IBDP Computer Science Extended Essay

**Topic**: AI-Generated Text Detection

**Research question:** To what extent can evaluating the perplexity and burstiness of a given piece of text help in determining whether the text was written by a human or generated by AI?

**Subject**: Computer Science

**Session**: May 2025

**Word count**: 3998

Table of Contents

[1. Introduction 4](#_Toc188858920)

[1.1 Societal applications and Importance of AI-generated text detection 4](#_Toc188858921)

[1.2 The approach to detecting AI-generated text 4](#_Toc188858922)

[2 Theoretical Background 5](#_Toc188858923)

[2.1 Large Language Model (LLM) 5](#_Toc188858924)

[2.1.1 Tokenization 5](#_Toc188858925)

[2.1.2 predicting 5](#_Toc188858926)

[2.2 Calculating the Perplexity and Burstiness of a Given Text 6](#_Toc188858927)

[2.2.3 Random Forest: 8](#_Toc188858928)

[3. Experiment Methodology 8](#_Toc188858929)

[3.1 The Datasets Used: 8](#_Toc188858930)

[3.2 Processing the Dataset: 9](#_Toc188858931)

[3.2.1 Extraction and Labelling: 9](#_Toc188858932)

[3.2.2 Storage Format: 9](#_Toc188858933)

[3.2.3 Standardisation: 10](#_Toc188858934)

[3.3 Variables: 10](#_Toc188858935)

[3.3.1 Independent Variable 10](#_Toc188858936)

[3.3.2 Dependent Variable 10](#_Toc188858937)

[3.3.2 Controlled Variable 10](#_Toc188858938)

[3.3.4 Uncontrolled Variable 11](#_Toc188858939)

[3.4 Metrics calculation: 11](#_Toc188858940)

[3.5 Training classifier based on the metrics: 12](#_Toc188858941)

[4. Experimental results 13](#_Toc188858942)

[4.1 perplexity and Burstiness distribution 13](#_Toc188858943)

[4.2 prediction accuracy 14](#_Toc188858944)

[5. Result analysis: 14](#_Toc188858945)

[5.1 Perplexity and burstiness distribution: 14](#_Toc188858946)

[5.2 prediction accuracy: 15](#_Toc188858947)

[6. Discussion: 16](#_Toc188858948)

[7. Conclusion 18](#_Toc188858949)

[Bibliography: 19](#_Toc188858950)

[Appendix A – Hyperparameter of the classifiers: 21](#_Toc188858951)

[Appendix B – program: 21](#_Toc188858952)

[Data preprocessing - Xsum: 21](#_Toc188858953)

[Data preprocessing - SQuARD: 25](#_Toc188858954)

[Data preprocessing - WritingPrompts: 27](#_Toc188858955)

[Displaying perplexity graph: 28](#_Toc188858956)

[Calculation of metrics: 32](#_Toc188858957)

[fine-tuning random forest: 37](#_Toc188858958)

[Appendix C – GPT-2 tokenizer performance: 42](#_Toc188858959)

[Appendix D – AUROC score visual representation: 43](#_Toc188858960)

[Appendix E – comparing different classifiers performance 44](#_Toc188858961)

# 1. Introduction

## 1.1 Societal applications and Importance of AI-generated text detection

As technology advances and interest in artificial intelligence grows, one of the most fascinating achievements in the field of machine learning is the ability of AI algorithms to generate text. AIGT (AI-generated text) can help people in every aspect of their lives, such as increasing the speed of research or providing new ideas. There are many commercial large language models (LLMs) with a huge variety of knowledge available to communicate with, namely ChatGPT from OpenAI, Gemini from Google, and others.

However, along with the development of these technologies, challenges arise, particularly concerning authenticity and credibility. Within journalism and social media, AIGT could be inconsistent and incorrect due to the nature of LLM (detail in section 2.1.2) that the content generated is based on probability, which makes readers misinformed. A way of preventing this situation is to make the reader aware that the text they are reading is generated by AI. Additionally, in academic dishonesty detection, ai has replaced plagiarism as a popular cheating tool, it can help identify and prevent academic dishonesty by detecting content that may be AI-generated.

## 1.2 The approach to detecting AI-generated text

AI text detection methods include Watermarking, Statistical and Stylistic Analysis, Language Model-Based Classification, and Off-the-Shelf Detection Tools (Fraser, Dawkins & Kiritchenko, 2024). This essay focuses on combining Statistical Analysis and Language Model-Based Classification to answer the research question. Watermarking is excluded as it impacts company profits (Davis, 2024). Stylistic Analysis is limited by variations in text domains, such as differences in style and lexical diversity between content types (Chakraborty et al., 2023). Off-the-Shelf Detection Tools, which integrate multiple models into complex frameworks for accuracy, are also not analysed.

Statistical Analysis identifies AI-generated text (AIGT) by detecting patterns in the probability distributions used by language models. A key measure is perplexity, indicating how well a model predicts the next word. AIGT typically shows lower perplexity and burstiness than human text, which is more diverse. Language Model-Based Classification, using pre-trained models like BERT and GPT-2, complements this analysis but requires significant resources and domain-specific data (Fraser, Dawkins & Kiritchenko, 2024).

This essay focuses on perplexity and burstiness, as GPTZero uses them to detect AIGT (Chakraborty et al., 2023). Unlike stylistic analysis, burstiness offers universal applicability, especially in varied, structured texts like academic writing.

# 2 Theoretical Background

## 2.1 Large Language Model (LLM)

LLMs are very large deep-learning models that are pre-trained on vast amounts of data.(*What is LLM? - Large Language Models Explained - AWS*, no date) The model is trained with texts, and its purpose is to communicate with humans. Since the goal of LLMs was to sound like humans as much as possible, the detection was destined for hardship due to the low interpretability of LLMs. In this session, LLMs will be explained at a level to understand burstiness in the next sub-section.

### 2.1.1 Tokenization

Since computers store only 0 and 1, computer scientists developed a process to convert human language into numeric representations, enabling computers to understand word relationships and polysemes in different contexts.

Tokenization, a key aspect of Natural Language Processing (NLP), converts text into smaller parts with numeric representations called tokens and token IDs (Ali Awan, 2023). The GPT-2 tokenizer, used as an example, has a vocabulary of 50,257 tokens, including prefixes, suffixes, roots, characters, or symbols, each assigned a number. The tokenizer assigns token IDs to known words and handles out-of-vocabulary (OOV) words by breaking them into sub words or letters within its vocabulary and assigning tokens using the BPE algorithm.

|  |  |  |
| --- | --- | --- |
| **Original Word** | **Tokenized Output** | **Token IDs** |
| resourcefulness | ['resource', 'fulness'] | [15432, 23145] |

*Table 1.* Example of tokenization using BPE in a large language model like GPT-2.

The BPE algorithm is beyond the scope of this essay, so it will not be discussed in detail, It can be found at (Sennrich, Haddow and Birch, 2016).

### 2.1.2 predicting

This part of the process involves taking input and generating responses. Most generative models today are based on transformer models. Words are represented numerically and then transformed into vectors matching the vocabulary size—a process called word embedding.

