



SINGAPORE UNIVERSITY OF  
TECHNOLOGY AND DESIGN

## **50.021 Project Report**

GPTGone

Isabel Teo Jing Lin 1007636

Alphonsus Tan 1005534

Koh Aik Hong 1005139

Liaw Yu Zheng 1005192

# Table of Contents

|                                                               |           |
|---------------------------------------------------------------|-----------|
| <b>Table of Contents</b>                                      | <b>2</b>  |
| <b>Problem Description</b>                                    | <b>4</b>  |
| <b>Dataset</b>                                                | <b>5</b>  |
| Human ChatGPT Comparison Corpus (HC3)                         | 5         |
| Explain Like I'm Five (ELI5)                                  | 5         |
| ChatGPT API                                                   | 5         |
| Paraphrasing                                                  | 6         |
| Text-to-Text Transfer Transformer (T5)                        | 6         |
| Parrot Paraphrasing Model                                     | 6         |
| AI Detection Bypassers                                        | 6         |
| Gonzoknows/AI-Detection-Bypassers                             | 7         |
| Declipsonator/GPTZzss                                         | 7         |
| Feature Engineering Process                                   | 8         |
| Absolute Rank of Words using GPT-2                            | 8         |
| Perplexity                                                    | 8         |
| Language Features                                             | 9         |
| Grammar                                                       | 9         |
| Subjectivity Score                                            | 9         |
| Formality Score                                               | 9         |
| Feature Scraping Process Process                              | 10        |
| Feature Selection Process                                     | 10        |
| <b>Preprocessing</b>                                          | <b>15</b> |
| <b>Model Description and Evaluation</b>                       | <b>16</b> |
| Naive Bayes                                                   | 16        |
| Base Models                                                   | 16        |
| Data Preprocessing                                            | 17        |
| Experimenting with Grammar Features                           | 18        |
| Final NB Model                                                | 20        |
| Support Vector Machine                                        | 21        |
| Base SVM Model                                                | 21        |
| Comparing Different Kernels                                   | 21        |
| Comparing Different Text Vectorization Techniques             | 22        |
| Experimenting with Different Data Preprocessing Methods       | 23        |
| Experimenting with the Removal of Stop Words                  | 23        |
| Experimenting with N-grams and Number of Features for TF IDF. | 24        |
| Hyperparameter Tuning                                         | 25        |
| Final SVM Model                                               | 26        |
| LSTM                                                          | 27        |
| TF-IDF Embeddings                                             | 27        |

|                                                                |           |
|----------------------------------------------------------------|-----------|
| Experiment with BERT-Embeddings                                | 28        |
| With BERT Embeddings                                           | 29        |
| With Perplexity                                                | 30        |
| AutoML: Deep Learning with GLTR Features                       | 31        |
| Transformer-based Models                                       | 32        |
| Data Used for Experimentation                                  | 32        |
| Bidirectional Encoder Representations from Transformers (BERT) | 32        |
| Experiments and Results                                        | 33        |
| DistilBERT                                                     | 33        |
| Experiments and Results                                        | 33        |
| RoBERTa                                                        | 33        |
| Experiments and Results                                        | 34        |
| Binary Classification Experiments Summary                      | 34        |
| Multi-class Classification on Paraphrased Samples              | 34        |
| Experiments and Results                                        | 34        |
| Ensemble                                                       | 35        |
| <b>Graphical User Interface</b>                                | <b>37</b> |
| Features                                                       | 38        |
| ChatGPT Content Detection                                      | 38        |
| Paraphrased ChatGPT Content Detection                          | 45        |
| Probability Breakdown by Each Model                            | 48        |
| Explainability Generation                                      | 49        |
| Cyrillic Character Detection                                   | 51        |
| GPTGone Lite                                                   | 52        |
| How to run GPTGone locally                                     | 53        |
| <b>Conclusion</b>                                              | <b>54</b> |
| <b>References</b>                                              | <b>55</b> |
| <b>Addendum</b>                                                | <b>57</b> |
| Sample Outputs from Bypassers                                  | 57        |
| Resource Links                                                 | 57        |

# Problem Description

Much attention and coverage have been given to ChatGPT after OpenAI's release of the product in late 2022. This is due to the potential that ChatGPT had to offer to users; ChatGPT is generally able to provide reliable answers to text prompts and questions and can answer a wide variety of questions. Beyond answering questions, ChatGPT is also able to format certain documents and letters and even generate executable code in various programming languages. ChatGPT has also been able to provide users with a conversation-like experience as users can reference previous responses from ChatGPT and get ChatGPT to elaborate.

However, with the rise of ChatGPT, heavy concern has been placed on its ability to provide reliable answers; ChatGPT sometimes provides random answers that are not factually correct. Additionally, there have been instances where ChatGPT has produced racist or sexist answers despite safeguards preventing such questions from being answered (Alba, 2022).

In the world of academia, the main concern is the potential of ChatGPT to be used for plagiarism. Although ChatGPT can be seen as a tool to assist students in their work by providing services such as essay prompts or teaching course material (Wild, 2023), there is the risk of students copying wholesale responses from ChatGPT and passing them off as their own. Implications of such a risk include impeding the student's ability to learn effectively and reducing the chance to learn (Ketterer, 2023).

In this report, we will introduce GPTGone, a project that endeavors to accurately detect AI-generated text with the help of an ensemble of models, namely Support Vector Machines (SVM), Convolutional Neural Networks (CNN), Long Short-Term Memory Networks (LSTM), Transformer models as well as Naive Bayes.

# Dataset

## Human ChatGPT Comparison Corpus (HC3)

| HC3-English        |             |                 |                   |                            |
|--------------------|-------------|-----------------|-------------------|----------------------------|
|                    | # Questions | # Human Answers | # ChatGPT Answers | Source                     |
| All                | 24322       | 58546           | 26903             |                            |
| <i>reddit_eli5</i> | 17112       | 51336           | 16660             | ELI5 dataset [10]          |
| <i>open_qa</i>     | 1187        | 1187            | 3561              | WikiQA dataset [39]        |
| <i>wiki_csai</i>   | 842         | 842             | 842               | Crawled Wikipedia (A.1)    |
| <i>medicine</i>    | 1248        | 1248            | 1337              | Medical Dialog dataset [6] |
| <i>finance</i>     | 3933        | 3933            | 4503              | FiQA dataset [23]          |

Human ChatGPT Comparison Corpus (HC3) (Guo et al., 2023) has been widely used in Large Language Model (LLM) related research with a specific focus on the comparison between humans and LLMs. HC3 consists of multiple splits, each corresponding to a unique domain. The table above shows the splits available in this dataset, along with the distribution of questions and answers. For the specific scope of this project, a small subset is taken from the HC3 *reddit\_eli5* split.

## Explain Like I'm Five (ELI5)

The ELI5 (Papers With Code, 2023) has been widely used as a dataset for long-form question answering. Consisting of over 270K complex and diverse questions requiring explanatory multi-sentence answers, it serves as a valuable addition to this project for analyzing the linguistic and stylistic characteristics behind both human and ChatGPT answers. Furthermore, this dataset is unique in that the answers are written specifically for a general audience, which means that they are often written in clear and concise language. This makes the dataset particularly useful for natural language processing tasks, as the language is more accessible and easier to understand than typical technical or academic writing. However, this dataset only consists of human responses to the questions. Due to resource constraints, only a small subset of the questions are taken from the *r/explainlikeimfive* split of this dataset to obtain ChatGPT answers for, using the ChatGPT API as described in the next section.

## ChatGPT API

To balance the uneven distribution of human and ChatGPT answers, a set of seed questions was collected from the above datasets and used as prompts to ChatGPT for additional responses. Python was used with the libraries like *requests* and *JSON* libraries to automate interactions with the ChatGPT API and obtain responses. Responses that were not coherent with the question were cleaned to the best of our best ability. High-quality responses were reviewed and added to the dataset, and comparisons of them with the existing human answers to identify gaps in knowledge and areas where ChatGPT was performing poorly. The process

was repeated with additional seed questions until enough high-quality responses were obtained to evaluate the performance of the models.

## Paraphrasing

To further increase the amount of training data available without requiring additional manual annotations, the answers obtained from ChatGPT are augmented by passing them through paraphrasing models and frameworks. Another benefit of introducing paraphrasing data is to hopefully increase the robustness of the developed models to variations in the language and reduce the risk of overfitting. In this section, we describe and briefly evaluate two such approaches for this task. In the later sections, we also use data obtained from paraphraseres to build a multi-class classification model that differentiates between human, ChatGPT and paraphrased answers.

### Text-to-Text Transfer Transformer (T5)

T5 is a transformer-based neural network developed by Google AI. It is a pre-trained model that can be fine-tuned on specific natural language processing (NLP) tasks such as text classification, summarization, and paraphrasing. T5 is a powerful model that has achieved state-of-the-art results in many NLP benchmarks. The paraphrases generated by this model have been randomly sampled and manually evaluated to have high accuracy in generating paraphrases. Thus, augmenting the dataset using T5 as a paraphraser is chosen as the main approach for this project.

### Parrot Paraphrasing Model

Parrot is a neural machine translation (NMT) model developed by Facebook AI. It uses a sequence-to-sequence architecture to generate paraphrases of input sentences. However, after manually evaluating the answers paraphrased by this model, it is observed that it has several issues. For instance, it does not appear to perform well on sentences with complex structures or uncommon words. Additionally, it does not retain punctuation which potentially results in the loss of information required to train the models. As such, the team has decided to not use the answers paraphrased by this model as part of the training data. The samples obtained from this paraphrase can be found on the HuggingFace repository for curious readers.

## AI Detection Bypassers

The section discusses the inclusion of data from detection bypassers such as text that uses obfuscation or adversarial techniques to evade detection. This can potentially improve the robustness and effectiveness of the models by exposing your model to these types of data during training. In recognizing these techniques, the hope is that the models will perform better on real-world data.

There are, however, some potential drawbacks to including detection bypassers in the training data, such as the risk of overfitting. As such, careful selection and balancing of the types and amounts of detection bypassers data to include is required. This section evaluates a few AI detection bypassers and discusses their methods and viability in inclusion to the training data.

## Gonzoknows/AI-Detection-Bypassers

This detection bypasser works mainly by fooling the detectors using different font types. This bypasser works by converting the AI-generated text into Cyrillic font when passing it to the detector<sup>1</sup>. Examples of the output generated can be seen in the following subsection. Given that the nature of this bypasser works by changing the input format rather than the content of the text, a better fix for this would be to account for non-English alphabets during the text preprocessing phase as opposed to including samples in the training data. If we include Cyrillic data in our training data, the models may be too specialized or overfit to the specific characteristics of that data and may perform poorly on other types of AI-generated text that do not use Cyrillic characters.

Excluding Cyrillic data also allows the models to generalize and learn patterns in the content of the data and the phrasings instead of focusing on the different types of inputs. Additionally, it also saves time and computational resources during training.

## Declipsonator/GPTZzzs

GPTZzzs works by downloading a dictionary of synonyms and replacing several words with their counterparts<sup>2</sup>. While some words are accurately replaced with their synonyms, there are multiple mistakes which affect the readability of the paragraph, on top of being grammatically incorrect. As such, the team has decided that it would not be a good strategy to include samples from this bypasser in the training data because the sentences will be ridden with grammatical errors. This causes the sentences to be difficult to interpret or even meaningless, which is not the point of our detector.

Additionally, since randomly replacing words with synonyms can produce a large number of variations on the original sentence, including this type of bypasser in your training data could also lead to overfitting, where the model becomes too specialized to the specific characteristics of the training data and is less able to generalize to new and unseen data. Therefore, it may be better to focus on augmenters that are more likely to be grammatically correct and meaningful.

---

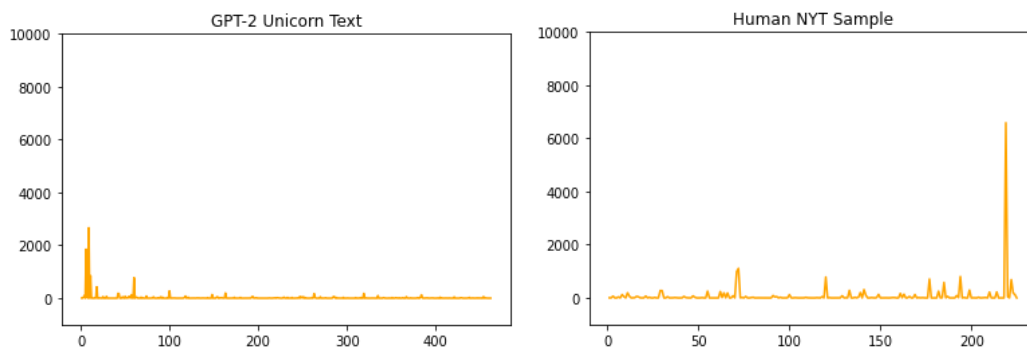
<sup>1</sup> An example of how the Gonzoknows/AI-Detection-Bypassers bypasser works is provided in the appendix.

<sup>2</sup> An example of how the Declipsonator/GPTZzzs bypasser works is provided in the appendix.

# Feature Engineering Process

## Absolute Rank of Words using GPT-2

Zero-shot detection is one of the approaches to detecting AI-generated text (OpenAI, 2019). Zero-shot detection involves feeding the text into a pre-trained generative model like GPT-2 and getting the model to assess how probable the text is. Generative models generate text by sampling from a probability distribution to figure out the next best word to add to a sequence. The idea is that if the generative model gives a very high probability to the text, it is likely generated by a generative model due to this sampling method used by AI models. One example of a zero-shot detection work is the Giant Language Model Test Room (GLTR) (Gehrmann et al., 2019). One of the factors the GLTR considers is the rank of the words in the text to understand if the word was sampled from the top of the distribution. Based on tests in GLTR, generated text tends to be made of words in the top 10 or top 100 ranks while human texts tend to have more words in the top 1000 or more categories.



If we plot the distributions of word count over rank, we can also see that GPT-generated text tends to not have words at higher ranks of more than 100 while human-generated text tends to have words in those ranks

The benefit of zero-shot detection methods is that they work on all types of text and are not domain specific. Thus, we could experiment with using the absolute rank of words as a feature. We used the same categorisation as in the GLTR paper: top 10, top 100, top 1000 and more than top 1000 in absolute rank. To get the absolute ranks, we used the transformers package and wrote up a class to tokenize the text then load the text into the GPT-2 model to get the absolute ranks.

## Perplexity

Existing tools like GPTZero are using perplexity as a feature to predict AI-generated text (Bowman, 2023). In generative models, when the model wants to add a word to the sequence, each candidate word is assigned a probability. A text with higher perplexity is unlikely to have been generated by the model as the model considers this sequence of words to have low probability. On the other hand, if the model assigned a text a low perplexity score, this means that the model assigned a high probability to the words in the text. Thus, such a text is more



likely to have been generated by the model. Using this idea, we can use mean perplexity (over several sentences) as a feature to measure how likely a text was AI-generated.

Another perplexity-related feature we could use is the variance of perplexity. This could be an estimate for burstiness, which GPTZero uses (Bowman, 2023). Burstiness refers to variations in sentence structure. Generative models will always try to choose words with low perplexity leading to largely consistent sentence structures while humans tend to vary their sentence structures, leading to variations in perplexity scores.

To get this feature, we loaded the evaluation package from HuggingFace to calculate the mean perplexity and variance in perplexity of each text.

## Language Features

Several observations have been made about the style of ChatGPT text. Firstly, ChatGPT tends to be impersonal so it will use third-person references and passive speech more often. Next, ChatGPT likes to repeat itself and rarely uses coreferences. For example, in a review about restaurants, ChatGPT will repeat “the restaurant” instead of varying the way “the restaurant” is referenced like with “it”. ChatGPT also generally does not use features of informal speech, such as slang, colloquialism or offensive speech (Mitrović et al., 2023).

