**Final Project Report**

**Deep Learning & Business Applications**

LLM - Detect AI Generated Text: Identify which essay was written by a large language model.

A close-up of a person writing on a notebook

Description automatically generated

By

Uday Adusumilli

Sowjanya Sakhamuri

Prameela Rani Vakkala Gadda

Shashidhar Naidu Boya

# Table of Contents

[Introduction 3](#_Toc153738843)

[Overview 3](#_Toc153738844)

[Motivation 3](#_Toc153738845)

[Research Questions 4](#_Toc153738846)

[Literature Survey 4](#_Toc153738847)

[Challenges 4](#_Toc153738848)

[State-of-the-Art Survey 4](#_Toc153738849)

[Dataset Description 5](#_Toc153738850)

[Data Preprocessing 6](#_Toc153738851)

[Method 1: Tokenization and Feature Extraction 6](#_Toc153738852)

[Method 2 - BERT Tokenization and Model Training 7](#_Toc153738857)

[Algorithm and Baseline Models for Comparison 9](#_Toc153738863)

[BPE (Byte Pair Encoding) Tokenizer 10](#_Toc153738864)

[Deep Learning Modules 11](#_Toc153738870)

[Experiments, Results, and Analysis 13](#_Toc153738871)

[Analysis 17](#_Toc153738872)

[Results 17](#_Toc153738873)

[Conclusion and Future Work 18](#_Toc153738874)

[References 18](#_Toc153738875)

# Introduction

## Overview

The rapid advancement and widespread use of large language models (LLMs) like GPT-3 and GPT-4 have revolutionized the way text is generated, posing unique challenges in differentiating between human and AI-generated content. This difficulty is particularly pronounced in educational and academic spheres, where the integrity and authenticity of written work are paramount.

As LLMs become more sophisticated, they can produce essays that closely mimic human writing in style, coherence, and complexity. This ability raises significant concerns about academic dishonesty and the potential undermining of educational standards. Students might use these models to generate essays, reports, or other assignments, posing a threat to the traditional evaluation methods in educational institutions. Similarly, in academic research, the authenticity of authorship becomes questionable, potentially impacting the credibility of research outputs (Silvia Milano, 2023).

Addressing this issue, the project in question aims to develop and implement effective methods for detecting essays and texts generated by LLMs. Such detection mechanisms are vital to maintain academic integrity, ensure fair assessment in educational settings, and uphold the standards of original research and authorship. This involves creating algorithms and tools that can analyze writing patterns, stylistic markers, and other textual characteristics that differentiate AI-generated content from human writing. The success of this project would be a significant step in preserving the ethical use of AI in educational and academic contexts, ensuring that these advanced technologies augment rather than undermine the value of human intellect and creativity.

## Motivation

The rising use of LLMs in generating text poses significant challenges, especially in educational settings where they might be used for plagiarism. Our work aims to develop a reliable model to differentiate between student-written essays and those generated by LLMs.

## Research Questions

* How accurately can a machine learning model differentiate between student-written and LLM-generated essays?
* What features are most indicative of AI-generated text in this context?
* Can a model be both highly accurate and computationally efficient in detecting LLM-generated essays?

# Literature Survey

## Challenges

The primary challenge lies in addressing the increasingly sophisticated outputs of Large Language Models (LLMs) that closely replicate human writing styles and nuances. This advancement in LLMs presents a dual issue: firstly, distinguishing between AI-generated and human-written texts becomes more complex as AI texts become more nuanced and less distinguishable from human output (Silvia Milano, 2023). Secondly, the variability in essay topics and individual writing styles among students adds another layer of complexity (Heikkilä, 2022). This variability means that there isn't a one-size-fits-all approach to identifying AI-generated texts, as each student's writing is unique and can significantly differ in style and content, making standardization of detection methods challenging.

Heather Desaire et al.'s study illustrates this challenge, emphasizing the urgency and critical need to discriminate human writing from AI in the academic sphere. They developed a method with over 99% accuracy in discriminating AI from human text, based on features like paragraph length and language nuances. This highlights a growing trend in using sophisticated classification methods to tackle AI-generated text detection.

## State-of-the-Art Survey

Recent advances in NLP and machine learning have led to the development of models capable of text classification with high accuracy. Techniques like BERT, GPT, and other deep learning frameworks have set benchmarks in understanding and generating human-like text. However, their application in distinguishing between human and AI-generated content, especially in an educational context, remains underexplored.