A screenshot of a computer

Description automatically generated

*Figure 1.* Word predictions based on LLM, credits: https://perplexity.vercel.app/

The model generates responses by predicting the most probable next word based on previous ones. However, AI doesn't truly understand what it generates and relies on training data, which can lead to misinformation. From Figure 1, we can see that the model is confident of the word “House” in the phrase “the White House” by marking it green because that is common sense. Besides, after “his…”, the model can’t predict what will be the next word because there are too many possibilities.

## 2.2 Calculating the Perplexity and Burstiness of a Given Text

From the previous section, we know that the model will analyse the probability of the next word, and each place will have multiple possible words with different possibilities. Therefore, we can determine how certain the model thinks about the next word within a paragraph. For example, if a sentence has already “How” and “are”, the next word would probably be “you”. However, there are also other possibilities of the sentence being “How are the dogs?”. Therefore, “the” could also be an option, but it is less likely to be the right one than “you”. Let’s say the sentence that is given to the model is literally, “How are the dogs?”. The model would be questioning that it does not agree with the sentence. Therefore, the perplexity of the sentence is higher than “How are you?” since it is more common.

In mathematical representation, perplexity is given by(*What is perplexity? | Continuum Labs*, 2024):

The part is the cross-entropy of one word. If the probability of a word in a certain place is small, then the log of it will be large. Since the probability can only be between 0 and 1, and the value of the logarithm is always negative between the intervals, we take the opposite of the value. Then, we take the mean of all cross-entropy and do an exponentiating calculation. Then we get the perplexity.

From the equation, we can observe that the perplexity depends on the model used to check the text, if the model is good, there will probably be lower perplexity of the texts.

Burstiness is an additional measure of perplexity to further evaluate the given text. In the scenario, burstiness is fundamentally about changes within the text. For example, if a very simple sentence is followed by a long and complex sentence. We can see that the style of writing is changing, meaning the perplexity is varying. And that’s what human writers tend to have in our writing; we get carried away and improvise(Kolev, 2023). In contrast, the AIGT sounds uniform and regular.

In order to quantise burstiness, we split the text into segments and calculate the perplexity of each segment. In this way, the line segments will have different perplexities. As previously mentioned, the human written text will tend to vary the complexity of the sentence; therefore, the standard deviation of perplexity will be larger. In practice, we can divide the standard deviation of perplexity by the mean of perplexity to avoid directly comparing the standard deviation, which is similar to the Fano Factor and measures the dispersion of a counting process(‘Fano factor’, 2024). The greater the burstiness is, the more likely the text is written by humans.

## 2.2.3 Random Forest:

The random forest classifier is used for distinguishing between AIGTs and human texts using perplexity and burstiness values. It uses supervised learning, meaning that we have to first give it some labelled data for training and then use the trained model to predict an unknown piece of text. The full introduction of random forest classifier can be found in this paper (Louppe, 2015).

# 3. Experiment Methodology

This paper will conduct an experimental approach and will use the primary data as the main source of data. To evaluate the classifier's performance based on perplexity and burstiness, we will compare them with other detection approaches, such as Log-Likelihood Log-Rank Ratio (LRR)(Su *et al.*, 2023) and entropy(Moradi, Grzymala-Busse and Roberts, 1998) , since they all use statistical properties. The formula for these metrics is not going to be introduced in this essay. This comparison aims to understand the strengths and limitations of different methods and have an illustration of how reliable the perplexity and burstiness are. An experimental methodology was chosen because there is a lack of secondary data available online to answer this paper’s research question. Furthermore, self-implementing the algorithms allows us to use a balanced database involving a range of diverse scenarios to validate the metrics. For this purpose, I created a Python program to access datasets, tokenise texts, calculate perplexity and burstiness, use a random forest classifier to classify between the AIGT and human texts based on the calculated perplexity and burstiness and output the result by plotting the graphs, using the methods described in section 2.2. The results will be presented and analysed in this section. The original program is presented in Appendix B.

## 3.1 The Datasets Used:

In this scenario, both datasets with AIGT and human text are required. For the AI-generated text, we used the OpenOrca dataset, which contains millions of responses from GPT-3.5 and GPT-4 with system prompts and questions across a diverse range of topics (Mitra *et al.*, 2023). The OpenOrca dataset’s scale and diversity make it ideal for capturing a wide range of AI outputs, ensuring robustness in evaluation. No additional AIGT datasets were deemed necessary due to the comprehensiveness of OpenOrca.

For human texts, following (Chen *et al.*, 2023), three datasets were chosen to represent human text:

* XSum (Narayan et al., 2018): Contains concise news articles, emphasizing factual accuracy.
* SQuAD (Rajpurkar et al., 2016): Features Wikipedia paragraphs, providing structured and information-dense text.
* WritingPrompts (Fan et al., 2018): Includes creative stories, capturing imaginative and varied text styles.

These datasets were selected because they reflect areas where LLMs are frequently used and where their limitations may have significant consequences, such as misinformation in news or reduced creativity in storytelling.

Each experiment combines 5000 human-generated texts and 5000 AIGT samples into a new dataset, with 80% used for training and 20% for testing.

While XSum provides concise and factual articles, its structured nature may reduce variance in human writing, potentially affecting burstiness results. WritingPrompts, though diverse, emphasizes creativity, which could exaggerate perplexity variations. SQuAD's focus on structured paragraphs may limit the representation of more conversational or informal human text.

*Exclusion of Other Datasets:* Datasets like COCOCaptions or OpenWebText were excluded due to their limited scope in representing real-world human text complexity and diversity.

## 3.2 Processing the Dataset:

Before training and testing the models, the raw datasets are pre-processed by the steps.

### 3.2.1 Extraction and Labelling:

5000 samples are extracted from each human text dataset and 5000 from the OpenOrca dataset. A combined file of 10,000 samples is created for each experiment, with AIGT labelled as “1” and human text as “0.”

### 3.2.2 Storage Format:

The pre-processed data is stored in CSV files with two columns:

|  |  |
| --- | --- |
| **Text** | **Label** |
| Sample human text... | 0 |
| Sample AI text... | 1 |

*Table 2.* The storage format.

### 3.2.3 Standardisation:

To ensure fairness, the datasets are tokenized consistently using the GPT-2 tokenizer. The tokenization process ensures uniform handling of text across all samples.

## 3.3 Variables:

### 3.3.1 Independent Variable

The independent variable of the experiment will be the origin of the text, which can come from different datasets.

### 3.3.2 Dependent Variable

The accuracy of the classifiers’ output is the dependent variable measured in this paper. However, the intermediate dependent such as perplexity, burstiness, LRR, and entropy of the text, are also analysed. Their pattern has a direct effect on accuracy.

