], batch_size=bs, shuffle=
val_loader = DataLoader(tokenized_dataset["validation"], batch_size=bs)

best_val_f1 = 0.0
best_model_state = None
start_time = time.time()
start_mem = psutil.Process().memory_info().rss / 1024**2
epoch_logs = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for batch in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
        optimizer.zero_grad()
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["sentiment"].to(device)

        loss = model(input_ids=input_ids, attention_mask=attention_mask, labe
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    avg_loss = running_loss / len(train_loader)

    # Validation
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in val_loader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["sentiment"].to(device)

            logits = model(input_ids=input_ids, attention_mask=attention_mask
            preds = torch.argmax(logits, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    val_accuracy = accuracy_score(all_labels, all_preds)
    val_f1 = f1_score(all_labels, all_preds, average="macro")
    print(f"Epoch {epoch+1}/{num_epochs} - Loss: {avg_loss:.4f} | Val Acc: {v

    epoch_logs.append({
```

```
            "epoch": epoch + 1, "lr": lr, "batch_size": bs,
            "train_loss": avg_loss, "val_accuracy": val_accuracy, "val_f1": val_f
        })

        if val_f1 > best_val_f1:
            best_val_f1 = val_f1
            best_model_state = model.state_dict()

    end_time = time.time()
    end_mem = psutil.Process().memory_info().rss / 1024**2
    results.append({
        "lr": lr, "batch_size": bs, "best_val_f1": best_val_f1,
        "val_accuracy": val_accuracy,
        "runtime_sec": end_time - start_time,
        "memory_delta_mb": end_mem - start_mem
    })
    epoch_logs_all.extend(epoch_logs)

    if best_val_f1 > (best_overall_config or {}).get("best_val_f1", 0):
        best_overall_state = best_model_state
        best_overall_config = results[-1]

# Save
results_df = pd.DataFrame(results)
epoch_df = pd.DataFrame(epoch_logs_all)
results_df.to_csv("lora_results.csv", index=False)
epoch_df.to_csv("lora_epoch_logs.csv", index=False)
display(results_df.sort_values(by="best_val_f1", ascending=False))

# Final Test Evaluation
best_model = get_peft_model(
    OPTForSequenceClassification.from_pretrained(model_name, num_labels=num_label
    lora_config
)
best_model.load_state_dict(best_overall_state)

test_loader = DataLoader(tokenized_dataset["test"], batch_size=16)
best_model.eval()
all_preds, all_labels = [], []

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["sentiment"].to(device)
```

```
        preds = torch.argmax(best_model(input_ids=input_ids, attention_mask=atten
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

test_accuracy = accuracy_score(all_labels, all_preds)
test_f1 = f1_score(all_labels, all_preds, average="macro")
print(f"\nTest Accuracy: {test_accuracy:.4f} - Test F1 (macro): {test_f1:.4f}")
```

```
=== Training with learning rate = 5e-05, batch size = 8 ===
Some weights of OPTForSequenceClassification were not initialized from the moc
You should probably TRAIN this model on a down-stream task to be able to use i
trainable params: 591,360 || all params: 125,832,192 || trainable%: 0.4700
Epoch 1: 100%|██████████| 5000/5000 [09:09<00:00,  9.11it/s]
Epoch 1/3 - Loss: 0.4341 | Val Acc: 0.8284 | F1: 0.8282
Epoch 2: 100%|██████████| 5000/5000 [09:09<00:00,  9.10it/s]
Epoch 2/3 - Loss: 0.3723 | Val Acc: 0.8390 | F1: 0.8388
Epoch 3: 100%|██████████| 5000/5000 [09:08<00:00,  9.12it/s]
Epoch 3/3 - Loss: 0.3481 | Val Acc: 0.8392 | F1: 0.8391

