Attributing ChatGPT-generated Source Codes

Soohyeon Choi, David Mohaisen, *Senior Member, IEEE*

Abstract—AI assistants such as ChatGPT have remarkable human-like capabilities, producing natural language and programming language utterances. Despite that, ChatGPT could facilitate academic misconduct by easily generating codes and text as solutions for assignments. More alarmingly, ChatGPT can be used to write polymorphic malware. Moreover, ChatGPT-generated codes are shown to be less secure. While the detection of text generated by ChatGPT has been addressed, ChatGPT code authorship attribution is largely unexplored. In this paper, we examine attributing ChatGPT codes using off-the-shelf code authorship attribution techniques. We demonstrate that the answer to the question is negative, necessitating a new approach, which we also deliver by scrutinizing the outcomes of the off-the-shelf attribution technique. We found that grouping ChatGPT codes using the inference step of a pretrained model on non-ChatGPT codes can be used as an accurate attribution model. Compared with the 8.3%–29.2% accuracy of the naive approach, our approach delivers 81.3%–91.7% while costing a small trade-off in the accuracy of detecting target (non-ChatGPT) authors. Moreover, the straightforward authorship attribution model trained for the binary classification (ChatGPT versus *Human*) achieved a classification accuracy of 87% with 6K code samples. Our comprehensive analysis sheds light on the limitations of the styles generated by ChatGPT, making detecting codes generated by ChatGPT feasible.

Index Terms—Code Authorship Identification, Program Stylistic Features, Machine Learning, ChatGPT, Measurement

1 Introduction

OpenAI recently released "ChatGPT", a large language model (LLM) built on top of the generative pretrained transformer 3.5 (GPT-3.5) [1]. GPT-3.5 is an improvement of its predecessor, GPT-3, a set of models that can understand and generate natural language instances [2]. While GPT-3 is trained only for natural languages, GPT-3.5 is trained on natural and programming language generation tasks by including four models: "code-davinchi-002", "text-davinchi-002", "text-davinchi-003", and "gpt-3.5-turbo-0301" [3]. Through this combination of models and the training with over 750 gigabytes of plain text [3], ChatGPT has shown remarkable results interacting in conversational dialogue form with users and has human-level writing skills [4], [5], [6]. Thus, ChatGPT is used for writing essays or even books, and has been shown to provide remarkable capabilities in paraphrasing English texts and source codes (*e.g.*, C++, Java, etc.) by millions of users.

Due to its excellent skills, ChatGPT can be a source of significant abuse. Recent reports have shown that ChatGPT can be used to write malware [7], [8], [9], [10], [11]. Also, ChatGPT is used by students for doing their programming assignments, constituting an academic misconduct [12], [13], [14], [15]. Moreover, although that may not generalize to other AI assistants such as GitHub Copilot [16], ChatGPT-generated codes constitute multiple issues [17], [18], [19], including lower security levels and more security vulnerabilities across multiple questions [18]. For encryption and decryption, the resulting codes are significantly more likely to utilize basic ciphers [20], [21], [22] and disregard basic authenticity checks on the final returned value. Moreover, AI assistants use unsafe randomness more frequently than humans when generating codes for certain tasks. In the case of structured query language (SQL) queries, AI assistants frequently use the string concatenation function, which can potentially lead to SQL injection attacks [23].

Due to these concerns, authorship attribution techniques are needed to distinguish AI-generated content from source codes. We note that there are various solutions in the context of natural languages, such as "GPTZero" [24] and "ZeroGPT" [25], which were trained on paired human-written text and AI-generated text and achieved higher than 98% of accuracy on human-written text classification. Methods that do the same task in the context of programming languages are yet to materialize. Fortunately, code authorship attribution is heavily utilized for programming languages to identify source code's author(s) [26], [27], [28]. Code authorship attribution involves identifying the individual or group responsible for creating a particular source code based on unique stylistic features present within the code [29], [30], [31], [32], [33], [34]. Research studies have shown that programmers have distinct programming styles, which can be accurately identified using various lexical, layout, and syntactic features [29], [30]. Moreover, machine learning techniques are employed to achieve code authorship attribution automation [35], [36].