### 3.3.2 Controlled Variable

|  |
| --- |
| Algorithm 1 Perplexity & burstiness AIGT detection |
| 1: Input: passage , numbers of words , dimensions of word embeddings , segments , number of segments , source model , decision set |
| 2: *()* //log perplexity for in Eq. 2.2.1 |
| 3: //mean perplexity of in Eq. 2.2.2 |
| 4: //standard deviation of  in Eq.2.2.3 |
| 5: //burstiness of in Eq.2.2.4 |
| 6: **If** **Then** |
| 7: Return True //probably AIGT |
| 8: **else** |
| 9: return false //probably not AIGT |

*\*() represents the perplexity of the given text , the probability of source model will be shown as ().*  represents the burstiness of the given text of source model

We can see from algorithm 1 that the detection process deeply connects to the given LLM , so that the choose of language model should be the language model that outputs the probabilities of the next word. In this experiment, the GPT-2 is selected because it offers a robust and widely tested baseline for generating predictions according to research done by (Su *et al.*, 2023) with data presented in Appendix C. This ensures consistency across experiments, as all metrics depend on the probabilities predicted by the model. By standardising the tokenizer and model, the results are isolated from variations in model architecture, allowing a focus on the text properties and dataset origin.

Another variable to be controlled is the classifier. This essay choose the random forest classifier to determine the decision set , because it provides a more linear performance against number of sequence so that the result would be more stable with different number of samples (Chakraborty *et al.*, 2023). The graph is shown in appendix E.

The sample sizes are also under controlled (5000 AIGT and 5000 human texts) to avoid imbalanced training and testing sets.

### 3.3.4 Uncontrolled Variable

External factors such as inherent stylistic differences in human text, dataset quality, and biases in LLM-generated text could introduce noise into the metrics, potentially affecting classification accuracy. A mitigation strategy that we have used is to test on multiple datasets that represent a variety of text domains, aims to minimise the impact of such variability.

## 3.4 Metrics calculation:

At the beginning of the experiment, we must calculate the metrics, including perplexity and burstiness, by formula 2.2.1 and algorithm 1. As prementioned, I expect the perplexity of human texts to be higher than the AIGTs. The same as burstiness, human texts should have a greater perplexity variation across the paragraph. I introduced that the burstiness requires splitting original texts into segments in the background sections. In this experiment, the segments will be obtained by splitting the sentences using punctuations (e.g. ‘,’/’.’/’?’) to ensure each segment is a complete sentence.

A kernel density plot is used to visualize the distribution of perplexity and burstiness. These plots smooth data points into a continuous curve, offering a clear view of density distributions. LRR and entropy are included for comparison, but their density plots are not discussed in detail within this essay, as the primary focus is on perplexity and burstiness.

Metrics like perplexity and burstiness are inherently influenced by dataset characteristics. For concise datasets like Xsum, these often exhibit lower variability due to their focus on clarity, which might affect burstiness scores. On the other hand, for creative datasets like WritingPrompts, these encourage linguistic diversity, resulting in higher perplexity and burstiness scores for human text.

The importance of this context cannot be overstated. While the metrics themselves are powerful, their interpretation must account for dataset-specific nuances, as demonstrated in section 5.1.

## 3.5 Training classifier based on the metrics:

There will be nine random forest classifiers created for each approach among perplexity and burstiness, LRR and entropy and each pre-processed dataset. LRR and entropy are the baselines to evaluate the result of the essay’s approach, and the scores will be compared to other research papers. Each classifier will be fine-tuned to fit the dataset to ensure fairness instead of using the same hyperparameter for all classifiers by grid search. This algorithm tries all possible combinations of the hyperparameters, and it is especially suitable for random forests because the hyperparameters are discrete integers (the depth level, minimum number of sample splits and minimum number of sample leaves); the tunned hyperparameter will be presented in Appendix A. The algorithms are going to be evaluated using the AUROC score. The AUROC score indicates the separability of a model, meaning how well the model can separate the AIGT and human-generated text. An AUROC value close to 0.5 indicates poor separability of the model, and an AUROC value close to 1 indicates good separability. The score is usually presented with the AUROC diagram, but since it’s not necessary, the diagrams can be found in Appendix D.

# 4. Experimental results

## 4.1 perplexity and Burstiness distribution

A graph of a function

Description automatically generatedA graph of a function

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| *Figure 2.* Perplexity Distribution of Human vs. Machine Text Samples for XSum & Orca Dataset | *Figure 3.* Burstiness Distribution of Human vs. Machine Text Samples for XSum & Orca Dataset |

A graph of a number of people

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| *Figure 4.* Perplexity Distribution of Human vs. Machine Text Samples for SQuAD & Orca Dataset | *Figure 5.* Burstiness Distribution of Human vs. Machine Text Samples for SQuAD & Orca Dataset |

A graph of a function

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| *Figure 6.* Perplexity Distribution of Human vs. Machine Text Samples for WritingPrompts & Orca Dataset | *Figure 7.* Burstiness Distribution of Human vs. Machine Text Samples for WritingPrompts & Orca Dataset |

## 4.2 prediction accuracy

|  |  |  |  |
| --- | --- | --- | --- |
| Method\Dataset | XSum | SQuAD | WritingPrompts |
| Perplexity & Burstiness | 0.86\* | 0.80\* | **0.98** |
| LRR | 0.81 | 0.76 | 0.78 |
| entropy | **0.99** | **0.83** | 0.97 |
| Diff | -0.13 | -0.03 | 0.01 |

*Table 3.* AUROC for detecting samples from the given model on the given dataset for Perplexity&Burstiness and two previously proposed criteria (30000samples used for evaluation) classified by fine-tuned random forest classifiers. Bold shows the best AUROC score within each column (model-dataset combination); asterisk (\*) denotes the second-best AUROC. Values in the final row show Perplexity&Burstiness’ AUROC over the strongest baseline method in that column.

# 5. Result analysis:

## 5.1 Perplexity and burstiness distribution:

In figure 2, we can see that the red-shaded area is very concentrated, with a tall and thin shape between 15 and 40. The tall shape does not indicate that the AIGT has higher perplexity. Instead, it indicates that the majority of the AIGT has perplexity between the range 15-40. AIGTs tend to be more uniform and regular, whereas the human text has a larger range of perplexity, indicating the creative diversity of human beings. By visualising the distribution of the perplexities in practice, it is more confident to say the theory seems to be correct.

In Figure 3, the human text has reached its peak at a point very close to zero, indicating that in this dataset, the human text has low burstiness scores overall. When we move our attention to the red area, which represents the burstiness of AIGTs, its peak is slightly broader and extends further to the right than the human curve, indicating that AIGTs have a wider range of burstiness values. However, this result shows that the burstiness is lower in human text and higher in AIGT, which contradicts the theory.

In figure 4, the human text perplexity has a peak at a lower range of around 15, meaning that in this dataset, the human text is generally more predictable or fluent according to the model. The red curve lies on the right of the blue curve, showing that it generally has slightly higher perplexity values. Moreover, the significant overlap between the two areas tells us that to use perplexity as a metric to classify human text and AIGT might be less reliable.

In figure 6, perplexity is still lower, but there is less overlap between the points. Although it does not follow our assumption about perplexity, it is still possible to use it as a metric for classification. Figure 7 shows that the AIGTs have more density in the high burstiness range, and a wider peak compared to human text. This indicates that machine-generated text exhibits more variability, with higher burstiness scores.

