# OPT125 Sentiment Analysis Experiments on IMDb 50k 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 OPT transformer mod

!pip install torch transformers datasets -q
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.2 MB/s eta 0:00:0
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 94.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 89.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 56.0 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.7 MB/s eta 0:00:0
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 11.8 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 42.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 19.3 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.7 MB/s eta 0:00:0
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 102.7 MB/s eta 0:00:0
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.2/491.2 kB 41.3 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 10.6 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 183.9/183.9 kB 16.1 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 13.4 MB/s eta 0:00:
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 17.1 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 OPT model"""

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

import time
from sklearn.metrics import classification_report, f1_score
```

```
""" We next instantiate (load) our IMDb 50k dataset"""

dataset_imdb = load_dataset("imdb")

full_imdb = concatenate_datasets([dataset_imdb["train"], dataset_imdb["test"]])

full_imdb_split = full_imdb.train_test_split(test_size=0.2, seed=42)

full_train = full_imdb_split["train"]
dataset = {"test": full_imdb_split["test"]}

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

```
/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(
README.md: 100%  7.81k/7.81k [00:00<00:00, 882kB/s]
train-00000-of-00001.parquet: 100%  21.0M/21.0M [00:00<00:00, 33.4MB/s]
test-00000-of-00001.parquet: 100%  20.5M/20.5M [00:00<00:00, 189MB/s]
unsupervised-00000-of-00001.parquet: 100%  42.0M/42.0M [00:00<00:00, 198MB/s]
Generating train split: 100%  25000/25000 [00:00<00:00, 90714.76 examples/s]
```

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

model_name = "facebook/opt-125m"
num_labels = 2
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=
model = model.to(device)
```

```
tokenizer_config.json: 100%  685/685 [00:00<00:00, 82.5kB/s]
config.json: 100%  651/651 [00:00<00:00, 76.4kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 6.28MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 20.7MB/s]
special_tokens_map.json: 100%  441/441 [00:00<00:00, 51.0kB/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 [00:01<00:00, 208MB/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
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%  251M/251M [00:02<00:00, 148MB/s]
```

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

tokenized_train = full_train.map(tokenize, batched=True)
tokenized_test = dataset["test"].map(tokenize, batched=True)

tokenized_train = tokenized_train.rename_column("label", "labels")
tokenized_test = tokenized_test.rename_column("label", "labels")

tokenized_train = tokenized_train.remove_columns(["text"])
tokenized_test = tokenized_test.remove_columns(["text"])

tokenized_dataset = {"train": tokenized_train, "test": tokenized_test}
```

```
Map: 100% 40000/40000 [00:09<00:00, 4419.81 examples/s]

Map: 100% 10000/10000 [00:02<00:00, 4290.60 examples/
```

```
from huggingface_hub import login
token = "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"


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

print("\nSample training examples:")
display(full_train[:5])

print("\nSample test examples:")
display(dataset["test"][:5])
```

```
Sample training examples:
{'text': ["Eugene O'Neill is acclaimed by some as America's leading
playwright, but for things like The Iceman Cometh, Long Day's Journey Into
Night, The Emperor Jones. Strange Interlude was a piece of experimentation he
concocted where the characters on stage, look aside to the audience and say
what they really are thinking and then resume conversation. It was a nine
hour production with a dinner break on Broadway, so you can safely assume a
lot has been sacrificed here.<br /><br />For the screen the voice over
regarding the thoughts is used for all the characters. It probably is a
technique better suited to the screen. Sir Laurence Olivier did very well
with it in his version of Hamlet. But Bill Shakespeare gave Olivier a lot
better story than O'Neill gave his players in this instance.<br /><br
/>Players like Clark Gable, Norma Shearer, Ralph Morgan, May Robson, etc. are
a lot more animated in most of their films than they are in Strange
Interlude. The story takes place over a 20 year period. Norma Shearer is a
```

Interlude. The story takes place over a 20 year period. Norma Shearer is a young woman whose intended is killed in World War I. She starts playing around quite a bit, although that part is not shown in this version. She makes the acquaintance of Alexander Kirkland and his friend Clark Gable. She also has as a perennial suitor, Ralph Morgan, a friend of her father's Henry B. Walthall.<br /><br />She marries Kirkland, but then is warned by his mother May Robson and shown that insanity gallops in that family to quote another literary work. Since Kirkland wants kids and Shearer and Robson think Kirkland's train will slip the track if he doesn't get one, Gable is recruited for breeding purposes. Of course you can see all the complications this can cause and O'Neill explores them all.<br /><br />Gable is so terribly miscast in an O'Neill production, but he was an up and coming player at MGM and did what they told him. Shearer does what she can to lift a very dreary story, but she seems defeated at the start. Best in the film is possibly Robson who puts some real bite in her dialog.<br /><br />Strange Interlude ran for 426 showings on Broadway in 1928-1929 and starred Glenn Anders and Lynn Fontanne in the Gable and Shearer parts. Perhaps no one could really have saved the film because two years earlier, Groucho Marx lampooned the stuffings out of it in Animal Crackers. After seeing what he did, I don't think the movie going public took it too seriously.<br /><br />And since it's not the best of O'Neill, neither could I.",

'I saw this movie in 1959 when I was 11 years old at a drive-in theater with my family.<br /><br />Way back then, I thought it was very funny . . . even though I was too young to understand 90% of what makes this marvelous movie such a delight! I saw it again this morning on "Turner South". As I watched it, I was absolutely convulsed with laughter! "The Mating Game" is a unique classic from a by-gone age. If you\'re too young to have experienced the enchanting period in history that produced this film, I feel very sorry for you. There\'s no way you can watch movies like this and understand how they can (even today) deliver such a delightful slice of heaven to "old timers" like me.<br /><br />Having said that, all I can do is respectfully request that younger people refrain from commenting on films like "The Mating Game".<br /><br />Movies like this were made for the generation that preceded the current group of your people. And as such, these films speak a very different language than any of you can understand.<br /><br />In other words \x96 if you don\'t understand the issues the film is addressing, please don\'t embarrass yourself by offering comments which \x96 frankly \x96 make no sense.',

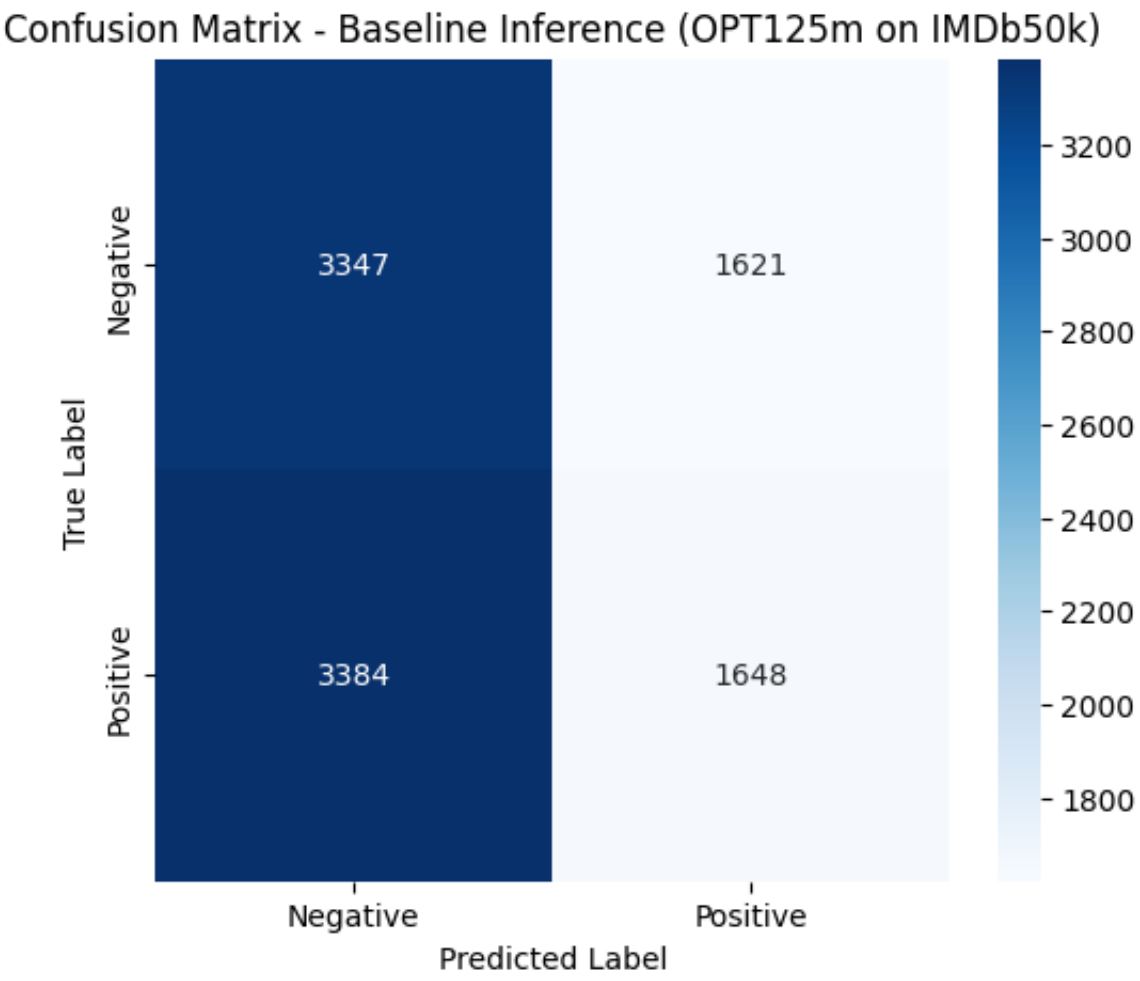
"It's not my fault. My girlfriend made me watch it.<br /><br />There is nothing positive to say about this film. There has been for many years an idea that Madonna could act but she can't. There has been an idea for years that Guy Ritchie is a great director but he is only middling. An embarrassment all round. <br /><br />",

"This is another Alien imitation and not a very good one at that.Replace outer space with the South African desert,throw in the same ingredients,a group of people stranded in an inhospitable landscape, have them hunted down by an alien creature and you have the same old story of a very ordinary film trying to ape a classic film. A group of miners and scientists go on a hunt for some missing colleagues and find their bones in the desert stripped clean of flesh.Their vehicle breaks down and they head for civilisation while being

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

from torch.utils.data import DataLoader
from transformers import default_data_collator

train_loader = DataLoader(tokenized_dataset["train"], batch_size=16, shuffle=True
test_loader = DataLoader(tokenized_dataset["test"], batch_size=16, shuffle=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
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["labels"].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 - OPT125m on IMDb50k\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 (OPT125m on IMDb50k)")
plt.show()
```

```
Baseline Inference Performance - OPT125m on IMDb50k


Test Accuracy   : 0.4995
F1 Score (macro): 0.4846
F1 Score (weighted): 0.4841
Inference Time  : 9.93s

Classification Report:
              precision    recall  f1-score   support

    Negative       0.50      0.67      0.57      4968
    Positive       0.50      0.33      0.40      5032

    accuracy                           0.50     10000
   macro avg       0.50      0.50      0.48     10000
weighted avg       0.50      0.50      0.48     10000
```



# 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 OPT-125m model using LoRA and output dataset generation (saved as

results = []

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

        # loading OPT model
        model_name = "facebook/opt-125m"
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        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=["q_proj", "v_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=True, desc=f"Epoch {epoch}/{epoch
    for step, batch in enumerate(loop):
        batch = {
            "input_ids": batch["input_ids"].to(device),
            "attention_mask": batch["attention_mask"].to(device),
            "labels": batch["labels"].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
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["labels"].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: OPT125m w/ LoRA on IMDb50k\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"imdb_opt_lora_epoch_logs_lr{lr}_bs{batch_size}.csv

    # saver inference metrics per lr/bs
    metrics_summary_df = pd.DataFrame(report).transpose()
    metrics_summary_df.to_csv(f"imdb_opt_lora_inference_metrics_summary_lr{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"imdb_opt_lora_inference_predictions_lr{lr}_bs{bat

    # 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("imdb_opt_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("imdb_opt_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])
```

```
Classification Report: OPT125M w/ LoRA on IMDb50k

              precision    recall  f1-score   support

    Negative       0.86      0.84      0.85      4968
    Positive       0.85      0.87      0.86      5032

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000

Running LoRA with LR=0.0001, batch_size=8
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
```