For instance, Caliskan Islam *et al.* [29] de-anonymized programmers using random forests (RFs) over abstract syntax tree-based features. They extracted lexical, layout, and syntactic features of code, built an RF classifier with the selected features, and achieved 93% in accuracy with 1,600 C++ authors. Similarly, Abuhamad *et al.* [30] utilized deep learning-based techniques, such as recurrent neural networks (RNN) and term frequency-inverse document frequency (TF-IDF) to extract stylistic features of code, with 92% of accuracy in identifying over 8,900 C++ authors.

Issues with Code Authorship Attribution. Despite their usefulness, code authorship methods can be easily circum-

• *Soohyeon Choi and David Mohaisen are with the Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA. E-mail: soohyeon.choi@ucf.edu, david.mohaisen@ucf.edu*

Manuscript received April 19, 2005; revised August 26, 2015.

vented through code *transformations* that change the stylistic patterns of source codes, which are shown highly effective, producing higher than 99% of evasion success rates [31], [37]. In the same line, recent work on understanding ChatGPT's capabilities for code generation have also alluded to its code transformation capabilities [38], with ChatGPT improving code quality, refactoring, requirements elicitation, and design as a result. Moreover, given that ChatGPT is an agglomeration of models, it is unclear whether its authorship stylistic characteristics could be narrowed down into a fixed author. A natural research question is the following:

- **RQ 1:** is it possible to detect codes generated by ChatGPT using off-the-shelf attribution techniques?

To address this question, we build a code authorship attribution model from the literature to evaluate the effectiveness of attributing codes generated by ChatGPT. The straightforward method performs poorly, even when relaxing the attribution settings. This negative answer, presented in detail in section 3.5, motivates the following question:

- **RQ 2:** Can we use the stylistic authorship features of codes generated by ChatGPT to attribute such codes?

We scrutinize the outcomes of the authorship attribution to understand the misclassifications of ChatGPT codes and to investigate how ChatGPT produces styles. To identify patterns in ChatGPT code, we start with a pretrained code attribution model trained explicitly on non-ChatGPT code as a proxy for regrouping codes generated by ChatGPT through the predicted labels. Our evaluation highlights the possibility of detecting ChatGPT codes with high accuracy (81.3% to 91.7%). While there were some limitations to the scope of styles that ChatGPT can produce, it can still generate source code in various styles.

Contributions. 1) We ask whether it is feasible to identify the authorship of ChatGPT codes using an existing code authorship attribution, and show that to be challenging. 2) We develop a new highly-accurate feature-based approach for ChatGPT code attribution. 3) We designed and trained an accurate code authorship model using ChatGPT codes and non-ChatGPT codes attribution technique in binary classification settings. 4) We provide a comprehensive and comparative analysis of the proposed approaches.

Organization. The prior work is reviewed in section 2. The technical methods, including an overview of ChatGPT, motivation, threat model, naive approach, and our technique, are presented in section 3. In section 4, we present our evaluation and discussion. We conclude in section 5.

2 Related Work

Code authorship attribution, in general, has been covered by various studies [30], [31], [32], [35], [37], which we highlight in section 3, as we customize some of them and explain in more detail. Moreover, to the best of our knowledge, no prior work in the literature explores the code authorship attribution task of source codes generated by chatbots or AI assistants, not ChatGPT or any other.

Since AI assistants, such as GitHub Copilot [39], OpenAI Codex [40], [41], ChatGPT [1], [3], and DeepMind AlphaCode [42], [43], are becoming proficient in programming, they are increasingly being used for various tasks, both positive and negative. Moreover, researchers studied the behavior and potential uses of these assistants [44], [45], [46].