=== Training with learning rate = 5e-05, batch size = 16 ===
Some weights of OPTForSequenceClassification were not initialized from the moc
You should probably TRAIN this model on a down-stream task to be able to use i
trainable params: 591,360 || all params: 125,832,192 || trainable%: 0.4700
Epoch 1: 100%|██████████| 2500/2500 [08:45<00:00,  4.76it/s]
Epoch 1/3 - Loss: 0.4453 | Val Acc: 0.8172 | F1: 0.8166
Epoch 2: 100%|██████████| 2500/2500 [08:45<00:00,  4.76it/s]
Epoch 2/3 - Loss: 0.3804 | Val Acc: 0.8358 | F1: 0.8358
Epoch 3: 100%|██████████| 2500/2500 [08:45<00:00,  4.76it/s]
Epoch 3/3 - Loss: 0.3608 | Val Acc: 0.8350 | F1: 0.8347

=== Training with learning rate = 0.0001, batch size = 8 ===
Some weights of OPTForSequenceClassification were not initialized from the moc
You should probably TRAIN this model on a down-stream task to be able to use i
trainable params: 591,360 || all params: 125,832,192 || trainable%: 0.4700
Epoch 1: 100%|██████████| 5000/5000 [09:05<00:00,  9.16it/s]
Epoch 1/3 - Loss: 0.4183 | Val Acc: 0.8326 | F1: 0.8326
Epoch 2: 100%|██████████| 5000/5000 [09:05<00:00,  9.16it/s]
Epoch 2/3 - Loss: 0.3585 | Val Acc: 0.8436 | F1: 0.8436
Epoch 3: 100%|██████████| 5000/5000 [09:05<00:00,  9.17it/s]
Epoch 3/3 - Loss: 0.3261 | Val Acc: 0.8400 | F1: 0.8397

=== Training with learning rate = 0.0001, batch size = 16 ===
Some weights of OPTForSequenceClassification were not initialized from the moc
You should probably TRAIN this model on a down-stream task to be able to use i
trainable params: 591,360 || all params: 125,832,192 || trainable%: 0.4700
Epoch 1: 100%|██████████| 2500/2500 [08:44<00:00,  4.76it/s]
```

```
Epoch 1/3 - Loss: 0.4300 | Val Acc: 0.8362 | F1: 0.8362
Epoch 2: 100%|██████████| 2500/2500 [08:45<00:00,  4.76it/s]
Epoch 2/3 - Loss: 0.3657 | Val Acc: 0.8410 | F1: 0.8409
Epoch 3: 100%|██████████| 2500/2500 [08:44<00:00,  4.76it/s]
Epoch 3/3 - Loss: 0.3371 | Val Acc: 0.8478 | F1: 0.8477
```

	lr	batch_size	best_val_f1	val_accuracy	runtime_sec	memory_delta_mb
3	0.00010	16	0.847686	0.8478	1667.428003	0.929688
2	0.00010	8	0.843577	0.8400	1731.900387	0.644531
0	0.00005	8	0.839095	0.8392	1742.056741	1.554688
1	0.00005	16	0.835800	0.8350	1668.280001	0.511719

```
Some weights of OPTForSequenceClassification were not initialized from the moc
You should probably TRAIN this model on a down-stream task to be able to use i