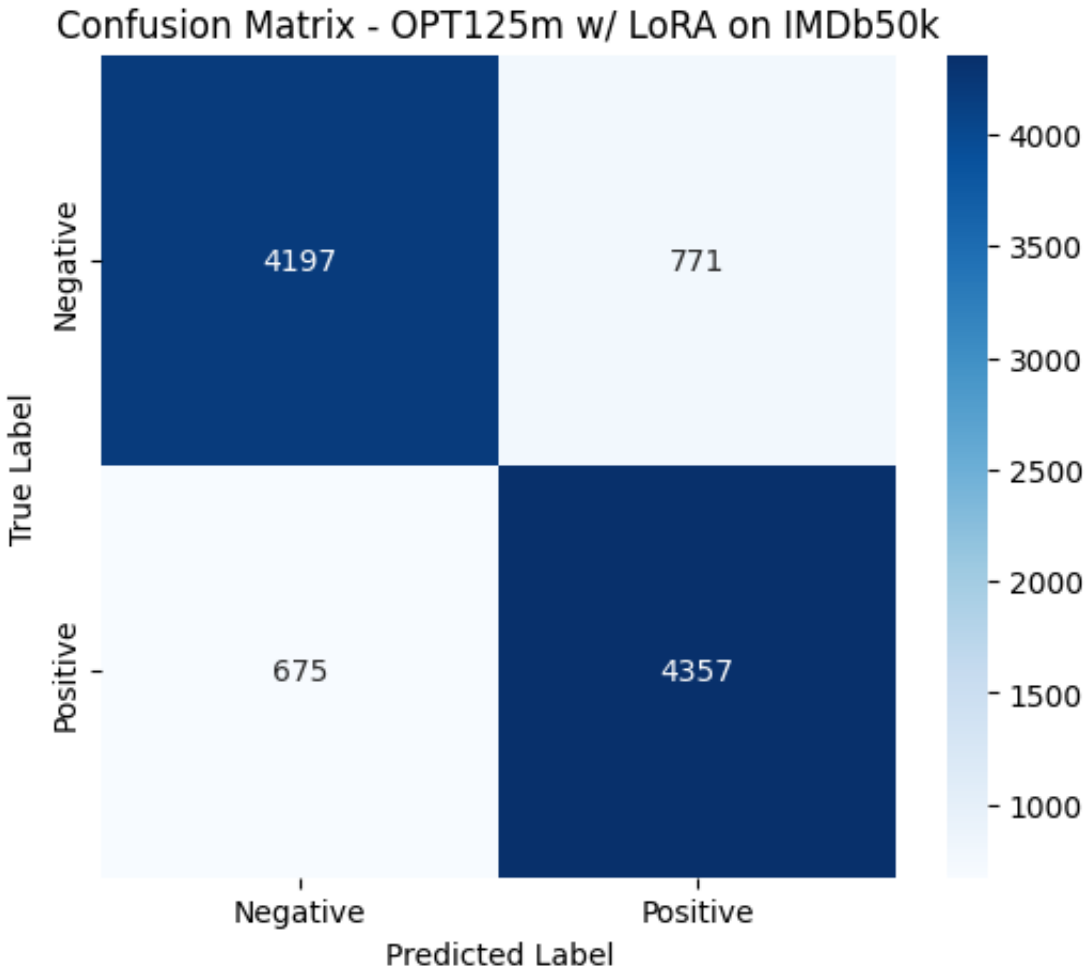
```
Epoch 1/6: 100%|██████████| 5000/5000 [04:06<00:00, 20.26it/s]
Epoch 2/6: 100%|██████████| 5000/5000 [04:08<00:00, 20.08it/s]
Epoch 3/6: 100%|██████████| 5000/5000 [04:07<00:00, 20.17it/s]
Epoch 4/6: 100%|██████████| 5000/5000 [04:06<00:00, 20.25it/s]
Epoch 5/6: 100%|██████████| 5000/5000 [04:07<00:00, 20.22it/s]
Epoch 6/6: 100%|██████████| 5000/5000 [04:07<00:00, 20.20it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8227
F1 Score (macro)   : 0.8227
F1 Score (weighted): 0.8227
Inference Time     : 4362.36 seconds

Classification Report: OPT125m w/ LoRA on IMDb50k

              precision    recall  f1-score   support

    Negative       0.81      0.84      0.82      4968
    Positive       0.83      0.81      0.82      5032

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000

Running LoRA with LR=0.0001, batch_size=16
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
Epoch 1/6: 100%|██████████| 2500/2500 [02:11<00:00, 19.06it/s]
Epoch 2/6: 100%|██████████| 2500/2500 [02:11<00:00, 19.00it/s]
Epoch 3/6: 100%|██████████| 2500/2500 [02:11<00:00, 18.96it/s]
Epoch 4/6: 100%|██████████| 2500/2500 [02:11<00:00, 18.98it/s]
Epoch 5/6: 100%|██████████| 2500/2500 [02:10<00:00, 19.13it/s]
Epoch 6/6: 100%|██████████| 2500/2500 [02:11<00:00, 18.99it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8210
F1 Score (macro)   : 0.8195
F1 Score (weighted): 0.8196
Inference Time     : 5201.84 seconds

Classification Report: OPT125m w/ LoRA on IMDb50k

              precision    recall  f1-score   support

    Negative       0.89      0.73      0.80      4968
    Positive       0.78      0.91      0.84      5032
```

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

```
# Construct filename
best_report_file = f"imdb_opt_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"imdb_opt_lora_inference_predictions_lr{lora_best_lr
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 - OPT125m w/ LoRA on IMDb50k")
plt.show()
```

```
Classification Report for Best Configuration:
     Unnamed: 0  precision    recall  f1-score     support
0             0   0.861453  0.844807  0.853049   4968.0000
1             1   0.849649  0.865859  0.857677   5032.0000
2      accuracy   0.855400  0.855400  0.855400      0.8554
3     macro avg   0.855551  0.855333  0.855363  10000.0000
4  weighted avg   0.855513  0.855400  0.855378  10000.0000

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

```
9999         1         1

[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 OPT-125m model using BitFit and output dataset generation (saved

results = []

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

        # loading OPT model
        model_name = "facebook/opt-125m"
        tokenizer = AutoTokenizer.from_pretrained(model_name)
        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=True, desc=f"Epoch {epoch}/{epoch
    for step, batch in enumerate(loop):
        batch = {
            "input_ids": batch["input_ids"].to(device),
            "attention_mask": batch["attention_mask"].to(device),
            "labels": batch["labels"].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
    model.eval()
    val_running_loss = 0.0
    y_true, y_pred = [], []
    inference_start = time.time()
    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["labels"].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: OPT125m w/ BitFit on IMDb50k\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"imdb_opt_bitfit_epoch_logs_lr{lr}_bs{batch_size}.c

        # saver inference metrics per lr/bs
        metrics_summary_df = pd.DataFrame(report).transpose()
        metrics_summary_df.to_csv(f"imdb_opt_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"imdb_opt_bitfit_inference_predictions_lr{lr}_bs{b

        # 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("imdb_opt_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("imdb_opt_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])
```

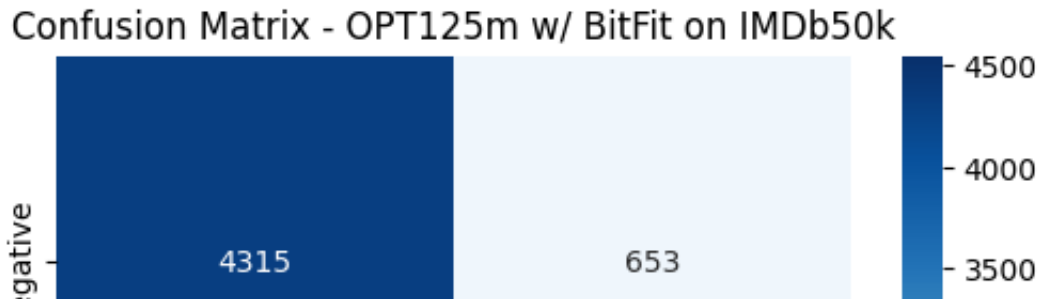
```
Running BitFit with LR=5e-05, batch_size=8
Some weights of OPTForSequenceClassification were not initialized from the mod
You should probably TRAIN this model on a down-stream task to be able to use i
Epoch 1/6: 100%|██████████| 5000/5000 [03:01<00:00, 27.57it/s]
Epoch 2/6: 100%|██████████| 5000/5000 [03:00<00:00, 27.76it/s]
Epoch 3/6: 100%|██████████| 5000/5000 [02:59<00:00, 27.84it/s]
Epoch 4/6: 100%|██████████| 5000/5000 [02:59<00:00, 27.83it/s]
Epoch 5/6: 100%|██████████| 5000/5000 [03:00<00:00, 27.67it/s]
Epoch 6/6: 100%|██████████| 5000/5000 [02:59<00:00, 27.79it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8736
F1 Score (macro)   : 0.8727
F1 Score (weighted): 0.8727
Inference Time     : 15.50 seconds

Classification Report: OPT125m w/ BitFit on IMDb50k

              precision    recall  f1-score   support

    Negative       0.94      0.79      0.86      4968
    Positive       0.82      0.95      0.88      5032

    accuracy                           0.87     10000
```

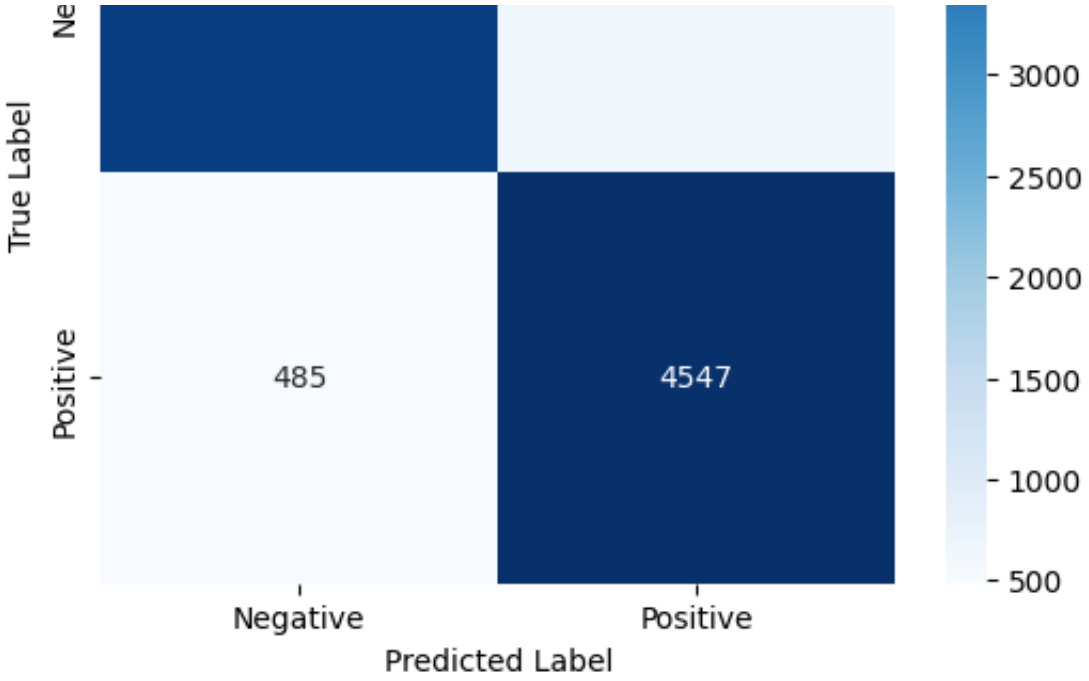
```
   macro avg       0.88      0.87      0.87     10000
weighted avg       0.88      0.87      0.87     10000