Silvia Milano et al. discuss the broader implications of LLM adoption in education. They argue that while LLMs like ChatGPT offer creative teaching and assessment opportunities, they also pose operational, financial, pedagogical, and ethical risks. This perspective is crucial for understanding the long-term implications of integrating LLMs into academic settings.  
BERT Module

BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pretrained Transformer) are two state-of-the-art deep learning models in natural language processing (NLP), both built upon the innovative transformer architecture. This architecture is renowned for its self-attention mechanisms, which allow the model to consider the importance of each word in a sentence, independent of their positional context.

BERT, developed by Google, represents a significant advancement in NLP. Its defining feature is its bidirectional approach to understanding the context of a word in a sentence. Traditional models processed text in a single direction, either from left to right or vice versa. BERT, however, examines the context from both directions, considering words that appear both before and after the target word. This bidirectional context comprehension enables BERT to achieve a more nuanced understanding of language. The training of BERT involves two stages: pre-training and fine-tuning.

During pre-training, BERT learns from a vast corpus of text, absorbing general patterns and structures of the language. In the fine-tuning stage, it is specifically tailored to perform various language tasks such as sentiment analysis, question answering, and language inference. The innovative aspect of BERT is its ability to process entire sentences, rather than in fragments, making it exceptionally adept at tasks requiring a deep understanding of context and language subtleties.

# Dataset Description

The dataset provided for this competition is a collection of approximately 35000 essays. These essays are a mix of student-written and LLM-generated texts. Key aspects of the dataset include:

**Diverse Sources:** Essays are generated by various LLMs, ensuring a wide range of writing styles and capabilities are represented.

**Response to Prompts:** Both student and LLM-generated essays are written in response to one of seven specific prompts. This setup mirrors typical academic assignments.

**Training and Test Sets:** The dataset is divided into a training set (comprising essays from two prompts) and a hidden test set (comprising essays from the remaining prompts). The training set primarily consists of student-written essays.

**Use of Dummy Data:** The `test\_essays.csv` file contains dummy data for solution development, which will be replaced with the actual test set during the evaluation phase.

# Data Preprocessing

In the data preprocessing phase of our project, we employed a series of techniques to prepare the essay texts for machine learning modeling. These techniques are focused on tokenization and feature extraction, crucial steps for transforming raw text into a format that can be effectively used by machine learning algorithms.

## Method 1: Tokenization and Feature Extraction

### Tokenization Using Byte-Pair Encoding (BPE)

The first step in our preprocessing was tokenization using Byte-Pair Encoding (BPE), a popular method in natural language processing (NLP). We utilized the `Tokenizer` class from a relevant NLP library, configuring it with several components:

1. Pre-Tokenization: Applied byte-level pre-tokenization, which helps in managing encoding issues and improves the model's ability to handle a variety of text inputs.
2. Normalization: Implemented a sequence of normalizers: NFD (Unicode Normalization Form D), Lowercase conversion, and Stripping of accents. This ensures uniformity in the text by converting all characters to lowercase and removing accent marks.
3. Special Tokens: Incorporated special tokens like `[UNK]`, `[CLS]`, `[SEP]`, `[PAD]`, and `[MASK]` in the tokenizer's vocabulary. These tokens are essential for various NLP tasks and model architectures.
4. Training the Tokenizer: The tokenizer was trained on our dataset, ensuring that the vocabulary is tailored to the specific language used in our essays.

### Feature Extraction with TF-IDF Vectorization

Post tokenization, we implemented feature extraction using the Term Frequency-Inverse Document Frequency (TF-IDF) method. This technique converts the tokenized texts into numerical vectors, capturing the importance of each token in the context of the entire dataset.

1. Custom Tokenizer and Preprocessor: A custom tokenizer and preprocessor function (`dummy`) was used, ensuring that the tokenization done in the previous step is retained in the TF-IDF process.
2. TF-IDF Configuration: The TF-IDF Vectorizer was configured with specific parameters:
   1. `ngram\_range` of (3, 5), focusing on capturing both short and longer phrase patterns in the text.
   2. `sublinear\_tf` set to `True`, applying a logarithmic scale to term frequencies for reducing the impact of very high term frequencies.
   3. Preservation of the original case (`lowercase=False`) and use of the previously established vocabulary.
3. Vectorization of Train and Test Sets: Both the training and test datasets were transformed using the configured TF-IDF Vectorizer, resulting in numerical feature representations of the essays.

### Memory Management

After vectorization, to optimize memory usage and improve computational efficiency, we performed garbage collection using `gc.collect()`. This step is crucial in managing resources, especially when dealing with large datasets.

### Implications

These preprocessing steps are vital in converting raw text into a structured format suitable for machine learning. The tokenization process, particularly using BPE, ensures that the model can understand and process the nuances of the text data. Following this, the TF-IDF Vectorization captures the importance of different words and phrases in the dataset, creating a feature set that can effectively train machine learning models to differentiate between student-written and LLM-generated essays.