Malware Authoring. Malware is a prominent (mis)use that makes a strong case for attribution [47], [48]. Pa *et al.* [10] demonstrated that ChatGPT can produce a functional malware of up to about 400 lines of code in about 90 minutes. Beckerich *et al.* [49] demonstrated the use of ChatGPT in producing a fully functional payload distribution, using ChatGPT as a command and control channel for malware distribution. Chatzoglou *et al.* [11] used ChatGPT to bypass several antivirus scanners by exploiting its power to produce ready-to-use malware. Several recent reports highlighted that ChatGPT can be used to produce general malware [9], polymorphic malware [7], and ransomware [8]. These studies highlight a need to attribute AI-generated codes.

Security of Produced Codes. Although advantageous, concerns regarding AI assistants' security have also been raised. For instance, Perry *et al.* [18] and Asare *et al.* [16] compared the security features of AI assistant-generated code with human-generated code. Perry *et al.* analyzed security-related questions, such as encryption & decryption, signing a message, SQL, etc, and answers from an AI assistant, OpenAI Codex [40], [41]. Asked to generate code for encryption and decryption using a specified symmetric key, code for signing a given message using a specified algorithm, and code related to SQL, the results indicated that the generated code often produced insecure solutions. Moreover, the AI-generated codes did not check for authenticity on the final returned value and often used unsafe randomness. Asare *et al.* [16] demonstrated that, while GitHub Copilot performed differently across different types of vulnerability reproduction, it did not perform as poorly as human in producing those vulnerabilities in general. Although positive, the results of Asare *et al.* [16] do not deny the need for attributing codes generated by AI assistants for further testing.

Additionally, Jess *et al.* [50] and Vasconcelos *et al.* [51] argued that LLMs could inherit bugs and vulnerabilities due to inherent limitations. In [50], the authors investigated how prone Codex is to generate simple and stupid bugs. They found that while Codex can aid in bug prevention, it is also up to twice as likely to generate simple and stupid bugs compared to generating accurate code snippets. AI models might struggle to understand the complex interactions between different code components. This could lead to suggestions that introduce security vulnerabilities when integrated into the larger codebase [51].

3 Technical Methods

We utilized ChatGPT as our AI assistant to generate source codes and applied code authorship attribution techniques to analyze the stylistic patterns of ChatGPT code and achieve authorship of them. This section summarizes this background in addition to our threat and attribution models.

3.1 Code Authorship Attribution

A central component in our pursuit of ChatGPT's authorship attribution is the attribution technique itself, where there have been multiple competitive approaches in the

Table 1: Code authorship attribution models' accuracy.

Method	Dataset	Accuracy
Caliskan-Islam *et al.* [29]	GCJ 2017	90.4%
	GCJ 2018	80.5%
	GCJ 2019	85.8%
Average		85.6%
Abuhamad *et al.* [30]	GCJ 2017	84.9%
	GCJ 2018	72.9%
	GCJ 2019	84.6%
Average		80.8%

literature [29], [30], [31], [32], [35], [36]. Given the richness of this space and retrospective sufficiency, we use existing works in the literature for this part. Most remarkable is the work of Caliskan-Islam *et al.* [29], where they were able to accurately distinguish the programming styles and patterns of 1.6K programmers in C/C++ codes, allowing them to identify each individual with an impressive degree of accuracy (over 92%). To justify the choice of a code authorship attribution technique over others, we performed experiments with our datasets (more details are in section 4), employing Caliskan-Islam *et al.*'s approach [29] and AbuHamad *et al.*'s approach [30]. As shown in Table 1, Caliskan-Islam *et al.*'s method is shown to be superior by achieving an average accuracy of more than 85%, compared to an average accuracy of 80.8% by Abuhamad *et al.*'s. We explain details of this method in the following.

Caliskan-Islam *et al.* [29] introduced a machine learning-based method for code authorship attribution by analyzing stylistic patterns in C/C++ code, using features like lexical, layout, and syntactic elements to identify unique author characteristics. Thus, their model has two main phases: the feature extraction phase and the learning phase, and we explain those steps in the following.