Test Accuracy: 0.8414 - Test F1 (macro): 0.8412
```

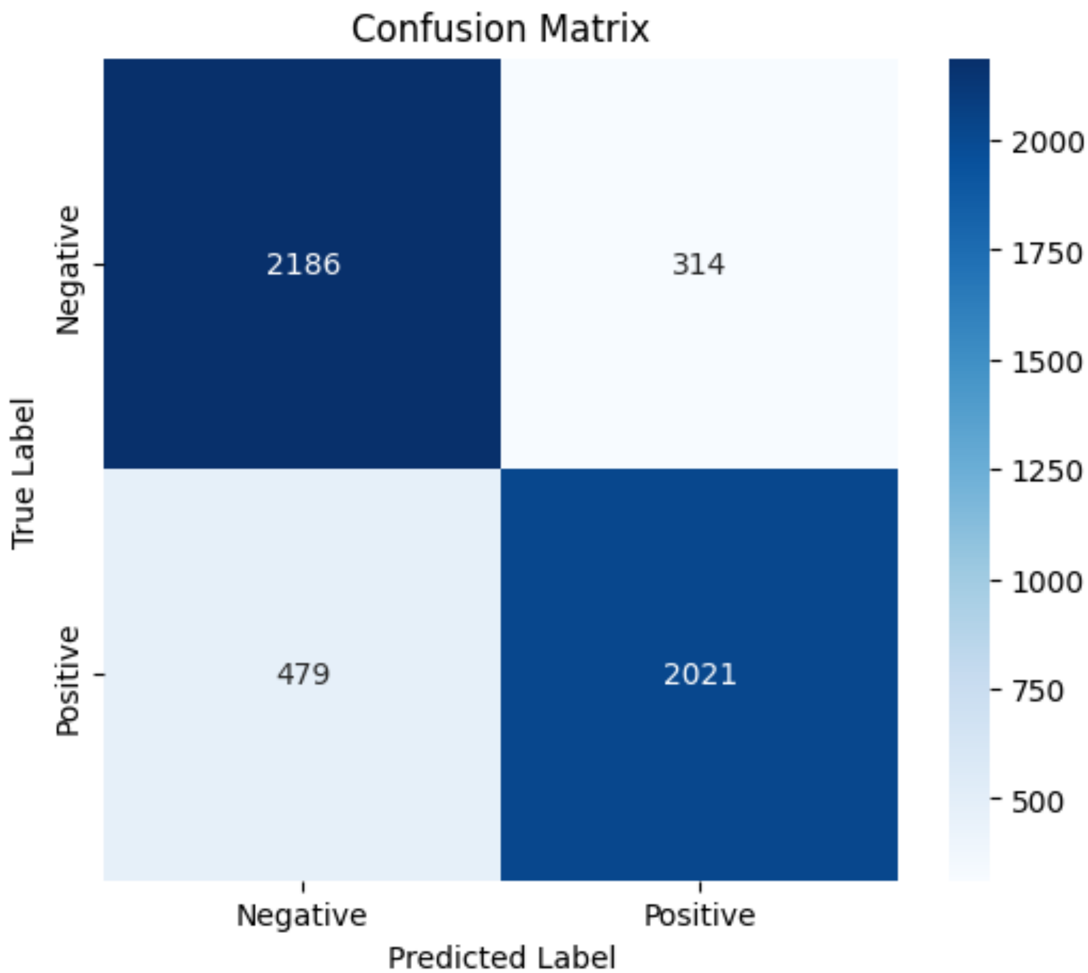
```
len(tokenized_dataset["validation"])
```

```
5000
```

```
""" Inference on trained OPT-125m model using LoRA """

import time
import torch
from sklearn.metrics import classification_report, confusion_matrix, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

inference_start = time.time()

model.eval()
total_correct = 0
total_samples = 0
all_preds = []
all_labels = []

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["sentiment"].to(device)
```

```
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)

        all_preds.extend(predictions.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

        total_correct += (predictions == labels).sum().item()
        total_samples += labels.size(0)

accuracy = total_correct / total_samples
f1_macro = f1_score(all_labels, all_preds, average="macro")
f1_weighted = f1_score(all_labels, all_preds, average="weighted")
inference_time = time.time() - inference_start

print(f"\nTest Accuracy   : {accuracy:.4f}")
print(f"F1 Score (macro): {f1_macro:.4f}")
print(f"F1 Score (weighted): {f1_weighted:.4f}")
print(f"Inference Time  : {inference_time:.2f}s")
print("\nClassification Report:")
print(classification_report(all_labels, all_preds, target_names=["Negative", "Posit

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative", "Positi
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Test Accuracy    : 0.8414
F1 Score (macro): 0.8412
F1 Score (weighted): 0.8412
Inference Time   : 31.11s

Classification Report:
              precision    recall  f1-score   support

    Negative       0.82      0.87      0.85      2500
    Positive       0.87      0.81      0.84      2500

    accuracy                           0.84      5000
   macro avg       0.84      0.84      0.84      5000
weighted avg       0.84      0.84      0.84      5000
```