## Method 2 - BERT Tokenization and Model Training

### BERT Tokenization

In this preprocessing approach, we utilize the BERT (Bidirectional Encoder Representations from Transformers) tokenizer from the Hugging Face library. BERT, known for its powerful contextual language understanding, is ideal for tasks requiring deep understanding of text semantics.

### Process

1. Initialization: We start by initializing the `BertTokenizer` with the 'bert-base-uncased' model, which processes text in lowercase and is one of the standard BERT models.
2. Tokenize and Encode: The `bert\_encode` function tokenizes the texts and then encodes them. Each text is truncated or padded to a maximum length of 512 tokens (though in the provided code it's set to 160 for efficiency). The function generates:
   1. Token IDs (`all\_tokens`) representing each token in the text.
   2. Attention Masks (`all\_masks`) indicating to the model which tokens should be focused on.
   3. Segment IDs (`all\_segments`) used for distinguishing different sequences (not essential in this single-sequence task).
3. Preparing Input Data: We then apply this function to both the training and test datasets to prepare our input data for the BERT model.

### Building and Training the BERT Model

Using the tokenized data, we build and train a BERT-based model for our classification task.

1. Model Loading: The `TFBertModel` is loaded with the 'bert-base-uncased' configuration.
2. Model Architecture: The model inputs include the token IDs, attention masks, and segment IDs. We extract the pooled output from BERT, which is a summary of the entire input sequence and is suitable for classification tasks.
3. Output Layer: A dense layer with a sigmoid activation function is added for binary classification (essay written by student vs. generated by LLM).
4. Compilation and Training: The model is compiled and then trained on the prepared input data, with binary crossentropy as the loss function and accuracy as the metric.

### Post-Training Steps

1. Extract BERT Embeddings: After training, we use the model to predict embeddings for the training data, essentially transforming the essays into high-dimensional feature vectors encapsulating their semantic content.
2. Data Splitting: We split these embeddings into training and validation sets.
3. Handling Class Imbalance with SMOTE: To address any class imbalance in the training data, we apply the Synthetic Minority Over-sampling Technique (SMOTE), creating synthetic examples of the minority class.

### Implications

This approach leverages the advanced NLP capabilities of BERT to understand the contextual nuances of the essays. By training a BERT-based model, we aim to capture complex patterns that could distinguish between student-written and LLM-generated texts. The use of embeddings as features in subsequent modeling stages can significantly enhance the detection capabilities, given the rich semantic information encapsulated in these embeddings. The application of SMOTE further ensures that our model is not biased towards the majority class, improving its generalization ability.

# Algorithm and Baseline Models for Comparison

In our project, a significant emphasis is placed on the preprocessing and tokenization of essay data. We utilize a custom tokenizer built with BPE (Byte Pair Encoding) techniques and further refine it with pre-tokenization strategies like ByteLevel processing and normalization steps (NFD normalization, lowercase conversion, and stripping accents). This tokenizer is trained on a dataset of essays, ensuring that it is well-tuned to the specific linguistic patterns present in the data. The essays are tokenized with special tokens ([CLS], [SEP], [PAD], [MASK]) incorporated, aligning with practices common in advanced language models.

For feature extraction, we implement a TfidfVectorizer with a specific focus on trigrams to pentagrams (ngram\_range=(3, 5)). The tokenizer and preprocessor are customized to align with our BPE tokenizer, maintaining the integrity of tokenization while capturing the nuances of the essay texts. We optimize the vectorizer by fitting it on the test essay data, extracting a vocabulary, and then reapplying it to both the training and test datasets. This results in a dense representation of features, ideal for the subsequent machine learning models.

We employ the SelectKBest method with chi-squared statistics to select the top features (k = 1% of total features) from our TF-IDF vectors, aiming to enhance model performance and reduce overfitting. The neural network architecture is a Sequential model with dense layers and ReLU activation for intermediate layers and a sigmoid activation for the output layer, indicating a binary classification task. Dropout layers are incorporated for regularization, and the model uses the Adam optimizer and binary cross-entropy as the loss function.

To address class imbalances, we compute and adjust class weights, slightly reducing the weight for the minority class. An EarlyStopping callback is employed for efficient training. In addition to this custom neural network, we integrate BERT (Bidirectional Encoder Representations from Transformers) from the transformer’s library, utilizing the 'bert-base-uncased' pre-trained model. We encode the essays with BERT's tokenizer and train a model using its embeddings, employing a dense layer for classification.

