

Assessing AI Detectors in Identifying AI-Generated Code: Implications for Education

ABSTRACT

Educators are increasingly concerned about the usage of Large Language Models (LLMs) such as ChatGPT in programming education, particularly regarding the potential exploitation of imperfections in Artificial Intelligence Generated Content (AIGC) Detectors for academic misconduct.

In this paper, we present an empirical study where the LLM is examined for its attempts to bypass detection by AIGC Detectors. This is achieved by generating code in response to a given question using different variants. We collected a dataset comprising 5,069 samples, with each sample consisting of a textual description of a coding problem and its corresponding human-written Python solution codes. These samples were obtained from various sources, including 80 from Quescol, 3,264 from Kaggle, and 1,725 from LeetCode. From the dataset, we created 13 sets of code problem variant prompts, which were used to instruct ChatGPT to generate the outputs. Subsequently, we assessed the performance of five AIGC detectors. Our results demonstrate that existing AIGC Detectors perform poorly in distinguishing between human-written code and AI-generated code.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**.

ACM Reference Format:

. 2023. Assessing AI Detectors in Identifying AI-Generated Code: Implications for Education. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

In recent years, an increase in capability has been observed in the development of LLMs, enabling them to generate high-quality, coherent paragraphs, answer questions, and even produce human-like code. LLM are designed to understand and generate human text, and they are trained using vast amounts of data scraped from the internet. Most of these LLMs, developed by major corporations, are generally made accessible to the public, allowing anyone with Internet access to utilize them. This accessibility enables individuals to swiftly obtain valuable and reference-worthy answers.

At the educational level, this accessibility can enhance learning and research efficiency while potentially influencing educational

assessment and evaluation [20]. The use of AI-based tools and resources has gradually led to a situation where students increasingly rely on these tools to quickly access answers and information. This growing dependence on AI-driven solutions has a noticeable impact on academic dishonesty.

As a consequence, educators find themselves compelled to utilize AIGC Detectors to ascertain whether students are involved in academic dishonesty. While existing AIGC Detectors have proven their proficiency in identifying AI-generated text, their effectiveness in recognizing AI-generated code remains uncertain due to the intricate nature of programming code. This discrepancy can lead to disparities in the evaluation of students' academic submissions, potentially resulting in unfair grading.

The paper aims to conduct an empirical study to evaluate the performance of different AIGC Detectors in detecting AIGC across diverse contextual and syntactical variations.

Overall, the main contributions of our paper are summarized as follows

- We conducted a comprehensive empirical study to assess the performance of five AIGC detectors using 13 variant prompts. To the best of our knowledge, this is the first study specifically evaluating the performance of different AIGC detectors on AIGC generated with various question prompts.
- We constructed 13 large datasets, each containing 5,069 samples representing specific code problem prompt variants. Each dataset comprises code problems, human-generated code for that problem, and AI-generated code for the same problem.

2 BACKGROUND AND MOTIVATIONS

In this section, we discuss some of the background related to our work, particularly in the domain of software engineering education and the use of generative AI technologies to help in teaching and learning processes.

2.1 Impact of AIGC on SE and CS Education

Software Engineering (SE) and Computer Science (CS) education form the bedrock of technological progress. These fields equip students with essential skills, such as problem-solving, logical thinking, and creativity. However, the cornerstone of SE and CS education is programming. Programming is not just a skill; it embodies the essence of SE and CS education. It empowers students to create innovative software solutions, analyze complex problems, and make significant contributions to the digital world.

The emergence of generative AI and the subsequent proliferation of AIGC have brought about a transformative era in education. AIGC represents a paradigm shift where automated content creation is revolutionizing learning experiences for both educators and learners. This shift, while revolutionary, introduces a complex interplay of challenges and opportunities in the realm of education. AIGC can personalize and enhance learning, but it also poses

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

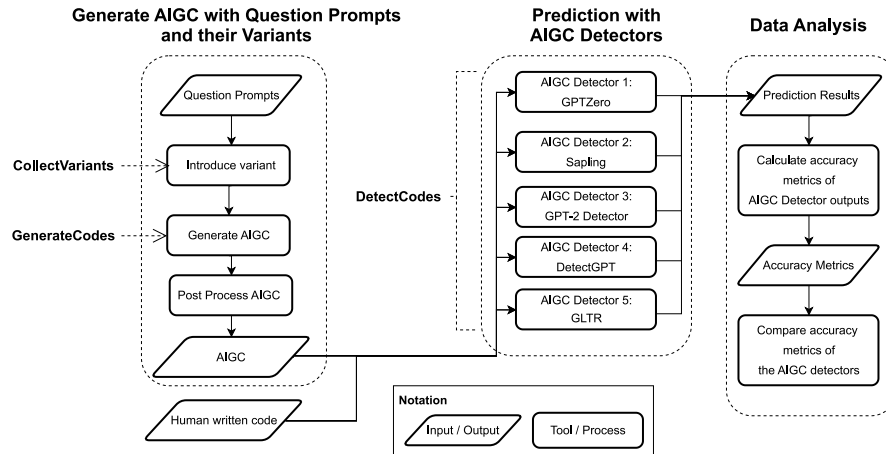


Figure 1: Workflow of the AIGC and their variants' generation process, AIGC prediction with AIGC Detectors, and comparative analysis of AIGC Detector outputs with accuracy metrics.

questions about the authenticity of content and the methods of assessment in the digital age [4].

Generative Pre-trained Transformers (GPTs) [3] have revolutionized natural language processing (NLP). It started with GPT-1 in 2018 (117 million parameters), laying the foundation. GPT-2 arrived in 2019 (1.5 billion parameters), improving text generation. GPT-3.5, part of OpenAI's API, introduced advanced features like editing and inserting. GPT-4 added multimodal capabilities, raising ethical concerns. Each iteration pushes AI language models forward with transformative capabilities.

The widespread adoption of AIGC has led to a growing need for reliable detection mechanisms. One notable development in this area is the AIGC Detector is detailed in the work by Wang et al. [23]. These detectors have demonstrated impressive accuracy in identifying AI-generated content. However, understanding the implications of the accuracy of AIGC Detector within educational settings, particularly in CS and SE learning environments, is essential.

In an era where these detectors ensure the authenticity of educational content, it is imperative to examine their effectiveness and limitations comprehensively. Relying solely on their accuracy might create a false sense of security, potentially allowing students to deceive these detectors. This raises questions about the robustness of educational assessments and the integrity of the learning process.

An intriguing aspect of this inquiry lies in potential scenarios where students might outsmart AIGC Detectors. For example, in specific situations, students may find ways to manipulate the system, deceiving even sophisticated AIs like ChatGPT. This raises concerns about the reliability of automated evaluations, especially if educators heavily depend on a single detector. Understanding these vulnerabilities is vital for ensuring the authenticity of educational assessments in the digital age.

