Installing Necessary Modules

```
!pip install transformers
!pip3 install datasets
!pip3 install py7zr
!pip3 install peft
!pip3 install evaluate
!pip install accelerate>=0.20.1
```

Show hidden output

Loading Dataset

```
from datasets import load_dataset

dataset = load_dataset("stanfordnlp/sst2")
dataset_train = dataset['train']
dataset_val = dataset['validation']
print(len(dataset_train))
print(len(dataset_val))
```

```
/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
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(
README.md: 100%  5.27k/5.27k [00:00<00:00, 564kB/s]
train-00000-of-00001.parquet: 100%  3.11M/3.11M [00:00<00:00, 15.9MB/s]
validation-00000-of-00001.parquet: 100%  72.8k/72.8k [00:00<00:00, 5.81MB/s]
test-00000-of-00001.parquet: 100%  148k/148k [00:00<00:00, 16.0MB/s]
Generating train split: 100%  67349/67349 [00:00<00:00, 799465.05 examples/s]
Generating validation split: 100%  872/872 [00:00<00:00, 74203.84 examples/s]
Generating test split: 100%  1821/1821 [00:00<00:00, 138725.82 examples/s]
67349
872
```

Fine Tuning BERT

Load Model

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from peft import LoraConfig, get_peft_model
import torch

bert_tokenizer = AutoTokenizer.from_pretrained("google-bert/bert-base-uncased")
bert_model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-uncased")
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 4.51kB/s]
config.json: 100%  570/570 [00:00<00:00, 67.9kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 573kB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 3.15MB/s]
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
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 down
model.safetensors: 100%  440M/440M [00:04<00:00, 117MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at google-bert/bert-base-uncased and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Tokenize data using Bert tokenizer

```
def tokenize_text(batch):
  return bert_tokenizer(batch["sentence"],padding="max_length", truncation=True)


dataset_train_tokenized = dataset_train.map(tokenize_text, batched=True, remove_columns=["idx"])
dataset_val_tokenized = dataset_val.map(tokenize_text, batched=True, remove_columns=["idx"])
```

```
Map: 100%  67349/67349 [00:17<00:00, 3566.32 examples/s]
Map: 100%  872/872 [00:00<00:00, 3841.20 examples/s]
```

```
import numpy as np
import evaluate

import torch
from sklearn.metrics import precision_recall_fscore_support, accuracy_score

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = torch.argmax(torch.tensor(logits), dim=-1).numpy()
    # labels = labels.numpy()
    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average='binary')
```

```
    acc = accuracy_score(labels, predictions)
    return {"accuracy": acc, "precision": precision, "recall": recall, "f1": f1}

batch_size = 32
num_epochs = 3
learning_rate = 1e-5
```

Define Training Arguments

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(
    output_dir="bert_sentiment_analysis",
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    learning_rate=learning_rate,
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    logging_steps=50,
    save_steps=1e6,
    gradient_accumulation_steps=16,
    load_best_model_at_end=True
)
```

Fine-tune model

```
# Initialize Trainer
trainer = Trainer(
    model = bert_model,
    args = training_args,
    train_dataset = dataset_train_tokenized,
    eval_dataset = dataset_val_tokenized,
    compute_metrics = compute_metrics,
    tokenizer = bert_tokenizer
)

trainer.train()
```

```
<ipython-input-9-0a02ab279296>:2: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_
  trainer = Trainer(
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: ··········
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: yashwanthsaipathipati (yashwanthsaipathipati-university-of-south-florida) to https://api.wandb.ai. Use `wandb login --rel
Tracking run with wandb version 0.19.9
Run data is saved locally in /content/wandb/run-20250415_184337-1l9zloof
Syncing run bert_sentiment_analysis to Weights & Biases (docs)
View project at https://wandb.ai/yashwanthsaipathipati-university-of-south-florida/huggingface
View run at https://wandb.ai/yashwanthsaipathipati-university-of-south-florida/huggingface/runs/1l9zloof
[393/393 1:03:09, Epoch 2/3]
```

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
0	0.282500	0.232203	0.913991	0.924138	0.905405	0.914676
1	0.193600	0.234153	0.908257	0.899123	0.923423	0.911111
2	0.176500	0.233880	0.909404	0.906459	0.916667	0.911534

```
TrainOutput(global_step=393, training_loss=0.24094253277960626, metrics={'train_runtime': 3820.7057, 'train_samples_per_second': 52.882,
'train_steps_per_second': 0.103, 'total_flos': 5.302634965303296e+16, 'train_loss': 0.24094253277960626, 'epoch': 2.9957244655581947})
```

Evaluate and Save Model

```
results = trainer.evaluate()
```

```
[28/28 00:05]
```

```
bert_accuracy = results['eval_accuracy']
bert_precision = results['eval_precision']
bert_recall = results['eval_recall']
bert_f1 = results['eval_f1']


trainer.save_model("bert_sentiment_analysis")
```

Fine Tuning DISTILBERT

Load Model

```
distilbert_tokenizer = AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased")
distilbert_model = AutoModelForSequenceClassification.from_pretrained("distilbert/distilbert-base-uncased")

lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    lora_dropout=0.1,
    task_type="SEQ_CLS",
    target_modules=["distilbert.transformer.layer.0.attention.q_lin",
                    "distilbert.transformer.layer.0.attention.k_lin",
                    "distilbert.transformer.layer.0.attention.v_lin",
                    "distilbert.transformer.layer.0.attention.out_lin",
                    "distilbert.transformer.layer.0.ffn.lin1",
                    "distilbert.transformer.layer.0.ffn.lin2",
                    "distilbert.transformer.layer.1.attention.q_lin",
                    "distilbert.transformer.layer.1.attention.k_lin",
                    "distilbert.transformer.layer.1.attention.v_lin",
                    "distilbert.transformer.layer.1.attention.out_lin",
                    "distilbert.transformer.layer.1.ffn.lin1",
                    "distilbert.transformer.layer.1.ffn.lin2",
                    "distilbert.transformer.layer.2.attention.q_lin",
                    "distilbert.transformer.layer.2.attention.k_lin",
                    "distilbert.transformer.layer.2.attention.v_lin",
                    "distilbert.transformer.layer.2.attention.out_lin",
                    "distilbert.transformer.layer.2.ffn.lin1",
                    "distilbert.transformer.layer.2.ffn.lin2",
                    "distilbert.transformer.layer.3.attention.q_lin",
                    "distilbert.transformer.layer.3.attention.k_lin",
                    "distilbert.transformer.layer.3.attention.v_lin",
                    "distilbert.transformer.layer.3.attention.out_lin",
                    "distilbert.transformer.layer.3.ffn.lin1",
                    "distilbert.transformer.layer.3.ffn.lin2",
                    "distilbert.transformer.layer.4.attention.q_lin",
                    "distilbert.transformer.layer.4.attention.k_lin",
                    "distilbert.transformer.layer.4.attention.v_lin",
                    "distilbert.transformer.layer.4.attention.out_lin",
                    "distilbert.transformer.layer.4.ffn.lin1",
                    "distilbert.transformer.layer.4.ffn.lin2",
                    "distilbert.transformer.layer.5.attention.q_lin",
                    "distilbert.transformer.layer.5.attention.k_lin",
                    "distilbert.transformer.layer.5.attention.v_lin",
                    "distilbert.transformer.layer.5.attention.out_lin",
                    "distilbert.transformer.layer.5.ffn.lin1",
                    "distilbert.transformer.layer.5.ffn.lin2"]
)

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

```
tokenizer_config.json: 100%                 48.0/48.0 [00:00<00:00, 6.25kB/s]
config.json: 100%                           483/483 [00:00<00:00, 63.7kB/s]
vocab.txt: 100%                             232k/232k [00:00<00:00, 2.91MB/s]
tokenizer.json: 100%                        466k/466k [00:00<00:00, 11.7MB/s]
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
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 down
model.safetensors: 100%                     268M/268M [00:01<00:00, 154MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert/distilbert-base-uncased and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
trainable params: 1,255,682 || all params: 68,210,692 || trainable%: 1.8409
```

Tokenize data using DistilBERT tokenizer

```
def distilbert_tokenize_text(batch):
  return distilbert_tokenizer(batch["sentence"],padding="max_length", truncation=True)


dataset_train_tokenized = dataset_train.map(distilbert_tokenize_text, batched=True, remove_columns=["idx"])
dataset_val_tokenized = dataset_val.map(distilbert_tokenize_text, batched=True, remove_columns=["idx"])
```

```
Map: 100%                                   67349/67349 [00:12<00:00, 5412.64 examples/s]
Map: 100%                                   872/872 [00:00<00:00, 5217.70 examples/s]
```

Define Training Arguments

```
training_args = TrainingArguments(
    output_dir="distilbert_sentiment_analysis",
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    learning_rate=learning_rate,
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    logging_steps=50,
    save_steps=1e6,
    gradient_accumulation_steps=16,
    load_best_model_at_end=True
)
```

Fine-tune Model

```
# Initialize Trainer
trainer = Trainer(
    model = distilbert_model,
    args = training_args,
    train_dataset = dataset_train_tokenized,
    eval_dataset = dataset_val_tokenized,
    compute_metrics = compute_metrics,
    tokenizer = distilbert_tokenizer
)

trainer.train()
```

```
<ipython-input-17-e1542281a8df>:2: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing
  trainer = Trainer(
No label_names provided for model class `PeftModelForSequenceClassification`. Since `PeftModel` hides base models input arguments, if label_names is not
```

[393/393 27:58, Epoch 2/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
0	0.645600	0.524526	0.816514	0.784000	0.882883	0.830508
1	0.356200	0.376966	0.826835	0.844706	0.808559	0.826237
2	0.345200	0.369490	0.829128	0.842227	0.817568	0.829714

```
TrainOutput(global_step=393, training_loss=0.45861285454747636, metrics={'train_runtime': 1682.485, 'train_samples_per_second': 120.088,
'train_steps_per_second': 0.234, 'total_flos': 2.747436572737536e+16, 'train_loss': 0.45861285454747636, 'epoch': 2.9957244655581947})
```

Evaluate and Save Model

```
results = trainer.evaluate()
```

[28/28 00:03]

```
distilbert_accuracy = results['eval_accuracy']
distilbert_precision = results['eval_precision']
distilbert_recall = results['eval_recall']
distilbert_f1 = results['eval_f1']


trainer.save_model("distilbert_sentiment_analysis")
```

Knowledge Distillation from finetuned BERT to untrained distilBERT model

Define DistillationTrainingArguments and DistillationTrainer Classes

```
from transformers import TrainingArguments

class DistillationTrainingArguments(TrainingArguments):
    def __init__(self, *args, alpha=0.5, temperature=2.0, **kwargs):
        super().__init__(*args, **kwargs)
        self.alpha = alpha
        self.temperature = temperature


import torch.nn as nn
import torch.nn.functional as F
from transformers import Trainer

class DistillationTrainer(Trainer):
    def __init__(self, *args, teacher_model=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.teacher_model = teacher_model

    def compute_loss(self, model, inputs, return_outputs=False, num_items_in_batch=None):
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        inputs = inputs.to(device)
        outputs_stu = model(**inputs)

        loss_ce = outputs_stu.loss
        logits_stu = outputs_stu.logits

        with torch.no_grad():
            outputs_tea = self.teacher_model(**inputs)
            logits_tea = outputs_tea.logits

        loss_fct = nn.KLDivLoss(reduction="batchmean")
        loss_kd = self.args.temperature ** 2 * loss_fct(
            F.log_softmax(logits_stu / self.args.temperature, dim=-1),
            F.softmax(logits_tea / self.args.temperature, dim=-1))

        loss = self.args.alpha * loss_ce + (1. - self.args.alpha) * loss_kd
        return (loss, outputs_stu) if return_outputs else loss
```