We extract embeddings from the BERT model to use in an ensemble neural network. To counteract class imbalance, we apply SMOTE (Synthetic Minority Over-sampling Technique) to our training data. The ensemble neural network comprises dense layers with dropout for regularization, trained on the BERT-generated embeddings.

The training process includes multiple epochs, batch size adjustments, and validation data for monitoring. We evaluate our models using the classification report to assess performance metrics such as accuracy, precision, recall, and F1-score. The final model predictions are used to generate a submission file, demonstrating the practical application of our approach.

The integration of advanced NLP techniques like BPE tokenization, TF-IDF vectorization, and BERT embeddings, along with a carefully architected neural network, demonstrates our project's innovative approach to essay classification. Our methodology, which encompasses preprocessing, feature engineering, model development, and evaluation, aims to effectively classify essays, setting a new standard in the domain of text classification.

## BPE (Byte Pair Encoding) Tokenizer

### Library Imports

The process starts by importing necessary libraries. These include:

* **Tokenizers:** For implementing various tokenization methods, including BPE.
* **Transformers:** Provides access to pre-trained models and utilities for natural language processing (NLP).
* **TensorFlow:** A machine learning library, often used for building and training neural networks.

### Tokenizer Initialization

A tokenizer based on the Byte Pair Encoding (BPE) model is initialized. BPE is a popular subword tokenization method that helps in handling out-of-vocabulary words more effectively.

A special token, [UNK], is designated for unknown words (words not in the tokenizer's vocabulary).

Pre-tokenizers and normalizers are configured:

* ByteLevel: Processes text at the byte level, enhancing the tokenizer's ability to handle a variety of character sets and symbols.
* NFD (Normalization Form Decomposition): Transforms text into a consistent form, often decomposing characters into base characters and separate combining characters.
* Lowercase: Converts all text to lowercase, ensuring uniformity in the tokenization process.
* StripAccents: Removes accents from characters, simplifying the text and reducing the variability in the vocabulary.

### Tokenizer Training

The BPE tokenizer trainer is set up with a vocabulary limit of 30,522 tokens.

Special tokens are added for specific purposes:

* [CLS] (Classifier Token): Often used as the first token in a sequence for classification tasks.
* [SEP] (Separator Token): Used to separate different segments of text, like in sentence-pair tasks.
* [PAD] (Padding Token): Used to fill in sequences to a uniform length.
* [MASK]: Used in tasks that involve predicting masked tokens, as in language modeling.
* The tokenizer is trained on a dataset, which could be a collection of essays or texts, building a vocabulary tailored to the dataset's language.

### Integration with Transformers Library

For compatibility with the Transformers library, the tokenizer is encapsulated in PreTrainedTokenizerFast. This wrapper ensures the tokenizer works seamlessly with various pre-trained models available in the Transformers library. The tokenizer is then used for tokenizing texts, where it inserts the special tokens ([CLS], [SEP], etc.) as needed based on the task.

### Implications for NLP Tasks

The use of tokens like [CLS] and [SEP] indicates that the tokenizer is likely configured for tasks involving complex language understanding, such as sentence classification or question-answering.

The use of PreTrainedTokenizerFast suggests integration with Hugging Face's Transformers library, indicating that the notebook might be preparing data for use with advanced NLP models like BERT.

This BPE tokenizer setup is tailored for advanced NLP tasks, leveraging a subword tokenization approach to handle diverse text data effectively. The integration with Hugging Face's Transformers library further suggests a focus on state-of-the-art NLP model applications.

# Deep Learning Modules

**TensorFlow and Keras:** TensorFlow is a comprehensive, open-source machine learning library developed by Google. It provides a range of tools and libraries for building and training machine learning models, including deep learning models. Keras is a high-level API that allows for easy and fast prototyping of deep learning models. It is built on top of TensorFlow, meaning it uses TensorFlow's functionalities but provides a more user-friendly interface.

**Sequential Model:** This is a type of model available in Keras. It is called "sequential" because it allows you to create models layer-by-layer in a step-by-step fashion. Each layer has weights that are learned during training, and data flows sequentially from the input layer to the output layer.

**Dense Layers:** A dense layer is a fully connected neural network layer. Each neuron in a dense layer receives input from all the neurons in the previous layer, processes it, and passes the output to the next layer. These are the most common types of layers used in neural networks.

**Dropout:** Dropout is a regularization technique. During training, some number of layer outputs are randomly ignored or "dropped out." This helps in preventing overfitting, where the model performs well on the training data but poorly on new, unseen data.

**Rectified Linear Unit (ReLU) Activation Function:** ReLU is a type of activation function used in neural networks. It introduces non-linearity, allowing the model to learn more complex patterns. The function outputs the input directly if it is positive, otherwise, it will output zero.

