


Step 1: Install Dependencies

This cell installs the required Python libraries for the project:

- `transformers`: For DistilBERT model and tokenizer.
- `datasets`: For loading the SST-2 dataset.
- `torch`: For PyTorch and GPU support.
- `scikit-learn`: For evaluation metrics.
- `matplotlib`: For plotting loss and metrics.
- `numpy`: For numerical operations.
- `tqdm`: For training progress bars.

Run this cell to install all dependencies.

```
1 !pip install transformers datasets torch scikit-learn matplotlib numpy
```

 Downloading dill-0.3.8-py3-none-any.whl (116 kB)
 Downloading fsspec-2024.12.0-py3-none-any.whl (183 kB)
 Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
 Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64_manylinux2014_x86_64.whl (194 kB)

Installing collected packages: xxhash, nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12, nvidia-cusparselt-cu12, nvidia-nccl-cu12, nvidia-nvtx-cu12, fsspec, multiprocess, datasets, transformers, torch, scikit-learn, matplotlib, numpy
 Attempting uninstall: nvidia-nvjitlink-cu12
 Found existing installation: nvidia-nvjitlink-cu12 12.5.82
 Uninstalling nvidia-nvjitlink-cu12-12.5.82:
 Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
 Attempting uninstall: nvidia-curand-cu12
 Found existing installation: nvidia-curand-cu12 10.3.6.82
 Uninstalling nvidia-curand-cu12-10.3.6.82:
 Successfully uninstalled nvidia-curand-cu12-10.3.6.82
 Attempting uninstall: nvidia-cufft-cu12
 Found existing installation: nvidia-cufft-cu12 11.2.3.61
 Uninstalling nvidia-cufft-cu12-11.2.3.61:
 Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
 Attempting uninstall: nvidia-cuda-runtime-cu12
 Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
 Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
 Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
 Attempting uninstall: nvidia-cuda-nvrtc-cu12
 Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
 Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
 Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
 Attempting uninstall: nvidia-cuda-cupti-cu12
 Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
 Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
 Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
 Attempting uninstall: nvidia-cublas-cu12
 Found existing installation: nvidia-cublas-cu12 12.5.3.2
 Uninstalling nvidia-cublas-cu12-12.5.3.2:
 Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
 Attempting uninstall: fsspec
 Found existing installation: fsspec 2025.3.2
 Uninstalling fsspec-2025.3.2:
 Successfully uninstalled fsspec-2025.3.2
 Attempting uninstall: nvidia-cusparselt-cu12
 Found existing installation: nvidia-cusparselt-cu12 12.5.1.3
 Uninstalling nvidia-cusparselt-cu12-12.5.1.3:
 Successfully uninstalled nvidia-cusparselt-cu12-12.5.1.3
 Attempting uninstall: nvidia-cudnn-cu12
 Found existing installation: nvidia-cudnn-cu12 9.3.0.75
 Uninstalling nvidia-cudnn-cu12-9.3.0.75:
 Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
 Attempting uninstall: nvidia-cusolver-cu12
 Found existing installation: nvidia-cusolver-cu12 11.6.3.83
 Uninstalling nvidia-cusolver-cu12-11.6.3.83:
 Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
 ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts:
 gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2024.12.0 which is incompatible.
 Successfully installed datasets-3.5.0 dill-0.3.8 fsspec-2024.12.0 multiprocess-0.70.16 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.5.82 nvidia-cuda-runtime-cu12-12.5.82 nvidia-cufft-cu12-11.2.3.61 nvidia-curand-cu12-10.3.6.82 nvidia-cusolver-cu12-11.6.3.83 nvidia-cusparselt-cu12-12.5.1.3 nvidia-cudnn-cu12-9.3.0.75 nvidia-cublas-cu12-12.5.3.2 nvidia-cuda-cupti-cu12-12.5.82 nvidia-cuda-nvrtc-cu12-12.5.82 nvidia-cuda-runtime-cu12-12.5.82 nvidia-nvjitlink-cu12-12.5.82 numpy-2.0.2 matplotlib-3.9.0 scikit-learn-1.6.1 torch-2.6.0 transformers-4.46.3

Step 2: Import Libraries and Set Up Environment

This cell imports necessary libraries, sets up logging, and configures the environment:

- Clears TPU-related environment variables to avoid conflicts.
- Configures logging to display info-level messages.
- Checks for GPU availability (T4 GPU required).
- Raises an error if no GPU is detected.

