# GPT2 Sentiment Analysis Experiments on Sentiment 140 Dataset using baseline model (and with PEFT -- LoRA, BitFit, Prompt Tuning)

```
"""We begin our process by installing packages such as pytorch, which is used ext
transformers and datasets packages, which are used to run the GPT2 transformer mo

!pip install torch transformers datasets -q
```

```
363.4/363.4 MB 4.2 MB/s eta 0:00:(
13.8/13.8 MB 96.6 MB/s eta 0:00:0(
24.6/24.6 MB 80.4 MB/s eta 0:00:0(
883.7/883.7 kB 50.6 MB/s eta 0:00
664.8/664.8 MB 1.7 MB/s eta 0:00:(
211.5/211.5 MB 11.4 MB/s eta 0:00
56.3/56.3 MB 42.6 MB/s eta 0:00:0(
127.9/127.9 MB 19.4 MB/s eta 0:00
207.5/207.5 MB 4.9 MB/s eta 0:00:(
21.1/21.1 MB 108.4 MB/s eta 0:00:(
491.2/491.2 kB 35.9 MB/s eta 0:00
116.3/116.3 kB 10.7 MB/s eta 0:00
183.9/183.9 kB 15.3 MB/s eta 0:00
143.5/143.5 kB 13.5 MB/s eta 0:00
194.8/194.8 kB 12.1 MB/s eta 0:00
ERROR: pip's dependency resolver does not currently take into account all the
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2024.12.0 which
```

```
"""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 GPT2 model"""

import torch
from torch.utils.data import DataLoader
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, DataC

import time
from sklearn.metrics import classification_report


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

dataset = load_dataset("/content/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 to
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 [00:05<00:00, 22.6MB/s]

Generating train split: 1600000/0 [00:50<00:00, 31417.54 examples/s]

Generating test split: 498/0 [00:00<00:00, 17652.47 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 [00:03<00:00, 14628.93 examples/

```
model_name = "gpt2"
num_labels = 2
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: Use
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pu
  warnings.warn(
```

tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 3.36kB/s]

config.json: 100% 665/665 [00:00<00:00, 75.2kB/s]

vocab.json: 100% 1.04M/1.04M [00:00<00:00, 2.42MB/s]

merges.txt: 100% 456k/456k [00:00<00:00, 41.2MB/s]

tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 20.9MB/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
```

model.safetensors: 100% 548M/548M [00:01<00:00, 559MB/s]

```
Some weights of GPT2ForSequenceClassification were not initialized from the mo
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 [00:03<00:00, 16894.08 examples/s]

```
Dataset size: 50000
```

DOWNSAMPLING FOR TRAINING

```
""" We next perform downsampling and stratified splitting of the data 80/20 Train

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

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

```
train_dataset = Dataset.from_pandas(df_train).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)
test_dataset  = test_dataset.map(tokenize, batched=True)

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

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

```
Initial class distribution: Counter({1: 25000, 0: 25000})
Train size: 40000
Test size: 10000
Train class distribution: Counter({1: 20000, 0: 20000})
Test class distribution: Counter({0: 5000, 1: 5000})
```

Map: 100% 40000/40000 [00:03<00:00, 15588.13 examples/s]

Map: 100% 10000/10000 [00:00<00:00, 15624.07 examples/

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

print("\nSample training examples:")
display(df_train.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 test examples:
```

| text | date | user | sentiment | query |
|---|---|---|---|---|

```
""" 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
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)
```

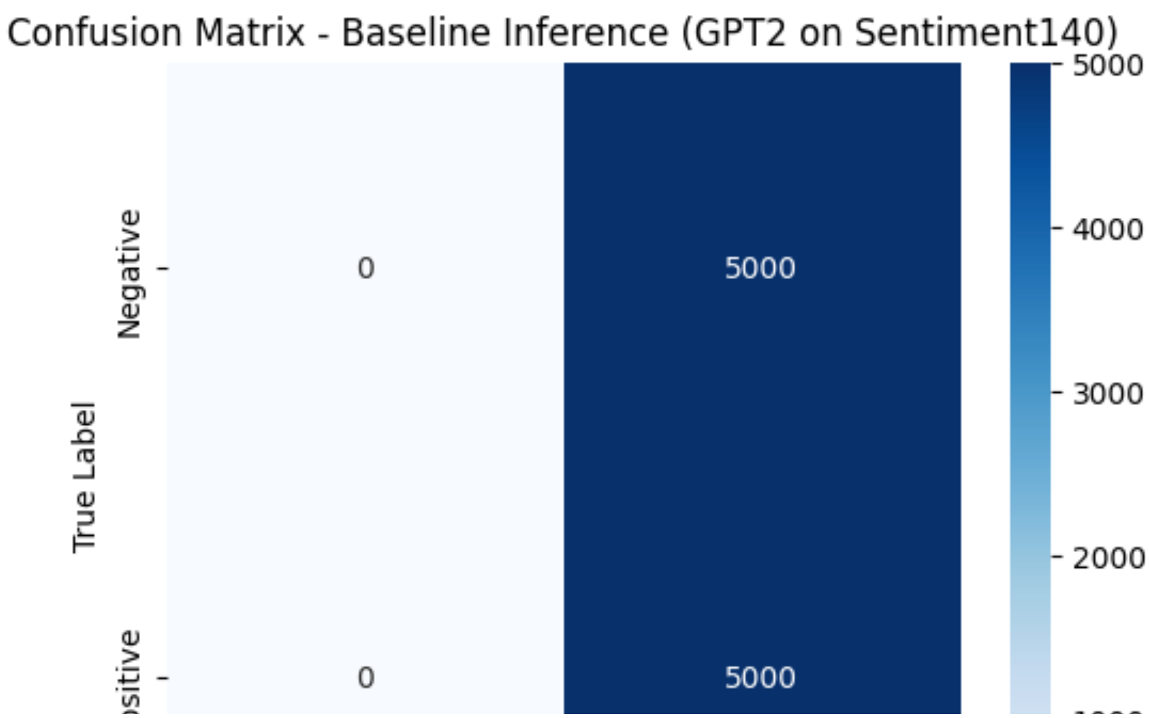
```
GPT2ForSequenceClassification(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D(nf=2304, nx=768)
          (c_proj): Conv1D(nf=768, nx=768)
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D(nf=3072, nx=768)
          (c_proj): Conv1D(nf=768, nx=3072)
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): 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 GPT2
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
from torch import autocast

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)

        with autocast(device_type='cuda'):
            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'\nBaseline Inference Performance - GPT2 on Sentiment140\n')
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", "Pos

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

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Baseline Inference (GPT2 on Sentiment140)")
plt.show()
```

```
Baseline Inference Performance - GPT2 on Sentiment140


Test Accuracy   : 0.5000
F1 Score (macro): 0.3333
F1 Score (weighted): 0.3333
Inference Time  : 10.26s

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00      5000
    Positive       0.50      1.00      0.67      5000

    accuracy                           0.50     10000
   macro avg       0.25      0.50      0.33     10000
weighted avg       0.25      0.50      0.33     10000

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```




# LORA

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

!pip install peft -q
```

```
""" Importing LoRA packages """

import gc
import torch
import time
import pandas as pd
from tqdm import tqdm
from transformers import AutoModelForSequenceClassification, AutoTokenizer, DataC
from peft import get_peft_model, LoraConfig, TaskType
from sklearn.metrics import classification_report, f1_score
from torch.utils.data import DataLoader
```

```
""" LoRA parameter setup """

learning_rates = [5e-5, 1e-4]
batch_sizes = [8, 16]
epochs = 6
```

```
""" Training on GPT2 model using LoRA and output dataset generation (saved as .cs

results = []

for lr in learning_rates:
    for batch_size in batch_sizes:
```

```
    print(f"Running LoRA with LR={lr}, batch_size={batch_size}")

    # loading GPT2 model
    model_name = "gpt2"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForSequenceClassification.from_pretrained(model_name, nu

    # LoRA param update config
    lora_config = LoraConfig(
        task_type=TaskType.SEQ_CLS,
        r=16,
        lora_alpha=32,
        lora_dropout=0.1,
        bias="none",
        target_modules=["c_attn", "c_proj"]
    )

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    # instantiate dataloader
    data_collator = DataCollatorWithPadding(tokenizer)
    train_dataloader = DataLoader(tokenized_dataset["train"], batch_size=batc
    test_dataloader = DataLoader(tokenized_dataset["test"], batch_size=batch_

    # adam optimizer
    optimizer = torch.optim.AdamW(filter(lambda p: p.requires_grad, model.par

    # begin training
    model.train()
    start_time = time.time()
    epoch_logs = []

    for epoch in range(1, epochs + 1):
        running_loss = 0.0
        correct = 0
        total = 0
        loop = tqdm(train_dataloader, leave=False)
        for step, batch in enumerate(loop):
            batch = {
                "input_ids": batch["input_ids"].to(device),
                "attention_mask": batch["attention_mask"].to(device),
                "labels": batch["sentiment"].to(device)
            }
```

```
            outputs = model(**batch)
            loss = outputs.loss
            preds = torch.argmax(outputs.logits, dim=1)
            correct += (preds == batch['labels']).sum().item()
            total += batch['labels'].size(0)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_train_loss = running_loss / (step + 1)
        train_accuracy = correct / total

        # perform per epoch evaluation
        model.eval()
        val_running_loss = 0.0
        y_true, y_pred = [], []
        inference_start = time.time()
        with torch.no_grad():
            for batch in test_dataloader:
                batch = {
                "input_ids": batch["input_ids"].to(device),
                "attention_mask": batch["attention_mask"].to(device),
                "labels": batch["sentiment"].to(device)
                }
                outputs = model(**batch)
                preds = torch.argmax(outputs.logits, dim=1)
                y_true.extend(batch["labels"].cpu().numpy())
                y_pred.extend(preds.cpu().numpy())
                val_running_loss += outputs.loss.item()

        avg_val_loss = val_running_loss / len(test_dataloader)
        inference_time = time.time() - inference_start

        report = classification_report(y_true, y_pred, output_dict=True)
        val_accuracy = report["accuracy"]
        val_f1 = report["weighted avg"]["f1-score"]

        epoch_logs.append({
            "epoch": epoch,
            "lr": lr,
            "batch_size": batch_size,
            "train_loss": avg_train_loss,
```

```
            "train_accuracy": train_accuracy,
            "val_loss": avg_val_loss,
            "val_accuracy": val_accuracy
        })

        if epoch == epochs:
            total_correct = sum(yt == yp for yt, yp in zip(y_true, y_pred))
            total_samples = len(y_true)
            accuracy = total_correct / total_samples
            f1_macro = f1_score(y_true, y_pred, average="macro")
            f1_weighted = f1_score(y_true, y_pred, average="weighted")

            print(f"\n[Final Epoch {epoch}] Inference Metrics:")
            print(f"Test 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} seconds")
            print("\nClassification Report: GPT2 w/ LoRA on Sentiment140\n")
            print(classification_report(y_true, y_pred, target_names=["Negati

        model.train()

    end_time = time.time()
    training_time = end_time - start_time

    # begin datalogging per lr/bs
    epoch_logs_df = pd.DataFrame(epoch_logs)
    epoch_logs_df.to_csv(f"sent_gpt2_lora_epoch_logs_lr{lr}_bs{batch_size}.cs

    # saver inference metrics per lr/bs
    metrics_summary_df = pd.DataFrame(report).transpose()
    metrics_summary_df.to_csv(f"sent_gpt2_lora_inference_metrics_summary_lr{l

    # save inference predictions for the final epoch
    predictions_df = pd.DataFrame({
        "y_true": y_true,
        "y_pred": y_pred
    })
    predictions_df.to_csv(f"sent_gpt2_lora_inference_predictions_lr{lr}_bs{ba

    # log memory usage
    max_memory = torch.cuda.max_memory_allocated() / (1024 ** 3) if torch.cud

    # save model params and metrics
    results.append({
```

```
            "method": "LoRA",
            "learning_rate": lr,
            "batch_size": batch_size,
            "accuracy": val_accuracy,
            "f1": val_f1,
            "training_time": training_time,
            "inference_time": inference_time,
            "max_memory": max_memory
        })

        # empty cache to conserve compute
        del model, tokenizer, optimizer
        torch.cuda.empty_cache()
        gc.collect()

# ranked performance by val acc
results = sorted(results, key=lambda x: x["accuracy"], reverse=True)

# save overall results
results_df = pd.DataFrame(results)
results_df.to_csv("sent_gpt2_lora_results.csv", index=False)

# save best final config and metrics
final_summary_df = pd.DataFrame({
    "Method": ["LoRA"],
    "Best LR": [results[0]["learning_rate"]],
    "Best Batch Size": [results[0]["batch_size"]],
    "Accuracy": [results[0]["accuracy"]],
    "F1 Score": [results[0]["f1"]],
    "Training Time (s)": [results[0]["training_time"]],
    "Inference Time (s)": [results[0]["inference_time"]],
    "Max GPU Memory (GB)": [results[0]["max_memory"]]
})
final_summary_df.to_csv("sent_gpt2_lora_final_comparison_lora.csv", index=False)

print("All LoRA Grid Search Results:")
for r in results:
    print(r)

print("\nBest LoRA Configuration:")
print(results[0])
```

```
Running LoRA with LR=5e-05, batch_size=8
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i
```