Feature Extraction. To train machine learning techniques to perform well for a given task, it is a prerequisite to provide a suitable representation of data. For code authorship attribution, the representation should contain each source code's stylistic patterns. Therefore, Caliskan-Islam *et al.* used three types of features, lexical, layout, and syntactic features, for this purpose. They obtained the lexical and layout features from the source code and the syntactic features from the abstract syntax tree (AST) of each code.

Lexical Features. The lexical features include the choice of variable and function names by the programmer. Each programmer may have a unique preference for the selection of words used in naming variables and functions. For instance, an author may choose to name their functions and variables based on their functionality, such as naming a function that calculates the maximum value of elements in an array as "CalculateMax", naming the array as "values", and the maximum value as "max_value". However, another author may opt for shorter forms, such as "c_Max", "v", and "m", as shown in the example in Figure 2. Those stylistic features may serve as a strong indicator (cue) for attribution.

Layout Features. The layout features contain various aspects of the formatting and presentation of the code, such as indentation, comment format, and the use of brackets. Indentation refers to the placement of tabs or spaces at the beginning of a line to indicate the level of nesting within the code. The form of comments, such as whether they are single- or multi-line, and how they are formatted, can also be indicative of an author's style. Finally, the use of brackets,

```
int main() {
    int a = 5;
    int b = 3;
    // Calculate the sum of a and b
    int sum = a + b;
    if (sum > 10) {
        cout << "Greater than 10." << endl;
    } else {
        cout << "Less than or equal to 10." << endl;
    }
    return 0;
}
```

```
int main()
{
  int a = 5;
  int b = 3;
  /* calculate
     the sum of a and b */
  int sum = a + b;
  if (sum > 10)
  {
    cout << "Greater than 10." << endl;
  }
  else
  {
    cout << "Less than or equal to 10." << endl;
  }
  return 0;
}
```

Figure 1: Examples of layout features with different styles of indentation, comment format, and the use of brackets. (top) Single-line comment, four spaces indentation, and same line opening bracket. (bottom) Multi-line comment, two spaces indentation, and separate line opening bracket.

including their placement and how they are aligned, can provide further clues as to the authorship of a code.

As an example, one author may have a preference for single-line comments marked by "//", four spaces of indentations, and opening brackets on the same line as the function. On the other hands, another author may prefer to use multi-line comment marked by "/* */", two spaces of indentation, and opening brackets on a separate line. This can be seen in the examples provided in Figure 1.

Syntactic Features. The coding style adopted by programmers can be distinguished by their usage of syntax and control flow, which can be examined through the fundamental data structures of compiler design, known as AST [52]. The AST enables the analysis of stylistic patterns related to syntax and control flow and forms an important part of the syntactic feature set that encompasses properties of language-dependent AST and keywords.

Machine Learning Component. Caliskan-Islam *et al.* utilized the three sets of features extracted from the feature extraction phase to train a random forest classifier [53] for the code authorship attribution task. To accomplish that, they first selected certain features among those they extracted and converted the code into a vector space using the selected features to allow the machine learning algorithm to access them. They then utilized a random forest ensemble classifier to identify the authorship of the codes.

Feature Selection. Due to this method's extensive reliance on unigram term frequency and TF-IDF calculations and a wide range of individual terms in the code, this method produced sparse feature vectors with high dimensionality. Sparse feature vectors can negatively impact the accuracy of random forest classifiers by limiting useful splits with zero-

valued features, leading to poorer fits and larger trees. As a solution, the authors utilized a feature selection technique using the information gain criterion [54]. Choosing a smaller set of more informative features significantly reduces the sparsity in the feature vector, enabling the classifier to generate more accurate results. This method evaluates the difference between the entropy of a class distribution and the entropy of the conditional class distribution given a specific feature as follows:

$$IG\left(A, M_i\right) = H(A) - H\left(A \mid M_i\right),$$

where A represents the category of an author, H represents the Shannon entropy, and M_i represents the $i-th$ feature of the dataset. In simpler terms, the information gain (IG) measures how much information is gained by knowing the value of a feature in relation to the corresponding class label, for example. The authors utilized feature selection by selecting only the features with a non-zero IG and IG is always non-negative due to $H\left(A \mid M_i\right) \leq H(A)$. This was done to reduce the size and sparsity of the feature vector, resulting in a more manageable set of features.