Run this cell to initialize the environment.

```
1 import numpy as np
2 import torch
3 from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments
4 from datasets import load_dataset
5 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
6 import matplotlib.pyplot as plt
7 import logging
8 import os
9
10 # Clear TPU-related environment variables
11 os.environ.pop('PJRT_DEVICE', None)
12
13 # Set up logging
14 logging.basicConfig(level=logging.INFO)
15 logger = logging.getLogger(__name__)
```

```
1 # Step 1: Set GPU device
2 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
3 if device.type != 'cuda':
4     raise RuntimeError("GPU not available. Ensure Colab runtime is set to T4 GPU.")
5 logger.info(f"Using device: {device}")
```


Step 3: Load Model and Dataset

This cell:

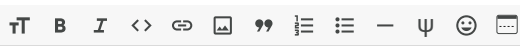
- Loads the DistilBERT tokenizer and model (distilbert-base-uncased) for binary classification (2 labels).
- Loads the SST-2 dataset and selects the full training (67,349 samples) and validation (872 samples) sets.
- Preprocesses the dataset by tokenizing sentences with a max length of 32.
- Formats the dataset for PyTorch (renames label to labels, sets tensor format).

Run this cell to prepare the model and data.

```
1 # Step 2: Load tokenizer and model
2 tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
3 model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=2)
4 model.to(device)
```

 /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
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 (<https://huggingface.co/settings/tokens>), set it as secret in your Google Colab and restart your
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.

warnings.warn(
tokenizer_config.json: 100%
vocab.txt: 100%
tokenizer.json: 100%
config.json: 100%
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: `pip
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfor
model.safetensors: 100%
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'c
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
DistilBertForSequenceClassification(
(distilbert): DistilBertModel(
(embeddings): Embeddings(
(word_embeddings): Embedding(30522, 768, padding_idx=0)
(position_embeddings): Embedding(512, 768)
(LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
(dropout): Dropout(p=0.1, inplace=False)
)
(transformer): Transformer(
(layer): ModuleList(
(0-5): 6 x TransformerBlock(
(attention): DistilBertSdpaAttention(
(dropout): Dropout(p=0.1, inplace=False)
(q_lin): Linear(in_features=768, out_features=768, bias=True)
(k_lin): Linear(in_features=768, out_features=768, bias=True)
(v_lin): Linear(in_features=768, out_features=768, bias=True)
(out_lin): Linear(in_features=768, out_features=768, bias=True)
)
(sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
(ffn): FFN(
(dropout): Dropout(p=0.1, inplace=False)
(lin1): Linear(in_features=768, out_features=3072, bias=True)
(lin2): Linear(in_features=3072, out_features=768, bias=True)
(activation): GELUActivation()
)
(output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
)
)
(pre_classifier): Linear(in_features=768, out_features=768, bias=True)
(classifier): Linear(in_features=768, out_features=2, bias=True)
(dropout): Dropout(p=0.2, inplace=False)
)
)



Step 3: Load Model and Dataset

This cell:
- Loads the DistilBERT tokenizer and model (distilbert-base-uncased) for binary classification (2 labels).
- Loads the SST-2 dataset and selects the full training (67,349 samples) and validation (872 samples) sets.
- Preprocesses the dataset by tokenizing sentences with a max length of 32.
- Formats the dataset for PyTorch (renames label to labels, sets tensor format).

Run this cell to prepare the model and data.