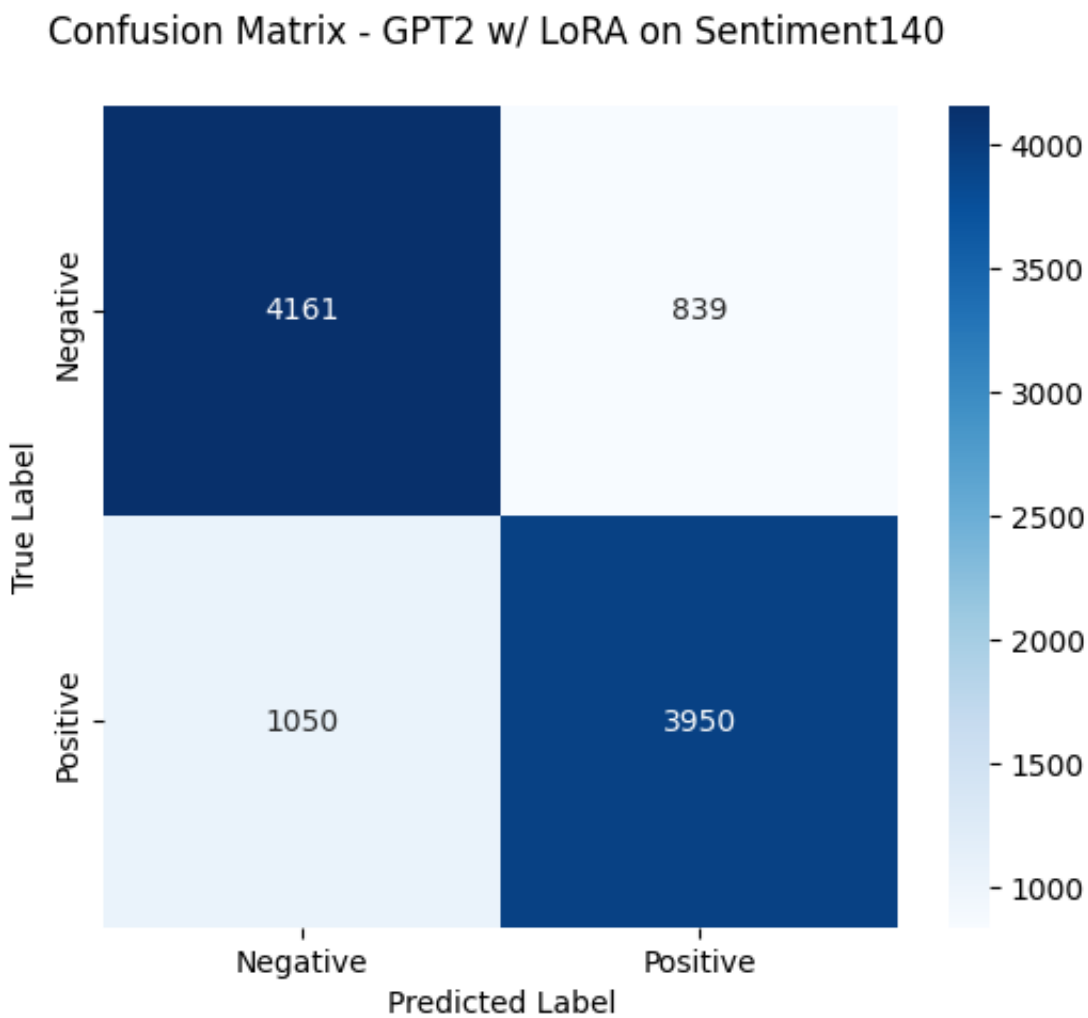
```
[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8062
F1 Score (macro)   : 0.8062
F1 Score (weighted): 0.8062
Inference Time     : 20.11 seconds

Classification Report: GPT2 w/ LoRA on Sentiment140

              precision    recall  f1-score   support

    Negative       0.80      0.81      0.81      5000
    Positive       0.81      0.80      0.80      5000

    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.81      0.81      0.81     10000

Running LoRA with LR=5e-05, batch_size=16
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8111
F1 Score (macro)   : 0.8110
F1 Score (weighted): 0.8110
Inference Time     : 18.98 seconds

Classification Report: GPT2 w/ LoRA on Sentiment140

              precision    recall  f1-score   support

    Negative       0.80      0.83      0.82      5000
    Positive       0.82      0.79      0.81      5000

    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.81      0.81      0.81     10000

Running LoRA with LR=0.0001, batch_size=8
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7879
F1 Score (macro)   : 0.7876
F1 Score (weighted): 0.7876
Inference Time     : 19.59 seconds

Classification Report: GPT2 w/ LoRA on Sentiment140
```

```
              precision    recall  f1-score   support

    Negative       0.81      0.75      0.78      5000
    Positive       0.77      0.83      0.80      5000
```

```
lora_best_lr = results[0]["learning_rate"]
lora_best_bs = results[0]["batch_size"]

# Construct filename
best_report_file = f"sent_gpt2_lora_inference_metrics_summary_lr{lora_best_lr}_bs

# Load the saved best report
best_report_df = pd.read_csv(best_report_file)
print("\nClassification Report for Best Configuration:")
print(best_report_df)



best_preds_df = pd.read_csv(f"sent_gpt2_lora_inference_predictions_lr{lora_best_l
print("\nInference Predictions for Best Configuration:")
print(best_preds_df)

y_true = best_preds_df["y_true"]
y_pred = best_preds_df["y_pred"]



cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative", "Posi
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - GPT2 w/ LoRA on Sentiment140 \n")
plt.show()
```

```
Classification Report for Best Configuration:
     Unnamed: 0  precision  recall  f1-score     support
0             0   0.798503  0.8322  0.815003   5000.0000
1             1   0.824807  0.7900  0.807028   5000.0000
2      accuracy   0.811100  0.8111  0.811100      0.8111
3     macro avg   0.811655  0.8111  0.811016  10000.0000
4  weighted avg   0.811655  0.8111  0.811016  10000.0000

Inference Predictions for Best Configuration:
      y_true  y_pred
0          0       0
```

```
1          0        0
2          1        1
3          0        0
4          0        0
...      ...      ...
9995       0        0
9996       0        0
9997       0        0
9998       1        1
9999       0        0

[10000 rows x 2 columns]
```




# BITFIT

```
""" Importing BitFit packages """

import gc
import torch
import time
import pandas as pd
from tqdm import tqdm
from transformers import AutoModelForSequenceClassification, AutoTokenizer, DataC
from peft import get_peft_model, LoraConfig, TaskType
from sklearn.metrics import classification_report, f1_score
from torch.utils.data import DataLoader


""" BitFit parameter setup """
learning_rates = [5e-5, 1e-4]
batch_sizes = [8, 16]
epochs = 6


""" Training on GPT2 model using BitFit and output dataset generation (saved as .

results = []

for lr in learning_rates:
    for batch_size in batch_sizes:
        print(f"Running BitFit with LR={lr}, batch_size={batch_size}")

        # loading GPT2 model
        model_name = "gpt2"
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        tokenizer.pad_token = tokenizer.eos_token
        model = AutoModelForSequenceClassification.from_pretrained(model_name, nu

        # BitFit param update config
        for name, param in model.named_parameters():
            if "bias" in name:
                param.requires_grad = True
            else:
                param.requires_grad = False

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        model.to(device)

        # instantiate dataloader
        data_collator = DataCollatorWithPadding(tokenizer)
```

```
    train_dataloader = DataLoader(tokenized_dataset["train"], batch_size=batc
    test_dataloader = DataLoader(tokenized_dataset["test"], batch_size=batch_

    # adam optimizer
    optimizer = torch.optim.AdamW(filter(lambda p: p.requires_grad, model.par

    # begin training
    model.train()
    start_time = time.time()
    epoch_logs = []

    for epoch in range(1, epochs + 1):
        running_loss = 0.0
        correct = 0
        total = 0
        loop = tqdm(train_dataloader, leave=False)
        for step, batch in enumerate(loop):
            batch = {
                "input_ids": batch["input_ids"].to(device),
                "attention_mask": batch["attention_mask"].to(device),
                "labels": batch["sentiment"].to(device)
            }
            outputs = model(**batch)
            loss = outputs.loss
            preds = torch.argmax(outputs.logits, dim=1)
            correct += (preds == batch['labels']).sum().item()
            total += batch['labels'].size(0)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_train_loss = running_loss / (step + 1)
        train_accuracy = correct / total

        # perform per epoch evaluation
        model.eval()
        val_running_loss = 0.0
        y_true, y_pred = [], []
        inference_start = time.time()
        with torch.no_grad():
            for batch in test_dataloader:
                batch = {
```

```
                "input_ids": batch["input_ids"].to(device),
                "attention_mask": batch["attention_mask"].to(device),
                "labels": batch["sentiment"].to(device)
                }
                outputs = model(**batch)
                preds = torch.argmax(outputs.logits, dim=1)
                y_true.extend(batch["labels"].cpu().numpy())
                y_pred.extend(preds.cpu().numpy())
                val_running_loss += outputs.loss.item()

        avg_val_loss = val_running_loss / len(test_dataloader)

        inference_time = time.time() - inference_start

        report = classification_report(y_true, y_pred, output_dict=True)
        val_accuracy = report["accuracy"]
        val_f1 = report["weighted avg"]["f1-score"]

        epoch_logs.append({
            "epoch": epoch,
            "lr": lr,
            "batch_size": batch_size,
            "train_loss": avg_train_loss,
            "train_accuracy": train_accuracy,
            "val_loss": avg_val_loss,
            "val_accuracy": val_accuracy
        })

        if epoch == epochs:
            total_correct = sum(yt == yp for yt, yp in zip(y_true, y_pred))
            total_samples = len(y_true)
            accuracy = total_correct / total_samples
            f1_macro = f1_score(y_true, y_pred, average="macro")
            f1_weighted = f1_score(y_true, y_pred, average="weighted")

            print(f"\n[Final Epoch {epoch}] Inference Metrics:")
            print(f"Test 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} seconds")
            print("\nClassification Report: GPT2 w/ BitFit on Sentiment140\n"
            print(classification_report(y_true, y_pred, target_names=["Negati

        model.train()
```

```
        end_time = time.time()
        training_time = end_time - start_time

        # begin datalogging per lr/bs
        epoch_logs_df = pd.DataFrame(epoch_logs)
        epoch_logs_df.to_csv(f"sent_gpt2_bitfit_epoch_logs_lr{lr}_bs{batch_size}.

        # saver inference metrics per lr/bs
        metrics_summary_df = pd.DataFrame(report).transpose()
        metrics_summary_df.to_csv(f"sent_gpt2_bitfit_inference_metrics_summary_lr

        # save inference predictions for the final epoch
        predictions_df = pd.DataFrame({
            "y_true": y_true,
            "y_pred": y_pred
        })
        predictions_df.to_csv(f"sent_gpt2_bitfit_inference_predictions_lr{lr}_bs{

        # log memory usage
        max_memory = torch.cuda.max_memory_allocated() / (1024 ** 3) if torch.cud

        # save model params and metrics
        results.append({
            "method": "BitFit",
            "learning_rate": lr,
            "batch_size": batch_size,
            "accuracy": val_accuracy,
            "f1": val_f1,
            "training_time": training_time,
            "inference_time": inference_time,
            "max_memory": max_memory
        })

        # empty cache to conserve compute
        del model, tokenizer, optimizer
        torch.cuda.empty_cache()
        gc.collect()

# ranked performance by val acc
results = sorted(results, key=lambda x: x["accuracy"], reverse=True)

# save overall results
results_df = pd.DataFrame(results)
results_df.to_csv("sent_gpt2_bitfit_results.csv", index=False)
```

```
# save best final config and metrics
final_summary_df = pd.DataFrame({
    "Method": ["BitFit"],
    "Best LR": [results[0]["learning_rate"]],
    "Best Batch Size": [results[0]["batch_size"]],
    "Accuracy": [results[0]["accuracy"]],
    "F1 Score": [results[0]["f1"]],
    "Training Time (s)": [results[0]["training_time"]],
    "Inference Time (s)": [results[0]["inference_time"]],
    "Max GPU Memory (GB)": [results[0]["max_memory"]]
})
final_summary_df.to_csv("sent_gpt2_bf_final_comparison_bitfit.csv", index=False)

print("All BitFit Grid Search Results:")
for r in results:
    print(r)

print("\nBest BitFit Configuration:")
print(results[0])
```

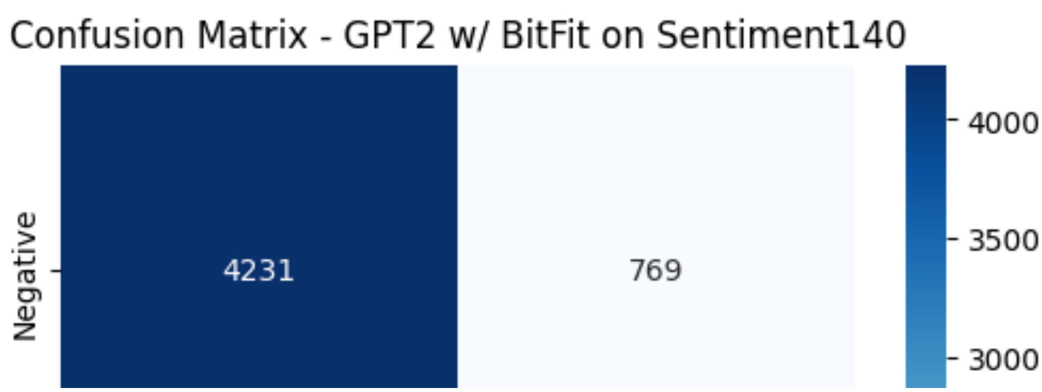
```
[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7953
F1 Score (macro)   : 0.7953
F1 Score (weighted): 0.7953
Inference Time     : 19.09 seconds

Classification Report: GPT2 w/ BitFit on Sentiment140

              precision    recall  f1-score   support

    Negative       0.79      0.81      0.80      5000
    Positive       0.80      0.78      0.79      5000

    accuracy                           0.80     10000
   macro avg       0.80      0.80      0.80     10000
weighted avg       0.80      0.80      0.80     10000

Running BitFit with LR=0.0001, batch_size=8
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use :

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8007
F1 Score (macro)   : 0.8003
F1 Score (weighted): 0.8003
Inference Time     : 19.60 seconds
```

