OPEN ACCESS | https://doi.org/10.25080/DHKD1726

AI-Driven Watermarking Technique for Safeguarding Text Integrity in the Digital Age

Atharva Rasane[1]

[1] KLE Technology University

SciPy 2024

July 8 - July 14, 2024

Proceedings of the 23rd Python in Science Conference
ISSN: 2575-9752

Published Jul 10, 2024

Correspondence to
Atharva Rasane
rratharva@gmail.com

Abstract

The internet's growth has led to a surge in text usage. Now, with public access to generative AI models like ChatGPT/Bard, identifying the source is vital. This is crucial due to concerns about copyright infringement and plagiarism. Moreover, it is essential to differentiate AI-generated text to curb misinformation from AI model hallucinations.

In this paper, we explore text watermarking as a potential solution, focusing on plain ASCII text in English. We investigate techniques including physical watermarking (e.g., UniSpaCh by Por et al.), which modifies text to hide a binary message using Unicode Spaces, and logical watermarking (e.g., word context by Jalil et al.), which generates a watermark key via a defined process. While logical watermarking is difficult to break but undetectable without prior knowledge, physical watermarks are easily detected but also easy to break.

This paper presents a unique physical watermarking technique based on word substitution to address these challenges. The core idea is that AI models consistently produce the same output for the same input. Initially, we replaced every i-th word (for example, every 5th word) with a "[MASK]," a placeholder token used in natural language processing models to indicate where a word has been removed and needs to be predicted. Then, we used a BERT model to predict the most probable token in place of "[MASK]." The resulting text constitutes the watermarked text. To verify, we reran the algorithm on the watermarked text and compared the input and output for similarity.

The Python implementation of the algorithm in this paper employs models from the HuggingFace Transformer Library, namely "bert-base-uncased" and "distilroberta-base". The "[MASK]" placeholder was generated by splitting the input string using the split() function and then replacing every 5th element in the list with "[MASK]". This modified list served as the input text for the BERT model, where the output corresponding to each "[MASK]" was replaced accordingly. Finally, applying the join() function to the list produces the watermarked text.

This technique tends to generate nearly invisible watermarked text, preserving its integrity or completely changing the meaning of the text based on how similar the text is to the training dataset of BERT. This was observed when the algorithm was run on the story of Red Riding Hood, where its meaning was altered. However, the nature of this watermark makes it extremely difficult to break due to the black-box nature of the AI model.

Keywords physical watermark, logical watermark, HuggingFace Transformer Library, BERT

1. Introduction

The growth of the internet is driven by the spread of web pages, which are written in HTML (Hyper Text Markup Language). These web pages contain large amounts of text. Almost every webpage, in some form or another, contains text, making it a popular mode of communication, whether it be blogs, posts, articles, comments, etc. Text can be represented as a collection of ASCII or Unicode values, where each value corresponds to a specific character. Given the text-focused nature of the internet and tools like ChatGPT or Bard, it is crucial to identify the source of text. This helps to manage copyright issues and distinguish between AI-generated and human-written text, thereby preventing the spread of misinformation. Currently, detecting AI-generated text relies on machine learning classifiers that need frequent retraining with the latest AI-generated data. However, this method has drawbacks, such as the rapid evolution of AI models producing increasingly human-like text. Therefore, a more stable approach is needed, one that does not depend on the specific AI model generating the text.

Watermarks are an identifying pattern used to trace the origin of the data. In this case, we specifically want to focus on text watermarking (watermarking of plain text). Text watermarking can broadly be classified into 2 types, Logical Embedding, and Physical Embedding, which in turn can be classified further [1]. Logical Embedding involves the user generating a watermark key by some logic from the input text. Note that this means that the input text is not altered, and the user instead keeps the generated watermark key to identify the text. Physical Embedding involves the user altering the input text itself to insert a message into it, and the user instead runs an algorithm to find this message to identify the text. In this paper, we will propose an algorithm to watermark text using BERT (Bidirectional Encoder Representations from Transformers), a model introduced by Google, whose main purpose is to replace a special symbol “[MASK]” with the most probable word given the context.

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained model introduced by Google in 2018, which has revolutionized natural language processing (NLP) [2]. “Pre-trained” means the model has already been trained on a large dataset before being fine-tuned for specific tasks. This allows the model to learn general features and patterns from a broad range of text data. For BERT, this pre-training involves vast amounts of text from books, articles, and websites, enabling it to understand the intricacies of human language. This pre-training allows BERT to be adapted quickly to various NLP tasks with relatively small amounts of task-specific data. Traditional models read text sequentially, either left-to-right or right-to-left. In contrast, BERT reads text in both directions simultaneously, providing a deeper understanding of context and meaning. This bidirectional approach allows BERT to perform exceptionally well in various NLP tasks, including question answering, text classification, and named entity recognition. By grasping the nuances of language more effectively, BERT sets a new standard for accuracy and efficiency in NLP applications [2].

At its core, BERT employs a bi-directional Transformer encoder, which helps understand the relationships between words in a sentence. This enhances its comprehension of text by understanding context from both directions simultaneously. BERT undergoes pre-training through two tasks: Masked Language Modeling (MLM), where certain words in a sentence are masked and the model predicts them based on surrounding words, and Next Sentence Prediction (NSP), which involves determining if one sentence logically follows another. This comprehensive training enables BERT to excel in numerous NLP applications like question answering, text classification, and named entity recognition. Given its deep understanding of context and semantics, BERT is highly relevant to text watermarking. Watermarking text involves embedding identifying patterns within the text to trace its origin, which can be critical for copyright protection and distinguishing between AI-generated and human-written content. BERT’s sophisticated handling of language makes it ideal for embedding watermarks in a way that is subtle yet robust, ensuring that the text remains natural while the watermark is detectable. This capability provides a more stable and reliable method for watermarking text, irrespective of the model generating the text, therefore offering a concrete solution amidst the evolving landscape of AI-generated content.

2. Related Work

In this section, we will review two text watermarking algorithms before introducing our proposed technique. Let's first look at the current standards for text watermarking. Text watermarking algorithms embed unique identifiers in text to protect copyright and verify authenticity. They are important because they help prevent unauthorized use, copying, and distribution of text.

The first algorithm is Word Context, developed by Z. Jalil and A. M. Mirza [3]. It is a type of logical watermarking that generates a watermark key without altering the original text [3]. Logical watermarking involves embedding a watermark key without changing the original text. Word Context generates a watermark key by analyzing the structure of the text around selected keywords and creating a pattern based on word lengths [3]. In Word Context, a keyword is selected. For example, using the keyword `'is'` in the text `'Pakistan is a developing country, with Islamabad is the capital of Pakistan. It is located in Asia.'` The lengths of the words before and after `'is'` are recorded: `'Pakistan'` (8) and `'a'` (1), `'Islamabad'` (9) and `'the'` (3), `'It'` (2) and `'located'` (7). The watermark is then 8-1-9-3-2-7 [3]. The keyword is chosen based on its significance in the text. Word lengths are used to create the watermark because they provide a unique pattern without altering the text, ensuring the watermark is imperceptible [3].