In summary, this research is motivated by the transformative impact of AIGC on SE and CS education. Our study aims to significantly contribute by unraveling the complexities surrounding AIGC Detectors, exploring their accuracy, vulnerabilities, and implications for educational practices. By shedding light on these critical

aspects, our work strives to enhance the understanding of AIGC Detection within the context of CS and SE education. Ultimately, our research seeks to ensure the integrity of educational assessments, thereby fostering a secure and genuine learning environment for students and educators alike.

2.2 Selection of AIGC Detectors

The integration of AIGC Detection in SE education significantly enhances academic integrity and ensures the authenticity of educational content. Educators can leverage these detectors to identify instances of plagiarism, unauthorized use of AI-generated code, and academic dishonesty among students, thereby fostering fairness and promoting originality within SE coursework. Moreover, AIGC Detectors play a pivotal role in verifying the authenticity of educational materials, allowing educators to maintain the credibility of resources utilized in SE courses. This technological integration not only strengthens the overall educational ecosystem but also stimulates meaningful discussions among students about responsible AI usage, emphasizing the importance of ethical practices in SE education.

Five AIGC Detectors are instrumental in achieving these objectives. GPTZero [1], Sapling [2], GPT-2 Detector [21], DetectGPT [5, 16], and GLTR [11, 12, 18] offer educators reliable tools to preserve academic integrity and encourage ethical AI usage. However, it is crucial for educators to thoroughly evaluate these tools and consider their limitations to ensure their effective use in SE education.

3 EMPIRICAL STUDY DESIGN AND METHODOLOGY

This section discusses the methodology and process of our empirical study. Figure 1 offers an overview of the empirical study process, highlighting the key phase of data collection, prediction, and analysis. We have made the replication package publicly available.¹

The data collection phase is essential for ensuring the integrity and robustness of our study. We gathered coding problems from

¹https://figshare.com/articles/dataset/Replication_Package/24298036

various sources and introduced different variants based on the collected coding problems. Then, by utilizing OpenAI API service,² we collected AI-generated code and performed manual validation to ensure the correctness of the dataset. Moving on to the data prediction phase, by using AI-generated code and human-written code as input, we employed selected AIGC Detectors to perform AIGC Detection and collect the prediction results. In the data analysis phase, using the compiled prediction results, we evaluated the efficacy of each AIGC Detector via predetermined performance metrics and utilised the results analysed to answer our research questions.

3.1 Data Collection

We gathered fundamental code problems along with corresponding solutions code from various online sources, such as using the existing datasets from Kaggle³ and web scraping technique to crawl data from Quescol.⁴ For the AIGC dataset, we generated different variants of AI-generated code to explore different scenarios and find out situations that might be possible for students to fool AIGC Detectors. The details about the different variants are discussed in Section 3.3.

For the compilation of the AIGC dataset, we developed an automation script leveraging the API service from OpenAI.² The script will use the collected code problems as inputs, and then prompt ChatGPT to generate the outputs, subsequently label and store them as AI-generated code. The script was executed 13 times to generate different code variants, such as removing stop words or adding dead code, ensuring that a dataset of initial size 70,966 was collected for thorough analysis.

The coding problems and human-written code are obtained from three primary sources:

- Python Coding Question [26]: This source from Quescol contributed 80 samples to our dataset, which includes some common interview questions for Python.
- Coding Problems and Solution Python Code [25]: This source from Kaggle contributes approximately 3,264 samples to our dataset. It encompasses a wide range of Python code solutions to various coding problems.
- LeetCode Solutions and Content KPIs [24]: This source from Kaggle contributes around 1,725 samples to our dataset. It includes solutions to coding problems from the popular online coding platform, LeetCode, along with additional content and key performance indicators.

The collected dataset comprises 5,069 samples where each sample consists of a textual description of a coding problem and the corresponding solution code that were labeled as human-written code. The dataset covers diverse programming concepts, algorithms, and coding challenges. Based on collected code problems, we have created 13 variants of AIGC content.

3.2 Workflow Explanation

To ensure the replicability and clarity of our empirical study procedure, we have provided our procedure in a systematic, step-by-step

²<https://platform.openai.com/>

³<https://www.kaggle.com/>

⁴<https://quescol.com/>

Algorithm 1 DynCodeMetrics: Dynamic Prompt Variation and Code Detection with Metric Calculation

Input:

- P : Collection of Prompt Dataset
- $AIGCD$: Artificial Intelligence Generated Content Detector
- $VarPrompt$: Variant Modification

Output: R : Metric Results

```

1: procedure DYNCODEMETRICS( $P, AIGCD, VarPrompt$ )
2:    $V \leftarrow \{\}$  // Collection of various prompts
3:    $D \leftarrow \{\}$  // Collection Results of 1(Human) and 0(AI)
4:    $C \leftarrow \{\}$  // Collection of Human and AI-generated Code
5:    $V \leftarrow \text{COLLECTVARIANTS}(P, VarPrompt, V)$ 
6:    $\text{GENERATECODES}(V, C)$ 
7:    $C' \leftarrow \text{Post Process on } C$  // Post-processing on generated codes
8:    $\text{DETECTCODES}(AIGCD, C', D)$ 
9:    $R \leftarrow \text{Calculate accuracy, precision, recall, TPR, FPR based on } D$ 
10:  return  $R$  // Result of Metrics
11: end procedure
```

Algorithm 2 CollectVariants: Generate Different Variation of Prompt

Input:

- P : Collection of Prompt Dataset
- $VarPrompt$: Variant Modification
- V : Collection of various prompts

Output: V : Collection of Various Prompts

```

1: procedure COLLECTVARIANTS( $P, VarPrompt, V$ )
2:   for  $p \in P$  do
3:      $v \leftarrow \text{Variant Modification on } p$ 
4:     Include  $v$  in collection of various prompts  $V$ 
5:   end for
6:   return  $V$  // Collection of various prompts
7: end procedure
```

Algorithm 3 GenerateCodes: Collecting AI-Generated Code

Input:

- V : Collection of various prompts
- C : Data Collection of Human and AI-generated Code

Output: C : Modified Data Collection

```

1: procedure GENERATECODES( $V, C$ )
2:   for  $v \in V$  do
3:      $C \leftarrow \{\}$  // Clear the Data Collection
4:     for  $i \leftarrow 1$  to  $N$  do // Generate AI Code
5:        $c \leftarrow \text{AI Code Generation based on } v$ 
6:       Include  $c$  in Data Collection  $C$ 
7:     end for
8:   end for
9:   return  $C$  // Code Data Collection
10: end procedure
```

algorithm format to eliminate misinterpretation of our procedure in Figure 1. As shown in Algorithms 1 to 4, four procedures are included on how variant dataset is collected and metric is calculated:

- DynCodeMetrics: Algorithm 1 introduces the DynCodeMetrics procedure, which serves as a pivotal algorithm responsible for coordinating an array of sub-algorithms. Its primary purpose is to