Running BitFit with LR=5e-05, batch_size=16
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
Epoch 1/6: 100%|██████████| 2500/2500 [01:36<00:00, 26.01it/s]
Epoch 2/6: 100%|██████████| 2500/2500 [01:35<00:00, 26.06it/s]
Epoch 3/6: 100%|██████████| 2500/2500 [01:36<00:00, 25.95it/s]
Epoch 4/6: 100%|██████████| 2500/2500 [01:35<00:00, 26.25it/s]
Epoch 5/6: 100%|██████████| 2500/2500 [01:38<00:00, 25.49it/s]
Epoch 6/6: 100%|██████████| 2500/2500 [01:36<00:00, 26.02it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.8769
F1 Score (macro)   : 0.8768
F1 Score (weighted): 0.8768
Inference Time     : 8.89 seconds

Classification Report: OPT125m w/ BitFit on IMDb50k

              precision    recall  f1-score   support

    Negative       0.85      0.91      0.88      4968
    Positive       0.90      0.85      0.87      5032

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Running BitFit with LR=0.0001, batch_size=8
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
Epoch 1/6: 100%|██████████| 5000/5000 [02:59<00:00, 27.86it/s]
Epoch 2/6: 100%|██████████| 5000/5000 [02:59<00:00, 27.82it/s]
```

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

# Construct filename
best_report_file = f"imdb_opt_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"imdb_opt_bitfit_inference_predictions_lr{bf_best_lr
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 - OPT125m w/ BitFit on IMDb50k")
plt.show()
```

```
Classification Report for Best Configuration:
     Unnamed: 0  precision    recall  f1-score     support
0             0   0.898958  0.868559  0.883497   4968.0000
1             1   0.874423  0.903617  0.888780   5032.0000
2      accuracy   0.886200  0.886200  0.886200      0.8862
3     macro avg   0.886691  0.886088  0.886139  10000.0000
4  weighted avg   0.886612  0.886200  0.886156  10000.0000

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

[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 """

results = []

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

```
# loading OPT model
model_name = "facebook/opt-125m"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, nu

# 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=batc
test_dataloader = DataLoader(tokenized_dataset["test"], batch_size=batch_

# 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}/{epoch
    for step, batch in enumerate(loop):
        batch = {
          "input_ids": batch["input_ids"].to(device),
          "attention_mask": batch["attention_mask"].to(device),
          "labels": batch["labels"].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["labels"].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"]

    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: OPT125m w/ Prompt Tuning on IMDb5
            print(classification_report(y_true, y_pred, target_names=["Negati

        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"imdb_opt_prompt_epoch_logs_lr{lr}_bs{batch_size}.c

    # save inference metrics per lr/bs
    metrics_summary_df = pd.DataFrame(report).transpose()
    metrics_summary_df.to_csv(f"imdb_opt_prompt_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"imdb_opt_prompt_inference_predictions_lr{lr}_bs{b

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

    # 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("imdb_opt_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("imdb_opt_prompt_final_comparison_prompt_tuning.csv", ind

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

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

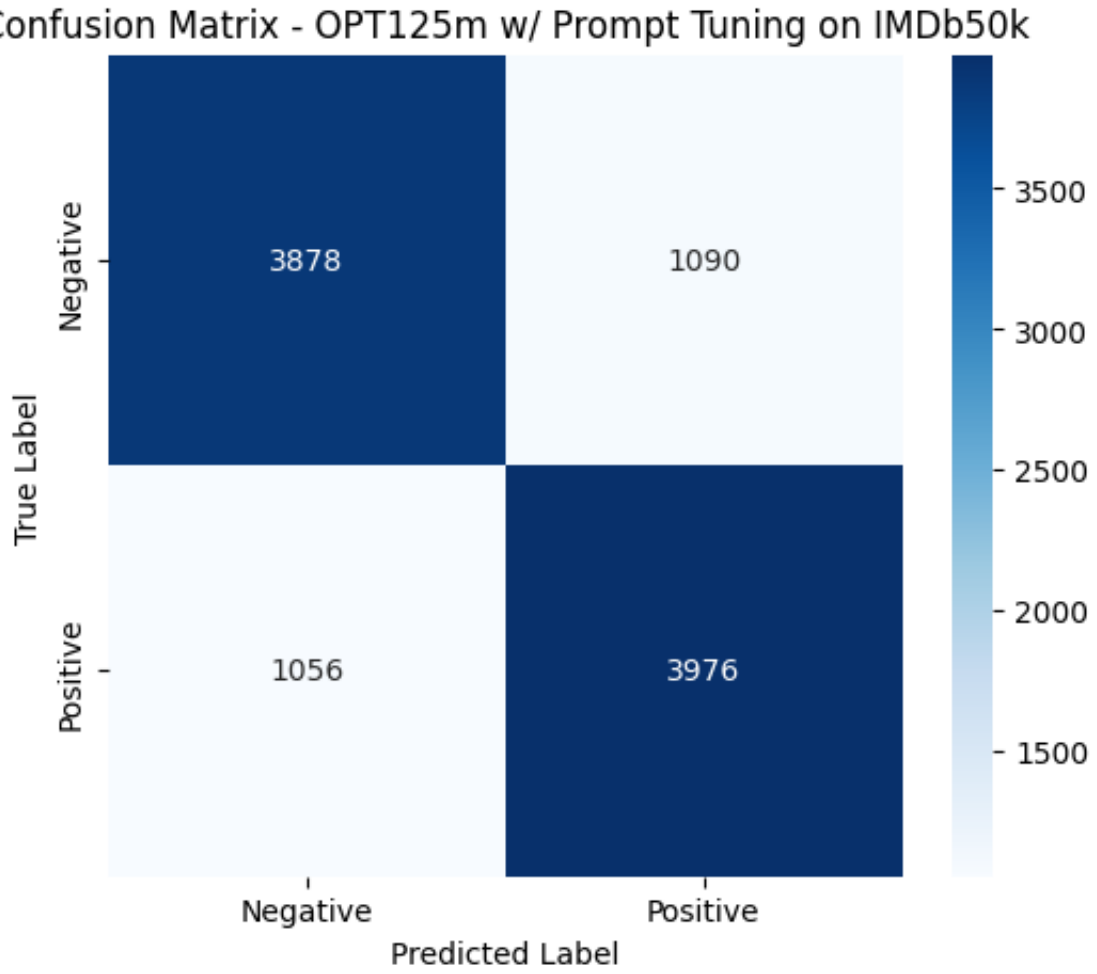
```
              precision    recall  f1-score   support

    Negative       0.79      0.77      0.78      4968
    Positive       0.78      0.79      0.78      5032

    accuracy                           0.78     10000
```

```
   macro avg       0.78      0.78      0.78     10000
weighted avg       0.78      0.78      0.78     10000

Running Prompt Tuning with LR=0.0001, batch_size=8
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
Epoch 1/6: 100%|██████████| 5000/5000 [01:21<00:00, 61.60it/s]
Epoch 2/6: 100%|██████████| 5000/5000 [01:20<00:00, 62.23it/s]
Epoch 3/6: 100%|██████████| 5000/5000 [01:20<00:00, 61.73it/s]
Epoch 4/6: 100%|██████████| 5000/5000 [01:21<00:00, 61.68it/s]
Epoch 5/6: 100%|██████████| 5000/5000 [01:20<00:00, 61.75it/s]
Epoch 6/6: 100%|██████████| 5000/5000 [01:20<00:00, 62.01it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7854
F1 Score (macro)   : 0.7854
F1 Score (weighted): 0.7854
Inference Time     : 584.57 seconds

Classification Report: OPT125m w/ Prompt Tuning on IMDb50k

              precision    recall  f1-score   support

    Negative       0.79      0.78      0.78      4968
    Positive       0.78      0.79      0.79      5032

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.79     10000
weighted avg       0.79      0.79      0.79     10000

Running Prompt Tuning with LR=0.0001, batch_size=16
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
Epoch 1/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.76it/s]
Epoch 2/6: 100%|██████████| 2500/2500 [00:44<00:00, 56.07it/s]
Epoch 3/6: 100%|██████████| 2500/2500 [00:45<00:00, 55.42it/s]
Epoch 4/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.78it/s]
Epoch 5/6: 100%|██████████| 2500/2500 [00:44<00:00, 55.80it/s]
Epoch 6/6: 100%|██████████| 2500/2500 [00:44<00:00, 56.38it/s]

[Final Epoch 6] Inference Metrics:
Test Accuracy      : 0.7847
F1 Score (macro)   : 0.7846
F1 Score (weighted): 0.7846
Inference Time     : 324.31 seconds

Classification Report: OPT125m w/ Prompt Tuning on IMDb50k

              precision    recall  f1-score   support
```

```
        Negative       0.77      0.81      0.79      4968
        Positive       0.80      0.76      0.78      5032

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

# Construct filename
best_report_file = f"imdb_opt_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"imdb_opt_prompt_inference_predictions_lr{prompt_bes
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 - OPT125m w/ Prompt Tuning on IMDb50k")
plt.show()
```

```
Classification Report for Best Configuration:
     Unnamed: 0  precision    recall  f1-score     support
0             0   0.785975  0.780596  0.783276   4968.0000
1             1   0.784840  0.790143  0.787483   5032.0000
2      accuracy   0.785400  0.785400  0.785400      0.7854
3     macro avg   0.785407  0.785369  0.785379  10000.0000
4  weighted avg   0.785404  0.785400  0.785393  10000.0000

Inference Predictions for Best Configuration:
      y_true  y_pred
0          0       0
1          1       1
2          1       1
3          0       0
4          1       1
```

```
...      ...      ...
9995       1        1
9996       1        1
9997       1        1
9998       1        1
9999       1        1

[10000 rows x 2 columns]
```




