American International University-Bangladesh (AIUB)

Faculty of Science & Technology (FST)

Department of Computer Science

Natural Language Processing

Final-Term Project Report

Fall 2025-2026

Section: B

Group: 6

Project Contributions

SL #	Student Name & ID	Contributions (mention every part of the project where you contributed)
1.	Name: Md. Jamir Shekh ID: 22-48468-3	Dataset,Code,Methodology
2.	Name: Jabedul Islam ID: 23-50993-1	Code,Implementation,Result Analysis
3.	Name: Mahmudul Hasan Maruf ID: 22-48580-3	Abstract, Introduction, Background,Conclusion

AI Content Detection Using BERT and DeBERTa

Md. Jamir Shekh, Jabedul Islam, Mahmudul Hasan Maruf
American International University-Bangladesh
Dhaka, Bangladesh
22-48468-3@student.aiub.edu, 23-50993-1@student.aiub.edu, 22-48580-3@student.aiub.edu

Abstract

In this project we made a system that can tell if a text was written by a human or by a computer program like an AI. These days lots of people are using tools, like ChatGPT so it is getting hard to figure out who actually wrote something. This is a problem because schools need to make sure students are writing their own work and being honest. We also need to stop false information from spreading all over the internet. To solve this problem, we cleaned up our data through preprocessing, and then we fine-tuned two models called BERT and DeBERTa. We trained them using a mixed dataset of articles written by humans and AI. DeBERTa performed way better than BERT. This is mainly because DeBERTa is better at understanding the context and the position of words in a sentence. Overall, the system we created is reliable and can spot AI writing with good accuracy.

1 Introduction

In this project, we worked on the problem of "AI Content Detection" or identifying AI writing. Currently, large language models like ChatGPT are generating text so accurately that it has become almost impossible for ordinary people to distinguish between human writing and AI writing. Our goal is to create an automated system that can accurately tell whether a text is written by a human or created by an AI tool.

Our main motivation behind working on this problem is to protect the authenticity of information and the quality of education. Currently, students in educational institutions often write assignments or essays using AI tools instead of writing them themselves, which hinders their learning and is a kind of 'plagiarism'. On the other hand, fake news is spread very easily using AI on social media and online. It was necessary to build a system that could restore people’s trust in the information on the internet and assess the talent of real content writers.

To solve this problem, we used ‘Deep Learning’ and ‘Natural Language Processing’ (NLP) technologies. Instead of building a model from scratch, we took the help of ‘Transfer Learning’ methods. For this, we used two very powerful pre-trained transformer models BERT and DeBERTa.

We trained these models on a large dataset of human writing and AI-generated writing. The models learned to distinguish between subtle patterns within the writing, word usage, and sentence structure. Finally, we compared the results of these two models to see which one provided the best solution for the task.

2 Background

2.1 BERT Architecture

BERT's full form is Bidirectional Encoder Representations from Transformers. It is basically based on the 'Encoder' mechanism of the Transformer architecture. Ordinary models read text from only one direction (left to right), but the biggest feature of BERT is that it is 'Bidirectional'. It can read the words of a sentence from both the left and right sides at the same time, which helps it understand the deeper meaning of the sentence.

Working Process of BERT:

1. Input Representation:

   When a sentence is input to the BERT model, it is processed at three different levels:
   Token Embeddings: Converting words into mathematical vectors.
   Segment Embeddings: Identifying different parts of the sentence.
   Position Embeddings: Remembering the position or position of each word in the sentence.

2. **Self-Attention Mechanism:**

   The main strength of BERT is its 'attention' layer. It analyses the relationship between each word in a sentence and all the other words. For example, BERT can understand whether the word "bank" in a sentence means a riverbank or a financial institution by looking at the surrounding words.

3. Pre-training Techniques:

   BERT mainly learns using two techniques:

Masked Language Model (MLM): It hides some words in the sentence (Masks) and tries to guess what that word could be based on the context.

Next Sentence Prediction (NSP): It predicts whether it is logical for a sentence to be followed by another sentence.

In our project, we used this pre-trained BERT model and 'fine-tune' it on our human and AI text datasets.

2.2 DeBERTa Architecture

The second model we used is DeBERTa (Decoding-enhanced BERT with Disentangled Attention). It is an advanced model than BERT and RoBERTa. Microsoft created this model to overcome the limitations of Google's BERT and Meta's RoBERTa.

Working Process of DeBERTa:

DeBERTa mainly works based on two main innovative techniques, which make it different and powerful than others:

1. **Disentangled Attention:**

   In the normal BERT model, the meaning (Content) of a word and its position (Position) are added into a vector. But DeBERTa keeps these two things separate (Disentangled). It uses two different vectors:

- One vector for the content or meaning of the word.
- The other vector for the relative position of the word. This allows the model to understand what the "semantic" relationship is between words and what their "positional" relationship is in the sentence. This separate attention mechanism makes the model much more efficient at understanding sentence structure.

2. **Enhanced Mask Decoder (EMD):**

Like BERT, DeBERTa also learns by guessing masked words (MLM). However, DeBERTa adds an extra step here. In the very last layer of the model (Decoding Layer), just before predicting the masked word, it takes the 'Absolute Position' information of the word as input. This is called the Enhanced Mask Decoder. This helps the model recognize the correct word and syntax.

Simply put, DeBERTa performed better than BERT in recognizing the complex sentences written by AI in our project because it gives different importance to the meaning and position of words.

3 Methodology

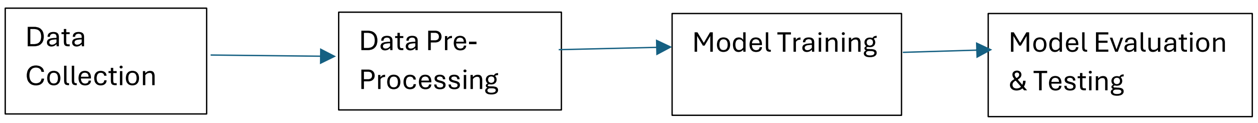


Fig: Working Process

3.1 Data Collection

We have collected a dataset called Training_Essay_Data.csv from Kaggle. This dataset has a total of 20,000 rows and 2 columns (text and generated). The dataset contains 10,000 human-written essays classify as 0 and 10,000 AI-generated essays classify as 1. This balanced distribution will prevent our model from being biased towards any particular class.

3.2 Data Preprocessing

We have done some important technical work in the data pre-processing step to prepare the text data as input for the model. The steps in this process are discussed in detail below:

Tokenization: We used bert-base-uncased for BERT and microsoft/deberta-v3-small pre-trained tokenizers for DeBERTa. These tokenizers convert human-readable text into small sub-words or tokens that the model can understand.

Sequence length configuration: We have set a maximum length of 512 tokens for each input sequence. If any essay is larger than this, it is truncated , and if it is smaller, additional tokens are added to make it equal.

Padding & Truncation: The use of truncation=True and padding=True in the code ensures that all input data is of the same size, which is essential for batch processing.

Label Alignment: Our original dataset had a generated column to differentiate between AI and human. We used a pre-processing function to map the values of this generated column directly to the labels column so that the model could learn the output directly.

Column Removal: After the tokenized sort was created, we removed both the original text and generated columns from the dataset. This resulted in a fully prepared dataset with only the mathematical tokens and labels required by the model.

