Technical Documentation:

Classifying Essays into LLM Generated or Human Generated Texts

Course:
Deep Learning I

Group Members:
Dabas, Rashika
Dimaculangan, Roan Kathrina
Ip, Chak Leung Vincent
Suresh, Priyanka Yadhav
Muthusamy, Karthikeyan

TABLE OF CONTENTS

I. Background and Problem Statement

Background

The rapid advancement and integration of Large Language Models (LLMs) such as GPT-3 into various areas have resulted to significant shifts, particularly in academic settings and the broader domain of information dissemination.

In the area of education, the growing reliance on AI for academic tasks has emerged as a double-edged sword. While LLMs offer exceptional convenience in content creation, they simultaneously pose risks to the foundational pillars of educational values. Critical thinking, creativity, and original research are increasingly under threat as AI-generated content becomes more accessible and sophisticated. This trend, as discussed by Michael Perkins in his 2023 article in the Journal of University Teaching & Learning Practice, leads to a diminished student engagement with learning materials, potentially degrading the quality of learning and hindering the development of crucial academic skills. Moreover, the shift demands a redefinition of academic plagiarism in the AI era to establish new academic frameworks and policies.

Parallelly, in the domain of information dissemination, the impact of LLMs, underscored by the Center for Security and Emerging Technology, raises alarms in the digital age. The ability of LLMs like GPT-3 to produce highly articulate and persuasive content has significant implications for the spread of misinformation. The realistic nature of AI-generated content can be exploited in disinformation campaigns, as evidenced in the study "Truth, Lies, and Automation" by the Center for Security and Emerging Technology. This study revealed GPT-3's effectiveness in crafting narratives, such as climate change denial, and its proficiency in complex tasks like rewriting news articles to sway public opinion. These capabilities of LLMs underscore the potential risks in scenarios where misinformation could have profound social, political, or economic repercussions, necessitating robust countermeasures and strategies for content verification.

Problem Definition:

This study aims to address the crucial task of distinguishing between texts authored by humans and those generated by LLMs. In an era where LLMs like GPT-3 have demonstrated the ability to produce text that is strikingly realistic and coherent, the need for accurate classification has never been more important. This challenge is particularly evident in academic settings, where the integrity of academic work is at risk due to the ease of AI-assisted content creation, and in the broader context of media and information, where the rapid dissemination of AI-generated misinformation could have widespread social, political, and economic impacts.

II. Strategy for Implementation

For our deep learning project – classifying whether essays are generated by Large Language Models (LLMs) or by human, we're embarking on a strategic approach, beginning with the essential task of accumulating a substantial and diverse dataset for training. This foundational step is crucial as it ensures our models are trained on a rich variety of textual inputs, enhancing their ability to discern subtle differences between machine-generated and human-written texts.

Following this, our team plans to meticulously prepare the data through processes such as tokenization, padding, and vectorization. Tokenization will break the essays into smaller, manageable units, padding will standardize these units to a uniform length, and vectorization will transform these textual elements into numerical data. These steps are integral for uncovering the nuanced meanings and contexts embedded within the essays, ensuring our models can effectively interpret and learn from the data.

We will then proceed to train three distinct types of neural network models: RNN, LSTM, and FNN. This phase will involve a detailed analysis of each model's learning efficiency, resource requirements, training duration, and accuracy in classification. The insights gained from this evaluation will guide us in selecting the most suitable model that excels in identifying the origin of the essays with precision.

Incorporating the best-performing model into a user-friendly interface will be the final step in our strategy. This integration aims to enhance the user experience, allowing for an intuitive interaction with our classification system. By making the technology accessible through a well-designed UI, users can effortlessly utilize our solution to determine whether essays are LLM-generated or human-written.

Adopting this comprehensive approach ensures that no critical aspect of the essays is overlooked. By securing a diverse dataset, applying thorough data preparation techniques, evaluating multiple models, and prioritizing user experience in the application of our findings, we position ourselves to effectively address the challenge at hand, optimizing our use of time and resources throughout the project.

III. Overview and Exploratory Analysis of the Dataset

Dataset Source

In our project aimed at distinguishing between AI-generated and human-written texts, we leverage a diverse array of datasets sourced from Kaggle. Each dataset plays a crucial role in refining our model's accuracy and reliability:

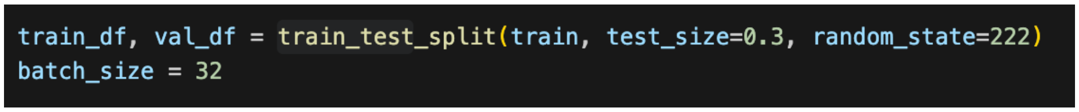
Table 1 Dataset Sources and Description

No.	Kaggle Dataset	Description
1	/kaggle/input/daigt-external-dataset/daigt_external_dataset.csv	A curated collection designed to differentiate between AI-generated and human-authored texts, complete with origin labels for each entry.
2	/kaggle/input/llm-mistral-7b-instruct-texts/Mistral7B_CME_v7.csv	Essays and texts produced by the Mistral 7B model, offering insights into the capabilities of this particular LLM.
3	/kaggle/input/hello-claude-1000-essays-from-anthropic/persuade15_claude_instant1.csv	A compilation of persuasive essays from Anthropic's Claude model, aimed at various topics to analyze argumentative capabilities.
4	/kaggle/input/daigt-data-llama-70b-and-falcon180b/llama_falcon_v3.csv	This dataset merges outputs from both the LLaMA and Falcon models, providing a rich basis for comparative analysis and detection training.
5	/kaggle/input/llm-generated-essay-using-palm-from-google-gen-ai/LLM_generated_essay_PaLM.csv	Showcases essays by Google's PaLM, highlighting its proficiency in generating contextually rich and coherent texts.
6	/kaggle/input/daigt-v2-train-dataset/train_v2_drcat_02.csv	The second version of our training dataset, containing labeled data essential for the ongoing refinement of our model.
7	/kaggle/input/4k-mixtral87b-crafted-essays-for-detect-ai-comp/Mixtral8x7b_4k_essays_for_DetectAIGeneratedTextCompetition.csv	Designed for a competition, this dataset includes 4,000 essays from the Mixtral 87B model, challenging our detection algorithms with a wide array of texts.
8	/kaggle/input/daigt-v3-train-dataset/train_v3_drcat_01.csv	An updated training dataset for the project's third iteration, featuring enhanced labeling and new features to better distinguish between AI-generated and human texts.

Dataset Acquisition and Preparation

The project adopts a 70/30 split for dividing the dataset into training and testing sets, a widely accepted practice in machine learning. This ratio ensures a substantial amount of data for training the model, allowing it to learn diverse linguistic patterns, while still reserving enough unseen data for testing, to evaluate the model's generalization capability accurately.

Random selection in splitting the dataset mitigates potential bias was also performed to ensure that both subsets are representative of the overall data distribution, crucial for the model's ability to perform well on new, unseen texts.