```
""" Save inference results and confusion matrix to .csv for visual generation"""

import pandas as pd

# Save prediction results
df_preds = pd.DataFrame({
    "true_label": all_labels,
    "predicted_label": all_preds
})
df_preds.to_csv("lora_inference_predictions.csv", index=False)

# Save confusion matrix
cm_df = pd.DataFrame(
    cm,
    index=["Actual Negative", "Actual Positive"],
    columns=["Predicted Negative", "Predicted Positive"]
)
cm_df.to_csv("lora_inference_confusion_matrix.csv")

# Save summary stats
metrics_summary = {
    "accuracy": [accuracy],
    "f1_macro": [f1_macro],
    "f1_weighted": [f1_weighted],
    "inference_time_sec": [inference_time]
}
df_metrics = pd.DataFrame(metrics_summary)
df_metrics.to_csv("lora_inference_metrics_summary.csv", index=False)


""" Resource tracking (for memory-intensive comparison with baseline and BitFit)"

if torch.cuda.is_available():
    max_memory = torch.cuda.max_memory_allocated(device) / (1024 ** 2)
    print(f"Max GPU memory used: {max_memory:.2f} MB")
```

```
Max GPU memory used: 5070.27 MB
```

BITFIT

```
model_name = "facebook/opt-125m"
num_labels = 2
tokenizer = AutoTokenizer.from_pretrained(model_name, token=token)


""" Function to bypass weight gradients (BitFit only updates biases)"""

def apply_bitfit(model):
  for name, param in model.named_parameters():
    param.requires_grad = "bias" in name


""" Training on OPT-125m model using BitFit and output dataset generation (saved 

from itertools import product
from tqdm import tqdm
import time
import psutil
from sklearn.metrics import accuracy_score, f1_score
from transformers import logging

logging.set_verbosity_error()

# Range of hyperparameters to explore for optimization
learning_rates = [5e-5, 1e-4]
batch_sizes = [8, 16]
num_epochs = 3

best_overall_state = None
best_overall_config = {}

results = []

for lr, bs in product(learning_rates, batch_sizes):
    print(f"\n=== Training with learning rate = {lr}, batch size = {bs} ===")

    # Re-initialize model and optimizer for each run
    model = OPTForSequenceClassification.from_pretrained(model_name, num_labels=n

    apply_bitfit(model)

    optimizer = torch.optim.AdamW(filter(lambda p: p.requires_grad, model.paramet
    train_loader = DataLoader(tokenized_dataset["train"], batch_size=bs, shuffle=
    val_loader   = DataLoader(tokenized_dataset["validation"], batch_size=bs)
```

```
    best_val_f1 = 0.0
    best_model_state = None
    start_time = time.time()
    start_mem = psutil.Process().memory_info().rss / 1024**2  # MB

    epoch_logs = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for batch in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
            optimizer.zero_grad()
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["sentiment"].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, l
            loss = outputs.loss

            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_loss = running_loss / len(train_loader)
        print(f"Epoch {epoch+1}/{num_epochs} - Training Loss: {avg_loss:.4f}")

        # Validation
        model.eval()
        all_preds = []
        all_labels = []

        with torch.no_grad():
            for batch in val_loader:
                input_ids = batch["input_ids"].to(device)
                attention_mask = batch["attention_mask"].to(device)
                labels = batch["sentiment"].to(device)

                outputs = model(input_ids=input_ids, attention_mask=attention_mas
                preds = torch.argmax(outputs.logits, dim=1)

                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        val_accuracy = accuracy_score(all_labels, all_preds)
```