Random Forest Classification. The authors trained a random forest ensemble classifier using the selected features. To balance accuracy and processing time well, they constructed a random forest consisting of 300 trees (the same setting followed in Abuhamad *et al.* [30]). The classification process involved applying each decision tree in the random forest to the given code and its features by following the binary decisions at each node until reaching a leaf and then combining the results to produce the outcome. The output was determined by selecting the most frequent label, making the classification process straightforward.

3.2 ChatGPT: An Overview

ChatGPT (GPT 3.5) is a large-scale language model developed by OpenAI [1], [3], based on GPT architecture. GPT is a type of neural network architecture that achieved state-of-the-art performance on a wide range of natural language processing (NLP) tasks, including language modeling, text generation, text translation, and text classification [55].

The architecture of GPT is based on the transformer model and was introduced by Vaswani *et al.* [56]. The transformer model uses self-attention mechanisms to process input sequences of tokens and produce output sequences. The self-attention mechanism allows the model to attend to different parts of the input sequence when generating each output token, making it well-suited for generating long sequences of text. The GPT architecture is specifically designed for language modeling [57], which is the task of predicting the probability distribution of the next word in a sequence given the previous words. To achieve this, GPT is pretrained on large amounts of text data using an unsupervised learning algorithm called masked language modeling [58]. This involves randomly masking out some of the input tokens and training the model to predict the masked tokens based on the context of the surrounding tokens. Once trained, the model can be fine-tuned on specific downstream NLP tasks, such as text classification, text translation, question answering, or dialogue generation.

ChatGPT has been further trained with the reinforcement learning from human feedback (RLHF) technique [59], [60]. In this technique, a human AI trainer engages in conversations while playing both the user's and an AI assistant's roles to achieve supervised fine-tuning. During this process, the trainer can access the model's suggested responses to assist them in creating their own. As a result, the model has been fine-tuned on a vast dataset of conversational data types, allowing it to generate responses that resemble human-like reactions to various input prompts.

To generate a response, ChatGPT takes the input prompt and uses its learned knowledge about the language and conversation to generate a new sequence of text that continues the conversation in a meaningful way. The generated response is not simply pre-programmed, but rather a novel combination of words and phrases that the model has learned from the training data. Our experiments revealed that ChatGPT is able to generate diverse responses to a single question, which we elaborated on in section 4.2.

ChatGPT is also trained using programming languages data by exposing it to a vast amount of code snippets and examples from various programming languages, such as Python, JavaScript, Java, C++, etc. This exposure helps ChatGPT learn programming concepts, syntax, structure, and problem-solving approaches to some extent.

In general, ChatGPT is a significant advance in the field of natural language processing as well as programming language processing and has the potential of enabling new forms of human-machine interaction and communication, making it an important innovation in many areas.

3.3 Motivation: Concerns with ChatGPT

A central motivation of our work is the questionable use of AI assistants, *e.g.,* producing malware, potentially less secure code, or code for an assignment (in a classroom setting) that could be considered misconduct. All those uses, with varying degrees, make a strong case for ChatGPT-generated source code attribution.

Addressing Malware Authorship. Authorship attribution in the context of malware analysis is a first step towards defending against it. For instance, knowing the source of a piece of malware to be ChatGPT would allow a collaborative effort in patching ChatGPT to prevent the production of such malicious codes or their variants.

Addressing Code Security by Attribution. It is essential to consider the security implications of AI assistants' code generation capabilities and ensure the security of the generated code. As discussed in section 2, several works explore the security features of AI-generated codes (*e.g.,* using OpenAI Codex [40], [41]) and compared them with human-generated code, showing that the security level of the code produced by the AI assistant might be substandard. Consequently, there is a need to distinguish between AI-generated and human-written codes to prevent those issues in the security domain. For instance, if a code is identified as written by AI assistants (*e.g.,*ChatGPT), we need to assess its security features—often using customized tools.