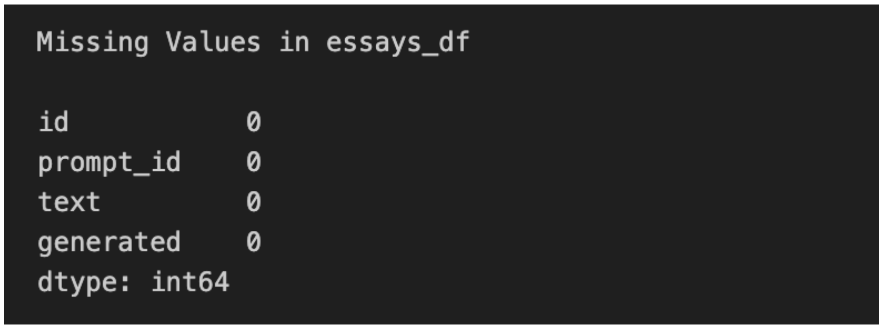
```
train_df, val_df = train_test_split(train, test_size=0.3, random_state=222)
batch_size = 32
```

Figure 1 Division of dataset and random selection

Data Pre-processing

1. Exploratory Data Analysis (EDA) and Data Integrity

initial EDA is conducted to assess the dataset's quality and structure. This includes checking for missing values and analyzing the distribution of different types of texts (e.g., generated vs. student-written). This phase is crucial for identifying potential biases and ensuring the dataset's integrity. In this specific case, the dataset showed no missing values, which streamlined the preprocessing steps. However, the practice of addressing missing values is emphasized as essential for eliminating biases and inaccuracies during model training, even though it was not needed in this particular dataset.

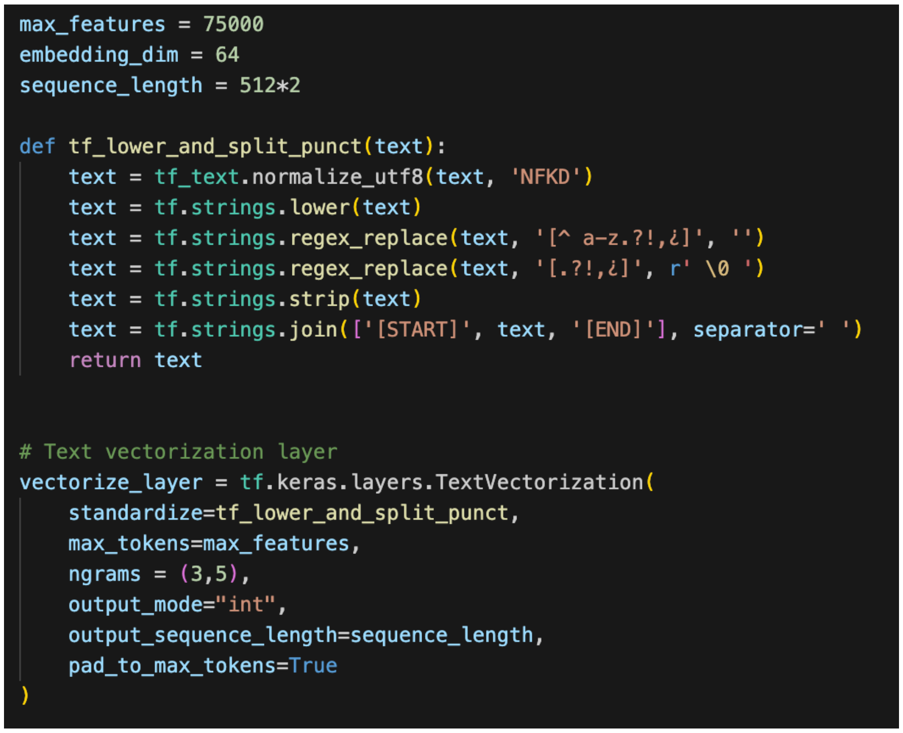


```
Missing Values in essays_df

id           0
prompt_id    0
text         0
generated    0
dtype: int64
```

Figure 2 No missing values in the dataset

2. Standardization and Cleaning

Next step involve transforming all text to lowercase, removing punctuation, and tokenizing texts. This process is aimed at reducing irrelevant variations in the data, such as differences in capitalization and punctuation, which do not contribute to the model's understanding of the content. The goal is to help models focus on the meaningful linguistic features that differentiate between different types of texts, such as those generated by large language models (LLMs) and those written by humans. This standardization enhances the accuracy and efficiency of models by ensuring they concentrate on essential linguistic patterns.

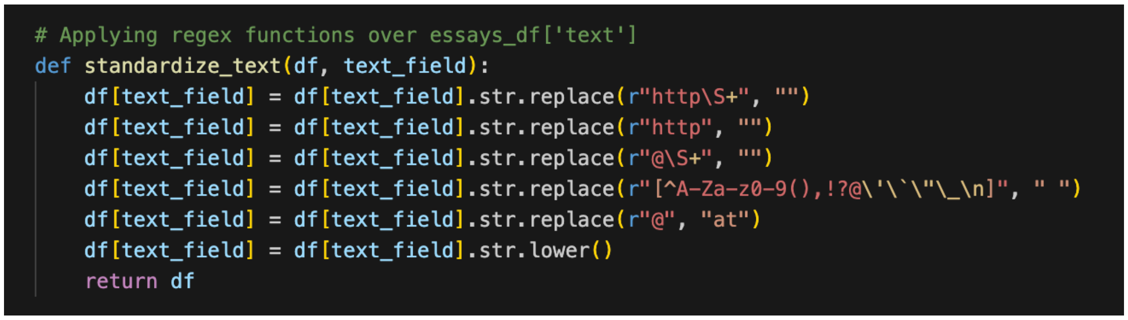


```python
max_features = 75000
embedding_dim = 64
sequence_length = 512*2

def tf_lower_and_split_punct(text):
    text = tf_text.normalize_utf8(text, 'NFKD')
    text = tf.strings.lower(text)
    text = tf.strings.regex_replace(text, '[^ a-z.?!,¿]', '')
    text = tf.strings.regex_replace(text, '[.?!,¿]', r' \0 ')
    text = tf.strings.strip(text)
    text = tf.strings.join(['[START]', text, '[END]'], separator=' ')
    return text


# Text vectorization layer
vectorize_layer = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_features,
    ngrams = (3,5),
    output_mode="int",
    output_sequence_length=sequence_length,
    pad_to_max_tokens=True
)
```