Load Student Model

```
distilbert_kd_model = AutoModelForSequenceClassification.from_pretrained("distilbert/distilbert-base-uncased")

lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    lora_dropout=0.1,
    task_type="SEQ_CLS",
```

```
        target_modules=["distilbert.transformer.layer.0.attention.q_lin",
                        "distilbert.transformer.layer.0.attention.k_lin",
                        "distilbert.transformer.layer.0.attention.v_lin",
                        "distilbert.transformer.layer.0.attention.out_lin",
                        "distilbert.transformer.layer.0.ffn.lin1",
                        "distilbert.transformer.layer.0.ffn.lin2",
                        "distilbert.transformer.layer.1.attention.q_lin",
                        "distilbert.transformer.layer.1.attention.k_lin",
                        "distilbert.transformer.layer.1.attention.v_lin",
                        "distilbert.transformer.layer.1.attention.out_lin",
                        "distilbert.transformer.layer.1.ffn.lin1",
                        "distilbert.transformer.layer.1.ffn.lin2",
                        "distilbert.transformer.layer.2.attention.q_lin",
                        "distilbert.transformer.layer.2.attention.k_lin",
                        "distilbert.transformer.layer.2.attention.v_lin",
                        "distilbert.transformer.layer.2.attention.out_lin",
                        "distilbert.transformer.layer.2.ffn.lin1",
                        "distilbert.transformer.layer.2.ffn.lin2",
                        "distilbert.transformer.layer.3.attention.q_lin",
                        "distilbert.transformer.layer.3.attention.k_lin",
                        "distilbert.transformer.layer.3.attention.v_lin",
                        "distilbert.transformer.layer.3.attention.out_lin",
                        "distilbert.transformer.layer.3.ffn.lin1",
                        "distilbert.transformer.layer.3.ffn.lin2",
                        "distilbert.transformer.layer.4.attention.q_lin",
                        "distilbert.transformer.layer.4.attention.k_lin",
                        "distilbert.transformer.layer.4.attention.v_lin",
                        "distilbert.transformer.layer.4.attention.out_lin",
                        "distilbert.transformer.layer.4.ffn.lin1",
                        "distilbert.transformer.layer.4.ffn.lin2",
                        "distilbert.transformer.layer.5.attention.q_lin",
                        "distilbert.transformer.layer.5.attention.k_lin",
                        "distilbert.transformer.layer.5.attention.v_lin",
                        "distilbert.transformer.layer.5.attention.out_lin",
                        "distilbert.transformer.layer.5.ffn.lin1",
                        "distilbert.transformer.layer.5.ffn.lin2"]
    )

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

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert/distilbert-base-uncased and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
trainable params: 1,255,682 || all params: 68,210,692 || trainable%: 1.8409
```

```
batch_size = 48
num_epochs = 3
learning_rate = 2e-5
```

Define Training Arguments

```
kd_training_args = DistillationTrainingArguments(
    output_dir="distilbert_kd_sentiment_analysis",
    per_device_train_batch_size=batch_size,
    weight_decay=0.01,
    eval_strategy = "epoch",
    learning_rate = learning_rate,
    gradient_accumulation_steps=4,
    num_train_epochs = num_epochs,
    logging_steps=500,
    save_steps=1000,
    eval_steps=1000,
    save_total_limit=2,
    fp16=True,
    report_to="none"
)
```

```
import torch
from transformers import AutoConfig

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

def student_init():
    return (AutoModelForSequenceClassification
            .from_pretrained("distilbert-base-uncased", config=distilbert_kd_model.config).to(device))
```

Perform KD

```
trainer = DistillationTrainer(model_init=student_init,
    teacher_model=bert_model, args=kd_training_args,
    train_dataset=dataset_train_tokenized, eval_dataset=dataset_val_tokenized,
    compute_metrics=compute_metrics, tokenizer=distilbert_tokenizer)
```

```
<ipython-input-22-2423a8622cb5>:7: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `DistillationTrainer.__init__`. Use
  super().__init__(*args, **kwargs)
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
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 down
model.safetensors: 100%                                                 268M/268M [00:01<00:00, 148MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
trainer.train()
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

[1053/1053 26:27, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	No log	0.151425	0.903670	0.909091	0.900901	0.904977
2	0.185300	0.144507	0.907110	0.904232	0.914414	0.909295
3	0.106300	0.147760	0.905963	0.900442	0.916667	0.908482

```
TrainOutput(global_step=1053, training_loss=0.14339027649317032, metrics={'train_runtime': 1588.8815, 'train_samples_per_second': 127.163,
'train_steps_per_second': 0.663, 'total_flos': 2.676464049624883e+16, 'train_loss': 0.14339027649317032, 'epoch': 3.0})
```

Evaluate and Save Model

```
results = trainer.evaluate()
```

[109/109 00:06]

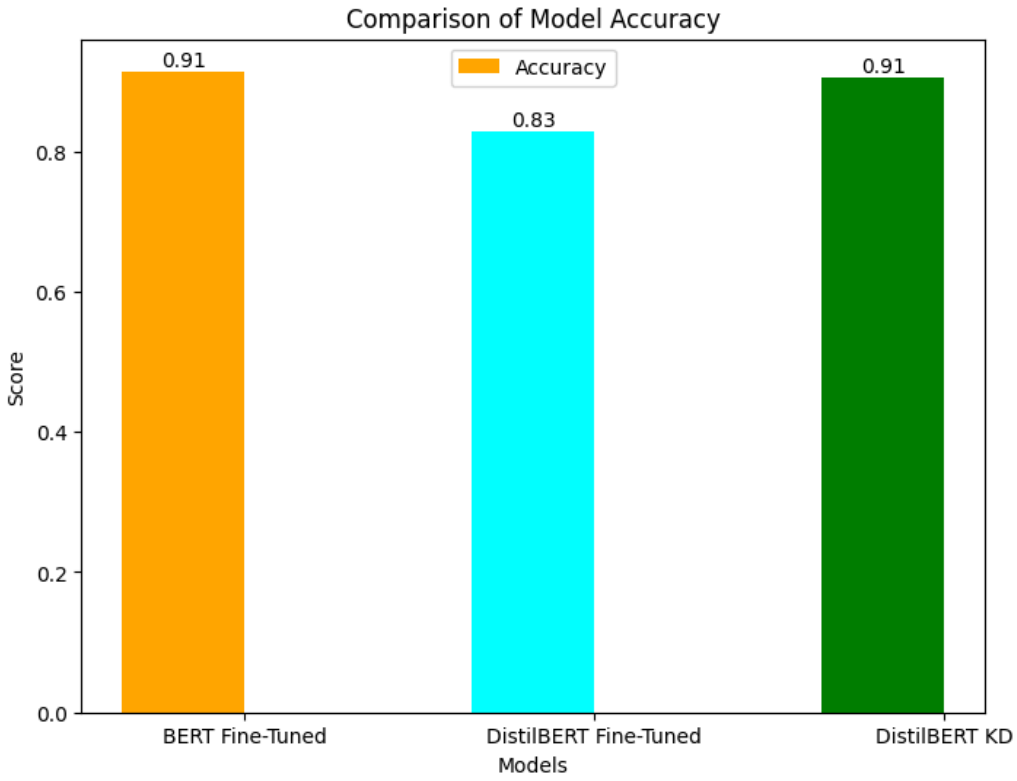
```
distilbert_kd_accuracy = results['eval_accuracy']
distilbert_kd_precision = results['eval_precision']
distilbert_kd_recall = results['eval_recall']
distilbert_kd_f1 = results['eval_f1']


trainer.save_model("distilbert_kd_sentiment_analysis")
```

Plot the Results

Accuracy

