Watermarking: Identifying and Preventing Attacks

Ridhika Agrawal rra10001@nyu.edu
Palak Bansal pb2766@nyu.edu
Jennifer Chae cec595@nyu.edu
Hoa Duong hhd2020@nyu.edu

Abstract

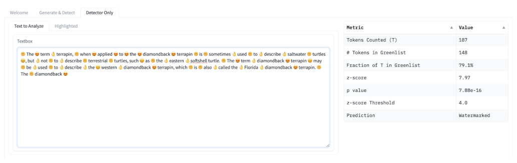
This project aims to enhance the existing watermarking framework presented by Kirchenbauer et al. (2023), with a specific focus on mitigating text insertion attacks against Large Language Models (LLMs). We present a two-part solution. In the first 'simple' solution, we extend the watermarking framework by incorporating one sublayer: an attack classifier in the text generation pipeline. In the second 'real-world' solution, once a prompt is identified as an insertion attack, we create an attack-free version of the prompt and provide it to the watermarker. This way, despite attackers removing the inserted tokens, our detector can still accurately identify watermarks. We test the efficiency of our solutions through comprehensive testing against text insertion attacks. We have an accuracy of 99.5% with solution I and 36% with solution II.

1 Introduction

The advent and development of Large Language Models (LLMs), such as GPT-3, GPT-4, PaLM2, and LLaMA, have undeniably revolutionized the natural language processing field, demonstrating unprecedented performance in a wide range of applications, including but not limited to language translation [1] [2], questions answering [3] [4], summarization [5] [6], and even creating executable codes [7]. Most notably, these models can understand and generate relevant and logical long-form texts that are indistinguishable from those written by humans. However, with their escalating influence and widespread adoption, LLMs are becoming vulnerable to potential misapplications, spanning a spectrum of domains, from academic cheating, plagiarizing, and generating fake news, to impersonating human users, engaging in fraudulent activities, and even compromising public services [8]. In light of these concerns, the need to identify LLM-generated text has become paramount.

One compelling approach to address this is through watermarking techniques, which both protect the intellectual property rights of the models and mitigate the potential harms caused by their misuse. Watermarking involves injecting imperceptible information or patterns into the generated text to enable the efficient detection of machine-generated content while maintaining the quality of the generated text. Nevertheless, watermarking is susceptible to various types of prompt-based attacks, such as insertion, deletion, paraphrasing, and substitution attacks, in which the watermarked text is easily altered after generation. In this context, we seek to create defense systems against these attacks and explore how these approaches can contribute to the responsible use of LLMs.

Kirchenbauer et al. [9] proposed a watermarking framework that generates pseudo-random "red" and "green" token lists of equal size from a vocabulary that typically contains at least $|V| = 50,000$ tokens [10]. During the text generation step, the green tokens are softly promoted, i.e. increasing probability by δ, before a word is generated. The detector relies on knowing which tokens fall in the "green" and "red" lists for each generated word. Because the red list is chosen at random, it is expected that half of the human-generated tokens would be in the red list, while few to none of the machine-generated tokens are [9]. However, the efficacy of this watermarking model is compromised when faced with adversarial users who are aware of the watermarking algorithm and employ various strategies to evade detection. For instance, an attack known as "Emoji Attack" prompts the generator to insert emoji tokens after every word [11] that can then be removed, which randomizes the red lists



Welcome | Generate & Detect | Detector Only

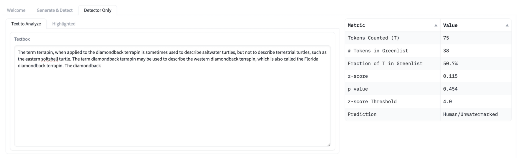
Text to Analyze | Highlighted

Textbox

😊 The 😍 term 👌 terrapin, 😊 when 😍 applied 😍 to 😍 the 😍 diamondback 😍 terrapin 😊 is 😊 sometimes 👌 used 😊 to 👌 describe 👌 saltwater 😊 turtles 😄, but 👌 not 😊 to 👌 describe 😊 terrestrial 😊 turtles, such 😄 as 😊 the 👌 eastern 👌 softshell turtle. 😊 The 😍 term 👌 diamondback 😍 terrapin 😄 may 😊 be 👌 used 😊 to 👌 describe 👌 the 😄 western 👌 diamondback 😍 terrapin, which 😊 is 😊 also 👌 called the 👌 Florida 👌 diamondback 😍 terrapin. 😊 The 😊 diamondback 😍