Figure 3 Standardization and Data Cleaning

3. Noise Reduction and Complexity Reduction

The last preprocessing stage includes removing new line tags and URLs, and removing non-alphanumeric characters which could introduce noise into the data which may distract the model from the semantic content of the text. These actions reduce the complexity of text data which help ensure that the presence of non-alphanumeric characters do not impact the model's learning process. These steps are pivotal for preparing the data in an optimal state for model training.



```python
# Applying regex functions over essays_df['text']
def standardize_text(df, text_field):
    df[text_field] = df[text_field].str.replace(r"http\S+", "")
    df[text_field] = df[text_field].str.replace(r"http", "")
    df[text_field] = df[text_field].str.replace(r"@\S+", "")
    df[text_field] = df[text_field].str.replace(r"[^A-Za-z0-9(),!?@\'\`\"\_\n]", " ")
    df[text_field] = df[text_field].str.replace(r"@", "at")
    df[text_field] = df[text_field].str.lower()
    return df
```

Figure 4 Noise Reduction and Complexity Reduction

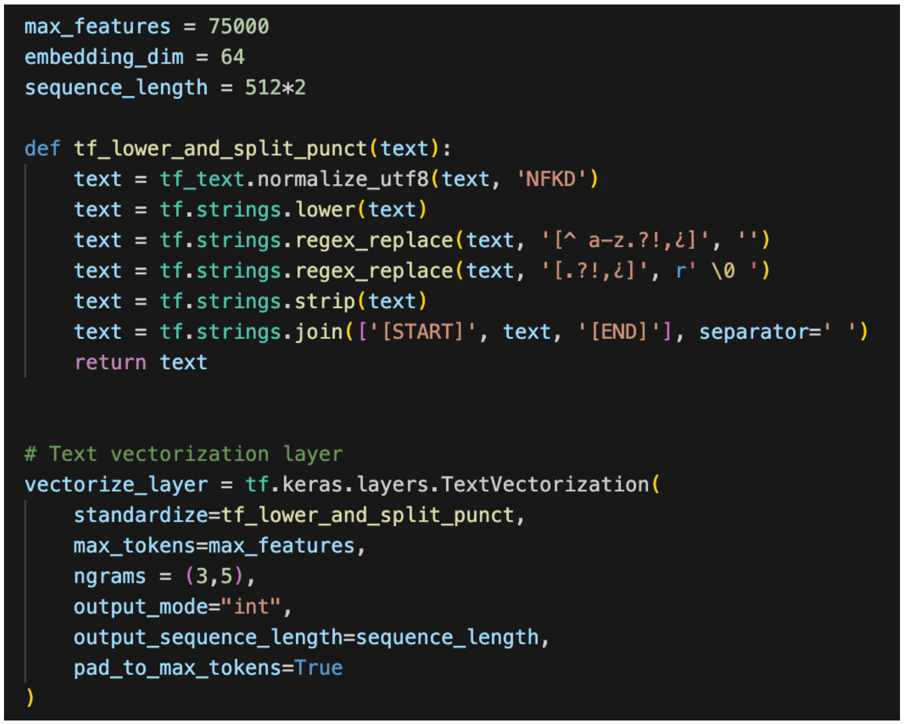
4. Tokenization, Sequence Padding and Vectorization

In classification project which focuses on identifying whether an essay is either generated by Large Language Models (LLMs) or written by humans, employing techniques such as tokenization, padding, and vectorization is important. These preprocessing steps are the foundation for transforming raw text data into a structured form that deep learning models—like RNNs, LSTMs, and FNN can interpret and learn from efficiently. The following sections detail the specific techniques utilized for each model type:

a) RNN (Recurrent Neural Network) Model:

Tokenization and Padding: Tokenization breaks down text into smaller units, and padding ensures uniform sequence lengths, vital for RNN processing. The tf_lower_and_split_punct function standardizes and tokenizes the input, preparing it for efficient learning by reducing vocabulary size and maintaining computational consistency.

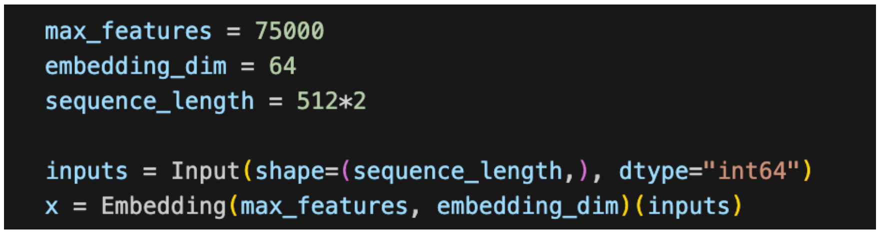
Text Vectorization: The **`TextVectorization`** layer transforms textual data into numerical vectors, using parameters like **`max_tokens`** for vocabulary size and **`output_sequence_length`** for sequence length. Incorporating n-grams (3 to 5) captures broader language patterns, crucial for distinguishing between LLM-generated and human-generated texts.



```
max_features = 75000
embedding_dim = 64
sequence_length = 512*2

def tf_lower_and_split_punct(text):
    text = tf_text.normalize_utf8(text, 'NFKD')
    text = tf.strings.lower(text)
    text = tf.strings.regex_replace(text, '[^ a-z.?!,¿]', '')
    text = tf.strings.regex_replace(text, '[.?!,¿]', r' \0 ')
    text = tf.strings.strip(text)
    text = tf.strings.join(['[START]', text, '[END]'], separator=' ')
    return text


# Text vectorization layer
vectorize_layer = tf.keras.layers.TextVectorization(
    standardize=tf_lower_and_split_punct,
    max_tokens=max_features,
    ngrams = (3,5),
    output_mode="int",
    output_sequence_length=sequence_length,
    pad_to_max_tokens=True
)
```

Figure 5 Tokenization and Padding and Text Vectorization Implementation on RNN training

Embeddings: This technique represents words in a high-dimensional space, allowing the RNN to learn nuanced language representations. This advanced approach to vectorization, leveraging embeddings over simpler encodings, significantly enhances the model's ability to classify texts accurately.

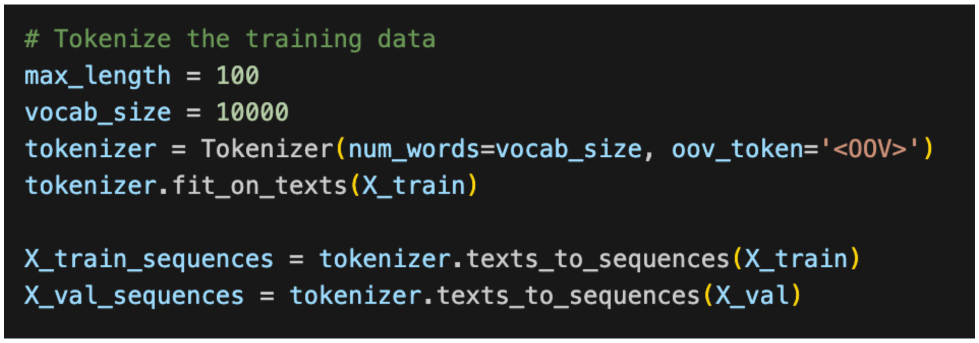


```
max_features = 75000
embedding_dim = 64
sequence_length = 512*2

inputs = Input(shape=(sequence_length,), dtype="int64")
x = Embedding(max_features, embedding_dim)(inputs)
```