The second algorithm, UniSpaCh by Kamaruddin et al. in 2018, modifies the white spaces in text to embed a binary message directly into it [4]. Modifying white spaces changes the spacing patterns in the text, embedding binary information. A binary message is a sequence of bits (0s and 1s) that represents data. This method uses different types of spaces to encode these bits [4]. UniSpaCh uses 2-bit categorization to create a binary string (e.g., `'10'`, `'01'`, `'00'`, `'11'`). Each pair of bits is replaced with a unique type of space (like punctuation space, thin space). These spaces are then placed in areas like between words, sentences, and paragraphs. This method is highly invisible but has low capacity, making it unsuitable for embedding long messages [4]. 2-bit categorization assigns pairs of bits to specific types of spaces. This method is considered invisible because the changes are subtle and not easily noticeable by readers. It has low capacity because only a few bits can be embedded per space, limiting the amount of information that can be hidden [4].

The first approach by Z. Jalil and A. M. Mirza [3] is not suitable for today's fast-paced generation of AI text, as it is impractical to store a logical watermark for each new text [3]. It is impractical to store a logical watermark for each text because the volume of generated text is too high, making it difficult to manage and store all watermarks. AI text generation has made it easier and faster to produce large amounts of text, increasing the need for scalable watermarking solutions [3]. The second approach by Por et al. (2012) is also not suitable because the watermark can be easily removed by reformatting the text. We need a robust and imperceptible watermarking technique [4]. The watermark can be removed by reformatting because changes in text layout, such as altering spaces or reformatting paragraphs, can disrupt the embedded watermark. A robust watermarking technique can withstand such changes and remain detectable, while an imperceptible technique ensures the watermark is not noticeable to the reader [4].

Our proposed technique is based on a method by Lancaster (2023) for ChatGPT [5]. It replaces every fifth word in a sequence of five consecutive words (non-overlapping 5-gram) with a word generated using a fixed random seed. For example, in the sentence `'The friendly robot greeted the visitors with a cheerful beep and a wave of its metal arms,'` the non-overlapping 5-grams are `'The friendly robot greeted the,'` `'visitors with a cheerful beep,'` and `'and a wave of its metal.'` We replace the words `'the,'` `'visitors,'` and `'metal'` with words generated by ChatGPT using a fixed random seed [5]. A non-overlapping 5-gram is a sequence of five consecutive words without any overlap. Replacing every fifth word embeds the watermark without altering the overall meaning of the text, making it a subtle and effective method for embedding the watermark [5].

We check the watermark using overlapping 5-grams, which overlap by four words. For example, `'The friendly robot greeted the,'` `'friendly robot greeted the visitors,'` `'robot greeted the visitors with,'` etc. This method uses ChatGPT to watermark its own text, but it requires running two ChatGPT models to ensure consistency across different outputs from the same seed. Overlapping 5-grams are sequences of five words that overlap by four words. Two models of ChatGPT are needed to ensure consistent watermarking across different outputs because different models might produce different results with the same random seed, and consistency is crucial for verifying the watermark.

We propose using BERT, a model designed to find missing words, as a better alternative to ChatGPT. BERT is more precise and smaller. Its bidirectional nature uses more context for word prediction, potentially leading to better results. While ChatGPT-based algorithms are best for ChatGPT text, BERT can be used for any text, regardless of its origin. BERT is better than ChatGPT for this purpose because it is more precise and smaller, making it more efficient. BERT's bidirectional nature means it uses context from both the preceding and following words to predict a missing word, which can lead to more accurate results.

3. Proposed Model

"BERT-based watermarking is based on the 5-gram approach by Lancaster[5]. However, our focus is on watermarking any text, regardless of its origin. This paper will use **bert-base-uncased** model, which finds the most probable uncased English word to replace the [MASK] token.

Note that a different variant of BERT can be trained on different language datasets and thus will generate a different result and as such the unique identity to consider here is the BERT model i.e. if the user wants a unique watermark they need to train/develop the BERT model on their own. This paper is not concerned with the type of BERT model and is focused on its conceptual application for watermarking. Thus for us, BERT is a black box model that returns the most probable word given the context with the only condition being that it has a constant temperature i.e. it does not hallucinate (produce different results for the same input). For our purposes, you can think of the proposed algorithm as a many to one function which is responsible for converting the input text into a subset of watermarked set.

4. Algorithm

Watermark Encoding

The above is a simple implementation of the algorithm where we are assuming

1. The only white spaces in the text are `" "`.
2. BERT model has infinite context.

This simplified code allows us to grasp the core of the algorithm. First, we split the input text into a list of words using the split() function. Next, we replace every 5th word with the string "[MASK]" which represents a special token indicating where BERT should predict a word. For each [MASK] token, we pass the preceding words and the 4 following words to the BERT model, assuming BERT can handle an infinite context. In reality, BERT has a limited context, so we use up to maximum_context_size - 5 words along with the [MASK] token. The missing_word_form_BERT() function returns the most probable word, which replaces the [MASK] token in the list. We continue this process until all [MASK] tokens are replaced, then convert the list of words back into a string using `" ".join()`.

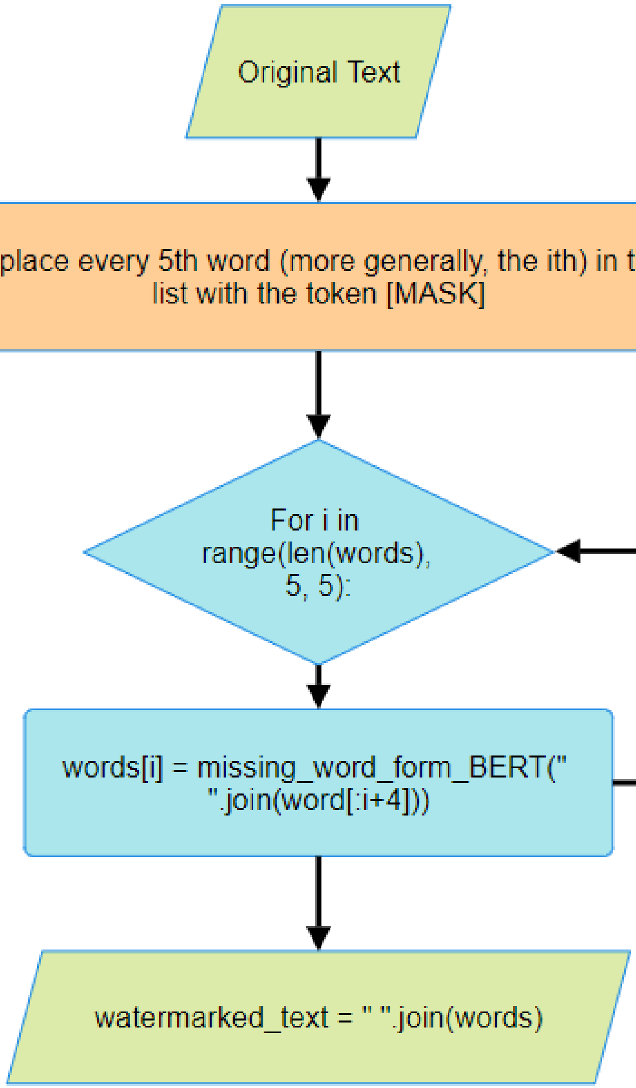
Original Text
Replace every 5th word (more generally, the ith) in the list with the token [MASK]
For i in range(len(words), 5, 5):
words[i] = missing_word_form_BERT(" ".join(word[:i+4]))
watermarked_text = " ".join(words)

Figure 1. *Encoding algorithm to watermark input text*

The beauty of the algorithm is that if we were to run it again on the watermarked text the output that we would get would be the same as the input thus to check if a given text is watermarked we simply need to compare the input and output to determine if a given text is watermarked we simply need to run the above algorithm again, but with a few changes we will have to take in offset as a consideration as the one plagiarizing the text might insert additional words that may lead to the text

Watermark Detection

The algorithm checks if a given text is watermarked by comparing the input and output texts, considering possible word insertions that may offset the watermark pattern.