In conclusion, the perplexity and burstiness distribution is not the same as we assumed. Human-written text shows a general pattern with relatively low burstiness, and perplexity compared to AIGTs. This clearly illustrates how powerful the LLMs are due to their high perplexity and burstiness, which increases the difficulty of AIGT detection. Nevertheless, we still have to find out potential reasons why the burstiness and perplexity of human-generated text is lower. For datasets like SQuAD or XSum, where clarity and conciseness are valued, human authors are likely to produce text that’s easier to understand and predict, hence lower perplexity and burstiness. On the other hand, the system prompt in Orca that is given to ChatGPT has never explicitly asked for written style, and the machine may have more freedom to generate different styles of responses.

## 5.2 prediction accuracy:

The classifier based on perplexity and burstiness achieves an AUROC score of 0.86, indicating strong discriminatory power as AUROC scores above 0.8 denote excellent performance (‘Measuring Performance: AUC (AUROC)’, 2019). This shows it can effectively distinguish between human and AI-generated text with occasional misclassification. The LRR classifier has a slightly lower AUC score of 0.81, showing reasonable but less effective performance.

The entropy classifier achieves an AUROC score of 0.99, demonstrating near-perfect discrimination. This is supported by the entropy distribution of the Xsum and Orca dataset, where minimal overlap highlights its reliability. While entropy is excellent for structured news summaries, perplexity and burstiness provide additional value.

On the SQuAD and Orca dataset, all classifiers show reduced performance. Entropy drops from 0.99 to 0.83 but remains the best, though the decline suggests question-answering text’s predictability reduces its efficacy. Perplexity and burstiness remain stable, with minor fluctuations.

For WritingPrompts, the perplexity and burstiness model achieves an AUROC of 0.98, and entropy scores 0.97. Across nine evaluations, entropy is the most effective overall. LRR, while decent, is less informative on its own. Both entropy and perplexity-burstiness excel at distinguishing human from AI text in creative contexts, where variability and complexity highlight their strengths.

# 6. Discussion:

The best single measure for identifying AI-generated text was entropy, which achieved the highest AUROC scores and proved to be the most dependable and efficient metric across all datasets. Perplexity and burstiness also performed well, particularly in creative writing scenarios, as demonstrated by the WritingPrompts dataset, where they captured crucial characteristics such as diversity and unpredictability, which are often indicative of human-written material. However, LRR consistently achieved the lowest AUROC scores over all the dataset, which is the least beneficial metric. This did not take away the possibility for LRR's value to be added in addition to other measures enhancing their strengths. However, these results differ from previous research, like (Su et al., 2023), where LRR was determined to be the best AIGT detecting algorithm and entropy the least effective. There is likely a methodological difference here, as (Su et al., 2023) used self-trained LLMs to compute word probabilities while this paper used GPT-2, pre trained on different datasets. It sheds light into the huge influence that training data and model architecture can have on metrics like perplexity and burstiness. These findings underscore a critical limitation of AIGT detection: the variation in results stems from the quality, balance and biases of the underlying data, and as such there is variation in outcomes across other investigations.

The type of content being analysed also affected how effective each metric would be. However, the utility of entropy in structured text, like the question-answering formats in the SQuAD dataset, was decreased due to limited variability and limited entropy’s ability to differentiate between human and AI generated text. Unlike writing prompts, the WritingPrompts dataset gave more of a contrast between human and machine generated text and it is these metrics (perplexity, burstiness, entropy, etc) that did a good job of isolating key characteristics. It was shown that perplexity and burstiness are good signatures for differentiating AI and human writing, but their effectiveness skyrocketed when entropy is brought into the mix.

Perplexity and burstiness are effective at distinguishing human and AI generated text when combined with other metrics such as entropy, to answer the research question. They may not be sufficient on their own, especially in structured datasets, but in creative, diverse situations, they can give tremendous discrimination. An approach based on multiple metrics, with entropy being the prime indicator and providing additional information via perplexity and burstiness, is the most robust way to detect AI generated content for a variety of text formats. Complementarity of these metrics boosts detection accuracy for various text forms.

Such findings have implications for the development of the text detection tools based on AI generated text. By combining the statistical metrics like entropy with neural network-based classifiers, the hybrid detection systems could provide a great boost to the reliability of detection in various domains. For instance, the combination of perplexity and burstiness with stylistic features, such as sentence length variability or syntactic complexity, might improve classification accuracy even further. This lends particularly well to such applications as academic integrity where plagiarism detection is key, or journalism and media regulation that require the authenticity of published content.

Although its strengths, this study has several limitations. But it’s limited by its reliance on GPT-2 for probability calculations, which may not capture the full potential of more powerful models like GPT-3.5 or GPT-4. This may result in different perplexity, burstiness distributions resulting in different effectiveness of the metrics in these newer models. At the end of the writing process of this essay, a Chinese company Deepseek published an open source model deepseek-v3 that rivals GPT-4o from OpenAI, it is a regret of this paper that deepseek-v3 was not utilised for the experiments and it would improve the results’ credibility (*DeepSeek-AI*, 2025). Additionally, the study cannot perceive nuances in high structured or domain specific datasets lacking stylistic analyses like lexical diversity or sentence structure variability. While the scope of datasets used in this study is diverse, it does not cover all types of humans or AI generated text and future work can investigate more varied datasets, e.g., conversational or multilingual text. This study solely used Random Forest Classifiers, which are very powerful here, but may have missed the intricacies of text patterns that deep learning based methods could have captured. Future research could integrate neural network classifiers used to handle those more complicated relationships between statistical and stylistic features.

Eventually, the development of AI-detection tools can be further expanded by incorporating ensemble methods associated with statistical metrics combined with advanced language model-based tools to better AI detection at more different text formats. In addition, a standardized benchmark for the detection of AI generated text in different datasets, models, and contexts could provide a more consistent and reliable metric for comparison. Also adding support for multilingual datasets would extend the study to take into consideration the use of AI to create content in many languages, thereby increasing the tool’s applicability to a larger set of content. On top of promoting accuracy, establishing ethical principles and transparency for chasing AI content from academia, journalism and social media would also be served by these advancements.

# 7. Conclusion

In conclusion, this study highlights the importance of a multi-metric approach for reliably identifying AI-generated text. Entropy has proven to be a powerful and consistent indicator, particularly in structured datasets, while perplexity and burstiness are most effective in creative or varied contexts. By combining these metrics and accounting for dataset-specific characteristics, future detection tools can achieve higher accuracy and adaptability across diverse applications. This integrated approach provides a strong foundation for addressing the growing challenge of distinguishing between human and AI-generated text in a rapidly evolving technological landscape.

# 

# Bibliography:

Abdali, S. *et al.* (2024) ‘Decoding the AI Pen: Techniques and Challenges in Detecting AI-Generated Text’, in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 6428–6436. Available at: https://doi.org/10.1145/3637528.3671463.