3.3 Model Training

The modeling process basically consists of two distinct phases—pre-training and fine-tuning. Below is the detailed and technical discussion:

BERT Pre-training

The BERT model is trained in such a way that it understands the context from both sides of the text simultaneously. There are basically two special methods used for this bidirectional ability. The first method called Masked Language Modeling (MLM): In this method 15% of the words of the input text are randomly selected. Of these, 80% of words are replaced with a special [MASK] token, 10% of words are replaced with wrong or random words, and the remaining 10% are left unchanged. The model tries to guess these missing words by looking at the nearby context, which helps it understand the deeper structure of the language. This prevents the model from copying words directly and forces it to understand the true meaning of words.

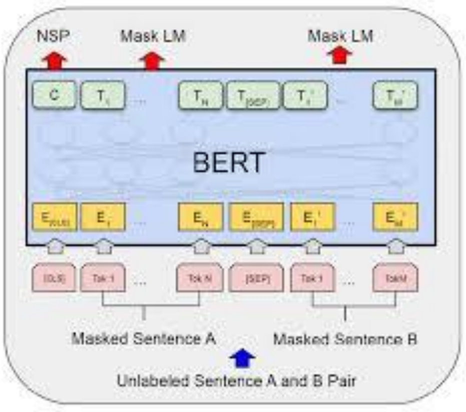


Fig1: Pre-training BERT

The second method is Next Sentence Prediction (NSP). In this method the model is given a pair of sentences. Half the time the second sentence is a logical continuation of the first (IsNext) and the other half the time it is an unrelated random sentence (NotNext). The model learns whether the second sentence is actually a sentence after the first. This process is very useful for understanding the interrelationship and logical flow between sentences.

About 800 million words of BooksCorpus and 2.5 billion words of English Wikipedia are used for pre-training. Only original text passages from Wikipedia are used here to maintain the normal flow of sentences and paragraph structure. It helps the model to understand the overall meaning of the document.

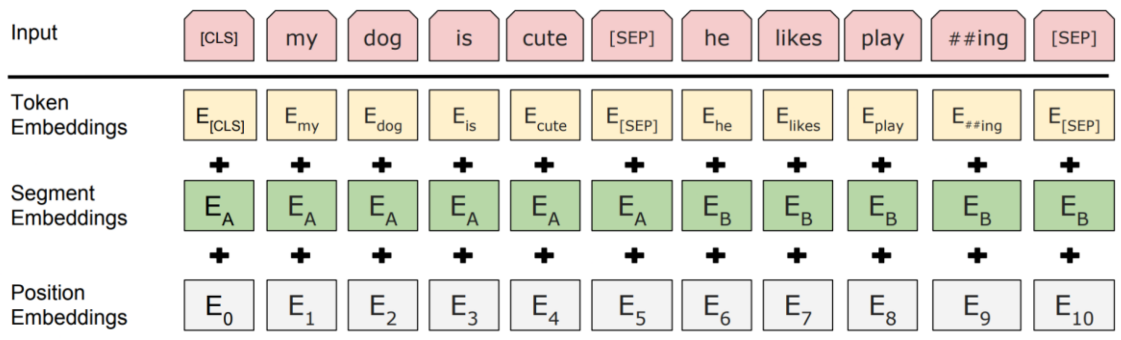


Fig2: BERT Model

Fine-tuning

The fine-tuning procedure is much easier and faster than pre-training. In our project this step ensures the following technical aspects.

Fine tuning BERT is self-attention mechanism enables it to work with single text or text pairs. In our case, each essay is given as input and the model verifies its validity using its internal knowledge. According to our code, we added a binary classification head on top of the pre-trained model.

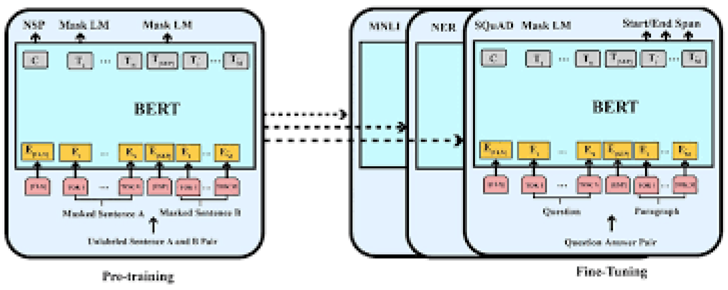


Fig3: Pre-training and Fine tuning of BERT

The model classifies the entire text into the 'AI' or 'HUMAN' category using the representation of the special [CLS] token at the beginning of the input. The input format is kept as pre-training, where text is tokenized to 512 sequence length and passed to the model. We used WordPiece embedding for tokenization. Fine-tuning is a very resource-saving method. Using modern hardware such as GPU, we were able to train the model for our specific dataset in just 3 epochs. It provides improved results in much less time than traditional models.

DeBERTa

DeBERTa is a state-of-the-art language representation model that uses special techniques to overcome the limitations of BERT and RoBERTa. In our project we used the microsoft/deberta-v3-small checkpoint to further improve the accuracy of AI content detection. Its pre-training and fine-tuning process is discussed in detail below:

Pre-training DeBERTa

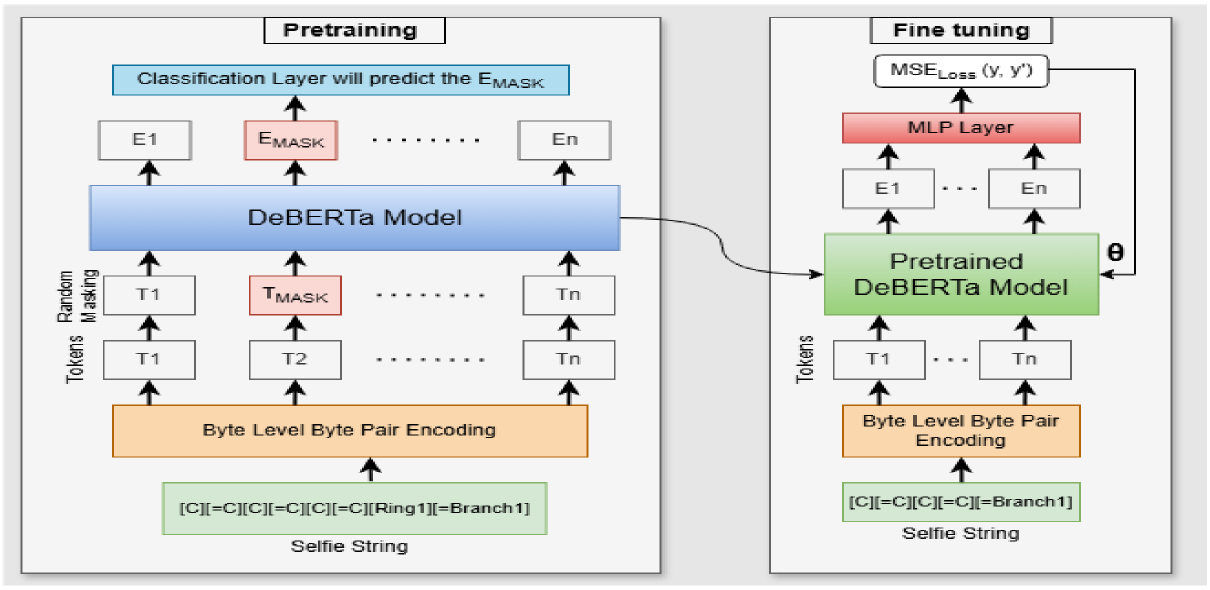


Fig4: Pre-training and Fine tuning of DeBERTa

DeBERTa's pre-training process is much more advanced than that of ordinary language models and is based on two main innovative approaches:

In traditional BERT models each token is represented as a vector where content and position are together. But in DeBERTa each token is represented by two separate vectors (content and relative position) 1 . This approach allows the model to more accurately understand distance relationships and logical sequences between words, helping to identify subtle patterns in AI-generated text.