```
import numpy as np
import matplotlib.pyplot as plt

models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
accuracy = [bert_accuracy, distilbert_accuracy, distilbert_kd_accuracy]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x - width/2, accuracy, width, label='Accuracy', color=colors)
```

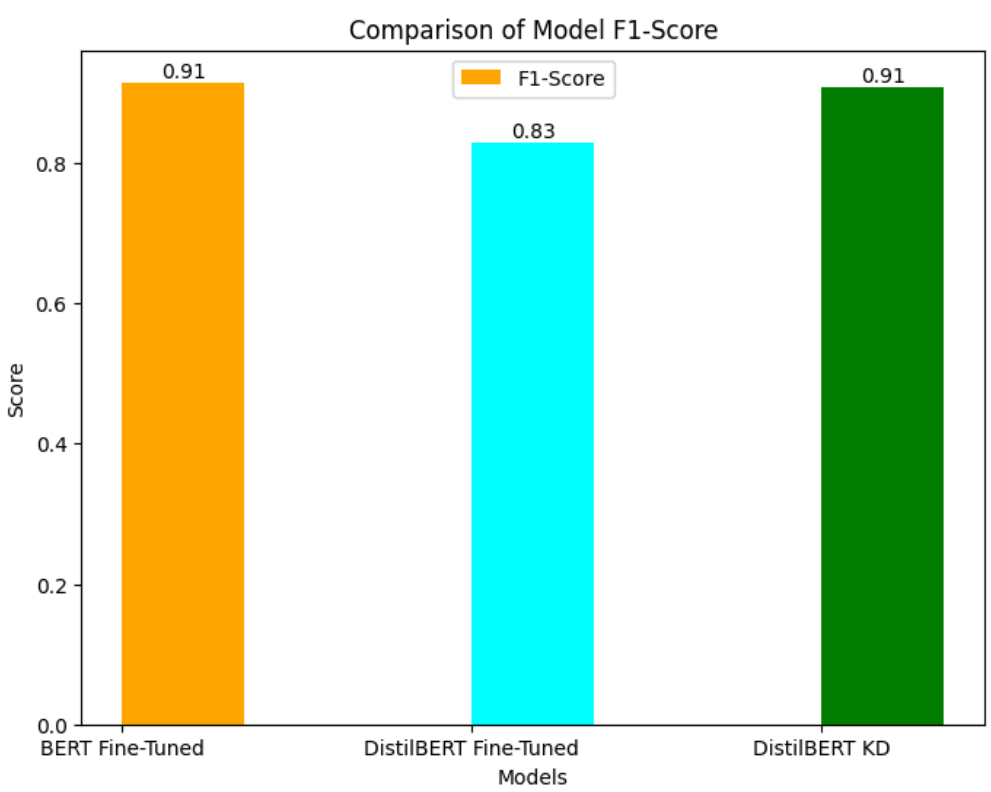
```
for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')

ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model Accuracy")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