## Begin Visualization of IMDb 50k Results

Load all dataframes from above training of 3 PEFT methods

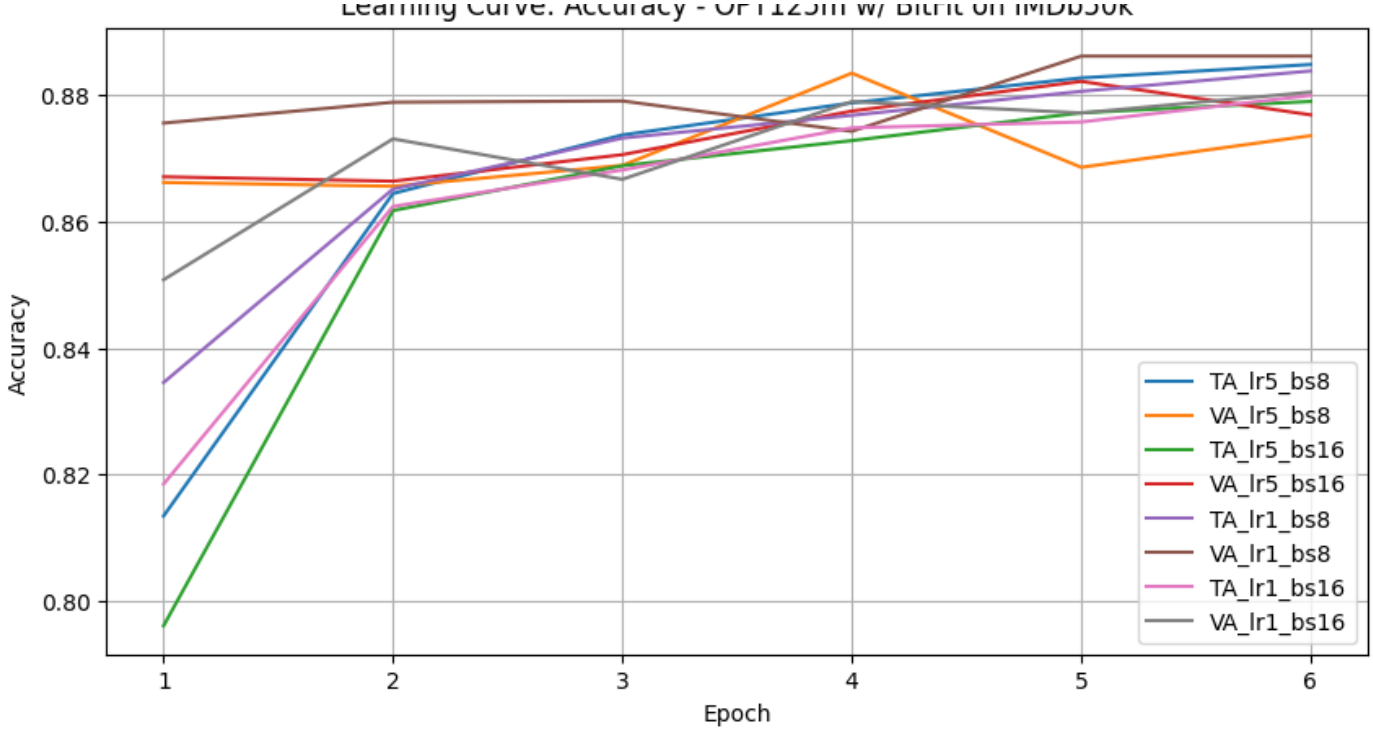
```
""" Output from our OPT125M_Sentiment140_IMDb50k.ipynb file, we read in the .csv
Note that these .csv file paths suggest they should be uploaded to session storag

# OPT125m PEFT-wise results across bf/lora/prompt tuning
```

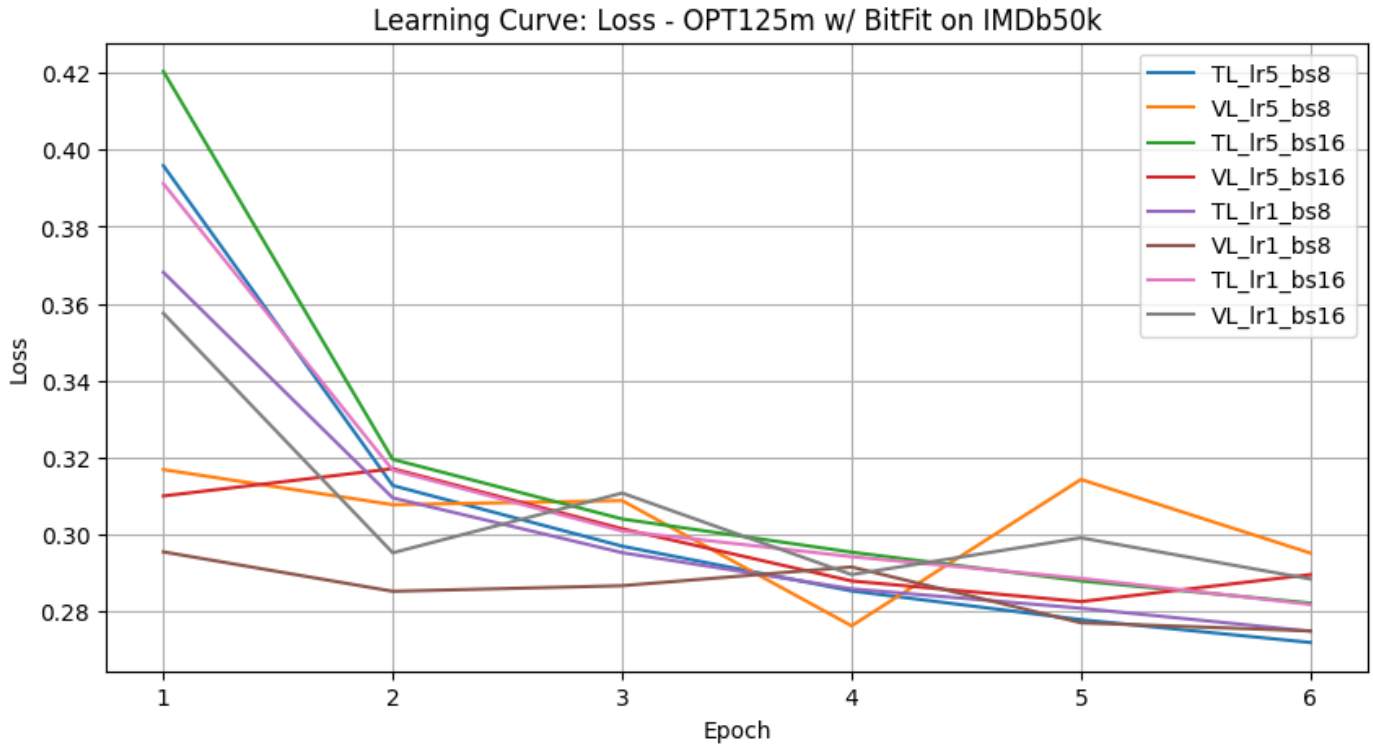
```
opt_bf_results = pd.read_csv('/content/imdb_opt_bitfit_results.csv')
opt_lora_results = pd.read_csv('/content/imdb_opt_lora_results.csv')
opt_prompt_results = pd.read_csv('/content/imdb_opt_prompt_results.csv')

# OPT125m per-epoch performance logs (for LC generation) across bf/lora/prompt tu
opt_lora_epochs_lr5_bs8 = pd.read_csv('/content/imdb_opt_lora_epoch_logs_lr5e-05_
opt_lora_epochs_lr5_bs16 = pd.read_csv('/content/imdb_opt_lora_epoch_logs_lr5e-05
opt_lora_epochs_lr1_bs8 = pd.read_csv('/content/imdb_opt_lora_epoch_logs_lr0.0001
opt_lora_epochs_lr1_bs16 = pd.read_csv('/content/imdb_opt_lora_epoch_logs_lr0.000
opt_bf_epochs_lr5_bs8 = pd.read_csv('/content/imdb_opt_bitfit_epoch_logs_lr5e-05_
opt_bf_epochs_lr5_bs16 = pd.read_csv('/content/imdb_opt_bitfit_epoch_logs_lr5e-05
opt_bf_epochs_lr1_bs8 = pd.read_csv('/content/imdb_opt_bitfit_epoch_logs_lr0.0001
opt_bf_epochs_lr1_bs16 = pd.read_csv('/content/imdb_opt_bitfit_epoch_logs_lr0.000
opt_prompt_epochs_lr5_bs8 = pd.read_csv('/content/imdb_opt_prompt_epoch_logs_lr5e
opt_prompt_epochs_lr5_bs16 = pd.read_csv('/content/imdb_opt_prompt_epoch_logs_lr5
opt_prompt_epochs_lr1_bs8 = pd.read_csv('/content/imdb_opt_prompt_epoch_logs_lr0.
opt_prompt_epochs_lr1_bs16 = pd.read_csv('/content/imdb_opt_prompt_epoch_logs_lr0

# OPT125m inference performance metric summary across bf/lora/prompt tuning
opt_bf_inf_lr5_bs8 = pd.read_csv('/content/imdb_opt_bitfit_inference_metrics_summ
opt_bf_inf_lr5_bs16 = pd.read_csv('/content/imdb_opt_bitfit_inference_metrics_sum
opt_bf_inf_lr1_bs8 = pd.read_csv('/content/imdb_opt_bitfit_inference_metrics_summ
opt_bf_inf_lr1_bs16 = pd.read_csv('/content/imdb_opt_bitfit_inference_metrics_sum
opt_lora_inf_lr5_bs8 = pd.read_csv('/content/imdb_opt_lora_inference_metrics_summ
opt_lora_inf_lr5_bs16 = pd.read_csv('/content/imdb_opt_lora_inference_metrics_sum
opt_lora_inf_lr1_bs8 = pd.read_csv('/content/imdb_opt_lora_inference_metrics_summ
opt_lora_inf_lr1_bs16 = pd.read_csv('/content/imdb_opt_lora_inference_metrics_sum
opt_prompt_inf_lr5_bs8 = pd.read_csv('/content/imdb_opt_prompt_inference_metrics_
opt_prompt_inf_lr5_bs16 = pd.read_csv('/content/imdb_opt_prompt_inference_metrics
opt_prompt_inf_lr1_bs8 = pd.read_csv('/content/imdb_opt_prompt_inference_metrics_
opt_prompt_inf_lr1_bs16 = pd.read_csv('/content/imdb_opt_prompt_inference_metrics

# OPT125m inference predictions across bf/lora/prompt tuning
opt_bf_preds_lr5_bs8 = pd.read_csv('/content/imdb_opt_bitfit_inference_prediction
opt_bf_preds_lr5_bs16 = pd.read_csv('/content/imdb_opt_bitfit_inference_predictio
opt_bf_preds_lr1_bs8 = pd.read_csv('/content/imdb_opt_bitfit_inference_prediction
opt_bf_preds_lr1_bs16 = pd.read_csv('/content/imdb_opt_bitfit_inference_predictio
opt_lora_preds_lr5_bs8 = pd.read_csv('/content/imdb_opt_lora_inference_prediction
opt_lora_preds_lr5_bs16 = pd.read_csv('/content/imdb_opt_lora_inference_predictio
opt_lora_preds_lr1_bs8 = pd.read_csv('/content/imdb_opt_lora_inference_prediction
opt_lora_preds_lr1_bs16 = pd.read_csv('/content/imdb_opt_lora_inference_predictio
opt_prompt_preds_lr5_bs8 = pd.read_csv('/content/imdb_opt_prompt_inference_predic
opt_prompt_preds_lr5_bs16 = pd.read_csv('/content/imdb_opt_prompt_inference_predi
opt_prompt_preds_lr1_bs8 = pd.read_csv('/content/imdb_opt_prompt_inference_predic
opt_prompt_preds_lr1_bs16 = pd.read_csv('/content/imdb_opt_prompt_inference_predi
```

```
# OPT125m PEFT method intra-comparison based on hyperparameter settings, per bf/l
opt_bf_final_comparison = pd.read_csv('/content/imdb_opt_bf_final_comparison_bitf
opt_lora_final_comparison = pd.read_csv('/content/imdb_opt_lora_final_comparison_
opt_prompt_final_comparison = pd.read_csv('/content/imdb_opt_prompt_final_compari
```