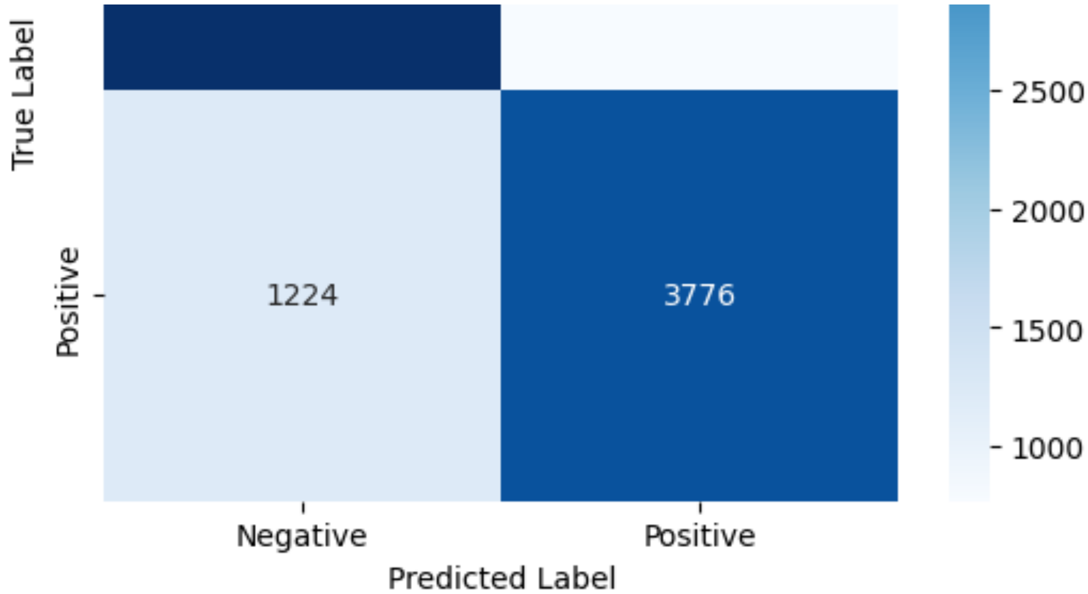
```
Classification Report: GPT2 w/ BitFit on Sentiment140

              precision    recall  f1-score   support

    Negative       0.78      0.85      0.81      5000
    Positive       0.83      0.76      0.79      5000

    accuracy                           0.80     10000
   macro avg       0.80      0.80      0.80     10000
weighted avg       0.80      0.80      0.80     10000

Running BitFit with LR=0.0001, batch_size=16
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7987
F1 Score (macro)   : 0.7983
F1 Score (weighted): 0.7983
Inference Time     : 19.06 seconds

Classification Report: GPT2 w/ BitFit on Sentiment140

              precision    recall  f1-score   support

    Negative       0.77      0.84      0.81      5000
    Positive       0.83      0.75      0.79      5000

    accuracy                           0.80     10000
   macro avg       0.80      0.80      0.80     10000
weighted avg       0.80      0.80      0.80     10000

All BitFit Grid Search Results:
```

```
bf_best_lr = results[0]["learning_rate"]
bf_best_bs = results[0]["batch_size"]

# Construct filename
best_report_file = f"sent_gpt2_bitfit_inference_metrics_summary_lr{bf_best_lr}_bs

# Load the saved best report
best_report_df = pd.read_csv(best_report_file)
print("\nClassification Report for Best Configuration:")
print(best_report_df)


best_preds_df = pd.read_csv(f"sent_gpt2_bitfit_inference_predictions_lr{bf_best_l
print("\nInference Predictions for Best Configuration:")
```

```
print(best_preds_df)

y_true = best_preds_df["y_true"]
y_pred = best_preds_df["y_pred"]


cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative", "Posi
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - GPT2 w/ BitFit on Sentiment140")
plt.show()
```

```
Classification Report for Best Configuration:
     Unnamed: 0  precision  recall  f1-score     support
0             0   0.775619  0.8462  0.809374   5000.0000
1             1   0.830803  0.7552  0.791200   5000.0000
2      accuracy   0.800700  0.8007  0.800700      0.8007
3     macro avg   0.803211  0.8007  0.800287  10000.0000
4  weighted avg   0.803211  0.8007  0.800287  10000.0000

Inference Predictions for Best Configuration:
      y_true  y_pred
0          0       0
1          0       0
2          1       0
3          0       0
4          0       0
...      ...     ...
9995       0       0
9996       0       0
9997       0       0
9998       1       1
9999       0       0

[10000 rows x 2 columns]
```






## Prompt Tuning

```
""" Importing prompt tuning packages from PEFT """

import gc
from peft import PromptTuningConfig, PromptTuningInit, get_peft_model, TaskType


""" Prompt tuning parameter setup """

lrs = [5e-5, 1e-4]
bs = [8, 16]
num_tokens = 20
epochs = 6


""" Training and evaluation loop with hyperparamter grid search """
from torch import autocast


results = []

for lr in lrs:
    for batch_size in bs:
        print(f"Running Prompt Tuning with LR={lr}, batch_size={batch_size}")

        # loading GPT2 model
```

```
model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_

# prompt tuning config
peft_config = PromptTuningConfig(
    task_type=TaskType.SEQ_CLS,
    num_virtual_tokens=num_tokens,
    tokenizer_name_or_path=tokenizer.name_or_path,
    prompt_tuning_init=PromptTuningInit.RANDOM,
)
prompt_model = get_peft_model(model, peft_config)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
prompt_model.to(device)

# instantiate dataloader
data_collator = DataCollatorWithPadding(tokenizer)
train_dataloader = DataLoader(tokenized_dataset["train"], batch_size=batch_
test_dataloader = DataLoader(tokenized_dataset["test"], batch_size=batch_si

# adam optimization
optimizer = torch.optim.AdamW(prompt_model.parameters(), lr=lr)

# begin training
prompt_model.train()
start_time = time.time()
epoch_logs = []

for epoch in range(1, epochs + 1):
    running_loss = 0.0
    correct = 0
    total = 0
    loop = tqdm(train_dataloader, leave=True, desc=f"Epoch {epoch}/{epochs}
    for step, batch in enumerate(loop):
        batch = {
            "input_ids": batch["input_ids"].to(device),
            "attention_mask": batch["attention_mask"].to(device),
            "labels": batch["sentiment"].to(device)
        }

        with autocast(device_type='cuda'):
            outputs = model(**batch)
            loss = outputs.loss
            preds = torch.argmax(outputs.logits, dim=1)
```

```
            correct += (preds == batch['labels']).sum().item()
            total += batch['labels'].size(0)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

    avg_train_loss = running_loss / (step + 1)
    train_accuracy = correct / total

    # perform per epoch evaluation
    prompt_model.eval()
    val_running_loss = 0.0
    y_true, y_pred = [], []
    with torch.no_grad():
        with autocast(device_type='cuda'):
            for batch in test_dataloader:
                batch = {
                    "input_ids": batch["input_ids"].to(device),
                    "attention_mask": batch["attention_mask"].to(device),
                    "labels": batch["sentiment"].to(device)
                }
                outputs = model(**batch)
                preds = torch.argmax(outputs.logits, dim=1)
                y_true.extend(batch["labels"].cpu().numpy())
                y_pred.extend(preds.cpu().numpy())
                val_running_loss += outputs.loss.item()

    avg_val_loss = val_running_loss / len(test_dataloader)

    inference_time = time.time() - start_time

    report = classification_report(y_true, y_pred, output_dict=True)
    val_accuracy = report["accuracy"]
    val_f1 = report["weighted avg"]["f1-score"]

    # print classification report on final epoch
    if epoch == epochs:
        total_correct = sum(yt == yp for yt, yp in zip(y_true, y_pred))
        total_samples = len(y_true)

        accuracy = total_correct / total_samples
        f1_macro = f1_score(y_true, y_pred, average="macro")
```

```
                f1_weighted = f1_score(y_true, y_pred, average="weighted")

                print(f"\n[Final Epoch {epoch}] Inference Metrics:")
                print(f"Test 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} seconds")
                print("\nClassification Report: - GPT2 w/ Prompt Tuning on Sentimer
                print(classification_report(y_true, y_pred, target_names=["Negative

            epoch_logs.append({
                "epoch": epoch,
                "lr": lr,
                "batch_size": batch_size,
                "train_loss": avg_train_loss,
                "train_accuracy": train_accuracy,
                "val_loss": avg_val_loss,
                "val_accuracy": val_accuracy
            })

            prompt_model.train()

        end_time = time.time()
        training_time = end_time - start_time

        # begin datalogging per lr/bs
        epoch_logs_df = pd.DataFrame(epoch_logs)
        epoch_logs_df.to_csv(f"sent_gpt2_prompt_epoch_logs_lr{lr}_bs{batch_size}.cs

        # save inference metrics per lr/bs
        metrics_summary_df = pd.DataFrame(report).transpose()
        metrics_summary_df.to_csv(f"sent_gpt2_prompt_inference_metrics_summary_lr{l

        # Save inference predictions for the final epoch
        predictions_df = pd.DataFrame({
            "y_true": y_true,
            "y_pred": y_pred
        })
        predictions_df.to_csv(f"sent_gpt2_prompt_inference_predictions_lr{lr}_bs{ba

        # log memory usage
        max_memory = torch.cuda.max_memory_allocated() / (1024 ** 3) if torch.cuda.

        # save model params and metrics
        results.append({
```

```
            "method": "Prompt Tuning",
            "learning_rate": lr,
            "batch_size": batch_size,
            "accuracy": val_accuracy,
            "f1": val_f1,
            "training_time": training_time,
            "inference_time": inference_time,
            "max_memory": max_memory
        })

        # empty cache to conserve compute
        del prompt_model, model, tokenizer, optimizer
        torch.cuda.empty_cache()
        gc.collect()

# ranked performance by val acc
results = sorted(results, key=lambda x: x["accuracy"], reverse=True)

# save overall results
results_df = pd.DataFrame(results)
results_df.to_csv("sent_gpt2_prompt_results.csv", index=False)

# save best final config and metrics
final_summary_df = pd.DataFrame({
    "Method": ["Prompt Tuning"],
    "Best LR": [results[0]["learning_rate"]],
    "Best Batch Size": [results[0]["batch_size"]],
    "Accuracy": [results[0]["accuracy"]],
    "F1 Score": [results[0]["f1"]],
    "Training Time (s)": [results[0]["training_time"]],
    "Max GPU Memory (GB)": [results[0]["max_memory"]]
})
final_summary_df.to_csv("sent_gpt2_prompt_final_comparison_prompt_tuning.csv", inde

print("All Prompt Tuning Grid Search Results:")
for r in results:
    print(r)

print("\nBest Configuration:")
print(results[0])
```

```
              precision    recall  f1-score   support

    Negative       0.73      0.77      0.75      5000
    Positive       0.76      0.72      0.74      5000
```

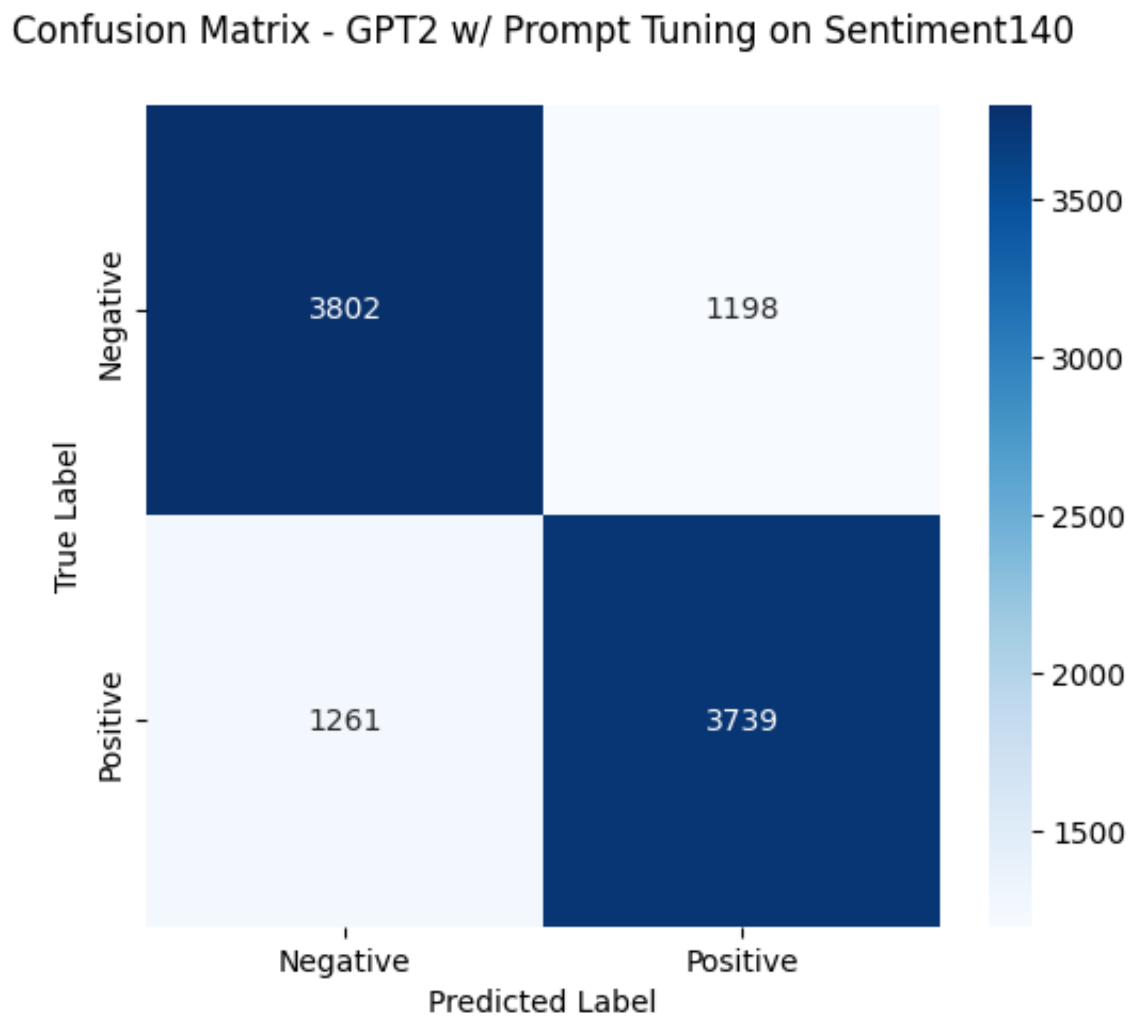
```
    accuracy                           0.74     10000
   macro avg       0.74      0.74      0.74     10000
weighted avg       0.74      0.74      0.74     10000

Running Prompt Tuning with LR=0.0001, batch_size=8
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i
Epoch 1/6: 100%|██████████| 5000/5000 [01:25<00:00, 58.32it/s]
Epoch 2/6: 100%|██████████| 5000/5000 [01:25<00:00, 58.19it/s]
Epoch 3/6: 100%|██████████| 5000/5000 [01:26<00:00, 57.81it/s]
Epoch 4/6: 100%|██████████| 5000/5000 [01:25<00:00, 58.48it/s]
Epoch 5/6: 100%|██████████| 5000/5000 [01:25<00:00, 58.20it/s]
Epoch 6/6: 100%|██████████| 5000/5000 [01:25<00:00, 58.18it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7541
F1 Score (macro)   : 0.7541
F1 Score (weighted): 0.7541
Inference Time     : 626.96 seconds

Classification Report: - GPT2 w/ Prompt Tuning on Sentiment140

              precision    recall  f1-score   support

    Negative       0.75      0.76      0.76      5000
    Positive       0.76      0.75      0.75      5000

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000

Running Prompt Tuning with LR=0.0001, batch_size=16
Some weights of GPT2ForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i
Epoch 1/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.74it/s]
Epoch 2/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.89it/s]
Epoch 3/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.64it/s]
Epoch 4/6: 100%|██████████| 2500/2500 [00:45<00:00, 55.54it/s]
Epoch 5/6: 100%|██████████| 2500/2500 [00:45<00:00, 55.38it/s]
Epoch 6/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.72it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7490
F1 Score (macro)   : 0.7476
F1 Score (weighted): 0.7476
Inference Time     : 328.09 seconds

Classification Report: - GPT2 w/ Prompt Tuning on Sentiment140
```

```
              precision    recall  f1-score   support

    Negative       0.72      0.82      0.77      5000
    Positive       0.79      0.68      0.73      5000
```

```
prompt_best_lr = results[0]["learning_rate"]
prompt_best_bs = results[0]["batch_size"]

# Construct filename
best_report_file = f"sent_gpt2_prompt_inference_metrics_summary_lr{prompt_best_lr

# Load the saved best report
best_report_df = pd.read_csv(best_report_file)
print("\nClassification Report for Best Configuration:")
print(best_report_df)


best_preds_df = pd.read_csv(f"sent_gpt2_prompt_inference_predictions_lr{prompt_be
print("\nInference Predictions for Best Configuration:")
print(best_preds_df)

y_true = best_preds_df["y_true"]
y_pred = best_preds_df["y_pred"]


cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Negative", "Posi
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - GPT2 w/ Prompt Tuning on Sentiment140\n")
plt.show()
```

```
Classification Report for Best Configuration:
     Unnamed: 0  precision  recall  f1-score     support
0             0   0.750938  0.7604  0.755639   5000.0000
1             1   0.757343  0.7478  0.752541   5000.0000
2      accuracy   0.754100  0.7541  0.754100      0.7541
3     macro avg   0.754140  0.7541  0.754090  10000.0000
4  weighted avg   0.754140  0.7541  0.754090  10000.0000

Inference Predictions for Best Configuration:
      y_true  y_pred
0          0       0
1          0       0
2          1       0
```

```
3          0        0
4          0        0
...        ...      ...
9995       0        0
9996       0        1
9997       0        0
9998       1        1
9999       0        0

[10000 rows x 2 columns]
```