Figure 6 Embedding application

b) LSTM (Long Short-Term Memory) Model:

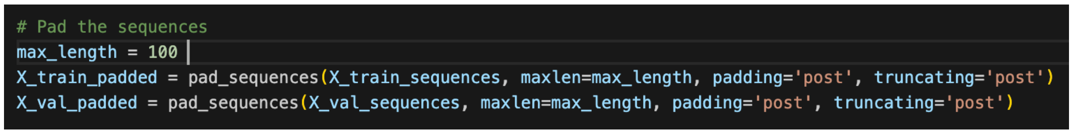
Tokenization: Tokenization converted the text into sequences of integers, with each integer representing a specific word or token, within a defined **`vocab_size`** of 10,000. This size struck a balance between capturing a comprehensive vocabulary and maintaining model manageability and efficiency.



```
# Tokenize the training data
max_length = 100
vocab_size = 10000
tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_val_sequences = tokenizer.texts_to_sequences(X_val)
```

Figure 7 Tokenization on LSTM Model Training

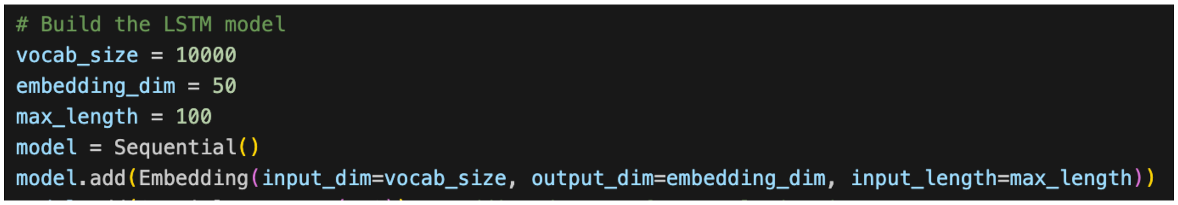
Sequence Padding: Sequence padding standardized the length of these sequences to 100, ensuring uniform input size for the LSTM model. This uniformity is essential for efficient batch processing and allows the model to efficiently process the data, capturing the necessary information within a manageable computational framework.



```
# Pad the sequences
max_length = 100
X_train_padded = pad_sequences(X_train_sequences, maxlen=max_length, padding='post', truncating='post')
X_val_padded = pad_sequences(X_val_sequences, maxlen=max_length, padding='post', truncating='post')
```

Figure 8 Sequence Padding on LSTM Model Training

Vectorization: Vectorization was implicitly conducted through the embedding process following tokenization and padding. The embedding layer maps the integer-encoded tokens to dense vectors of fixed size, facilitating the model's understanding of semantic relationships within the text. This step is crucial for enabling the LSTM to differentiate between LLM-generated and human-generated texts based on nuanced language use and structure.



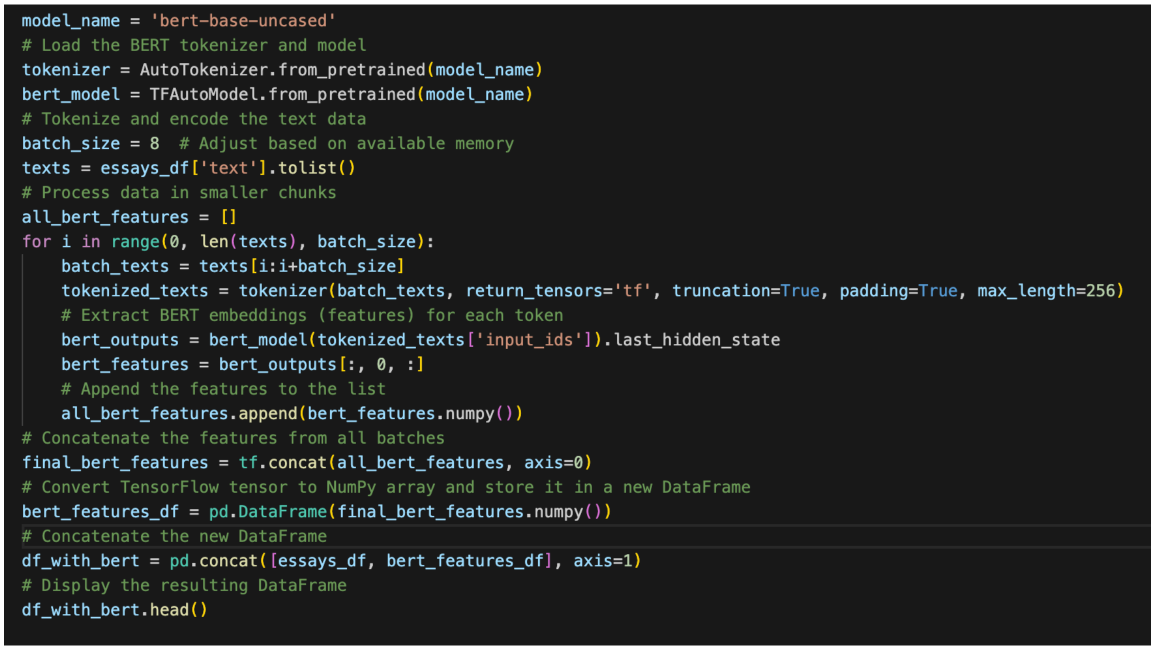
```
# Build the LSTM model
vocab_size = 10000
embedding_dim = 50
max_length = 100
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length))
```

Figure 9 Vectorization on LSTM Model Training

c) **FNN (Feedforward Neural Network) Model:**

Tokenization and Padding: Tokenization segments text into tokens using the BERT tokenizer, structuring raw text into a format that retains context and meaning. Padding ensures all sequences are the same length for efficient batching, adjusting shorter sequences with zeros to achieve consistent dimensions. A maximum sequence length of 256 tokens balances the need for sufficient context against computational efficiency.