```
        val_f1 = f1_score(all_labels, all_preds, average="macro")
        print(f"Validation Accuracy: {val_accuracy:.4f} - F1 (macro): {val_f1:.4f

        epoch_logs.append({
            "epoch": epoch + 1,
            "lr": lr,
            "batch_size": bs,
            "train_loss": avg_loss,
            "val_accuracy": val_accuracy,
            "val_f1": val_f1
        })

        if val_f1 > best_val_f1:
            best_val_f1 = val_f1
            best_model_state = model.state_dict()

    end_time = time.time()
    end_mem = psutil.Process().memory_info().rss / 1024**2

    results.append({
        "lr": lr,
        "batch_size": bs,
        "best_val_f1": best_val_f1,
        "val_accuracy": val_accuracy,
        "runtime_sec": end_time - start_time,
        "memory_delta_mb": end_mem - start_mem
    })

    if best_val_f1 > (best_overall_config or {}).get("best_val_f1", 0):
        best_overall_state = best_model_state
        best_overall_config = results[-1]

# Display results
import pandas as pd
results_df = pd.DataFrame(results)
display(results_df.sort_values(by="best_val_f1", ascending=False))


results_df.to_csv("bitfit_results.csv", index=False)

print(f"\nBest Hyperparameters -> lr: {best_overall_config['lr']}, batch_size: {b


# Optional final evaluation
best_model = OPTForSequenceClassification.from_pretrained(model_name, num_labels=
```

```
apply_bitfit(best_model)
best_model.load_state_dict(best_overall_state)

test_loader = DataLoader(tokenized_dataset["test"], batch_size=16)
best_model.eval()
all_preds, all_labels = [], []

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["sentiment"].to(device)

        outputs = best_model(input_ids=input_ids, attention_mask=attention_mask)
        preds = torch.argmax(outputs.logits, dim=1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

test_accuracy = accuracy_score(all_labels, all_preds)
test_f1 = f1_score(all_labels, all_preds, average="macro")
print(f"\nTest Accuracy: {test_accuracy:.4f} - Test F1 (macro): {test_f1:.4f}")


print(f"\nTest Accuracy (best lr={best_overall_config['lr']}, bs={best_overall_co

df_epochs = pd.DataFrame(epoch_logs)
df_epochs.to_csv("bitfit_epoch_logs.csv", index=False)

print("\nSaved per-epoch logs to bitfit_epoch_logs.csv")
```

```
=== Training with learning rate = 5e-05, batch size = 8 ===
Epoch 1: 100%|██████████| 5000/5000 [08:51<00:00,  9.41it/s]
Epoch 1/3 - Training Loss: 0.4851
Validation Accuracy: 0.8084 - F1 (macro): 0.8082
Epoch 2: 100%|██████████| 5000/5000 [08:47<00:00,  9.49it/s]
Epoch 2/3 - Training Loss: 0.4203
Validation Accuracy: 0.8152 - F1 (macro): 0.8150
Epoch 3: 100%|██████████| 5000/5000 [08:46<00:00,  9.50it/s]
Epoch 3/3 - Training Loss: 0.4018
Validation Accuracy: 0.8254 - F1 (macro): 0.8254

=== Training with learning rate = 5e-05, batch size = 16 ===
Epoch 1: 100%|██████████| 2500/2500 [08:26<00:00,  4.94it/s]
Epoch 1/3 - Training Loss: 0.5051
Validation Accuracy: 0.8048 - F1 (macro): 0.8044
```

```
Epoch 2: 100%|██████████| 2500/2500 [08:26<00:00,  4.94it/s]
Epoch 2/3 - Training Loss: 0.4231
Validation Accuracy: 0.8032 - F1 (macro): 0.8015
Epoch 3: 100%|██████████| 2500/2500 [08:26<00:00,  4.94it/s]
Epoch 3/3 - Training Loss: 0.4071
Validation Accuracy: 0.8050 - F1 (macro): 0.8031

=== Training with learning rate = 0.0001, batch size = 8 ===
Epoch 1: 100%|██████████| 5000/5000 [08:46<00:00,  9.50it/s]
Epoch 1/3 - Training Loss: 0.4745
Validation Accuracy: 0.8126 - F1 (macro): 0.8122
Epoch 2: 100%|██████████| 5000/5000 [08:44<00:00,  9.53it/s]
Epoch 2/3 - Training Loss: 0.4181
Validation Accuracy: 0.8270 - F1 (macro): 0.8270
Epoch 3: 100%|██████████| 5000/5000 [08:46<00:00,  9.50it/s]
Epoch 3/3 - Training Loss: 0.4003
Validation Accuracy: 0.8312 - F1 (macro): 0.8312