## Begin Visualization of Sentiment140 Results

Load all dataframes from above training of 3 PEFT methods

```
""" Output from our GPT2_Sentiment140_IMDb50k.ipynb file, we read in the .csv fil
```

Note that these .csv file paths suggest they should be uploaded to session storag

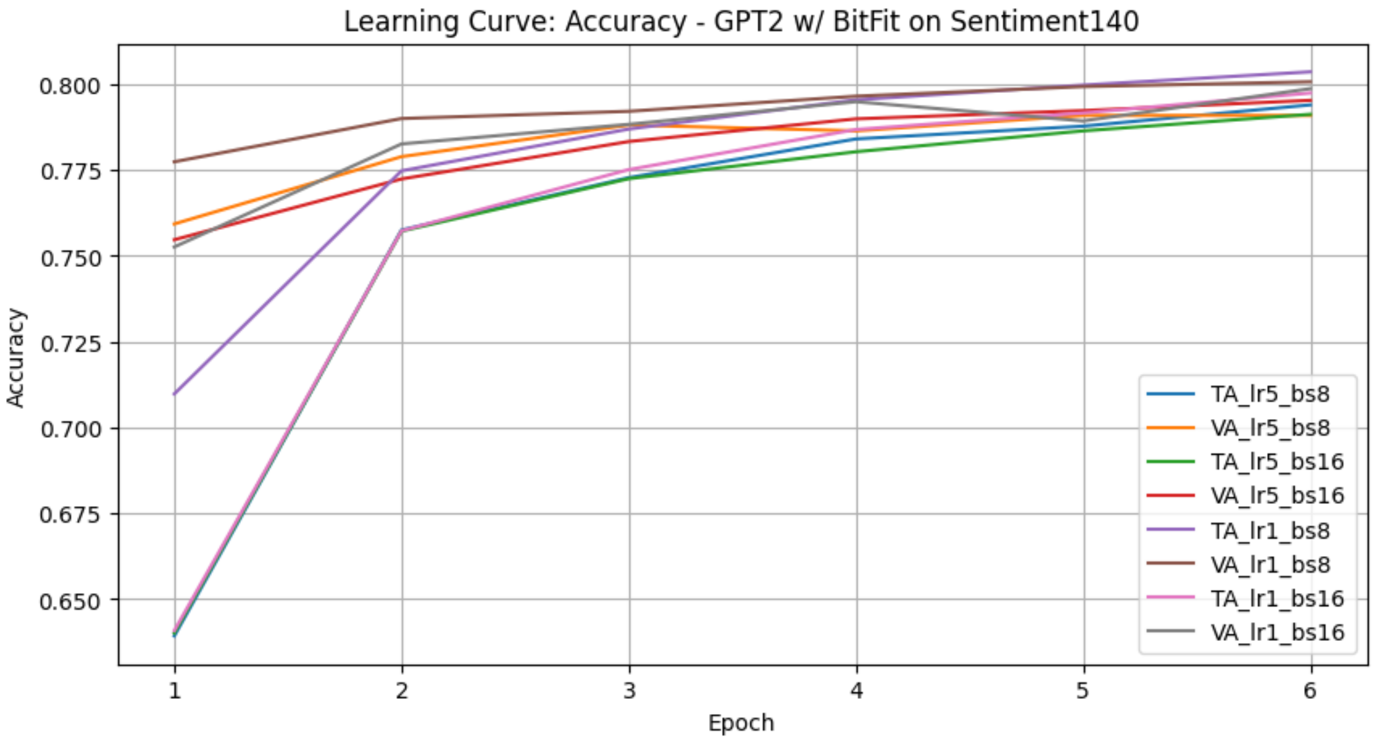
```
# GPT2 PEFT-wise results across bf/lora/prompt tuning
gpt2_bf_results = pd.read_csv('/content/sent_gpt2_bitfit_results.csv')
gpt2_lora_results = pd.read_csv('/content/sent_gpt2_lora_results.csv')
gpt2_prompt_results = pd.read_csv('/content/sent_gpt2_prompt_results.csv')

# GPT2 per-epoch performance logs (for LC generation) across bf/lora/prompt tunin
gpt2_lora_epochs_lr5_bs8 = pd.read_csv('/content/sent_gpt2_lora_epoch_logs_lr5e-0
gpt2_lora_epochs_lr5_bs16 = pd.read_csv('/content/sent_gpt2_lora_epoch_logs_lr5e-
gpt2_lora_epochs_lr1_bs8 = pd.read_csv('/content/sent_gpt2_lora_epoch_logs_lr0.00
gpt2_lora_epochs_lr1_bs16 = pd.read_csv('/content/sent_gpt2_lora_epoch_logs_lr0.0
gpt2_bf_epochs_lr5_bs8 = pd.read_csv('/content/sent_gpt2_bitfit_epoch_logs_lr5e-0
gpt2_bf_epochs_lr5_bs16 = pd.read_csv('/content/sent_gpt2_bitfit_epoch_logs_lr5e-
gpt2_bf_epochs_lr1_bs8 = pd.read_csv('/content/sent_gpt2_bitfit_epoch_logs_lr0.00
gpt2_bf_epochs_lr1_bs16 = pd.read_csv('/content/sent_gpt2_bitfit_epoch_logs_lr0.0
gpt2_prompt_epochs_lr5_bs8 = pd.read_csv('/content/sent_gpt2_prompt_epoch_logs_lr
gpt2_prompt_epochs_lr5_bs16 = pd.read_csv('/content/sent_gpt2_prompt_epoch_logs_l
gpt2_prompt_epochs_lr1_bs8 = pd.read_csv('/content/sent_gpt2_prompt_epoch_logs_lr
gpt2_prompt_epochs_lr1_bs16 = pd.read_csv('/content/sent_gpt2_prompt_epoch_logs_l

# GPT2 inference performance metric summary across bf/lora/prompt tuning
gpt2_bf_inf_lr5_bs8 = pd.read_csv('/content/sent_gpt2_bitfit_inference_metrics_su
gpt2_bf_inf_lr5_bs16 = pd.read_csv('/content/sent_gpt2_bitfit_inference_metrics_s
gpt2_bf_inf_lr1_bs8 = pd.read_csv('/content/sent_gpt2_bitfit_inference_metrics_su
gpt2_bf_inf_lr1_bs16 = pd.read_csv('/content/sent_gpt2_bitfit_inference_metrics_s
gpt2_lora_inf_lr5_bs8 = pd.read_csv('/content/sent_gpt2_lora_inference_metrics_su
gpt2_lora_inf_lr5_bs16 = pd.read_csv('/content/sent_gpt2_lora_inference_metrics_s
gpt2_lora_inf_lr1_bs8 = pd.read_csv('/content/sent_gpt2_lora_inference_metrics_su
gpt2_lora_inf_lr1_bs16 = pd.read_csv('/content/sent_gpt2_lora_inference_metrics_s
gpt2_prompt_inf_lr5_bs8 = pd.read_csv('/content/sent_gpt2_prompt_inference_metric
gpt2_prompt_inf_lr5_bs16 = pd.read_csv('/content/sent_gpt2_prompt_inference_metri
gpt2_prompt_inf_lr1_bs8 = pd.read_csv('/content/sent_gpt2_prompt_inference_metric
gpt2_prompt_inf_lr1_bs16 = pd.read_csv('/content/sent_gpt2_prompt_inference_metri

# GPT2 inference predictions across bf/lora/prompt tuning
gpt2_bf_preds_lr5_bs8 = pd.read_csv('/content/sent_gpt2_bitfit_inference_predicti
gpt2_bf_preds_lr5_bs16 = pd.read_csv('/content/sent_gpt2_bitfit_inference_predict
gpt2_bf_preds_lr1_bs8 = pd.read_csv('/content/sent_gpt2_bitfit_inference_predicti
gpt2_bf_preds_lr1_bs16 = pd.read_csv('/content/sent_gpt2_bitfit_inference_predict
gpt2_lora_preds_lr5_bs8 = pd.read_csv('/content/sent_gpt2_lora_inference_predicti
gpt2_lora_preds_lr5_bs16 = pd.read_csv('/content/sent_gpt2_lora_inference_predict
gpt2_lora_preds_lr1_bs8 = pd.read_csv('/content/sent_gpt2_lora_inference_predicti
gpt2_lora_preds_lr1_bs16 = pd.read_csv('/content/sent_gpt2_lora_inference_predict
gpt2_prompt_preds_lr5_bs8 = pd.read_csv('/content/sent_gpt2_prompt_inference_pred
```

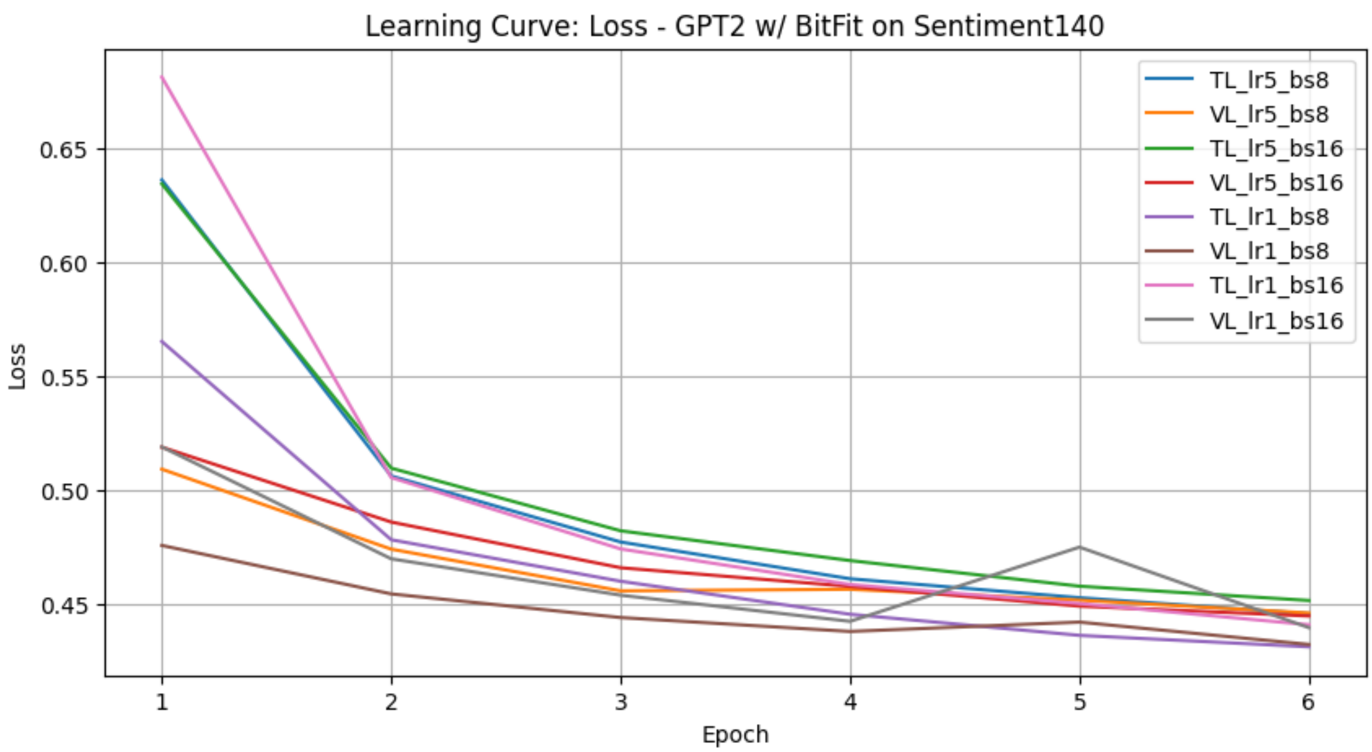
```
gpt2_prompt_preds_lr5_bs16 = pd.read_csv('/content/sent_gpt2_prompt_inference_pre
gpt2_prompt_preds_lr1_bs8 = pd.read_csv('/content/sent_gpt2_prompt_inference_pred
gpt2_prompt_preds_lr1_bs16 = pd.read_csv('/content/sent_gpt2_prompt_inference_pre

# GPT2 PEFT method intra-comparison based on hyperparameter settings, per bf/lora
gpt2_bf_final_comparison = pd.read_csv('/content/sent_gpt2_bf_final_comparison_bi
gpt2_lora_final_comparison = pd.read_csv('/content/sent_gpt2_lora_final_compariso
gpt2_prompt_final_comparison = pd.read_csv('/content/sent_gpt2_prompt_final_compa
```