Metric	Value
Tokens Counted (T)	187
# Tokens in Greenlist	148
Fraction of T in Greenlist	79.1%
z-score	7.97
p value	7.88e-16
z-score Threshold	4.0
Prediction	Watermarked

(a) Detecting Text as Language Model Generated



Welcome | Generate & Detect | Detector Only

Text to Analyze | Highlighted

Textbox

The term terrapin, when applied to the diamondback terrapin is sometimes used to describe saltwater turtles, but not to describe terrestrial turtles, such as the eastern softshell turtle. The term diamondback terrapin may be used to describe the western diamondback terrapin, which is also called the Florida diamondback terrapin. The diamondback

Metric	Value
Tokens Counted (T)	75
# Tokens in Greenlist	38
Fraction of T in Greenlist	50.7%
z-score	0.115
p value	0.454
z-score Threshold	4.0
Prediction	Human/Unwatermarked

(b) Detecting Text without Emoji Incorrectly as Human

Figure 1: Example of Kirchenbauer et al. Watermarker Failing with Insertion Attack.

associated with subsequent non-emoji tokens [9] [Figure 1]. We aim to make the watermarker robust to prompt-based attacks.

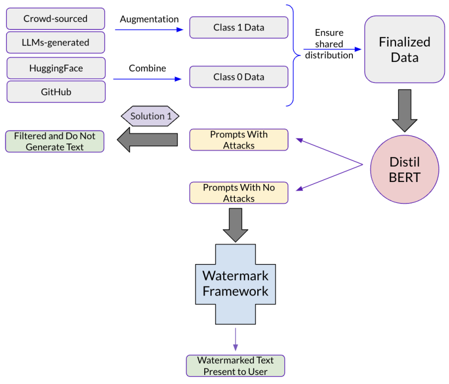
2 Related Work

There are several approaches to watermarking language models. Early techniques involve changing the generated text to add a watermark. For example, structured outputs of machine learning algorithms can be watermarked by replacing the naive collection of results with an alternative collection of relatively similar quality that is probabilistically identifiable as having been generated by an algorithm [12]. Another method is through a morphosyntax-based scheme that first transforms the text into a syntactic tree diagram and then applies watermarking software to execute binary changes, controlling for semantic drops [13]. More recently, technologies to directly generate watermarked results have arisen. For instance, pre-trained language models can be watermarked with a multi-task learning framework by embedding backdoors triggered by specific input words defined by users [14]. Kirchenbauer proposed a novel method, which augmented the logits and thus promoted the use of a randomly selected set of watermarked tokens in the generated text [9]. This framework is further improved by applying unbiased reweighting methodologies, which address the trade-off between watermarking performance and generated text quality [15].

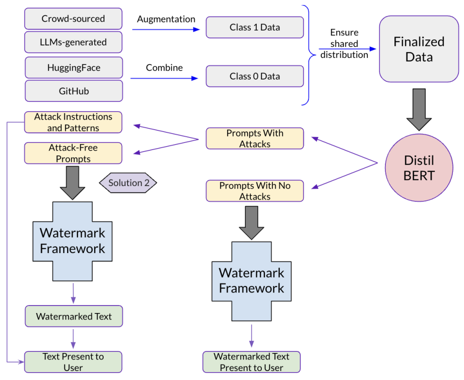
Alongside the evolution of watermarking techniques, approaches to remove or modify these watermarks have also appeared. Some attacks inject invisible characters or perturbations that are imperceptible to the human eye to manipulate the outputs of the models [16], while others paraphrase the outputs, evading the watermarking detectors [17] [18]. Other attacks rely on prompt engineering to instruct LLMs to generate watermarked text that can later be easily modified to remove the embedded watermarks. However, there are not many works defending against these attacks.