Algorithm 4 DetectCodes: Code Detection Using AIGC Detector**Input:**

- *AIGCD*: Artificial Intelligence Generated Content Detector
- *C'*: Processed Data Collection of Human and AI-generated Code
- *D*: Collection Results of 1(Human) and 0(AI)

Output: *D*: Detection Result Collection

```

1: procedure DETECTCODES(AIGCD, C', D)
2:   for aigcd  $\in$  AIGCD do
3:     D  $\leftarrow$  {} // Result Collection of 1 (Human) and 0 (AI)
4:     for c  $\in$  C' do
5:       d  $\leftarrow$  Code Detection using AIGCD
6:       Include d in Result Collection D
7:     end for
8:   end for
9:   return D // Detection Result Collection
10: end procedure

```

streamline the entire workflow, encompassing the generation of diverse dynamic prompts, post-processing of AI-generated code, code detection, and the subsequent computation of pertinent metrics. This comprehensive procedure ensures the acquisition of AI-generated code, its identification, and the subsequent metric assessment, contributing to a comprehensive and insightful analysis. It takes three inputs: *P* (Collection of Prompts Datasets), *AIGCD* (AIGC Detector), and *VarPrompt* (Variant Modification Method). It initializes three collections: *V* for collection of various prompts, *D* for AIGC Detector's results and *C* for collection of human and AI-generated codes. The collection *V* (collection of various prompts) is created to store the outcomes of various variants prompts. The collection *D* (collection of AIGC Detector's results) is designed to store the outcomes of the AIGC Detector, with the convention that it uses '1' to indicate code written by humans and '0' to signify AI-generated code. The collection *C* (Collection of Human and AI-generated codes) is designed to store the human and AI-generated codes that based on each prompt.

First, it calls the **CollectVariants** procedure which takes three inputs: *P* (Collection of Prompts Datasets), *VarPrompt* (Variant Modification Method) and *V* (Collection of various prompts), to modify the prompts by applying the variant modification method and store the outcome in *V* (collection of various prompts). Next, it calls the **GenerateCodes** procedure which takes two inputs: *V* (Collection of various prompts) and *C* (Human and AI-generated codes), to generate AI-generated codes based on the modified prompts. After generating the AI-generated codes, it performs post-processing on the *C* (Human and AI-generated codes), which involves applying cleaning steps to remove unwanted content such as ``python {AI-generated code}``. Besides, AI-generated codes that is invalid are removed from the dataset for the correctness of the dataset. For AIGC Detector, such as GPT-2 Detector, with maximum token size, the data are separated into multiple parts for AIGC Detector to evaluate; the final prediction is the average probability of all separated parts. It returns the *C'* (Postprocessed Human and AI-generated codes). The next step is to detect codes using the AIGC Detector by calling the **DetectCodes** procedure that takes three inputs,

namely *AIGCD* (AIGC Detector), *C'* (Postprocessed Human and AI-generated codes) and *D* (Collection of AIGC Detector's results).

Finally, it calculates various metrics, such as accuracy, precision, true positive rate (TPR), false positive rate (FPR), true negative rate (TNR), and false negative rate (FNR) based on the detection results. These metrics provide insights into the performance of the AIGC Detector. *R* (Calculated metrics) are returned as the output.

- **CollectVariants**: From Algorithm 2, the **CollectVariants** procedure takes three inputs: *P* (Collection of Prompts Datasets), *VarPrompt* (Variant Modification Method), and empty collection *V* (collection of various prompts). It iterates over each coding problem in the dataset and applies the variant modification method to modify the prompt. The modified prompts are then included in the *V* (collection of various prompts) and being returned.
- **GenerateCodes**: From Algorithm 3, the **GenerateCodes** procedure takes *V* (collection of various prompts) and empty collection *C* (Collection of Human and AI-generated codes) as input. It iterates over each variant prompt and generates AI codes based on the variant prompt. The procedure generates the specified number of AI codes for each variant prompt and includes them in the data collection. The data collection *C* (Collection of Human and AI-generated codes) contains the code from humans and AI are returned.
- **DetectCodes**: From Algorithm 4, the **DetectCodes** procedure takes the *AIGCD* (AIGC Detector), *C'* (Postprocessed Human and AI-generated codes), and an empty *D* collection (collection of AIGC Detector's results) for storing the detection results as input. It iterates over each AIGC Detector and performs classification on each code in the post-processed data using the corresponding model. The detection results are then included in the result *D* collection (collection of AIGC Detector's results).

Algorithm 1 combines Algorithms 2 to 4 to dynamically vary prompts, generate AI codes, perform code detection, and calculate relevant metrics. The resulting metrics can be used to evaluate the effectiveness and accuracy of the AIGC Detector.

3.3 Data Variations

Table 1 displays the dataset sizes used for each AIGC Detector across various variants. The sizes of datasets for Variants 2 and 3 (we will discuss the specific details of each variant in the following paragraph) are smaller across all AIGC Detectors due to the removal of data resulting from invalid outputs by ChatGPT. Specifically, when applying the Sapling detector on Variant 2, the dataset size is further reduced as our initial approach has a minimum word requirement for code detection (which will be discussed in detail in the following paragraph). Similarly, the DetectGPT detector excludes data with fewer than 100 words due to its minimum word requirement, leading to a reduced dataset size.

In the process of preparing our dataset, we introduced 13 variations of AI-generated code. These variations were achieved by altering the prompts provided to ChatGPT or the solution code received from ChatGPT. Our aim was to find out the limitations and performance of each AIGC Detector under different variants

Table 1: Post-processed dataset size for each Detector across variant

Detector	Variant												
	1	2	3	4	5	6	7	8	9	10	11	12	13
GLTR	5,069	5,065*	5,064*	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,068*
Sapling	5,069	5,055*	5,064*	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,068*
GPT Zero	5,069	5,065*	5,064*	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,068*
GPT-2 Detector	5,069	5,065*	5,064*	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,069	5,068*
DetectGPT	4,016*	4,674*	4,509*	4,097*	4,785*	4,702*	4,888*	3,932*	3,997*	3,952*	4,211*	3,925*	4,785*

Note: * indicates a deviation from the initial dataset size of 5,069.

of programming code. Detailed descriptions of each variation are available on Figshare.⁵

1. Without modification: We have adapted the prompt according to an example provided by CEO of OpenAI, Sam Altman.⁶ We prompted ChatGPT with original code problems without any modification made to the problem statement and added some preliminary conditions [6] in the prompt before generating code with ChatGPT. This variant aims to mimic the typical approach users take when trying to resolve coding problems by directly prompting the coding problems with some preliminary conditions.

2. Removal of Stopwords: We performed preprocessing on the coding problems by removing common stopwords to reduce noise and improve the model's focus on essential keywords. We utilized the Natural Language Toolkit (NLTK) library to identify and eliminate stopwords specific to the coding questions. This variant aims to simulate a scenario where users modify questions and eliminate stopwords to achieve a better output.