## BitFit Learning Curves

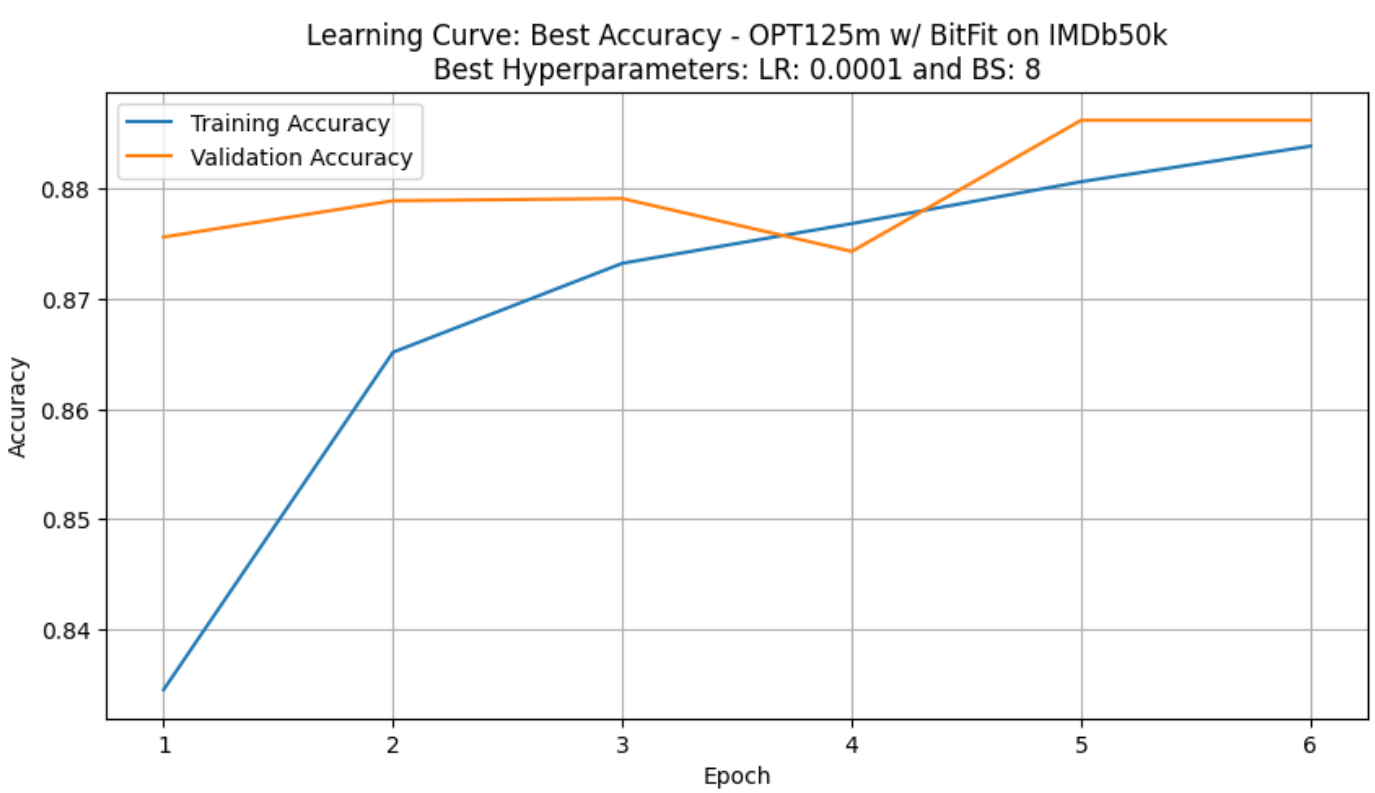
```
# All BitFit Train/Val Acc Learning Curve
plt.figure(figsize=(10,5))
sns.lineplot(data=opt_bf_epochs_lr5_bs8, x="epoch", y="train_accuracy", label="TA
sns.lineplot(data=opt_bf_epochs_lr5_bs8, x="epoch", y="val_accuracy", label="VA_l
sns.lineplot(data=opt_bf_epochs_lr5_bs16, x="epoch", y="train_accuracy", label="T
sns.lineplot(data=opt_bf_epochs_lr5_bs16, x="epoch", y="val_accuracy", label="VA_
sns.lineplot(data=opt_bf_epochs_lr1_bs8, x="epoch", y="train_accuracy", label="TA
sns.lineplot(data=opt_bf_epochs_lr1_bs8, x="epoch", y="val_accuracy", label="VA_l
sns.lineplot(data=opt_bf_epochs_lr1_bs16, x="epoch", y="train_accuracy", label="T
sns.lineplot(data=opt_bf_epochs_lr1_bs16, x="epoch", y="val_accuracy", label="VA_
plt.title("Learning Curve: Accuracy – OPT125m w/ BitFit on IMDb50k")
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=opt_bf_epochs_lr5_bs8, x="epoch", y="train_loss", label="TL_lr5
sns.lineplot(data=opt_bf_epochs_lr5_bs8, x="epoch", y="val_loss", label="VL_lr5_b
sns.lineplot(data=opt_bf_epochs_lr5_bs16, x="epoch", y="train_loss", label="TL_lr
sns.lineplot(data=opt_bf_epochs_lr5_bs16, x="epoch", y="val_loss", label="VL_lr5_
sns.lineplot(data=opt_bf_epochs_lr1_bs8, x="epoch", y="train_loss", label="TL_lr1
sns.lineplot(data=opt_bf_epochs_lr1_bs8, x="epoch", y="val_loss", label="VL_lr1_b
sns.lineplot(data=opt_bf_epochs_lr1_bs16, x="epoch", y="train_loss", label="TL_lr
sns.lineplot(data=opt_bf_epochs_lr1_bs16, x="epoch", y="val_loss", label="VL_lr1_
plt.title("Learning Curve: Loss – OPT125m w/ BitFit on IMDb50k")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

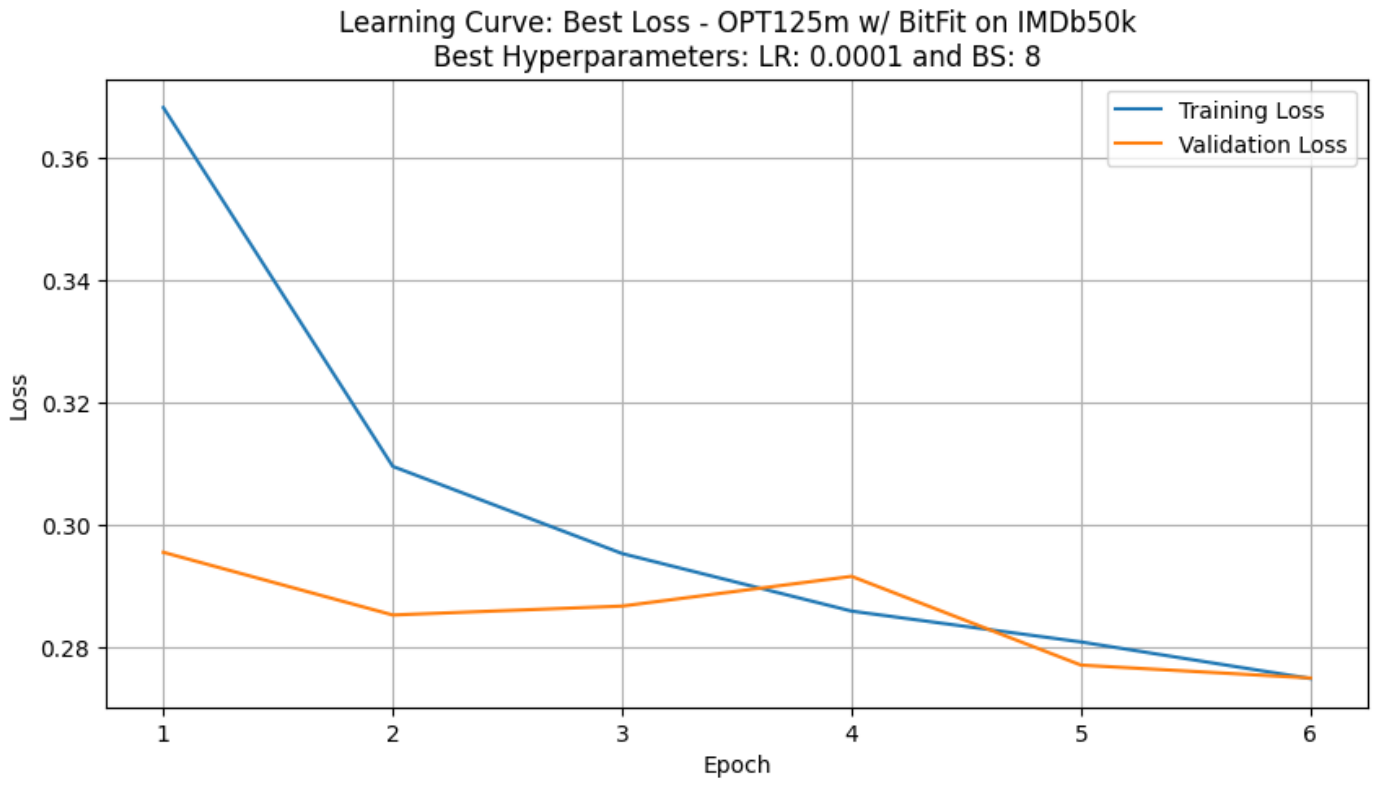

Learning Curve: Accuracy - OPT125m w/ BitFit on IMDb50k
Accuracy
Epoch
TA_lr5_bs8
VA_lr5_bs8
TA_lr5_bs16
VA_lr5_bs16
TA_lr1_bs8
VA_lr1_bs8
TA_lr1_bs16
VA_lr1_bs16


Learning Curve: Loss - OPT125m w/ BitFit on IMDb50k
Loss
Epoch
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

opt_bf_epochs_map = {
    (5, 8): opt_bf_epochs_lr5_bs8,
    (5, 16): opt_bf_epochs_lr5_bs16,
    (1, 8): opt_bf_epochs_lr1_bs8,
    (1, 16): opt_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 = opt_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 - OPT125m w/ BitFit on IMDb50k\nBest Hy
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 - OPT125m w/ BitFit on IMDb50k\nBest Hyperp
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
```

```
plt.show()
```

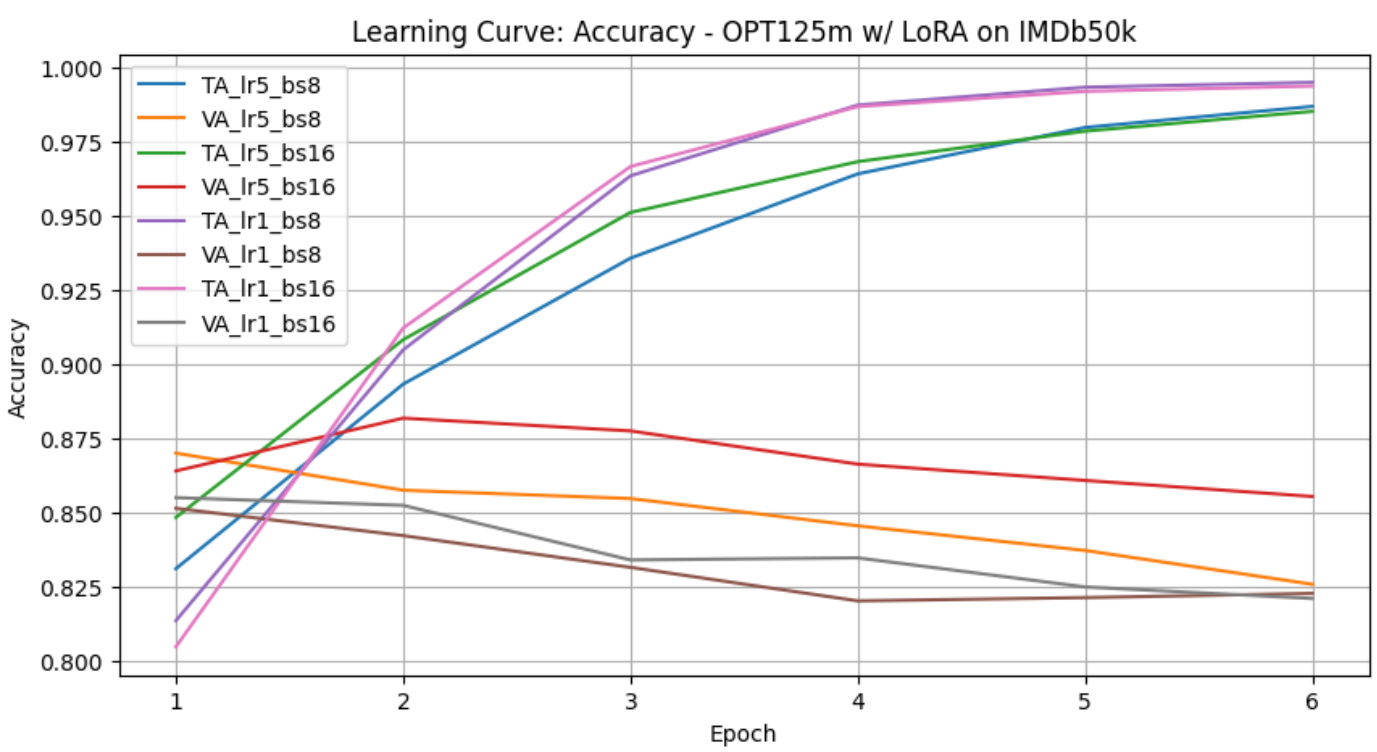

Learning Curve: Best Accuracy - OPT125m w/ BitFit on IMDb50k
Best Hyperparameters: LR: 0.0001 and BS: 8
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.84
0.85
0.86
0.87
0.88
1
2
3
4
5
6