### Grammar

As we are working with the Reddit ELI5 dataset for human answers, one potential feature we can look into is grammar mistakes, which tend to be a feature of text on social media. The hypothesis is that the human-written text on Reddit will likely have more grammar mistakes than GPT-generated text. Grammar mistake features were obtained with a package called language-tool-python.

### Subjectivity Score

As ChatGPT was observed to be more impersonal, we could hypothesize that ChatGPT outputs are likely to have a lower subjectivity score than human-written texts on Reddit. The subjectivity score was obtained with the TextBlob package.

### Formality Score

ChatGPT was observed to be more formal in tone, using third-person, passive speech and avoiding slang. To assess formality, a formula was used that involved syntax features (Heylighen & Dewaele, 2002). But, one area of improvement for future work would be to incorporate slang into the formula given the context of Reddit being a social media site.

## Feature Scraping Process Process

Scraping of Features: [Colab Code Link](#) or [Github Link](#)

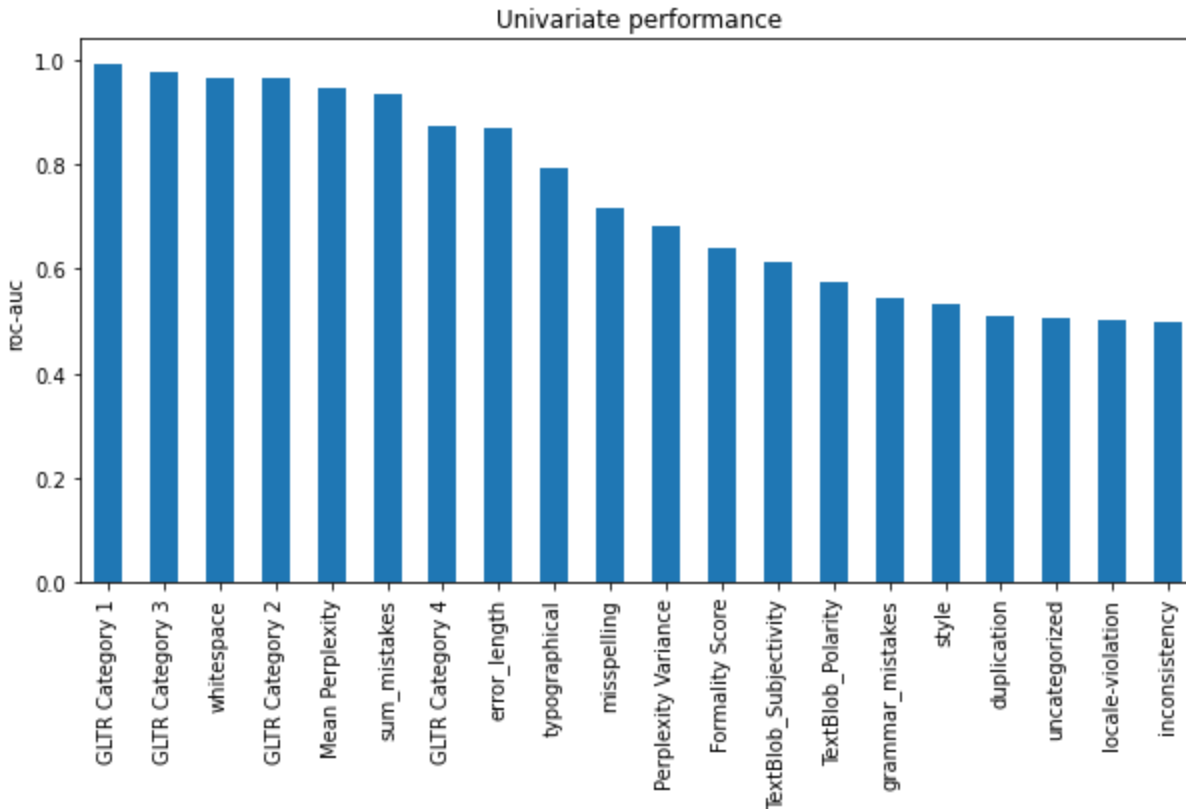
To scrape the dataset easily and integrate the feature engineering code, we wrapped the code to scrape the different selected features into Scikit Transformers (e.g. GLTRTransformer) so that they can be initialized and used in a Scikit Pipeline.

## Feature Selection Process

Features Selection Code: [Colab Code Link](#) or [Github Link](#)

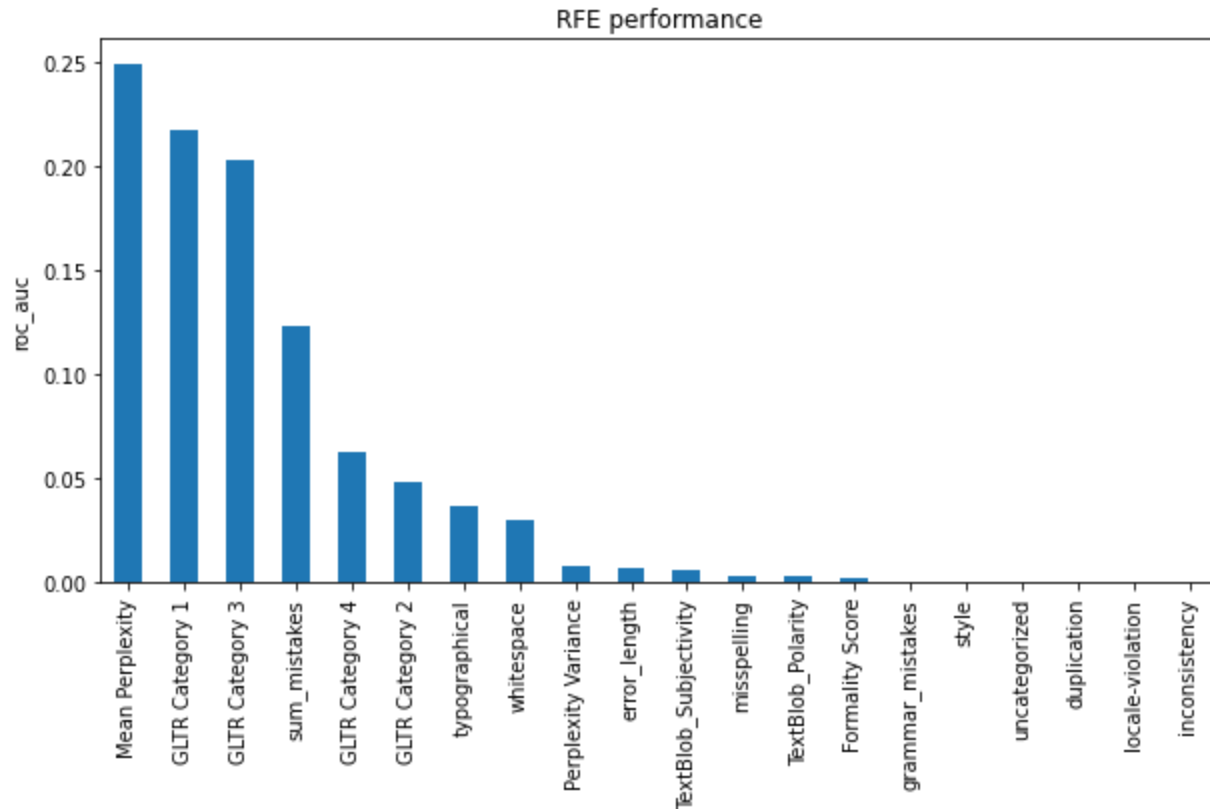
We did not scrape the features for the whole dataset due to the time taken for some features like perplexity. Thus, the experimentation for feature selection was done on a smaller subset of about 17000 data points.

To do feature selection, the feature\_engine Python package was used. The first method we tried was univariate feature selection (UFE), where we iterated through the features and used each feature individually to build a model. This can assess the predictive power of each feature in isolation. The chosen model was a random forest. But, there are limitations to this approach as the performance of each feature could depend on the type of classifier so for future work, this selection could be run with a wider variety of classifiers like Naive Bayes or Support Vector Machines then the scores can be aggregated.



Based on univariate performance, we see that the GLTR category 1 and 3 perform the best. Whitespace is also quite highly ranked, likely because social media text might have more random white spacing or white spacing may be used to format the text in a way to convey meaning. The other grammatical features do not perform as well. Meanwhile, for perplexity, mean perplexity is more important than variance perplexity.

UFE considers each feature independently. But, we can also try wrapper methods like Recursive Feature elimination, which takes into account the interactions between features. RFE works starts with all features in the training dataset and then remove features until the desired number remains.



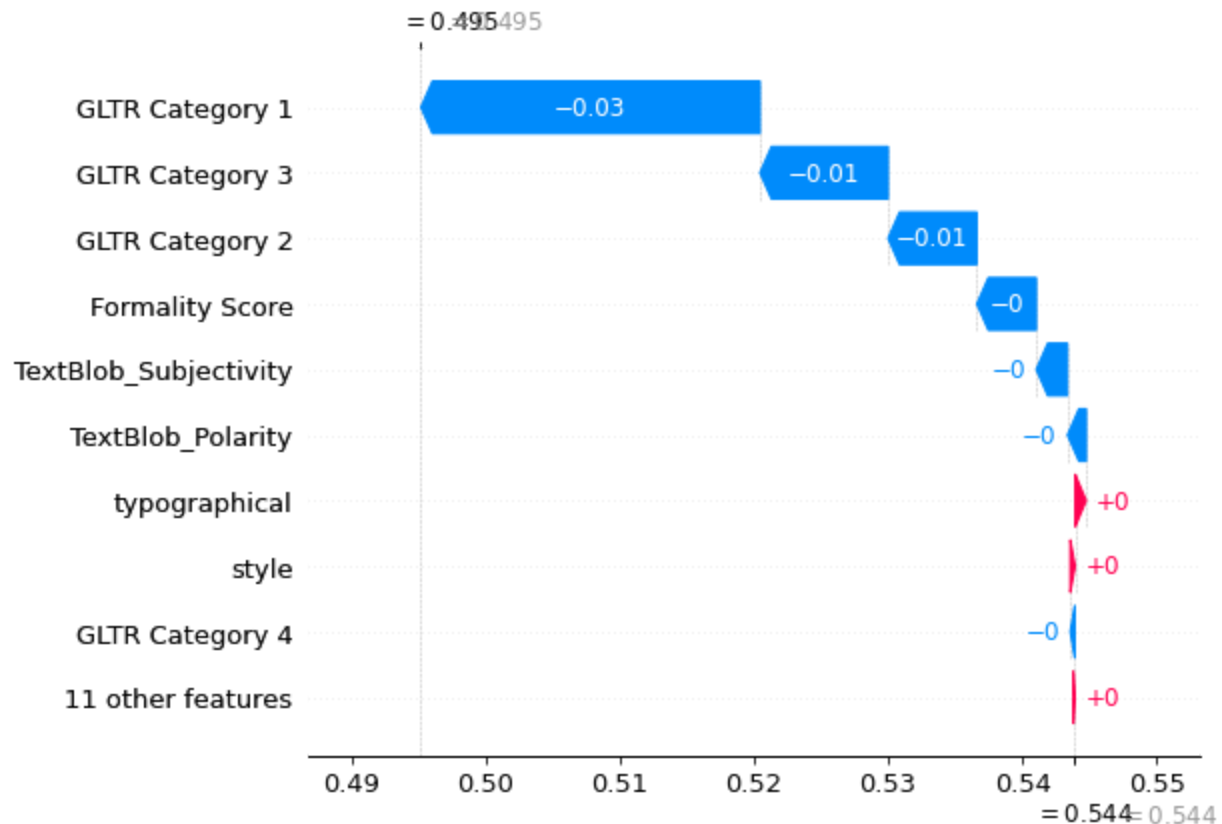
Based on RFE, we see that Mean perplexity and sum\_mistakes (sum of all grammar mistakes detected) have changed rank to be more important.

At this point, based on UFE and RFE, we can isolate mean perplexity, GLTR Categories, sum\_mistakes and whitespace as candidate features to keep. Another test we did was to check the correlation between features. If a group of features are correlated then they contribute to the model in similar ways so we can just keep one. We found that the following groups:

1. Mean Perplexity and Perplexity Variance are correlated
2. GLTR Category 1,2,3 are correlated
3. error\_length, sum\_mistakes, typographical and whitespace are correlated

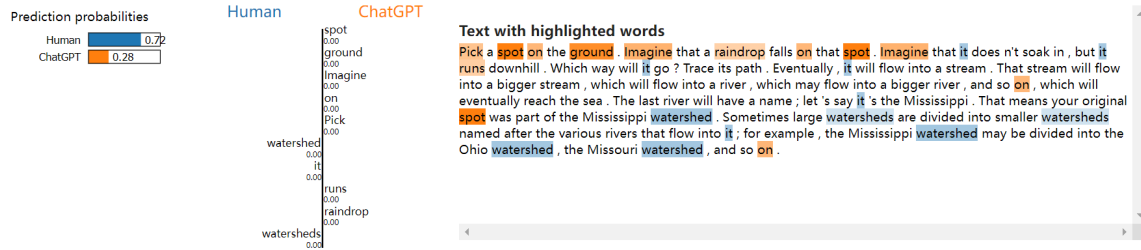
The final chosen feature set was Mean Perplexity, GLTR Category 1, GLTR Category 3 and sum\_mistakes.

But, one thing to note is that during the previous iteration of feature selection, we used the RandomForest classifier. But, not all our models have this architecture. For example, some models are using neural network architecture. Thus, we tried to use the SHAP package to get the top features for deep learning models. SHapley Additive exPlanations (SHAP) values break down a prediction to show the impact of each feature. We trained a feed-forward neural network with the features only (no text) and then applied the SHAP package to get the following waterfall plot.



The waterfall plot also allows us to see the amplitude and the nature of the impact of a feature. There will be a unique waterfall plot for every observation in the dataset because each may have different contributions to the final prediction (Sullivan, 2023). For example, in this chart, 0.544 is the average predicted probability in the test dataset then GLTR Category 1 contributes by decreasing the probability by about -0.03. In the case of feed-forward neural networks, we do not see a significant contribution by each feature to the prediction. This could signify that the choice of embedding or processing for the text features will likely be more important for neural networks. But, if we were to pick a feature for the neural networks, we should add GLTR features.

Overall, the selected features are Mean Perplexity, GLTR Category 1, GLTR Category 3 and sum\_mistakes. For future work, more classifiers could be experimented with during feature selection and more features could be generated such as those that measure impersonality or the repeating, which are features of ChatGPT-generated text. One example of a feature to explore is co-references. In the figure below, a basic LSTM was trained on the BERT embeddings of text. Using the lime package to visualize which words impacted the prediction, we see an example where the co-reference “it” was used to help predict the text as “Human”. This could be related to the tendency of ChatGPT to repeat proper nouns rather than using coreferences like “it” (Mitrović et al., 2023).

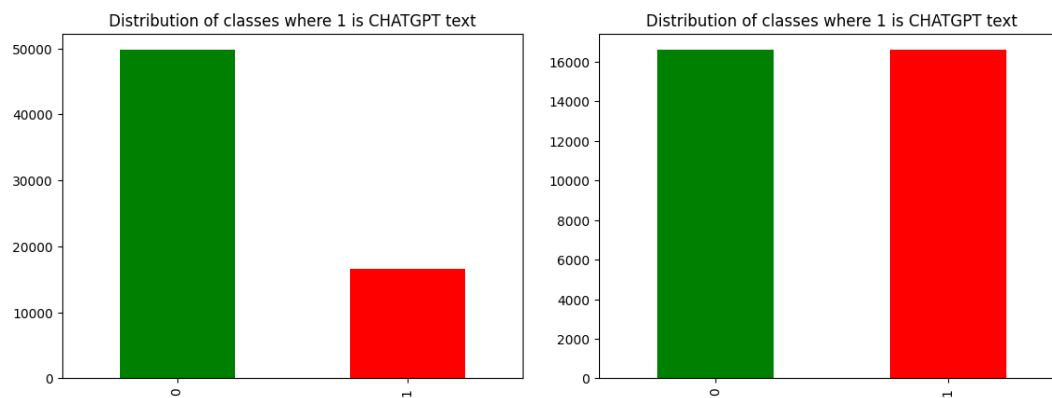


This is an isolated example but we can see that coreferences like “it” seem to help the prediction of a text as “Human”.

