```
# Colab: run in a code cell (Linux shell)
!pip install -q transformers datasets evaluate scikit-learn pandas matplotlib

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import Pipeline
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 5.2 MB/s eta 0:00:00
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Load dataset
file_path = "/content/drive/My Drive/train_v2_drcat_02 2.csv"  # Update this if needed
df = pd.read_csv(file_path)
```

```
# Display the first few rows and basic info to understand the data
print("Dataset Head:")
print(df.head())
print("\nDataset Info:")
df.info()
```

```
Dataset Head:
                                                text  label  \
0  Phones\n\nModern humans today are always on th...      0
1  This essay will explain if drivers should or s...      0
2  Driving while the use of cellular devices\n\nT...      0
3  Phones & Driving\n\nDrivers should not be able...      0
4  Cell Phone Operation While Driving\n\nThe abil...      0

          prompt_name           source  RDizzl3_seven
0  Phones and driving  persuade_corpus          False
1  Phones and driving  persuade_corpus          False
2  Phones and driving  persuade_corpus          False
3  Phones and driving  persuade_corpus          False
4  Phones and driving  persuade_corpus          False

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44868 entries, 0 to 44867
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   text           44868 non-null  object
 1   label          44868 non-null  int64
 2   prompt_name    44868 non-null  object
 3   source         44868 non-null  object
 4   RDizzl3_seven  44868 non-null  bool
dtypes: bool(1), int64(1), object(3)
memory usage: 1.4+ MB
```

```
# PREPROCESS
# Check the distribution of labels (0 for human, 1 for AI)
print("\nLabel Distribution:")
print(df['label'].value_counts())

# Separate the two classes
df_human = df[df['label'] == 0]
df_ai = df[df['label'] == 1]

# Downsample the majority class (human) to match the size of the minority class (AI)
# This creates a balanced dataset for training
df_human_downsampled = df_human.sample(len(df_ai), random_state=42)

# Combine the downsampled human data with the AI data
df_balanced = pd.concat([df_human_downsampled, df_ai])

# Verify the new balanced distribution
print("\nBalanced Label Distribution:")
print(df_balanced['label'].value_counts())

# Define our features (X) and target (y)
X = df_balanced['text']
y = df_balanced['label']
```

```
Label Distribution:
label
0    27371
1    17497
Name: count, dtype: int64

Balanced Label Distribution:
label
0    17497
1    17497
Name: count, dtype: int64
```

```
# Split the balanced dataset into training and testing sets
# We'll use 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print(f"\nTraining set size: {len(X_train)}")
print(f"Testing set size: {len(X_test)}")

# Create a machine learning pipeline
# Pipeline Step 1: TfidfVectorizer - Converts text into numerical vectors.
# Pipeline Step 2: LogisticRegression - A simple but effective classification model.
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('classifier', LogisticRegression(random_state=42))
])
```

```
Training set size: 27995
Testing set size: 6999
```

```
# Fine-tune lightweight models

# ==============================================================================
# STEP 1: SETUP AND IMPORTS
# ==============================================================================
# Ensure all libraries are installed
!pip install transformers datasets evaluate scikit-learn pandas -q

import pandas as pd
import numpy as np
import time
import torch
import os
import shutil
import evaluate
from datasets import Dataset
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer

print("Setup complete. Libraries are ready.")


# ==============================================================================
# STEP 2: REUSABLE FUNCTIONS
# ==============================================================================

def prepare_data(file_path):
    """Loads, balances, and prepares the data for Hugging Face."""
    df = pd.read_csv(file_path)
    df_human = df[df['label'] == 0]
    df_ai = df[df['label'] == 1]
    df_human_downsampled = df_human.sample(len(df_ai), random_state=42)
    df_balanced = pd.concat([df_human_downsampled, df_ai])

    train_texts, test_texts, train_labels, test_labels = train_test_split(
        df_balanced['text'].tolist(),
        df_balanced['label'].tolist(),
        test_size=0.2,
        random_state=42,
        stratify=df_balanced['label'].tolist()
    )

    train_dataset = Dataset.from_dict({'text': train_texts, 'label': train_labels})
    test_dataset = Dataset.from_dict({'text': test_texts, 'label': test_labels})

    return train_dataset, test_dataset

def fine_tune_and_evaluate(model_checkpoint, train_dataset, test_dataset):
    """Fine-tunes a model and returns the trainer and evaluation results."""
    tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
    def tokenize_function(examples):
        return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=512)

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

    model = AutoModelForSequenceClassification.from_pretrained(model_checkpoint, num_labels=2)

    # Define metrics
    accuracy_metric = evaluate.load("accuracy")
    f1_metric = evaluate.load("f1")

    def compute_metrics(eval_pred):
        logits, labels = eval_pred
        predictions = np.argmax(logits, axis=-1)
        accuracy = accuracy_metric.compute(predictions=predictions, references=labels)
        f1 = f1_metric.compute(predictions=predictions, references=labels, average="weighted")
        return {"accuracy": accuracy["accuracy"], "f1": f1["f1"]}

    training_args = TrainingArguments(
        output_dir=f"./{model_checkpoint}-finetuned",
        eval_strategy="epoch",
