1. Introduction

This project explores a variety of machine-learning algorithms, including traditional ML and deep learning models, to distinguish between machine and human-generated text.

2. Domain Difference

We have two training data: one coming from domain1, and another coming from domain2.

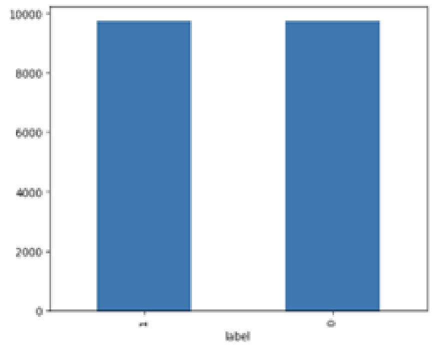

Figure 1 Domain1

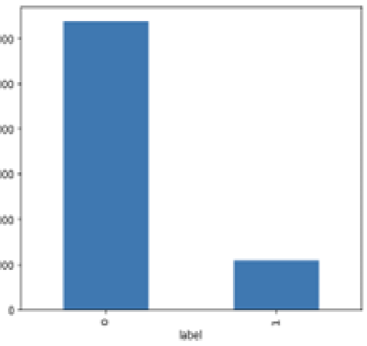

Figure 2 Domain2

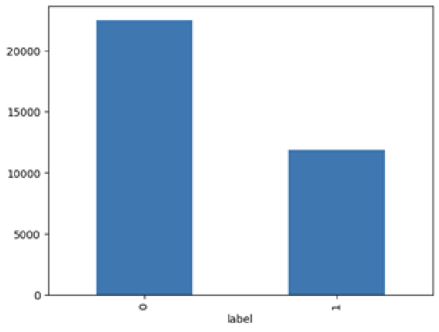

Figure 3 Combined Data

We can easily find that domain1 has balanced data, but domain2(middle) does not. But if we only choose domain1 as our training data, we may get an overfitting model due to the insufficient data, so we'd better combine these two datasets.
But the label1 and label0 are still imbalanced. If we want to train a good model for the binary classification, we'd better make the training data balanced. For this reason, we decided to use **Weighted Cross Entropy** as the way we process the data. So that we can give more importance to the minority class (i.e., it penalizes misclassifications of the minority class more severely) during training. In addition, we also let 20% of the data be the testing set to validate our model accuracy.
Here is our training data plot(left), we can easily find that the training data has a large proportion of text length which is less than 50, and they are balanced(right). In that case, we decided to divide the model into 3 parts as our final model.

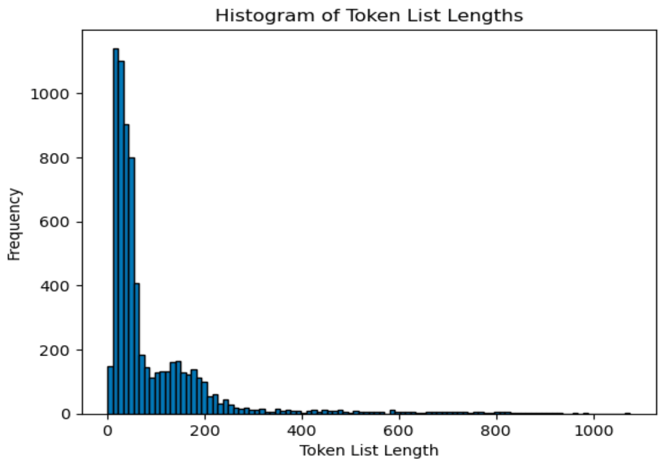

Figure 4 Training Data

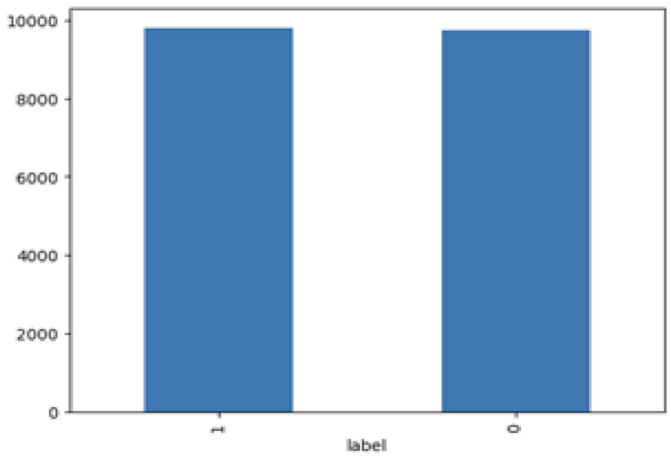

Figure 5 <50 Data

3. Final Approach

3.1 Data Preprocessing

As we mentioned before. We first combine the Domain 1 and Domain 2 datasets together. After observation, we discovered that the data is imbalanced based on the token lengths as well. Therefore, we split the data into below 3 segments based on the token lengths. Then, we train and fine-tune 3 different models to predict the data based on the token lengths.