BitFit Learning Curves

```
# All BitFit Train/Val Acc Learning Curve
plt.figure(figsize=(10,5))
sns.lineplot(data=gpt2_bf_epochs_lr5_bs8, x="epoch", y="train_accuracy", label="T
sns.lineplot(data=gpt2_bf_epochs_lr5_bs8, x="epoch", y="val_accuracy", label="VA_
sns.lineplot(data=gpt2_bf_epochs_lr5_bs16, x="epoch", y="train_accuracy", label="
sns.lineplot(data=gpt2_bf_epochs_lr5_bs16, x="epoch", y="val_accuracy", label="VA
sns.lineplot(data=gpt2_bf_epochs_lr1_bs8, x="epoch", y="train_accuracy", label="T
sns.lineplot(data=gpt2_bf_epochs_lr1_bs8, x="epoch", y="val_accuracy", label="VA_
sns.lineplot(data=gpt2_bf_epochs_lr1_bs16, x="epoch", y="train_accuracy", label="
sns.lineplot(data=gpt2_bf_epochs_lr1_bs16, x="epoch", y="val_accuracy", label="VA
plt.title("Learning Curve: Accuracy - GPT2 w/ BitFit on Sentiment140")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# All BitFit Training and Validation Loss
plt.figure(figsize=(10,5))
sns.lineplot(data=gpt2_bf_epochs_lr5_bs8, x="epoch", y="train_loss", label="TL_lr
sns.lineplot(data=gpt2_bf_epochs_lr5_bs8, x="epoch", y="val_loss", label="VL_lr5_
sns.lineplot(data=gpt2_bf_epochs_lr5_bs16, x="epoch", y="train_loss", label="TL_l
sns.lineplot(data=gpt2_bf_epochs_lr5_bs16, x="epoch", y="val_loss", label="VL_lr5
sns.lineplot(data=gpt2_bf_epochs_lr1_bs8, x="epoch", y="train_loss", label="TL_lr
sns.lineplot(data=gpt2_bf_epochs_lr1_bs8, x="epoch", y="val_loss", label="VL_lr1_
sns.lineplot(data=gpt2_bf_epochs_lr1_bs16, x="epoch", y="train_loss", label="TL_l
sns.lineplot(data=gpt2_bf_epochs_lr1_bs16, x="epoch", y="val_loss", label="VL_lr1
plt.title("Learning Curve: Loss - GPT2 w/ BitFit on Sentiment140")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
```

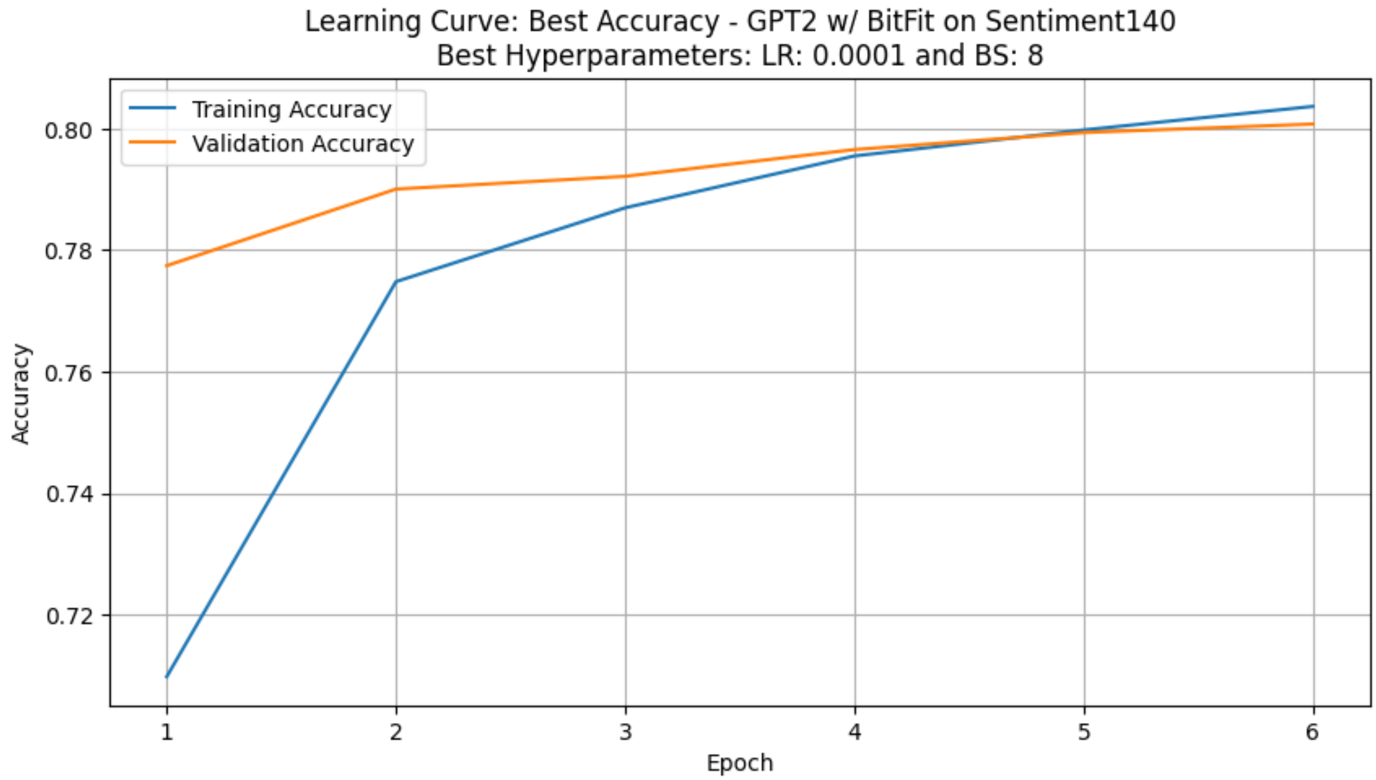
```
plt.show()
```

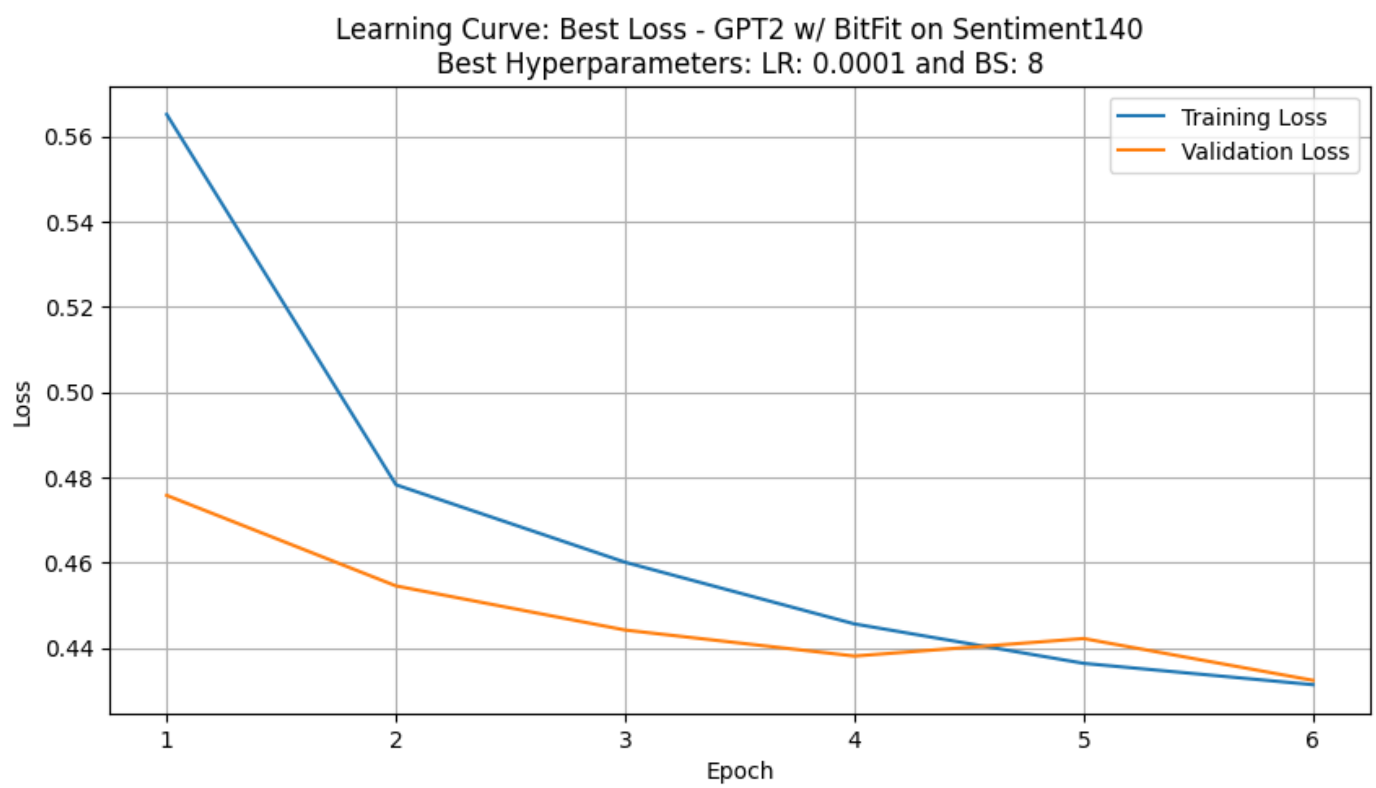

Learning Curve: Accuracy - GPT2 w/ BitFit on Sentiment140
Accuracy
Epoch
0.800
0.775
0.750
0.725
0.700
0.675
0.650
1
2
3
4
5
6
TA_lr5_bs8
VA_lr5_bs8
TA_lr5_bs16
VA_lr5_bs16
TA_lr1_bs8
VA_lr1_bs8
TA_lr1_bs16
VA_lr1_bs16


Learning Curve: Loss - GPT2 w/ BitFit on Sentiment140
Loss
Epoch
0.65
0.60
0.55
0.50
0.45
1
2
3
4
5
6
TL_lr5_bs8
VL_lr5_bs8
TL_lr5_bs16
VL_lr5_bs16
TL_lr1_bs8
VL_lr1_bs8
TL_lr1_bs16
VL_lr1_bs16

```
# Best BitFit Train/Val Acc Learning Curve

gpt2_bf_epochs_map = {
    (5, 8): gpt2_bf_epochs_lr5_bs8,
    (5, 16): gpt2_bf_epochs_lr5_bs16,
    (1, 8): gpt2_bf_epochs_lr1_bs8,
    (1, 16): gpt2_bf_epochs_lr1_bs16
}

bf_lr_mapping = {
    5e-5: 5,
    1e-4: 1
}

bf_best_lr_tag = bf_lr_mapping[bf_best_lr]
bf_best_bs_tag = bf_best_bs

bf_epochs = gpt2_bf_epochs_map[(bf_best_lr_tag, bf_best_bs_tag)]

# Best BitFit Training and Validation Accuracy
plt.figure(figsize=(10,5))
sns.lineplot(data=bf_epochs, x="epoch", y="train_accuracy", label="Training Accur
sns.lineplot(data=bf_epochs, x="epoch", y="val_accuracy", label="Validation Accur
plt.title(f"Learning Curve: Best Accuracy - GPT2 w/ BitFit on Sentiment140\nBest 
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# Best BitFit Training and Validation Loss
plt.figure(figsize=(10,5))
sns.lineplot(data=bf_epochs, x="epoch", y="train_loss", label="Training Loss")
sns.lineplot(data=bf_epochs, x="epoch", y="val_loss", label="Validation Loss")
plt.title(f"Learning Curve: Best Loss - GPT2 w/ BitFit on Sentiment140\nBest Hype
plt.xlabel("Epoch")
plt.ylabel("Loss")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

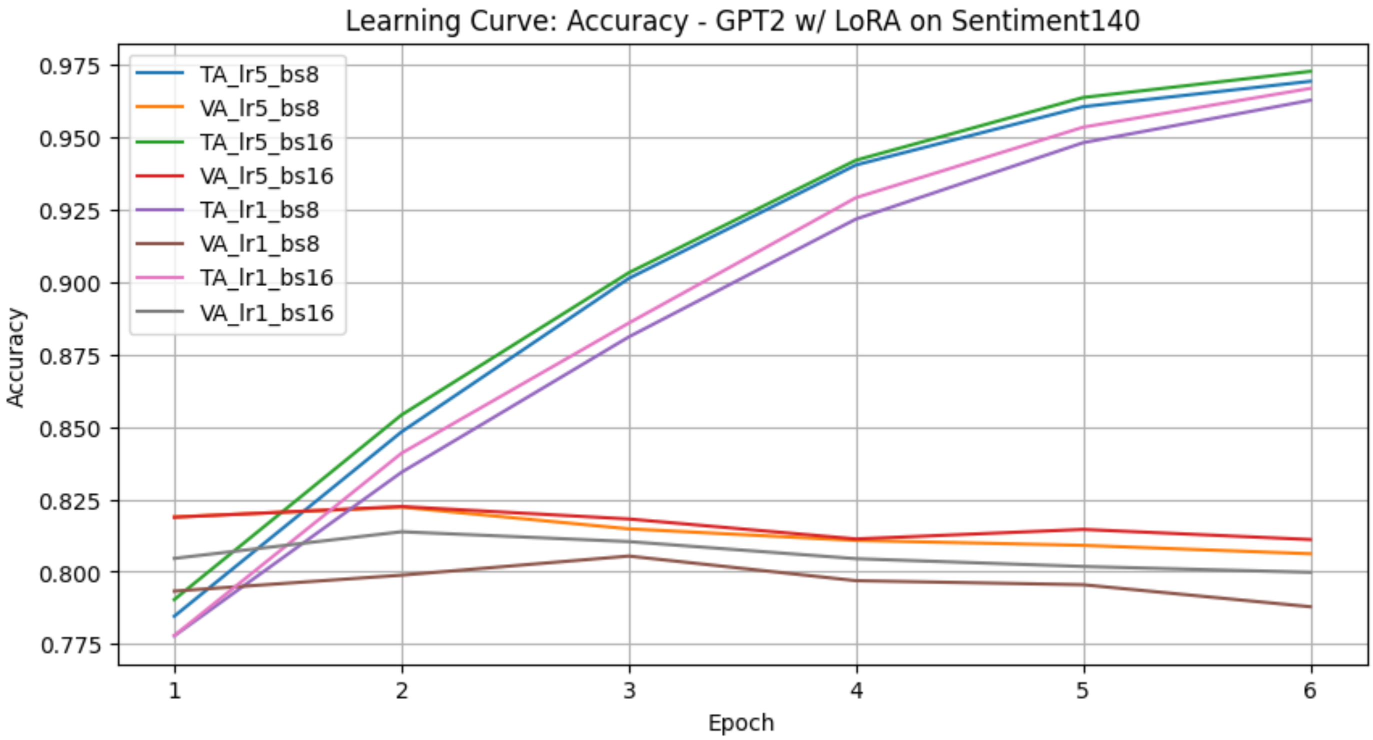


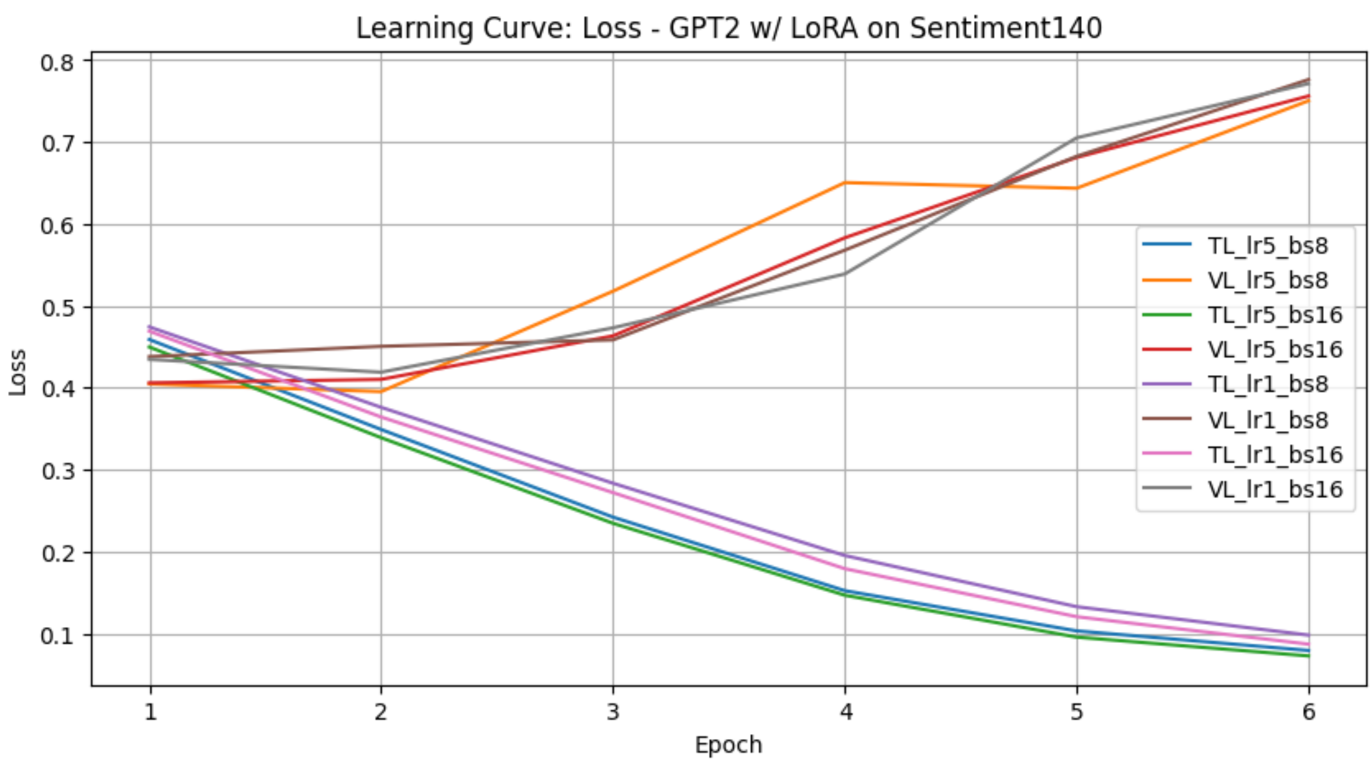



## LoRA Learning Curves