1. **Input Text Preparation :** Obtain the suspected watermarked text as input.
2. **Run Watermark Detection Algorithm:** Run the watermark detection algorithm on the input text.

3. **Compare Input and Output:** If the input matches the output, the text is watermarked.If not, proceed to check with offsets.
4. **Offset Consideration:** Initialize an array to store match percentages for each offset: `offsets = [0, 1, 2, 3, 4]`.For each offset, adjust the input text by removing `n % 5` words where `n` is the number of words added.
5. **Check for Matches:** For each offset, count the matches where the watermark pattern (every 5th word replaced) aligns.
6. **Store Match Percentages:** Calculate the percentage of matches for each offset and store them.
7. **Statistical Analysis:** Compute the highest percentage of matches (`Highest Ratio`). Compute the average percentage of matches for the remaining offsets (`Average Others`). Calculate the T-Statistic and P-Value to determine the statistical difference between `Highest Ratio` and `Average Others`. The T-Statistic measures the difference between groups, and the P-Value indicates the significance of this difference.
8. **Classification:** Use a pre-trained model to classify the text based on the metrics (`Highest Ratio`, `Average Others`, T-Statistic, P-Value) as watermarked or not.

5. Implementation - Encoding module

Let’s examine a Python implementation of the proposed watermarking model. The watermark_text module identifies every 5th word in the input string, splits them using Python’s built-in split() function, and marks them for modification using BERT. These placeholders are replaced with the [MASK] token. Although we use BERT here, the module can be adapted to other AI models. We chose BERT due to its efficiency in altering individual words. The 5th word is selected to ensure a consistent and detectable pattern. **The choice of index = 5 is because? Also is this code picked from a paper?**

```python
import os
os.environ['HUGGINGFACEHUB_API_TOKEN'] = '<ENTER_HUGGING_FACE_API_KEY>'
from transformers import pipeline, AutoTokenizer, AutoModelForMaskedLM
import torch

def watermark_text(text, model_name="bert-base-uncased", offset=0):
    # Clean and split the input text
    text = " ".join(text.split())
    words = text.split()

    # Replace every fifth word with [MASK], starting from the offset
    for i in range(offset, len(words)):
        if (i + 1 - offset) % 5 == 0:
            words[i] = '[MASK]'

    # Initialize the tokenizer and model, move to GPU if available
    device = 0 if torch.cuda.is_available() else -1
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForMaskedLM.from_pretrained(model_name).to(device)

    # Initialize the fill-mask pipeline
    classifier = pipeline("fill-mask", model=model, tokenizer=tokenizer, device=device)

    # Make a copy of the words list to modify it
    watermarked_words = words.copy()

    # Process the text in chunks
    for i in range(offset, len(words), 5):
        chunk = " ".join(watermarked_words[:i+9])
        if '[MASK]' in chunk:
            try:
                tempd = classifier(chunk)
            except Exception as e:
                print(f"Error processing chunk '{chunk}': {e}")
                continue

            if tempd:
                templ = tempd[0]
                temps = templ['token_str']
                watermarked_words[i+4] = temps.split()[0]

    return " ".join(watermarked_words)

# Example usage
text = "Quantum computing is a rapidly evolving field that leverages the principles of quantum mechanics
to perform computations that are infeasible for classical computers. Unlike classical computers, which
use bits as the fundamental unit of information, quantum computers use quantum bits or qubits. Qubits
can exist in multiple states simultaneously due to the principles of superposition and entanglement,
providing a significant advantage in solving complex computational problems."
watermark_text(text, offset=0)
result = "Quantum computing is a rapidly evolving field that leverages the principles of quantum
mechanics to perform computations that are impossible for classical computers. Unlike quantum computers,
which use bits as the fundamental unit of , quantum computers use quantum bits or qubits. Qubits
can exist in multiple states simultaneously according to the principles of symmetry and entanglement,
providing a significant advantage in solving complex mathematical problems."
```

In the result, the module has replaced each 5th word with the most probable replacement word selected by BERT. There will always be some words that AI would not alter. For example the 10th word “the” and the 15th word “to”. These cannot be changed by AI without altering the entire sentence. Further, to speed up the AI computing, we can employ GPUs in this module as well as the Detection module.

6. Implementation - Detection Module

Now that we have our watermarked text, we need to identify potential copyright infringement. We assume this text is what a plagiarizer has access to.

For this, we create a module to check the number of word matches if the AI model with the same offset parameter is run on the watermarked text again. The algorithm’s elegance lies in its consistency: if we run it again on the watermarked text, the output will match the input because the most probable words are already present at every 5th offset. Consequently, we get a 100% match rate with a match ratio of 1. If all the 5th words were altered, our match rate would be 0.

Altering written text is a posibility we cannot ignore. consider a scenario where a plagiarizer might insert extra words, causing the input not to match the output exactly. This means our model needs to check for watermarks not only at a specific index but also in the surrounding words. Therefore, our model needs to check for the watermark at different offsets (0 to 4) to account for potential word insertions.

Here is how the offset works:

- If 1 word is added at the start, the offset is 1.
- If 2 words are added, the offset is 2.
- If 3 words are added, the offset is 3.
- If 4 words are added, the offset is 4.
- If 5 words are added, the offset is 0 (since the algorithm replaces every 5th word).

In general, if ‘n’ words are added, the offset is `n % 5`. Since we do not know how many words were added, we need to check all possible offsets (0, 1, 2, 3, 4).

If words are added in the middle of the text, the majority of the watermark pattern (every 5th word replaced) will still be detectable at some offset. The idea is that one offset will show a higher number of matches compared to others, indicating a watermark.

For detection, we store the percentage of matches for each offset. There is no fixed threshold for determining a watermark, as the choice of words affects the number of matches. For non-watermarked text, the percentage of matches at each offset will be similar. For watermarked text, one offset will have a significantly higher percentage of matches.