Enhanced Masked Language Modeling (EMLM) Like BERT, it learns to predict masked tokens, but it can master more complex grammatical structures using word positional information.

For DeBERTa we have used Byte-Pair Encoding (BPE) tokenization method. It can process particularly rare words by breaking them into smaller units, which helps the model understand new types of words or phrases created by the AI. DeBERTa comes pre-trained from large text corpora (eg: Wikipedia and Common Crawl), enabling it to understand languages from different domains.

Fine-tuning DeBERTa

We performed a fine-tuning process to apply this vast knowledge base of pre-training to our specific task (AI Content Detection):

We loaded the DeBERTa model using AutoModelForSequenceClassification and defined 2 labels (HUMAN and AI) for binary classification. Each essay is tokenized to a maximum sequence length of 512 and dynamic padding is ensured using DataCollatorWithPadding. We prepared the dataset for training by removing the original text and generated columns.

According to our code, we trained the model for 3 epochs at a learning rate of 1×10^{-5}.The training batch size is kept at 8 and we use mixed precision or fp16=True for faster processing. During fine-tuning we routinely monitored Accuracy, Precision, Recall and F1-score through the compute_metrics function. DeBERTa's advanced positional encoding system helped us deliver extremely high accuracy in AI-generated content recognition in our dataset.

By using the load_best_model_at_end=True argument we ensure that only the most efficient or best version is saved at the end of training.

3.4 Model Evaluation

After completing model training, we followed a systematic evaluation procedure to verify its performance and accuracy. The main goal of this step is to mathematically confirm how reliable the model is in identifying AI-generated content. The main aspects of assessment are discussed below:

Evaluation Metrics: We used a custom compute_metrics function to measure model performance. With this function we regularly monitor the following four metrics:

Accuracy: This expresses the percentage of content in the validation dataset that the model has correctly identified as 'AI' or 'HUMAN'.

Precision: When the model identifies a text as 'AI', it indicates how accurate it actually was.

Recall: This shows how many of the real AI contents in the dataset were successfully found by the model.

F1-Score: It is a Harmonic Mean of Precision and Recall, which ensures the overall reliability of the model.

Validation Strategy: We used the eval_strategy="epoch" order, whereby the model was tested on a separate validation dataset at the end of each epoch. These data were never seen before during training of the model, which helped us to ensure accurate evaluation.

Confusion Matrix Analysis: We created a confusion matrix to see in detail where the model is going wrong. Through this we closely monitored the rate of False Positives (mistaking human writing for AI) and False Negatives (failing to identify AI writing).

Best Model Selection: Based on evaluation we used load_best_model_at_end=True argument. This ensures that the model with the lowest loss and highest performance in the validation set among the 3 training epochs is saved as the final one.

4 Implementation

This section describes the technical implementation details of the proposed AI Content Detection system, including the programming environment, libraries used, dataset handling process, model configuration, and training setup. The system was implemented using modern deep learning tools and transformer-based architectures to ensure reliable performance.

The complete implementation was carried out using the Python programming language and executed on Google Colab with GPU support, which significantly reduced training time for large transformer models.

Programming Environment and Libraries

- Programming Language: Python 3

- Platform: Google Colab (GPU enabled)
- Deep Learning Framework: PyTorch

The following libraries were used during implementation:

- transformers (Hugging Face) for pre-trained models and tokenizers
- datasets for dataset loading and processing
- torch for model training
- numpy and pandas for data handling
- scikit-learn for evaluation metrics
- matplotlib for result visualization

These libraries provided all necessary tools for model training, evaluation, and analysis.

Dataset Processing

The system was trained using a customized dataset named Training_Essay_Data.csv, which contains a total of 20,000 essays. The dataset includes two columns:

- text: the essay content
- generated: the label (0 for human-written text and 1 for AI-generated text)

The dataset is balanced, consisting of 10,000 human-written essays and 10,000 AI-generated essays. This balanced distribution helps reduce bias during training. Before training, the dataset was shuffled and split into training and validation sets.

Text Preprocessing and Tokenization

Text preprocessing was performed using Hugging Face tokenizers. All essays were converted into tokens that the models can process.

- Maximum sequence length: 512 tokens
- Padding: Enabled
- Truncation: Enabled

These steps ensured that all input sequences had a consistent length, which is required for efficient batch processing.

4.1 BERT Implementation

For the baseline model, BERT (bert-base-uncased) was used for binary text classification. Since the task is to distinguish between AI-generated and human-written text, the model was implemented using `AutoModelForSequenceClassification` with two output labels.

Model Configuration

- Model checkpoint: `bert-base-uncased`
- Tokenizer: BERT tokenizer
- Maximum sequence length: 512
- Number of labels: 2 (Human, AI)

Training Setup

- Epochs: 3
- Batch size: 8
- Learning rate: 1×10^{-5}
- Optimizer: AdamW
- Evaluation strategy: Evaluation after each epoch
- load_best_model_at_end: Enabled
- Mixed precision (fp16): Enabled

During training, performance metrics such as Accuracy, Precision, Recall, and F1-score were monitored. After fine-tuning, the BERT model achieved very strong performance, showing that it is effective for AI content detection.

4.2 DeBERTa Implementation

In addition to BERT, DeBERTa (microsoft/deberta-v3-small) was used as the second model. DeBERTa is an advanced transformer model that improves upon BERT by using a disentangled attention mechanism, allowing it to better capture semantic and positional relationships between words.

Model Configuration

- Model checkpoint: `microsoft/deberta-v3-small`
- Tokenizer: DeBERTa tokenizer
- Maximum sequence length: 512
- Number of labels: 2 (Human, AI)

Training Setup

The training configuration for DeBERTa was kept similar to BERT to ensure a fair comparison.

- Epochs: 3
- Batch size: 8
- Learning rate: 1×10^{-5}
- Optimizer: AdamW
- Evaluation strategy: Evaluation after each epoch
- load_best_model_at_end: Enabled
- Mixed precision (fp16): Enabled

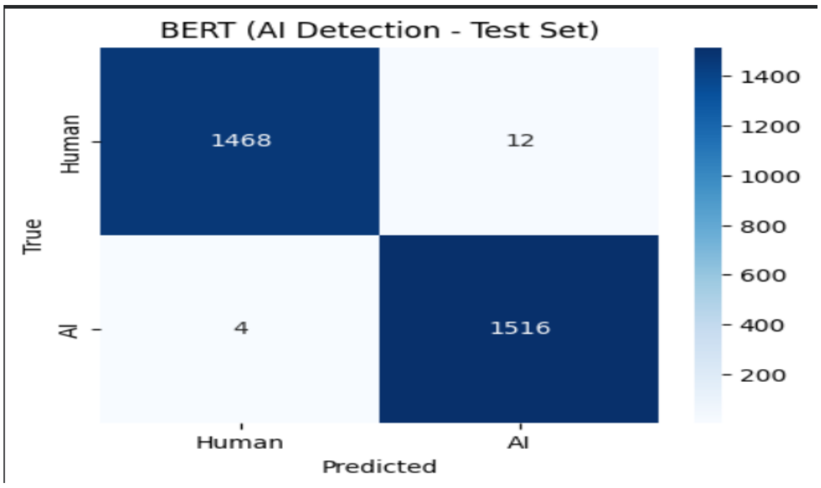
Due to its improved attention mechanism and positional encoding strategy, DeBERTa demonstrated slightly better and more stable performance compared to BERT in detecting AI-generated text.