F1-Score

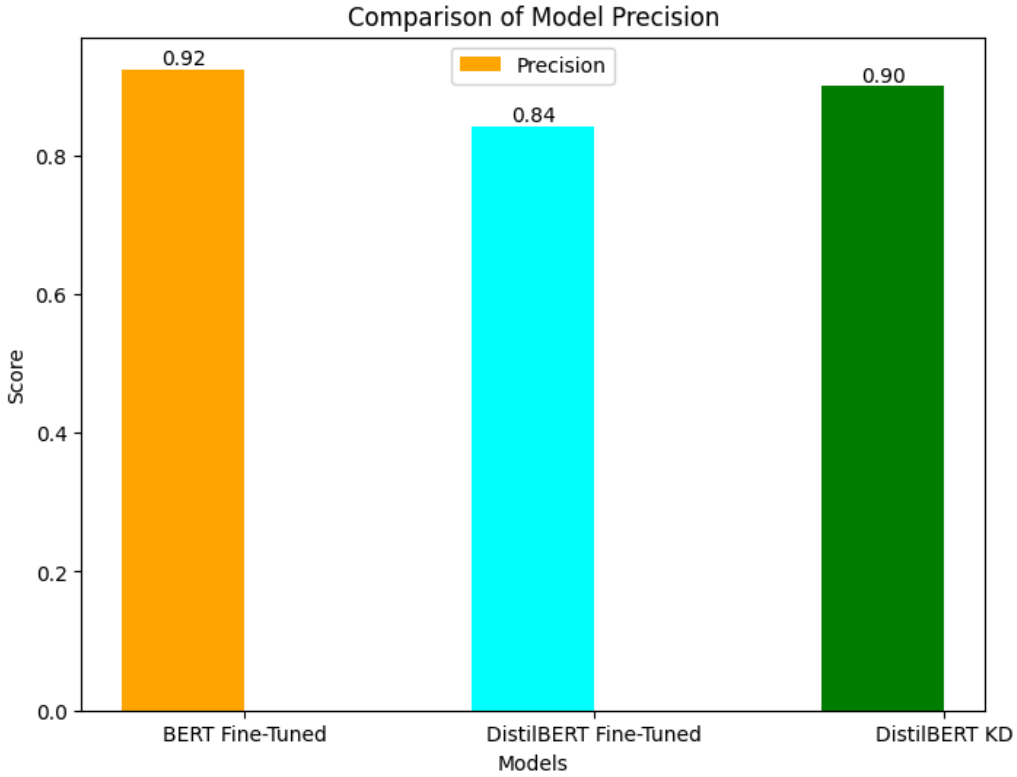
```
models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
f1_score = [bert_f1, distilbert_f1, distilbert_kd_f1]
colors = ['orange', 'cyan', 'green']
```

```
x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x + width/2, f1_score, width, label='F1-Score', color=colors)

for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')

ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model F1-Score")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



Precision

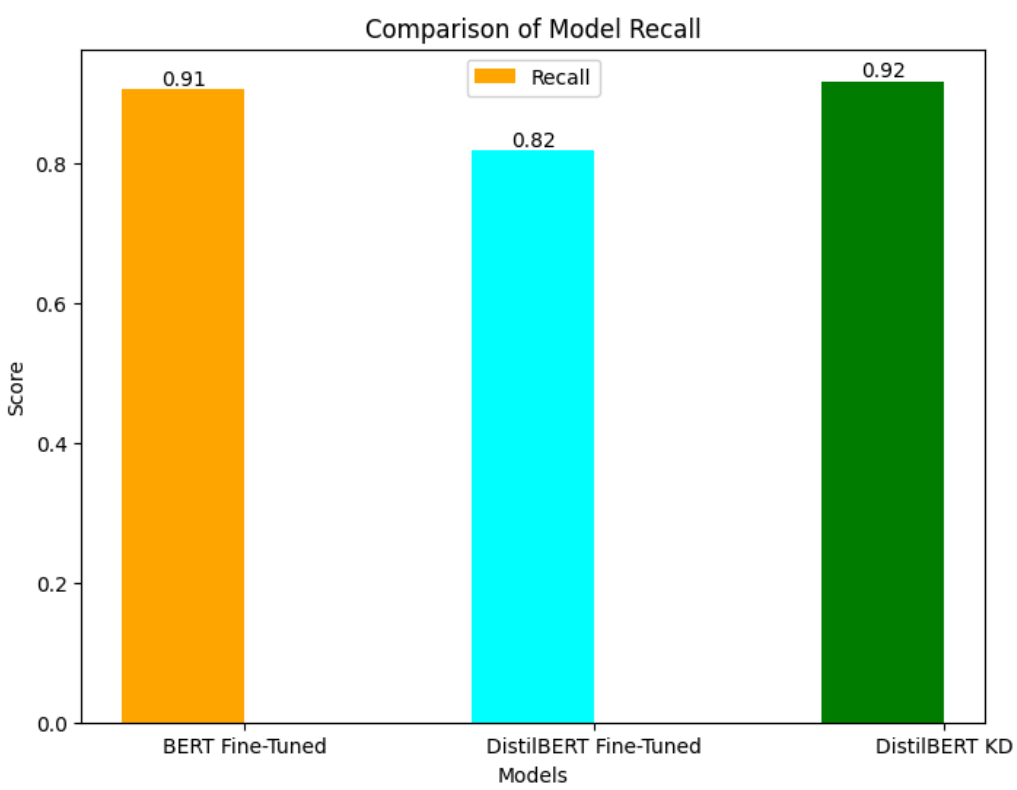
```
import numpy as np
import matplotlib.pyplot as plt

models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
precision = [bert_precision, distilbert_precision, distilbert_kd_precision]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x - width/2, precision, width, label='Precision', color=colors)


for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')

ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model Precision")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



Recall

```
import numpy as np
import matplotlib.pyplot as plt

models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
recall = [bert_recall, distilbert_recall, distilbert_kd_recall]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x - width/2, recall, width, label='Recall', color=colors)


for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')
```

```
ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model Recall")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



Teacher-Student Model Parameters Comparision

```
from transformers import AutoModelForSequenceClassification
import os

def compute_parameters(model_path):
  model = AutoModelForSequenceClassification.from_pretrained(model_path)
  parameters = model.num_parameters()
  return parameters
```

```
bert_model_parameters = compute_parameters(model_path="/content/bert_sentiment_analysis")
bert_model_parameters
```

```
109483778
```

```
distilbert_model_parameters = compute_parameters(model_path="/content/distilbert_sentiment_analysis")
distilbert_model_parameters
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert/distilbert-base-uncased and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
68210692
```

```
distilbert_model_kd_parameters = compute_parameters(model_path="/content/distilbert_kd_sentiment_analysis")
distilbert_model_kd_parameters
```

```
66955010
```

```
decrease = (bert_model_parameters-distilbert_model_kd_parameters)/bert_model_parameters
print(decrease*100)
```

```
38.84481224241275
```

Teacher-Student Model Size Comaprision

```
!ls /content/bert_sentiment_analysis -al --block-size=MB
```

```
total 439MB
drwxr-xr-x 5 root root   1MB Apr 15 19:47 .
drwxr-xr-x 1 root root   1MB Apr 15 20:15 ..
drwxr-xr-x 2 root root   1MB Apr 15 19:04 checkpoint-131
drwxr-xr-x 2 root root   1MB Apr 15 19:25 checkpoint-262
drwxr-xr-x 2 root root   1MB Apr 15 19:46 checkpoint-393
-rw-r--r-- 1 root root   1MB Apr 15 19:47 config.json
-rw-r--r-- 1 root root 438MB Apr 15 19:47 model.safetensors
-rw-r--r-- 1 root root   1MB Apr 15 19:47 special_tokens_map.json
-rw-r--r-- 1 root root   1MB Apr 15 19:47 tokenizer_config.json
-rw-r--r-- 1 root root   1MB Apr 15 19:47 tokenizer.json
-rw-r--r-- 1 root root   1MB Apr 15 19:47 training_args.bin
-rw-r--r-- 1 root root   1MB Apr 15 19:47 vocab.txt
```

```
!ls /content/distilbert_kd_sentiment_analysis -al --block-size=MB
```

```
total 269MB
drwxr-xr-x 4 root root   1MB Apr 15 20:42 .
drwxr-xr-x 1 root root   1MB Apr 15 20:15 ..
drwxr-xr-x 2 root root   1MB Apr 15 20:40 checkpoint-1000
drwxr-xr-x 2 root root   1MB Apr 15 20:41 checkpoint-1053
-rw-r--r-- 1 root root   1MB Apr 15 20:42 config.json
-rw-r--r-- 1 root root 268MB Apr 15 20:42 model.safetensors
-rw-r--r-- 1 root root   1MB Apr 15 20:42 special_tokens_map.json
-rw-r--r-- 1 root root   1MB Apr 15 20:42 tokenizer_config.json
```