Addressing Plagiarism. Due to the lack of explicit policies surrounding using ChatGPT for text generation, there is a growing concern that this generation could be considered

```cpp
int CalculateMax(int values[], int size){
 int max_value = max_element(values, values+size);
 return max_value;
}
int main(){
 int size = 3;
 int values[size] = {5, 10, 7};
 int max_value = CalculateMax(values, size);
 cout << "The maximum value is: " << max_value <<endl;
 return 0;
}
```

(a) Functionality naming

```cpp
int c_Max(int v[], int size){
 int m = max_element(v, v+size);
 return m;
}
int main(){
 int size = 3;
 int v[size] = {5, 10, 7};
 int m = c_Max(values, size);
 cout << "The maximum value is: " << m <<endl;
 return 0;
}
```

(b) Short form naming

Figure 2: Examples of lexical features with functionality naming and short form naming. The names of functions (line 1 (`Calculate max`)) ↔ line 1 (`c_Max`)), arrays (line 7 (`values`) ↔ line 7 (`v`)), and variables (line 8 (`max_value`) ↔ line 8 (`m`)) have different naming styles.

plagiarism [61]. Consequently, there has been considerable research addressing these issues through investigation and quantification [12], [13], [14] as well as providing actual solutions [24], [25]. In [12], Susnjak conducted a study to explore the potential use of ChatGPT as a tool for academic misconduct in online exams and suggested that ChatGPT could be a significant threat to the integrity, especially in education where such exams are increasingly common.

Tools like GPTZero [24], which detect text or writing generated by ChatGPT, exist. However, these tools are specific for text and do not address the detection of codes generated by ChatGPT. This is mainly due to the various styles and formats that ChatGPT is capable of generating code in, which poses a significant challenge when using traditional methods of code authorship attribution, as we show later.

Technical Challenges. Code authorship attribution can be approached by extracting stylistic features from each source code and utilizing them to determine their authorship. However, this approach often fails when it encounters multi-authored codes or authors who can implement various styles in their code. Moreover, this approach is easily circumvented by code stylistic transformation methods as previous research has demonstrated in [31], [37]. AI assistants are usually trained with numerous data to obtain high-quality performance. Therefore, AI-generated codes often present diverse styles even generated from the same model and same question which makes the authorship attribution of AI-generated code is challenging for the current approach. Therefore, we examined stylistic patterns of AI-generated codes to create dataset by re-grouping for training authorship model. As such, the authorship attribution model can classify AI-generated code accurately.

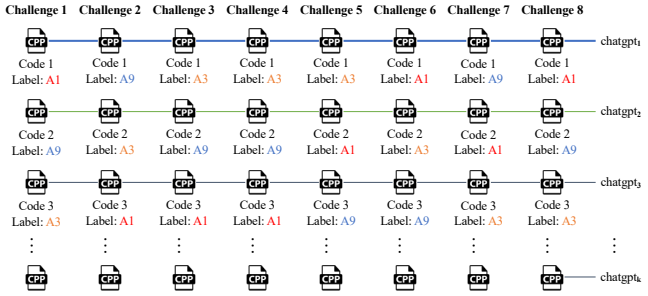


Figure 3: An illustration of naive approach that highlights the mental model used for attributing codes authored by ChatGPT: responses (codes) to challenges are assumed to be generated by the same model in the order of the run.

3.4 Use and Threat Model

Our main goal of this study is to accurately distinguish source codes generated by AI assistants, particularly ChatGPT [1], from non-AI generated source codes using existing code authorship attribution methods to address the unethical and security-associated issues.

For our use and threat models, we assume an adversary has access to the ChatGPT model and instances of code samples generated by the model. The adversary (ChatGPT; *e.g.,* could be a student using ChatGPT for generating code as an assignment, engaging in academic misconduct) is interested in generating programming language codes with high quality that satisfy certain constraints.

