```
# Install required libraries
!pip install transformers[torch] datasets evaluate kaggle scikit-learn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets) (3.6.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.12/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages (from kaggle) (6.3.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2025.11.12)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.4.4)
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from kaggle) (3.11)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from kaggle) (5.29.5)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.9.0.post
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.12/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.12/dist-packages (from kaggle) (1.3)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.12/dist-packages (from kaggle) (2.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from kaggle) (0.5.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate>=0.26.0->transformers[to
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]<=2
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->transformers[tor
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->transformers[t
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->transformers[torch]) (3
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->t
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->t
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->tr
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2-
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2-
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->trans
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->tr
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->tran
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2-
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->t
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch>=2.2->transformers[tor
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->datasets) (2025.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch>=2
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=2.2->transf
Downloading evaluate-0.4.6-py3-none-any.whl (84 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 4.5 MB/s eta 0:00:00
Installing collected packages: evaluate
Successfully installed evaluate-0.4.6
```

```
import os
from google.colab import files

# Upload the kaggle.json API key
print("Please upload your kaggle.json file")
files.upload()
```

```
Please upload your kaggle.json file
Sélect. fichiers kaggle.json
kaggle.json(application/json) - 65 bytes, last modified: 16/11/2025 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"gamos3476","key":"e15addf5c4419f151c0322270ae3d7e7"}'}
```

```
# Set up Kaggle directory and permissions
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download and unzip the dataset
!kaggle datasets download -d ilhamfp31/yelp-review-dataset
!unzip -o yelp-review-dataset.zip
```

```
Dataset URL: https://www.kaggle.com/datasets/ilhamfp31/yelp-review-dataset
License(s): unknown
Downloading yelp-review-dataset.zip to /content
 88% 142M/162M [00:00<00:00, 1.48GB/s]
100% 162M/162M [00:00<00:00, 1.26GB/s]
Archive:  yelp-review-dataset.zip
  inflating: yelp_review_polarity_csv/readme.txt
  inflating: yelp_review_polarity_csv/test.csv
  inflating: yelp_review_polarity_csv/train.csv
```

```
import pandas as pd

# 1. Load the pre-split data
train_df_raw = pd.read_csv('yelp_review_polarity_csv/train.csv', header=None)
test_df_raw = pd.read_csv('yelp_review_polarity_csv/test.csv', header=None)

# 2. Preprocess and format the data
# Column 0 is the label (1 or 2), Column 1 is the text.
train_df_raw.columns = ['label', 'text']
test_df_raw.columns = ['label', 'text']

# Create our dataframes
# We convert labels: 1 -> 0 (Negative), 2 -> 1 (Positive)
train_df = pd.DataFrame({
    'text': train_df_raw['text'],
    'label': train_df_raw['label'].apply(lambda x: 0 if x == 1 else 1)
})

test_df = pd.DataFrame({
    'text': test_df_raw['text'],
    'label': test_df_raw['label'].apply(lambda x: 0 if x == 1 else 1)
})
```

```
# 3. Verify the data
print(f"Total training reviews: {len(train_df)}")
print(f"Total test reviews: {len(test_df)}")

# We will use a smaller sample to avoid Colab time limits.
TRAIN_SAMPLE_SIZE = 50000
TEST_SAMPLE_SIZE = 5000

# Create the sample using 'random_state=42' so the sample is the same every time
train_df = train_df.sample(n=TRAIN_SAMPLE_SIZE, random_state=42)
test_df = test_df.sample(n=TEST_SAMPLE_SIZE, random_state=42)

print(f"\n- USING A SMALLER SAMPLE")
print(f"New training reviews: {len(train_df)}")
print(f"New test reviews: {len(test_df)}")
```

```
Total training reviews: 560000
Total test reviews: 38000

- USING A SMALLER SAMPLE
New training reviews: 50000
New test reviews: 5000
```

```
import torch
import numpy as np
import evaluate # Hugging Face's evaluation library
from datasets import Dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)

# 1. Define Model and Tokenizer
MODEL_NAME = 'google/mobilebert-uncased'

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=2)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# 2. Convert Pandas to Hugging Face Datasets
train_dataset = Dataset.from_pandas(train_df.reset_index(drop=True))
test_dataset = Dataset.from_pandas(test_df.reset_index(drop=True))
```

```
config.json: 100%  847/847 [00:00<00:00, 98.6kB/s]
vocab.txt: 232k/? [00:00<00:00, 16.5MB/s]
tokenizer.json: 466k/? [00:00<00:00, 31.6MB/s]
pytorch_model.bin: 100%  147M/147M [00:01<00:00, 152MB/s]
Some weights of MobileBertForSequenceClassification were not initialized from the model checkpoint at google/mobilebert-unca
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# 3. Tokenize the Data
def tokenize_function(examples):
    # 'truncation=True' cuts reviews longer than the model's max length
    return tokenizer(examples['text'], truncation=True, max_length=512)

tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_test = test_dataset.map(tokenize_function, batched=True)

# 4. Define Evaluation Metric
metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