‘AI detection tools: the challenge of today’s digital age. | CONTADU’ (no date). Available at: https://contadu.com/ai-detection-tools-the-challenge-of-todays-digital-age/ (Accessed: 3 July 2024).

Chakraborty, S. *et al.* (2023) ‘On the Possibilities of AI-Generated Text Detection’. arXiv. Available at: https://doi.org/10.48550/arXiv.2304.04736.

Chen, Y. *et al.* (2023) ‘GPT-Sentinel: Distinguishing Human and ChatGPT Generated Content’. arXiv. Available at: https://doi.org/10.48550/arXiv.2305.07969.

Chiusano, F. (2022) ‘Two minutes NLP — Perplexity explained with simple probabilities’, *NLPlanet*, 27 January. Available at: https://medium.com/nlplanet/two-minutes-nlp-perplexity-explained-with-simple-probabilities-6cdc46884584 (Accessed: 5 November 2024).

Davis, W. (2024) *OpenAI won’t watermark ChatGPT text because its users could get caught*, *The Verge*. Available at: https://www.theverge.com/2024/8/4/24213268/openai-chatgpt-text-watermark-cheat-detection-tool (Accessed: 10 January 2025).

*DeepSeek-R1/DeepSeek\_R1.pdf at main · deepseek-ai/DeepSeek-R1* (no date) *GitHub*. Available at: https://github.com/deepseek-ai/DeepSeek-R1/blob/main/DeepSeek\_R1.pdf (Accessed: 26 January 2025).

‘Fano factor’ (2024) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Fano\_factor&oldid=1227911597 (Accessed: 5 November 2024).

Gehrmann, S., Strobelt, H. and Rush, A.M. (2019) ‘GLTR: Statistical Detection and Visualization of Generated Text’. arXiv. Available at: https://doi.org/10.48550/arXiv.1906.04043.

Guo, X. *et al.* (2024) ‘DeTeCtive: Detecting AI-generated Text via Multi-Level Contrastive Learning’. arXiv. Available at: https://doi.org/10.48550/arXiv.2410.20964.

He, X. *et al.* (2024) ‘MGTBench: Benchmarking Machine-Generated Text Detection’. arXiv. Available at: https://doi.org/10.48550/arXiv.2303.14822.

Kolev, V. (2023) *Perplexity and Burstiness in AI and Human Writing: Two Important Concepts*, *AILC Portal*. Available at: https://www.unic.ac.cy/telblog/2023/04/11/perplexity-and-burstiness-in-ai-and-human-writing-two-important-concepts/ (Accessed: 5 November 2024).

Louppe, G. (2015) ‘Understanding Random Forests: From Theory to Practice’. arXiv. Available at: https://doi.org/10.48550/arXiv.1407.7502.

‘Measuring Performance: AUC (AUROC)’ (2019) *Glass Box*, 23 February. Available at: https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/ (Accessed: 6 November 2024).

Mitchell, E. *et al.* (2023) ‘DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature’. arXiv. Available at: https://doi.org/10.48550/arXiv.2301.11305.

Mitra, A. *et al.* (2023) ‘Orca 2: Teaching Small Language Models How to Reason’. arXiv. Available at: https://doi.org/10.48550/arXiv.2311.11045.

Moradi, H., Grzymala-Busse, J.W. and Roberts, J.A. (1998) ‘Entropy of English text: Experiments with humans and a machine learning system based on rough sets’, *Information Sciences*, 104(1–2), pp. 31–47. Available at: https://doi.org/10.1016/S0020-0255(97)00074-1.

Mukherjee, S. *et al.* (2023) ‘Orca: Progressive Learning from Complex Explanation Traces of GPT-4’. arXiv. Available at: https://doi.org/10.48550/arXiv.2306.02707.

*Multilayer Perceptrons in Machine Learning: A Comprehensive Guide | DataCamp* (no date). Available at: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning (Accessed: 12 September 2024).

‘Neural network (machine learning)’ (2024) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Neural\_network\_(machine\_learning)&oldid=1244900498 (Accessed: 12 September 2024).

*Open-Orca/OpenOrca · Datasets at Hugging Face* (2023). Available at: https://huggingface.co/datasets/Open-Orca/OpenOrca (Accessed: 5 November 2024).

*Paper Agent* (no date). Available at: https://pasa-agent.ai/home?query=detecting+ai+generated+text&session=1738448265437545536 (Accessed: 1 February 2025).

Sasse, K. *et al.* (2024) ‘To Burst or Not to Burst: Generating and Quantifying Improbable Text’. arXiv. Available at: http://arxiv.org/abs/2401.15476 (Accessed: 6 November 2024).

Sennrich, R., Haddow, B. and Birch, A. (2016) ‘Neural Machine Translation of Rare Words with Subword Units’. arXiv. Available at: https://doi.org/10.48550/arXiv.1508.07909.

Su, J. *et al.* (2023) ‘DetectLLM: Leveraging Log Rank Information for Zero-Shot Detection of Machine-Generated Text’. arXiv. Available at: https://doi.org/10.48550/arXiv.2306.05540.

Tulchinskii, E. *et al.* (2023) ‘Intrinsic Dimension Estimation for Robust Detection of AI-Generated Texts’. arXiv. Available at: https://doi.org/10.48550/arXiv.2306.04723.

Verma, V. *et al.* (2024) ‘Ghostbuster: Detecting Text Ghostwritten by Large Language Models’. arXiv. Available at: https://doi.org/10.48550/arXiv.2305.15047.

*What is LLM? - Large Language Models Explained - AWS* (no date) *Amazon Web Services, Inc.* Available at: https://aws.amazon.com/what-is/large-language-model/ (Accessed: 5 November 2024).

*What is Perceptron? A Beginners Guide for 2024* (2021) *Simplilearn.com*. Available at: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron (Accessed: 12 September 2024).

*What is perplexity? | Continuum Labs* (2024). Available at: https://training.continuumlabs.ai/data/datasets/what-is-perplexity (Accessed: 5 November 2024).

*What is Tokenization? Types, Use Cases, Implementation* (no date). Available at: https://www.datacamp.com/blog/what-is-tokenization (Accessed: 3 July 2024).

Yu, X. *et al.* (2024) ‘LLM Paternity Test: Generated Text Detection with LLM Genetic Inheritance’. arXiv. Available at: http://arxiv.org/abs/2305.12519 (Accessed: 7 November 2024).

# Appendix A – Hyperparameter of the classifiers:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Xsum | | | | | SQuAD | | | WritingPrompts | | |
| hyperparameter | | MD | MSL | MSS | MD | MSL | MSS | MD | | MSL | MSS |
| Perplexity & Burstiness | | 10 | 4 | 2 | 10 | 4 | 10 | 10 | | 4 | 2 |
| LRR | | 10 | 1 | 10 | 10 | 2 | 10 | 10 | | 2 | 10 |
| extropy | | 10 | 4 | 10 | 10 | 1 | 10 | 10 | | 4 | 10 |

*Table 4.* Fine-tuned hyperparameters for the random forest classifiers. MD representing max depth, meaning the max level that the tree can grow up to, MSL representing minimum samples leaf, meaning the minimum number of samples in a leaf, and MSS representing minimum samples split, meaning the minimum sample that a node must have to split.