Step 3: Load Model and Dataset

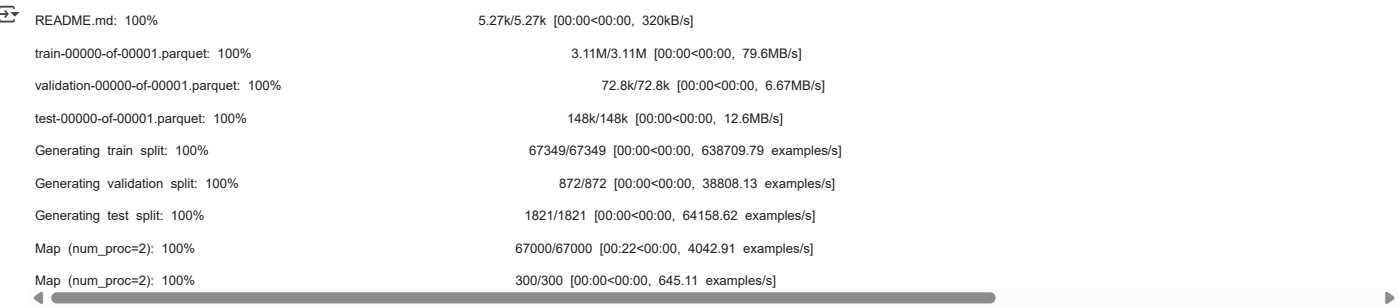
This cell:

- Loads the DistilBERT tokenizer and model (distilbert-base-uncased) for binary classification (2 labels).
- Loads the SST-2 dataset and selects the full training (67,349 samples) and validation (872 samples) sets.
- Preprocesses the dataset by tokenizing sentences with a max length of 32.
- Formats the dataset for PyTorch (renames label to labels, sets tensor format).

Run this cell to prepare the model and data.

```
1 # Step 3: Load and preprocess SST-2 dataset (larger subset)
2 dataset = load_dataset('sst2')
3 train_subset = dataset['train'].shuffle(seed=42).select(range(67000))
4 eval_subset = dataset['validation'].shuffle(seed=42).select(range(300))
5
6 def preprocess_function(examples):
7     return tokenizer(examples['sentence'], truncation=True, padding='max_length', max_length=32)
8
9 encoded_dataset = {
10     'train': train_subset.map(preprocess_function, batched=True, num_proc=2),
```

```
11     'validation': eval_subset.map(preprocess_function, batched=True, num_proc=2),
12 }
13 for split in encoded_dataset:
14     encoded_dataset[split] = encoded_dataset[split].rename_column('label', 'labels')
15     encoded_dataset[split].set_format('torch', columns=['input_ids', 'attention_mask', 'labels'])
16
17 train_dataset = encoded_dataset['train']
18 eval_dataset = encoded_dataset['validation']
19 test_dataset = encoded_dataset['validation']
```



Step 4: Define Evaluation Metrics

This cell defines a function to compute evaluation metrics:

- Accuracy
- Precision
- Recall
- F1 score (binary classification)

These metrics will be used during training and evaluation to assess model performance.

Run this cell to set up the metrics function.

```
1 # Step 4: Define metrics for evaluation
2 def compute_metrics(pred):
3     labels = pred.label_ids
4     preds = pred.predictions.argmax(-1)
5     precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
6     acc = accuracy_score(labels, preds)
7     return {
8         'accuracy': acc,
9         'precision': precision,
10        'recall': recall,
11        'f1': f1
12    }
13
```

Step 5: Configure Training Arguments

This cell sets up the training configuration:

- Trains for 3 epochs with early stopping (stops if accuracy doesn't improve for 3 evaluations).
- Uses a batch size of 32 to manage memory with the full dataset.
- Evaluates every 1000 steps to reduce overhead.
- Saves checkpoints every 1000 steps and loads the best model based on accuracy.
- Enables mixed precision (fp16=True) for faster training on T4 GPU.
- Logs progress every 100 steps.