5 Result Analysis:

This project uses two strong transformer-based deep learning models BERT and DeBERTa to solve the AI Content Detection issue. A balanced customized dataset (Training_Essay_Data.csv) is used to verify the performance of the models, which contains a total of 20,000 essays (10,000 human-written and 10,000 AI-generated). Four standard performance metrics, Accuracy, Precision, Recall, and F1-score, are used to evaluate the models.

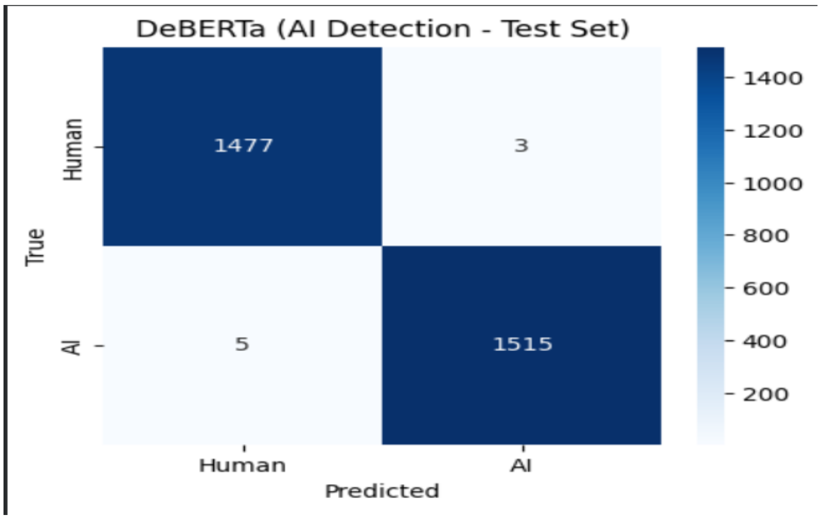
Performance of BERT Model:

After completing the fine-tuning, the BERT model has provided very satisfactory results. According to the test results, the BERT model achieved 0.9947 accuracy, 0.9921 precision, 0.9974 recall, and 0.9948 F1-score. The high recall value indicates that the model is very good at correctly identifying AI-generated texts, meaning that the rate of AI texts being incorrectly identified as human is very low. At the same time, the high precision indicates that most of the text that the model identifies as AI is actually AI-generated.



Performance of DeBERTa Model:

The DeBERTa model has pointed out better performance than BERT. The test results show that the DeBERTa model achieves 0.9973 accuracy, 0.9980 precision, 0.9967 recall, and 0.9974 F1-score. These results indicate that the DeBERTa model is more proper and consistent in prediction. Its high precision and F1-score prove that the model has successfully identified subtle differences between AI and human-written text.



Comparative Analysis between BERT and DeBERTa:

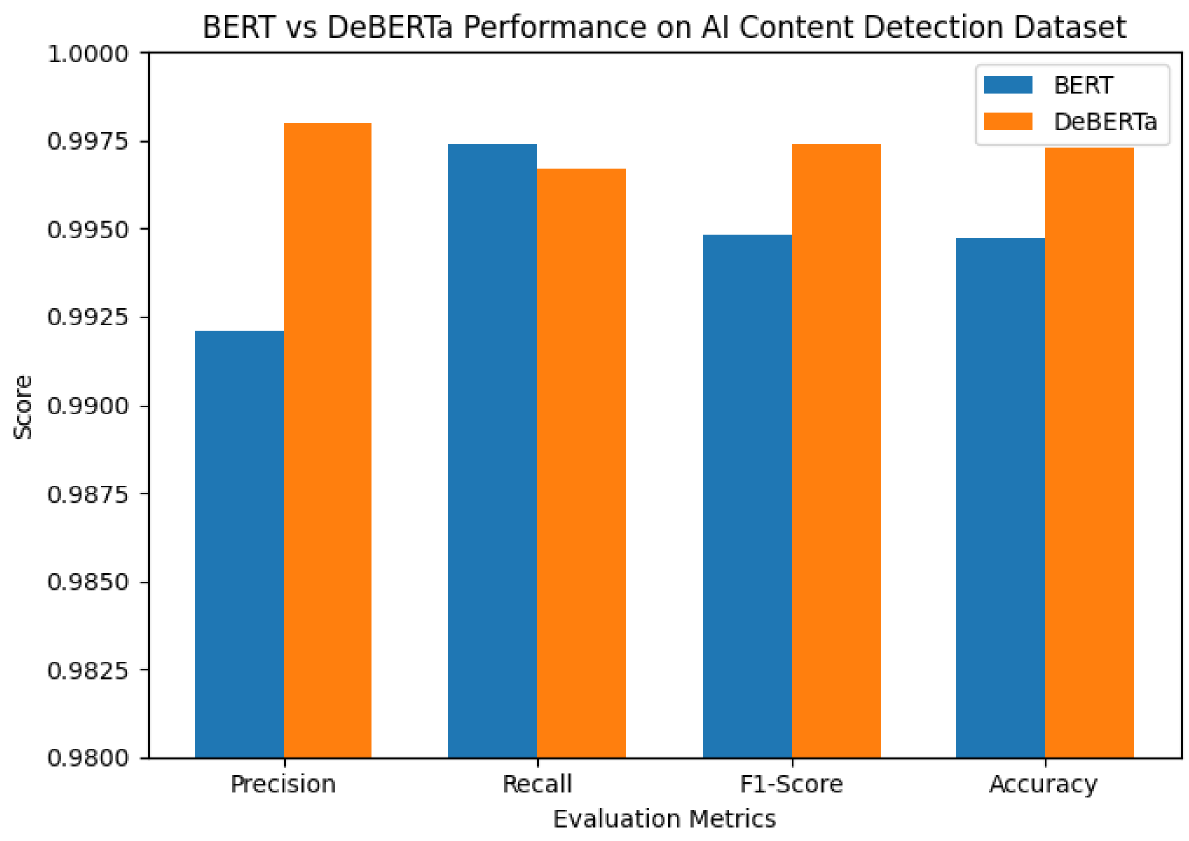


Fig5: BERT vs DeBERTa Performance

A comparative analysis of the two models shows that although both models achieve near-perfect performance, DeBERTa provides little bit better results than BERT in all evaluation metrics. The main reason for DeBERTa's better performance is its disentangled attention mechanism, where the semantic meaning and positional information of words are considered separately.This allows for a deep analysis of sentence structure,word dependency,and contextual patterns, which is particularly useful for AI-generated text recognition.

Model	Precision	Recall	F1 Score	Accuracy
BERT	0.9921	0.9974	0.9948	0.9947
DeBERTa	0.9980	0.9967	0.9974	0.9973

Overall Discussion:

Overall,the experimental results demonstrate that AI content detection using a transformer-based model is a very successful and dependable method.Both models reach very high accuracy due to the use of a balanced dataset and a proper fine-tuning strategy.However,in practical applications,the DeBERTa model gives a comparisonly more accurate and stable solution.

6 Conclusion:

In this project,we have developed an effective AI Content Detection system that is capable of accurately recognizing human-written and AI-generated text.With the world wide use of modern Large Language Models (LLMs) like ChatGPT, AI content detection has become an important research problem, and this project provides a practical solution to that problem.In this study, BERT and DeBERTa models are fine-tuned on a balanced dataset (20,000 essays) using Transfer Learning. The experimental results show that both models achieve high accuracy, but the DeBERTa model provides relatively better and more stable performance, which is a result of its improved attention mechanism and positional encoding.