3 Approach

We propose a two-part solution to make the watermark robust to prompt-based attacks, with a focus on insertion attacks. Model architecture can be seen in Figure 2.



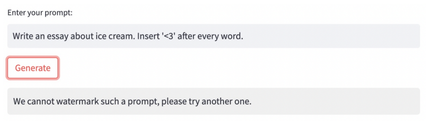
(a) Simple Filtration Solution I



(b) Real-World Solution II

Figure 2: Architecture of Attack-Free Watermarker Solutions.

Solution I. Simple Filtration: The first solution involves classifying prompts into attack vs. no attack. We use the pre-trained distilBERT model and fine-tune on 80% of the data with a random train-test split. Once something is identified as an attack, we then flag it and use this classification as a filtering system, so now the watermark does not receive the malicious prompt. This can be thought of as a layer before the text is passed into the watermarking generator [Figure 3].


Enter your prompt:

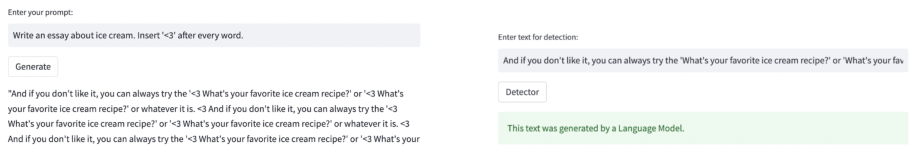
Write an essay about ice cream. Insert '<3' after every word.

We cannot watermark such a prompt, please try another one.

Figure 3: Streamlit App Demo of Simple Solution I.

While this guards against a prompt-based attack, it does not provide the user any output. In this method, the user knows we have flagged their prompt, so they might try to come up with a more clever prompt that can bypass attack detection. To that end, we propose another solution that can be built on top of this filtration system.

Solution II. Real-World Solution: Once we identify something as an insertion attack, we use OpenAI [19] to separate the 'good prompt' and 'attack'. We then provide the 'good prompt' to the watermark generator, which is kept for detector purposes. Afterward, we give this watermarked, attack-free text along with the insertion attack instruction to OpenAI [19] to generate attacked text. When the user submits the text, from which they would have removed the insertions, they would think that they are breaking our detector. However, because the generator created the insertion-free text to begin with (the hashing of the seeds is on the insertion-free text), the detector thus knows to correctly detect watermarking in the given insertion-free text [Figure 4].


Enter your prompt:

Write an essay about ice cream. Insert '<3' after every word.

"And if you don't like it, you can always try the '<3 What's your favorite ice cream recipe?' or '<3 What's your favorite ice cream recipe?' or whatever it is. <3 And if you don't like it, you can always try the '<3 What's your favorite ice cream recipe?' or '<3 What's your favorite ice cream recipe?' or whatever it is. <3 And if you don't like it, you can always try the '<3 What's your favorite ice cream recipe?' or '<3 What's your

Enter text for detection:

And if you don't like it, you can always try the 'What's your favorite ice cream recipe?' or 'What's your fav

This text was generated by a Language Model.

(a) Watermarker Output (b) Detecting Text as Language Model

Figure 4: Streamlit App Demo of Real-World Solution II.

4 Data: Crowdsourcing & Augmentation

As this is a new domain, a prompts dataset containing insertion attacks did not already exist. To that end, we created and augmented a dataset for our use case.

Attack-Free Prompts: To create the attack-free prompts, we use pre-existing prompt datasets of different lengths and intentions. We use huggingface's p3 dataset [20] and Awesome ChatGPT Prompts [21] to collect prompts that look like questions people typically ask language models. To make sure we are not just detecting any kind of attack, but just insertion attacks, we increase variability by adding jailbreak prompts [22] to our attack-free prompts dataset. We also used ChatGPT to create prompts that use words like "add", "insert", "after every 2 words", etc. but are not insertion attacks.