```
-rw-r--r-- 1 root root   1MB Apr 15 20:42 tokenizer.json
-rw-r--r-- 1 root root   1MB Apr 15 20:42 training_args.bin
-rw-r--r-- 1 root root   1MB Apr 15 20:42 vocab.txt
```

Teacher-Student Inference Time Comparision

```
from transformers import pipeline
import time

pipe = pipeline("text-classification", model="/content/bert_sentiment_analysis", tokenizer='bert-base-uncased', device='cpu')

sample_input = dataset['train']['sentence'][101]

for _ in range(10):
  _ = pipe(sample_input)

start = time.time()
for _ in range(100):
  _ = pipe(sample_input)
total_time_bert_model = time.time()-start
print("Total time to process 100 requests for BERT Model: ",total_time_bert_model)
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 5.79kB/s]
config.json: 100%  570/570 [00:00<00:00, 74.1kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 2.50MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 10.2MB/s]
Device set to use cpu
Total time to process 100 requests for BERT Model:  2.0406835079193115
```

```
from transformers import pipeline
import time

pipe = pipeline("text-classification", model="/content/distilbert_kd_sentiment_analysis", tokenizer='distilbert-base-uncased', device='cpu')

sample_input = dataset['train']['sentence'][101]

for _ in range(10):
  _ = pipe(sample_input)

start = time.time()
for _ in range(100):
  _ = pipe(sample_input)
total_time_distilbert_kd_model = time.time()-start
print("Total time to process 100 requests for DISTILBERT KD Model: ",total_time_distilbert_kd_model)
```

```
tokenizer_config.json: 100%                 48.0/48.0 [00:00<00:00, 5.62kB/s]
config.json: 100%                           483/483 [00:00<00:00, 57.7kB/s]
vocab.txt: 100%                             232k/232k [00:00<00:00, 23.8MB/s]
tokenizer.json: 100%                        466k/466k [00:00<00:00, 13.6MB/s]
Device set to use cpu
Total time to process 100 requests for DISTILBERT KD Model:  1.0931005477905273
```

```
decrease_in_time = (total_time_bert_model-total_time_distilbert_kd_model)/total_time_bert_model
print(decrease_in_time*100)
```

```
46.43458706122162
```

Trusworthy (Robustness) Evaluation

Load IMDB dataset

```
from datasets import load_dataset

imdb_dataset = load_dataset("stanfordnlp/imdb")
imdb_dataset_test = imdb_dataset['test']
print(len(imdb_dataset_test))
```

```
README.md: 100%                             7.81k/7.81k [00:00<00:00, 956kB/s]
train-00000-of-00001.parquet: 100%          21.0M/21.0M [00:00<00:00, 55.7MB/s]
test-00000-of-00001.parquet: 100%           20.5M/20.5M [00:00<00:00, 77.9MB/s]
unsupervised-00000-of-00001.parquet: 100%   42.0M/42.0M [00:00<00:00, 82.7MB/s]
Generating train split: 100%                25000/25000 [00:00<00:00, 145737.66 examples/s]
Generating test split: 100%                 25000/25000 [00:00<00:00, 157265.10 examples/s]
Generating unsupervised split: 100%         50000/50000 [00:00<00:00, 200333.14 examples/s]
25000
```

Teacher Model evaluation

```
model_path = "/content/bert_sentiment_analysis"
model = AutoModelForSequenceClassification.from_pretrained(model_path)


model.eval()
```

Show hidden output

```
def preprocess(batch):
  return bert_tokenizer(batch["text"],padding="max_length", truncation=True, return_tensors="pt")


from sklearn.metrics import accuracy_score, precision_recall_fscore_support

y_true = []
y_pred = []

for sample in imdb_dataset_test:
    inputs = preprocess(sample)
    input_ids = inputs["input_ids"].squeeze(0)
    attention_mask = inputs["attention_mask"].squeeze(0)

    with torch.no_grad():
        outputs = model(input_ids.unsqueeze(0), attention_mask=attention_mask.unsqueeze(0))
        logits = outputs.logits
        prediction = torch.argmax(logits, dim=1).item()

    y_pred.append(prediction)
    y_true.append(torch.tensor([sample["label"]]))


accuracy = accuracy_score(y_true, y_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average="macro")


bert_accuracy = accuracy*100
bert_precision = precision*100
bert_recall = recall*100
bert_f1 = f1*100


print(f"Accuracy:  {bert_accuracy:.4f}")
print(f"Precision: {bert_precision:.4f}")
print(f"Recall:    {bert_recall:.4f}")
print(f"F1 Score:  {bert_f1:.4f}")
```

```
Accuracy:  86.0840
Precision: 86.9774
Recall:    86.0840
F1 Score:  85.9994
```

Student Model Evaluation

```
model_path = "/content/distilbert_sentiment_analysis"
model = AutoModelForSequenceClassification.from_pretrained(model_path)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert/distilbert-base-uncased and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
model.eval()
```

```
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): lora.Linear(
              (base_layer): Linear(in_features=768, out_features=768, bias=True)
              (lora_dropout): ModuleDict(
                (default): Dropout(p=0.1, inplace=False)
              )
              (lora_A): ModuleDict(
                (default): Linear(in_features=768, out_features=8, bias=False)
              )
              (lora_B): ModuleDict(
                (default): Linear(in_features=8, out_features=768, bias=False)
              )
              (lora_embedding_A): ParameterDict()
              (lora_embedding_B): ParameterDict()
              (lora_magnitude_vector): ModuleDict()
            )
            (k_lin): lora.Linear(
              (base_layer): Linear(in_features=768, out_features=768, bias=True)
              (lora_dropout): ModuleDict(
                (default): Dropout(p=0.1, inplace=False)
              )
              (lora_A): ModuleDict(
                (default): Linear(in_features=768, out_features=8, bias=False)
              )
              (lora_B): ModuleDict(
                (default): Linear(in_features=8, out_features=768, bias=False)
              )
              (lora_embedding_A): ParameterDict()
              (lora_embedding_B): ParameterDict()
              (lora_magnitude_vector): ModuleDict()
            )
            (v_lin): lora.Linear(
              (base_layer): Linear(in_features=768, out_features=768, bias=True)
              (lora_dropout): ModuleDict(
                (default): Dropout(p=0.1, inplace=False)
              )
              (lora_A): ModuleDict(
                (default): Linear(in_features=768, out_features=8, bias=False)
              )
              (lora_B): ModuleDict(
```