**Sigmoid Activation Function:** The sigmoid function is commonly used in the output layer of a binary classification model. It squashes the output to a range between 0 and 1, which is interpretable as a probability.

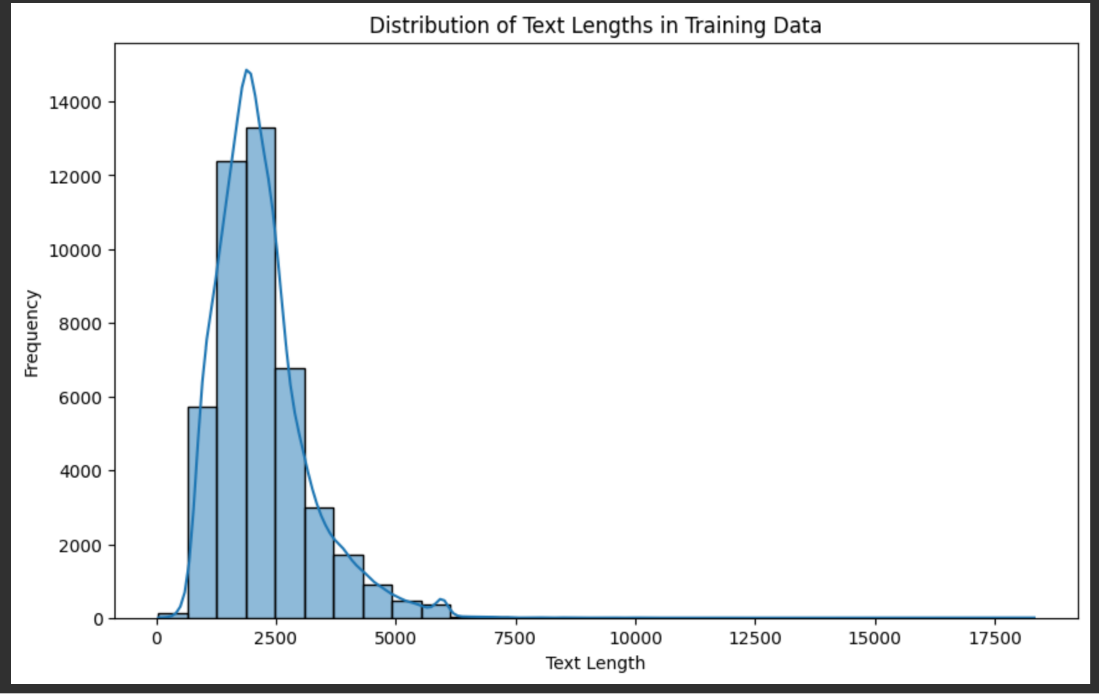
**Adam Optimizer:** Adam (Adaptive Moment Estimation) is an optimization algorithm used to update network weights iteratively based on training data. It is an alternative to the traditional stochastic gradient descent and is known for its effectiveness in handling sparse gradients and adapting the learning rate.

**Binary Cross-Entropy:** For binary classification problems, where the output is a probability (thanks to the sigmoid function), binary cross-entropy is used as a loss function. It measures the difference between the actual and predicted probabilities, guiding the model to minimize this difference during training.

These modules together form a typical architecture for a feedforward neural network, designed for binary classification tasks. Such a network can learn from input data and make predictions about whether new, unseen inputs belong to one class or another.