We note that ChatGPT model has been trained with the same pretraining dataset as GPT-3, its predecessor, and this dataset comprises over 750 gigabytes of plain text and contains approximately 500 billion tokens. In addition, the model underwent a fine-tuning process to attain human-level capabilities in both natural language and programming language [1], [2]. Thus, we use this setting to spell out two constraints. First, given the large number of human-generated tokens used for training ChatGPT, we naturally assume that the source code generated by ChatGPT will likely exhibit diverse styles and patterns, rendering code authorship attribution techniques ineffective (which we verify in our experimental evaluation against the naive setting). Second, given the effort required for training the ChatGPT model, it is also fair to assume that the adversary does not have adaptive capabilities: we test ChatGPT's generated codes as-is, and ChatGPT is not aware of the effort to detect the codes generated by it and attributing them to it. This adversary, while theoretically possible, is unlikely to be realized due to the amount of effort of modifying ChatGPT.

3.5 Synchronized Attribution and Limitations

Naive Approach. For a plausible operation of ChatGPT, a user would interact with ChatGPT by posing questions and obtaining responses (answers). As such, the user is assumed to utilize the first answer of ChatGPT unless this answer is wrong. Moreover, it is natural to assume—in the same session—that the user will not repeat the same question. We denote this scenario as the "naive" use of ChatGPT for obtaining code samples that we use for code authorship attribution. The naive approach is to be associated with code generation, reflecting the natural prompting of ChatGPT.

Table 2: The accuracy (with average accuracy) of the naive approach using Caliskan-Islam *et al.*'s method [29] over GCJ 2017, 2018, and 2019. The challenge-level accuracy (fold accuracy), the granular (G) accuracy (average of per-fold accuracy), and the holistic (H) accuracy (as percentage).

	GCJ 2017			GCJ 2018			GCJ 2019		
c	210	(G)	(H)	210	(G)	(H)	206	(G)	(H)
1	85.7	0.0	16.7	64.3	0.0	0.0	90.8	0.0	50.0
2	86.7	16.7	33.3	78.1	0.0	0.0	89.8	50.0	100
3	85.2	0.0	33.3	86.7	0.0	0.0	77.7	50.0	50.0
4	86.7	33.3	66.7	88.6	0.0	0.0	91.7	50.0	100
5	86.2	0.0	16.7	71.0	0.0	0.0	92.7	50.0	50.0
6	89.5	16.7	16.7	80.5	0.0	0.0	86.9	0.0	50.0
7	88.6	0.0	33.3	68.1	0.0	0.0	80.6	50.0	100
8	90.5	0.0	16.7	81.4	0.0	0.0	78.2	50.0	100
a	87.4	8.3	29.2	77.3	0.0	0.0	86.0	37.5	75.0

Table 3: The accuracy (with average accuracy) of the naive approach using Abuhamad *et al.*'s [30] over GCJ 2017, 2018, and 2019. The challenge-level accuracy (fold accuracy), the granular (G) accuracy (average of per-fold accuracy), and the holistic (H) accuracy (as percentage).

	GCJ 2017			GCJ 2018			GCJ 2019		
c	210	(G)	(H)	210	(G)	(H)	206	(G)	(H)
1	86.2	16.7	16.7	51.0	0.0	0.0	85.4	0.0	50.0
2	78.6	0.0	16.7	73.8	0.0	0.0	90.3	50.0	100
3	83.3	0.0	33.3	80.5	0.0	16.7	77.7	0.0	0.0
4	82.9	33.3	83.3	80.0	0.0	0.0	85.4	50.0	100
5	84.8	0.0	33.3	69.5	0.0	0.0	90.8	50.0	50.0
6	83.8	16.7	33.3	74.8	0.0	0.0	80.6	0.0	50.0
7	76.7	0.0	0.0	63.3	0.0	0.0	76.2	0.0	0.0
8	86.2	0.0	33.3	73.8	16.7	33.3	80.1	50.0	100
a	82.8	8.3	31.3	70.8	2.1	6.3	83.3	25.0	56.3

Let c_{ij} denote the code generated by user u_i for challenge c_j. We generate k codes for that user and denote the set as $\mathsf{chatgpt}_i$. We repeat this process for k users, obtaining $\mathsf{chatgpt}_1, \ldots, \mathsf{chatgpt}_k$. For instance, when fixing the number of sessions of prompting ChatGPT to six—which corresponds to the number of users of ChatGPT—to solve eight challenges, we will end up with 48 ChatGPT codes.