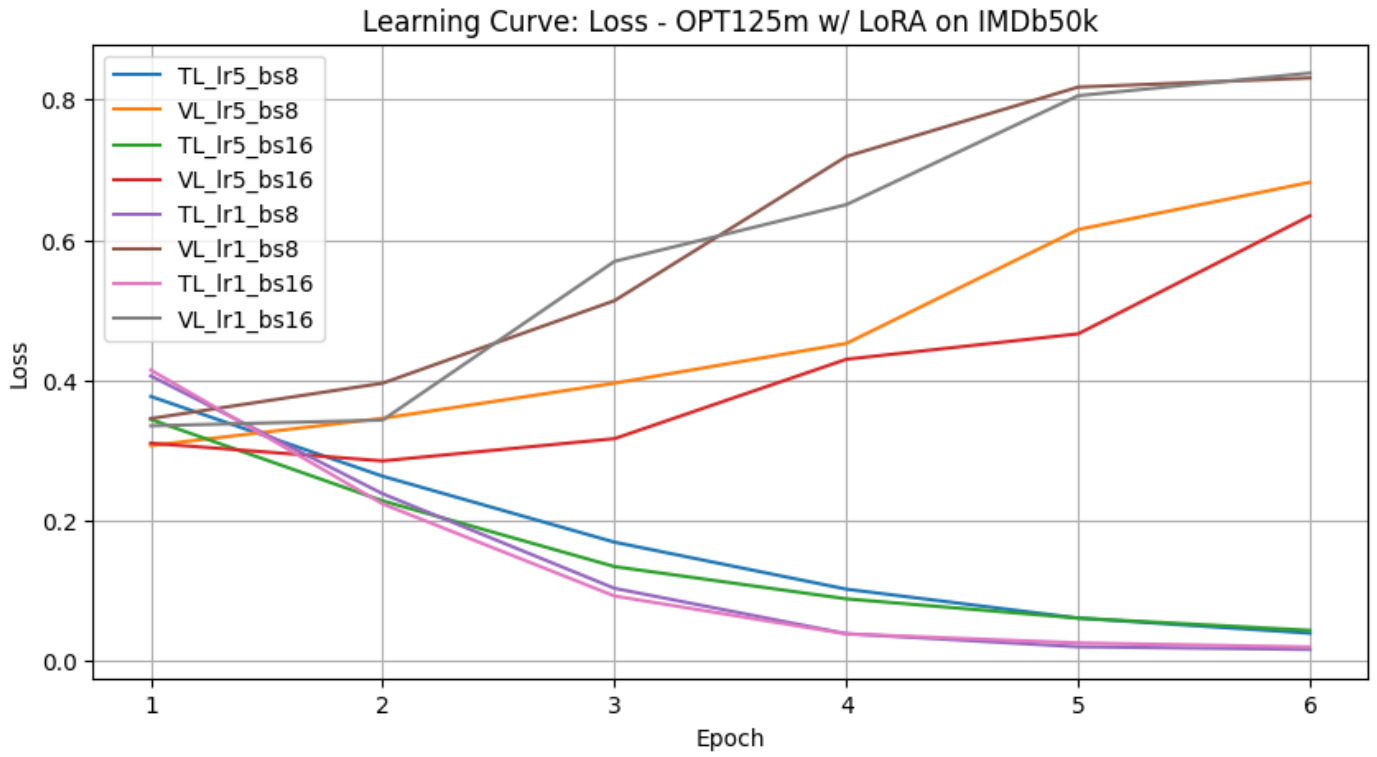

Learning Curve: Best Loss - OPT125m w/ BitFit on IMDb50k
Best Hyperparameters: LR: 0.0001 and BS: 8
Training Loss
Validation Loss
Loss
Epoch
0.28
0.30
0.32
0.34
0.36
1
2
3
4
5
6

## LoRA Learning Curves

```
# All LoRA Train/Val Acc Learning Curve
plt.figure(figsize=(10,5))
sns.lineplot(data=opt_lora_epochs_lr5_bs8, x="epoch", y="train_accuracy", label="
sns.lineplot(data=opt_lora_epochs_lr5_bs8, x="epoch", y="val_accuracy", label="VA
sns.lineplot(data=opt_lora_epochs_lr5_bs16, x="epoch", y="train_accuracy", label=
sns.lineplot(data=opt_lora_epochs_lr5_bs16, x="epoch", y="val_accuracy", label="V
sns.lineplot(data=opt_lora_epochs_lr1_bs8, x="epoch", y="train_accuracy", label="
sns.lineplot(data=opt_lora_epochs_lr1_bs8, x="epoch", y="val_accuracy", label="VA
sns.lineplot(data=opt_lora_epochs_lr1_bs16, x="epoch", y="train_accuracy", label=
sns.lineplot(data=opt_lora_epochs_lr1_bs16, x="epoch", y="val_accuracy", label="V
plt.title("Learning Curve: Accuracy – OPT125m w/ LoRA on IMDb50k")
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=opt_lora_epochs_lr5_bs8, x="epoch", y="train_loss", label="TL_l
sns.lineplot(data=opt_lora_epochs_lr5_bs8, x="epoch", y="val_loss", label="VL_lr5
sns.lineplot(data=opt_lora_epochs_lr5_bs16, x="epoch", y="train_loss", label="TL_
sns.lineplot(data=opt_lora_epochs_lr5_bs16, x="epoch", y="val_loss", label="VL_lr
sns.lineplot(data=opt_lora_epochs_lr1_bs8, x="epoch", y="train_loss", label="TL_l
sns.lineplot(data=opt_lora_epochs_lr1_bs8, x="epoch", y="val_loss", label="VL_lr1
sns.lineplot(data=opt_lora_epochs_lr1_bs16, x="epoch", y="train_loss", label="TL_
sns.lineplot(data=opt_lora_epochs_lr1_bs16, x="epoch", y="val_loss", label="VL_lr
plt.title("Learning Curve: Loss – OPT125m w/ LoRA on IMDb50k")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
```