However, this project has some limitations. The dataset used consists only of English essay-based texts, so the performance may differ for other languages or different types of content. In addition, being a transformer-based model, this system requires relatively more computational resources.Overall, this project demonstrates that using a transformer-based deep learning model to detect AI-generated content is a reliable and effective method. This work can be further improved in the future by including more diverse datasets and new types of AI-generated content.

References

[1] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol.*, vol. 1, Jun. 2019, pp. 4171–4186.

[2] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with disentangled attention," in Proc. 9th Int. Conf. Learn. Represent. (ICLR), Oct. 2021, pp. 1–23.

Project Code

Load Python Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from datasets import Dataset, DatasetDict
!pip install evaluate
import evaluate

from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, confusion_matrix

import torch

print("All libraries loaded successfully.")
```

```
Collecting evaluate
  Downloading evaluate-0.4.6-py3-none-any.whl.metadata (9.5 kB)
Requirement already satisfied: datasets>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from evaluate) (4.0.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from evaluate) (2.0.2)
Requirement already satisfied: dill in /usr/local/lib/python3.12/dist-packages (from evaluate) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from evaluate) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.12/dist-packages (from evaluate) (2.32.4)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.12/dist-packages (from evaluate) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from evaluate) (3.6.0)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.12/dist-packages (from evaluate) (0.70.16)
Requirement already satisfied: fsspec>=2021.05.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]>=2021.05.0->evaluate) (2025.3.0)
Requirement already satisfied: huggingface-hub>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from evaluate) (0.36.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from evaluate) (25.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets>=2.0.0->evaluate) (3.20.2)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets>=2.0.0->evaluate) (18.1.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets>=2.0.0->evaluate) (6.0.3)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]>=2021.05.0->evaluate) (3.13.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.7.0->evaluate) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.7.0->evaluate) (1.2.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (2026.1.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->evaluate) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->evaluate) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->evaluate) (2025.3)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05.0->eva
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05.0->evalua
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05.0->ev
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05.0->
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05.0->eva
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05.0->ev
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->evaluate) (1.17.0)
Downloading evaluate-0.4.6-py3-none-any.whl (84 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 5.8 MB/s eta 0:00:00
Installing collected packages: evaluate
Successfully installed evaluate-0.4.6
All libraries loaded successfully.
```

Load DataSet

```
file_path = 'Training_Essay_Data.csv'
df = pd.read_csv(file_path)

import re
def clean_text(text):
    if pd.isna(text): return ""
    return re.sub(r'\s+', ' ', text).strip()

df['text'] = df['text'].apply(clean_text)
df = df[df['text'] != ""]

full_dataset = Dataset.from_pandas(df)

train_test_split_dataset = full_dataset.train_test_split(test_size=0.3, seed=42)

remaining_dataset = train_test_split_dataset['test'].train_test_split(test_size=0.5, seed=42)

dataset_dict = DatasetDict({
    'train': train_test_split_dataset['train'],
    'valid': remaining_dataset['train'],
    'test': remaining_dataset['test']
})
```

```
print("Dataset Split: 70% Train, 15% Valid, 15% Test - Done.")
print(dataset_dict)
```

```
Dataset Split: 70% Train, 15% Valid, 15% Test - Done.
DatasetDict({
    train: Dataset({
        features: ['text', 'generated'],
        num_rows: 14000
    })
    valid: Dataset({
        features: ['text', 'generated'],
        num_rows: 3000
    })
    test: Dataset({
        features: ['text', 'generated'],
        num_rows: 3000
    })
})
```

Tokenization

```
bert_checkpoint = "bert-base-uncased"
deberta_checkpoint = "microsoft/deberta-v3-small"

tokenizer_bert = AutoTokenizer.from_pretrained(bert_checkpoint)
tokenizer_deberta = AutoTokenizer.from_pretrained(deberta_checkpoint)

def preprocess_bert(examples):
    return tokenizer_bert(examples["text"], truncation=True, padding=True, max_length=512)

def preprocess_deberta(examples):
    return tokenizer_deberta(examples["text"], truncation=True, padding=True, max_length=512)

print("Tokenizing for BERT...")
tokenized_bert = dataset_dict.map(preprocess_bert, batched=True)

print("Tokenizing for DeBERTa...")
tokenized_deberta = dataset_dict.map(preprocess_deberta, batched=True)

tokenized_bert = tokenized_bert.remove_columns(["text"])
tokenized_deberta = tokenized_deberta.remove_columns(["text"])

print("Tokenizations 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/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(
tokenizer_config.json: 100%                                  48.0/48.0  [00:00<00:00,  5.20kB/s]
config.json:  100%                                  570/570  [00:00<00:00,  56.0kB/s]
vocab.txt:  100%                                  232k/232k  [00:00<00:00,  1.44MB/s]
tokenizer.json:  100%                                  466k/466k  [00:00<00:00,  1.99MB/s]
tokenizer_config.json:  100%                                  52.0/52.0  [00:00<00:00,  6.16kB/s]
config.json:  100%                                  578/578  [00:00<00:00,  66.1kB/s]
spm.model:  100%                                  2.46M/2.46M  [00:01<00:00,  2.88MB/s]
/usr/local/lib/python3.12/dist-packages/transformers/convert_slow_tokenizer.py:566: UserWarning: The sentencepiece tokenizer that you are converting to a
  warnings.warn(
Tokenizing for BERT...
Map:  100%                                  14000/14000  [00:41<00:00,  278.93  examples/s]
Map:  100%                                  3000/3000  [00:13<00:00,  222.93  examples/s]
Map:  100%                                  3000/3000  [00:14<00:00,  205.21  examples/s]
Tokenizing for DeBERTa...
Map:  100%                                  14000/14000  [00:47<00:00,  378.00  examples/s]
Map:  100%                                  3000/3000  [00:07<00:00,  438.15  examples/s]
Map:  100%                                  3000/3000  [00:09<00:00,  308.37  examples/s]
Tokenizations are ready.
```

```
def preprocess_bert(examples):
    # Text-ke token-e rupantor
    result = tokenizer_bert(examples["text"], truncation=True, padding=True, max_length=512)

    result["labels"] = examples["generated"]
    return result

print("Tokenizing with labels, please wait...")
tokenized_bert = dataset_dict.map(preprocess_bert, batched=True)
tokenized_bert = tokenized_bert.remove_columns(["text", "generated"])

print("Now the dataset is ready.")
```

```
Tokenizing with labels, please wait...
Map: 100%                                   14000/14000 [00:28<00:00, 689.80 examples/s]
Map: 100%                                   3000/3000 [00:05<00:00, 606.44 examples/s]
Map: 100%                                   3000/3000 [00:04<00:00, 741.78 examples/s]
Now the dataset is ready.
```

BERT

```
id2label = {0: "HUMAN", 1: "AI"}
label2id = {"HUMAN": 0, "AI": 1}

model_bert = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=2,
    id2label=id2label,
    label2id=label2id
)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)

    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average='binary')
    acc = accuracy_score(labels, predictions)

    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }

print("BERT Model Setup Completed.")
```

```
model.safetensors:  100%                                   440M/440M  [00:05<00:00,  72.3MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
BERT Model Setup Completed.
```

BERT Training Arguments

```
data_collator = DataCollatorWithPadding(tokenizer=tokenizer_bert)

training_args = TrainingArguments(
    output_dir="./bert-ai-detection",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    load_best_model_at_end=True,
    logging_steps=100,
    report_to="none"
)

trainer_bert = Trainer(
    model=model_bert,
    args=training_args,
    train_dataset=tokenized_bert["train"],
    eval_dataset=tokenized_bert["valid"],
    tokenizer=tokenizer_bert,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

print("Training Arguments setup ready.")
```

```
Training Arguments setup ready.
/tmp/ipython-input-3102388312.py:21: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processin
  trainer_bert = Trainer(
```