        num_train_epochs=3,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        warmup_steps=100,
        weight_decay=0.01,
        logging_steps=50,
        save_strategy="epoch",
        load_best_model_at_end=True,
        report_to="none",
    )

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

    print(f"\n--- Starting Training for {model_checkpoint} ---")
    trainer.train()
    print(f"--- Finished Training for {model_checkpoint} ---")

    eval_results = trainer.evaluate(tokenized_test)
    return trainer, eval_results


def measure_inference_speed(trainer, tokenizer, test_dataset, num_samples=200):
    """Measures the average inference time in milliseconds."""
    total_time = 0
    # Use a small sample of the test set for speed measurement
    sample_dataset = test_dataset.select(range(num_samples))

    print(f"\n--- Measuring inference speed on {num_samples} samples... ---")
    for i in range(num_samples):
        text = sample_dataset[i]['text']
        inputs = tokenizer(
            text,
            return_tensors="pt",
            truncation=True,      # <-- Add this
            max_length=512        # <-- And this
        ).to(trainer.model.device)

        start_time = time.time()
        with torch.no_grad():
            _ = trainer.model(**inputs)
        end_time = time.time()

        total_time += (end_time - start_time)

    avg_time_ms = (total_time / num_samples) * 1000
    return avg_time_ms

def get_model_size(trainer):
    """Calculates the size of the model in MB."""
    # Save the model to disk to measure its size
    temp_dir = "./temp_model_size_check"
    trainer.save_model(temp_dir)
    # Calculate size
    total_size = 0
```

```
    for path, dirs, files in os.walk(temp_dir):
        for f in files:
            fp = os.path.join(path, f)
            total_size += os.path.getsize(fp)
    # Clean up
    import shutil
    shutil.rmtree(temp_dir)
    return total_size / (1024 * 1024) # Convert bytes to MB


# ==============================================================================
# STEP 3: MAIN EXPERIMENT EXECUTION
# ==============================================================================

# List of models to compare
models_to_test = [
    "distilbert-base-uncased",
]

# Store results
all_results = []

# Your baseline results from the previous step
baseline_results = {
    'Model': 'Baseline (TF-IDF + LogReg)',
    'Accuracy': 0.9913,
    'F1-Score': 0.99, # From the weighted avg in your screenshot
    'Inference Time (ms)': 1.5, # Placeholder: A simple model is very fast, adjust if you measure it
    'Model Size (MB)': 5, # Placeholder: TF-IDF models are usually small
}
all_results.append(baseline_results)

# Load the data once
file_path = "/content/drive/My Drive/train_v2_drcat_02 2.csv"
train_dataset, test_dataset = prepare_data(file_path)

# Loop through each model, train, evaluate, and measure
for model_name in models_to_test:
    # Fine-tune the model
    trainer, eval_metrics = fine_tune_and_evaluate(model_name, train_dataset, test_dataset)

    # Measure efficiency
    inference_time = measure_inference_speed(trainer, trainer.tokenizer, test_dataset)
    model_size = get_model_size(trainer)

    # Store the results
    result = {
        'Model': model_name,
        'Accuracy': eval_metrics['eval_accuracy'],
        'F1-Score': eval_metrics['eval_f1'],
        'Inference Time (ms)': inference_time,
        'Model Size (MB)': model_size,
    }
    all_results.append(result)
    print(f"\n--- Results for {model_name} collected ---")


# ==============================================================================
# STEP 4: DISPLAY THE FINAL RESULTS TABLE
# ==============================================================================

print("\n\n--- ALL EXPERIMENTS COMPLETE ---")

results_df = pd.DataFrame(all_results)
results_df['Accuracy'] = results_df['Accuracy'].map(lambda x: f"{x:.4f}")
results_df['F1-Score'] = results_df['F1-Score'].map(lambda x: f"{x:.4f}")
results_df['Inference Time (ms)'] = results_df['Inference Time (ms)'].map(lambda x: f"{x:.2f}")
results_df['Model Size (MB)'] = results_df['Model Size (MB)'].map(lambda x: f"{x:.2f}")

print("\nFinal Comparative Results Table:")
print(results_df.to_string())
```

```
Setup complete. Libraries are ready.
/usr/local/lib/python3.12/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/token
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% 48.0/48.0 [00:00<00:00, 5.55kB/s]

config.json: 100% 483/483 [00:00<00:00, 60.3kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 3.07MB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 3.50MB/s]

Map: 100% 27995/27995 [00:38<00:00, 782.11 examples/s]

Map: 100% 6999/6999 [00:10<00:00, 750.52 examples/s]

model.safetensors: 100% 268M/268M [00:05<00:00, 52.0MB/s]

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

Downloading builder script: 4.20k/? [00:00<00:00, 222kB/s]

Downloading builder script: 6.79k/? [00:00<00:00, 436kB/s]

```
/tmp/ipython-input-1500925267.py:85: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 fo
  trainer = Trainer(

--- Starting Training for distilbert-base-uncased ---
```

[5250/5250 1:12:31, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.029400	0.015587	0.995857	0.995857
2	0.023900	0.034599	0.993142	0.993142
3	0.000000	0.017317	0.996714	0.996714

```
--- Finished Training for distilbert-base-uncased ---
```

[438/438 01:46]

```
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.

--- Measuring inference speed on 200 samples... ---

--- Results for distilbert-base-uncased collected ---


--- ALL EXPERIMENTS COMPLETE ---

Final Comparative Results Table:
                       Model Accuracy F1-Score Inference Time (ms) Model Size (MB)
0  Baseline (TF-IDF + LogReg)   0.9913   0.9900               1.50            5.00
1     distilbert-base-uncased   0.9959   0.9959               4.88          256.33
```