3. Ask to Mimic Human: We included an additional sentence to guide the model's behaviour and encourage it to generate more contextually appropriate code snippets, trying to tell ChatGPT to mimic human responses. For every prompt given to ChatGPT, we appended the "Please mimic a human response." sentence at the end of the prompt. This explicit instruction aimed to guide the model's behaviour more closely with human responses, improving its ability to generate code that mimics code written by humans.

4. Solution Without Comment: We instructed ChatGPT not to include any comment in the generated solution. This variant aims to simulate typical cases where students violate good programming practices and fail to include comments in the solution code.

5. Assertion Test Code: We instructed ChatGPT to include assertion test code in the output. This aims to assess the AIGC Detectors' ability to identify code that not only appears syntactically valid but also includes test assertions which is one of the common approaches used to verify the correctness of a program's functionality.

6. Solution with Test Case: We instructed ChatGPT to include test cases in the output. This aims to assess the AIGC Detectors' ability to identify code that not only appears syntactically valid but also includes test cases which is one of the common approaches used to verify the correctness of a program's functionality.

7. Unittest Test Case: We instructed ChatGPT to include unittest test case in the output. This aims to assess the AIGC Detectors' ability to identify code that not only appears syntactically valid

but also includes unittest test cases which is one of the common approaches used to verify the correctness of a program's functionality.

8. Replace variable names: Based on the AI-generated code in variant 1, we developed a Python script using Abstract Syntax Trees (AST) library to replace all variable names in the code with single-character letters from a to z only. This aims to simulate cases where students violate the basic programming rules for naming variables using meaningless words or letters in the code.

9. Replace function names: Based on the AI-generated code in variant 1, we have developed a Python script using AST library to replace all function names in the code with single-character letters from a to z only. This aims to simulate cases where students violate the basic programming rules for naming functions using meaningless words or letters in the code.

10. Replace variable and function names: Based on the AI-generated code in variant 1, we have developed a Python script using AST library to replace all variable names and function names in the code with single-character letters from a to z only. This aims to simulate extreme cases where programmers violate the basic programming rules for naming variables and functions using meaningless words or letters in the code.

11. Long method: Long method, as a term associated with code smells, often indicates weaknesses in design that increase the risk of bugs in the development process. We asked ChatGPT to provide a longer output in function block format. This variation aims to simulate cases where long method code smell exists in the solution code which reduces the readability and maintainability of the code.

12. Short method: In contrast to the long method variant, we asked ChatGPT to provide a shorter output in function block format. This variation aims to simulate cases where best practices in programming are followed by keeping the method concise and focusing on a single functionality only.

13. Adding 5 snippets of dead code: We inject 5 snippets of dead code in the output which acts as some noise for the AIGC Detector. We aim to explore whether dead code in the solution code affects the performance of the AIGC Detector. This variant simulates a scenario where useless code is included in the solution code due to oversight during development.

⁵https://figshare.com/articles/dataset/Variant_Description/24265018

⁶<https://twitter.com/sama/status/1682826943312326659>

Table 2: Confusion Matrix used in this study

Predicted Value	Actual Value	
	HUMAN (1)	AI (0)
HUMAN (1)	True Positive (TP)	False Positive (FP)
AI (0)	False Negative (FN)	True Negative (TN)

4 EXPERIMENTAL DESIGN AND RESEARCH QUESTION

4.1 Experimental setup

In this section, we discuss the research questions and the experimental setup.

- (1) **RQ1: How accurate are existing AIGC Detectors at detecting AI-generated code?**
- (2) **RQ2: What are the limitations of existing AIGC Detectors when it comes to detecting AI-generated code?**

Our objectives of RQ1 and RQ2 is to assess the performance of the chosen AIGC Detectors in identifying AI-generated code and investigate the limitations of AIGC Detectors in identifying AI-generated code. Both research questions share a similar experimental setup.

- (1) **Data Evaluation:** Evaluation of the AIGC Detectors' performance is based on the 13 variants of the dataset we have meticulously collected.
- (2) **Experimental Configuration:** For the selected AIGC Detectors, we have established a discriminative threshold of 0.5. This threshold serves as a criterion to distinguish AI-generated code samples. Specifically, if the output probability surpasses the 0.5 threshold, it signifies the presence of AI-generated content within the input samples.
- (3) **Performance Assessment Metrics:** The performance of the detectors is measured using a range of metrics such as accuracy, precision, true positive rate (TPR), false positive rate (FPR), true negative rate (TNR) and false negative rate (FNR). These metrics provide a comprehensive understanding of how effectively the detectors identify AI-generated code.

4.2 Metrics Evaluation

For this empirical study, we have selected a few metrics [14, 22] to evaluate the performance of selected AIGC Detectors. Table 2 shows the confusion matrix used in the paper where 1 indicates human-written code, and 0 indicates AI-generated code.

TPR/Recall: True Positive Rate, also known as Recall, calculated as $TPR/Recall = \frac{TP}{TP+FN}$, where TP is the number of human-written codes correctly labelled as human-written, FN is the number of human-written codes incorrectly labelled as AI-generated and TP+FN represents the total number of human-written codes.

FNR: False Negative Rate, calculated as $FNR = \frac{FN}{TP+FN}$, where FN is the number of human-written codes incorrectly labelled as AI-generated, TP is the number of human-written codes correctly labelled as human-written and TP+FN represents the total number of human-written codes.

TNR: True Negative Rate, calculated as $TNR = \frac{TN}{TN+FP}$, where TN is the number of AI-generated codes correctly labelled as AI-generated, FP is the number of AI-generated codes incorrectly labelled as human-written and TN+FP represents the total number of AI-generated codes.

FPR: False Positive Rate, calculated as $FPR = \frac{FP}{TN+FP}$, where FP is the number of AI-generated codes incorrectly labelled as human-written, TN is the number of AI-generated codes correctly labelled as AI-generated and TN+FP represents the total number of AI-generated codes.

Accuracy (ACC): Calculated as $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$, where TP is the number of human-written codes correctly labelled as human-written, TN is the number of AI-generated codes correctly labelled as AI-generated and TP+TN+FP+FN is the total number of codes.

Precision: Calculated as $Precision = \frac{TP}{TP+FP}$, where TP is the number of human-written codes correctly labelled as human-written, FP is the number of AI-generated codes incorrectly labelled as human-written and TP+FP is the total number of codes labelled as human-written.

F1 Score: Calculated as $F_1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$, which represents the harmonic mean of precision and recall.

AUC: Area Under Curve, which is used to evaluate how much the model is capable of distinguishing between classes. A higher AUC score means that it has a better capability to distinguish between positive and negative classes. When the AUC score is around 0.5, it indicates the model performs random choice when predicting the samples.