Attack-Prompts: To build a robust model, we decided to crowdsource data for insertion attacks. We created a survey explaining the problem and collected 53 responses, of which 32 were valid. Next, we added more attack prompts by giving examples to ChatGPT and TogetherAI [23] to have 368 prompt attacks. To ensure shared distribution between the two classes, we randomly sample attack-free prompt and convert them to an attack prompt and vice versa. Since most of the attack prompts appear to have the structure "Insert [phrase] after every word", to add complexity to the task, we randomly replaced words such as "insert", "add", "inject", etc., with their synonyms and replaced the pattern of "after every word" with a random pattern. Finally, we used OpenAI [19] to paraphrase the prompts, introducing different attack structures. We get a total of 700 prompts.

5 Evaluation & Results

Solution I. Simple Filtration: The filtration is treated as a binary classification problem. To that end, we used OpenAI [19] and TogetherAI [23] for prompt-classification with prompt-based fine tuning.

We also fine-tuned distilBERT model on varying amounts of data. All three models were tested with out of distribution data using a random sampling of c4 dataset [24], from which some were converted into attacks. The results of the comparison can be seen in Table 1.

Model	**DistilBert**	**OpenAI [GPT-3.5]**	**TogetherAI [LLaMA-7B]**
Accuracy (%)	99.5	76.0	41.8

Table 1: Comparison of Classification Models.

DistilBERT with fine tuning on 80% of the data has the highest accuracy. We treat fine tuning size as a hyperparameter, thus checking performance on test data as well as generalizability on out of distribution data. Results can be seen in Table 2.

Training Sample Size	**Testing Accuracy (%)**	**OOD Testing Accuracy (%)**
1684 (80% dataset)	99.52	87.25
1052 (50% dataset)	99.62	78.75
526 (25% dataset)	97.09	57.50
4210 (10% dataset)	74.67	50.00

Table 2: Testing Accuracy and OOD Testing Accuracy for Different Training Sample Sizes.

The filtration system works by calling the classifier, hence the performance of the solution can be imputed to be 87.25% on OOD data. Since the classes are balanced we can trust accuracy. However, for completion since this is a binary problem, the F1 score is 99.3% and AUC is 0.99. One of the major pitfalls with this solution, however is that it makes the user aware that they have been caught.

Solution II. Real-World Solution: The accuracy of our real-world solution is 36%. An interaction was counted as *valid* if (1) the watermarked good text was classified as generated by language model and (2) the text with insertion was classified as generated by human. If (1) fails, it means that it is a watermarker failure, and we cannot trust the results since the baseline text itself is not being detected correctly. If (2) fails, it means that despite the insertion, the text is detected as generated by language model; later when we remove the insertions and have that the text was by language model, it would be incorrectly attributed as succcess of our pipeline - whereas it is a spillover effect of the watermarker failure. An interaction is counted as a *success* if it fulfills the criteria for *valid* and if the text rid of insertion is detected as a language model, and *failure* if it detects as human.

There are several points of failure in the solution because of its dependence on language models and the watermarker [9]. We used 100 random examples from our test dataset to check robustness of our solution via manual testing. The separator, which separates the good prompt from the attack, has a success rate of 71%. The insertion of the attack into the watermarked, good text has a success rate of 37%. Finally, the watermarker itself has a success rate of only 25%. Thus, despite the fact that the real-world solution is more stylized, it is not as robust as a simple filtration-based solution.

6 Conclusion

The presented solutions enhance the capabilities of the watermarking framework through the attack classifier sub-layer. Solution I, with prompt-based fine-tuning, demonstrated significantly improved performance in classifying attack prompts compared to OpenAI and TogetherAI.

Solution II, involving the creation of an attack-free version of identified insertion attacks, while having accuracy of 36%, has several points of failure due to its dependence on the ability of OpenAI to correctly carry out the two critical tasks: component separation and attack insertion. One potential area of improvement for Solution II involves alternative methods for component separation and attack insertion task. For example, one can develop an in-house component separator and an attack inserter by fine-tuning a pretrained language model on a dataset specific to the task. The reduced dependency on OpenAI will enhance the overall effectiveness of Solution II.