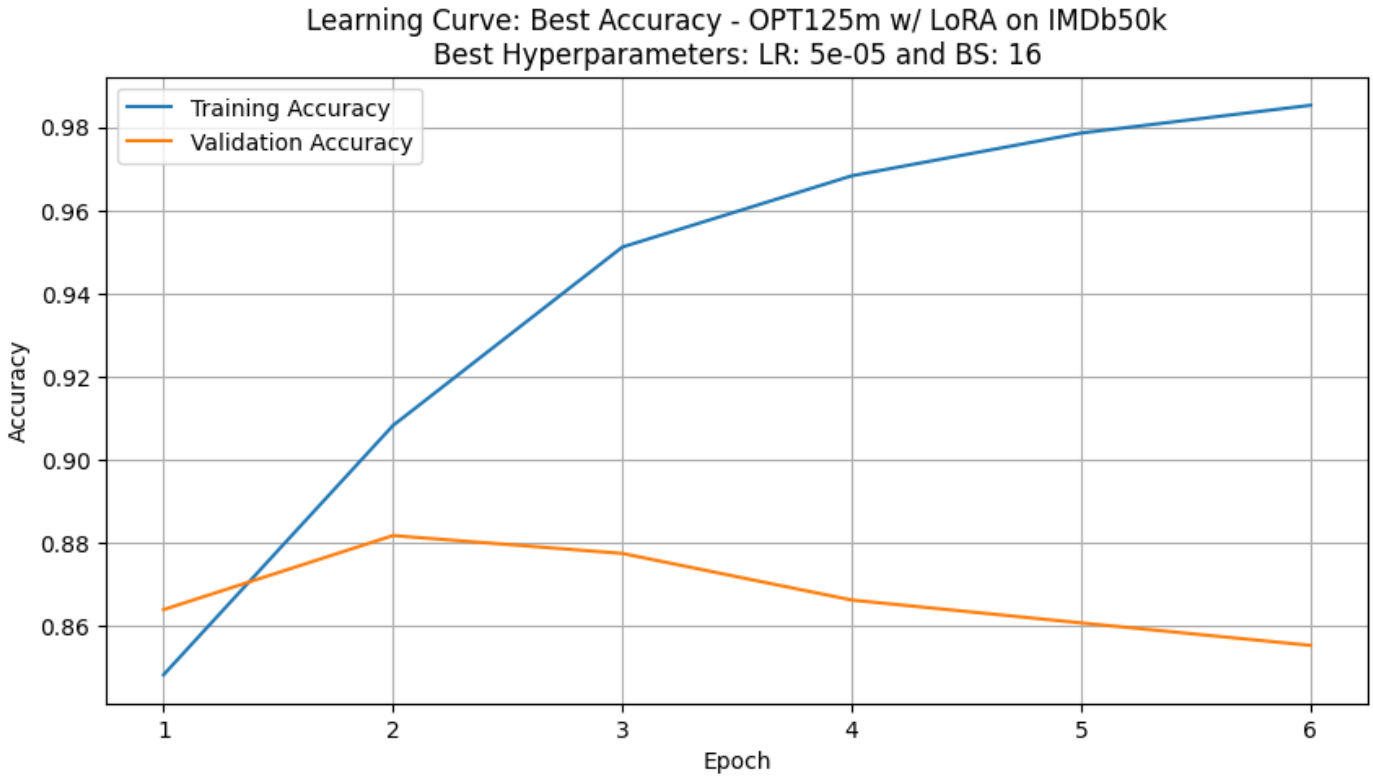
```
plt.show()
```

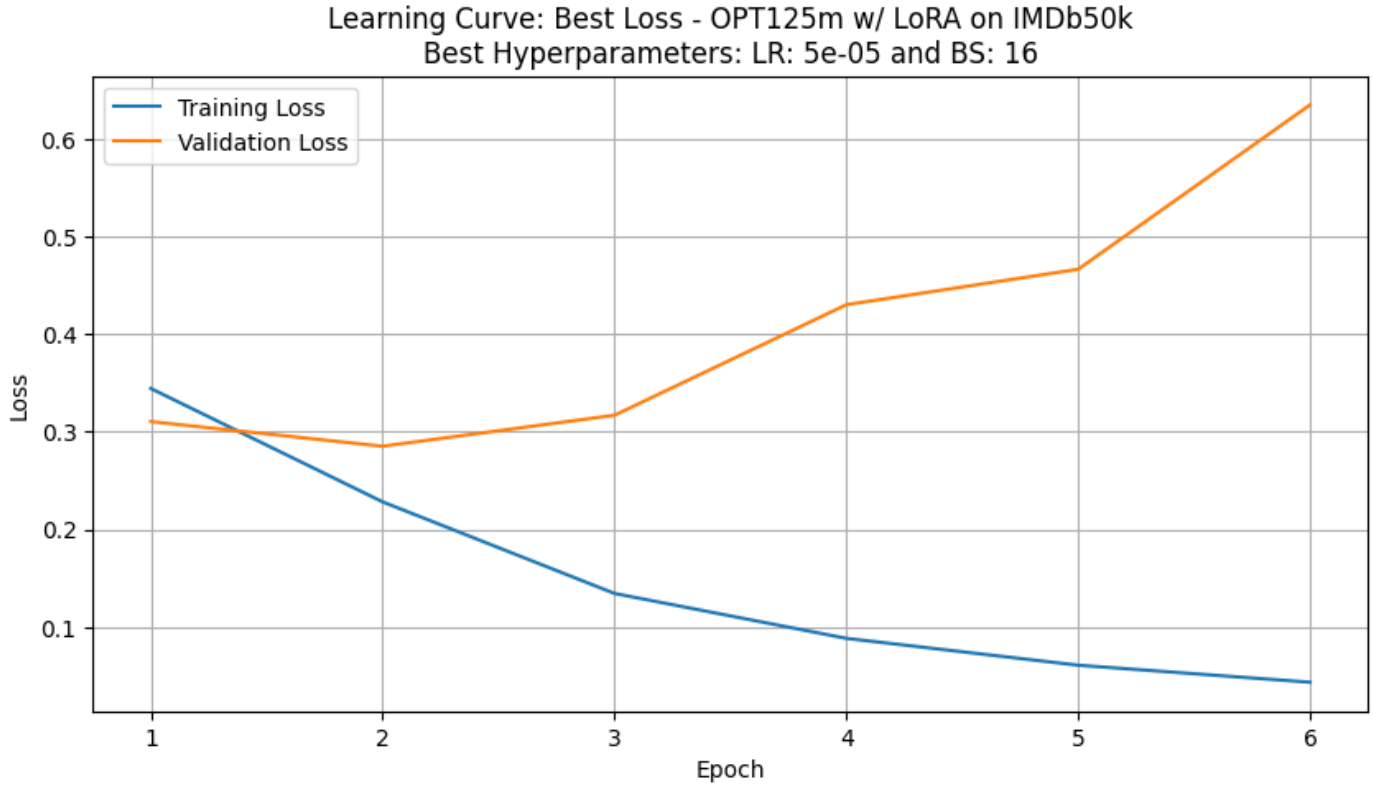

Learning Curve: Accuracy - OPT125m w/ LoRA on IMDb50k
TA_lr5_bs8
VA_lr5_bs8
TA_lr5_bs16
VA_lr5_bs16
TA_lr1_bs8
VA_lr1_bs8
TA_lr1_bs16
VA_lr1_bs16
Accuracy
Epoch


Learning Curve: Loss - OPT125m w/ LoRA on IMDb50k
TL_lr5_bs8
VL_lr5_bs8
TL_lr5_bs16
VL_lr5_bs16
TL_lr1_bs8
VL_lr1_bs8
TL_lr1_bs16
VL_lr1_bs16
Loss
Epoch

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

opt_lora_epochs_map = {
    (5, 8): opt_lora_epochs_lr5_bs8,
    (5, 16): opt_lora_epochs_lr5_bs16,
    (1, 8): opt_lora_epochs_lr1_bs8,
    (1, 16): opt_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 = opt_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 - OPT125m w/ LoRA on IMDb50k\nBest Hype
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 - OPT125m w/ LoRA on IMDb50k\nBest Hyperpar
plt.xlabel("Epoch")
plt.ylabel("Loss")
```

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

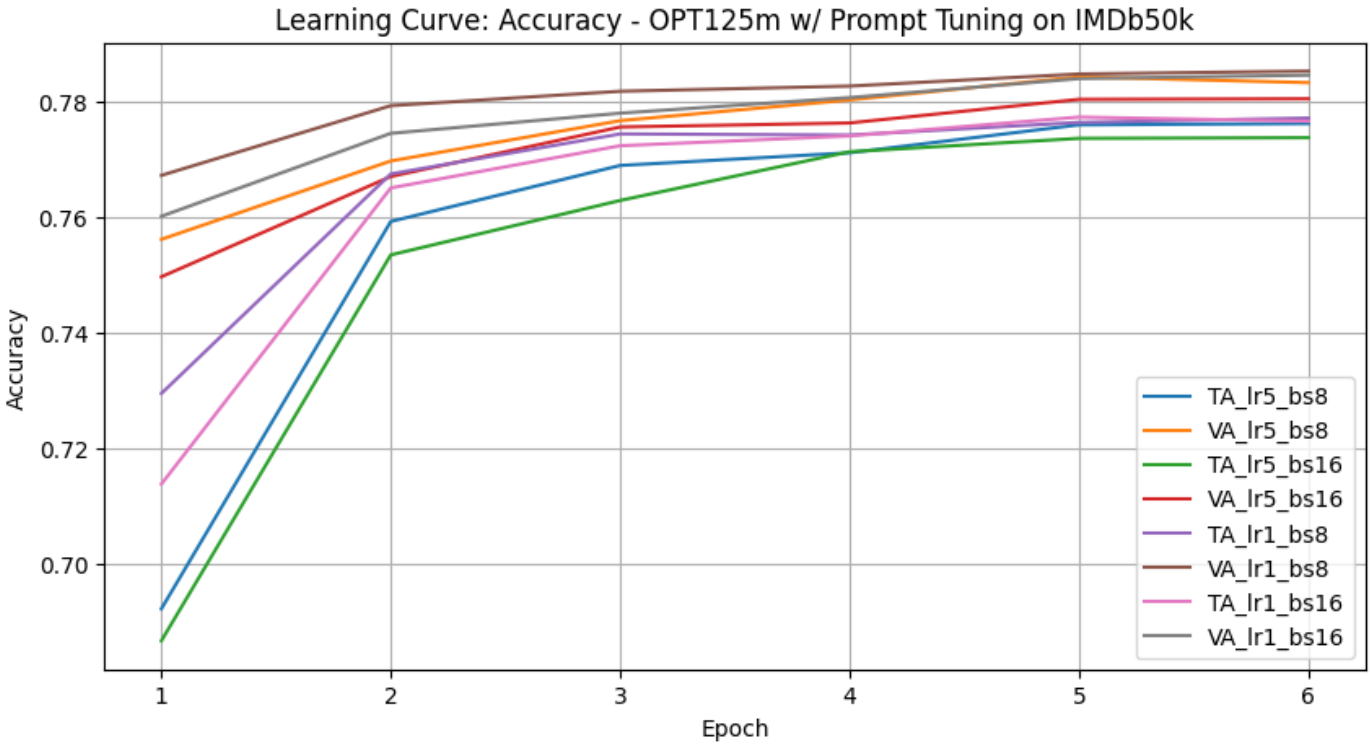


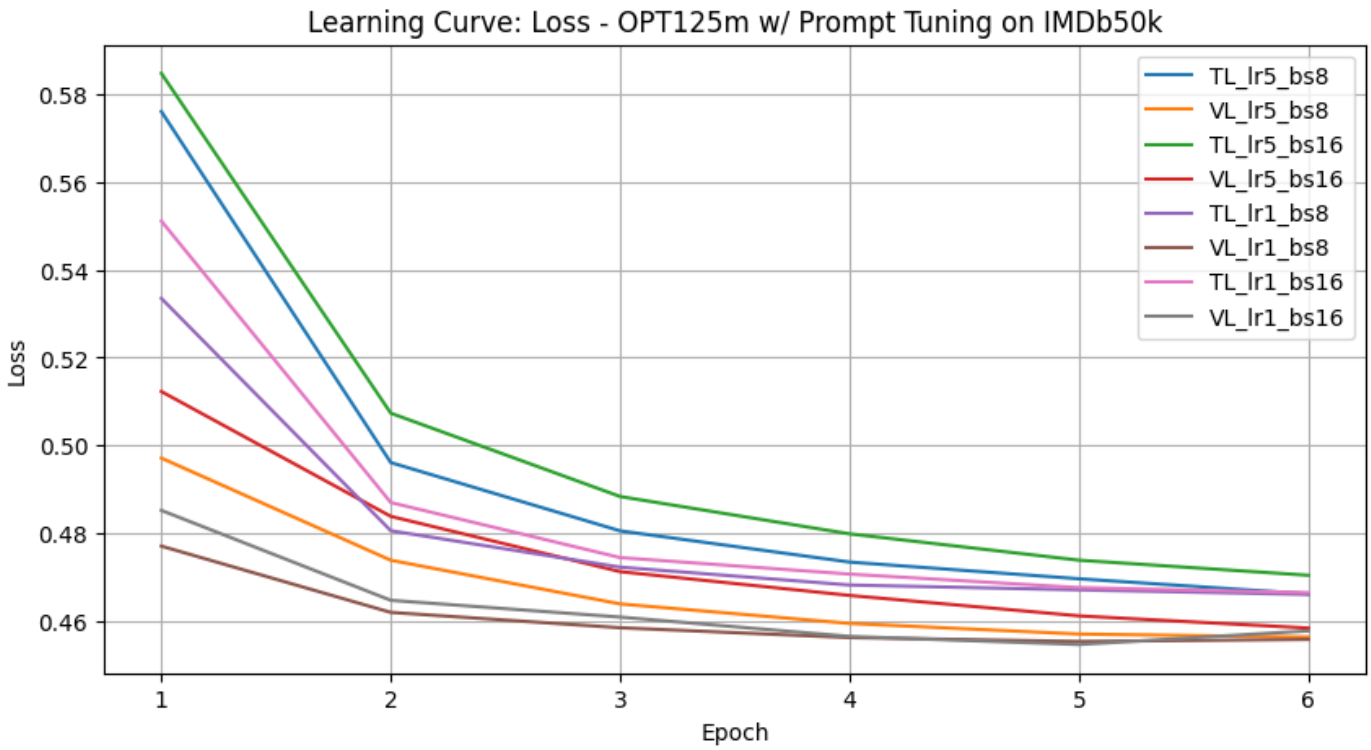



## Prompt Tuning Learning Curves