5 RESULTS

In this section, we showcase our experimental results and provide an analysis to address each research question. We have uploaded the full set of results on Figshare.⁷

5.1 RQ1: How accurate are existing AIGC Detectors at detecting AI-generated code?

Table 3 shows the performance of GPT-2 Detector and GPTZero, where both detectors exhibit poor performance when compared with baseline variant 1. Their Accuracy (ACC) results hover around 0.5, suggesting a lack of effectiveness in distinguishing AI-generated code from human-written code. They tend to classify a significant portion of input code as human-generated rather than machine-generated. This performance issue could stem from their primary training on natural language data or potential overfitting to natural language.

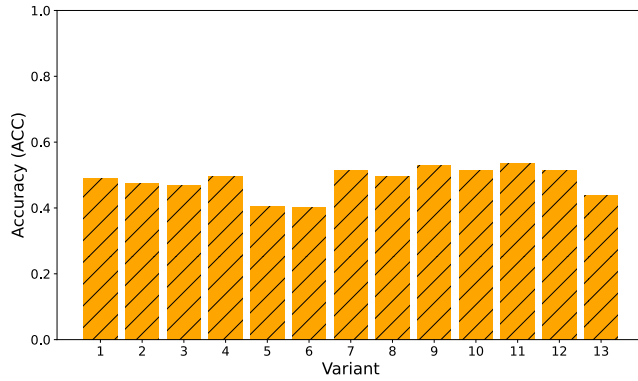
In the assessment of AIGC Detectors, both the GPT-2 Detector and GPTZero exhibited challenges in effectively distinguishing between human-written and AI-generated code across all evaluated variants. In Table 3, their performance was characterized by high TPR and low TNR while having an ACC of around 0.5, indicating a tendency to classify both human-written and AI-generated code as human-written code.

⁷https://figshare.com/articles/dataset/Variants_Result/24265015

Table 3: Accuracy, TPR, and TNR of All 5 AIGC Detectors

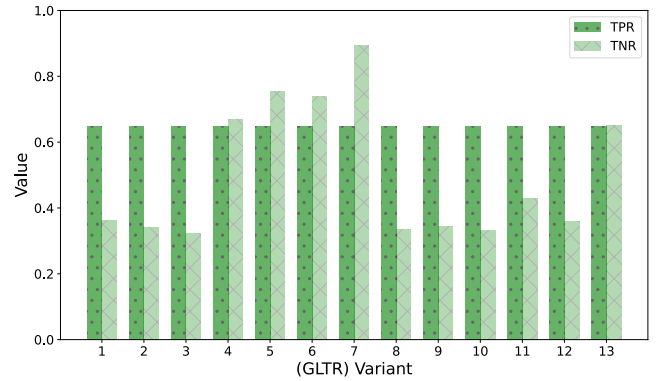
Detector	Metric	Variant												
		1	2	3	4	5	6	7	8	9	10	11	12	13
GPT Zero	ACC	0.4971	0.4988	0.4986	0.4966	0.4969	0.4976	0.4989	0.5108	0.4972	0.4973	0.4970	0.4965	0.5824
	TPR	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927	0.9927
	TNR	0.0016	0.0049	0.0045	0.0006	0.0012	0.0026	0.0051	0.0288	0.0018	0.0020	0.0014	0.0004	0.1721
GPT-2 Detector	ACC	0.5043	0.5005	0.4961	0.5013	0.4716	0.4788	0.4828	0.5134	0.5100	0.5252	0.4911	0.4958	0.4922
	TPR	0.9128	0.9127	0.9127	0.9128	0.9128	0.9128	0.9128	0.9128	0.9128	0.9128	0.9128	0.9128	0.9128
	TNR	0.0959	0.0883	0.0796	0.0898	0.0304	0.0448	0.0529	0.1140	0.1071	0.1375	0.0694	0.0787	0.0716
DetectGPT	ACC	0.4893	0.4742	0.4685	0.4941	0.4055	0.4014	0.5132	0.4943	0.5278	0.5125	0.5354	0.5153	0.4373
	TPR	0.2756	0.2993	0.2978	0.2824	0.3083	0.3056	0.3124	0.2693	0.2737	0.2677	0.2769	0.2662	0.3093
	TNR	0.7029	0.6491	0.6392	0.7059	0.5028	0.4972	0.7140	0.7192	0.7818	0.7573	0.7939	0.7643	0.5653
GLTR	ACC	0.5040	0.4936	0.4841	0.6569	0.6999	0.6920	0.7693	0.4908	0.4952	0.4881	0.5375	0.5020	0.6478
	TPR	0.6461	0.6464	0.6461	0.6461	0.6461	0.6461	0.6461	0.6461	0.6461	0.6461	0.6461	0.6461	0.6460
	TNR	0.3620	0.3408	0.3221	0.6678	0.7538	0.7378	0.8925	0.3356	0.3442	0.3300	0.4289	0.3579	0.6496
Sapling	ACC	0.6056	0.5961	0.6031	0.6048	0.5425	0.5828	0.6083	0.6630	0.6258	0.6811	0.6187	0.6059	0.6528
	TPR	0.4780	0.4797	0.4783	0.4780	0.4780	0.4780	0.4780	0.4780	0.4780	0.4780	0.4780	0.4780	0.4781
	TNR	0.7333	0.7126	0.7279	0.7315	0.6070	0.6875	0.7386	0.8479	0.7735	0.8842	0.7593	0.7339	0.8275

Note: *green* indicates an increase compared to Variant 1, while *red* indicates a decrease

**Figure 2: Accuracy Performance for DetectGPT**

DetectGPT, akin to the aforementioned AIGC Detectors, showcases an ACC near 0.5 in Figure 2, indicating its struggle in distinguishing AI-generated from human-written code. Notably, DetectGPT tends to misclassify the majority of human-generated code as AI-generated. However, a more positive aspect emerges when we compare DetectGPT to the baseline (Variant 1) and other variants like Function Name (Variant 9), Variable and Function Name (Variant 10), Long Method (Variant 11), and Short Method (Variant 12). DetectGPT demonstrates an improvement in TNR ranging from 5% to 9%. This improvement signifies its competence in accurately identifying AIGC.

The ACC performance of GLTR (Giant Language Model Test Room) exhibits variability across different AIGC variants. Approximately half of the variants surpass the 0.6 Accuracy (ACC) threshold, while the remainder approach a more modest 0.5. In our evaluation of GLTR and Figure 3, we conducted a comparative analysis involving the baseline Variant 1 and several other variants, including "No Comment (Variant 4)," "Assertion (Variant 5)," "Test Case (Variant 6)," "Unittest Test Case (Variant 7)," and "Dead Code (Variant

**Figure 3: TPR and TNR Performance for GLTR**

13)." The aim was to assess the system's performance in detecting GPT-generated code as AI-generated content.

The results uncovered a significant enhancement in GLTR's ability to identify AI-generated code when compared to the baseline variant. This improvement ranged impressively from 18% to 53% across the diverse set of variants. This suggests that the incorporation of these new variants, such as "No Comment", "Assertion", "Test Case", "Unittest Test Case", and "Dead Code", substantially bolstered the system's proficiency in AI-generated code detection. Additionally, it is noteworthy that the GLTR exhibited a notable 6% increase in the TNR when tasked with evaluating the "Long Method" variant 11. This improvement signifies an augmented capability to accurately categorize non-AI-generated code, particularly in the context of extensive and intricate code segments.