The output of “watermark_text_and_calculate_matches” module is a match ratio for offsets 0-4 acting as a seed for the next stage of detection. Below is the python code for generating the list of match ratios.

```
def watermark_text_and_calculate_matches(text, model_name="bert-base-uncased", max_offset=5):
    # Clean and split the input text
    text = " ".join(text.split())
    words = text.split()

    # Initialize the tokenizer and model, move to GPU if available
    device = 0 if torch.cuda.is_available() else -1
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForMaskedLM.from_pretrained(model_name).to(device)

    # Initialize the fill-mask pipeline
    classifier = pipeline("fill-mask", model=model, tokenizer=tokenizer, device=device)

    # Dictionary to store match ratios for each offset
    match_ratios = {}

    # Loop over each offset
    for offset in range(max_offset):
        # Replace every fifth word with [MASK], starting from the offset
        modified_words = words.copy()
        for i in range(offset, len(modified_words)):
            if (i + 1 - offset) % 5 == 0:
                modified_words[i] = '[MASK]'

        # Make a copy of the modified words list to work on
        watermarked_words = modified_words.copy()
        total_replacements = 0
        total_matches = 0

        # Process the text in chunks
        for i in range(offset, len(modified_words), 5):
            chunk = " ".join(watermarked_words[:i+9])
            if '[MASK]' in chunk:
                try:
                    tempd = classifier(chunk)
                except Exception as e:
                    print(f"Error processing chunk '{chunk}': {e}")
                    continue

                if tempd:
                    templ = tempd[0]
                    temps = templ['token_str']
                    original_word = words[i+4]
                    replaced_word = temps.split()[0]
                    watermarked_words[i+4] = replaced_word

                    # Increment total replacements and matches
                    total_replacements += 1
                    if replaced_word == original_word:
                        total_matches += 1

        # Calculate the match ratio for the current offset
        if total_replacements > 0:
            match_ratio = total_matches / total_replacements
        else:
            match_ratio = 0

        match_ratios[offset] = match_ratio

    # Return the match ratios for each offset
    return match_ratios

# Example usage
text = "Quantum computing is a rapidly evolving field that leverages the principles of quantum mechanics
to perform computations that are infeasible for classical computers. Unlike classical computers, which
use bits as the fundamental unit of information, quantum computers use quantum bits or qubits. Qubits
can exist in multiple states simultaneously due to the principles of superposition and entanglement,
providing a significant advantage in solving complex computational problems."

# Calculate match ratios
match_ratios = watermark_text_and_calculate_matches(text, max_offset=5)
# (result rounded) match_ratio = {0: 0.54, 1: 0.62, 2: 0.58, 3: 0.67, 4: 0.58}
```

The final stage of detection involves determining if the match ratios are statistically significant. To determine whether the text is watermarked, we rely on a binary classification of whether a text is watermarked. For this, we use a pre-trained model based on metrics including Highest Ratio, Average Others, T-Statistic, and P-Value. This approach is necessary because, as illustrated in the graphs later, there is no discernible or easily observable difference between the T-statistics and P-values of watermarked and non-watermarked texts. Consequently, we resort to using a pre-trained model for classification, which has achieved the highest accuracy of 94%.

The module `check_significant_difference` generates a list of significance.

```python
from scipy.stats import ttest_1samp
import numpy as np

def check_significant_difference(match_ratios):
    # Extract ratios into a list
    ratios = list(match_ratios.values())

    # Find the highest ratio
    highest_ratio = max(ratios)

    # Find the average of the other ratios
    other_ratios = [r for r in ratios if r != highest_ratio]
    average_other_ratios = np.mean(other_ratios)

    # Perform a t-test to compare the highest ratio to the average of the others
    t_stat, p_value = ttest_1samp(other_ratios, highest_ratio)

    # Print the results
    print(f"Highest Match Ratio: {highest_ratio}")
    print(f"Average of Other Ratios: {average_other_ratios}")
    print(f"T-Statistic: {t_stat}")
    print(f"P-Value: {p_value}")

    # Determine if the difference is statistically significant (e.g., at the 0.05 significance level)
    if p_value < 0.05:
        print("The highest ratio is significantly different from the others.")
    else:
        print("The highest ratio is not significantly different from the others.")

    return [highest_ratio, average_other_ratios, t_stat, p_value]

# Example usage
text = "Quantum computing is a rapidly evolving field that leverages the principles of quantum mechanics
to perform computations that are infeasible for classical computers. Unlike classical computers, which
use bits as the fundamental unit of information, quantum computers use quantum bits or qubits. Qubits
can exist in multiple states simultaneously due to the principles of superposition and entanglement,
providing a significant advantage in solving complex computational problems."
match_ratios = watermark_text_and_calculate_matches(text, max_offset=5)
check_significant_difference(match_ratios)
```

The module `randomly_add_words` was created to simulate the scenario where additional words have been added to the watermarked test for testing purposes.

```python
import random

def randomly_add_words(text, words_to_add, num_words_to_add):
    # Clean and split the input text
    text = " ".join(text.split())
    words = text.split()

    # Insert words randomly into the text
    for _ in range(num_words_to_add):
        # Choose a random position to insert the word
        position = random.randint(0, len(words))
        # Choose a random word to insert
        word_to_insert = random.choice(words_to_add)
        # Insert the word at the random position
        words.insert(position, word_to_insert)

    # Join the list back into a string and return the modified text
    modified_text = " ".join(words)
    return modified_text

# Example usage
text = "Quantum computing is a rapidly evolving field that leverages the principles of quantum mechanics
to perform computations that are infeasible for classical computers. Unlike classical computers, which
use bits as the fundamental unit of information, quantum computers use quantum bits or qubits. Qubits
can exist in multiple states simultaneously due to the principles of superposition and entanglement,
providing a significant advantage in solving complex computational problems."
words_to_add = ["example", "test", "random", "insert"]
num_words_to_add = 5

# modified_text = randomly_add_words(text, words_to_add, num_words_to_add)
modified_text = randomly_add_words(watermark_text(text, offset=0), words_to_add, num_words_to_add)
(result) modified_text = "Quantum computing is example a rapidly evolving field that leverages the
principles of quantum mechanics to perform random computations that are impossible for classical
computers. Unlike quantum computers, which use bits as the random insert fundamental unit of , quantum
computers use quantum bits or qubits. Qubits can exist in multiple states simultaneously according
random to the principles of symmetry and entanglement, providing a significant advantage in solving
complex mathematical problems."

match_ratios = watermark_text_and_calculate_matches(modified_text, max_offset=5)
# (result rounded) match_ratios = {0: 0.57, 1: 0.57, 2: 0.54, 3: 0.38, 4: 0.77}

check_significant_difference(match_ratios)
# (result rounded)
#    Highest Match Ratio: 0.77
#    Average of Other Ratios: 0.52
#    T-Statistic: -5.66
#    P-Value: 0.01
# The highest ratio is significantly different from the others.
```

Once the list of significance is defined, to show the significance of using a pre-trained model, lets plot them to futher understand the statistical summary. Here is the python code used to generate the plots.

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Assuming list_of_significance and list_of_significance_watermarked are already defined
# Create DataFrames from the lists
df_significance = pd.DataFrame(list_of_significance, columns=['Highest Ratio', 'Average Others', 'T-Statistic', 'P-Value'])
df_significance_watermarked = pd.DataFrame(list_of_significance_watermarked, columns=['Highest Ratio', 'Average Others', 'T-Statistic', 'P-Value'])

# Add a label column to distinguish between the two sets
df_significance['Label'] = 'Original'
df_significance_watermarked['Label'] = 'Watermarked'

# Combine the DataFrames
combined_df = pd.concat([df_significance, df_significance_watermarked], ignore_index=True)

# Perform EDA
def perform_eda(df):
    # Display the first few rows of the DataFrame
    print("First few rows of the DataFrame:")
    print(df.head())