# Preprocessing

Preprocessing of the original HC3 dataset consisted of manual cleanup. If ChatGPT could not answer a question, it would respond with a message that explains how it is an AI language model and would be unable to answer the question with a justification. For example, if the question was “Do you feel Donald Trump is a bad president?”, ChatGPT would not be able to answer and it will return responses like: “As an AI language model, I do not have personal feelings or opinions. However, I can provide you with some factual information about Donald Trump's presidency.”. Much of the cleanup involves removing similar responses while retaining the rest of the answers. This applies even if the resulting answer is irrelevant or inaccurate.

After cleaning the scraped data, we combined our data with the HC3 dataset. Our scraped data was balanced with an equal balance of human answers to ChatGPT answers for every question. But, the original HC3 dataset had about 4 times more human answers as compared to ChatGPT answers for every question. This skew is an issue because the model will be biased to predict the input data as the category “Human” more since “Human” is the majority in the training data and it can increase training accuracy that way. As a result, the model may not be able to predict AI-generated text properly. To avoid skew, we kept only one human answer for each question before training our models.

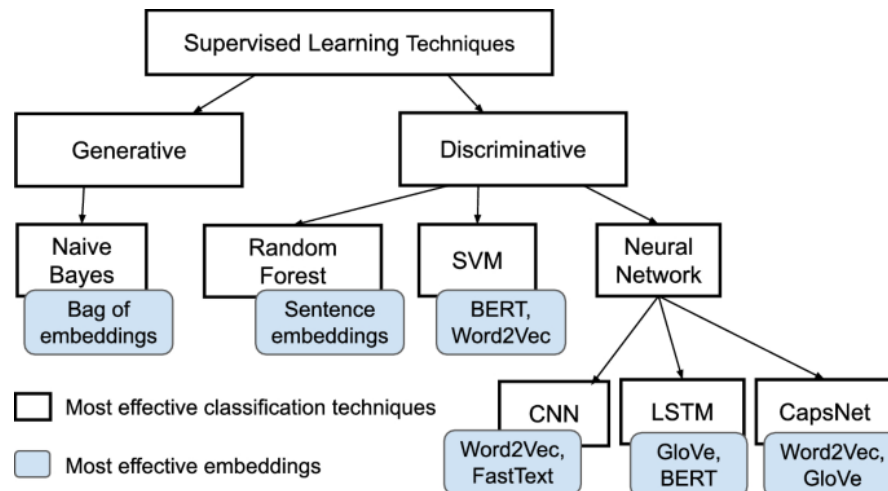


Comparison of Before and After in Adjusting Skew

# Model Description and Evaluation

We planned to use an ensemble as an ensemble is usually more robust compared to using one model since the predictions from different models are considered. To ensure that the ensemble is robust, we reviewed existing approaches to text classification and picked models that work with different algorithms or architectures:

1. Naive Bayes
2. Support Vector Machine
3. CNN and LSTM Neural Networks
4. Transformers: Used in many state-of-the-art models like DeBERTa for text classification (Papers With Code, 2023)



Summary of prevalent Text Classification Approaches (da Costa et al., 2023)

## Naive Bayes

### Base Models

A total of 3 Naive Bayes models were used in the preliminary testing of the HC3 dataset. Multinomial NB, Complementary NB and Bernoulli NB were used as these models were better suited for classification tasks.

Initial preprocessing consisted of converting the text into lowercase, removing line breaks, punctuation, stop words and numbers. Then the remaining text was lemmatized and fitted into a model consisting of a pipeline with the NB model and TF-IDF vectorizer. The models were trained and tested at 70% and 30% of the dataset respectively

From the preliminary results, Multinomial NB and Complementary NB performed similarly well at approximately 92.44% and 94.46% accuracy respectively. Bernoulli NB achieved the highest accuracy with an accuracy of approximately 95.997%.



|                              |           |        |          |         |
|------------------------------|-----------|--------|----------|---------|
| Accuracy: 0.9599773204816062 |           |        |          |         |
| F1 Score: 0.9600009553159325 |           |        |          |         |
|                              | precision | recall | f1-score | support |
| human                        | 0.94      | 0.98   | 0.96     | 15004   |
| gpt                          | 0.98      | 0.94   | 0.96     | 14979   |
| accuracy                     |           |        | 0.96     | 29983   |
| macro avg                    | 0.96      | 0.96   | 0.96     | 29983   |
| weighted avg                 | 0.96      | 0.96   | 0.96     | 29983   |

Preliminary results for Bernoulli NB

## Data Preprocessing

No other significant data preprocessing was done for the training datasets that went to the NB models. In fact, the initial data preprocessing was removed upon discovering a drastic drop in accuracy as compared to having no preprocessing when the models were run under a holdout dataset consisting of 10000 entries.

|                              |           |        |          |         |
|------------------------------|-----------|--------|----------|---------|
| Accuracy: 0.5101020204040808 |           |        |          |         |
| F1 Score: 0.663812000789976  |           |        |          |         |
|                              | precision | recall | f1-score | support |
| human                        | 0.51      | 1.00   | 0.67     | 4999    |
| gpt                          | 0.95      | 0.02   | 0.04     | 4999    |
| accuracy                     |           |        | 0.51     | 9998    |
| macro avg                    | 0.73      | 0.51   | 0.36     | 9998    |
| weighted avg                 | 0.73      | 0.51   | 0.36     | 9998    |

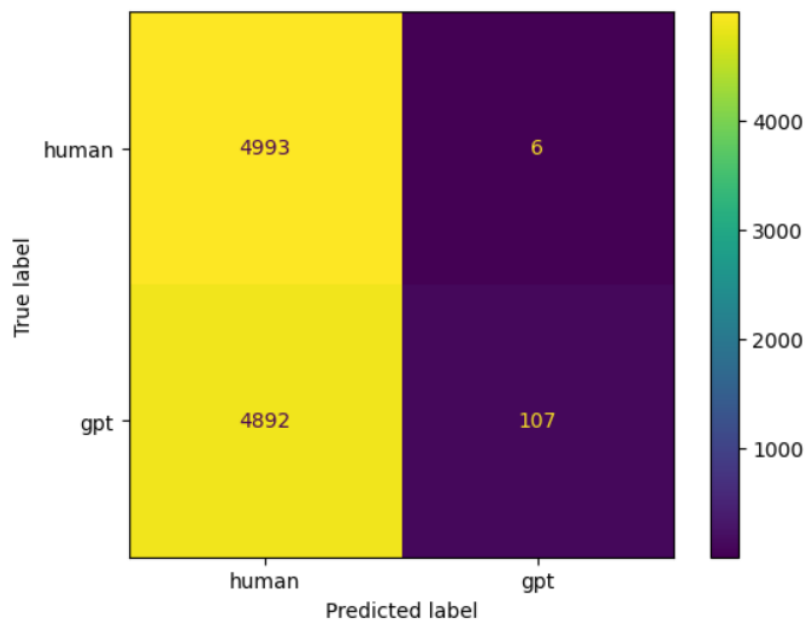


Figure A: Multinomial NB with preprocessing

Accuracy: 0.9140828165633127  
F1 Score: 0.9142157733217278

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| human        | 0.95      | 0.87   | 0.91     | 4999    |
| gpt          | 0.88      | 0.95   | 0.92     | 4999    |
| accuracy     |           |        | 0.91     | 9998    |
| macro avg    | 0.92      | 0.91   | 0.91     | 9998    |
| weighted avg | 0.92      | 0.91   | 0.91     | 9998    |

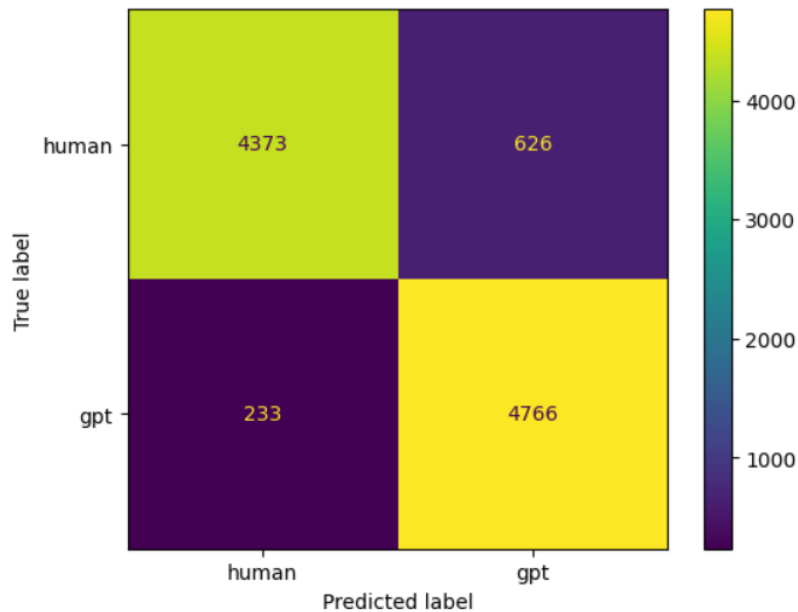


Figure B: Multinomial NB without preprocessing

Comparing Figures A and B, there is an accuracy difference of approximately 40.4% due to preprocessing. Looking at the confusion matrix in Figure A, the model made a lot more human predictions than ChatGPT predictions. As a result, the bottom left corner of the matrix (false negatives) has a value comparable to the top left corner of the matrix (true negatives). This suggests that the preprocessing might have cleaned out certain key text or words that ChatGPT uses in general, thus making the model unable to detect ChatGPT-generated text.

## Experimenting with Grammar Features

Additionally, the grammar features TextBlob Subjectivity and Formality Score were scraped. There was an attempt to incorporate grammar features into NB models to better distinguish between human text and ChatGPT-generated text. The response was directly fitted into the TF-IDF vectorizer with 5000 maximum features (instead of using a pipeline). A limit was placed on the features as it would be computationally intensive to fit all the responses with an unlimited number of features. Afterwards, the fitted values were stacked with the TextBlob Subjectivity and Formality Score. Similar to the preliminary dataset, the dataset was split into 30% testing and 70% training.

Apart from being used due to being the highest performing model during the preliminary test, Bernoulli NB was the only model suitable to be used, likely because the TF-IDF vectorizer had mapped certain words into negative values, thus the other models could not receive these values as input.

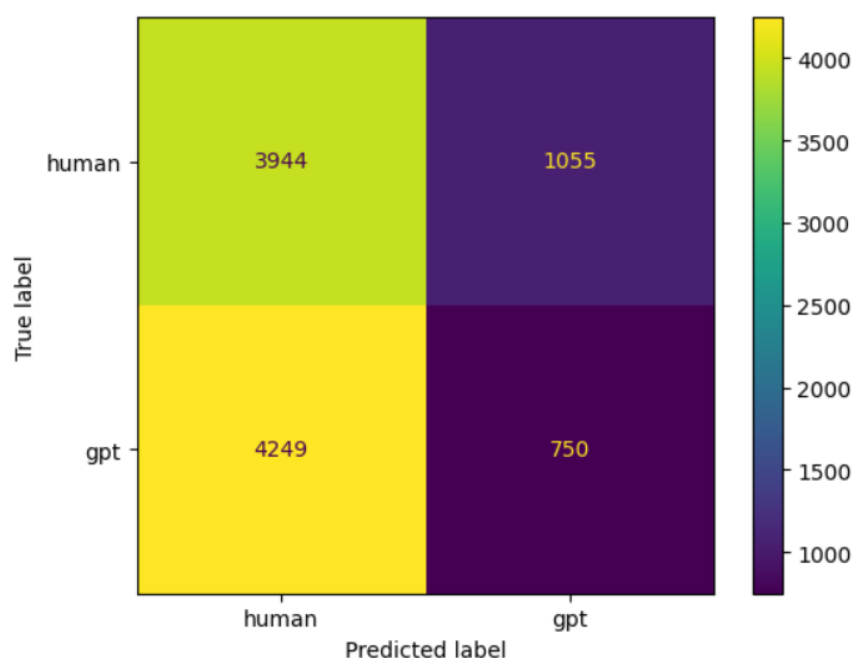
```

Accuracy: 0.46949389877975595
F1 Score: 0.52978944849862
      precision    recall  f1-score   support

   human         0.48      0.79      0.60       4999
    gpt         0.42      0.15      0.22       4999

 accuracy          0.47       9998
  macro avg         0.45      0.47      0.41       9998
 weighted avg         0.45      0.47      0.41       9998

```



Accuracy and confusion matrix of Bernoulli NB with grammar features

The accuracy of the model dropped as a result of implementing grammar features. Based on the confusion matrix, there were more false negatives classified. This could be because the number of features at this point is 5002, which meant that the new model had too few features, as compared to 77952 features in the pipeline model.

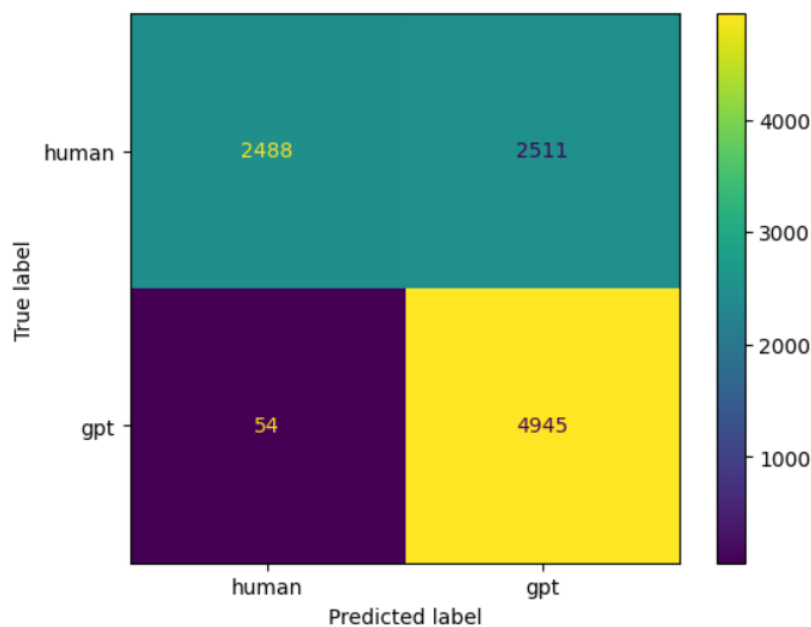
As a result of this experiment, grammar features were not implemented in the NB models and reverted to simply using the responses without preprocessing.

## Final NB Model

Multinomial NB was used without preprocessing. Despite Bernoulli NB having the highest accuracy score at the start, its training accuracy was lower than Multinomial NB's at 84% and 91% respectively. The test accuracy with the holdout dataset was 74.3% for Bernoulli NB and 91% for Multinomial NB.

```
Accuracy: 0.7434486897379476
F1 Score: 0.7599383564320036
```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| human        | 0.98      | 0.50   | 0.66     | 4999    |
| gpt          | 0.66      | 0.99   | 0.79     | 4999    |
| accuracy     |           |        | 0.74     | 9998    |
| macro avg    | 0.82      | 0.74   | 0.73     | 9998    |
| weighted avg | 0.82      | 0.74   | 0.73     | 9998    |



Bernoulli NB without preprocessing, holdout dataset

This is due to the false positives predicted by the model. In this case, this is likely due to the classifier being unable to distinguish between the text in the holdout dataset as the text is likely not discrete enough.

# Support Vector Machine

This section covers the different experiments we conducted with support vector machines (SVM). From our experimentation, we obtained the best combination of kernel methods, data preprocessing and word vectorization techniques to achieve the highest accuracy for SVM. The testing of the models is conducted with a 7:3 training-test split on the full training dataset unless otherwise specified.