Sapling detector outperforms other AIGC Detectors, consistently achieving ACC values above 0.6 in ten out of fourteen AIGC variants. In our evaluation of Sapling's AI Detector and Figure 4, we compared its performance to the baseline (Variant 1) across various code variants. Our findings highlight significant improvements,

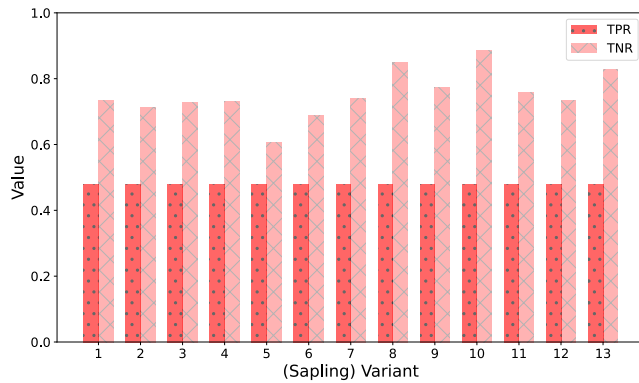


Figure 4: TPR and TNR Performance for Sapling

particularly in TNR, when the Sapling detector is applied to specific variants such as "Variable Name" (Variant 8), "Variable and Function Name" (Variant 10), and "DeadCode" (Variant 13). These variants exhibited a substantial increase in TNR, ranging from 9% to 15%, indicating their enhanced ability to detect AI-generated code accurately.

Answer to RQ1: Existing AIGC Detectors perform poorly in distinguishing between human-written code and AI-generated code, indicating the inherent weaknesses of current detectors. This underscores the need for further research and development in this domain to enhance their efficacy.

5.2 RQ2: What are the limitations of existing AIGC Detectors when it comes to detecting AI-generated code?

From the results in Table 3, we have revealed significant sensitivity to code variants, particularly with GLTR, as demonstrated by a Samples Paired t Test [10] comparing ACC differences between the original (Variant 1) and all other versions of variants (Variant 2 to 13), where the p-values for all five AIGC Detectors are 0.0296 (GLTR), 0.1032 (GPT-2 Detector), 0.2479 (GPTZero), 0.3816 (Sapling) and 0.5714 (DetectGPT). GLTR demonstrated the lowest p-value, signifying a significant performance difference across the variants. This underscores GLTR's heightened sensitivity to specific variant patterns, resulting in misclassifications, as evident in the wide range of accuracy from 0.4841 (Variant 3) to 0.7693 (Variant 7). This finding highlights a potential vulnerability of AIGC Detectors, particularly when students prompt ChatGPT to mimic a human's response when generating the code as GLTR tends to treat most AIGC incorrectly as human-written, potentially enabling academic plagiarism. Moreover, Variants 2, 3, and 5 induced a drop in performance (both ACC and TNR) across multiple models when compared to that of Variant 1, underlining the need for further research to address these specific vulnerabilities and ensure the integrity of AI-generated content detection methods in educational settings.

After some thorough analysis of each AIGC Detector's performance across each variant, we discuss some of the limitations that we observed for each AIGC Detector.

GPTZero is introduced as an AI model capable of distinguishing between human-authored and AI-generated text by analyzing perplexity and burstiness. While these properties are effective for general text, they may not translate seamlessly to the domain of programming code. The limitations of GPTZero in detecting AI-generated code become evident when we consider that programming code follows a significantly different syntax and structure compared to natural language text. Code often adheres to strict rules and conventions, making it less prone to the kinds of linguistic patterns that GPTZero is designed to detect. Therefore, GPTZero's effectiveness in detecting AI-generated code might be limited, and it could produce false positives or false negatives when applied to this specific context.

Sapling is specifically tailored to analyze text generated by language models like GPT-3 and ChatGPT. While it excels in identifying content produced by these models, it may not be optimized for programming code detection. AI-generated code can exhibit unique characteristics, such as specific coding patterns or syntactical errors, which might not align with the patterns typically seen in natural language text. Therefore, the limitations of Sapling AI Detector include its focus on text generated by language models and the potential challenges it may face when applied to programming code detection.

The GPT-2 Detector leverages a dataset comprised of text samples generated by GPT-2 models. While this approach is valuable for identifying outputs generated by GPT-2, it may not generalize well to other AI models or programming code. Different AI models may produce code with distinct characteristics that are not represented in the dataset, leading to limitations in the detector's ability to detect code generated by these models. Additionally, programming code can vary significantly in complexity and structure, making it challenging to rely solely on a dataset-based approach for detection.

DetectGPT introduces an innovative approach to text classification by introducing perturbations to input text and measuring perturbation discrepancies. While this approach shows promise in identifying AI-generated content in general, it may face limitations when applied to programming code. Code often contains intricate logic and syntax that may not exhibit the same perturbation patterns as natural language text. Therefore, DetectGPT's limitations include its adaptability to the unique characteristics of programming code and the potential for false detections or missed detections in this context.

GLTR relies on language models like GPT-2 for predictive analysis of text, utilizing word rankings and colour coding to enhance interpretability. Hence, the limitations of GLTR become obvious when considering its suitability for programming code detection as it does not follow the same linguistic patterns as natural language text, and the colour-coded word rankings may not effectively capture the nuances of code generation. Therefore, GLTR may struggle to provide accurate detections when applied to programming code.

Our findings indicate a common limitation shared among current AIGC Detectors in their ability to detect AI-generated code: they consistently perform poorly. In our comprehensive assessment of various AI detection tools, we observed a lack of accuracy and reliability in their capacity to identify AI-generated code content.

These limitations encompass several key areas:

1. Detection Accuracy: Across the board, these tools struggled to accurately identify AI-generated code, often producing high rates of false negatives and false positives, leading to inaccuracies.

2. Lack of Specificity: Existing tools encountered difficulties distinguishing between human-generated and AI-generated code, resulting in misclassifications and a lack of precision.

3. Generalization Challenges: While fine-tuning improved performance within specific code domains, these tools faced difficulty in generalizing their capabilities. They often faltered when confronted with code from diverse domains or when AI-generated code closely resembled human coding styles.

Answer to RQ2: The limitations of AIGC Detectors such as GPTZero, Sapling AI Detector, GPT-2 Detector, DetectGPT, and GLTR become evident when applied to the detection of AI-generated code. Variants 2, 3, and 5 enable students to deceive most models, leading to a significant decrease in accuracy and TNR across the majority of models. These limitations stem from the fundamental differences between programming code and natural language text. Code adheres to strict rules, follows distinct patterns, and may not exhibit the linguistic characteristics that these models are designed to detect.