# 

# Appendix B – program:

## Data preprocessing - Xsum:

import os

import pandas as pd

import re

import torch

from transformers import GPT2LMHeadModel, GPT2Tokenizer

from tqdm import tqdm

# Define paths

ai\_file\_path = '3\_5M-GPT3\_5-Augmented.parquet'

human\_folder\_path = 'bbc-summary-data'

output\_combined\_file = 'combined\_ai\_human\_texts.csv'

# Load GPT-2 model and tokenizer for content processing

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = GPT2LMHeadModel.from\_pretrained('gpt2').to(device)

tokenizer = GPT2Tokenizer.from\_pretrained('gpt2')

model.eval()

# Helper function to extract content from human-generated .summary files

def extract\_content(file\_path):

try:

with open(file\_path, 'r', encoding='utf-8') as file:

content = file.read()

except UnicodeDecodeError:

print(f"Skipping {file\_path} due to encoding error.")

return None

# Extract the [SN]FIRST-SENTENCE[SN] and [SN]RESTBODY[SN] sections

first\_sentence\_start = content.find("[SN]FIRST-SENTENCE[SN]") + len("[SN]FIRST-SENTENCE[SN]")

first\_sentence\_end = content.find("[SN]RESTBODY[SN]")

first\_sentence = content[first\_sentence\_start:first\_sentence\_end].strip()

rest\_body\_start = content.find("[SN]RESTBODY[SN]") + len("[SN]RESTBODY[SN]")

rest\_body = content[rest\_body\_start:].strip()

return first\_sentence + " " + rest\_body

# Load AI-generated data

print("Loading AI-generated text from Parquet file...")

df\_ai = pd.read\_parquet(ai\_file\_path)

# Select rows from the 10,000th to the 15,000th entry (index 9999 to 14999)

ai\_texts = df\_ai['response'].iloc[:5000].tolist()

print(f"Extracted {len(ai\_texts)} AI-generated responses.")

# Label AI-generated text

ai\_data = [{'text': text, 'label': 1} for text in ai\_texts]

# Extract and label human-generated data

print("Extracting human-generated text from .summary files...")

human\_files = [f for f in os.listdir(human\_folder\_path) if f.endswith('.summary')]

human\_data = []

for file\_name in tqdm(human\_files[:5000], desc="Processing human-generated files"):

file\_path = os.path.join(human\_folder\_path, file\_name)

content = extract\_content(file\_path)

if content:

human\_data.append({'text': content, 'label': 0})

print(f"Extracted {len(human\_data)} human-generated summaries.")

# Combine AI and human data

combined\_data = ai\_data + human\_data

df\_combined = pd.DataFrame(combined\_data)

# Save combined data to CSV

print(f"Saving combined data to {output\_combined\_file}...")

df\_combined.to\_csv(output\_combined\_file, index=False)

print("Combined dataset saved successfully.")

## Data preprocessing - SQuARD:

import pandas as pd

# Paths for the files

first\_file\_path = 'train-00000-of-00001.parquet'

second\_file\_path = '3\_5M-GPT3\_5-Augmented.parquet'

output\_file\_path = 'output.csv'

# Initialize variables

rows\_to\_extract\_first = 5000

rows\_to\_extract\_second = 5000

step = 5

text\_column\_name = "text"

label\_column\_name = "label"

# Part 1: Process the first file

data\_first = pd.read\_parquet(first\_file\_path)

output\_data = pd.DataFrame(columns=[text\_column\_name, label\_column\_name])

# Extract every 5th row up to 5000 rows in total, taking only the column with index 2

for i in range(0, min(len(data\_first), rows\_to\_extract\_first \* step), step):

context = data\_first.iloc[i, 2] # Extract the column with index 2

row = pd.DataFrame({text\_column\_name: [context], label\_column\_name: [0]})

output\_data = pd.concat([output\_data, row], ignore\_index=True)

if len(output\_data) >= rows\_to\_extract\_first:

break

# Part 2: Process the second file

data\_second = pd.read\_parquet(second\_file\_path)

# Extract rows from 5000 to 10000 from the 'response' column

for i in range(5000, min(10000, len(data\_second))):

response = data\_second.iloc[i]["response"] # Replace "response" with the actual column name if different

row = pd.DataFrame({text\_column\_name: [response], label\_column\_name: [1]})

output\_data = pd.concat([output\_data, row], ignore\_index=True)

# Save combined data to CSV

output\_data.to\_csv(output\_file\_path, index=False)

print(f"CSV file with combined data saved to {output\_file\_path}")

## Data preprocessing - WritingPrompts:

import pandas as pd

# Paths for the files

first\_file\_path = 'writingprompts.parquet'

second\_file\_path = '3\_5M-GPT3\_5-Augmented.parquet'

output\_file\_path = 'writingprompts.csv'

# Initialize variables

rows\_to\_extract\_first = 5000

rows\_to\_extract\_second = 5000

step = 5

text\_column\_name = "text"

label\_column\_name = "label"

# Part 1: Process the new first file

data\_first = pd.read\_parquet(first\_file\_path)

output\_data = pd.DataFrame(columns=[text\_column\_name, label\_column\_name])

# Extract consecutive rows up to 5000 rows in total, taking only the column with index 1

for i in range(0, min(len(data\_first), rows\_to\_extract\_first)):

story\_content = data\_first.iloc[i, 1] # Extract content from column index 1

row = pd.DataFrame({text\_column\_name: [story\_content], label\_column\_name: [0]})

output\_data = pd.concat([output\_data, row], ignore\_index=True)

if len(output\_data) >= rows\_to\_extract\_first:

break

# Part 2: Process the second file

data\_second = pd.read\_parquet(second\_file\_path)

# Extract rows from 5000 to 10000 from the 'response' column

for i in range(10000, min(15000, len(data\_second))):

response = data\_second.iloc[i]["response"] # Replace "response" with the actual column name if different

row = pd.DataFrame({text\_column\_name: [response], label\_column\_name: [1]})

output\_data = pd.concat([output\_data, row], ignore\_index=True)

# Save combined data to CSV

output\_data.to\_csv(output\_file\_path, index=False)

print(f"CSV file with combined data saved to {output\_file\_path}")

## Displaying perplexity graph:

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score, roc\_auc\_score, roc\_curve

import matplotlib.pyplot as plt

import os

# Set up directories and file paths

result\_file\_path = 'classification\_results/classification\_results\_tuned.txt'

os.makedirs('roc\_curves', exist\_ok=True)

# Load datasets

file\_paths = {

'combined\_ai\_human\_texts': 'Xsum&AIGT.csv',

'squard': 'SQuAD&AIGT.csv',

'writingprompts': 'WritingPrompts&AIGT.csv'

}

# Function to train and evaluate using only the perplexity feature

def train\_and\_evaluate\_perplexity\_classifier(df, target='label'):