```
Map: 100%  50000/50000 [00:26<00:00, 1844.44 examples/s]
model.safetensors: 100%  147M/147M [00:03<00:00, 54.9MB/s]
Map: 100%  5000/5000 [00:02<00:00, 2155.40 examples/s]
Downloading builder script: 4.20k/? [00:00<00:00, 392kB/s]
```

```
# 5. Define Training Arguments
training_args = TrainingArguments(
    output_dir="yelp_sentiment_model",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=2,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    push_to_hub=False,
    report_to="none",
)

# 6. Create and Run the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    compute_metrics=compute_metrics,
    data_collator=data_collator,
)

print(" Starting Fine-Tuning --")
trainer.train()
print(" Fine-Tuning Complete-")
```

```
 Starting Fine-Tuning --
```

[6250/6250 1:01:37, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.133300	0.135368	0.959000
2	0.095600	0.146506	0.960800

```
 Fine-Tuning Complete-
```

```
from transformers import pipeline
from sklearn.metrics import accuracy_score
from tqdm.auto import tqdm

# Load a larger, general-purpose model for zero-shot classification
classifier = pipeline("zero-shot-classification",
                      model="facebook/bart-large-mnli",
                      device=0) # Use 0 for GPU
```

```
# Define labels
candidate_labels = ["positive", "negative"]

# Running on all 5000 test reviews takes a long time, so we ensure we use the sampled set
test_sample = test_df.copy() # reusing the 5000 sample from earlier

predictions_large_model = []
true_labels_large_model = []

print(f" Running Zero-Shot Evaluation on {len(test_sample)} reviews -")
```

```
Device set to use cuda:0
 Running Zero-Shot Evaluation on 5000 reviews -
```

```
# Loop through the sample
for index, row in tqdm(test_sample.iterrows(), total=test_sample.shape[0]):
    text = row['text']
    true_label_int = row['label']

    # Get the model's prediction
    result = classifier(
        text,
        candidate_labels,
        hypothesis_template="The sentiment of this review is {}."
    )

    # The result['labels'][0] is the label with the highest score
    predicted_label_str = result['labels'][0]

    # Convert string "positive"/"negative" back to 0/1
    predicted_label_int = 1 if predicted_label_str == "positive" else 0

    predictions_large_model.append(predicted_label_int)
    true_labels_large_model.append(true_label_int)
```

```
100%                                                5000/5000 [08:59<00:00,  8.27it/s]
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
```

```
large_model_accuracy = accuracy_score(true_labels_large_model, predictions_large_model)

print(f"\n Evaluating Large Zero-Shot Model (BART-Large) --")
print(f"Test Set (Sample) Accuracy: {large_model_accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(true_labels_large_model, predictions_large_model, target_names=['Negative', 'Positive']))

print(" Final Experiment Results --\n")
print(f"Small, Fine-Tuned Model (MobileBERT):")
print(f" Accuracy on {len(test_df)} reviews: {small_model_accuracy:.4f}")
print("\n")
print(f"Large, Zero-Shot Model (BART-Large):")
print(f" Accuracy on {len(test_sample)} reviews: {large_model_accuracy:.4f}")
print("\n")

print("Conclusion")
if small_model_accuracy > large_model_accuracy:
    print("Hypothesis confirmed: The small model fine-tuned on Yelp data")
    print("outperformed the large, general-purpose model.")
else:
    print("Hypothesis not confirmed: The large, general-purpose model")
    print("was more accurate than the small, fine-tuned model.")
```