=== Training with learning rate = 0.0001, batch size = 16 ===
Epoch 1: 100%|██████████| 2500/2500 [08:26<00:00,  4.93it/s]
Epoch 1/3 - Training Loss: 0.4665
Validation Accuracy: 0.8150 - F1 (macro): 0.8150
Epoch 2: 100%|██████████| 2500/2500 [08:26<00:00,  4.93it/s]
Epoch 2/3 - Training Loss: 0.4109
Validation Accuracy: 0.8134 - F1 (macro): 0.8125
Epoch 3: 100%|██████████| 2500/2500 [08:26<00:00,  4.93it/s]
Epoch 3/3 - Training Loss: 0.3991
Validation Accuracy: 0.8160 - F1 (macro): 0.8158
```

	lr	batch_size	best_val_f1	val_accuracy	runtime_sec	memory_delta_mb
2	0.00010	8	0.831165	0.8312	1667.348322	0.000000
0	0.00005	8	0.825385	0.8254	1675.313485	0.132812
3	0.00010	16	0.815767	0.8160	1607.933084	0.000000
1	0.00005	16	0.804386	0.8050	1606.963252	0.136719

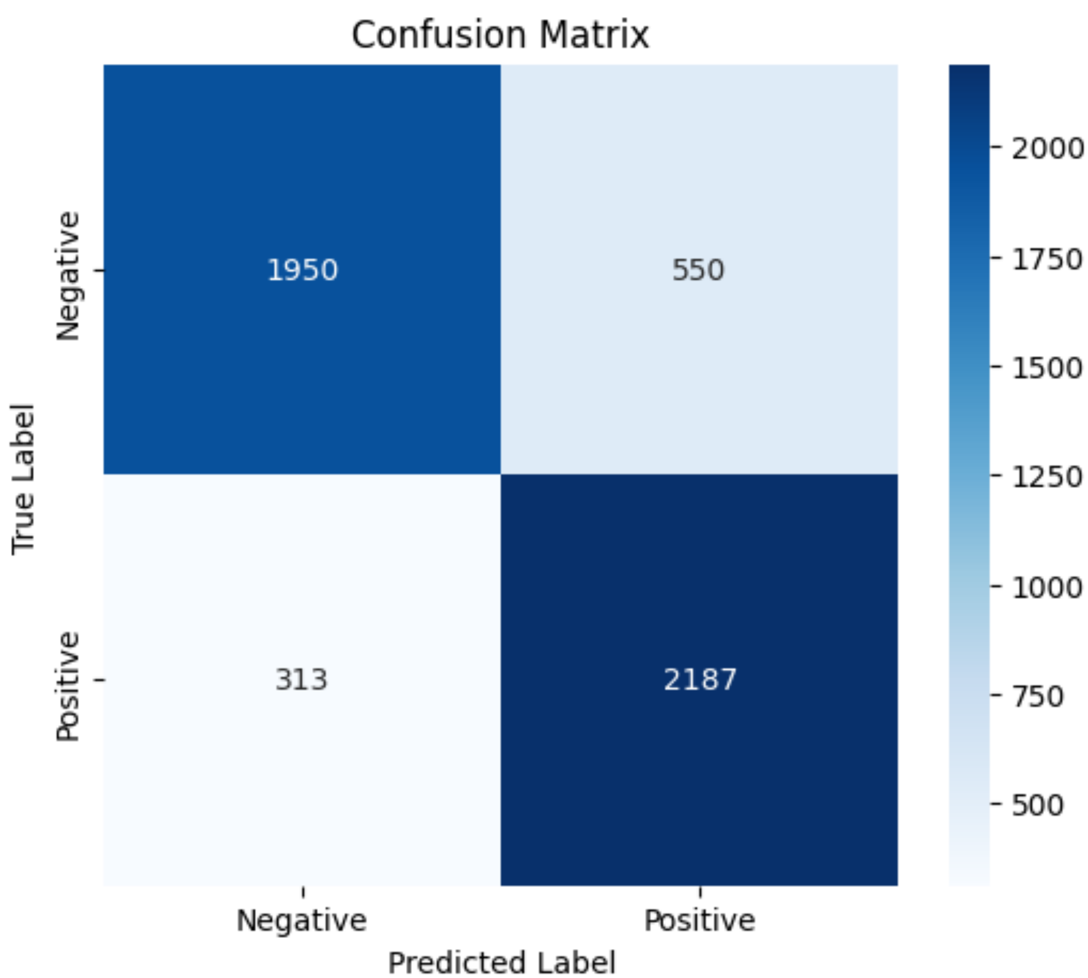
```
Best Hyperparameters -> lr: 0.0001, batch_size: 8