# Split data: first 10% and last 10% for testing, middle 80% for training

num\_rows = len(df)

test\_indices = list(range(int(num\_rows \* 0.1))) + list(range(int(num\_rows \* 0.9), num\_rows))

train\_indices = list(range(int(num\_rows \* 0.1), int(num\_rows \* 0.9)))

df\_train = df.iloc[train\_indices]

df\_test = df.iloc[test\_indices]

# Prepare training and test data with only the perplexity feature

X\_train = df\_train[['perplexity']].values

y\_train = df\_train[target].values

X\_test = df\_test[['perplexity']].values

y\_test = df\_test[target].values

# Train Random Forest Classifier with default parameters

rf\_classifier = RandomForestClassifier(n\_estimators=100, max\_depth=10, random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

# Predict and evaluate

y\_pred = rf\_classifier.predict(X\_test)

y\_prob = rf\_classifier.predict\_proba(X\_test)[:, 1]

# Calculate metrics

accuracy = accuracy\_score(y\_test, y\_pred)

auc\_score = roc\_auc\_score(y\_test, y\_prob)

report = classification\_report(y\_test, y\_pred)

# Calculate ROC curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

return accuracy, auc\_score, report, fpr, tpr

# Train and evaluate the classifier on each dataset and store results

with open(result\_file\_path, 'a') as result\_file: # Open in append mode to add results

for dataset\_name, file\_path in file\_paths.items():

result\_file.write(f"Dataset: {dataset\_name} (Perplexity Only)\n")

df = pd.read\_csv(file\_path)

# Filter out any non-numeric or NaN data in perplexity column

df['perplexity'] = pd.to\_numeric(df['perplexity'], errors='coerce')

df = df.dropna(subset=['perplexity'])

# Train and evaluate

accuracy, auc\_score, report, fpr, tpr = train\_and\_evaluate\_perplexity\_classifier(df)

# Write results to text file

result\_file.write(f"Classifier based on perplexity:\n")

result\_file.write(f"Accuracy: {accuracy:.2f}\n")

result\_file.write(f"AUC Score: {auc\_score:.2f}\n")

result\_file.write("Classification Report:\n")

result\_file.write(report)

result\_file.write("\n" + "-"\*50 + "\n")

# Plot and save ROC curve

plt.figure()

plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {auc\_score:.2f})')

plt.plot([0, 1], [0, 1], color='grey', linestyle='--') # Diagonal line for random guessing

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(f'ROC Curve for {dataset\_name} - Perplexity Only')

plt.legend(loc="lower right")

plt.savefig(f'roc\_curves/{dataset\_name}\_perplexity\_only\_roc\_curve.png')

plt.close()

# Save FPR and TPR values to a CSV file

roc\_data = pd.DataFrame({'FPR': fpr, 'TPR': tpr})

roc\_data.to\_csv(f'roc\_curves/{dataset\_name}\_perplexity\_only\_roc\_data.csv', index=False)

result\_file.write("\n" + "="\*100 + "\n\n")

print("Perplexity-only classifier training and evaluation complete. Results appended to 'classification\_results\_tuned.txt'.")

## Calculation of metrics:

import pandas as pd

import numpy as np

import re

import torch

import matplotlib.pyplot as plt

from transformers import GPT2LMHeadModel, GPT2Tokenizer

import seaborn as sns

from tqdm import tqdm

# Initialize GPT-2 model and tokenizer for calculating probabilities and ranks

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

gpt\_model = GPT2LMHeadModel.from\_pretrained("gpt2").to(device)

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

gpt\_model.eval()

# Replace NaN in text data or skip entries with NaN values

def preprocess\_text(df):

df['text'] = df['text'].fillna('') # Replace NaN with empty strings

# Function to calculate token log-likelihoods and ranks

def calculate\_log\_likelihood\_and\_rank(text):

if not text.strip(): # Skip if text is empty

return np.nan

inputs = tokenizer(text, return\_tensors="pt", truncation=True, max\_length=1024).to(device)

with torch.no\_grad():

outputs = gpt\_model(\*\*inputs)

logits = outputs.logits[0]

log\_probs = torch.log\_softmax(logits, dim=-1)

log\_likelihood = 0

log\_rank = 0

tokens = inputs["input\_ids"][0]

for i, token\_id in enumerate(tokens[1:], 1):

log\_likelihood += log\_probs[i-1, token\_id].item()

rank = (log\_probs[i-1] > log\_probs[i-1, token\_id]).sum().item() + 1

log\_rank += np.log(rank) if rank > 0 else 0

avg\_log\_likelihood = log\_likelihood / max(len(tokens[1:]), 1)

avg\_log\_rank = log\_rank / max(len(tokens[1:]), 1)

lrr = abs(avg\_log\_likelihood / avg\_log\_rank) if avg\_log\_rank != 0 else np.nan

return lrr

# Other metric functions (with NaN handling)

def calculate\_perplexity(text):

if not text.strip(): # Skip if text is empty

return np.nan

inputs = tokenizer(text, return\_tensors="pt", truncation=True, max\_length=1024).to(device)

with torch.no\_grad():

outputs = gpt\_model(\*\*inputs, labels=inputs["input\_ids"])

loss = outputs.loss

perplexity = torch.exp(loss)

return perplexity.item()

def calculate\_burstiness(text):

if not text.strip(): # Skip if text is empty

return np.nan

segments = re.split(r'(?<=[.!?]) +', text)

segment\_perplexities = [calculate\_perplexity(segment) for segment in segments if segment.strip()]

if len(segment\_perplexities) == 0:

return np.nan

mean\_perplexity = np.mean(segment\_perplexities)

std\_perplexity = np.std(segment\_perplexities)

burstiness = std\_perplexity / mean\_perplexity if mean\_perplexity != 0 else np.nan

return burstiness

def calculate\_entropy(text):

if not text.strip(): # Skip if text is empty

return np.nan

tokenized\_text = tokenizer.encode(text, add\_special\_tokens=False)

if len(tokenized\_text) == 0:

return np.nan

token\_counts = np.bincount(tokenized\_text)

probabilities = token\_counts / np.sum(token\_counts)

entropy = -np.sum(probabilities \* np.log2(probabilities + 1e-9))

return entropy

# Load files

file\_paths = {

'squard': 'squard.csv',

'combined\_ai\_human\_texts': 'combined\_ai\_human\_texts.csv',

'writingprompts': 'writingprompts.csv'

}

# Calculate metrics and plot distributions

for file\_name, path in file\_paths.items():

print(f"Processing file: {file\_name}")

df = pd.read\_csv(path)

preprocess\_text(df)

# Calculate metrics with progress monitoring

print("Calculating Perplexity...")

df['perplexity'] = [calculate\_perplexity(text) for text in tqdm(df['text'], desc="Perplexity")]

print("Calculating Burstiness...")

df['burstiness'] = [calculate\_burstiness(text) for text in tqdm(df['text'], desc="Burstiness")]

print("Calculating LRR...")