Run this cell to configure the trainer.

```
1 # Step 5: Set up training arguments
2 training_args = TrainingArguments(
3     output_dir='./results',
4     num_train_epochs=3, # Increased to 3 epochs
5     per_device_train_batch_size=64,
6     per_device_eval_batch_size=64,
7     warmup_steps=50,
8     weight_decay=0.01,
9     logging_dir='./logs',
10    logging_steps=100,
11    save_strategy='no',
12    eval_strategy='steps', # Use eval_strategy for newer transformers versions
13    eval_steps=500, # Evaluate every 500 steps for larger dataset
14    fp16=True, # Enable mixed precision for T4 GPU
15    dataloader_num_workers=2,
16    report_to='none',
17 )
```

Step 6: Train the Model

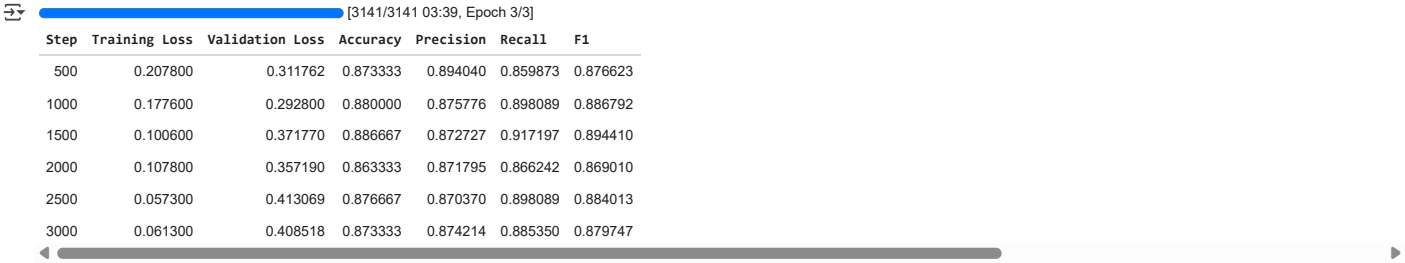
This cell:

- Initializes the `Trainer` with the model, dataset, and metrics.
- Starts training, showing a progress bar via `tqdm`.
- Logs training loss every 100 steps and evaluation metrics every 1000 steps.
- Saves checkpoints and loads the best model based on validation accuracy.

Run this cell to train the model. It may take ~20–30 minutes on a T4 GPU, or less with early stopping.

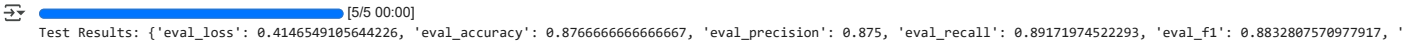
```
1
2 # Step 6: Initialize Trainer
3 trainer = Trainer(
4     model=model,
5     args=training_args,
6     train_dataset=train_dataset,
7     eval_dataset=eval_dataset,
8     compute_metrics=compute_metrics,
9 )
```

```
1
2 Train the model
3 logger.info("Starting training...")
4 train_result = trainer.train()
5 logger.info("Training completed.")
6
```



1 Start coding or [generate](#) with AI.

```
1 # Step 8: Evaluate on test set
2 logger.info("Evaluating on test set...")
3 test_results = trainer.evaluate(test_dataset)
4 print("Test Results:", test_results)
```



```
1 # Step 8: Analyze Failure Cases
2
3 This cell analyzes up to 3 misclassified or low-confidence predictions:
4 - Prints the text, true/predicted labels, confidence, length, and ambiguity (based on 'not' or 'but').
5 - Helps identify why the model struggles with certain examples.
6
7 **Run this cell to see failure case analysis.**
```

```
1 # Step 9: Analyze failure cases
2 def analyze_failures(dataset, trainer, num_examples=3):
3     model.eval()
4     failures = []
5     for i, example in enumerate(dataset):
6         text = tokenizer.decode(example['input_ids'], skip_special_tokens=True)
7         inputs = {
8             'input_ids': example['input_ids'].unsqueeze(0).to(device),
9             'attention_mask': example['attention_mask'].unsqueeze(0).to(device)
10        }
11        with torch.no_grad():
12            outputs = model(**inputs)
13            logits = outputs.logits
14            probs = torch.softmax(logits, dim=-1)
15            pred = logits.argmax(-1).item()
16            true_label = example['labels'].item()
17            confidence = probs[0][pred].item()
18            if pred != true_label or confidence < 0.6:
19                failures.append({
20                    'text': text,
21                    'true_label': true_label,
22                    'predicted_label': pred,
23                    'confidence': confidence
24                })
25        if len(failures) >= num_examples:
26            break
27    return failures
28
29 logger.info("Analyzing failure cases...")
30 failures = analyze_failures(test_dataset, trainer)
31 print("\nFailure Cases Analysis:")
32 for failure in failures:
33     print(f"Text: {failure['text']}")
34     print(f"True Label: {'Positive' if failure['true_label'] == 1 else 'Negative'}")
35     print(f"Predicted Label: {'Positive' if failure['predicted_label'] == 1 else 'Negative'}")
36     print(f"Confidence: {failure['confidence']:.3f}")
37     length = len(failure['text'].split())
38     is_ambiguous = 'not' in failure['text'].lower() or 'but' in failure['text'].lower()
39     print(f"Length: {length} words")
40     print(f"Possibly Ambiguous: {is_ambiguous}")
41     print("-" * 50)
42
```