6 DISCUSSION

6.1 Suggestions to SE and CS Educators

AIGC has emerged as a transformative force in education, particularly in the domains of CS/SE. This section aims to distill insights from recent research papers, to provide educators with strategies and best practices for the successful integration of AI into CS/SE education.

Researchers from Hong Kong introduced the IDEE Framework, which serves as a guiding framework for utilizing generative AI in education [19]. This framework emphasizes the importance of identifying desired outcomes, determining the appropriate level of automation, ensuring ethical considerations, and evaluating effectiveness. Moreover, the work by Kaplan et al. explores teachers' perspectives on generative AI technology and its potential implementation in education [13]. The authors suggest that teachers exhibit positive perspectives towards generative AI, with more frequent usage leading to increased positivity. Educators perceive generative AI as a tool for enhancing professional development and student learning. This suggests that embracing generative AI in education has the potential to positively impact both educators and students, fostering continuous growth and improved learning experiences.

On the other hand, the work by Chan et al. delves into the experiences and perceptions of Gen Z students and Gen X/Y teachers regarding the use of generative AI in higher education [9]. Students express optimism about the benefits of generative AI, while teachers emphasize the need for guidelines and policies to ensure responsible use. Furthermore, an AI Ecological Education Policy Framework for higher education [8]. It addresses three dimensions: Pedagogical, Governance, and Operational, providing a comprehensive structure to navigate the implications of AI integration.

Based on the literature and research findings, the following recommendations are provided for educators:

1. Define Objectives: Precisely outline the educational objectives that are in harmony with the application of generative AI. This step ensures a seamless integration of technology into educational purposes, fostering alignment between technology utilization and educational aspirations.

2. Automation Level: When incorporating generative AI into education, it is essential to carefully consider the level of automation to employ. This decision hinges on whether to pursue full automation, where AI systems handle educational tasks entirely on their own, or opt for a supplementary approach that blends AI capabilities with human involvement. This choice plays a pivotal role in shaping the educational landscape, as it determines the extent to which technology should be seamlessly integrated into the learning process, aligning with the unique goals and requirements of each educational scenario.

3. Ethical Focus: Give paramount importance to ethical concerns. Develop comprehensive guidelines and policies to safeguard responsible and ethical usage of AI in the educational context.

4. Continuous Evaluation: It is essential to maintain an ongoing process of assessing the effectiveness of generative AI in education. This involves systematically monitoring how well the technology serves educational objectives, identifying areas for improvement, and adapting strategies as needed, ultimately contributing to improved outcomes and enhanced educational experiences.

5. Comprehensive Policies: In the integration of generative AI within educational settings, it is imperative to develop comprehensive guidelines and policies based on empirical evidence and best practices. By adopting an evidence-based approach, educational institutions can ensure that their policies are not only robust and compliant but also rooted in the real-world outcomes and experiences, safeguarding both the educational mission and the well-being of all stakeholders.

6. Stay Informed: Promote ongoing research and evaluation of AI integration in educational settings to stay informed about advancements, benefits, and risks associated with AI technology.

6.2 Threats to validity

6.2.1 Internal Validity. The study faces challenges related to the varied prompts used to generate AIGC with ChatGPT. While these prompts aim to simulate average user inputs, they may not fully mirror real-world scenarios. Additionally, ChatGPT's non-deterministic nature, leading to diverse responses from the same prompt, could impact result reproducibility.

Another challenge pertains to verifying the authenticity of source code written by humans. For instance, datasets sourced from platforms such as Kaggle could include content generated by LLM tools, blurring the distinction between human and AI contributions. However, since these datasets were released to the public before LLM models went mainstream, the impact is estimated to be minimal.

In specific scenarios, vague queries leading to responses such as "I'm sorry, as an AI language model, I am unable to provide code as the question lacked specific details about the expression to be evaluated" pose a risk to internal validity. Ambiguous or open-ended coding questions can result in inaccurate AIGC. Therefore,

the datasets of the question prompts underwent preprocessing. During this process, rows containing responses such as the one mentioned were removed, ensuring the integrity and accuracy of the dataset used in our study.

6.2.2 Construct Validity. Construct validity explores the intricate interplay between theoretical constructs and empirical data, posing concerns about potential biases arising from platforms such as Kaggle, where datasets are curated and shared, and ChatGPT may have been trained on datasets similar to the one used in this study. However, understanding ChatGPT's ability to generate diverse and contextually relevant content, rather than simply replicate existing snippets, this research maintains objectivity and rigor. These insights allow for a nuanced approach, ensuring the study's authenticity and safeguarding the integrity of the research findings.

6.2.3 External Validity. It is crucial to acknowledge that the conclusions drawn in this study might be specific to the datasets analyzed. To enhance the generalizability of our study, we meticulously curated our dataset by gathering data from diverse sources, aiming to replicate real-world software development scenarios as closely as possible. Furthermore, we concentrated our efforts on the Python programming language, a deliberate choice made to maintain consistency and control within our study.

7 RELATED WORK

This section discusses several studies and research articles that explore the detection of AI-generated content, particularly in academic and educational contexts.

The study by Otterbacher [17] highlights the need to develop a culture that promotes responsible and ethical use of generative AI in various domains, including science and education. Rather than relying solely on technical solutions to combat AI-generated content, the authors argue for a holistic approach that considers the broader implications of AI in academia. On the other hand, the work by Chaka [7] evaluated AI content detection tools, highlighting their limitations in accurately detecting AI-generated content, especially in academic contexts. These limitations can lead to academic integrity issues, including plagiarism. Chaka's study [7] also emphasizes the need for ongoing research to evaluate the accuracy and reliability of AI content detection tools. Improving these tools' ability to detect AI-generated content is crucial for combating AI-generated plagiarism in academia. AIGC, notably code produced by ChatGPT, shows promise in software-related tasks but raises concerns in education, particularly regarding plagiarism [23].

The rapid review presented in [15] highlights the varied performance of ChatGPT in different subject domains and raises concerns about its accuracy and potential for plagiarism. It underscores the need for schools and universities to update their assessment methods and policies to address these challenges. For instance, REDACTED University promotes responsible and ethical AI usage, including generative AI like ChatGPT, as part of our commitment to preparing graduates for emerging technologies. The assessment procedures require REDACTED to align units with our AI guidelines.

In relation to our study, these works provide valuable insights into the limitations of existing AI content detection tools, especially

in the context of AI-generated code. They underscore the need for a comprehensive evaluation of AIGC Detectors' performance and the development of ethical guidelines to ensure responsible AI-generated content use in education.

8 CONCLUSION AND FUTURE WORK

The rise of generative AI models presents both opportunities and challenges, particularly in education and programming. Our study aimed to assess AI content detection tools' effectiveness in identifying AI-generated code, revealing their limitations and offering insights for educators and students. We examined various AI content detection models using a dataset containing human-written and AI-generated code. Our evaluation focused on metrics such as recall, precision, F1 score, accuracy, and AUC.