Vectorization: BERT embeddings vectorize tokenized inputs into high-dimensional vectors, capturing the semantic features essential for understanding language nuances. The choice of BERT embeddings, derived from the model's last hidden state, offers a deep understanding of language context, suitable for complex language comprehension tasks.



```
model_name = 'bert-base-uncased'
# Load the BERT tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_name)
bert_model = TFAutoModel.from_pretrained(model_name)
# Tokenize and encode the text data
batch_size = 8  # Adjust based on available memory
texts = essays_df['text'].tolist()
# Process data in smaller chunks
all_bert_features = []
for i in range(0, len(texts), batch_size):
    batch_texts = texts[i:i+batch_size]
    tokenized_texts = tokenizer(batch_texts, return_tensors='tf', truncation=True, padding=True, max_length=256)
    # Extract BERT embeddings (features) for each token
    bert_outputs = bert_model(tokenized_texts['input_ids']).last_hidden_state
    bert_features = bert_outputs[:, 0, :]
    # Append the features to the list
    all_bert_features.append(bert_features.numpy())
# Concatenate the features from all batches
final_bert_features = tf.concat(all_bert_features, axis=0)
# Convert TensorFlow tensor to NumPy array and store it in a new DataFrame
bert_features_df = pd.DataFrame(final_bert_features.numpy())
# Concatenate the new DataFrame
df_with_bert = pd.concat([essays_df, bert_features_df], axis=1)
# Display the resulting DataFrame
df_with_bert.head()
```

Figure 10 Tokenization, Padding and Tokenization using BERT

IV. Model Performance Evaluation and Selection (RNN, LSTM and FNN)

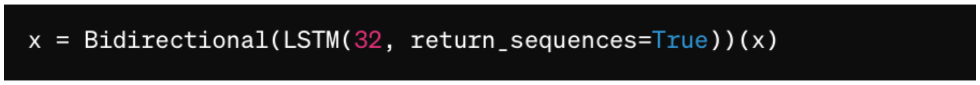
RNN Model

1. RNN Model Training:

The integration of RNN and Transformer models within the architecture capitalizes on RNNs' proficiency in processing sequential data and Transformers' ability to emphasize significant input features via attention mechanisms. This combination is important for discerning the complex linguistic signals that differentiate LLM-generated texts from those authored by humans.

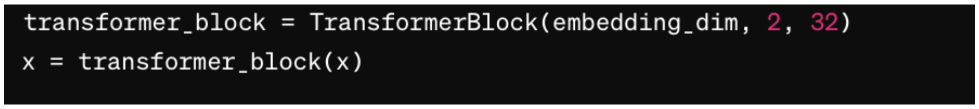
The model's compilation with binary `crossentropy` loss and the Adam optimizer complements the architectural choices by ensuring efficient training dynamics and optimization pathways. Progressing to a second epoch, informed by a marked improvement in accuracy, underscores a strategic commitment to refining the model's predictive accuracy.

Sequential Processing and Attention Mechanism: The use of a Bidirectional LSTM layer, followed by a `TransformerBlock`, underscores the model's dual strategy. This setup not only captures the essential sequence of textual data but also prioritizes critical contextual information, enhancing the model's discernment between human and LLM-generated texts.



```
x = Bidirectional(LSTM(32, return_sequences=True))(x)
```

Figure 11 Bidirectional LSTM layer



```
transformer_block = TransformerBlock(embedding_dim, 2, 32)
x = transformer_block(x)
```

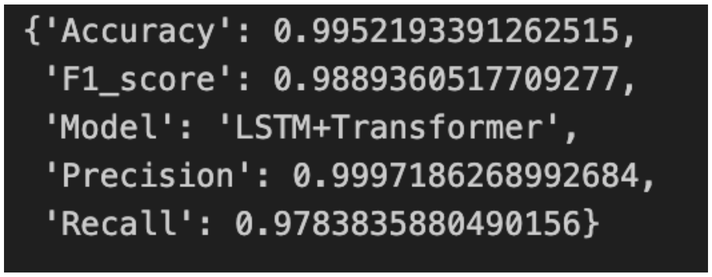
Figure 12 Transformer Block

Epochs and Learning Rate: The specified training epochs, facilitated by the Adam optimizer's dynamic learning rate adjustments, shows the model's iterative learning and optimization strategy. This approach ensures that each epoch contributes to the model's evolving understanding and classification capabilities, with the decision to extend training being validated by observable performance gains.

Embedding and Network Configuration: Optimal selection of the embedding dimension and the architectural configuration of LSTM and Transformer components are essential. this enable the model to transmute raw text into meaningful representations which then captues the essential sequential and contextual nuances vital for accurate classification. The model's compilation, featuring binary `crossentropy` and the Adam optimizer, further optimizes the representations towards achieving high classification accuracy.

2. RNN Model Evaluation:

Evaluation metrics demonstrate the model's high performance, with notable precision and accuracy. However, the contrast between high precision and slightly lower recall suggests potential overfitting, indicating areas for future improvement. Strategies like introducing dropout layers, regularization, or dataset augmentation could further optimize the model.



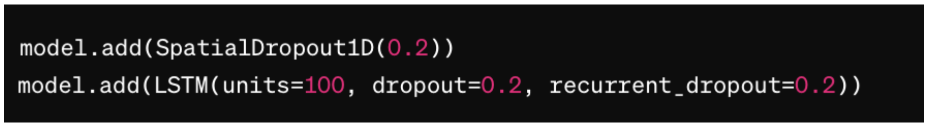
```
{'Accuracy': 0.9952193391262515,
 'F1_score': 0.9889360517709277,
 'Model': 'LSTM+Transformer',
 'Precision': 0.9997186268992684,
 'Recall': 0.9783835880490156}
```

Figure 13 RNN Model Evaluation

LSTM Model

1. LSTM Model Training:

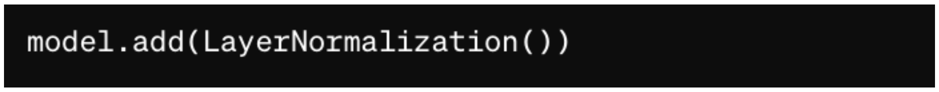
Dropout: To prevent overfitting, dropout technique was employed. Dropout involves randomly setting a fraction of input units to 0 at each update during training time, which helps in preventing complex co-adaptations on training data.



```
model.add(SpatialDropout1D(0.2))
model.add(LSTM(units=100, dropout=0.2, recurrent_dropout=0.2))
```

Figure 14 Dropout Technique applied in LSTM Model Training

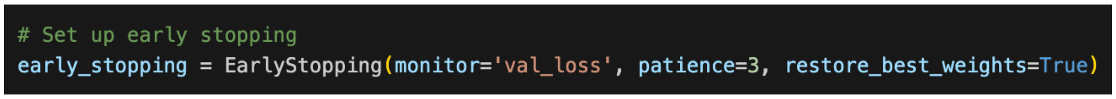
Layer Normalization: Layer normalization is another technique used to improve the training of the LSTM model. By normalizing the inputs across the features, it stabilizes the hidden layer activations which makes the training process more efficient and stable. This is beneficial for LSTM where maintaining stable gradients is crucial due to the recurrent nature of the networks.



```
model.add(LayerNormalization())
```

Figure 15 Layer Normalization applied in LSTM Model Training

Early Stopping: The training process utilized early stopping based on validation loss. This approach monitors a specific performance metric – in this case, the loss on the validation set – and stops the training process if the performance ceases to improve which prevents overfitting. The decision to stop training after three epochs was based on observing diminishing returns in loss reduction and an optimal stopping point for training to maximize model performance without overfitting.