Test Accuracy: 0.8260 - Test F1 (macro): 0.8260

Test Accuracy (best lr=0.0001, bs=8): 0.8260 - Test F1 (macro): 0.8260

Saved per-epoch logs to bitfit_epoch_logs.csv
```

```
""" Inference task for OPT125m pre-trained with BitFit PEFT and confusion matrix"""
```

```
import time
import torch
from sklearn.metrics import classification_report, confusion_matrix, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

inference_start = time.time()

model.eval()
total_correct = 0
total_samples = 0
all_preds = []
all_labels = []

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["sentiment"].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=-1)

        all_preds.extend(predictions.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

        total_correct += (predictions == labels).sum().item()
        total_samples += labels.size(0)

accuracy = total_correct / total_samples
f1_macro = f1_score(all_labels, all_preds, average="macro")
f1_weighted = f1_score(all_labels, all_preds, average="weighted")
inference_time = time.time() - inference_start

print(f"\nTest Accuracy   : {accuracy:.4f}")
print(f"F1 Score (macro): {f1_macro:.4f}")
print(f"F1 Score (weighted): {f1_weighted:.4f}")
print(f"Inference Time  : {inference_time:.2f}s")
print("\nClassification Report:")
print(classification_report(all_labels, all_preds, target_names=["Negative", "Posit

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative", "Positi
```

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Test Accuracy    : 0.8274
F1 Score (macro): 0.8270
F1 Score (weighted): 0.8270
Inference Time   : 28.51s

Classification Report:
              precision    recall  f1-score   support

    Negative       0.86      0.78      0.82      2500
    Positive       0.80      0.87      0.84      2500

    accuracy                           0.83      5000
   macro avg       0.83      0.83      0.83      5000
weighted avg       0.83      0.83      0.83      5000
```



```
""" Dataset creation for BitFit results, saved to .csv for later visual generation'

import pandas as pd

# Save prediction results (true vs predicted)
df_preds = pd.DataFrame({
    "true_label": all_labels,
    "predicted_label": all_preds
})
df_preds.to_csv("bitfit_inference_predictions.csv", index=False)

# Save confusion matrix as table
cm_df = pd.DataFrame(
    cm,
    index=["Actual Negative", "Actual Positive"],
    columns=["Predicted Negative", "Predicted Positive"]
)
cm_df.to_csv("bitfit_inference_confusion_matrix.csv")

# Save summary stats (for accuracy, F1, inference time)
metrics_summary = {
    "accuracy": [accuracy],
    "f1_macro": [f1_macro],
    "f1_weighted": [f1_weighted],
    "inference_time_sec": [inference_time]
}
df_metrics = pd.DataFrame(metrics_summary)
df_metrics.to_csv("bitfit_inference_metrics_summary.csv", index=False)
```

```
""" Logging of memory usage for comparison of BitFit against baseline and LoRA """

if torch.cuda.is_available():
    max_memory = torch.cuda.max_memory_allocated(device) / (1024 ** 2)
    print(f"Max GPU memory used: {max_memory:.2f} MB")
```

```
Max GPU memory used: 5070.27 MB
```

Start coding or generate with AI.