- <50 tokens
- 50 - 250 tokens
- 250-1075 tokens

3.2 Model Architecture

After comparing several model architectures, we decided to select Transformer Encoder as our final to classify the data. Transformers with self-attention mechanisms recently performed well on NLP tasks such as text generation, translation, and classification. Therefore, we experimented Transformer encoder in the assignment and concluded that it performs the best in terms of accuracy among all other model architectures, which will be discussed in the next section. However, we didn't reference the LLM architectures such as GPT and LLAMA, which are trained on massive text data. As we were given a small dataset, if we use LLM architectures, the model will overfit the data easily. Instead, we designed a simple and small transformer encoder to handle the classification problem and avoid overfitting.

Our final approach consists of 3 models handling 3 types of token lengths.

- **<50 token length:**
  Embedding layer: 16-dimension
  PositionalEncoding layer
  Layer Normalization layer
  Transformer Encoding layer: 2 layers, 32-hidden-dimension, 2 attention heads
  Average pooling layer
  Fully connected layer
- **50-250 token length:**
  Embedding layer: 16-dimension
  PositionalEncoding layer
  Layer Normalization layer
  Transformer Encoding layer: 2 layers, 16-hidden-dimension, 2 attention heads
  Average pooling layer
  Fully connected layer
- **250-1075 token length:**
  Embedding layer: 32-dimension
  PositionalEncoding layer
  Layer Normalization layer
  Transformer Encoding layer: 2 layers, 32-hidden-dimension,42 attention heads
  Average pooling layer
  Fully connected layer

We decided to split into 3 models because training a long sequence model is difficult, and requires massive training data. We tried to use a single transformer model to handle all the data, i.e. maximum sequence length is 1075, but the accuracy was unsatisfactory due to the long sequence length. Therefore, we split the data into 3 segments, and the <50 token model performs better with a validation accuracy of 90%+ for each class.

4. Other Alternatives

4.1 Logistic Regression

Logistic regression is designed for binary classification, and we need to predict if the text is written by a machine or a human. We first chose the logistic regression as our training model, but there are some disadvantages to this model, so we shortly decided not to proceed:

- Imbalanced data from domain1 and domain2.
- If we only choose domain1 for our training data (which is balanced), it doesn't have enough data and may lead to overfitting.
- The model may have a valid accuracy that is not too bad (like 0.7), but when we put the data from the test set to our model and then put the result on Kaggle, its score is bad (like 0.6).

4.2 MLP (Multi-Layer Perceptrons)

Our next attempted method was MLP. However, we faced overfitting issues. To address this, we used multiple techniques such as Batch Normalization, Dropout, Early Stopping, and reducing the number of layers and nodes. Additionally, we incorporated an Embedding layer to capture the semantic relationships between words. Despite these efforts, we were unable to completely eliminate overfitting, and our accuracy on Kaggle was stuck at 0.72.

4.3 LSTM (Long Short-Term Memory)

Considering the inherent sequential structure present in the text data, we used LSTM networks to capture long-term dependencies within the series, with the hope that it might bolster model performance. After hyperparameter tuning, we observed improvements in accuracy and loss on both the training and test datasets. However, the model's performance on Kaggle remained suboptimal.

4.4 Evaluation Metrics

To evaluate the model performance, we use the confusion matrix, F1-score, and computational efficiency.
Confusion matrix: As the data is imbalanced, it is inappropriate to use accuracy to evaluate the performance because the model might misclassify all the minor class data but with high accuracy. By using the confusion matrix, we can calculate the accuracy of each class and choose a model that has balanced accuracy in each class.
Weighted F1-score: The weighted F1-score takes the precision, recall, and data imbalance into account. We can get a brief idea of the average performance of the model.
Computational efficiency: Accuracy is important, but in real-world scenarios, we also have to consider the cost versus accuracy. For example, LLM gets the best accuracy, but it requires massive computational resources, so sometimes we might need to trade-off between accuracy and computational efficiency.

The below table shows the metrics of 4 different models.

	Class 0 Validation Accuracy	Class 1 Validation Accuracy	Weighted F1 Score	Computational Efficiency
Transformer	83	86	82	Low
Logistic Regression	72	68	71	High
MLP	78	72	74	Medium
LSTM	71	70	69	Low

Since our goal is to consider the accuracy of the test dataset, we selected the Transformer model architecture as our final model based on the evaluation metrics.