    # Display statistical summary
    print("\nStatistical Summary:")
    print(df.describe())

    # Check for missing values
    print("\nMissing Values:")
    print(df.isnull().sum())

    # Visualize the distributions of the features
    plt.figure(figsize=(12, 8))
    sns.histplot(data=df, x='Highest Ratio', hue='Label', element='step', kde=True)
    plt.title('Distribution of Highest Ratio')
    plt.show()

    plt.figure(figsize=(12, 8))
    sns.histplot(data=df, x='Average Others', hue='Label', element='step', kde=True)
    plt.title('Distribution of Average Others')
    plt.show()

    plt.figure(figsize=(12, 8))
    sns.histplot(data=df, x='T-Statistic', hue='Label', element='step', kde=True)
    plt.title('Distribution of T-Statistic')
    plt.show()

    plt.figure(figsize=(12, 8))
    sns.histplot(data=df, x='P-Value', hue='Label', element='step', kde=True)
    plt.title('Distribution of P-Value')
    plt.show()

    # Pairplot to see relationships
    sns.pairplot(df, hue='Label')
    plt.show()

    # Correlation matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(df.drop(columns=['Label']).corr(), annot=True, cmap='coolwarm')
    plt.title('Correlation Matrix')
    plt.show()

    # T-test to check for significant differences
    original = df[df['Label'] == 'Original']
    watermarked = df[df['Label'] == 'Watermarked']

    for column in ['Highest Ratio', 'Average Others', 'T-Statistic', 'P-Value']:
        t_stat, p_value = ttest_ind(original[column], watermarked[column])
        print(f"T-test for {column}: T-Statistic = {t_stat}, P-Value = {p_value}")
```

```
# Perform EDA on the combined DataFrame
perform_eda(combined_df)

# Check if the data is ready for machine learning classification

# Prepare the data
X = combined_df.drop(columns=['Label'])
y = combined_df['Label']

# Convert labels to numerical values for ML model
y = y.map({'Original': 0, 'Watermarked': 1})

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a RandomForestClassifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Feature importances
feature_importances = clf.feature_importances_

# Create a DataFrame for feature importances
feature_importances_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importances_df, palette='viridis')
plt.title('Feature Importances')
plt.show()