```
                    (default): Linear(in_features=8, out_features=768, bias=False)
                  )
                  (lora_embedding_A): ParameterDict()
                  (lora_embedding_B): ParameterDict()
                  (lora_magnitude_vector): ModuleDict()
                )

def preprocess(batch):
  return distilbert_tokenizer(batch["text"],padding="max_length", truncation=True, return_tensors="pt")


from sklearn.metrics import accuracy_score, precision_recall_fscore_support

y_true = []
y_pred = []

for sample in imdb_dataset_test:
    inputs = preprocess(sample)
    input_ids = inputs["input_ids"].squeeze(0)
    attention_mask = inputs["attention_mask"].squeeze(0)

    with torch.no_grad():
        outputs = model(input_ids.unsqueeze(0), attention_mask=attention_mask.unsqueeze(0))
        logits = outputs.logits
        prediction = torch.argmax(logits, dim=1).item()

    y_pred.append(prediction)
    y_true.append(torch.tensor([sample["label"]]))


accuracy = accuracy_score(y_true, y_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average="macro")


distilbert_accuracy = accuracy*100
distilbert_precision = precision*100
distilbert_recall = recall*100
distilbert_f1 = f1*100


print(f"Accuracy:  {distilbert_accuracy:.4f}")
print(f"Precision: {distilbert_precision:.4f}")
print(f"Recall:    {distilbert_recall:.4f}")
print(f"F1 Score:  {distilbert_f1:.4f}")
```

```
Accuracy:  83.1120
Precision: 84.7049
Recall:    83.1120
F1 Score:  82.9160
```

Distilled Model Evaluation

```
model_path = "/content/distilbert_kd_sentiment_analysis"
model = AutoModelForSequenceClassification.from_pretrained(model_path)


model.eval()
```

```
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)
)
```

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

y_true = []
y_pred = []

for sample in imdb_dataset_test:
    inputs = preprocess(sample)
    input_ids = inputs["input_ids"].squeeze(0)
    attention_mask = inputs["attention_mask"].squeeze(0)

    with torch.no_grad():
        outputs = model(input_ids.unsqueeze(0), attention_mask=attention_mask.unsqueeze(0))
        logits = outputs.logits
        prediction = torch.argmax(logits, dim=1).item()
```

```
        y_pred.append(prediction)
        y_true.append(torch.tensor([sample["label"]]))


accuracy = accuracy_score(y_true, y_pred)
precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average="macro")


distilbert_kd_accuracy = accuracy*100
distilbert_kd_precision = precision*100
distilbert_kd_recall = recall*100
distilbert_kd_f1 = f1*100


print(f"Accuracy:  {distilbert_kd_accuracy:.4f}")
print(f"Precision: {distilbert_kd_precision:.4f}")
print(f"Recall:    {distilbert_kd_recall:.4f}")
print(f"F1 Score:  {distilbert_kd_f1:.4f}")
```

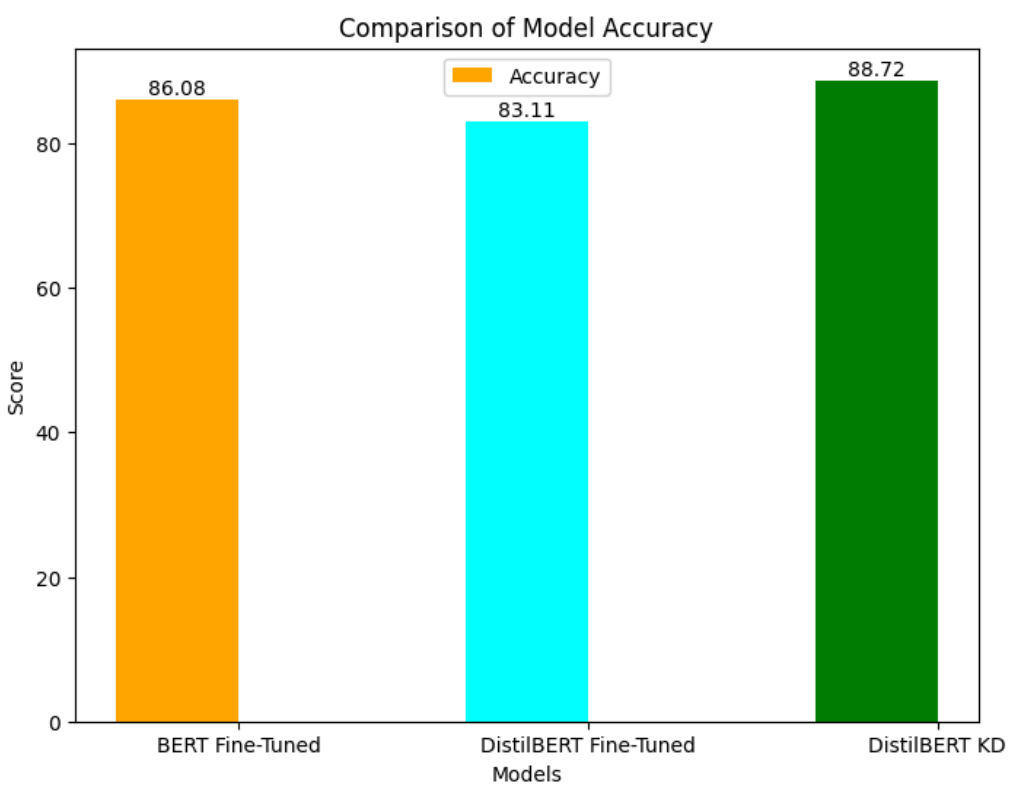
```
Accuracy:  88.7240
Precision: 88.8967
Recall:    88.7240
F1 Score:  88.7115
```

Comparing Results

Accuracy Plot

```
import numpy as np
import matplotlib.pyplot as plt

models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
accuracy = [bert_accuracy, distilbert_accuracy, distilbert_kd_accuracy]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x - width/2, accuracy, width, label='Accuracy', color=colors)


for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')

ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model Accuracy")
```