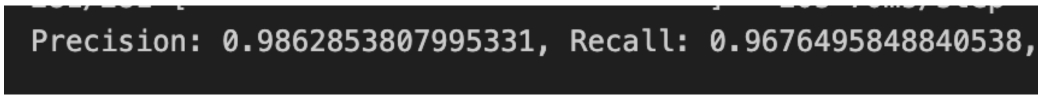
```
# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

Figure 16 Early Stopping in LSTM Model Training

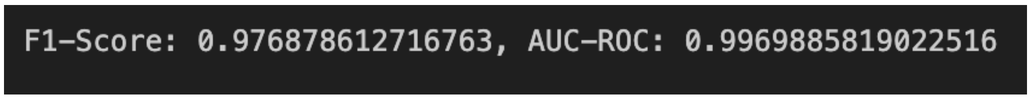
2. LSTM Model Evaluation

Evaluation metrics including precision, recall, F1-score, and AUC-ROC demonstrated the model's high efficacy in accurately classifying texts. These metrics indicated a strong ability to distinguish between LLM-generated and human-generated texts, with minimal false positives and negatives, and a high degree of accuracy across various thresholds.

The model showed no signs of overfitting, attributed to the implementation of regularization techniques and the early stopping mechanism. Continuous evaluation and potential future improvements such as exploring new model architectures or incorporating diverse training data were recognized as essential for sustaining and enhancing model robustness and accuracy.



```
Precision: 0.9862853807995331, Recall: 0.9676495848840538,
```



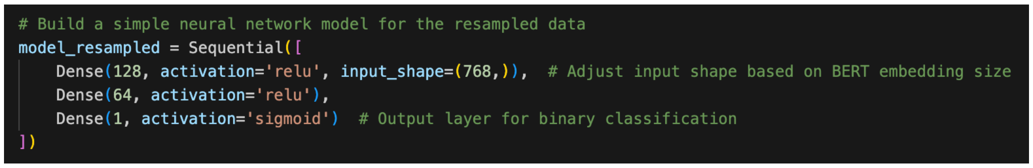
```
F1-Score: 0.976878612716763, AUC-ROC: 0.9969885819022516
```

Figure 17 LSTM Metrics

FNN Model

1. FNN Model Training:

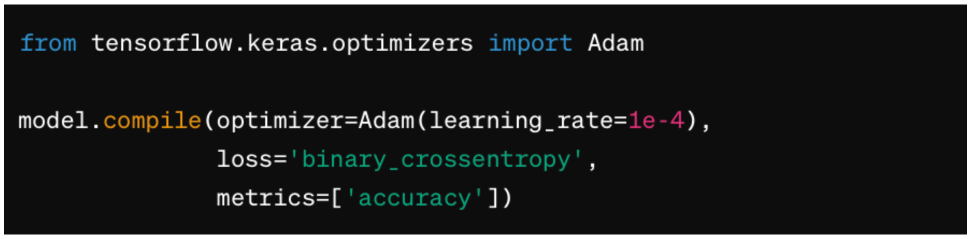
Model Architecture: The architecture, starting with an input layer tailored to BERT embedding dimensions, followed by multiple hidden layers with ReLU, and culminating in a sigmoid output layer for binary classification, is well-suited for the task. The sigmoid function is ideal for binary classification as it maps input values to a probability score between 0 and 1, indicating the likelihood of the input being in a particular class.



```
# Build a simple neural network model for the resampled data
model_resampled = Sequential([
    Dense(128, activation='relu', input_shape=(768,)),  # Adjust input shape based on BERT embedding size
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])
```

Figure 18 Model Architecture in FNN Model Training

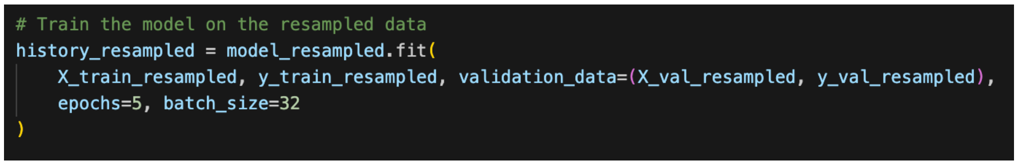
Adam Optimizer and Binary Cross entropy: The choice of the Adam optimizer is justified by its ability to adjust learning rates dynamically, making it more effective for datasets with complex and high-dimensional feature spaces like those encountered in NLP. Binary `crossentropy` is the preferred loss function for binary classification tasks because it quantifies the difference between the predicted probabilities and the actual binary outcomes, offering a clear gradient for optimization.



```python
from tensorflow.keras.optimizers import Adam

model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Figure 19 Adam Optimizer and Binary Cross entropy in FNN Model Training

Batch Size and Epochs: The selection of a batch size of 32 and 5 training epochs represents a balanced approach, optimizing the trade-off between computational efficiency and the model's exposure to the training data. A batch size of 32 is large enough to ensure stable gradient estimates while small enough to maintain computational efficiency. Training for 5 epochs allows the model to learn from the data sufficiently without overfitting, given the complexity of the input features and the effectiveness of the chosen optimizer.

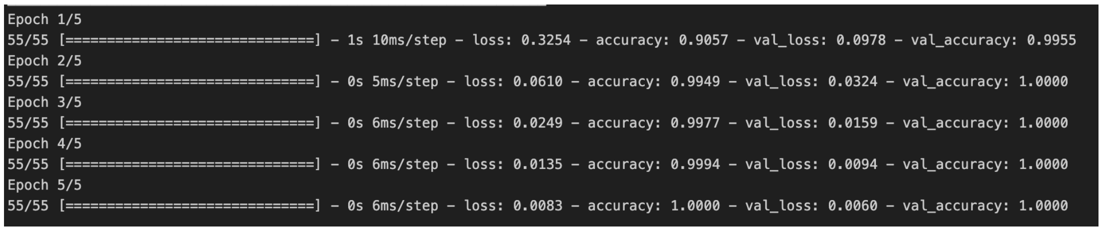


```python
# Train the model on the resampled data
history_resampled = model_resampled.fit(
    X_train_resampled, y_train_resampled, validation_data=(X_val_resampled, y_val_resampled),
    epochs=5, batch_size=32
)
```

Figure 20 Batch Size and Epochs in FNN Model Training

2. FNN Model Evaluation

The model's performance metrics suggest it is learning effectively and generalizing well to unseen data, however, the attainment of 100% accuracy on both training and validation datasets raises questions about its potential overfitting. To ensure the model's robustness and its ability to generalize across different contexts, further validations such as evaluating the model on a diverse test set, implementing cross-validation, and considering regularization techniques should be performed.



```
Epoch 1/5
55/55 [==============================] - 1s 10ms/step - loss: 0.3254 - accuracy: 0.9057 - val_loss: 0.0978 - val_accuracy: 0.9955
Epoch 2/5
55/55 [==============================] - 0s 5ms/step - loss: 0.0610 - accuracy: 0.9949 - val_loss: 0.0324 - val_accuracy: 1.0000
Epoch 3/5
55/55 [==============================] - 0s 6ms/step - loss: 0.0249 - accuracy: 0.9977 - val_loss: 0.0159 - val_accuracy: 1.0000
Epoch 4/5
55/55 [==============================] - 0s 6ms/step - loss: 0.0135 - accuracy: 0.9994 - val_loss: 0.0094 - val_accuracy: 1.0000
Epoch 5/5
55/55 [==============================] - 0s 6ms/step - loss: 0.0083 - accuracy: 1.0000 - val_loss: 0.0060 - val_accuracy: 1.0000
```