# Heatmap for feature importances
plt.figure(figsize=(10, 8))
sns.heatmap(feature_importances_df.set_index('Feature').T, annot=True, cmap='viridis')
plt.title('Heatmap of Feature Importances')
plt.show()
```

The plots are created using the result from our previous example with check_significant_difference returned: Highest Match Ratio: 0.7692307692307693 Average of Other Ratios: 0.5164835164835164 T-Statistic: -5.66220858504931 P-Value: 0.010908789440745323

Missing Values: Highest Ratio 0 Average Others 0 T-Statistic 1 P-Value 1 Label 0 dtype: int64

From the graphs and statistical summaries, several inferences can be drawn regarding the distributions and relationships between the variables in the dataset:

Distribution of Highest Ratio: The distribution of the “Highest Ratio” variable shows a clear distinction between the “Original” and “Watermarked” categories. The “Original” category has a peak around 0.4, while the “Watermarked” category peaks around 0.5, indicating a shift in the distribution towards higher values for the watermarked data.

Distribution of Average Others: Similarly, the “Average Others” variable shows a distinction between the two categories. The “Original” category peaks around 0.3, whereas the “Watermarked” category peaks slightly higher, around 0.4. This suggests that the average

Table 1. *First few rows of the DataFrame*

	Highest Ratio	Average Others	T-Statistic	P-Value	Label
0	0.233333	0.182203	-3.532758	0.038563	Original
1	0.203390	0.139195	-3.440591	0.041218	Original
2	0.338983	0.270339	-2.228608	0.112142	Original
3	0.254237	0.168362	-2.451613	0.246559	Original
4	0.288136	0.210876	-5.467540	0.012026	Original

Table 2. *Statistical Summary*

	Highest Ratio	Average Others	T-Statistic	P-Value
count	4000.000000	4000.000000	3999.000000	3999.000000
mean	0.490285	0.339968	-6.076672	0.036783
std	0.128376	0.082900	5.580957	0.043217
min	0.101695	0.066667	-111.524590	0.000002
25%	0.416667	0.296610	-6.938964	0.006418
50%	0.491525	0.354732	-4.431515	0.021973
75%	0.573770	0.398224	-3.176861	0.052069
max	0.868852	0.580601	-1.166065	0.451288

values for other ratios are higher in the watermarked data compared to the original data. Distribution of T-Statistic:

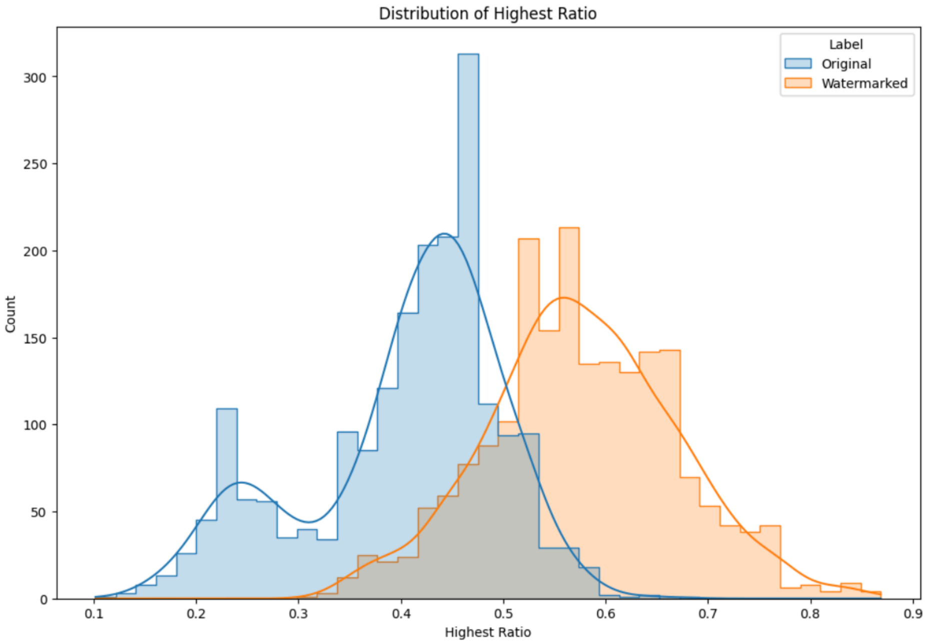

Figure 2. *Distribution of highest ratio*

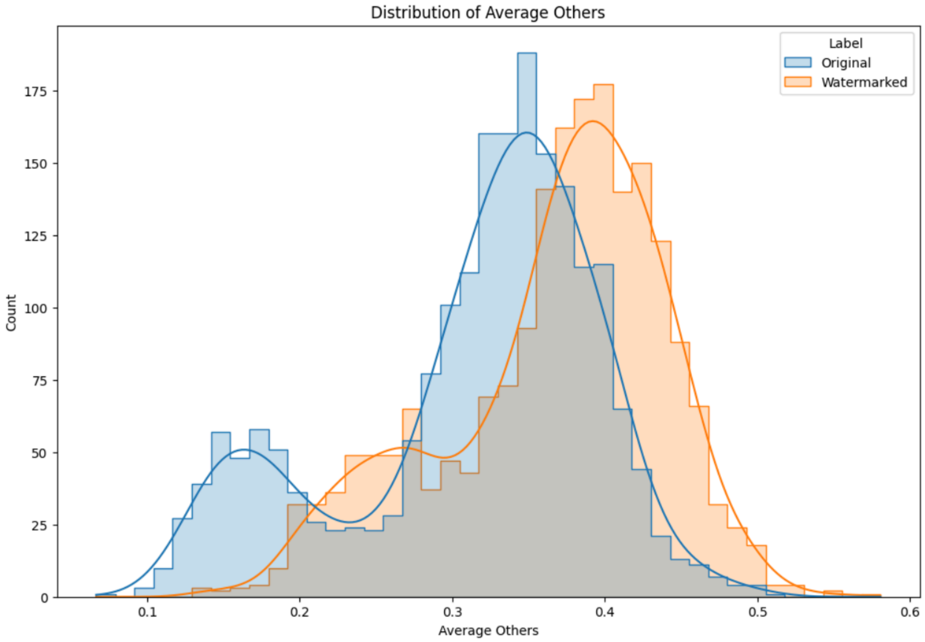

Figure 3. *Distribution of average others*

Distribution of T-statistic: The distribution of the T-statistic is highly skewed to the left for both categories, with a long tail extending to very negative values. The "Original" category appears to have a more pronounced peak near 0, while the "Watermarked" category has a lower count at the peak and a wider spread.

Distribution of P-Value: The P-value distribution is heavily skewed towards 0 for both categories, with the "Watermarked" category showing a sharper peak at 0. This suggests that most of the tests result in very low p-values, indicating strong statistical significance in the differences observed.

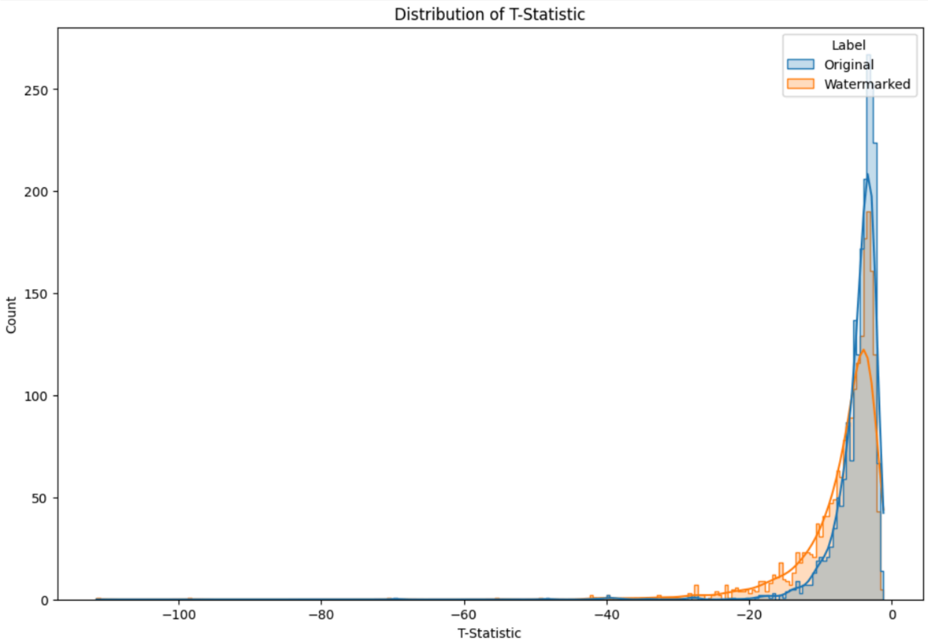

Figure 4. *Distribution of t-statistics*

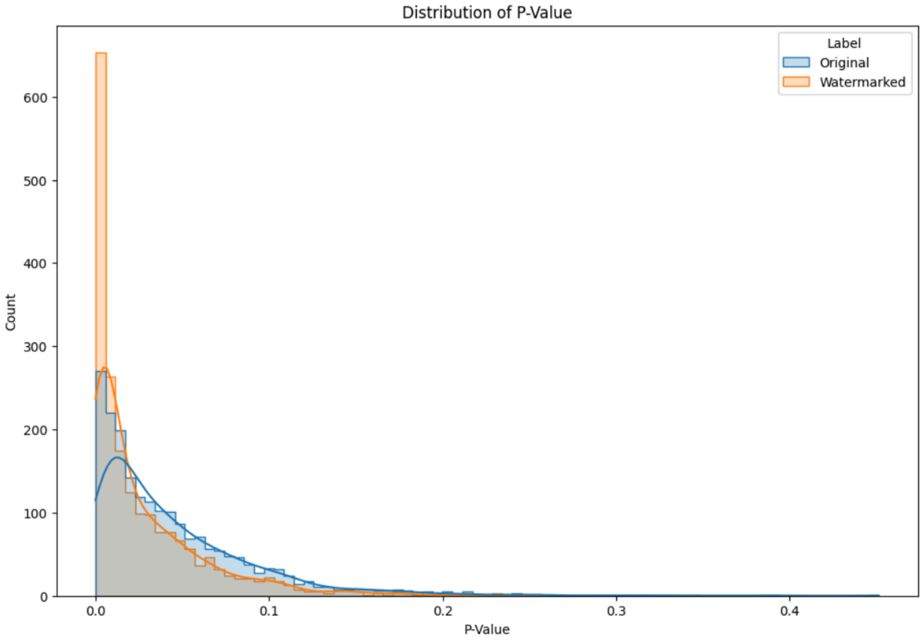

Figure 5. *Distribution of P-value*

Pair Plot: The pair plot provides a visual comparison of the relationships between the variables for the two categories. There are clear clusters and separations between the "Original" and "Watermarked" categories in the scatter plots, particularly for "Highest Ratio" vs. "Average Others" and "Highest Ratio" vs. "P-Value". This reinforces the idea that the watermarked data exhibits different characteristics compared to the original data.

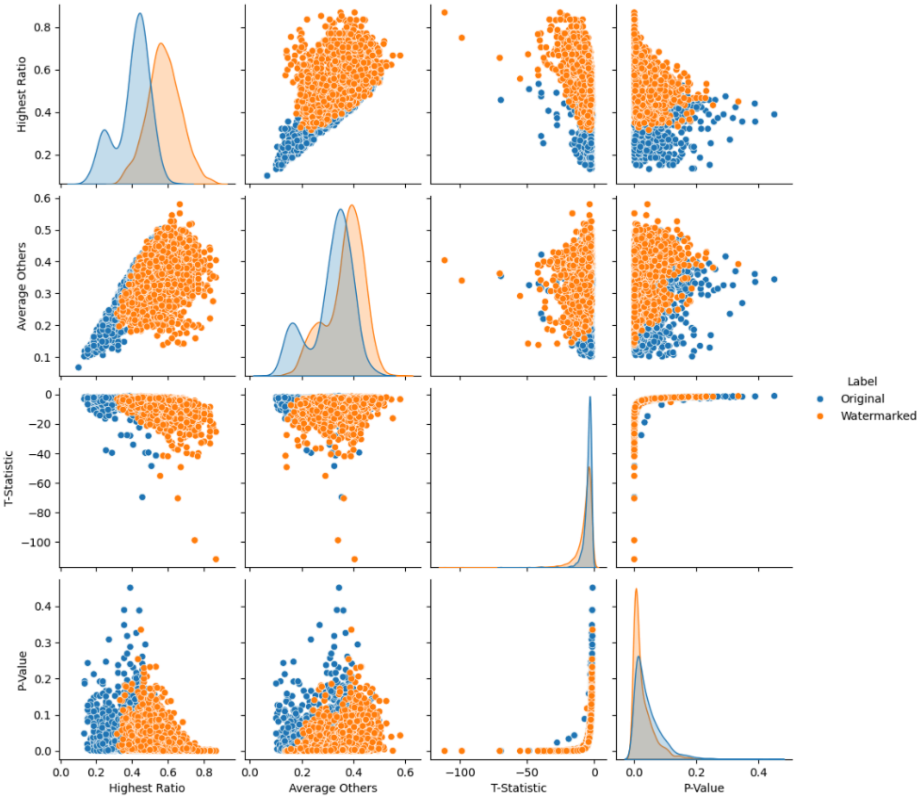

Figure 6. *Dataset*

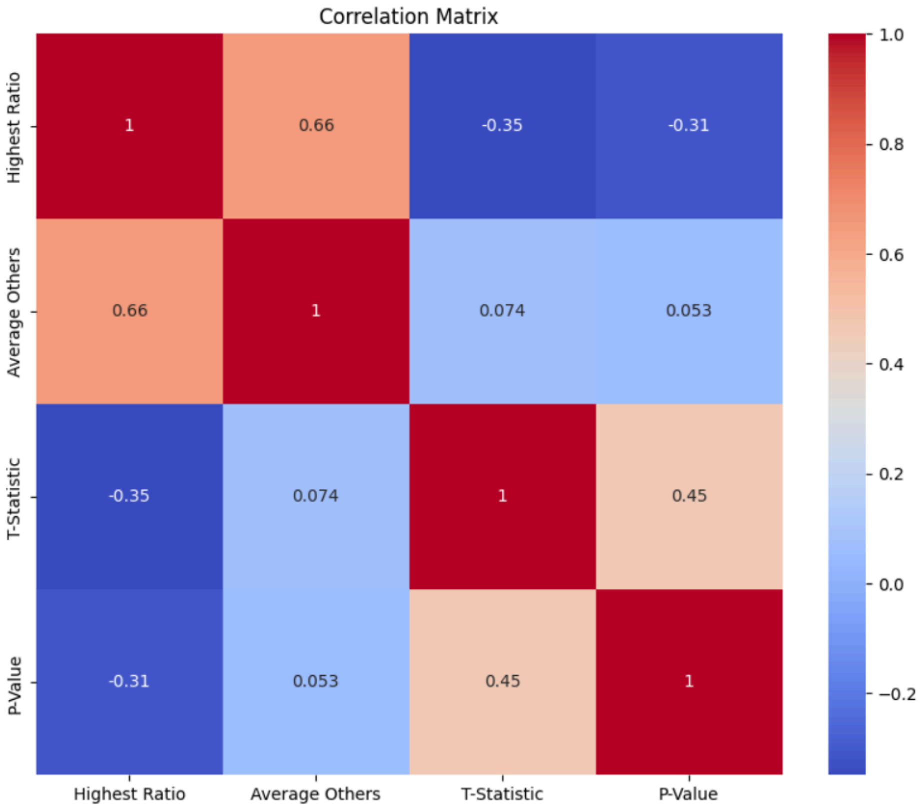

Figure 7. *Correlation_Matrix*

Correlation Matrix: The correlation matrix shows the pairwise correlation coefficients between the variables. "Highest Ratio" and "Average Others" are positively correlated (0.66), indicating that higher values of the highest ratio tend to be associated with higher average values of other ratios. "T-Statistic" has a negative correlation with "Highest Ratio" (-0.35) and "P-Value" (-0.31), suggesting that higher ratios tend to result in more negative T-statistics and lower p-values.

Overall Observations: The "Watermarked" data tends to have higher ratios and averages compared to the "Original" data. The T-statistics and p-values indicate strong statistical differences between the original and watermarked categories. The pair plot and correlation matrix provide further evidence of distinct patterns and relationships in the watermarked data compared to the original data.

While these plots do show a difference between the watermarked and non-watermarked text, using a pre-trained model help us achieve higher efficiency and consistency in our comparisons.

7. Model Training, Testing and Efficiency

The algorithm in this paper was trained using a dataset generated from Gutenberg's top 10 books [6], [7], [8], [9], [10], [11], [12], [13], [14], [15]. Specifically, 2000 random 300-word paragraphs were taken from these books, ensuring an equal number of paragraphs from each book. Each paragraph was watermarked, and then statistical analysis was performed. The Highest Match Ratio, Average of Other Ratios, T-Statistic, and P-Value were calculated and stored in Results.csv. The models were trained using an 80/20 split of the dataset, with the following models being trained: Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, Gradient Boosting, AdaBoost, Naive Bayes, and K-Nearest Neighbors. Gradient Boosting gave the highest accuracy, resulting in an overall accuracy of 94% in identifying watermarked text

Code used for model training

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Assuming list_of_significance and list_of_significance_watermarked are already defined
# Create DataFrames from the lists
df_significance = pd.DataFrame(list_of_significance, columns=['Highest Ratio', 'Average Others', 'T-Statistic', 'P-Value'])
df_significance_watermarked = pd.DataFrame(list_of_significance_watermarked, columns=['Highest Ratio', 'Average Others', 'T-Statistic', 'P-Value'])

# Add a label column to distinguish between the two sets
df_significance['Label'] = 'Original'
df_significance_watermarked['Label'] = 'Watermarked'

# Combine the DataFrames
combined_df = pd.concat([df_significance, df_significance_watermarked], ignore_index=True)
combined_df = combined_df.dropna()

# Prepare the data
X = combined_df.drop(columns=['Label'])
y = combined_df['Label']

# Convert labels to numerical values for ML model
y = y.map({'Original': 0, 'Watermarked': 1})

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize models
models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Support Vector Machine': SVC(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'Naive Bayes': GaussianNB(),
    'K-Nearest Neighbors': KNeighborsClassifier()
}

# Train and evaluate models
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_test, y_pred))
    print(f"\n{model_name} Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    # Feature importances (only for models that provide it)
    if hasattr(model, 'feature_importances_'):
        feature_importances = model.feature_importances_
        feature_importances_df = pd.DataFrame({
            'Feature': X.columns,
            'Importance': feature_importances
        }).sort_values(by='Importance', ascending=False)

        # Plot feature importances
        plt.figure(figsize=(12, 8))
        sns.barplot(x='Importance', y='Feature', data=feature_importances_df, palette='viridis')
        plt.title(f'{model_name} Feature Importances')
        plt.show()
```

