ChatGPT-Generated Code Assignment Detection Using Perplexity of Large Language Models (Student Abstract)

Zhenyu Xu, Ruoyu Xu, Victor S. Sheng

Department of Computer Science, Texas Tech University
zhenxu@ttu.edu, ruoyxu@ttu.edu, victor.sheng@ttu.edu

Abstract

In the era of large language models like ChatGPT, maintaining academic integrity in programming education has become challenging due to potential misuse. There's a pressing need for reliable detectors to identify ChatGPT-generated code. While previous studies have tackled model-generated text detection, identifying such code remains uncharted territory. In this paper, we introduce a novel method to discern ChatGPT-generated code. We employ targeted masking perturbation, emphasizing code sections with high perplexity. Fine-tuned CodeBERT is utilized to replace these masked sections, generating subtly perturbed samples. Our scoring system amalgamates overall perplexity, variations in code line perplexity, and burstiness. In this scoring scheme, a higher rank for the original code suggests it's more likely to be ChatGPT-generated. The underlying principle is that code generated by models typically exhibits consistent, low perplexity and reduced burstiness, with its ranking remaining relatively stable even after subtle modifications. In contrast, human-written code, when perturbed, is more likely to produce samples that the model prefers. Our approach significantly outperforms current detectors, especially against OpenAI's text-davinci-003 model, with the average AUC rising from 0.56 (GPTZero baseline) to 0.87.

Introduction

Large language models, like ChatGPT from OpenAI, can now generate code autonomously. This is revolutionizing software development but also poses a challenge for education. Students might use these models to complete coding assignments without doing the work themselves. This misuse undermines the purpose of programming education. Most text detectors today are designed to detect model-generated text, not code. The study DetectGPT (Mitchell et al. 2023) presented an interesting finding: making small changes to model-generated text typically results in lower log probabilities of the model of interest than the original text, while changes to human-written text can increase or decrease these probabilities. This means that model-produced tokens usually sit at the peak of the log probability curve. The concept of code naturalness implies that a similar behavior should manifest within code contexts. We tested the idea that small modifications can clearly separate ChatGPT-generated code from human-written code. Our experiments found that minor changes to ChatGPT-produced code often make it more perplexing for the model than the original. However, for human-written code, such changes can have mixed results. We also suggest that code generated from large language models shows steadier perplexity line-by-line than human code, and our results support this.

We harness these two observations from the code context to construct our detector. Our strategy seeks to discern whether a given piece of code is ChatGPT-generated by perturbation and scoring. First, a perturbation mechanism applies a mask modeling task to slightly modify segments of code that exhibit higher perplexity (PPL). Second, a scoring mechanism is influenced by three metrics: perplexity of the code, standard deviation of perplexity across lines of code, and the code's burstiness. A lower score indicates a greater probability that the code generated from model. This process is shown in Figure 1. The foundational premise of our methodology is that ChatGPT-generated code inherently possesses a low score, gauged by the combined metrics of overall perplexity, standard deviation of perplexity across code lines, and burstiness. Consequently, post-perturbation, it becomes challenging to generate samples that score lower than the original ChatGPT-generated code. In contrast, human-authored code might be perceived as more ambiguous by the model, so minor tweaks might yield an "optimized" version that bears a score lower than the original code.

Approach

Dataset Description

Our CGCode (ChatGPT-generated code) dataset, based on IBM's CodeNet, aims to mimic student submissions for programming tasks. We focused on six primary programming languages: C, C++, C#, Java, JavaScript, and Python. After ensuring each code met criteria for quality and length and removing duplicates and comments, we used 80% of the data to fine-tune and validate CodeBERT, setting aside 10% each for validation and testing. We enhanced the test subset, and leveraged OpenAI's text-davinci-003 for text-to-code generation and code translation tasks for 400 problems in each test subset. The test set now contains 5,214

	C			C++			C#			Java			JavaScript			Python		
Detectors	AUC	FPR	FNR	AUC	FPR	FNR	AUC	FPR	FNR	AUC	FPR	FNR	AUC	FPR	FNR	AUC	FPR	FNR
GPT2-Detector	0.64	0.83	0.03	0.68	0.84	0.05	0.46	0.92	0.13	0.44	0.91	0.11	0.39	0.90	0.26	0.38	0.89	0.25
DetectGPT	0.56	0.00	1.00	0.42	0.00	1.00	0.49	0.00	1.00	0.43	0.00	1.00	0.51	0.00	1.00	0.49	0.00	1.00
RoBERTa-QA	0.68	0.00	1.00	0.53	0.00	1.00	0.48	0.00	1.00	0.64	0.00	1.00	0.52	0.00	1.00	0.60	0.00	1.00
Writer	0.78	0.13	0.91	0.62	0.15	0.98	0.52	0.13	0.91	0.54	0.01	0.96	0.56	0.10	0.89	0.51	0.08	0.97
GPTZero	0.90	0.06	0.20	0.83	0.20	0.68	0.29	0.18	0.96	0.28	0.18	0.88	0.41	0.08	0.90	0.59	0.00	1.00
CGCode Detector	**0.95**	0.16	0.08	**0.88**	0.13	0.02	**0.86**	0.12	0.07	**0.82**	0.15	0.05	**0.81**	0.21	0.15	**0.92**	0.14	0.03