## Base SVM Model

Our initial SVM model was a linear kernel SVM. The text was preprocessed by: converting all characters into all lowercase, tokenizing, removing stop words and special characters, and lemmatizing each word, before vectorizing the text using TF IDF.

```
SVM Accuracy Score -> 92.46433739921439
              precision    recall  f1-score   support

         0           0.93       0.92       0.92         4768
         1           0.92       0.93       0.93         4906

 accuracy                   0.92         9674
 macro avg              0.92       0.92       0.92         9674
weighted avg              0.92       0.92       0.92         9674
```

Accuracy score and classification report of the initial SVM model

The base SVM model achieved an accuracy of ~92% on the HC3 dataset, but we believe that it still can be improved.

## Comparing Different Kernels

Building on the results we obtained from the base model, we varied the kernels for the SVM while keeping the other factors constant.

| SVM Kernel | Accuracy (HC3 Dataset) | Accuracy (Full Training Dataset) |
|------------|------------------------|----------------------------------|
| Linear     | 92.46%                 | 93.21%                           |
| RBF        | 93.47%                 | 94.46%                           |
| Sigmoid    | 92.21%                 | 92.71%                           |

We obtained the highest accuracy using an RBF kernel. However, that could be due to the `max_features` parameter used for the `TfidfVectorizer` being only 5000, allowing for the SVM to better linearly separate data points after the RBF kernel transforms the data points into a higher dimensional space. Additionally, it is generally recommended to use a linear kernel for text classification since text has a lot of features and is often linearly separable (KOWALCZYK,

2014). Thus, we will mainly use the RBF kernel for further experimentation but we will also explore different values of max\_features to see if that improves the accuracy of the linear kernel SVM.

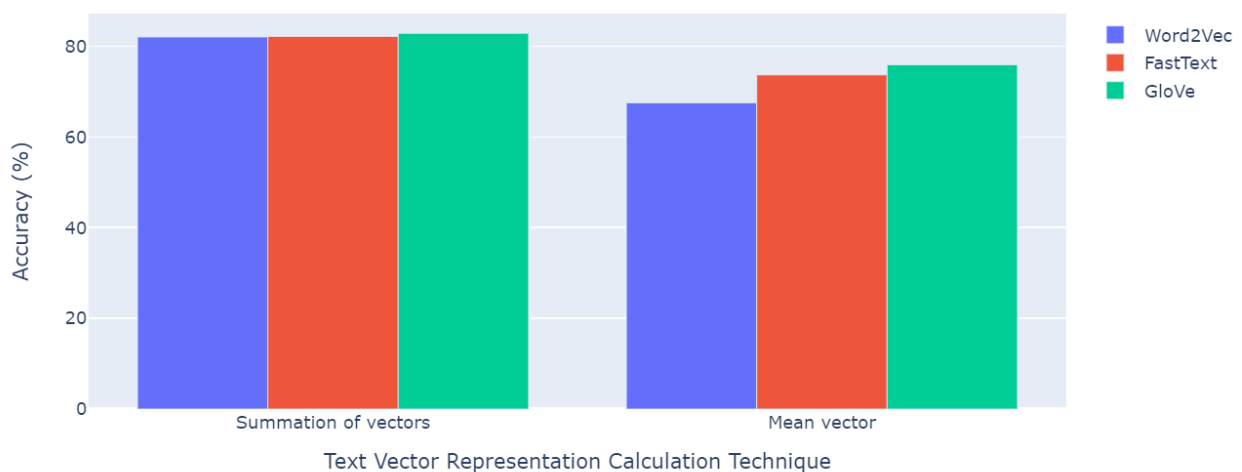
## Comparing Different Text Vectorization Techniques

In addition to TF IDF, we also tried out other text vectorization techniques, including using word embedding models such as word2vec, FastText and GloVe. To keep the number of experiments manageable, we used an RBF kernel SVM for all the experiments and kept the data preprocessing method the same.

For the word embedding models, there are many ways to obtain the vector representation of a string of words, we experimented with 2 methods: the summation of vectors and the mean vector.

| Word Embedding Model                                            | Text Vector Representation Calculation Technique | Accuracy |
|-----------------------------------------------------------------|--------------------------------------------------|----------|
| Word2Vec<br>( <a href="#">word2vec-google-news-300</a> )        | Summation of vectors                             | 82.05%   |
|                                                                 | Mean vector                                      | 67.53%   |
| FastText<br>( <a href="#">fasttext-wiki-news-subwords-300</a> ) | Summation of vectors                             | 82.17%   |
|                                                                 | Mean vector                                      | 73.68%   |
| GloVe<br>( <a href="#">glove-wiki-gigaword-300</a> )            | Summation of vectors                             | 82.85%   |
|                                                                 | Mean vector                                      | 75.91%   |

Accuracy of RBF SVM Model with Different Word Embedding Models



We found out that the summation of vectors technique worked better than using the mean vector. However, TF IDF remains the best text vectorization technique for our application since it has a higher accuracy. This could be due to the TfidfVectorizer's vocabulary being more suited for our application, since it was fitted with text that is generated by ChatGPT, in addition to human-written content. Another reason could be due to the greater number of features generated by TF IDF in comparison to word embedding models, allowing for a better linear separation of data points.

## Experimenting with Different Data Preprocessing Methods

We also experimented with just feeding the responses into the [TfidfVectorizer](#) from scikit-learn without any prior preprocessing since the TfidfVectorizer already processes the input. The experiment was conducted with an RBF kernel SVM, with max\_features = 5000.

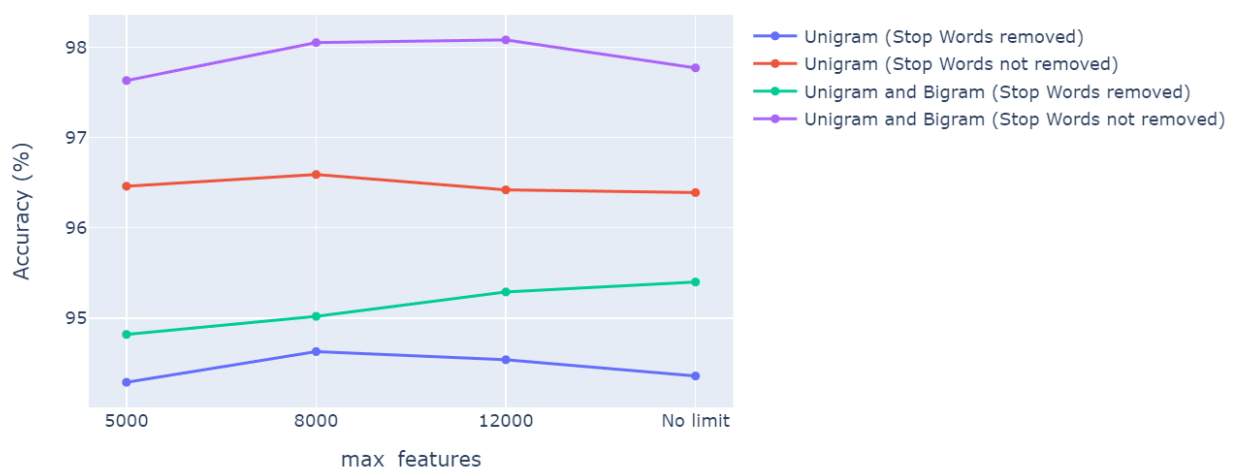
| Preprocessing                                                                            | Accuracy |
|------------------------------------------------------------------------------------------|----------|
| Convert to lowercase, stopword and special character removal, lemmatization <sup>3</sup> | 94.46%   |
| No prior preprocessing                                                                   | 95.86%   |

The SVM trained on data without prior preprocessing performed better than the SVM that was trained on the raw data, Thus, we chose to experiment more with purely using the TfidfVectorizer to process and vectorize the text data.

## Experimenting with the Removal of Stop Words

We experimented with TfidfVectorizer's stopword removal to investigate if removing stop words will result in a more accurate model.

Accuracy of RBF SVM Model with or without Stop Words



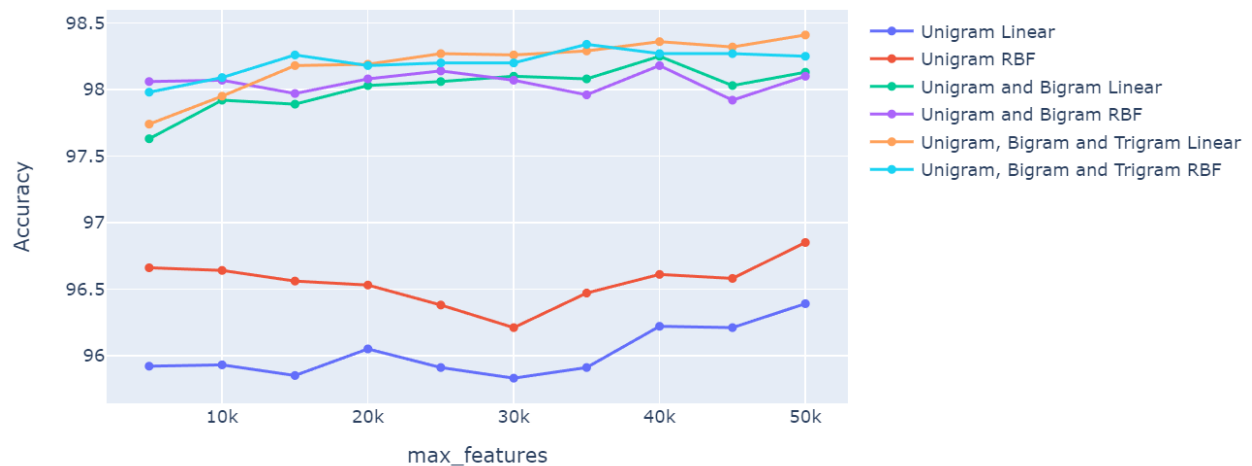
<sup>3</sup> Implementation of the preprocessing can be found in our github repository

Our experimentation revealed that our models consistently performed better when stop words were not removed, regardless of the value of max\_features and N-gram combination. We also found out that models trained with unigrams and bigrams had a higher accuracy than models trained with just unigrams.

### Experimenting with N-grams and Number of Features for TF IDF.

Building on the results of the previous experiments, we decide to test out trigrams as well, to see if it further improves the accuracy of the SVM model. For this experiment, we varied the max\_features, N-gram combination and kernel.

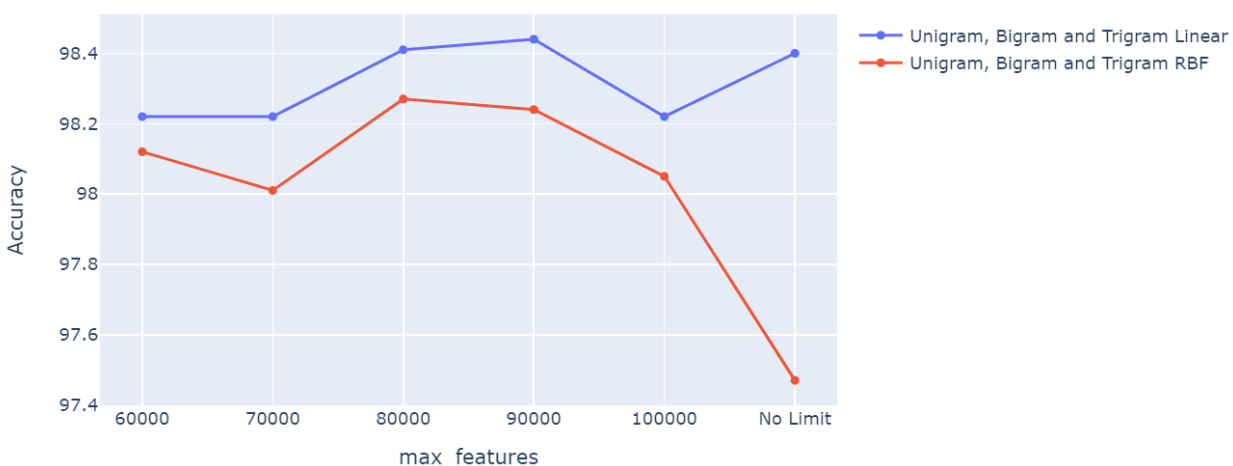
Accuracy of SVM Model with Different Kernels and N-grams



Our initial experiment tested up to max\_features = 50k since it was getting increasingly computationally expensive to train an SVM with an increasing number of max\_features. Using the initial data collected, we saw that the models trained on a combination of unigram, bigram and trigrams performed the best and at max\_features = 40k, the linear kernel SVM starts to outperform the RBF kernel SVM, which proves our earlier hypothesis that the reason why the linear kernel SVM was performing more poorly than the RBF kernel SVM is because max\_features was too low. To investigate further, we focused on just the best-performing unigram, bigram and trigram combination and continued to vary the max\_features.



Accuracy of SVM Model with Different Kernels with Unigram, Bigram and Trigram

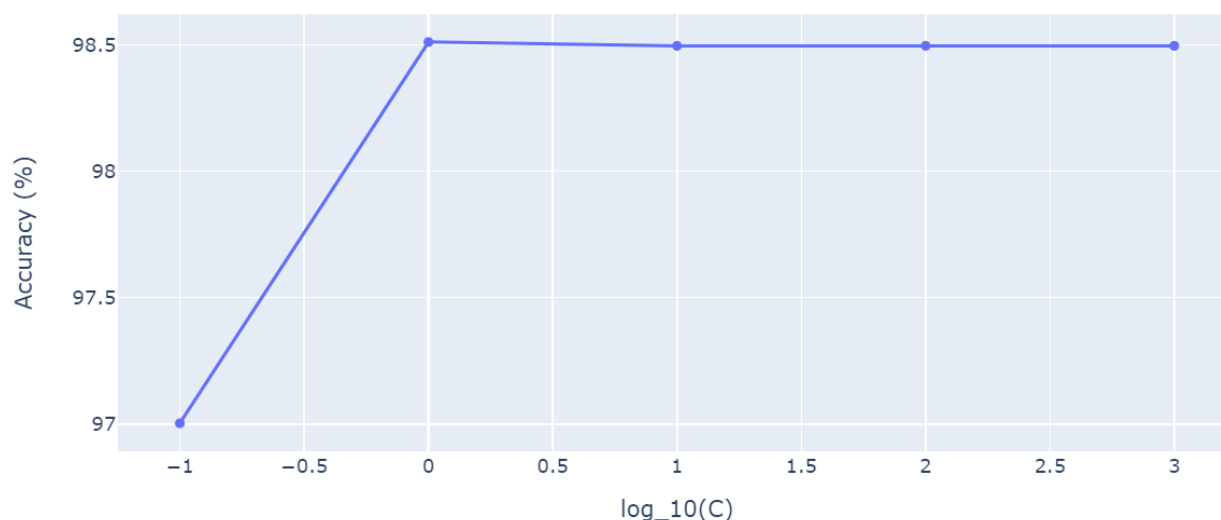


From our experiment, the linear kernel SVM consistently performs better than the RBF kernel SVM when `max_features > 60000`. This could be due to the RBF kernel SVM being overfitted to the training data, causing it to perform worse when being exposed to unseen data. Another observation we obtained from the experiment is that the peak accuracy is attained at `max_features = 90000` for the linear kernel, which is the optimal number of features before the model suffers from overfitting.