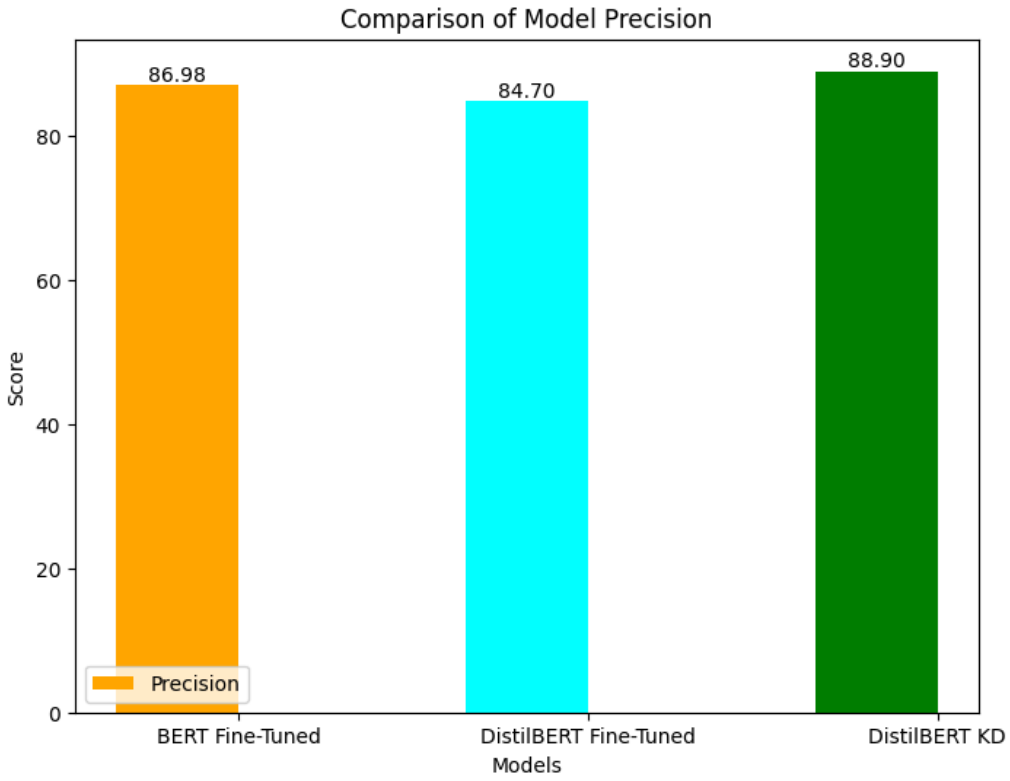
```
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



Precision Plot

```
import numpy as np
import matplotlib.pyplot as plt

models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
precision = [bert_precision, distilbert_precision, distilbert_kd_precision]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x - width/2, precision, width, label='Precision', color=colors)
```

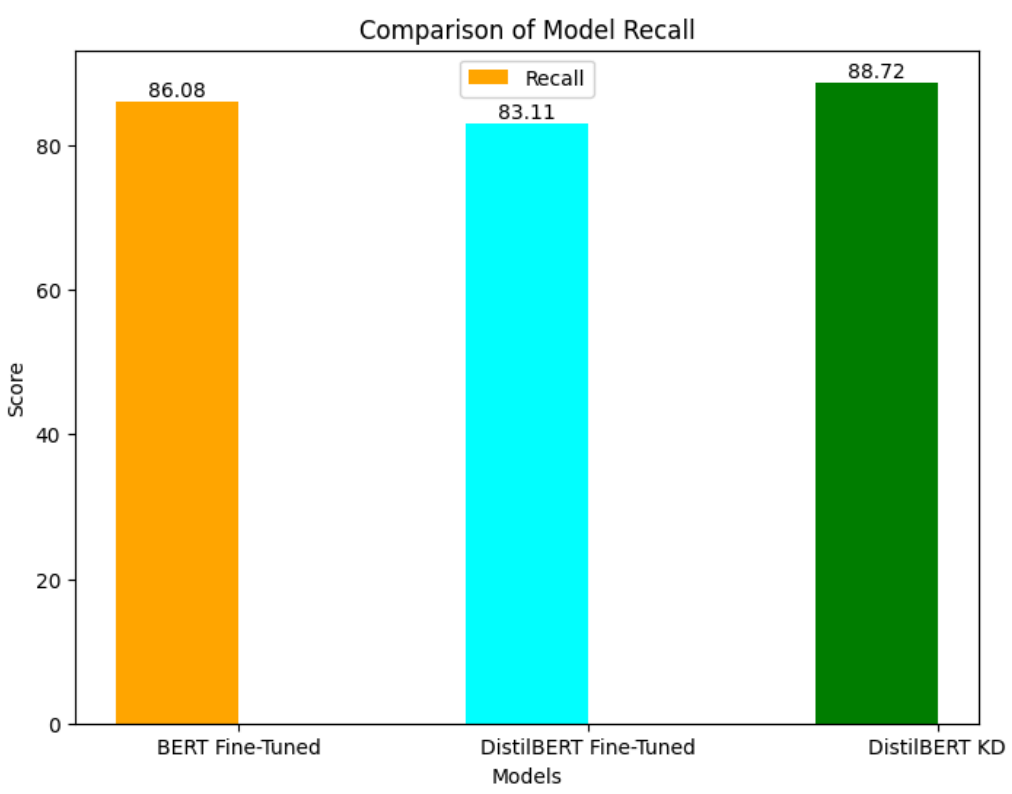
```
for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')

ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model Precision")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



Recall Plot

```
import numpy as np
import matplotlib.pyplot as plt

models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
```

```
recall = [bert_recall, distilbert_recall, distilbert_kd_recall]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x - width/2, recall, width, label='Recall', color=colors)


for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')

ax.set_xlabel("Models")
ax.set_ylabel("Score")
ax.set_title("Comparison of Model Recall")
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.show()
```



f1-score Plot

```
models = ["BERT Fine-Tuned", "DistilBERT Fine-Tuned", "DistilBERT KD"]
f1_score = [bert_f1, distilbert_f1, distilbert_kd_f1]
colors = ['orange', 'cyan', 'green']

x = np.arange(len(models))
width = 0.35

fig, ax = plt.subplots(figsize=(8, 6))

bars = ax.bar(x + width/2, f1_score, width, label='F1-Score', color=colors)

for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.2f}', ha='center', va='bottom')
```