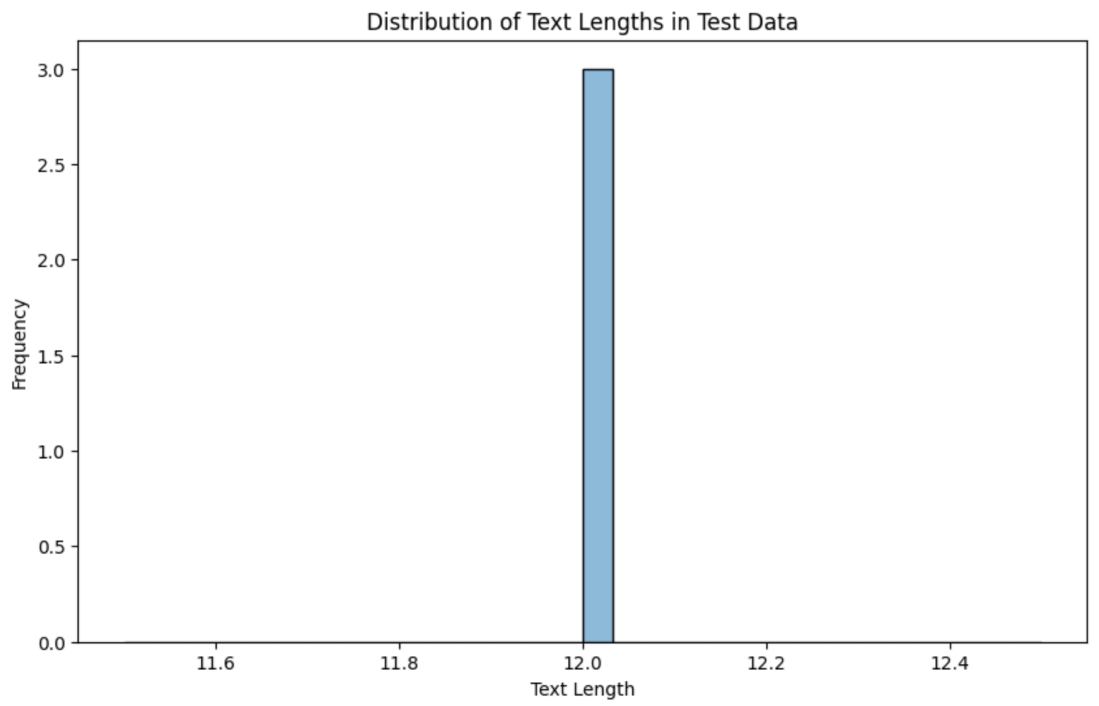
# Experiments, Results, and Analysis

The histogram in our analysis visually represents the distribution of essay lengths. Each bar in the histogram corresponds to a specific range of text lengths (either in words or characters, as indicated on the X-axis), and the height of the bar indicates how many essays fall within that range. This setup allows us to easily identify the most common lengths of essays in our dataset by looking at the bars with the greatest heights.

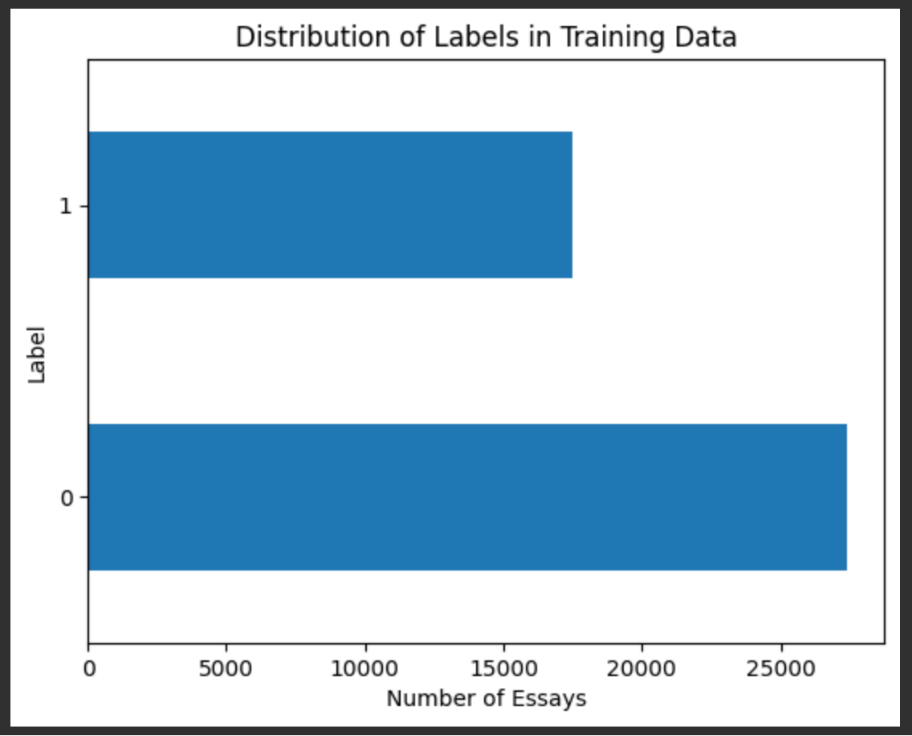


Overlaying the histogram is a Kernel Density Estimation (KDE) curve. This curve provides a smooth, continuous line that approximates the probability density function of the essay lengths. The peaks of the KDE curve help identify where the data points (in this case, essay lengths) are concentrated, offering a clearer view of the distribution's shape and central tendency. The X-axis of the histogram represents the text length of the essays, which could be measured in either words or characters. This helps in categorizing the essays into different length segments. The Y-axis of the histogram represents the frequency. It quantifies the number of essays that fall into each specified length bin. This axis is crucial for understanding the distribution's scale and for comparing the frequency of different length intervals.