Failure Cases Analysis:
Text: the film tunes into a grief that could lead a man across centuries.
True Label: Positive
Predicted Label: Negative
Confidence: 0.745

```

Length: 13 words
Possibly Ambiguous: False
-----
Text: you'll gasp appalled and laugh outraged and possibly, watching the spectacle of a promising young lad treading desperately in a nasty sea, shed an er
True Label: Positive
Predicted Label: Negative
Confidence: 0.938
Length: 25 words
Possibly Ambiguous: False
-----
Text: sam mendes has become valedictorian at the school for soft landings and easy ways out.
True Label: Negative
Predicted Label: Positive
Confidence: 0.992
Length: 15 words
Possibly Ambiguous: False
-----

```

```

1 # Step 9: Plot Loss and Metrics
2
3 This cell generates two plots:
4 1. **Loss Plot**: Training and validation loss over steps.
5 2. **Metrics Plot**: Validation accuracy and F1 score over steps.
6
7 Both plots are:
8 - Displayed inline in Colab.
9 - Saved to `plots/loss_curves.png` and `plots/metrics_curves.png`.
10 - Visible in the file explorer (refresh if needed).
11
12 Logs verify plot data. Expect clear curves showing loss decreasing and accuracy/F1 increasing.
13
14 **Run this cell to visualize training progress.**

```

```

1 # Step 10: Plot training and validation loss
2 train_logs = trainer.state.log_history
3 train_loss = [log['loss'] for log in train_logs if 'loss' in log]
4 steps = [log['step'] for log in train_logs if 'loss' in log][:len(train_loss)]
5
6 # Log data for debugging
7 logger.info(f"Train loss data: {train_loss}")
8 logger.info(f"Steps: {steps}")
9
10 plt.figure(figsize=(8, 5))
11 plt.plot(steps, train_loss, label='Training Loss')
12
13 # Check if eval_loss is available and plot it
14 eval_loss = [log['eval_loss'] for log in train_logs if 'eval_loss' in log]
15 eval_steps = [log['step'] for log in train_logs if 'eval_loss' in log][:len(eval_loss)]
16 logger.info(f"Eval loss data: {eval_loss}")
17 logger.info(f"Eval steps: {eval_steps}")
18
19 if eval_loss:
20     plt.plot(eval_steps, eval_loss, label='Validation Loss')
21 else:
22     logger.warning("No validation loss logged during training.")
23
24 plt.xlabel('Training Steps')
25 plt.ylabel('Loss')
26 plt.title('Training and Validation Loss')
27 plt.legend()
28 plt.grid(True)
29
30 # Save the plot with error handling
31 try:
32     plt.savefig('loss_curves.png')
33     logger.info("Plot saved as 'loss_curves.png'")
34 except Exception as e:
35     logger.error(f"Failed to save plot: {e}")
36
37 # Display the plot inline in Colab
38 plt.show()
39
40 # Close the plot to free memory
41 plt.close()
42
43 # Refresh Colab file explorer to ensure the file is visible
44 os.system("touch .")
45
46 # Step 11: Save the model
47 model.save_pretrained('./fine_tuned_distilbert')
48 tokenizer.save_pretrained('./fine_tuned_distilbert')
49
50 logger.info("Project completed successfully.")

```