## Hyperparameter Tuning

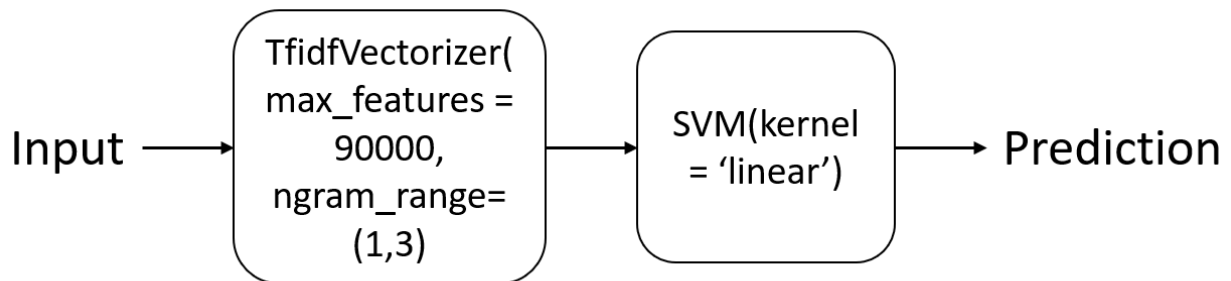
The regularization parameter `C` was varied from 0.1 to 1000 to investigate what is the best value for `C` that maximizes the accuracy of the model. This experiment is done using the most optimal combination of parameters described in the previous section.

Effect of regularization parameter `C` on accuracy



Our investigation showed that the best accuracy is achieved when  $C = 1$ . Therefore, we will use 1 as the regularization parameter for our final SVM model.

### Final SVM Model



Our final SVM model utilizes scikit-learn's TfidfVectorizer, with `max_features = 90000`, and uses unigrams, bigrams and trigrams. The vectorized output from the TfidfVectorizer will be fed into the SVM model with a linear kernel and regularization parameter = 1. This combination has achieved an accuracy of 98.33% on the training dataset and 95.89% on the holdout dataset.

## LSTM

The neural networks were built with TensorFlow. All of them used Adam as an optimizer and Binary Cross-Entropy as the loss function. Early stopping was implemented as well to reduce the chance of overfitting.

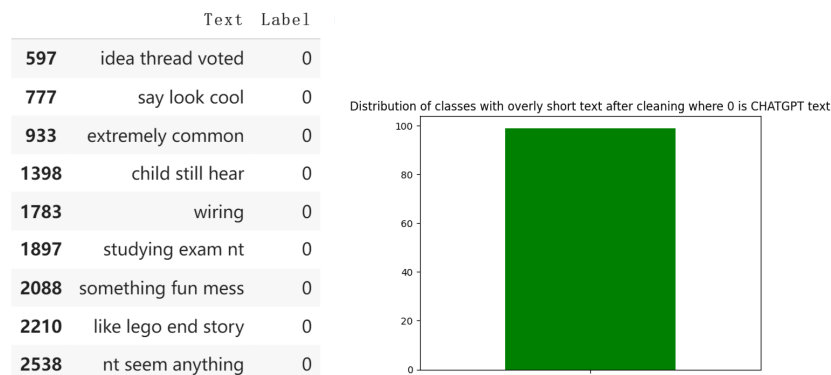
Bidirectional LSTMs are suitable for text classification because of their ability to model sequential information. For longer texts, Bidirectional LSTMs stand out in their ability to capture long-range dependencies while taking into account both forward and backward context. In a study, Bidirectional-LSTMs have also been recognised for their performance in shorter texts (Karl & Scherp, 2022). This is important because our dataset contains a mixture of shorter and longer text whereby the human answers on Reddit ELI5 tend to be shorter.

| Length of Text before any processing (Characters) | Human Answer | ChatGPT Answer |
|---------------------------------------------------|--------------|----------------|
| 25th Percentile                                   | 244          | 781            |
| 50th Percentile                                   | 475          | 991            |
| 75th Percentile                                   | 914          | 1194           |

## TF-IDF Embeddings

An initial basic Bidirectional-LSTM was built with TFIDF vectorization after some preprocessing steps (e.g. lowercasing, removal of stopwords and lemmatization). The Colab link can be found [here](#) or [Github link](#).

The model trained achieved an accuracy of 96% after 3 epochs. It had a similar test dataset accuracy. We tried to find out the reasons why. We found that after further cleaning to make the TF-IDF vector, this resulted in the already short text from the Human written category becoming even shorter. For example, if we were to take the number of texts that were left with less than 20 characters after the removal of stopwords, all of the texts are from the human class.



One reason for the high accuracy could be that for example, since the Reddit ELI5 responses tend to be shorter, a sparser TF-IDF vector could be a feature that helps the model decide in classifying the text as human. This sparsity cannot be captured as well with word embeddings.

## Experiment with BERT-Embeddings

In the context of using a bidirectional LSTM, word embeddings are suitable since the strength of Bidirectional-LSTMs is in their ability to account for contextual information by capturing both forward and backward dependencies over a long range. For this task of differentiating AI-generated text from human-generated text, contextual information could be helpful due to potentially different approaches to explaining the answer to the question made by AI and humans. To help the LSTM better capture contextual information, word embeddings are better as the vectors for each word are made in reference to the surrounding words.

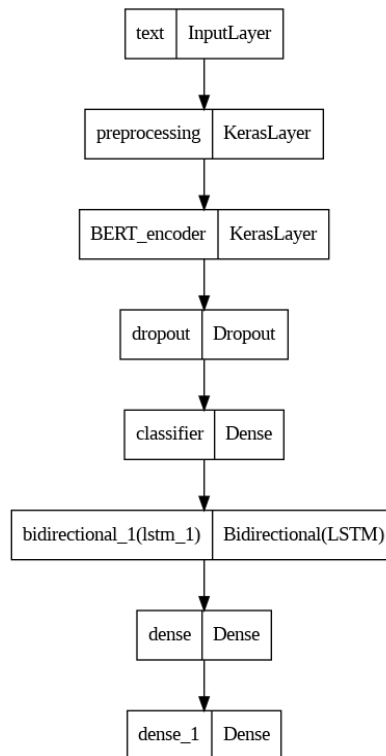
There are several different choices for word embeddings. In a study, Bidirectional-LSTMs with BERT embeddings or GLOVE embeddings performed comparably in different datasets involving the classification of short texts (Karl & Scherp, 2022). As a result, based on considerations of the strengths of the Bidirectional-LSTM and in the interest of reducing correlation between the models (like with the SVM which is using TF-IDF and GloVe embeddings), we decided to go with the BERT embeddings for the Bi-LSTM over GloVe.

Table 2: Accuracy on short text classification datasets. The “Short Text” column indicates whether the model makes claims about its ability to categorize short texts. Provenance refers to the source of the accuracy scores.

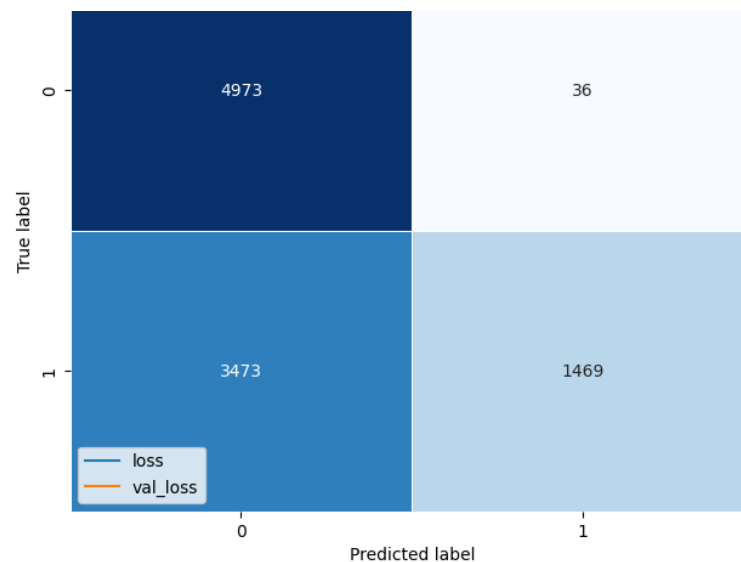
| Inductive Models                   | Short Text | R8                 | MR           | SearchSnippets           | Twitter      | TREC        | SST-2        | Provenance        |
|------------------------------------|------------|--------------------|--------------|--------------------------|--------------|-------------|--------------|-------------------|
| <i>Transformer Models</i>          |            |                    |              |                          |              |             |              |                   |
| BERT                               | N          | 98.17 <sup>1</sup> | 86.94        | 88.20                    | 99.96        | <b>99.4</b> | 91.37        | Own experiment    |
| RoBERTa                            | N          | 98.17 <sup>1</sup> | 89.42        | 85.22                    | 99.9         | 98.6        | 94.01        | Own experiment    |
| DeBERTa                            | N          | 98.45 <sup>1</sup> | <b>90.21</b> | 86.14                    | 99.93        | 98.8        | <b>94.78</b> | Own experiment    |
| ERNIE 2.0                          | N          | 98.04 <sup>1</sup> | 88.97        | 89.12                    | <b>99.97</b> | 98.8        | 93.36        | Own experiment    |
| ERNIE 2.0 (optimized)              | N          | 98.17 <sup>1</sup> | 89.53        | 89.17                    | <b>99.97</b> | 99          | 94.07        | Own experiment    |
| DistilBERT                         | N          | 97.98 <sup>1</sup> | 85.31        | 89.69                    | 99.96        | 99          | 90.49        | Own experiment    |
| ALBERTv2                           | N          | 97.62              | 86.02        | 87.68                    | <b>99.97</b> | 98.6        | 91.54        | Own experiment    |
| <i>BoW Models</i>                  |            |                    |              |                          |              |             |              |                   |
| WideMLP                            | N          | 96.98              | 76.48        | 67.28                    | 99.86        | 97          | 82.26        | Own experiment    |
| fastText                           | N          | 96.13              | 75.14        | 88.56 <sup>4</sup>       | —            | —           | —            | Zhao et al. [42]  |
| <i>Graph-based Models</i>          |            |                    |              |                          |              |             |              |                   |
| HGAT <sup>2</sup>                  | Y          | —                  | 62.75        | 82.36                    | 63.21        | —           | —            | Yang et al. [37]  |
| NC-HGAT <sup>2</sup>               | Y          | —                  | 62.46        | —                        | 63.76        | —           | —            | Sun et al. [29]   |
| SGNN <sup>3</sup>                  | Y          | 98.09              | 80.58        | 90.68 <sup>4</sup>       | —            | —           | —            | Zhao et al. [42]  |
| ESGNN <sup>3</sup>                 | Y          | 98.23              | 80.93        | <b>90.80<sup>4</sup></b> | —            | —           | —            | Zhao et al. [42]  |
| C-BERT (ESGNN + BERT) <sup>3</sup> | Y          | 98.28              | 86.06        | 90.43 <sup>4</sup>       | —            | —           | —            | Zhao et al. [42]  |
| DADGNN                             | Y          | 98.15              | 78.64        | —                        | —            | 97.99       | 84.32        | Liu et al. [18]   |
| DADGNN                             | Y          | 97.28              | 74.54        | 84.91                    | 98.16        | 97.54       | 82.81        | Own experiment    |
| HyperGAT                           | N          | 97.97              | 78.32        | —                        | —            | —           | —            | Ding et al. [5]   |
| InducT-GCN                         | N          | 96.67              | 75.25        | 76.68                    | 88.53        | 92.42       | 79.98        | Own experiment    |
| ConTextING-BERT                    | N          | 97.91              | 86.01        | —                        | —            | —           | —            | Huang et al. [10] |
| ConTextING-RoBERTa                 | N          | 98.13              | 89.43        | —                        | —            | —           | —            | Huang et al. [10] |
| <i>CNN and LSTMs</i>               |            |                    |              |                          |              |             |              |                   |
| SECNN <sup>3</sup>                 | Y          | —                  | 83.89        | —                        | —            | 91.34       | 87.37        | Wang et al. [32]  |
| LSTM (BERT)                        | Y          | 94.28              | 75.10        | 65.13                    | 99.83        | 97          | 81.38        | Own experiment    |
| Bi-LSTM (BERT)                     | Y          | 95.52              | 75.30        | 66.79                    | 99.76        | 97.2        | 80.83        | Own experiment    |
| LSTM (GloVe)                       | Y          | 96.34              | 74.99        | 67.67                    | 95.23        | 97.4        | 79.95        | Own experiment    |
| Bi-LSTM (GloVe)                    | Y          | 96.84              | 75.32        | 68.15                    | 95.53        | 97.2        | 80.17        | Own experiment    |
| Bi-LSTM (GloVe)                    | Y          | 96.31              | 77.68        | 84.81 <sup>4</sup>       | —            | —           | —            | Zhao et al. [42]  |

Bi-LSTM (BERT) and Bi-LSTM (GloVe) performed comparably across the different classification tasks for short texts (Karl & Scherp, 2022).

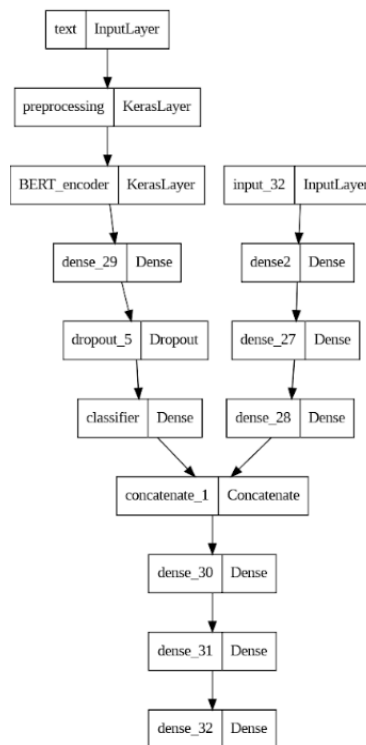
## With BERT Embeddings



Even though a dropout layer was added and there was early stopping, the LSTM with BERT seemed to have overfitted. It had 99% training accuracy and performed poorly with 64% accuracy on the test dataset because of wrongly predicting the AI-written text as human. This was an unexpected result given that from previous studies, this combination has not led to very poor results. One reason for this overfitting could be because compared to the study, the size of our training data is smaller so the model may have memorized the data instead of learning the patterns.



## With Perplexity



To further increase diversity in the models, each model was allocated a feature to experiment with so that the different models used different features. For the neural networks, it was perplexity and GLTR. A Bidirectional LSTM was trained with the mean perplexity feature. The LSTM was trained on a smaller subset of about 17000 data points due to the time taken to scrape the perplexity feature. The model got a training accuracy of 76% and test accuracy of 74% but on the holdout dataset of about 2000 rows, it achieved the highest accuracy for that dataset with 96% accuracy. This suggests that the perplexity feature could help the Bidirectional-LSTM with BERT embeddings generalize well.

### RNN Perplexity Classifier

Accuracy: 0.964

F1: 0.9642857142857142

ROC\_AUC: 0.9641188144164906

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Human        | 0.96      | 0.97   | 0.96     | 984     |
| ChatGPT      | 0.97      | 0.96   | 0.96     | 1016    |
| accuracy     |           |        | 0.96     | 2000    |
| macro avg    | 0.96      | 0.96   | 0.96     | 2000    |
| weighted avg | 0.96      | 0.96   | 0.96     | 2000    |

## AutoML: Deep Learning with GLTR Features

[Colab Link](#) and [Github Link](#)

To experiment with the GLTR Features and conduct hyperparameter tuning, we decided to try the AutoKeras library to automate experiments with different neural network architectures and hyperparameters. After multiple trials, the final model output was a convolutional neural network. CNN achieved a training and test accuracy of 96%. On the holdout dataset of about 10000 data points, it achieved an accuracy of 83%. The reason why a CNN could have been considered more optimal than an LSTM could be because local context is more important in the case of this classification task. CNN is good at recognising local features due to the filter and pooling layers. This could also explain why the TF-IDF vector performed well previously since TF-IDF vectors capture local features like the presence and absence of individual words. On the other hand, bidirectional-LSTMs could be better at global context by capturing dependencies between the words.

| CNN_GLTR     | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Human        | 0.86      | 0.78   | 0.82     | 4999    |
| ChatGPT      | 0.80      | 0.87   | 0.84     | 4999    |
| accuracy     |           |        | 0.83     | 9998    |
| macro avg    | 0.83      | 0.83   | 0.83     | 9998    |
| weighted avg | 0.83      | 0.83   | 0.83     | 9998    |

CNN\_GLTR performance on holdout dataset of nearly 10000 data points

## Transformer-based Models

In this section, we will discuss three different transformer-based language models that are widely used in natural language processing (NLP): BERT, RoBERTa, and DistilBERT. Each of these models has its unique architecture and pre-training techniques, which allow them to effectively learn from large amounts of text data and perform well on a wide range of NLP tasks.