```
trainer_bert.train()
```

[2494/2625 1:09:24 < 03:38, 0.60 it/s, Epoch 2.85/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.019700	0.021428	0.995000	0.995012	0.990073	1.000000
2	0.003300	0.144061	0.978000	0.978417	0.957746	1.000000

[2625/2625 1:15:16, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.019700	0.021428	0.995000	0.995012	0.990073	1.000000
2	0.003300	0.144061	0.978000	0.978417	0.957746	1.000000
3	0.002600	0.035411	0.995000	0.995012	0.990073	1.000000

```
TrainOutput(global_step=2625, training_loss=0.020190554152309362, metrics={'train_runtime': 4519.3226, 'train_samples_per_second': 9.293,
'train_steps_per_second': 0.581, 'total_flos': 1.105066432512e+16, 'train_loss': 0.020190554152309362, 'epoch': 3.0})
```

```
print("calculating metrics for BERT on Test Set...")
bert_results = trainer_bert.evaluate(tokenized_bert["test"])

print("\n" + "="*30)
print("BERT PERFORMANCE REPORT")
print("="*30)
print(f"Accuracy  : {bert_results['eval_accuracy']:.4f}")
print(f"Precision : {bert_results['eval_precision']:.4f}")
print(f"Recall    : {bert_results['eval_recall']:.4f}")
print(f"F1 Score  : {bert_results['eval_f1']:.4f}")
print("="*30)
```

```
calculating metrics for BERT on Test Set...
```

[188/188 03:50]

```
==============================
BERT PERFORMANCE REPORT
==============================
Accuracy  : 0.9947
Precision : 0.9921
Recall    : 0.9974
F1 Score  : 0.9948
==============================
```

DeBERTa

```
deberta_checkpoint = "microsoft/deberta-v3-small"
tokenizer_deberta = AutoTokenizer.from_pretrained(deberta_checkpoint)

def preprocess_deberta(examples):
    result = tokenizer_deberta(examples["text"], truncation=True, padding=True, max_length=512)
    result["labels"] = examples["generated"]
    return result

print("Tokenizing for DeBERTa with labels...")
tokenized_deberta = dataset_dict.map(preprocess_deberta, batched=True)

tokenized_deberta = tokenized_deberta.remove_columns(["text", "generated"])

print("DeBERTa Tokenization ready with labels.")
```

```
/usr/local/lib/python3.12/dist-packages/transformers/convert_slow_tokenizer.py:566: UserWarning: The sentencepiece tokenizer that you are converting to a
  warnings.warn(
Tokenizing for DeBERTa with labels...
Map: 100%                                   14000/14000  [00:27<00:00,  606.85  examples/s]
Map: 100%                                   3000/3000  [00:03<00:00,  867.54  examples/s]
Map: 100%                                   3000/3000  [00:04<00:00,  649.26  examples/s]
DeBERTa Tokenization ready with labels.
```

```
model_deberta = AutoModelForSequenceClassification.from_pretrained(
    deberta_checkpoint,
    num_labels=2,
    id2label=id2label,
    label2id=label2id
)

data_collator_deberta = DataCollatorWithPadding(tokenizer=tokenizer_deberta)

training_args_deberta = TrainingArguments(
    output_dir="./deberta-ai-detection",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=1e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    load_best_model_at_end=True,
    logging_steps=100,
    fp16=True,
    report_to="none"
)

trainer_deberta = Trainer(
    model=model_deberta,
    args=training_args_deberta,
    train_dataset=tokenized_deberta["train"],
    eval_dataset=tokenized_deberta["valid"],
    tokenizer=tokenizer_deberta,
    data_collator=data_collator_deberta,
    compute_metrics=compute_metrics,
)

print("DeBERTa Setup ready.")
```

```
Some weights of DebertaV2ForSequenceClassification were not initialized from the model checkpoint at microsoft/deberta-v3-small 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.
/tmp/ipython-input-1208038812.py:29: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processin
  trainer_deberta = Trainer(
DeBERTa Setup ready.
```

```
trainer_deberta.train()
```

```
The tokenizer has new PAD/BOS/EOS tokens that differ from the model config and generation config. The model config and generation config were aligned acc
```

[5250/5250 35:16, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.021500	0.028116	0.995000	0.994992	0.993996	0.995989
2	0.001500	0.035375	0.995333	0.995336	0.992032	0.998663
3	0.008700	0.035232	0.994667	0.994677	0.990066	0.999332

```
TrainOutput(global_step=5250, training_loss=0.01997801638411225, metrics={'train_runtime': 2118.9282, 'train_samples_per_second': 19.821,
'train_steps_per_second': 2.478, 'total_flos': 5563828924416000.0, 'train_loss': 0.01997801638411225, 'epoch': 3.0})
```

```
print("Calculating metrics for DeBERTa on Test Set...")
deberta_results = trainer_deberta.evaluate(tokenized_deberta["test"])

print("\n" + "="*30)
print("DeBERTa PERFORMANCE REPORT")
print("="*30)
print(f"Accuracy  : {deberta_results['eval_accuracy']:.4f}")
print(f"Precision : {deberta_results['eval_precision']:.4f}")
print(f"Recall    : {deberta_results['eval_recall']:.4f}")
print(f"F1 Score  : {deberta_results['eval_f1']:.4f}")
print("="*30)
```

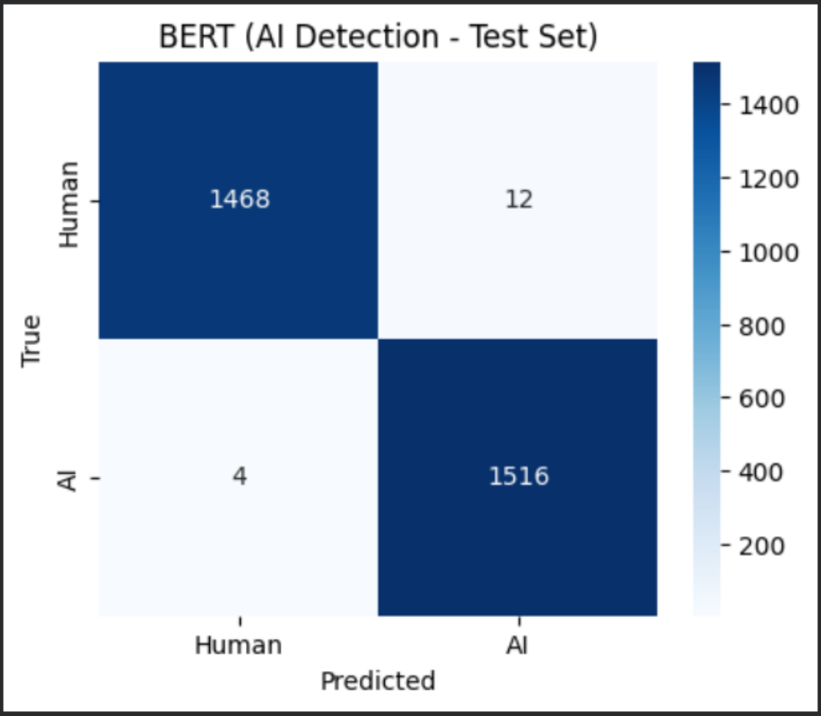
```
Calculating metrics for DeBERTa on Test Set...
[375/375 00:34]