Table 1: Performance of Different Detectors Across Six Programming Languages

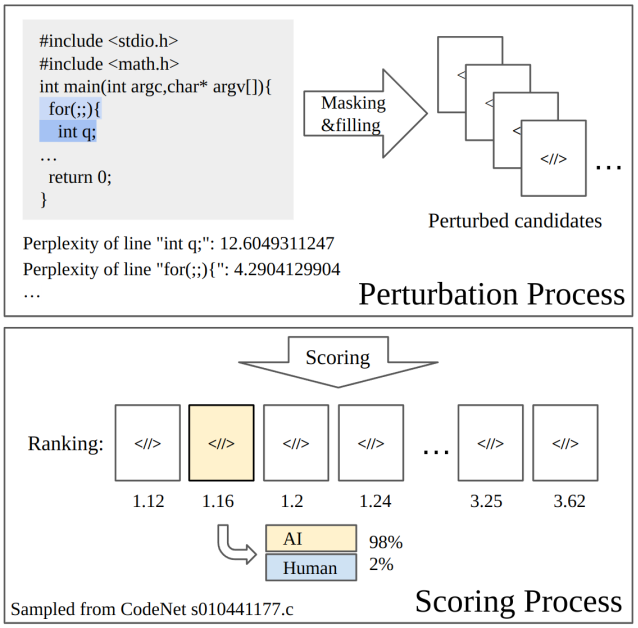

Figure 1: Illustration of the perturbation and scoring procedure. Weights are assigned to code segments based on line-level perplexity, higher weight means more allocated masks, followed by mask-filling task for slight modifications. A lower score suggests a higher likelihood of the code being generated by ChatGPT.

Chatpt-generated codes, balanced with an equivalent number of human-written ones, matched by language and problem type.

Perturbation and Scoring Process

Masking Unlike the random masking strategy of DetectGPT on text, our method for code uses a measured strategy. We utilize Perplexity (PPL) to gauge the complexity of each line of code, and apply masks based on this. Lines of code with high PPL values receive more masks. This method proves to be more efficient than random masking, leading to less required samples.

Mask-filling After fine-tuning CodeBERT across a spectrum of programming languages, it's harnessed to fill our code masks. We opt for Nucleus Sampling to obtain a diverse set of token suggestions, ensuring the creation of a rich variety of modified samples.

Scoring Our evaluation of code integrates three metrics: PPL, PPL variation across code lines, and code burstiness. The scores of original and altered code are juxtaposed. ChatGPT-generated code often attains better scores due to its intrinsic coherence. Altered versions of ChatGPT-generated code seldom score lower, while human-written code might see lower scores with minimal modifications.

Experiment and Results

To evaluate the efficacy of our Detector, we benchmark its performance against five prominent open-source and commercial text detectors: GPT2-Detector, DetectGPT, RoBERTa-QA, GPTZero, and The Writer AI Detector. Our comparisons are conducted on the CGCode dataset. Within the CGCode Detector framework, we employ CodeBERT as the mask-filling model and text-davinci-003 as the primary scoring model. For those detectors giving probability, we choose its best performance threshold. For detectors having requirement of input length, we truncate and use the prior tokens as input. Results are shown in Table 1. CGCode Detector has relatively high AUC on all programming languages, as well as low FPR and FNR.

Conclusion

In conclusion, we present the CGCode Detector, an innovative tool adept at pinpointing ChatGPT-generated code assignments using perplexity analysis and targeted perturbations. Building on the foundation of DetectGPT, we have addressed the gap in ChatGPT-generated code detection by enhancing current zero-shot detection method and developing a specialized detector for ChatGPT-generated code. Empirical results show that our detector surpasses both leading open-source and commercial alternatives in the code domain. Due to its flexibility, our method can also be extended to other code generation models.

References

Mitchell, E.; Lee, Y.; Khazatsky, A.; Manning, C. D.; and Finn, C. 2023. Detectgpt: Zero-shot machine-generated text detection using probability curvature. *arXiv preprint arXiv:2301.11305*.