We will discuss three models. Firstly, BERT, which is a highly effective model for text classification tasks. Next, we will discuss RoBERTa, which is an improved version of BERT that uses a larger pre-training corpus and optimized pre-training techniques. Finally, we will discuss DistilBERT, which is a smaller and faster version of BERT that was created by distilling the knowledge from the larger BERT model into a smaller and more efficient architecture.

These models are believed to be well-suited for the task of distinguishing between human and AI-generated answers because they can effectively learn the underlying patterns and characteristics of both types of text. This is because these models are pre-trained on a large corpus of text data and can extract features and representations that are relevant to the task at hand. By fine-tuning these models, the objective is for them to learn to classify human and AI-generated answers accurately and efficiently.

### Data Used for Experimentation

Experiments done with the above-mentioned transformer models utilize two main training datasets, and one final holdout dataset consisting of around 10,000 samples.

The first training dataset corresponds to the HC3 question and answers, undersampled to ensure even distribution between human and ChatGPT answers. This consists of around 30,000 samples in total.

The second dataset additionally consists of the answers obtained from the ChatGPT API, as well as the paraphrased answers on top of the baseline HC3 answers.

### Bidirectional Encoder Representations from Transformers (BERT)

BERT (Bidirectional Encoder Representations from Transformers) is a large-scale transformer-based language model that was introduced by Google in 2018. In BERT, the model is conditioned on both the left and right context by using a technique called masked language modeling (MLM). During training, BERT randomly masks some of the input tokens and trains the model to predict the masked tokens based on their surrounding context. This encourages the model to learn representations that capture the context of each word by looking at both the words to its left and right. It is a highly effective model for natural language processing (NLP) tasks, including text classification, named entity recognition, and question answering.



## Experiments and Results

| Pre-trained model | Accuracies obtained from training on HC3 |              | Accuracy on HC3 with stopwords removed | Accuracy on full dataset with paraphrased |              |
|-------------------|------------------------------------------|--------------|----------------------------------------|-------------------------------------------|--------------|
|                   | HC3 Test                                 | Holdout Test |                                        | Full Test                                 | Holdout Test |
| bert-base-cased   | 94.3%                                    | 88.8%        | 90.6%                                  | 95.7%                                     | 90.8%        |

## DistilBERT

DistilBERT is a smaller and faster version of BERT, which is a widely used transformer-based model for natural language processing tasks. DistilBERT was created by distilling the knowledge from the larger BERT model into a smaller and more efficient architecture, without sacrificing too much performance.

DistilBERT has several advantages over BERT, such as being faster and requiring less memory to train and run, which makes it more suitable for use in applications where resources are limited, such as mobile devices and embedded systems. Additionally, it has been shown to achieve comparable performance to BERT on many NLP tasks, such as text classification, named entity recognition, and question answering.

## Experiments and Results

| Pre-trained model     | Accuracies obtained from training on HC3 |              | Accuracy on full dataset with paraphrased |              |
|-----------------------|------------------------------------------|--------------|-------------------------------------------|--------------|
|                       | HC3 Test                                 | Holdout Test | Full Test                                 | Holdout Test |
| distilbert-base-cased | 93.5%                                    | 87.8%        | 98.0%                                     | 92.9%        |

## RoBERTa

RoBERTa (Robustly Optimized BERT Pretraining Approach) is a large-scale transformer-based language model, similar to BERT, that was released by Facebook AI Research in 2019. RoBERTa is designed to improve upon the performance of BERT on natural language processing tasks by using a larger pre-training corpus and optimized pre-training techniques.

RoBERTa is trained on a diverse set of large-scale datasets and is pre-trained using dynamic masking techniques that enable it to learn more effectively from the data. It has been shown to

outperform BERT on several benchmark NLP tasks, including text classification, named entity recognition, and question answering.

## Experiments and Results

| Pre-trained model | Accuracies obtained from training on HC3 |              | Accuracy on full dataset with paraphrased |              |
|-------------------|------------------------------------------|--------------|-------------------------------------------|--------------|
|                   | HC3 Test                                 | Holdout Test | Full Test                                 | Holdout Test |
| roberta-base      | 88.9%                                    | 78.0%        | 92.1%                                     | 93.0%        |

## Binary Classification Experiments Summary

Looking at the results from the experiments from all 3 models of interest as described above, the final pre-trained model used for ensembling experiments is bert-base-cased as it results in the highest accuracy in the train vs holdout generalizability tradeoff.

## Multi-class Classification on Paraphrased Samples

Instead of considering paraphrased answers as AI-generated, this section explores the idea of distinguishing between ChatGPT answers and its paraphrased variant. By training a multiclass model, the idea is to provide the model with additional information about the type of response it is dealing with, which can improve its accuracy in predicting the correct response. This is particularly important if the paraphrased responses are substantially different from the human and ChatGPT responses, and the model needs to learn to distinguish between them. Additionally, a multiclass model allows for a better understanding of the model's performance on different types of responses. By examining the accuracy of the model for each class, it is possible to then identify areas where the model is performing well and areas where it needs improvement. This allows for the refining of training data and the improvement of the model over time.

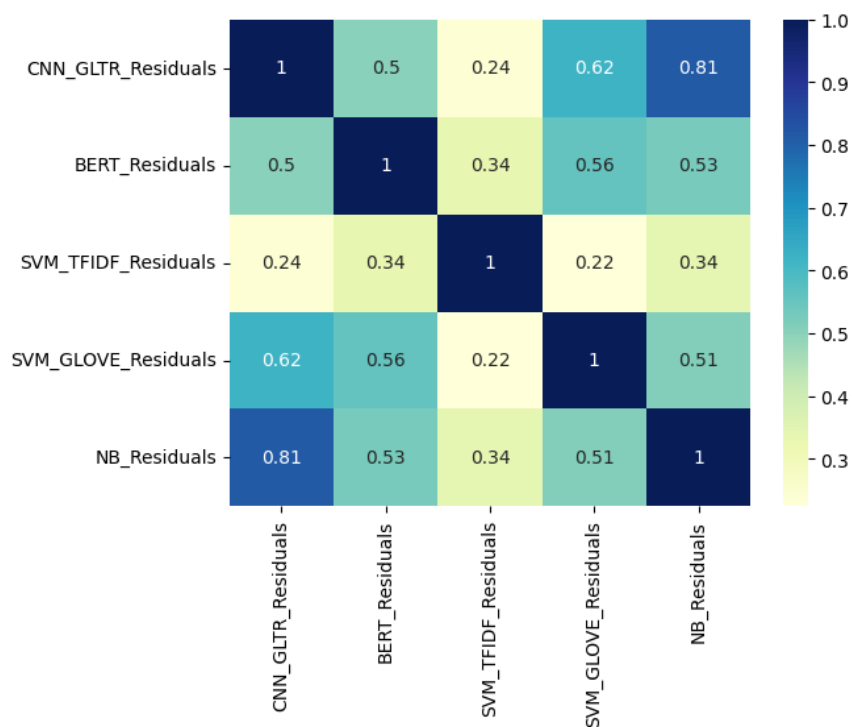
## Experiments and Results

| Pre-trained model | Accuracy on full dataset with paraphrased |
|-------------------|-------------------------------------------|
|                   | Full Test                                 |
| bert-base-cased   | 84.1%                                     |

## Ensemble

The main benefit of ensembles is that they could help to reduce generalization errors when faced with unseen data<sup>4</sup>. To achieve this, the model should consist of models with different decision boundaries. To achieve this, we could vary the data samples, vary the features used in each model or vary the model architecture and hyperparameters. We did not choose to vary the data so that each model can be exposed to more data for training due to an already limited number of samples. But, the transformer models were exposed to additional paraphrased data in training. We conducted feature sampling to increase diversity in the models. For example, the Keras neural networks built had GLTR and Perplexity in the training dataset while the others did not. Finally, we varied the architecture of the models with RNN, CNN, Naive Bayes, SVM and Transformer architecture.

To check if the models have different decision boundaries, we created a holdout dataset of 10000 data points with equal distribution of “Human” and “AI” labels. We then measured how correlated the errors of the different models were. The models that were correlated with each other were mainly SVM\_GLOVE and CNN\_GLTR as well as CNN\_GLTR with Naive Bayes.



Another criterion for a well-performing ensemble is that it should be high-performing. Thus, amongst the correlated models, we would choose the one with the highest accuracy, ROC-AUC score and F1 score. ROC\_AUC and F1 scores were added as it is important to balance precision and recall to ensure that human text is not falsely labeled as AI. As we are using the

<sup>4</sup> <https://www.sciencedirect.com/topics/computer-science/ensemble-modeling>

weighted average ensemble, after selecting the models, we applied grid search to find the optimal weights. The details can be seen in this [notebook](#).

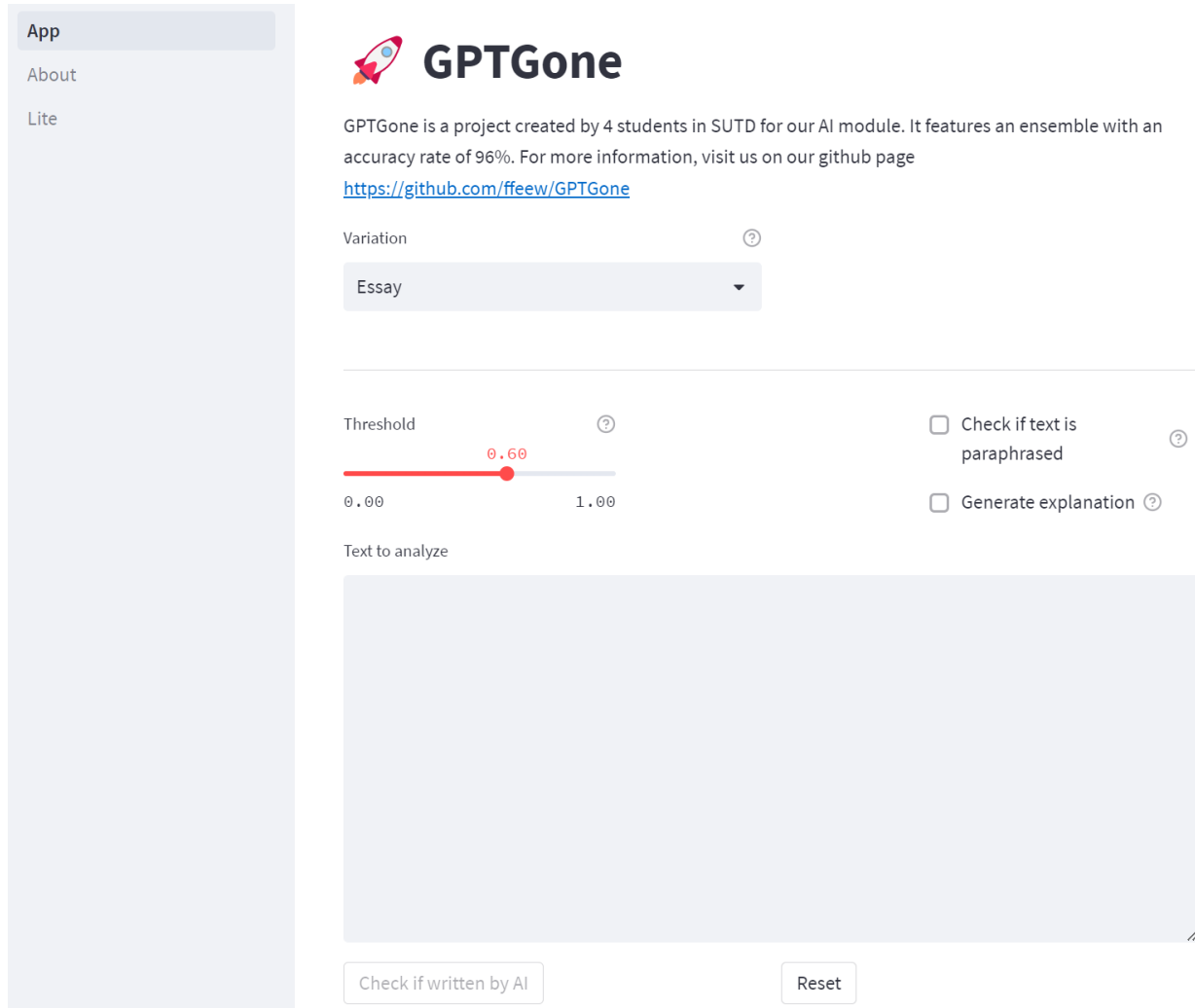
The Naive Bayes results were achieved at a later time after adjustments were made to the model late into the project timeline. When the GUI was built, based on performance, the top models were SVM\_TFIDF, BERT and GLTR as NB had an accuracy of 50%, thus our demonstration will involve these models. But, NB could be a model type with good potential to be explored further in the future.

| Model     | Accuracy | ROC-AUC | F1    |
|-----------|----------|---------|-------|
| SVM_TFIDF | 0.958    | 0.958   | 0.960 |
| NB        | 0.92     | 0.92    | 0.92  |
| BERT      | 0.910    | 0.910   | 0.905 |
| CNN_GLTR  | 0.827    | 0.827   | 0.835 |
| SVM_GLOVE | 0.651    | 0.652   | 0.613 |

Due to the time taken to scrape perplexity, it will not be included in our main demonstration due to user experience concerns. We did conduct an evaluation with a smaller holdout dataset of 2000 data points. This dataset included a Perplexity feature column. The LSTM trained with Perplexity as a feature performed the best at 96% accuracy, showing that the feature has potential. A Google Colab demonstration that involves scraping perplexity is available [here](#).

# Graphical User Interface

The graphical user interface (GUI) of GPTGone is an interactive web application built using [Streamlit](#) and hosted on [huggingface.co](#). The web application can be accessed [here](#).



The screenshot shows the GPTGone web application interface. On the left is a sidebar with navigation links: 'App' (selected), 'About', and 'Lite'. The main content area features the GPTGone logo (a rocket icon) and a description: 'GPTGone is a project created by 4 students in SUTD for our AI module. It features an ensemble with an accuracy rate of 96%. For more information, visit us on our github page <https://github.com/ffeew/GPTGone>'. Below this is a 'Variation' dropdown menu set to 'Essay'. A 'Threshold' slider is positioned at 0.60, with a range from 0.00 to 1.00. To the right of the slider are two checkboxes: 'Check if text is paraphrased' and 'Generate explanation', both of which are unchecked. A large text area labeled 'Text to analyze' is provided for input. At the bottom, there are two buttons: 'Check if written by AI' and 'Reset'.

The user interface of GPTGone

GPTGone has 5 main features:

1. ChatGPT content detection
2. Paraphrased ChatGPT content detection
3. Probability breakdown by each model (to know each model's prediction)
4. Explainability generation
5. Cyrillic character detection (to prevent bypassing by mapping English alphabets to Cyrillic characters)

## Features

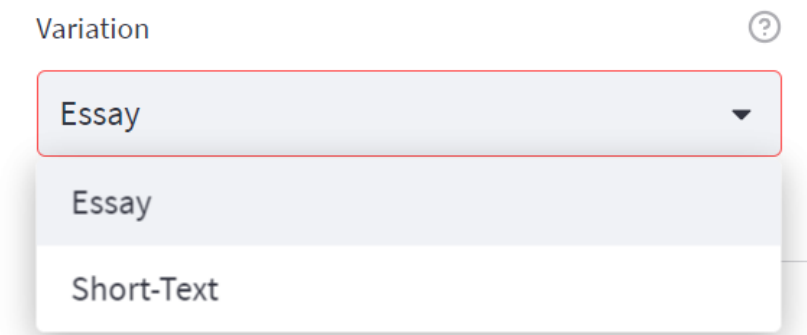
This section will provide an introduction to GPTGone's features. It will also provide documentation on how to use the features mentioned.