Figure 21 FNN Model Evaluation

V. Selection of LSTM as Optimal Model for Project Performance

After performing training on 3 different models, our group's choice of Long Short-Term Memory (LSTM) networks for classifying whether an essay is generated by a Large Language Model (LLM) or not is supported by several advantages of LSTMs over traditional

Recurrent Neural Networks (RNNs) and Feedforward Neural Networks (FNNs), especially in terms of learning efficiency, resource requirements, training duration, and accuracy in classification. The discussion below explains why LSTM was the optimal choice for this project, especially considering the training environment on Google Colab utilizing a T4 GPU.

Learning Efficiency

LSTMs excel over traditional RNNs by solving the vanishing gradient issue, allowing for the retention and processing of information across long sequences. This capability is crucial for text classification, especially in differentiating LLM-generated from human-generated texts, due to their architecture that manages long-term dependencies with specialized gates. Unlike Feedforward Neural Networks (FNNs) that overlook sequence dependencies, LSTMs effectively leverage the sequential context, markedly improving the efficiency and accuracy in handling complex textual analysis.

Resource Requirements and Training Duration

LSTMs offer significant advantages in resource utilization, training efficiency, and classification accuracy, especially when trained on platforms like Google Colab using T4 GPUs. Its architecture is well-suited for parallel processing, making it more efficient than RNNs, which is hampered by the vanishing gradient problem, and FNNs, which lack the ability to process sequential data effectively. Early stopping further optimizes LSTM training by minimizing resource use and preventing overtraining, enhancing time and resource efficiency.

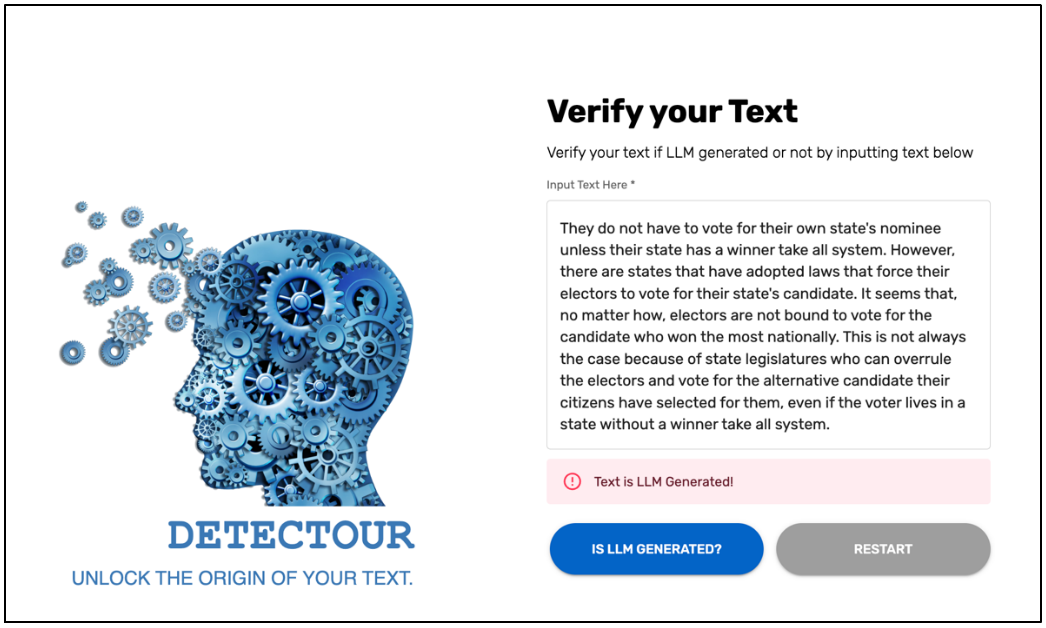
In terms of classification accuracy, LSTMs surpass RNNs and FNNs due to its superior ability to understand context and manage long sequences. Its capability allows LSTMs to accurately differentiate between LLM-generated and human-generated texts, as shown by high precision, recall, F1-score, and AUC-ROC metrics. Its proficiency in handling complex patterns and nuances in text makes it particularly effective for this project, outperforming models that cannot capture sequential dependencies as effectively.

VI. UI Demonstration

Our team has created a user-friendly interface to easily validate essays as LLM-generated or human-authored. We utilized ReactJS for the frontend to create an interactive experience and Python Django for the backend for its efficiency and seamless model integration. This setup ensures real-time essay processing and validation.

The interface features an essay input box and two buttons: "IS LLM GENERATED?" for initiating the analysis and "RESTART" for clearing inputs, facilitating a smooth user journey. This design not only highlights the model's capabilities but also provides an intuitive platform for users to verify essays, merging advanced machine learning with ease of use.

Validation Data No. 1:



Verify your Text

Verify your text if LLM generated or not by inputting text below

Input Text Here *

They do not have to vote for their own state's nominee unless their state has a winner take all system. However, there are states that have adopted laws that force their electors to vote for their state's candidate. It seems that, no matter how, electors are not bound to vote for the candidate who won the most nationally. This is not always the case because of state legislatures who can overrule the electors and vote for the alternative candidate their citizens have selected for them, even if the voter lives in a state without a winner take all system.

Text is LLM Generated!