```
# All Prompt Tuning Train/Val Acc Learning Curve
plt.figure(figsize=(10,5))
sns.lineplot(data=opt_prompt_epochs_lr5_bs8, x="epoch", y="train_accuracy", label=
sns.lineplot(data=opt_prompt_epochs_lr5_bs8, x="epoch", y="val_accuracy", label="
sns.lineplot(data=opt_prompt_epochs_lr5_bs16, x="epoch", y="train_accuracy", labe
sns.lineplot(data=opt_prompt_epochs_lr5_bs16, x="epoch", y="val_accuracy", label=
sns.lineplot(data=opt_prompt_epochs_lr1_bs8, x="epoch", y="train_accuracy", label=
sns.lineplot(data=opt_prompt_epochs_lr1_bs8, x="epoch", y="val_accuracy", label="
sns.lineplot(data=opt_prompt_epochs_lr1_bs16, x="epoch", y="train_accuracy", labe
sns.lineplot(data=opt_prompt_epochs_lr1_bs16, x="epoch", y="val_accuracy", label=
plt.title("Learning Curve: Accuracy - OPT125m w/ Prompt Tuning on IMDb50k")
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=opt_prompt_epochs_lr5_bs8, x="epoch", y="train_loss", label="TL
sns.lineplot(data=opt_prompt_epochs_lr5_bs8, x="epoch", y="val_loss", label="VL_l
sns.lineplot(data=opt_prompt_epochs_lr5_bs16, x="epoch", y="train_loss", label="T
sns.lineplot(data=opt_prompt_epochs_lr5_bs16, x="epoch", y="val_loss", label="VL_
sns.lineplot(data=opt_prompt_epochs_lr1_bs8, x="epoch", y="train_loss", label="TL
sns.lineplot(data=opt_prompt_epochs_lr1_bs8, x="epoch", y="val_loss", label="VL_l
sns.lineplot(data=opt_prompt_epochs_lr1_bs16, x="epoch", y="train_loss", label="T
sns.lineplot(data=opt_prompt_epochs_lr1_bs16, x="epoch", y="val_loss", label="VL_
plt.title("Learning Curve: Loss - OPT125m w/ Prompt Tuning on IMDb50k")
plt.xlabel("Epoch")
plt.ylabel("Loss")
```

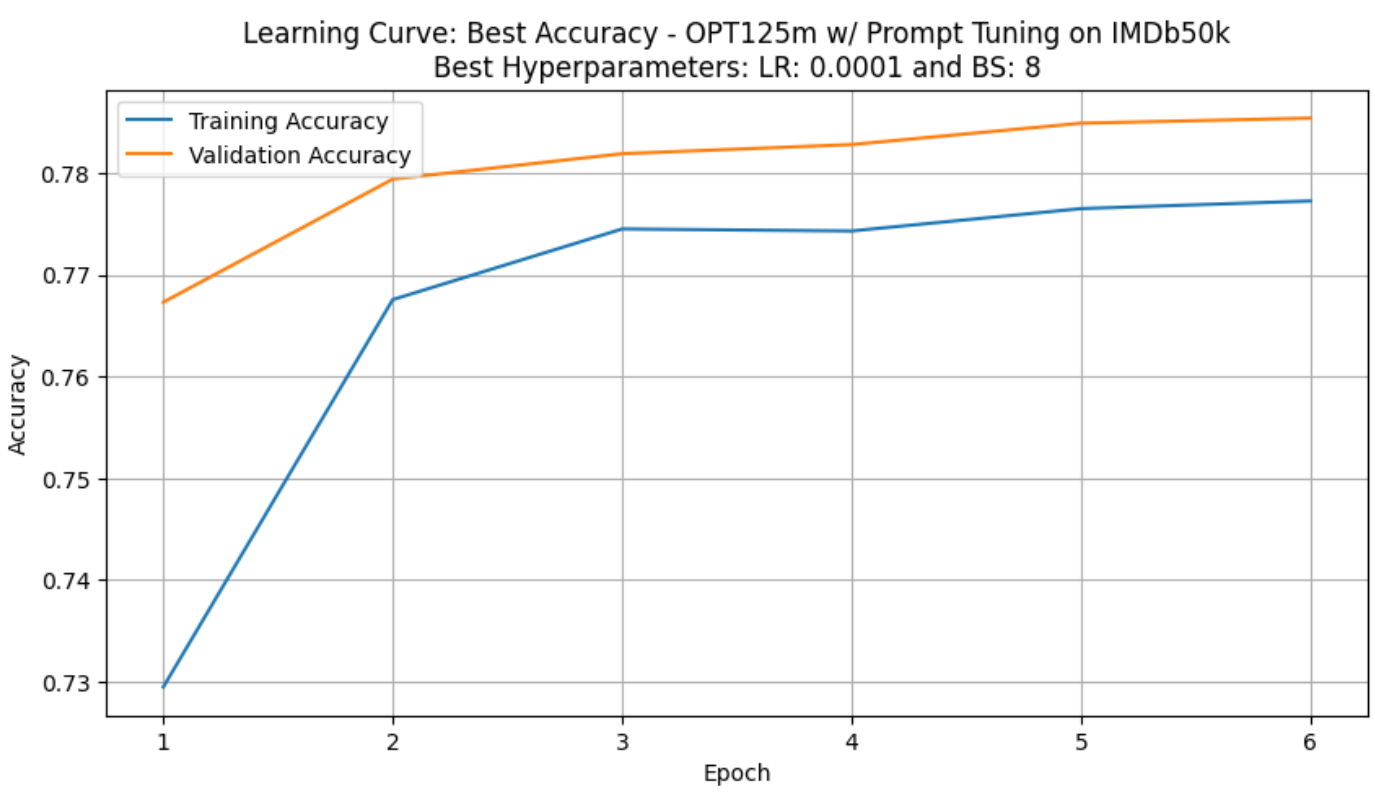
```
plt.legend()
plt.grid(True)
plt.show()
```

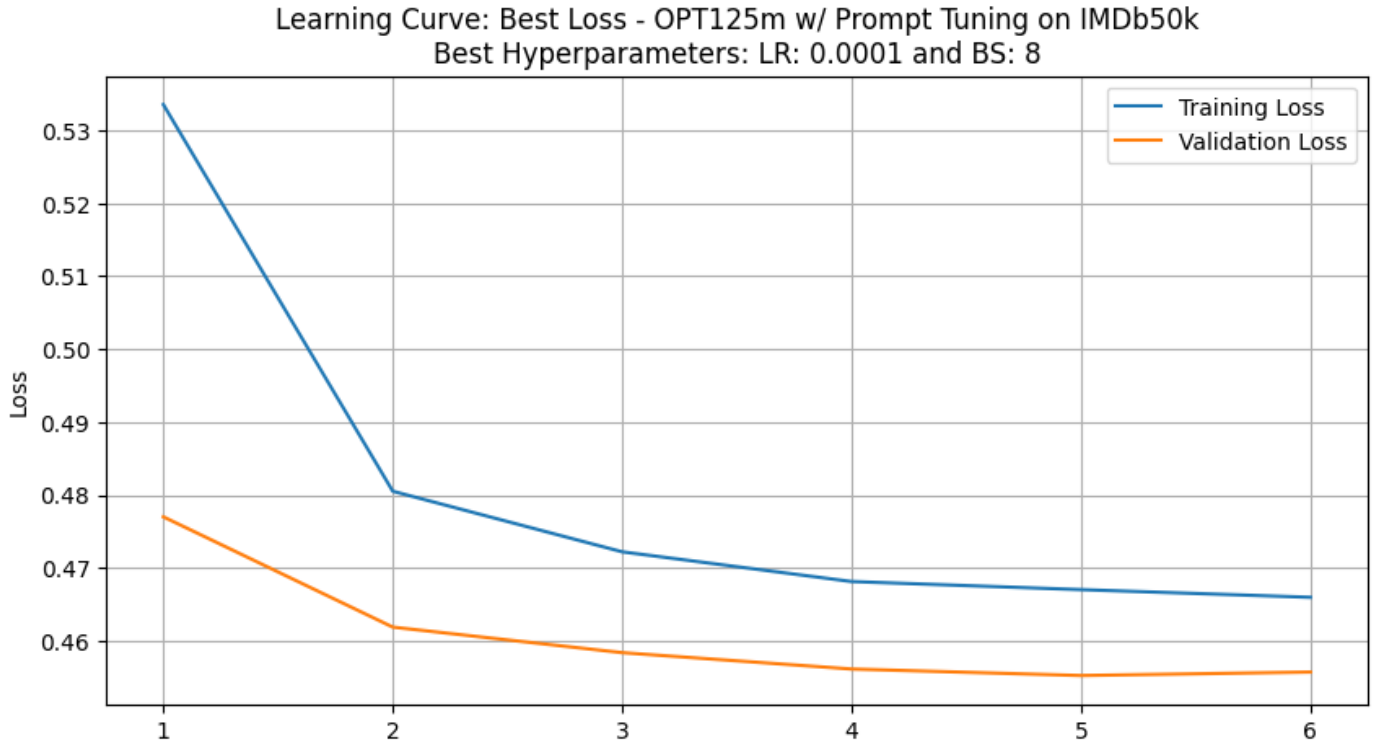


```
# Best Prompt Tuning Train/Val Acc Learning Curve

opt_prompt_epochs_map = {
    (5, 8): opt_prompt_epochs_lr5_bs8,
    (5, 16): opt_prompt_epochs_lr5_bs16,
    (1, 8): opt_prompt_epochs_lr1_bs8,
    (1, 16): opt_prompt_epochs_lr1_bs16
}

prompt_lr_mapping = {
    5e-5: "5e-5",
    1e-4: "41e-"
}

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

prompt_best_bs_tag = prompt_best_bs

prompt_epochs = opt_prompt_epochs_map[(prompt_best_lr_tag_for_map, prompt_best_bs_t

# Best Prompt Tuning Training and Validation Accuracy
plt.figure(figsize=(10,5))
sns.lineplot(data=prompt_epochs, x="epoch", y="train_accuracy", label="Training Acc
sns.lineplot(data=prompt_epochs, x="epoch", y="val_accuracy", label="Validation Acc
plt.title(f"Learning Curve: Best Accuracy - OPT125m w/ Prompt Tuning on IMDb50k\nBe
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 - OPT125m w/ Prompt Tuning on IMDb50k\nBest H
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```



Epoch

PEFT Method Comparison:

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

opt_bf_results = pd.read_csv('/content/imdb_opt_bitfit_results.csv')
opt_lora_results = pd.read_csv('/content/imdb_opt_lora_results.csv')
opt_prompt_results = pd.read_csv('/content/imdb_opt_prompt_results.csv')

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

```
        opt_prompt_results["inference_time"].sum()
    ],
    "Max GPU Memory (GB)": [
        opt_bf_results["max_memory"].max(),
        opt_lora_results["max_memory"].max(),
        opt_prompt_results["max_memory"].max()
    ]
})


print("\nFinal Validation Performance PEFT Comparison - OPT125m on IMDb50k:")
display(comparison)
```

```
Final Validation Performance PEFT Comparison - OPT125m on IMDb50k:
```

| | Method | Best Validation F1 | Best Validation Accuracy | Runtime (sec) | Inference Time (sec) | Max GPU Memory (GB) |
|---|---|---|---|---|---|---|
| 0 | BitFit | 0.886156 | 0.8862 | 3598.705708 | 48.718416 | 3.828124 |
| 1 | LoRA | 0.855378 | 0.8554 | 4785.197376 | 14310.164802 | 3.828124 |

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

```
# Load overall results where inference_time is stored
opt_prompt_results = pd.read_csv('/content/imdb_opt_prompt_results.csv')
opt_bf_results = pd.read_csv('/content/imdb_opt_bitfit_results.csv')
opt_lora_results = pd.read_csv('/content/imdb_opt_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/imdb_opt_bitfit_inference_metrics_summary_lr{bf_bes
lora_inf = pd.read_csv(f'/content/imdb_opt_lora_inference_metrics_summary_lr{lora_b
prompt_inf = pd.read_csv(f'/content/imdb_opt_prompt_inference_metrics_summary_lr{pr
```

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

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

prompt_inference_time = opt_prompt_results[
    (opt_prompt_results["learning_rate"] == prompt_best_lr) &
    (opt_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 - OPT125m on IMDb50k:
display(final_test_results)
```

Final Test Set Inference Performance PEFT Comparison - OPT125m on IMDb50k:

| | Method | Test Accuracy | F1 Macro | F1 Weighted | Inference Time (sec) |
|---|---|---|---|---|---|
| 0 | BitFit | 0.8862 | 0.886139 | 0.886156 | 15.483433 |
| 1 | LoRA | 0.8554 | 0.855363 | 0.855378 | 5201.837212 |
| 2 | Prompt | 0.7854 | 0.785379 | 0.785393 | 584.572935 |

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