### ChatGPT Content Detection

GPTGone provides a way for users to validate if a piece of text is written by humans or ChatGPT. The web app has 2 variations, "Essay" and "Short-Text". When using the "Essay" variation, there will be no character limit on the size of input a person can enter as the input will split into smaller chunks when the models' token limit is exceeded. On the other hand, when using the "Short-Text" variation, the user input will be truncated when it exceeds the model's token limit, however, it will return a more accurate prediction than the Essay variation since our model was trained on the whole text<sup>5</sup>. Hence it is most suited for text that is less than 2000 characters long, given the token limit for BERT is 512 tokens.

Using the web app is extremely intuitive.

First, choose the variation you would like to use by clicking on the variation select-box.



Variation select-box with "Essay" selected

---

<sup>5</sup> We did not enforce a word limit on texts used in training. For example, if the input had 4000 characters, the model would receive the full 4000 characters.

Secondly, enter the text into the “Text to analyze” text box. Doing so would enable the “Check if written by AI” button.

Text to analyze

The k-means clustering concept sounds pretty great, right? It’s simple to understand, relatively easy to implement, and can be applied in quite a number of use cases. But there are certain drawbacks and limitations that we need to be aware of.

Let’s take the same income-expenditure example we saw above. The k-means algorithm seems to be working pretty well, right? Hold on – if you look closely, you will notice that all the clusters created have a circular shape. This is because the centroids of the clusters are updated iteratively using the mean value.

Now, consider the following example where the distribution of points is not in a circular form. What do you think will happen if we use k-means clustering on this data? It would still attempt to group the data points in a circular fashion. That’s not great! k-means fails to identify the right clusters:

Check if written by AI

Reset

A populated text area with “Check if written by AI” button enabled

Thirdly, click on the “Check if written by AI” button and it would generate the results as seen in the image below.

AI

0.0%

The text is most likely written by a human

## Text analysis

The k-means clustering concept sounds pretty great, right? It's simple to understand, relatively easy to implement, and can be applied in quite a number of use cases. But there are certain drawbacks and limitations that we need to be aware of.

Hu  
m  
an

Let's take the same income-expenditure example we saw above. The k-means algorithm seems to be working pretty well, right? Hold on – if you look closely, you will notice that all the clusters created have a circular shape. This is because the centroids of the clusters are updated iteratively using the mean value.

H  
u  
m  
a  
n

Now, consider the following example where the distribution of points is not in a circular form. What do you think will happen if we use k-means clustering on this data? It would still attempt to group the data points in a circular fashion. That's not great! k-means fails to identify the right clusters:

H  
u  
m  
a  
n

The output from GPTGone (Essay variation) for an input obtained from a blog post<sup>6</sup>

---

<sup>6</sup> Blog post accessible at <https://www.analyticsvidhya.com/blog/2019/10/gaussian-mixture-models-clustering/>



The previous example shows a scenario of analyzing a human written input. Below is an output generated by entering text generated by ChatGPT.

AI

100.%

The text is highly likely written by AI

## Text analysis

|                                                                                                                                                                                                                                                                                                                                                                                                                                                |    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| A Gaussian Mixture Model (GMM) is a statistical model that is commonly used to model complex distributions of data. It is a type of clustering algorithm that assumes that the data is generated from a mixture of several Gaussian distributions.                                                                                                                                                                                             | AI |
| In a GMM, the data is modeled as a weighted sum of several Gaussian distributions, each with its own mean and standard deviation. The weights of the individual Gaussians represent the proportion of the total data that comes from each component. By fitting the model to the data, we can estimate the parameters of the individual Gaussians and the weights that describe the mixture.                                                   | AI |
| GMMs are used in a variety of applications, including image processing, speech recognition, and data mining. They are particularly useful in situations where the underlying data distribution is not well understood, and the goal is to identify the number of clusters that best describe the data. GMMs can also be used for anomaly detection, where the model is used to identify data points that do not fit the expected distribution. | AI |

The output from GPTGone (Essay variation) for an input generated by ChatGPT

GPTGone is able to detect ChatGPT written content when it is nested in human-written content as well. An example of such a scenario would be when a student uses ChatGPT to generate some part of their essay.

AI

28.5%

The text is very likely written by a human

## Text analysis

|                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| The k-means clustering concept sounds pretty great, right? It's simple to understand, relatively easy to implement, and can be applied in quite a number of use cases. But there are certain drawbacks and limitations that we need to be aware of.                                                                                                                                                                                            | Human |
| Let's take the same income-expenditure example we saw above. The k-means algorithm seems to be working pretty well, right? Hold on – if you look closely, you will notice that all the clusters created have a circular shape. This is because the centroids of the clusters are updated iteratively using the mean value.                                                                                                                     | Human |
| In a GMM, the data is modeled as a weighted sum of several Gaussian distributions, each with its own mean and standard deviation. The weights of the individual Gaussians represent the proportion of the total data that comes from each component. By fitting the model to the data, we can estimate the parameters of the individual Gaussians and the weights that describe the mixture.                                                   | AI    |
| You read that right! Gaussian Mixture Models are probabilistic models and use the soft clustering approach for distributing the points in different clusters. I'll take another example that will make it easier to understand.                                                                                                                                                                                                                | Human |
| Here, we have three clusters that are denoted by three colors – Blue, Green, and Cyan. Let's take the data point highlighted in red. The probability of this point being a part of the blue cluster is 1, while the probability of it being a part of the green or cyan clusters is 0.                                                                                                                                                         | Human |
| GMMs are used in a variety of applications, including image processing, speech recognition, and data mining. They are particularly useful in situations where the underlying data distribution is not well understood, and the goal is to identify the number of clusters that best describe the data. GMMs can also be used for anomaly detection, where the model is used to identify data points that do not fit the expected distribution. | AI    |
| Now, consider the following example where the distribution of points is not in a circular form. What do you think will happen if we use k-means clustering on this data? It would still attempt to group the data points in a circular fashion. That's not great! k-means fails to identify the right clusters:                                                                                                                                | Human |

The output of GPTGone (Essay variation) that identifies which paragraphs are AI generated.

Apart from the Essay variation outputs shown previously, here are some outputs generated by the Short-Text variation of GPTGone. The Short-Text variation of GPTGone only returns the binary classification result by the model, resulting in a more succinct output.

AI

100%

The text is highly likely written by AI

Output from GPTGone (Short-Text variation) when given a piece of text from ChatGPT

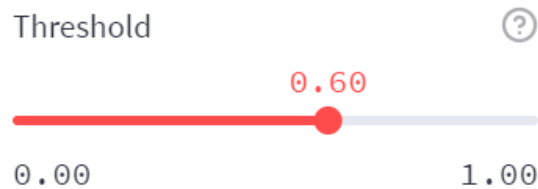
AI

0%

The text is most likely written by a human

Output from GPTGone (Short-Text variation) when given a piece of human-written text.

The percentage of the entire text being written by AI is determined by the number of paragraphs that were labeled as “AI”, divided by the total number of paragraphs (in the Short-Text variation, the entire text would be 1 paragraph). To adjust the sensitivity of the model, the threshold slider can be adjusted, increasing the threshold would cause the model to be less sensitive to AI-written content.



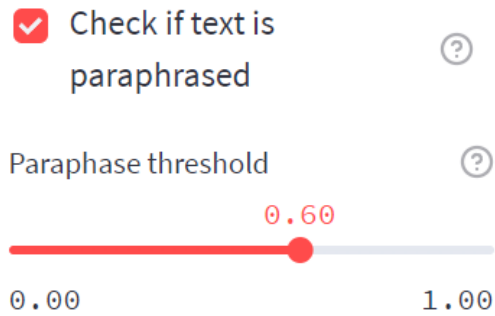
The threshold slider for adjusting model sensitivity.

GPTGone assigns the text label to the predictions by using the score it obtains and comparing it to this table. Adjusting the threshold would not affect this table, it would only affect the score that the model generates.

| Score            | Label                                      | Color |
|------------------|--------------------------------------------|-------|
| > 80%            | The text is highly likely written by AI    | Red   |
| > 60% and <= 80% | The text is likely written by AI           | Red   |
| > 40% and <= 60% | The text is may be written by AI           | Blue  |
| > 20% and <= 40% | The text is likely written by a human      | Green |
| <= 20%           | The text is most likely written by a human | Green |

## Paraphrased ChatGPT Content Detection

GPTGone is also capable of detecting ChatGPT-generated content that was paraphrased using a paraphraser like T5. To enable the paraphrasing checker, simply tick the “Check if text is paraphrased” checkbox. Doing so will display the “Paraphrase threshold” slider and the threshold can be adjusted if desired.



The image shows a user interface for enabling a paraphrasing checker. It features a checkbox labeled "Check if text is paraphrased" which is checked with a red checkmark. To the right of the checkbox is a help icon (a question mark in a circle). Below the checkbox is a slider control labeled "Paraphrase threshold". The slider has a red track and a red handle. The handle is positioned at the 0.60 mark on a scale from 0.00 to 1.00. A help icon is also present to the right of the slider label.

☒ Check if text is paraphrased ?

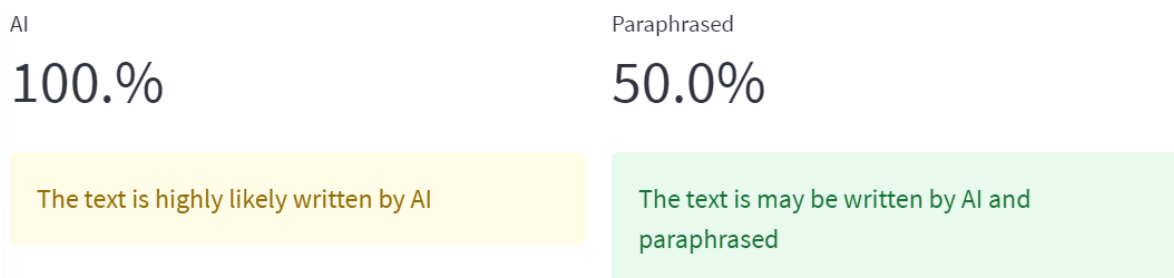
Paraphrase threshold ?

0.60

0.00 1.00

An enabled “Check if text is paraphrased” checkbox with visible “Paraphrase threshold” slider

After enabling the paraphrase checker, simply follow the previously mentioned steps on how to use GPTGone to generate the output.



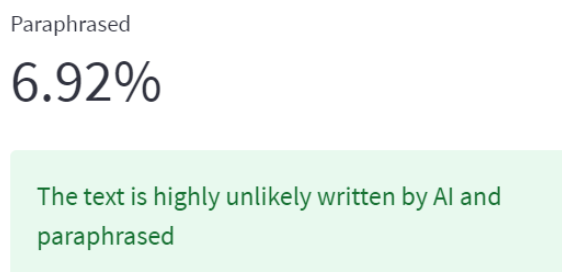
## Text analysis

|                                                                                                                                                                                                                                                                                                                                                                                                         |    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| In cardinality estimation, uniform data refers to data that is evenly distributed across a range of values. In other words, each value in the range is equally likely to occur, and there are no significant peaks or valleys in the distribution.                                                                                                                                                      | AI |
| When estimating the cardinality of a query, the database system uses statistical models to make predictions about the number of rows that will be returned. These models are based on assumptions about the distribution of the data, such as whether the data is uniform or skewed.                                                                                                                    | AI |
| If the data is uniform, the database system can use a simple statistical model, such as the uniform distribution or the equi-depth histogram, to estimate the cardinality. These models assume that each value in the range is equally likely to occur, and use this assumption to make predictions about the number of rows that will match a given query.                                             | AI |
| However, if the data is not uniform, such as if it is skewed towards certain values or if there are outliers, then more complex statistical models may be required to accurately estimate the cardinality. In such cases, the database system may use techniques such as multi-dimensional histograms or machine learning algorithms to estimate the cardinality based on the distribution of the data. | AI |

Legend: A red dotted border indicates that the text is paraphrased

Output from GPTGone (Essay variation) when part of the input is paraphrased from ChatGPT-generated content

Apart from the Essay variation outputs shown previously, here is an example of the paraphrase detection output generated by the Short-Text variation of GPTGone. The Short-Text variation of GPTGone only returns the paraphrased probability, generated by the model, resulting in a more succinct output.



Paraphrasing score output from GPTGone (Short-Text variation) when given a human-written text.

Similar to how the percentage of the entire text being written by AI is calculated, the percentage of the entire text being AI-paraphrased<sup>7</sup> from GPTGone's Essay variation output is calculated by the paragraphs that are labeled as AI-paraphrased, divided by the total number of paragraphs. For the Short-Text variation, the percentage of the entire text being AI-paraphrased is simply the probability generated by the paraphrase detection model.

The table below illustrates how GPTGone assigns the text label to the predictions; it does so by using the paraphrasing score it obtains and comparing it to this table. Adjusting the paraphrase threshold would not affect this table, it would only affect the score that the model generates.

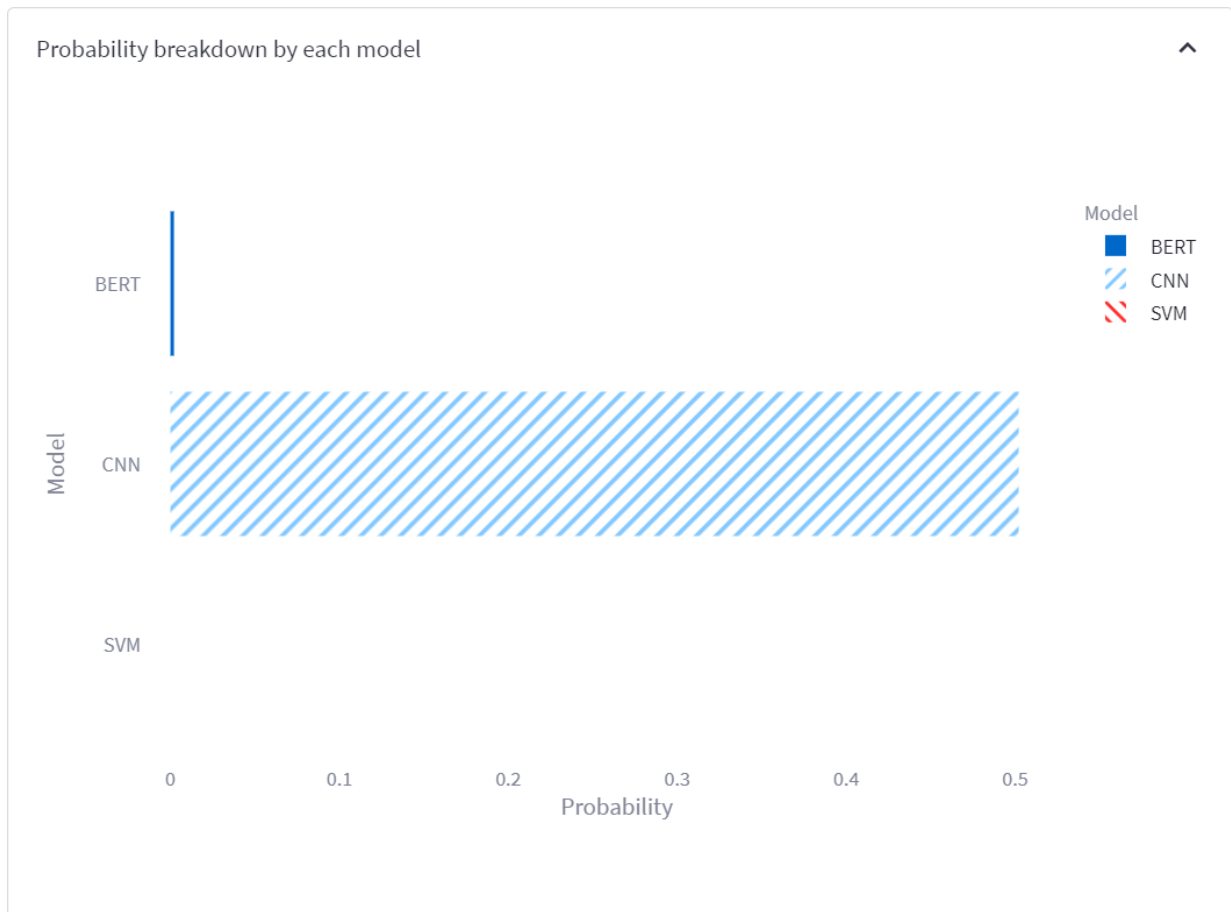
| Score            | Label                                                     | Color |
|------------------|-----------------------------------------------------------|-------|
| > 80%            | The text is highly likely written by AI and paraphrased   | Red   |
| > 60% and <= 80% | The text is likely written by AI and paraphrased          | Red   |
| > 40% and <= 60% | The text is may be written by AI and paraphrased          | Blue  |
| > 20% and <= 40% | The text is unlikely written by AI and paraphrased        | Green |
| <= 20%           | The text is highly unlikely written by AI and paraphrased | Green |

---

<sup>7</sup> AI-paraphrased refers to a piece of text that was generated by ChatGPT and subsequently paraphrased.