We have revealed significant sensitivity in GLTR, as indicated by its remarkably low p-value of 0.0296 in the Samples Paired t Test. This sensitivity led to notable accuracy discrepancies ranging from 0.4841 to 0.7693, exacerbated by specific code variants such as variants 2, 3, and 5. These findings underscore the imperative for immediate research efforts to enhance the reliability of AI-generated content detectors, safeguarding academic integrity in educational contexts. Addressing these challenges is pivotal for educators and institutions to adeptly navigate the complexities introduced by AI-generated code, ensuring the integrity of programming education.

Our findings suggest that these tools hold promise in distinguishing AIGC from human-written code but face challenges due to code complexity and writing style variations. Ethical guidelines for AIGC integration into education are essential. Moreover, we have found that GLTR is very sensitive to different variants introduced into the AIGC produced by ChatGPT, proven by the smallest p-value when the Samples Pair t-Test is applied to the AIGC Detector's accuracies and the wide range of the accuracy.

Educators and institutions should adopt strategies for responsible AI usage, considering curriculum design, assessment methods, and ethical directives. Additionally, the long-term impact of AIGC Detector tools on student skill development and creativity in programming warrants further exploration. In conclusion, our research highlights the evolving landscape of AI-generated code and the role of AIGC Detectors in education. Emphasizing responsible AI usage, ethical guidelines, and ongoing tool refinement can empower students in a technology-driven world while preserving academic integrity and fostering creativity.

Future research may prioritize the enhancement of AI content detection models, enabling them to effectively tackle a broader spectrum of code variations and writing styles. This, in turn, would bolster their dependability when applied within educational contexts. Furthermore, delving into the enduring effects of AIGC Detector tools on students' learning journeys and their creative engagement in programming could yield invaluable insights for educators and institutions navigating the technology-driven landscape of education.

9 DATA AVAILABILITY

The replication package, along with the associated data, has been made publicly available.⁸

⁸https://figshare.com/articles/dataset/Replication_Package/24298036

REFERENCES

- [1] [n. d.]. *GPTzero*. <https://gptzero.me/>
- [2] [n. d.]. *Sapling*. <https://sapling.ai/ai-content-detector> [Online]. Available.
- [3] Fawad Ali. 2023. GPT-1 to GPT-4: Each of OpenAI's GPT Models Explained and Compared. (11 April 2023). <https://www.makeuseof.com/gpt-models-explained-and-compared/>
- [4] David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *Journal of AI* 7, 1 (2023), 52–62.
- [5] BurhanUITayyab. 2023. DetectGPT. <https://github.com/BurhanUITayyab/DetectGPT>.
- [6] Ralph Cajipe. 2023. chatgpt-prompt-engineering. <https://github.com/ralphcajipe/chatgpt-prompt-engineering/blob/main/1-guidelines.ipynb>.
- [7] Chaka Chaka. 2023. Detecting AI content in responses generated by ChatGPT, YouChat, and Chatsonic: The case of five AI content detection tools. *Journal of Applied Learning and Teaching* 6, 2 (2023).
- [8] Cecilia Ka Yuk Chan. 2023. A comprehensive AI policy education framework for university teaching and learning. *International Journal of Educational Technology in Higher Education* 20, 1 (2023), 1–25.
- [9] Cecilia Ka Yuk Chan and Katherine KW Lee. 2023. The AI generation gap: Are Gen Z students more interested in adopting generative AI such as ChatGPT in teaching and learning than their Gen X and Millennial Generation teachers? *arXiv preprint arXiv:2305.02878* (2023).
- [10] Frances Chumney. 2018. PAIRED SAMPLES t & WILCOXON SIGNED RANKS TESTS. Retrieved January 24 (2018), 2022.
- [11] Sebastian Gehrmann, Hendrik Strobelt, and Alexander Rush. 2019. GLTR: Statistical Detection and Visualization of Generated Text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, Florence, Italy, 111–116. <https://doi.org/10.18653/v1/P19-3019>
- [12] Biyang Guo, Xin Zhang, Ziyuan Wang, Minqi Jiang, Jinran Nie, Yuxuan Ding, Jianwei Yue, and Yupeng Wu. 2023. How Close is ChatGPT to Human Experts? Comparison Corpus, Evaluation, and Detection. *arXiv preprint arxiv:2301.07597* (2023).
- [13] Regina Kaplan-Rakowski, Kimberly Grotewold, Peggy Hartwick, and Kevin Papin. 2023. Generative AI and Teachers' Perspectives on Its Implementation in Education. *Journal of Interactive Learning Research* 34, 2 (2023), 313–338.
- [14] Ajay Kulkarni, Deri Chong, and Feras A Batarseh. 2020. Foundations of data imbalance and solutions for a data democracy. In *Data democracy*. Elsevier, 83–106.
- [15] Chung Kwan Lo. 2023. What is the impact of ChatGPT on education? A rapid review of the literature. *Education Sciences* 13, 4 (2023), 410.
- [16] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. 2023. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 24950–24962. <https://proceedings.mlr.press/v202/mitchell23a.html>
- [17] Jahna Otterbacher. 2023. Why technical solutions for detecting AI-generated content in research and education are insufficient. *Patterns* 4, 7 (2023).
- [18] Hendrik Strobelt, Sebastian Gehrmann, and Alexander Rush. [n. d.]. Catching a Unicorn with GLTR: A tool to detect automatically generated text. Collaboration of MIT-IBM Watson AI lab and HarvardNLP. <http://gltr.io/>
- [19] Jiahong Su and Weipeng Yang. 2023. Unlocking the power of ChatGPT: A framework for applying generative AI in education. *ECNU Review of Education* (2023), 20965311231168423.
- [20] Teo Susnjak. 2022. ChatGPT: The end of online exam integrity? *arXiv preprint arXiv:2212.09292* (2022).
- [21] Chip Thien. 2023. *gpt-2-output-dataset*. <https://github.com/MacroChip/gpt-2-output-dataset>
- [22] Ž Vujović et al. 2021. Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications* 12, 6 (2021), 599–606.
- [23] Jian Wang, Shangqing Liu, Xiaofei Xie, and Yi Li. 2023. Evaluating AIGC Detectors on Code Content. *arXiv preprint arXiv:2304.05193* (2023).
- [24] www.kaggle.com. 2023. *Leetcode Solutions and Content KPIs*. <https://www.kaggle.com/datasets/jacobhds/leetcode-solutions-and-content-kpis> Last accessed on May 16, 2023.
- [25] www.kaggle.com. 2023. *Natural Language to Python Code*. <https://www.kaggle.com/datasets/linkanjarad/coding-problems-and-solution-python-code> Last accessed on May 16, 2023.
- [26] www.quescol.com. 2023. *Python Coding Question: 90+ Python Interview Coding Questions*. https://quescol.com/interview-preparations/python-coding-question#google_vignette Last accessed on May 16, 2023.