==============================
DeBERTa PERFORMANCE REPORT
==============================
Accuracy  : 0.9973
Precision : 0.9980
Recall    : 0.9967
F1 Score  : 0.9974
==============================
```

Comparison

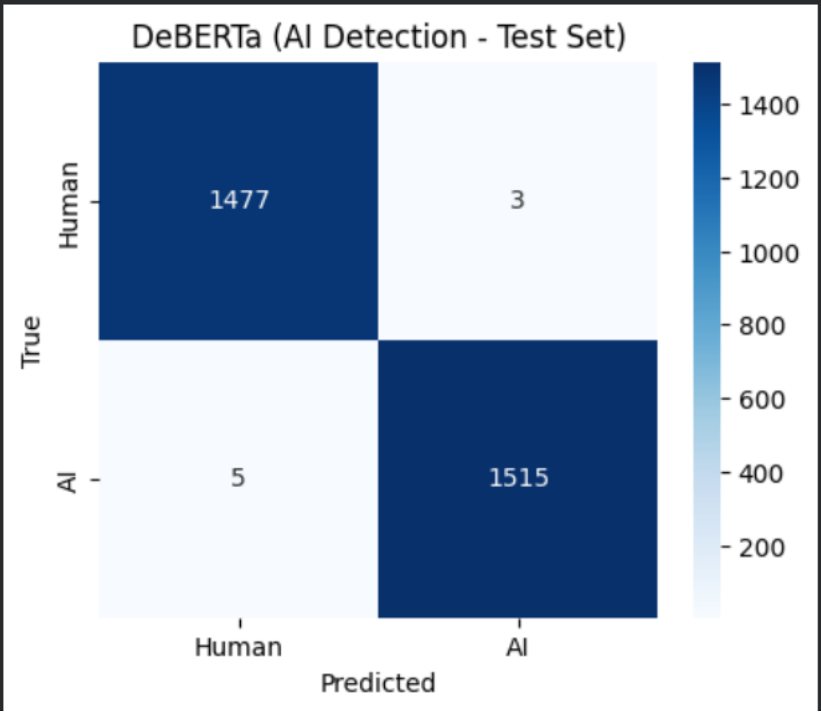
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np
import pandas as pd

summary = {
    "AI_Detection": {
        "bert": {
            "precision": bert_results['eval_precision'],
            "recall": bert_results['eval_recall'],
            "f1": bert_results['eval_f1'],
            "accuracy": bert_results['eval_accuracy']
        },
        "deberta": {
            "precision": deberta_results['eval_precision'],
            "recall": deberta_results['eval_recall'],
            "f1": deberta_results['eval_f1'],
            "accuracy": deberta_results['eval_accuracy']
        }
    }
}

def get_predictions(trainer, tokenized_dataset):
    preds = trainer.predict(tokenized_dataset)
    y_pred = np.argmax(preds.predictions, axis=-1)
    y_true = preds.label_ids
    return y_true, y_pred
```