## Probability Breakdown by Each Model

GPTGone is able to display the individual predictions of each model, for users who are curious about the predictions of the constituent models in the ensemble.



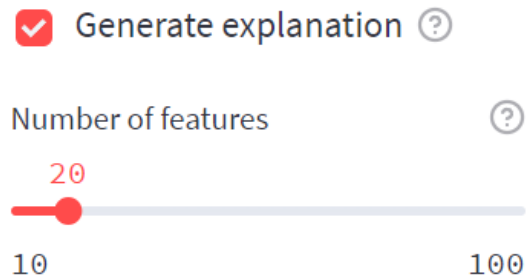
A breakdown of each model's prediction.

There is a difference in how the probabilities are calculated between the Short-Text and Essay variations. For the Short-Text variation, the probability breakdown is just the individual predictions of each model in the ensemble. On the other hand, for the Essay variation, the probability breakdown is the average prediction of each model across all the paragraphs.



## Explainability Generation

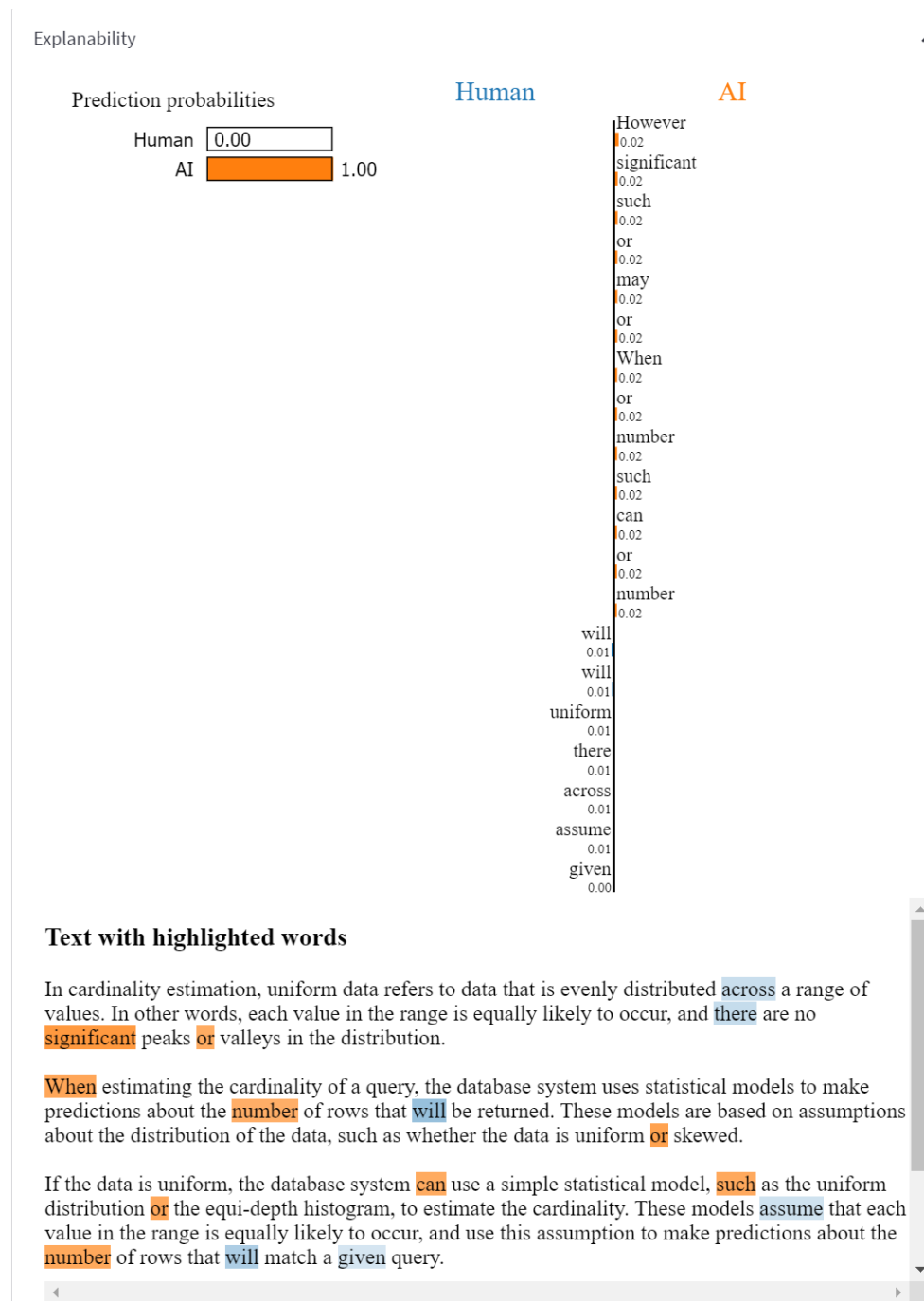
For the users that would like to understand why their input is classified as AI-written or human-written, GPTGone offers an explanation generator as well. To use the explanation generator, simply tick the “Generate explanation” checkbox, which will then display the slider to choose the number of features desired in the explanation. The greater the number of features selected, the more time it takes to generate the explanation. Currently, the explanation generator only supports explanations by our SVM model due to the expensive computational costs of the BERT and CNN models.



The image shows a user interface for generating explanations. At the top, there is a checkbox labeled "Generate explanation" with a red checkmark and a help icon (a circle with a question mark). Below this, there is a slider control labeled "Number of features" with a help icon. The slider has a red handle positioned at the value 20. The range of the slider is from 10 to 100, with the numbers 10 and 100 displayed at the ends of the track.

An enabled “Generate explanation” checkbox with visible “Number of features” slider.

After enabling the explanation generation and selecting the number of features, simply follow the previously mentioned steps on how to use GPTGone to generate the output.



An explanation report generated by GPTGone.

The explanation report highlights which part of the text contributes to the labeling of the paragraph as “AI” or “Human”, allowing users to better understand why the text is labeled as “AI” or “Human”.

## Cyrillic Character Detection

Another important feature of GPTGone is the capability of detecting the Cyrillic alphabet, to prevent bypassers from converting text to the Cyrillic alphabet and bypassing GPTGone.

Text to analyze

экономиа3r4

Check if written by AI

Reset

The text contains cyrillic characters, which is not supported by GPTGone

Output generated by GPTGone when cyrillic characters are detected in the input.

## GPTGone Lite

In addition to our full-fledged ChatGPT detector, GPTGone, we also developed a lightweight version of it that uses just the SVM model for classification. GPTGone Lite has the same text annotation capability as GPTGone, but it does not have the paraphrasing detection and explanation generation feature. This product is suited for applications where performance is extremely important and sacrificing a little accuracy is acceptable. Instructions for using GPTGone Lite are the same as using GPTGone's ChatGPT detection feature.



## GPTGone Lite

This model is optimized for speed, while sacrificing a little accuracy. It features a SVM with an accuracy rate of 95%. For more information, visit us on our github page <https://github.com/ffeew/GPTGone>

Variation



Essay



Text to analyze

Check if written by AI

Reset

GPTGone Lite's user interface.

# How to run GPTGone locally

Requirement: python version > 3.9

1. If git lfs is not installed, install it using this command: `git lfs install`:
2. Clone the repository from <https://huggingface.co/spaces/GPTGone/GPTGone>
  - a. Command: `git clone https://huggingface.co/spaces/GPTGone/GPTGone`
3. Change your directory into GPTGone using: `cd GPTGone`
4. Create a new virtual environment at the root of the project directory.
  - a. Command: `py -m venv venv`
  - b. Guide for Pycharm: [Link](#)
5. Activate the virtual environment.
  - a. Mac command: `source ./venv/bin/activate`
  - b. Windows command: `venv\Scripts\Activate.ps1`
  - c. Guide for Pycharm: [Link](#) (Change to use C:\WINDOWS\system32\cmd.exe for Shell Path in Windows if you have problems activating your virtual environment)
6. Install the requirements using: `pip install -r requirements.txt`
  - a. May take awhile so you can let it run in the background
7. Run the command: `streamlit run App.py` to spin up the local server.
8. Visit the URL displayed on the terminal to open up the web application.

```
o (venv) aikho > GPTGone > main > ✓ streamlit run App.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://10.12.189.47:8501

A new version of Streamlit is available.

See what's new at https://discuss.streamlit.io/c/announcements

Enter the following command to upgrade:
$ pip install streamlit --upgrade
```

Example output from the terminal after running 'streamlit run App.py'

# Conclusion

Overall, it seems like the classification task of identifying AI-generated text in the ELI5 domain is more contingent on the local context than the global context of the text (such as dependencies between words). Our best-performing models were mainly the classical machine learning models like SVM and Naive Bayes. This is a useful finding as classical machine learning models are generally faster to train and have faster prediction times due to simpler computation, making them easy to deploy on online forums that are interested in detecting ChatGPT. For future work, we could improve the performance of deep learning-based models like BERT and CNN by obtaining more data since deep learning's performance usually exceeds classical Machine learning when there is more high-quality data. More mechanisms can be put in place to prevent overfitting, like by adding L1 or L2 regularization. More features or methods of incorporating the features could be experimented with as well as the features that we explored: perplexity, GLTR and language features did not work as well as expected.

# References

- Alba, D. (2022, December 8). CHATGPT, open ai's chatbot, is spitting out biased, sexist results. Bloomberg.com. Retrieved April 12, 2023, from <https://www.bloomberg.com/news/newsletters/2022-12-08/chatgpt-open-ai-s-chatbot-is-spitting-out-biased-sexist-results>
- Bowman, E. (2023, January 9). *A college student made an app to detect AI-written text*. NPR. Retrieved April 12, 2023, from <https://www.npr.org/2023/01/09/1147549845/gptzero-ai-chatgpt-edward-tian-plagiarism>
- da Costa, L. S., Oliveira, I. L., & Fileto, R. (2023). Text classification using embeddings: a survey. *Knowledge and Information Systems*, 1-43.
- Guo, B., Zhang, X., Wang, Z., Jiang, M., Nie, J., Ding, Y., Yue, J., & Wu, Y. (2023, January 18). *How Close is ChatGPT to Human Experts? Comparison Corpus, Evaluation, and Detection*. arXiv. Retrieved April 14, 2023, from <https://arxiv.org/abs/2301.07597>
- Gehrmann, S., Strobelt, H., & Rush, A. M. (2019). GLTR: Statistical detection and visualization of generated text. *arXiv preprint arXiv:1906.04043*.
- Heylighen, F., & Dewaele, J.M. (2002). Variation in the contextuality of language: An empirical measure. *Context in Context, Special issue of Foundations of Science*, 7 (3), 293-340.
- Karl, F., & Scherp, A. (2022). Transformers are Short Text Classifiers: A Study of Inductive Short Text Classifiers on Benchmarks and Real-world Datasets. *arXiv preprint arXiv:2211.16878*.
- Ketterer, S. (2023, April 10). Despite plagiarism concerns, Houston colleges see chatgpt as a tool for teaching, not a problem. Houston Chronicle. Retrieved April 12, 2023, from <https://www.houstonchronicle.com/news/houston-texas/education/article/rice-uh-hcc-embrace-chatgpt-despite-plagiarism-17882497.php>
- KOWALCZYK, A. (2014, October 19). *Linear Kernel: Why is it recommended for text classification ?* SVM Tutorial. Retrieved April 12, 2023, from <https://www.svm-tutorial.com/2014/10/svm-linear-kernel-good-text-classification/>
- Mitrović, S., Andreoletti, D., & Ayoub, O. (2023). ChatGPT or Human? Detect and Explain. Explaining Decisions of Machine Learning Model for Detecting Short ChatGPT-generated Text. *arXiv preprint arXiv:2301.13852*.
- Papers With Code. (2023). *ELI5: Long Form Question Answering*. Papers With Code. Retrieved April 12, 2023, from <https://paperswithcode.com/paper/eli5-long-form-question-answering>

Papers With Code. (2023). *Text Classification*. Papers With Code. Retrieved April 12, 2023, from <https://paperswithcode.com/task/text-classification>

Wild, B. (2023, April 9). CHATGPT: Cardiff students admit using AI on essays. BBC News. Retrieved April 12, 2023, from <https://www.bbc.com/news/uk-wales-65167321>

Sullivan, C. O. (2023, March 12). *Introduction to SHAP with Python. How to create and interpret SHAP plots...* | by Conor O'Sullivan. Towards Data Science. Retrieved April 12, 2023, from <https://towardsdatascience.com/introduction-to-shap-with-python-d27edc23c454>



# Addendum

## Sample Outputs from Bypassers

ChatGPT answer based on the question “Explain quantum computing in simple terms” before and after Bypassers:

| Text before Bypasser                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Text after Bypasser<br>(Gonzoknows/AI-Detection-Bypassers)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Text after Bypasser<br>(Declipsonator/GPTZzzs)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Quantum computing is a type of computing that uses quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Unlike classical computers that use bits to represent information and perform calculations, quantum computers use quantum bits (qubits) which can be in multiple states at the same time. This allows quantum computers to perform certain calculations much faster than classical computers, making them well-suited for solving complex problems in fields like cryptography, chemistry, and machine learning. | Quantum computing is a type of computing that uses quantum-mechanical phenomena, such as superposition and entanglement, to perform operations on data. Unlike classical computers that use bits to represent information and perform calculations, quantum computers use quantum bits (qubits) which can be in multiple states at the same time. This allows quantum computers to perform certain calculations much faster than classical computers, making them well-suited for solving complex problems in fields like cryptography, chemistry, and machine learning. | Quantum computing is a typecast of computing that uses quantum-mechanical phenomena, such as superposition and entanglement, to do operations on information. Unlike graeco-roman computers that use bits to correspond entropy and do calculations, quantum computers use quantum bits (qubits) which can be in multiple states at the very same time. This allows quantum computers to do sure calculations often faster than classic computers, making them well-suited for solving composite problems in fields similar steganography, chemistry, and machine learning. |

## Resource Links

Git repository: <https://github.com/ffeew/GPTGone>

GUI / demo / source code: <https://huggingface.co/spaces/GPTGone/GPTGone>

Models: [https://huggingface.co/GPTGone/GPTGone\\_ensemble/tree/main](https://huggingface.co/GPTGone/GPTGone_ensemble/tree/main) or [Drive link](#)

Datasets: [https://huggingface.co/datasets/GPTGone/hc3\\_v2](https://huggingface.co/datasets/GPTGone/hc3_v2)