Code for Model testing

```python
import os
import random

def extract_test_cases(folder_path, num_cases=2000, words_per_case=300):
    test_cases = []
    book_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]

    # Calculate the number of test cases to extract from each book
    cases_per_book = num_cases // len(book_files)
    extra_cases = num_cases % len(book_files)

    for book_file in book_files:
        with open(os.path.join(folder_path, book_file), 'r', encoding='utf-8') as file:
            text = file.read()
            words = text.split()
            num_words = len(words)

            # Ensure enough words are available to extract the cases
            if num_words < words_per_case:
                continue

            # Determine the number of cases to extract from this book
            num_cases_from_book = cases_per_book
            if extra_cases > 0:
                num_cases_from_book += 1
                extra_cases -= 1

            for _ in range(num_cases_from_book):
                start_index = random.randint(0, num_words - words_per_case)
                case = ' '.join(words[start_index:start_index + words_per_case])
                test_cases.append(case)

                if len(test_cases) == num_cases:
                    return test_cases

    return test_cases

# Usage example
folder_path = 'books'
test_cases = extract_test_cases(folder_path)
```

```python
list_of_significance = []
list_of_significance_watermarked = []
count_t = 0
for text in test_cases:
  count_t+=1
  print("______________________________")
  print("Doing", count_t)
  print("______________________________")

  words_to_add = ["example", "test", "random", "insert"]
  num_words_to_add = 5

  modified_text = randomly_add_words(watermark_text(text, offset=0), words_to_add, num_words_to_add)

  match_ratios = watermark_text_and_calculate_matches(modified_text, max_offset=5)
  list_of_significance_watermarked.append(check_significant_difference(match_ratios))

  match_ratios = watermark_text_and_calculate_matches(text, max_offset=5)
  list_of_significance.append(check_significant_difference(match_ratios))

  print("______________________________")
  print("Done", count_t, )
  print("______________________________")
```