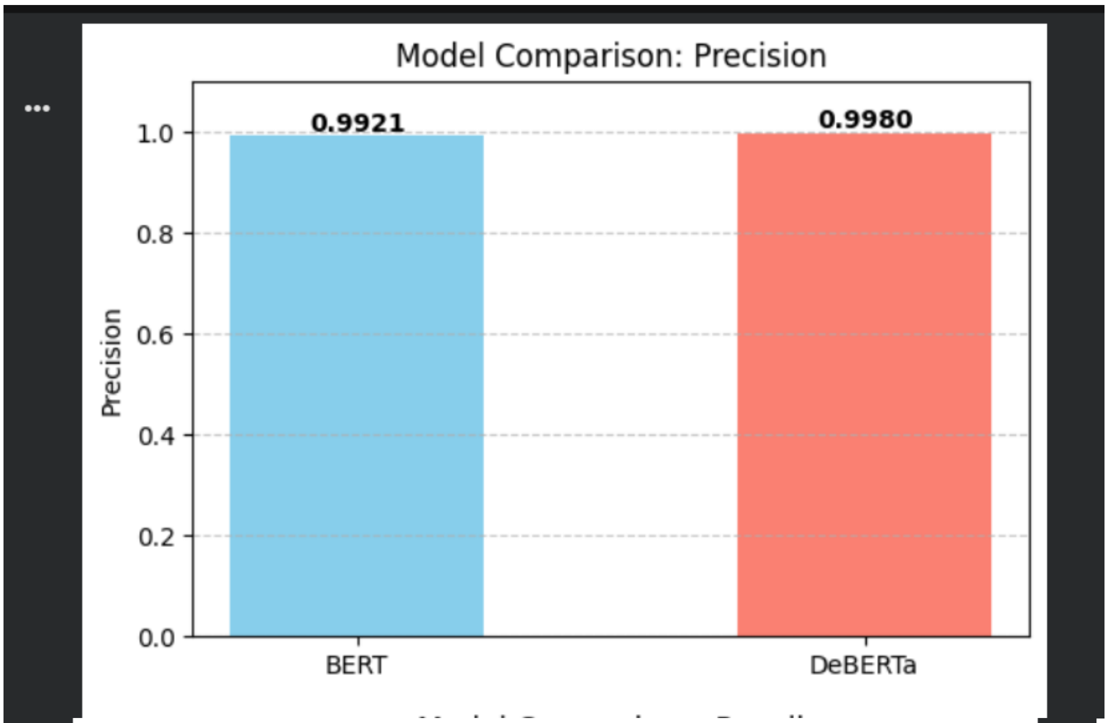
```
print("Fetching predictions for Confusion Matrix...")
y_true_bert, y_pred_bert = get_predictions(trainer_bert, tokenized_bert["test"])
y_true_deb, y_pred_deb = get_predictions(trainer_deberta, tokenized_deberta["test"])

def plot_confusion(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Human', 'AI'], yticklabels=['Human', 'AI'])
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()

print("\n--- Confusion Matrix: BERT ---")
plot_confusion(y_true_bert, y_pred_bert, "BERT (AI Detection - Test Set)")

print("\n--- Confusion Matrix: DeBERTa ---")
plot_confusion(y_true_deb, y_pred_deb, "DeBERTa (AI Detection - Test Set)")

metrics = ["precision", "recall", "f1", "accuracy"]
model_names = ["BERT", "DeBERTa"]

for metric in metrics:
    plt.figure(figsize=(6,4))

    values = [summary["AI_Detection"]["bert"][metric],
              summary["AI_Detection"]["deberta"][metric]]

    colors = ['skyblue', 'salmon']
    plt.bar(model_names, values, color=colors, width=0.5)

    for i, v in enumerate(values):
        plt.text(i, v + 0.01, f"{v:.4f}", ha='center', fontweight='bold')

    plt.ylim(0, 1.1)
    plt.ylabel(metric.capitalize())
    plt.title(f"Model Comparison: {metric.capitalize()}")
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```

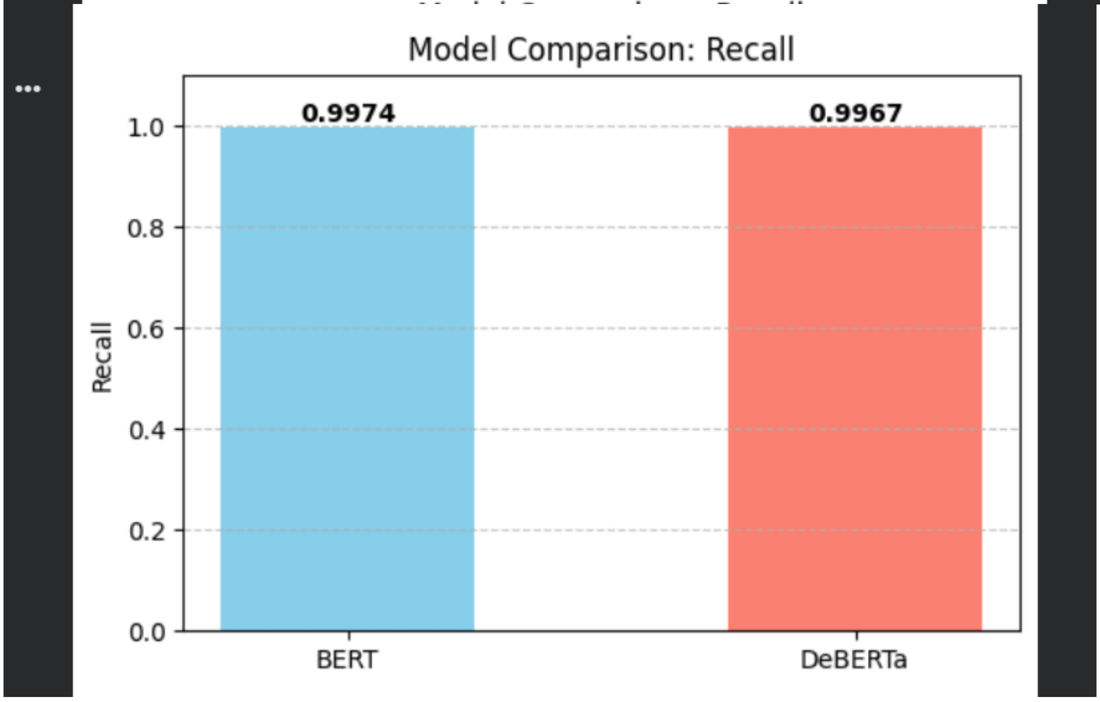
```
--- Confusion Matrix: BERT ---
```

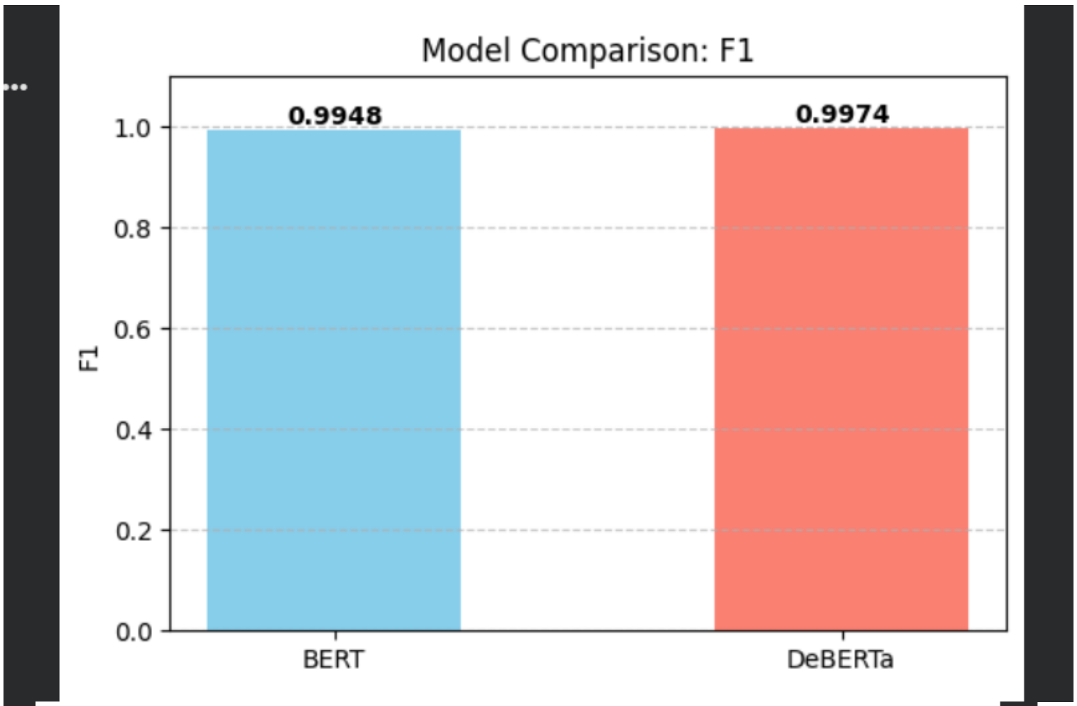


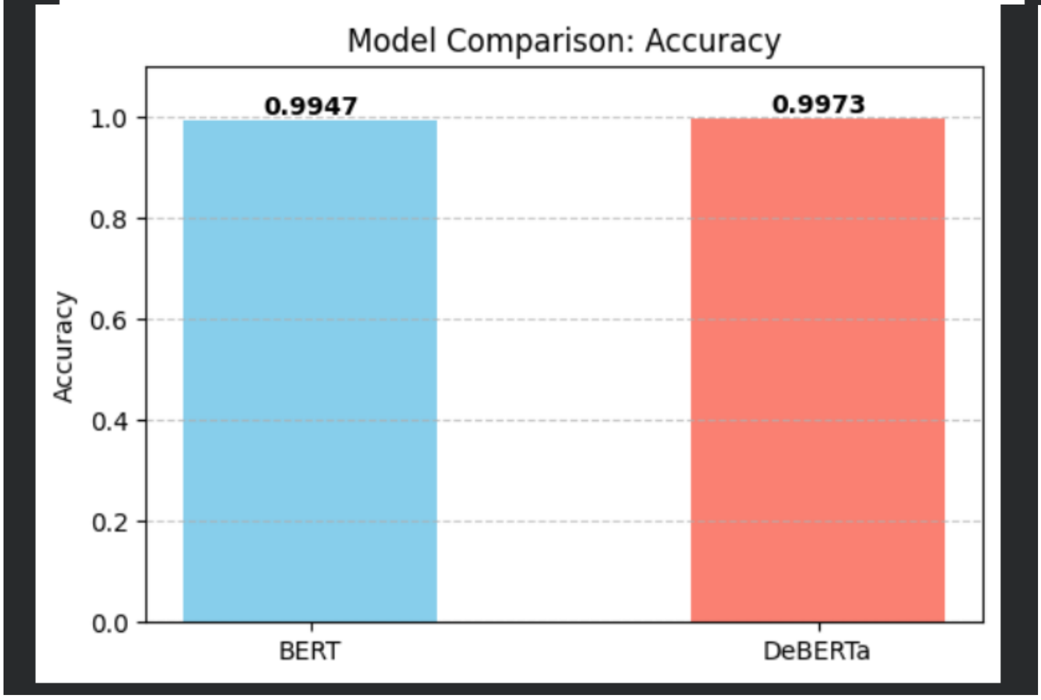
```
--- Confusion Matrix: DeBERTa ---
```




Model Comparison: Precision
Precision
0.9921
0.9980
1.0
0.8
0.6
0.4
0.2
0.0
BERT
DeBERTa


Model Comparison: Recall
Recall
0.9974
0.9967
1.0
0.8
0.6
0.4
0.2
0.0
BERT
DeBERTa


Model Comparison: F1
F1
0.9948
0.9974
1.0
0.8
0.6
0.4
0.2
0.0
BERT
DeBERTa


Model Comparison: Accuracy
Accuracy
0.9947
0.9973
1.0
0.8
0.6
0.4
0.2
0.0
BERT
DeBERTa