```
# All LoRA Train/Val Acc Learning Curve
plt.figure(figsize=(10,5))
sns.lineplot(data=gpt2_lora_epochs_lr5_bs8, x="epoch", y="train_accuracy", label=
sns.lineplot(data=gpt2_lora_epochs_lr5_bs8, x="epoch", y="val_accuracy", label="V
sns.lineplot(data=gpt2_lora_epochs_lr5_bs16, x="epoch", y="train_accuracy", label
sns.lineplot(data=gpt2_lora_epochs_lr5_bs16, x="epoch", y="val_accuracy", label="
sns.lineplot(data=gpt2_lora_epochs_lr1_bs8, x="epoch", y="train_accuracy", label=
sns.lineplot(data=gpt2_lora_epochs_lr1_bs8, x="epoch", y="val_accuracy", label="V
sns.lineplot(data=gpt2_lora_epochs_lr1_bs16, x="epoch", y="train_accuracy", label
sns.lineplot(data=gpt2_lora_epochs_lr1_bs16, x="epoch", y="val_accuracy", label="
plt.title("Learning Curve: Accuracy - GPT2 w/ LoRA on Sentiment140")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# All LoRA Training and Validation Loss
plt.figure(figsize=(10,5))
sns.lineplot(data=gpt2_lora_epochs_lr5_bs8, x="epoch", y="train_loss", label="TL_
sns.lineplot(data=gpt2_lora_epochs_lr5_bs8, x="epoch", y="val_loss", label="VL_lr
sns.lineplot(data=gpt2_lora_epochs_lr5_bs16, x="epoch", y="train_loss", label="TL
sns.lineplot(data=gpt2_lora_epochs_lr5_bs16, x="epoch", y="val_loss", label="VL_l
sns.lineplot(data=gpt2_lora_epochs_lr1_bs8, x="epoch", y="train_loss", label="TL_
sns.lineplot(data=gpt2_lora_epochs_lr1_bs8, x="epoch", y="val_loss", label="VL_lr
sns.lineplot(data=gpt2_lora_epochs_lr1_bs16, x="epoch", y="train_loss", label="TL
sns.lineplot(data=gpt2_lora_epochs_lr1_bs16, x="epoch", y="val_loss", label="VL_l
plt.title("Learning Curve: Loss - GPT2 w/ LoRA on Sentiment140")
plt.xlabel("Epoch")
plt.ylabel("Loss")
```

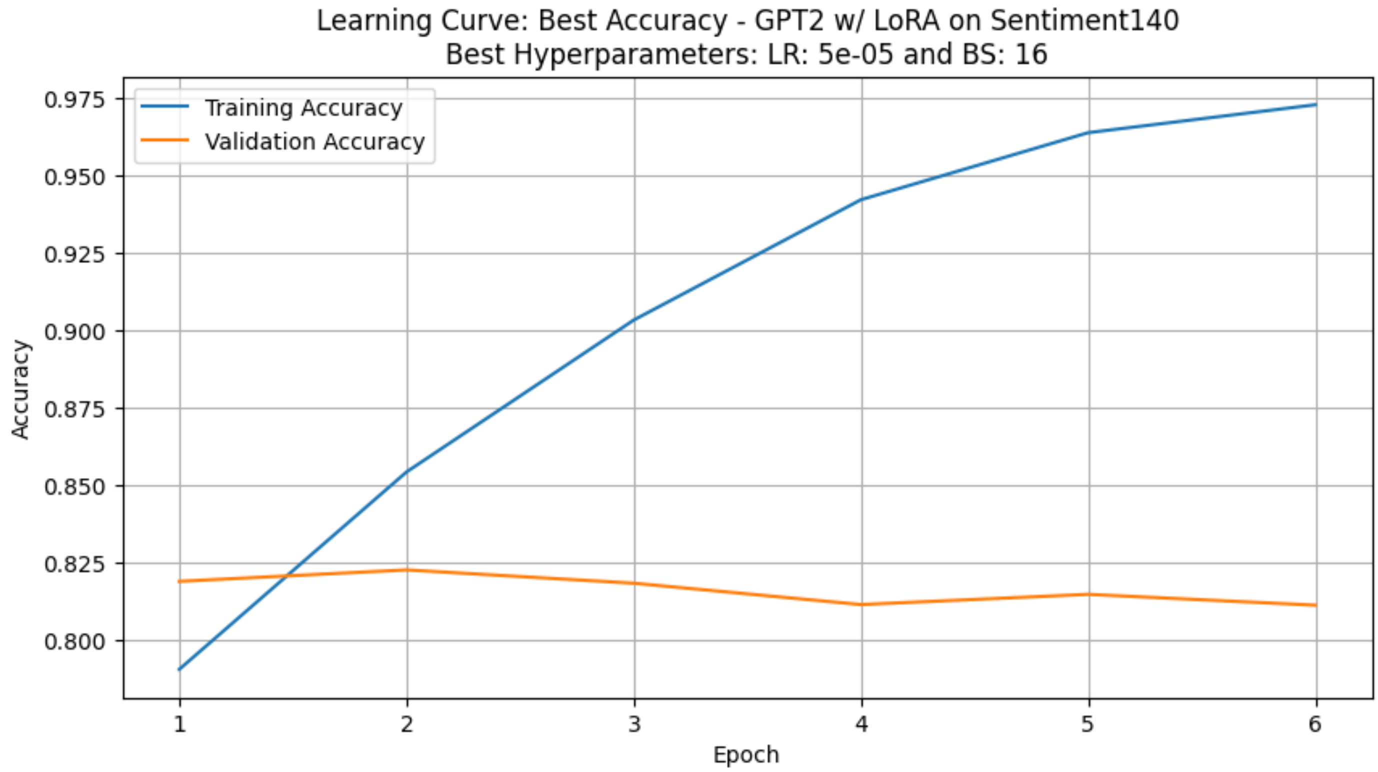
```
plt.legend()
plt.grid(True)
plt.show()
```

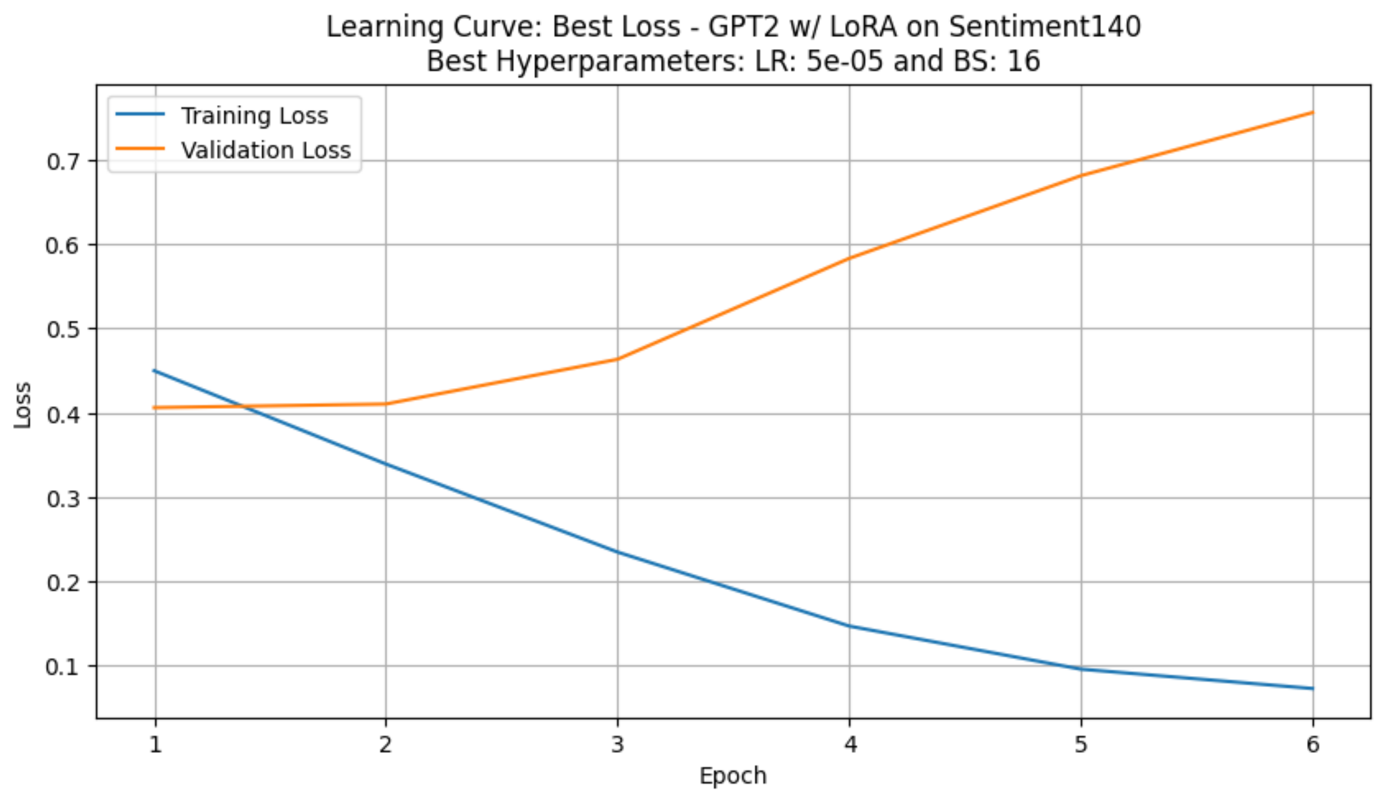


```
# Best LoRA Train/Val Acc Learning Curve

gpt2_lora_epochs_map = {
    (5, 8): gpt2_lora_epochs_lr5_bs8,
    (5, 16): gpt2_lora_epochs_lr5_bs16,
    (1, 8): gpt2_lora_epochs_lr1_bs8,
    (1, 16): gpt2_lora_epochs_lr1_bs16
}

lora_lr_mapping = {
    5e-5: 5,
    1e-4: 1
}

lora_best_lr_tag = lora_lr_mapping[lora_best_lr]
lora_best_bs_tag = lora_best_bs

lora_epochs = gpt2_lora_epochs_map[(lora_best_lr_tag, lora_best_bs_tag)]