df['LRR'] = [calculate\_log\_likelihood\_and\_rank(text) for text in tqdm(df['text'], desc="LRR")]

print("Calculating Entropy...")

df['entropy'] = [calculate\_entropy(text) for text in tqdm(df['text'], desc="Entropy")]

# Save calculated data to a CSV file

output\_data\_file = f"{file\_name}\_metrics.csv"

df.to\_csv(output\_data\_file, index=False)

print(f"Saved calculated data to {output\_data\_file}")

# Plot distributions for each metric

for metric in ['perplexity', 'burstiness', 'LRR', 'entropy']:

plt.figure(figsize=(10, 6))

sns.histplot(data=df, x=metric, hue='label', kde=True, palette={0: 'red', 1: 'blue'}, bins=30)

plt.title(f'{metric.capitalize()} Distribution - {file\_name}')

plt.xlabel(metric.capitalize())

plt.ylabel('Density')

plt.legend(['Human (0)', 'Machine (1)'])

output\_image\_path = f'{file\_name}\_{metric}\_distribution.png'

plt.savefig(output\_image\_path)

plt.close()

print(f"Saved plot: {output\_image\_path}")

print("All plots and data files generated and saved.")

## fine-tuning random forest:

import pandas as pd

import numpy as np

import re

import torch

from transformers import GPT2LMHeadModel, GPT2Tokenizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score, roc\_curve, auc

import matplotlib.pyplot as plt

from tqdm import tqdm

# Load GPT-2 model and tokenizer for calculating perplexity

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

gpt\_model = GPT2LMHeadModel.from\_pretrained("gpt2").to(device)

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

gpt\_model.eval()

# Function to calculate perplexity for a single text

def calculate\_perplexity(text):

inputs = tokenizer(text, return\_tensors="pt", truncation=True, max\_length=1024).to(device)

with torch.no\_grad():

outputs = gpt\_model(\*\*inputs, labels=inputs["input\_ids"])

loss = outputs.loss

perplexity = torch.exp(loss)

return perplexity.item()

# Function to calculate burstiness based on sentence-level perplexities

def calculate\_burstiness(text):

segments = re.split(r'(?<=[.!?]) +', text)

segment\_perplexities = [calculate\_perplexity(segment) for segment in segments if segment.strip()]

mean\_perplexity = np.mean(segment\_perplexities)

std\_perplexity = np.std(segment\_perplexities)

burstiness = std\_perplexity / mean\_perplexity if mean\_perplexity != 0 else 0

return mean\_perplexity, burstiness

# Load and preprocess the data

file\_path = 'combined\_ai\_human\_texts.csv'

df = pd.read\_csv(file\_path)

df['text'] = df['text'].fillna('') # Handle any NaN text values

# Calculate mean\_perplexity and burstiness for each text

print("Calculating perplexity and burstiness for all data...")

df["mean\_perplexity"], df["burstiness"] = zip(\*tqdm(df["text"].apply(calculate\_burstiness), total=len(df), desc="Perplexity and Burstiness"))

# Fill any NaN values in the calculated columns

df["mean\_perplexity"].fillna(df["mean\_perplexity"].mean(), inplace=True)

df["burstiness"].fillna(df["burstiness"].mean(), inplace=True)

# Split data: First 10% and last 10% for testing, middle 80% for training

num\_rows = len(df)

test\_indices = list(range(int(num\_rows \* 0.1))) + list(range(int(num\_rows \* 0.9), num\_rows))

train\_indices = list(range(int(num\_rows \* 0.1), int(num\_rows \* 0.9)))

df\_train = df.iloc[train\_indices]

df\_test = df.iloc[test\_indices]

# Prepare training and test data

X\_train = df\_train[['mean\_perplexity', 'burstiness']].values

y\_train = df\_train['label'].values

X\_test = df\_test[['mean\_perplexity', 'burstiness']].values

y\_test = df\_test['label'].values

# Train the Random Forest Classifier

rf\_classifier = RandomForestClassifier(

n\_estimators=100,

max\_depth=10,

min\_samples\_split=10,

min\_samples\_leaf=1,

random\_state=42

)

rf\_classifier.fit(X\_train, y\_train)

# Evaluate the model on the test data

y\_pred = rf\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

# Print classification report and accuracy

print("Classification Report on Test Data:")

print(classification\_report(y\_test, y\_pred))

print(f"Accuracy on Test Data: {accuracy:.2f}")

# Calculate and plot ROC AUC

y\_prob = rf\_classifier.predict\_proba(X\_test)[:, 1]

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

# Save accuracy and ROC AUC data to a text file

output\_txt\_file = 'model\_evaluation.txt'

with open(output\_txt\_file, 'w') as f:

f.write(f"Accuracy on Test Data: {accuracy:.2f}\n")

f.write(f"AUC: {roc\_auc:.2f}\n")

f.write("False Positive Rate (FPR), True Positive Rate (TPR):\n")

for fp, tp in zip(fpr, tpr):

f.write(f"{fp}, {tp}\n")

print(f"Results saved to {output\_txt\_file}")

# Appendix C – GPT-2 tokenizer performance:

The research from (Su *et al.*, 2023), has compared six models on the three datasets used in this paper against AIGT from T5-small and T5-base, and the results have revealed the leading position of GPT-2. The data presented in AUROC diagram is shown below:

A chart of different colored lines

Description automatically generated

The second set of models is against AIGT from T5-3b and T5-large, the results are also presented:

A chart of different colored lines

Description automatically generated

From the two sets, we can observe that the AUROC score for the GPT-2 model is generally higher and smoother than other models, representing it is well-balanced and can be well presented in diverse datasets.

# Appendix D – AUROC score visual representation:

The result presented in the AUROC diagram evaluates the accuracy of the approach. The x-axis of the diagram represents the false positive rate, which refers to the rate at which the model predicts the wrong answer and the y-axis represents the true positive rate, representing the rate at which the model predicts the right answer. The closer the cure is to the top left corner of the diagram, the better the performance of a model.

|  |  |  |  |
| --- | --- | --- | --- |
| Method\Dataset | XSum | SQuAD | WritingPrompts |
| Perplexity & Burstiness | A graph of a curve  Description automatically generated | A graph of a curve  Description automatically generated | *A graph of a curve  Description automatically generated* |
| LRR | A graph of a curve  Description automatically generated | A graph of a curve  Description automatically generated | *A graph of a curve  Description automatically generated* |
| extropy | A graph of a curve  Description automatically generated | *A graph of a curve  Description automatically generated* | *A graph of a curve  Description automatically generated* |

*Table 5.* AUROC scores visually resented for detecting samples from the given model on the given dataset for Perplexity&Burstiness and two previously proposed criteria.

# 

# Appendix E – comparing different classifiers performance

A graph showing different colored lines

Description automatically generated

*Figure 7.* demonstrates the improvement in AUROC with respect to sequence length using

various real detectors/classifiers. The orange line showing the variation of random forest is smoother than the other two classifiers. It is also shown that the random forest performs better when the numbers of sequences becomes large. We choose to use random forest since we have enough data.