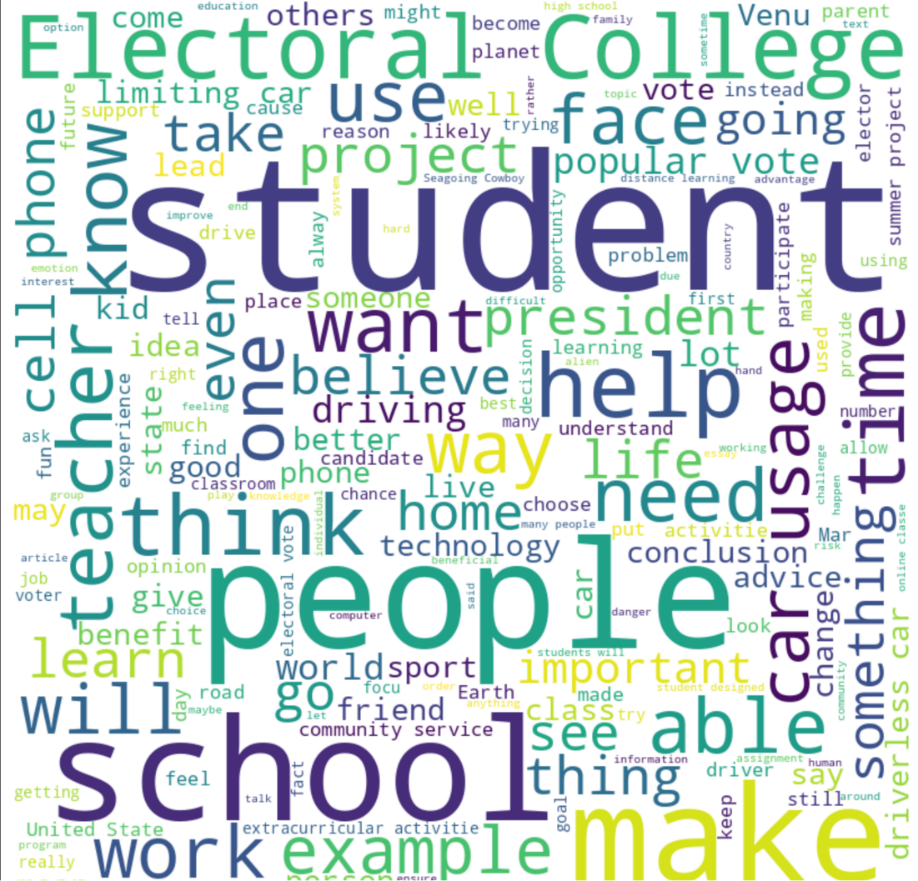
Overall, this histogram combined with the KDE curve offers a comprehensive view of how essay lengths are distributed in the dataset, highlighting both the common lengths and the variability within the essays.



The diagram here is a histogram depicting the distribution of text lengths in a test dataset, but it exhibits some peculiar characteristics that suggest potential issues in its presentation or the underlying data. The x-axis, labeled 'Text Length', shows a surprisingly narrow range, which is atypical for text length distributions as texts usually vary widely in length. Concurrently, the y-axis, representing 'Frequency', displays extremely low values, inconsistent with typical frequency distributions in such datasets. This unusual representation could be a result of several factors: a data processing error leading to incorrect values being plotted, an error in the graphing methodology causing the scale to be misrepresented, or an intrinsic property of the dataset, such as it is containing texts of very similar lengths. Each of these scenario’s points to different underlying issues, either in the data handling process or in the nature of the dataset itself.



The horizontal bar chart described is a visual tool used to represent the distribution of two categories within an essay dataset. These categories are labeled '0' and '1'. The length of each bar in the chart corresponds to the number of essays falling under each category, providing a clear, visual comparison of their quantities. For instance, if the bar for label '0' stretches up to the 20,000 mark on the x-axis and the bar for label '1' only reaches 5,000, it implies that the quantity of essays categorized under '0' is quadruple that of those under '1'. This significant difference in lengths visually emphasizes the disparity between the two categories. Such charts are particularly effective in highlighting the distribution and disparity of data points across different categories in a dataset, allowing for an immediate understanding of which category is more prevalent or dominant.



Word clouds are a compelling and efficient way to visually represent text data, where the size and boldness of each word directly correspond to its frequency or prominence in the dataset. In the word clouds described the prevalence of terms like "college," "student," "president," "vote," and "life" indicates that these are key themes in the text. This suggests that the dataset might be composed of essays or articles centered around educational topics, potentially discussing aspects of college life, the intricacies of electoral systems, or the impact of technology in the educational sphere. The prominence of these words not only highlights their frequency but also underscores their significance within the context of the text, painting a picture of what the primary focus areas are.