```
 Evaluating Large Zero-Shot Model (BART-Large) --
Test Set (Sample) Accuracy: 0.9568

Classification Report:
 Final Experiment Results --

Small, Fine-Tuned Model (MobileBERT):
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-2787566949.py in <cell line: 0>()
      8 print(" Final Experiment Results --\n")
      9 print(f"Small, Fine-Tuned Model (MobileBERT):")
---> 10 print(f" Accuracy on {len(test_df)} reviews: {small_model_accuracy:.4f}")
     11 print("\n")
     12 print(f"Large, Zero-Shot Model (BART-Large):")

NameError: name 'small_model_accuracy' is not defined
```

```
import torch
import numpy as np
import pandas as pd
from peft import LoraConfig, get_peft_model, TaskType
from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments, DataCollatorWithPad
from sklearn.metrics import classification_report
from datasets import Dataset
import evaluate, gc

# Clear memory
del model, trainer, classifier
torch.cuda.empty_cache()
gc.collect()

# Reuse the data from earlier steps
# Ensure labels are int and drop NaNs
train_df["label"] = train_df["label"].astype(int)
test_df["label"] = test_df["label"].astype(int)

# Setup Model + Tokenizer
MODEL_NAME = "google/mobilebert-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# Convert to HF datasets
train_dataset = Dataset.from_pandas(train_df.reset_index(drop=True))
test_dataset = Dataset.from_pandas(test_df.reset_index(drop=True))

# Tokenize (Reduced max_length for LoRA efficiency)
def tokenize_function(batch):
    return tokenizer(batch["text"], truncation=True, max_length=256)

tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_test = test_dataset.map(tokenize_function, batched=True)

print("Loading MobileBERT...")
base_model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, num_labels=2)

# Correct LoRA modules for MobileBERT
LORA_TARGET_MODULES = [
    "attention.self.query",
    "attention.self.key",
    "attention.self.value",
    "attention.output.dense"
]

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=LORA_TARGET_MODULES,
    lora_dropout=0.1,
    bias="none",
    task_type=TaskType.SEQ_CLS,
    inference_mode=False,
    modules_to_save=["classifier"] # ensures classifier head is trained
)

peft_model = get_peft_model(base_model, lora_config)
peft_model.print_trainable_parameters()
```

```
Map: 100%  50000/50000 [01:06<00:00, 862.10 examples/s]

Map: 100%  5000/5000 [00:05<00:00, 1094.87 examples/s]

Loading MobileBERT...
Some weights of MobileBertForSequenceClassification were not initialized from the model checkpoint at google/mobilebert-unca
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
trainable params: 541,698 || all params: 25,124,612 || trainable%: 2.1560
```

```
# Training Settings (Optimized for Colab T4)
metric = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return metric.compute(predictions=preds, references=labels)

training_args = TrainingArguments(
    output_dir="mobilebert_lora_yelp",
    learning_rate=3e-4,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=64,
```

```
    num_train_epochs=3,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    fp16=False, # IMPORTANT: MobileBERT becomes NaN with fp16!
    logging_steps=50,
    warmup_ratio=0.1,
    optim="adamw_torch",
    report_to="none",
    dataloader_pin_memory=True
)

trainer = Trainer(
    model=peft_model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    compute_metrics=compute_metrics,
    data_collator=data_collator,
)

print("Training started...")
trainer.train()
print("Training finished.")
```

```
Training started...
```


[4689/4689 39:53, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.177000	0.154786	0.948600
2	0.138500	0.145392	0.951200
3	0.110500	0.141881	0.953200

```
Training finished.
```

```
# Evaluation
results = trainer.evaluate()
print("\nEvaluation:", results)

preds = trainer.predict(tokenized_test).predictions
pred_labels = np.argmax(preds, axis=1)

print("\nClassification Report:")
#print(classification_report(test_df["label"], pred_labels, target_names=["Negative", "Positive"]))
```