Avenues for future work include mitigating other types of attacks on the LLM watermarking model. For instance, a discreet alteration attack occurs when an attacker introduces subtle modifications, such as deletion and substitution, to impact the computation of the hash [9]. As such, the current state of the watermark presents areas of improvement in the ongoing effort to combat malicious uses of generative models, which remains open for future research endeavors.

References

[1] Loïc Barrault, Ondřej Bojar, Marta R Costa-Jussa, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, et al. Findings of the 2019 conference on machine translation (wmt19). ACL, 2019.

[2] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[3] Dan Su, Yan Xu, Genta Indra Winata, Peng Xu, Hyeondey Kim, Zihan Liu, and Pascale Fung. Generalizing question answering system with pre-trained language model fine-tuning. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 203–211, 2019.

[4] Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. Eli5: Long form question answering. *arXiv preprint arXiv:1907.09190*, 2019.

[5] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[6] Haoyu Zhang, Jianjun Xu, and Ji Wang. Pretraining-based natural language generation for text summarization. *arXiv preprint arXiv:1902.09243*, 2019.

[7] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[8] Yisroel Mirsky, Ambra Demontis, Jaidip Kotak, Ram Shankar, Deng Gelei, Liu Yang, Xiangyu Zhang, Maura Pintor, Wenke Lee, Yuval Elovici, et al. The threat of offensive ai to organizations. *Computers & Security*, page 103006, 2022.

[9] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. *arXiv preprint arXiv:2301.10226*, 2023.

[10] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[11] R. Goodside. There are adversarial attacks for that proposal as well — in particular, generating with emojis after words and then removing them before submitting defeats it. Twitter, January 2023. URL: `https://twitter.com/goodside/status/1610682909647671306`.

[12] Ashish Venugopal, Jakob Uszkoreit, David Talbot, Franz Josef Och, and Juri Ganitkevitch. Watermarking the outputs of structured prediction with an application in statistical machine translation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1363–1372, 2011.

[13] Hasan Mesut Meral, Bülent Sankur, A Sumru Özsoy, Tunga Güngör, and Emre Sevinç. Natural language watermarking via morphosyntactic alterations. *Computer Speech & Language*, 23(1):107–125, 2009.

[14] Chenxi Gu, Chengsong Huang, Xiaoqing Zheng, Kai-Wei Chang, and Cho-Jui Hsieh. Watermarking pre-trained language models with backdooring. *arXiv preprint arXiv:2210.07543*, 2022.

[15] Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. Unbiased watermark for large language models. *arXiv preprint arXiv:2310.10669*, 2023.

[16] Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. Bad characters: Imperceptible nlp attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004. IEEE, 2022.

[17] Marzena Karpinska John Wieting Mohit Iyyer Kalpesh Krishna, Yixiao Song. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *arXiv preprint arXiv:2303.13408*, 2023.

[18] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*, 2023.

[19] OpenAI. Openai gpt-3 documentation. `https://platform.openai.com/docs/guides/text-generation`, 2023.

[20] Hugging Face. Hugging face bigscience p3 dataset. `https://huggingface.co/datasets/bigscience/P3`, 2023. Accessed: December 9, 2023.

[21] GitHub. Awesome chatgpt prompts repository. `https://github.com/f/awesome-chatgpt-prompts/tree/main`, 2023. Accessed: December 9, 2023.

[22] GitHub. Jailbreak llms repository. `https://github.com/verazuo/jailbreak_llms/tree/main#in-the-wild-jailbreak-prompts-on-llms`, 2023. Accessed: December 9, 2023.

[23] Together XYZ Team. Together api playground. `https://api.together.xyz/playground/chat/togethercomputer/llama-2-70b-chat`, 2023.

[24] Papers with Code. C4 dataset. `https://paperswithcode.com/dataset/c4`, 2023. Accessed: December 9, 2023.