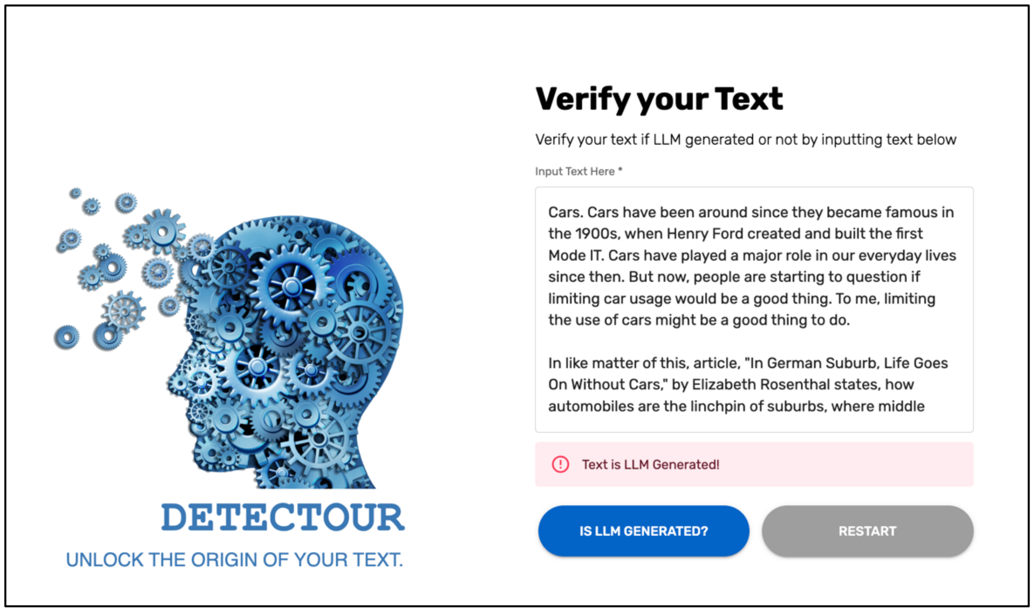
Figure 22 UI Demonstration for Validation 02

Input Text:

This essay will analyze, discuss and prove one reason in favor of keeping the Electoral College in the United States for its presidential elections. One of the reasons to keep the electoral college is that it is better for smaller, more rural states to have more influence as opposed to larger metropolitan areas that have large populations. The electors from these states are granted two votes each. Those from larger, more populated areas are granted just one vote each. Smaller states tend to hold significant power because their two votes for president and vice president add up more than the votes of larger states that have many electors. This is because of the split of the electoral votes. Some argue that electors are not bound to vote for the candidate who won the most votes nationally. They do not have to vote for their own state's nominee unless their state has a winner take all system. However, there are states that have adopted laws that force their electors to vote for their state's candidate. It seems that, no matter how, electors are not bound to vote for the candidate who won the most nationally. This is not always the case because of state legislatures who can overrule the electors and vote for the alternative candidate their citizens have selected for them, even if the voter lives in a state without a winner take all system.

Expected Result: Text is LLM Generated!
Actual Result: Text is LLM Generated!
Verdict: CORRECT

Validation Data No. 2:



Verify your Text

Verify your text if LLM generated or not by inputting text below

Input Text Here *

Cars. Cars have been around since they became famous in the 1900s, when Henry Ford created and built the first Mode IT. Cars have played a major role in our everyday lives since then. But now, people are starting to question if limiting car usage would be a good thing. To me, limiting the use of cars might be a good thing to do.

In like matter of this, article, "In German Suburb, Life Goes On Without Cars," by Elizabeth Rosenthal states, how automobiles are the linchpin of suburbs, where middle

Text is LLM Generated!

Figure 23 UI Demonstration for Validation 01

Input Text:

Cars. Cars have been around since they became famous in the 1900s, when Henry Ford created and built the first Mode IT. Cars have played a major role in our everyday lives since then. But now, people are starting to question if limiting car usage would be a good thing. To me, limiting the use of cars might be a good thing to do.

In like matter of this, article, "In German Suburb, Life Goes On Without Cars," by Elizabeth Rosenthal states, how automobiles are the linchpin of suburbs, where middle class families from either Shanghai or Chicago tend to make their homes. Experts say how this is a huge impediment to current efforts to reduce greenhouse gas emissions from tailpipe. Passenger cars are responsible for 12 percent of greenhouse gas emissions in Europe...and up to 50 percent in some car intensive areas in the United States. Cars are the main reason for the greenhouse gas emissions because of a lot of people driving them around all the time getting where they need to go. Article, "Paris bans driving due to smog," by Robert Duffer says, how Paris, after days of near record pollution, enforced a partial driving ban to clear the air of the global city. It also says, how on Monday, motorist with even numbered license plates were ordered to leave their cars at home or be fined a 22euro fine 31. The same order would be applied to odd numbered plates the following day. Cars are the reason for polluting entire cities like Paris. This shows how bad cars can be because, of all the pollution that they can cause to an entire city.

Likewise, in the article, "Car free day is spinning into a big hit in Bogota," by Andrew Selsky says, how programs that's set to spread to other countries, millions of Columbians hiked, biked, skated, or took the bus to work during a car free day, leaving streets of this capital city eerily devoid of traffic jams. It was the third straight year cars have been banned with only buses and taxis permitted for the Day Without Cars in the capital city of 7 million. People like the idea of having car free days because, it allows them to lesson the pollution that cars put out of their exhaust from people driving all the time. The article also tells how parks and sports centers have bustled throughout the city uneven, pitted sidewalks have been replaced by broad, smooth sidewalks rush hour restrictions have dramatically cut traffic and new restaurants and upscale shopping districts have cropped up. Having no cars has been good for the country of Columbia because, it has aloud *them to repair things that have needed repairs for a long time, traffic jams have gone down, and restaurants and shopping districts have popped up, all due to the fact of having less cars around.*

In conclusion, the use of less cars and having car free days, have had a big impact on the environment of cities because, it is cutting down the air pollution that the cars have majorly polluted, it has a loud countries like Columbia to repair sidewalks, and cut down traffic jams. Limiting the use of cars would be a good thing for America. So we should limit the use of cars by maybe riding a bike, or maybe walking somewhere that isn't that far from you and doesn't need the use of a car to get you there. To me, limiting the use of cars might be a good thing to do.

Expected Result: Text is NOT LLM Generated!
Actual Result: Text is LLM Generated!
Verdict: INCORRECT

VII. Conclusions

After conducting tests through the user interface, our team observed that the results often matched our expectations, accurately identifying essays as either LLM-generated or human-authored. However, there were instances of discrepancies between expected and actual outcomes. This led us to conclude that the LSTM model's performance could be further enhanced. One approach is by refining the preprocessing of the dataset to ensure that human-generated texts incorporate distinctive elements such as personalized vocabulary, slang, and a first-person narrative style. These modifications aim to create a test dataset with a tone distinctly different from that of LLM-generated texts, thereby improving the model's ability to differentiate between the two during the training phase.

In addition to dataset optimization, implementing techniques to prevent overfitting, such as dropout, is crucial for the model's subsequent development. Other strategies like regularization and early stopping could also be instrumental in enhancing the model's generalization capabilities. Regularization techniques, including L1 and L2 regularization, penalize model complexity, encouraging simpler models that are less likely to overfit. Early stopping, on the other hand, monitors the model's performance on a validation set and halts training when performance ceases to improve, preventing overfitting on the training data.

In summary, while the LSTM model demonstrates promising capabilities in distinguishing between LLM-generated and human-written texts, there is room for improvement. By carefully preprocessing the dataset to accentuate the unique characteristics of human writing and employing measures to prevent overfitting, we anticipate significant advancements in the model's accuracy and reliability. These enhancements will not only refine the model's performance but also contribute to the broader field of natural language processing, offering more nuanced and sophisticated tools for text analysis.