# Best LoRA Training and Validation Accuracy
plt.figure(figsize=(10,5))
sns.lineplot(data=lora_epochs, x="epoch", y="train_accuracy", label="Training Acc
sns.lineplot(data=lora_epochs, x="epoch", y="val_accuracy", label="Validation Acc
plt.title(f"Learning Curve: Best Accuracy - GPT2 w/ LoRA on Sentiment140\nBest Hy
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# Best LoRA Training and Validation Loss
plt.figure(figsize=(10,5))
sns.lineplot(data=lora_epochs, x="epoch", y="train_loss", label="Training Loss")
sns.lineplot(data=lora_epochs, x="epoch", y="val_loss", label="Validation Loss")
plt.title(f"Learning Curve: Best Loss - GPT2 w/ LoRA on Sentiment140\nBest Hyperp
```

```
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

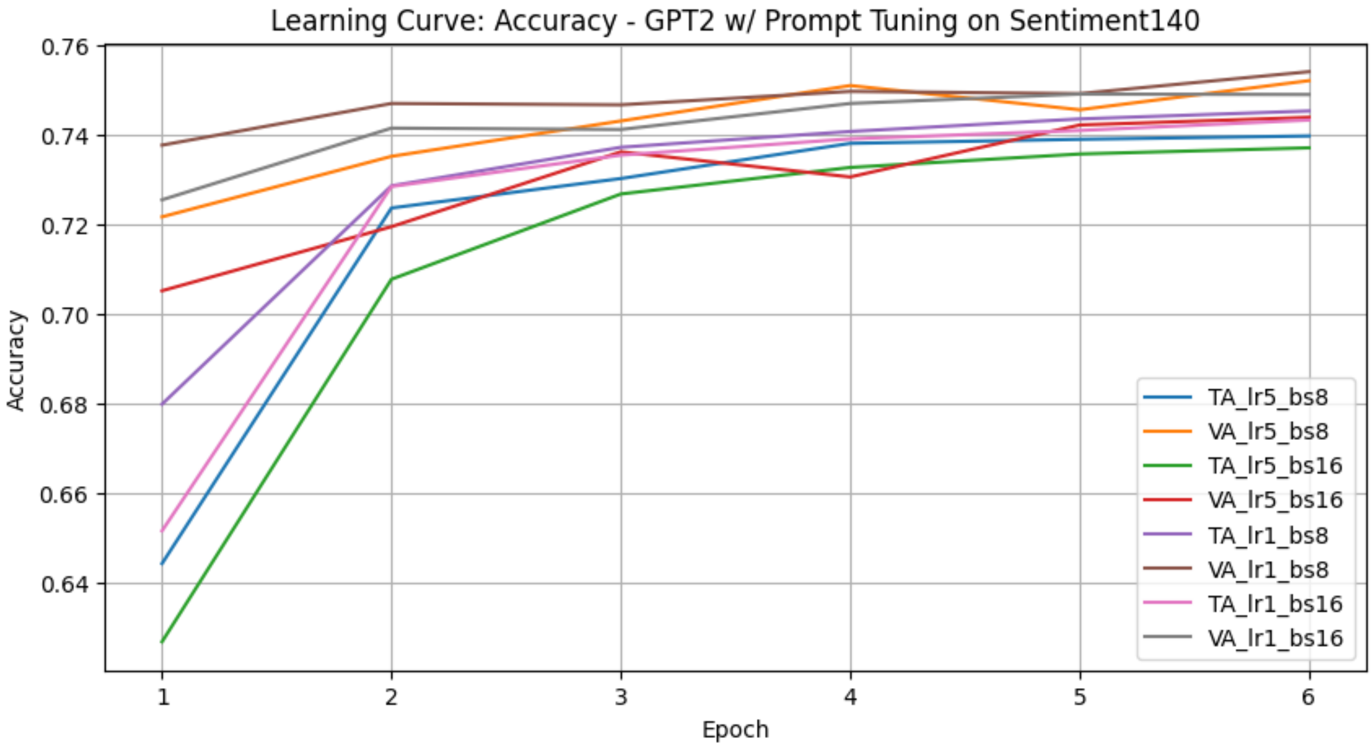


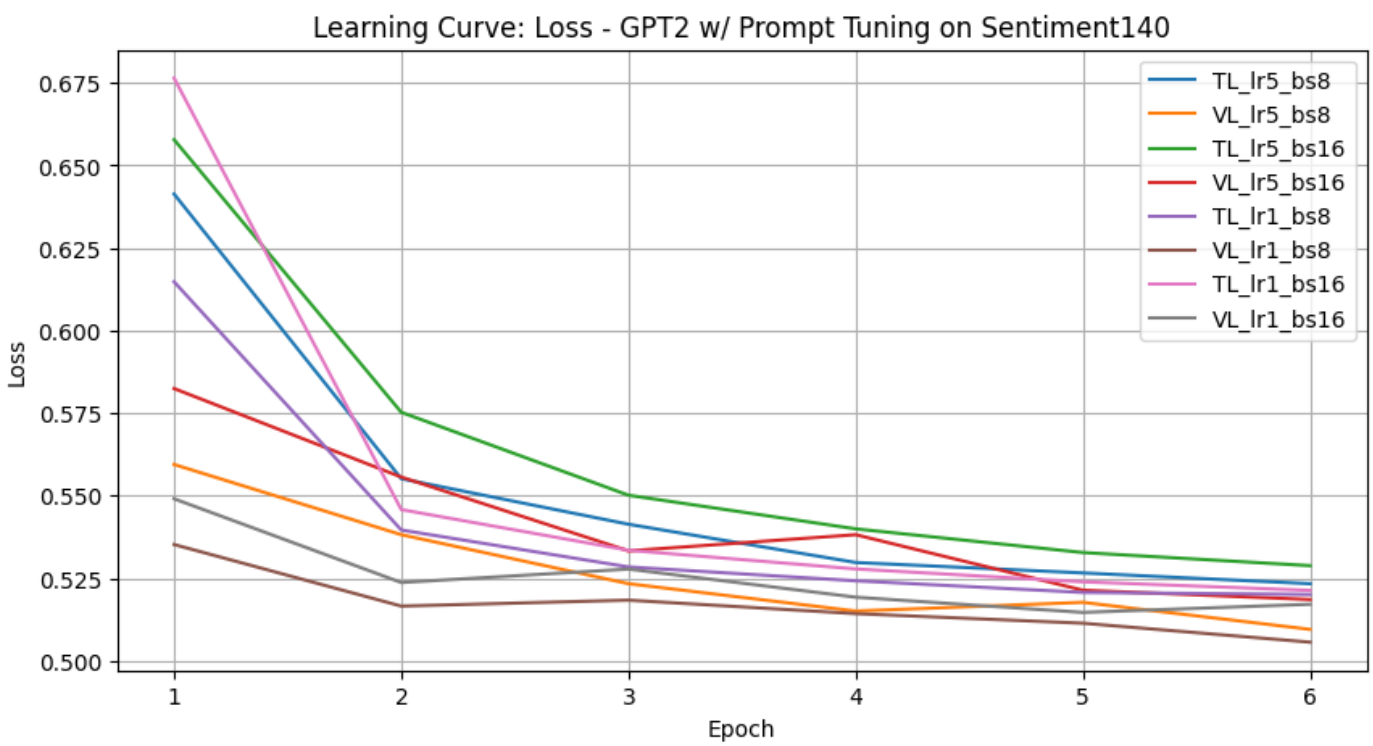



Prompt Tuning Learning Curves

```
# All Prompt Tuning Train/Val Acc Learning Curve
plt.figure(figsize=(10,5))
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs8, x="epoch", y="train_accuracy", labe
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs8, x="epoch", y="val_accuracy", label=
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs16, x="epoch", y="train_accuracy", lab
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs16, x="epoch", y="val_accuracy", label
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs8, x="epoch", y="train_accuracy", labe
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs8, x="epoch", y="val_accuracy", label=
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs16, x="epoch", y="train_accuracy", lab
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs16, x="epoch", y="val_accuracy", label
plt.title("Learning Curve: Accuracy - GPT2 w/ Prompt Tuning on Sentiment140")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# All Prompt Tuning Training and Validation Loss
plt.figure(figsize=(10,5))
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs8, x="epoch", y="train_loss", label="T
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs8, x="epoch", y="val_loss", label="VL_
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs16, x="epoch", y="train_loss", label="
sns.lineplot(data=gpt2_prompt_epochs_lr5_bs16, x="epoch", y="val_loss", label="VL
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs8, x="epoch", y="train_loss", label="T
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs8, x="epoch", y="val_loss", label="VL_
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs16, x="epoch", y="train_loss", label="
sns.lineplot(data=gpt2_prompt_epochs_lr1_bs16, x="epoch", y="val_loss", label="VL
plt.title("Learning Curve: Loss - GPT2 w/ Prompt Tuning on Sentiment140")
```

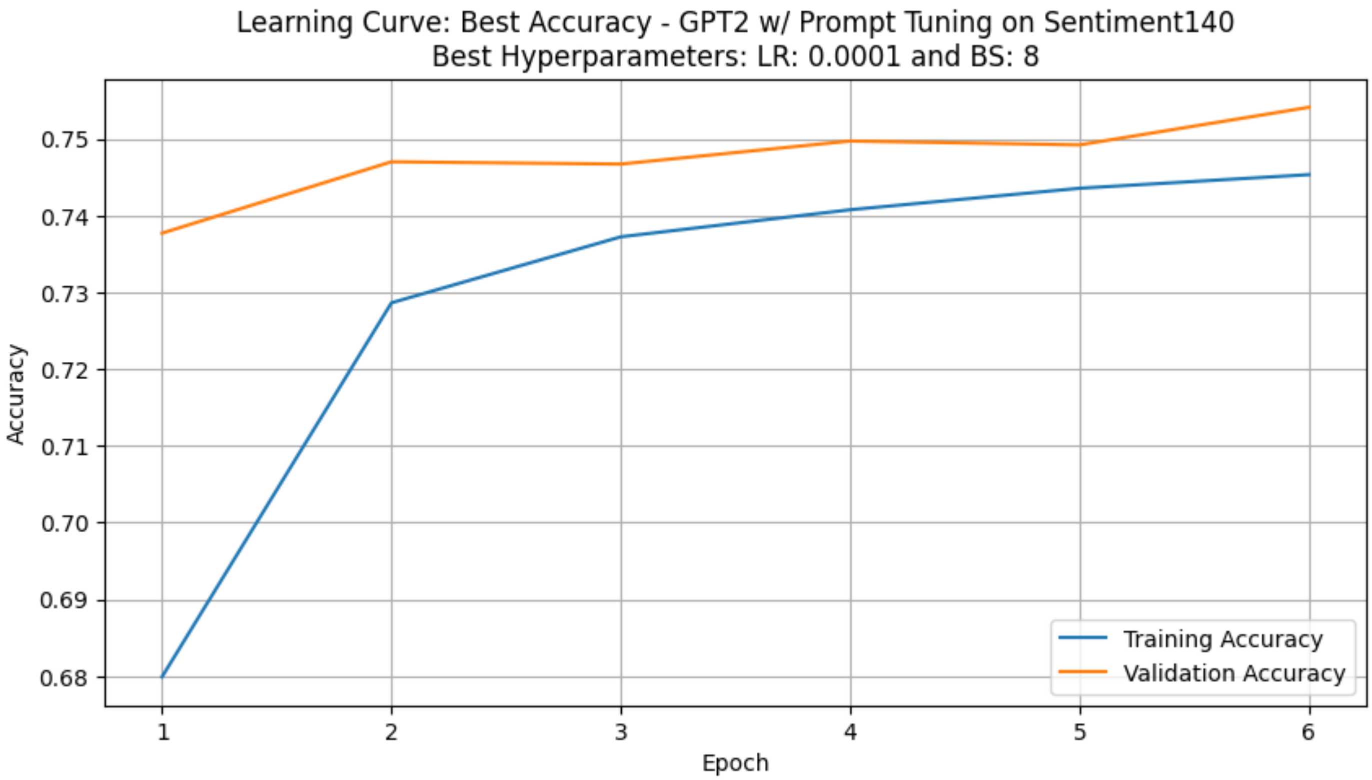
```
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

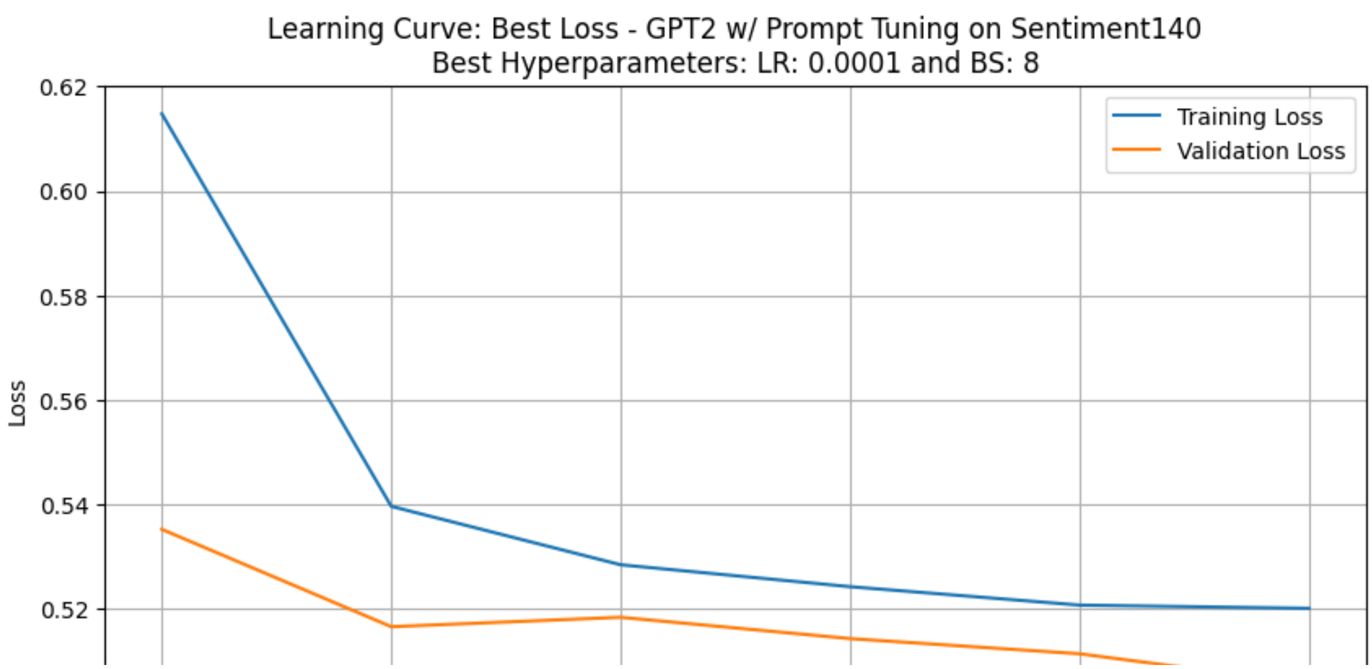


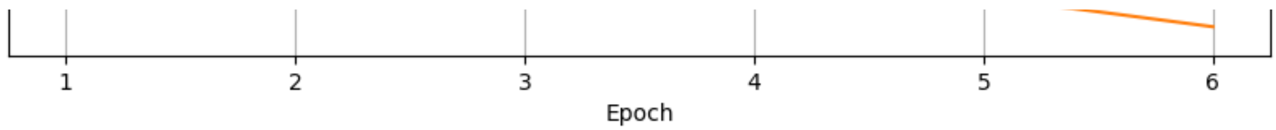