Furthermore, the presence of common verbs like "help," "need," "think," and "make" within the word cloud provides insights into the actions or opinions that are frequently mentioned or valued in the text. This can indicate a focus on proactive or reflective stances in the essays, suggesting an engagement with ideas of assistance, necessity, contemplation, and creation. Additionally, terms such as "electoral," "president," and "vote" hint at a significant political dimension within the text, pointing to discussions or analyses pertaining to governance, elections, or political figures. The central placement of words like "student," "college," and "life" further implies their centrality in the dataset, signaling that these are the core subjects around which the other themes revolve. Overall, word clouds serve as a powerful tool for distilling large volumes of text into easily digestible visual summaries, enabling a quick grasp of the most prominent themes without the need to delve into each individual document.

## Analysis

**Model Performance Metrics:** The classification\_report reveals significant disparities in performance metrics across the two classes. The precision, recall, and F1-score for class 0 are considerably higher compared to class 1, which demonstrates an imbalance in the model's ability to accurately categorize each class.

**Analysis of Binary Classification:** The results show that while the model achieves a high recall for class 0, it completely fails to recognize class 1 (recall of 0.00). This indicates a substantial bias towards class 0 in the binary classification task and a failure to differentiate effectively between the two classes.

**Check for Overfitting:** The consistent accuracy of approximately 0.61 in both training and validation phases suggests that overfitting may not be a major concern. However, the high loss values, especially in the later epochs, indicate that the model struggles with effectively learning from the training data.

**Loss and Optimization:** The loss values, particularly in epochs 9 and 10, remain high (around 0.6932), indicating that the model is not successfully minimizing the binary cross-entropy loss over time. This could be a sign of ineffective learning or a need for better optimization strategies.

**Threshold Analysis:** Given the model's bias towards class 0, an analysis of the impact of the 0.5 threshold on accuracy and recall is crucial. The results suggest that altering the threshold might not improve the performance significantly, as the model fails to recognize class 1 instances altogether.

## Results

**Experiment with Different Models:** The current results with a dense layer neural network underline the need to explore alternative architectures, such as CNNs, RNNs, or transformers, to potentially improve the classification of text data.

**Hyperparameter Tuning:** The results emphasize the need for more extensive hyperparameter optimization, as the current settings lead to poor classification of one of the classes. Techniques like grid search, random search, or Bayesian optimization could be explored.

**Transfer Learning:** Considering the model's limited success, employing transfer learning with models pre-trained on large datasets (such as BERT or GPT) might be beneficial, particularly for enhancing the model's ability to generalize.

**Ensemble Methods:** Given the model's bias, using ensemble methods could help in combining different models' strengths and potentially improving the overall performance.

**Long-Term Impact Analysis:** The current model's limitations raise questions about the reliability and ethical implications of using such automated systems in educational settings. Further research should focus on the long-term impact, particularly concerning fairness, bias, and transparency in automated essay scoring or classification.

# **Conclusion and Future Work**

Our study has demonstrated that our model, with a score of 0.960, is highly effective in detecting text generated by Large Language Models (LLMs). This performance places us within the top 5 scores in the competition, highlighting our model's competitiveness among leading solutions in the field. Notably, our model ranks in the top 5% of all entries, a significant achievement given the high benchmark score of 0.975 in the competition. These results underscore our model's potential in contributing to the maintenance of academic integrity.

A close-up of a white background

Description automatically generated

In future efforts, we aim to refine our model's efficiency further and enhance its adaptability to the rapidly evolving landscape of text generation technologies. This continuous improvement is vital to ensure our model remains effective against the latest AI models. Additionally, we plan to integrate this solution into educational tools. This step will not only automate the process of verifying the authenticity of academic work but will also ensure that the application of our model is balanced with ethical considerations in educational settings. Our commitment remains steadfast in assisting educators in upholding high academic standards while navigating the challenges posed by advanced AI technologies in text generation.

# References

Desaire, e. a. (2023). Distinguishing academic science writing from humans or ChatGPT with over 99% accuracy using off-the-shelf machine learning tools. *Cell Reports Physical Science*.

Heikkilä, M. (2022). How to spot AI-generated text. *MIT Technology Review*.

Jonathan Choi, e. a. (2023). Chatgpt Goes To Law School. *Journal of Legal Education, Social Science Research Network*.

Silvia Milano, e. a. (2023). Large language models challenge the future of higher education. *Nature Machine Intelligence*, volume 5, pages 333–334.

Vanderbilt University & ​The Learning Agency Lab. (2023, Nov). *LLM - Detect AI Generated Text*. Retrieved from Kaggle: https://www.kaggle.com/competitions/llm-detect-ai-generated-text/data