8. Analysis of the Algorithm

Strengths:

1. Robustness against attacks: The BERT-based watermarking algorithm uses the sophisticated context-understanding capability of BERT to embed watermarks. This makes the watermark integration deeply intertwined with the text's semantic structure, which is difficult to detect and remove without altering the underlying meaning, thus providing robustness against simple text manipulation attacks.
2. Comparison with existing methods: Compared to traditional watermarking methods like word context and UniSpaCh, the BERT-based approach offers a more adaptable and less detectable method. It does not rely on altering visible text elements or patterns easily erased, like white spaces or specific word sequences. Instead, it uses semantic embedding, making it superior in maintaining the natural flow and readability of the text.
3. Scalability and adaptability: The method is scalable to different languages and text forms by adjusting the BERT model used. It can be adapted to work with different BERT variants trained on specific datasets, enhancing flexibility in deployment.

Challenges:

1. Dependency on model consistency: The watermark detection relies heavily on the consistency of the BERT model's output. Any updates or changes in the model could potentially alter the watermark, making it undetectable. If the watermark can embbed some sort of version history and control, this could be managed.
2. Data Integrity is highly dependent on the Model: the integrity of the watermarked text depends on how good the model is at replacing the given word, due to the nature of AI-generated text where all the previous tokens are used to generate new ones BERT watermarking can preserve integrity much more effectively. However if it were to watermark text which is completely different from its training dataset it might return an incoherent output, for example if the dataset of BERT consists of scientific papers it will struggle immensely when trying to watermark fairy tails.
3. Potential for false positives/negatives: Given the probabilistic nature of BERT's predictions, there is a risk of incorrect watermark detection, especially in texts with complex semantics or those that closely mimic the watermark patterns without actually being watermarked.
4. Potential loss of context: When words are replaced, the intended context of delivery could be altered. However, AI models are continually improving, and we hope that a well-trained model can significantly mitigate this risk.

Real-world applicability:

1. Versatility in applications: This method can be applied across various fields such as copyright protection, and content authentication, and in legal and academic settings where proof of authorship is crucial. It is particularly beneficial for managing copyrights in digital media, academic papers, and any online content where text is dynamically generated or reused.
2. Integration with existing systems: The algorithm can be seamlessly integrated with current content management systems (CMS) and digital rights management (DRM) systems, enhancing their capabilities to include advanced text watermarking features. This integration helps organizations maintain control over their content distribution and monitor usage without invasive methods.
3. Application in AI-generated text: With the proliferation of AI-generated content from models like ChatGPT, GPT-4, and other AI writing assistants, distinguishing between human-generated and AI-generated text becomes crucial. The BERT-based watermarking can be used to embed unique, non-intrusive identifiers into AI-generated texts, ensuring that each piece of content can be traced back to its source. This is particularly valuable in preventing the spread of misinformation, verifying the

authenticity of content, and in applications where copyright claims on AI-generated content might be disputed.

4. Forensic Linguistics in Cybersecurity: In cybersecurity, determining the origin of phishing emails or malicious texts can be crucial. BERT-based watermarking can assist forensic linguists and security professionals by providing a means to trace the origins of specific texts back to their creators, helping to identify patterns or sources of cyber threats.
5. Enhanced Licensing Control for Digital Text: As digital content licensing becomes more complex with different rights for different geographies and platforms, watermarking can help content owners and licensing agencies enforce these rights more effectively. The watermark makes it easier to enforce and monitor compliance automatically.

9. Conclusion

By leveraging the BERT model and the proposed algorithm, we have achieved a 94% accuracy rate in detecting watermarked text. With an appropriate training dataset and ongoing advancements in AI technology, this approach promises even more robust watermarking techniques. This progress will enhance our ability to identify AI-generated content and provide an effective means for detecting plagiarism.

References

[1] N. S. Kamaruddin, A. Kamsin, L. Y. Por, and H. Rahman, “A Review of Text Watermarking: Theory, Methods, and Applications,” *IEEE Access*, vol. 6, no. 3, 2018, doi: 10.1109/ACCESS.2018.2796585.

[2] Y. Wu, Z. Jin, C. Shi, P. Liang, and T. Zhan, “Research on the Application of Deep Learning-based BERT Model in Sentiment Analysis,” *ArXiv*, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:268379403

[3] Z. Jalil and A. M. Mirza, “A Review of Digital Watermarking Techniques for Text Documents,” in *2009 International Conference on Information and Multimedia Technology*, 2009, pp. 230–234. doi: 10.1109/ICIMT.2009.11.

[4] L. Y. Por, K. Wong, and K. O. Chee, “UniSpaCh: A text-based data hiding method using Unicode space characters,” *Journal of Systems and Software*, vol. 85, no. 5, pp. 1075–1082, 2012, doi: https://doi.org/10.1016/j.jss.2011.12.023.

[5] T. Lancaster, “Artificial intelligence, text generation tools and ChatGPT - does digital watermarking offer a solution?,” *Int J Educ Integr*, vol. 19, no. 10, pp. 8011–8028, 2023, doi: https://doi.org/10.1007/s40979-023-00131-6.

[6] W. Shakespeare, *Romeo and Juliet*. USA, 1998. [Online]. Available: https://www.gutenberg.org/ebooks/1513

[7] H. Melville, *Moby Dick; Or, The Whale*. USA, 2001. [Online]. Available: https://www.gutenberg.org/ebooks/2701

[8] J. Austen, *Pride and Prejudice*. USA, 1998. [Online]. Available: https://www.gutenberg.org/ebooks/1342

[9] M. W. Shelley, *Frankenstein; Or, The Modern Prometheus*. USA, 1993. [Online]. Available: https://www.gutenberg.org/ebooks/84

[10] G. Eliot, *Middlemarch*. USA, 1994. [Online]. Available: https://www.gutenberg.org/ebooks/145

[11] W. Shakespeare, *The Complete Works of William Shakespeare*. USA, 1994. [Online]. Available: https://www.gutenberg.org/ebooks/100

[12] E. M. Forster, *A Room with a View*. USA, 2001. [Online]. Available: https://www.gutenberg.org/ebooks/2641

[13] L. M. Alcott, *Little Women; Or, Meg, Jo, Beth, and Amy*. USA, 2011. [Online]. Available: https://www.gutenberg.org/ebooks/37106

[14] L. M. Montgomery, *The Blue Castle*. USA, 2022. [Online]. Available: https://www.gutenberg.org/ebooks/67979

[15] E. V. Arnim, *The Enchanted April*. USA, 2005. [Online]. Available: https://www.gutenberg.org/ebooks/16389