```
# Best Prompt Tuning Train/Val Acc Learning Curve

gpt2_prompt_epochs_map = {
    (5, 8): gpt2_prompt_epochs_lr5_bs8,
    (5, 16): gpt2_prompt_epochs_lr5_bs16,
    (1, 8): gpt2_prompt_epochs_lr1_bs8,
    (1, 16): gpt2_prompt_epochs_lr1_bs16
}

prompt_lr_mapping = {
    5e-5: "5e-5",
    1e-4: "1e-4"
}

prompt_best_lr_tag_for_map = {5e-5: 5, 1e-4: 1}[prompt_best_lr]

prompt_best_bs_tag = prompt_best_bs

prompt_epochs = gpt2_prompt_epochs_map[(prompt_best_lr_tag_for_map, prompt_best_b

# Best Prompt Tuning Training and Validation Accuracy
plt.figure(figsize=(10,5))
sns.lineplot(data=prompt_epochs, x="epoch", y="train_accuracy", label="Training A
sns.lineplot(data=prompt_epochs, x="epoch", y="val_accuracy", label="Validation A
plt.title(f"Learning Curve: Best Accuracy - GPT2 w/ Prompt Tuning on Sentiment140
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

# Best Prompt Tuning Training and Validation Loss
plt.figure(figsize=(10,5))
```

```
sns.lineplot(data=prompt_epochs, x="epoch", y="train_loss", label="Training Loss"
sns.lineplot(data=prompt_epochs, x="epoch", y="val_loss", label="Validation Loss"
plt.title(f"Learning Curve: Best Loss - GPT2 w/ Prompt Tuning on Sentiment140\nBe
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```






PEFT Method Comparison:

```
# Final results per BitFit/LoRA/Prompt Tuning Implementation

gpt2_bf_results = pd.read_csv('/content/sent_gpt2_bitfit_results.csv')
gpt2_lora_results = pd.read_csv('/content/sent_gpt2_lora_results.csv')
gpt2_prompt_results = pd.read_csv('/content/sent_gpt2_prompt_results.csv')

# Table of comparisons
comparison = pd.DataFrame({
    "Method": ["BitFit", "LoRA", "Prompt Tuning"],
    "Best Validation F1": [
        gpt2_bf_results["f1"].max(),
        gpt2_lora_results["f1"].max(),
        gpt2_prompt_results["f1"].max()
    ],
    "Best Validation Accuracy": [
        gpt2_bf_results["accuracy"].max(),
        gpt2_lora_results["accuracy"].max(),
        gpt2_prompt_results["accuracy"].max()
    ],
    "Runtime (sec)": [
        gpt2_bf_results["training_time"].sum(),
        gpt2_lora_results["training_time"].sum(),
        gpt2_prompt_results["training_time"].sum()
    ],
    "Inference Time (sec)": [
```

```
        gpt2_bf_results["inference_time"].sum(),
        gpt2_lora_results["inference_time"].sum(),
        gpt2_prompt_results["inference_time"].sum()
    ],
    "Max GPU Memory (GB)": [
        gpt2_bf_results["max_memory"].max(),
        gpt2_lora_results["max_memory"].max(),
        gpt2_prompt_results["max_memory"].max()
    ]
})

print("\nFinal Validation Performance PEFT Comparison - GPT2 on Sentiment140:")
display(comparison)
```

```
Final Validation Performance PEFT Comparison - GPT2 on Sentiment140:
```

| | Method | Best Validation F1 | Best Validation Accuracy | Runtime (sec) | Inference Time (sec) | Max GPU Memory (GB) |
|---|---|---|---|---|---|---|
| 0 | BitFit | 0.800287 | 0.8007 | 4434.319335 | 77.362442 | 5.89722 |
| 1 | LoRA | 0.811016 | 0.8111 | 6469.675971 | 77.727232 | 5.89722 |

Next steps: Generate code with comparison | View recommended plots | New interactive sheet

```
# Load overall results where inference_time is stored
gpt2_prompt_results = pd.read_csv('/content/sent_gpt2_prompt_results.csv')
gpt2_bf_results = pd.read_csv('/content/sent_gpt2_bitfit_results.csv')
gpt2_lora_results = pd.read_csv('/content/sent_gpt2_lora_results.csv')

# Manually map best learning rates to filename tags (from before)
lr_tag_mapping = {
    5e-5: "5e-05",
    1e-4: "0.0001"
}
bf_best_lr_tag = lr_tag_mapping[bf_best_lr]
lora_best_lr_tag = lr_tag_mapping[lora_best_lr]
prompt_best_lr_tag = lr_tag_mapping[prompt_best_lr]

# Load best inference metric summaries
bf_inf = pd.read_csv(f'/content/sent_gpt2_bitfit_inference_metrics_summary_lr{bf_be
lora_inf = pd.read_csv(f'/content/sent_gpt2_lora_inference_metrics_summary_lr{lora_
```

```
prompt_inf = pd.read_csv(f'/content/sent_gpt2_prompt_inference_metrics_summary_lr{p

# Extract inference times
bf_inference_time = gpt2_bf_results[
    (gpt2_bf_results["learning_rate"] == bf_best_lr) &
    (gpt2_bf_results["batch_size"] == bf_best_bs)
]["inference_time"].values[0]

lora_inference_time = gpt2_lora_results[
    (gpt2_lora_results["learning_rate"] == lora_best_lr) &
    (gpt2_lora_results["batch_size"] == lora_best_bs)
]["inference_time"].values[0]

prompt_inference_time = gpt2_prompt_results[
    (gpt2_prompt_results["learning_rate"] == prompt_best_lr) &
    (gpt2_prompt_results["batch_size"] == prompt_best_bs)
]["inference_time"].values[0]

# Table of best per-implementation metrics (based on best lr and bs per PEFT methoc
final_test_results = pd.DataFrame({
    "Method": ["BitFit", "LoRA", "Prompt Tuning"],
    "Test Accuracy": [
        bf_inf.loc["accuracy", "precision"],
        lora_inf.loc["accuracy", "precision"],
        prompt_inf.loc["accuracy", "precision"]
    ],
    "F1 Macro": [
        bf_inf.loc["macro avg", "f1-score"],
        lora_inf.loc["macro avg", "f1-score"],
        prompt_inf.loc["macro avg", "f1-score"]
    ],
    "F1 Weighted": [
        bf_inf.loc["weighted avg", "f1-score"],
        lora_inf.loc["weighted avg", "f1-score"],
        prompt_inf.loc["weighted avg", "f1-score"]
    ],
    "Inference Time (sec)": [
        bf_inference_time,
        lora_inference_time,
        prompt_inference_time
    ]
})

print("\nFinal Test Set Inference Performance PEFT Comparison - GPT2 on Sentiment14
display(final_test_results)
```

Final Test Set Inference Performance PEFT Comparison - GPT2 on Sentiment140:

| | Method | Test Accuracy | F1 Macro | F1 Weighted | Inference Time (sec) |
|---|---|---|---|---|---|
| 0 | BitFit | 0.8007 | 0.800287 | 0.800287 | 19.601799 |
| 1 | LoRA | 0.8111 | 0.811016 | 0.811016 | 18.977706 |
| 2 | Prompt | 0.7541 | 0.754090 | 0.754090 | 626.959952 |

Next steps: Generate code with final_test_results | View recommended plots | New interactive s

```
# Zip the entire /content folder
!zip -r /content/GPT_Sentiment_solo.zip /content

# Download the zipped file
from google.colab import files
files.download('/content/GPT_Sentiment_solo.zip')
```

```
  adding: content/ (stored 0%)
  adding: content/.config/ (stored 0%)
  adding: content/.config/.last_opt_in_prompt.yaml (stored 0%)
  adding: content/.config/active_config (stored 0%)
  adding: content/.config/config_sentinel (stored 0%)
  adding: content/.config/.last_survey_prompt.yaml (stored 0%)
  adding: content/.config/configurations/ (stored 0%)
  adding: content/.config/configurations/config_default (deflated 15%)
  adding: content/.config/hidden_gcloud_config_universe_descriptor_data_cache_
  adding: content/.config/logs/ (stored 0%)
  adding: content/.config/logs/2025.04.24/ (stored 0%)
  adding: content/.config/logs/2025.04.24/18.19.38.522066.log (deflated 58%)
  adding: content/.config/logs/2025.04.24/18.19.46.929623.log (deflated 87%)
  adding: content/.config/logs/2025.04.24/18.19.56.709493.log (deflated 57%)
  adding: content/.config/logs/2025.04.24/18.19.57.353004.log (deflated 56%)
  adding: content/.config/logs/2025.04.24/18.19.48.089267.log (deflated 58%)
  adding: content/.config/logs/2025.04.24/18.19.17.922226.log (deflated 93%)
  adding: content/.config/default_configs.db (deflated 98%)
  adding: content/.config/gce (stored 0%)
  adding: content/.config/.last_update_check.json (deflated 23%)
  adding: content/sent_gpt2_bitfit_epoch_logs_lr0.0001_bs16.csv (deflated 48%)
  adding: content/sent_gpt2_bitfit_inference_predictions_lr5e-05_bs16.csv (def
  adding: content/sent_gpt2_lora_inference_metrics_summary_lr0.0001_bs8.csv (c
  adding: content/sent_gpt2_bitfit_epoch_logs_lr0.0001_bs8.csv (deflated 48%)
  adding: content/sent_gpt2_prompt_inference_predictions_lr5e-05_bs8.csv (defl
  adding: content/sent_gpt2_bitfit_inference_predictions_lr5e-05_bs8.csv (defl
  adding: content/sent_gpt2_lora_inference_metrics_summary_lr5e-05_bs16.csv (c
```

```
adding: content/sent_gpt2_prompt_inference_predictions_lr0.0001_bs16.csv (de
adding: content/sent_gpt2_bitfit_inference_predictions_lr0.0001_bs16.csv (de
adding: content/sent_gpt2_lora_epoch_logs_lr0.0001_bs8.csv (deflated 47%)
adding: content/sent_gpt2_prompt_inference_predictions_lr0.0001_bs8.csv (def
adding: content/sent_gpt2_bitfit_inference_predictions_lr0.0001_bs8.csv (def
adding: content/sentiment140.py (deflated 62%)
adding: content/sent_gpt2_lora_inference_metrics_summary_lr0.0001_bs16.csv (
adding: content/sent_gpt2_prompt_inference_metrics_summary_lr5e-05_bs16.csv
adding: content/sent_gpt2_prompt_epoch_logs_lr5e-05_bs8.csv (deflated 48%)
adding: content/sent_gpt2_lora_epoch_logs_lr5e-05_bs16.csv (deflated 47%)
adding: content/sent_gpt2_prompt_inference_metrics_summary_lr0.0001_bs16.csv
adding: content/sent_gpt2_bitfit_inference_metrics_summary_lr0.0001_bs8.csv
adding: content/sent_gpt2_lora_epoch_logs_lr5e-05_bs8.csv (deflated 46%)
adding: content/sent_gpt2_bitfit_epoch_logs_lr5e-05_bs16.csv (deflated 47%)
adding: content/sent_gpt2_lora_epoch_logs_lr0.0001_bs16.csv (deflated 48%)
adding: content/sent_gpt2_lora_inference_predictions_lr5e-05_bs8.csv (deflat
adding: content/sent_gpt2_lora_results.csv (deflated 42%)
adding: content/sent_gpt2_prompt_epoch_logs_lr5e-05_bs16.csv (deflated 48%)
adding: content/sent_gpt2_prompt_inference_predictions_lr5e-05_bs16.csv (def
adding: content/sent_gpt2_prompt_inference_metrics_summary_lr5e-05_bs8.csv (
adding: content/sent_gpt2_lora_inference_metrics_summary_lr5e-05_bs8.csv (de
adding: content/sent_gpt2_bf_final_comparison_bitfit.csv (deflated 19%)
adding: content/sent_gpt2_prompt_final_comparison_prompt_tuning.csv (deflate
adding: content/sent_gpt2_lora_final_comparison_lora.csv (deflated 19%)
adding: content/sent_gpt2_lora_inference_predictions_lr5e-05_bs16.csv (defla
adding: content/sent_gpt2_bitfit_results.csv (deflated 46%)
adding: content/sent_gpt2_lora_inference_predictions_lr0.0001_bs8.csv (defla
adding: content/sent_gpt2_prompt_epoch_logs_lr0.0001_bs8.csv (deflated 48%)
adding: content/sent_gpt2_prompt_results.csv (deflated 47%)
adding: content/sent_gpt2_prompt_inference_metrics_summary_lr0.0001_bs8.csv
adding: content/sent_gpt2_lora_inference_predictions_lr0.0001_bs16.csv (defl
adding: content/sent_gpt2_prompt_epoch_logs_lr0.0001_bs16.csv (deflated 49%)
adding: content/sent_gpt2_bitfit_epoch_logs_lr5e-05_bs8.csv (deflated 48%)
adding: content/sent_gpt2_bitfit_inference_metrics_summary_lr0.0001_bs16.csv
adding: content/sent_gpt2_bitfit_inference_metrics_summary_lr5e-05_bs8.csv (
adding: content/sent_gpt2_bitfit_inference_metrics_summary_lr5e-05_bs16.csv
adding: content/sample_data/ (stored 0%)
adding: content/sample_data/anscombe.json (deflated 83%)
adding: content/sample_data/README.md (deflated 39%)
adding: content/sample_data/mnist_test.csv (deflated 88%)
adding: content/sample_data/mnist_train_small.csv (deflated 88%)
adding: content/sample_data/california_housing_train.csv (deflated 79%)
adding: content/sample_data/california_housing_test.csv (deflated 76%)
```