Given sets of code samples generated by ChatGPT and others by actual users (non-ChatGPT), the task then becomes to attribute the codes to their original authors (ChatGPT vs. non-ChatGPT). Given the naive use scenario of ChatGPT we highlighted earlier, we end up with two evaluation cases: granular and holistic authorship attribution.

Granular Attribution. In the granular attribution, and given the above scenario, a code would be considered correctly attributed if labeled in the inference phase with the corresponding ChatGPT label (*e.g.*, $\mathsf{chatgpt}_i$). Let's denote the inference function by inf, which takes a code c and outputs an author label (*e.g.*, u_i). Then, the granular code authorship attribution success rate is computed as the probability (normalized by all inferences) of the inference matching the exact author of the code, which is formulated as:

$$P_r[(u_i \leftarrow \mathsf{inf}(c))|(c = c_{ij})] \text{ for some } j$$

Holistic Attribution. Given that the general purpose of this work is to attribute codes to ChatGPT as a model rather than to a user who has used ChatGPT in a synchronized way, the above assumption can be further relaxed to consider all inference results among the $\mathsf{chatgpt}$ users (*i.e.*, $\mathsf{chatgpt}_i$ for any i) to be correct inferences. For instance, an inference of $\mathsf{chatgpt}_4$ for a code in $\mathsf{chatgpt}_3$ would be considered a correct inference. We formulate that as:

$$P_r[(u_i \leftarrow \mathsf{inf}(c))|(c = c_{rj})] \text{ for any valid } r$$

To conduct code authorship attribution experiments with this scenario and the ChatGPT codes, we created six sets (for GCJ 2017 and 2018) and two sets (for GCJ 2019) of ChatGPT codes (assigned the labels $\mathsf{chatgpt}_1$ through $\mathsf{chatgpt}_6$). We then combined the sets with a dataset that contains 204 non-ChatGPT authors obtained from Google code jam (CGJ) [62]. In total, we have 210 authors (for GCJ 2017 and 2018) and 206 authors (for GCJ 2019). To train an authorship attribution model and examine the performance of detecting ChatGPT using the above scenarios.

Results. The results with Caliskan-Islam *et al.*'s [29] method are shown Table 2 for CGJ 2017 through 2019. The results show the accuracy per challenge (where each challenge corresponds to a fold), and the granular and holistic accuracy for the two evaluation settings highlighted above. We notice that the granular accuracy corresponds to the average per-challenge accuracy in the k-fold-cross validation setting. From these tables, we observe that the overall accuracies of the code authorship attribution experiments, notwithstanding the slight decline due to introducing the ChatGPT codes, are still high, and are around 85%. Moreover, we notice that the code authorship attribution accuracies with a more strict attribution setting (the granular attribution) are only 8.3%, 0%, and 37.5%. Moreover, even when relaxing the authorship attribution to be inclusive with the holistic notion of attribution, the accuracies are still under 30% for GCJ 2017 and 2018 and 75% for GCJ 2019. Both results combined highlight conclusively the insufficiency of the existing approaches for detecting codes authored by ChatGPT.

We also conducted experiments with Abuhamad *et al.*'s [30] method and the outcomes are presented in Table 3 for CGJ 2017 through 2019. These results exhibit similar trends to those observed in the Caliskan-Islam's work. The overall accuracies have decreased by as much as 2% due to the inclusion of ChatGPT codes, although they remain consistently high at an average of 78%. Both the granular and holistic analysis settings underscore the inadequacy of existing methods in detecting code produced by ChatGPT.

Results Interpretation. To further understand the output of the code authorship attribution technique in the pursuit of devising a mechanism to successfully attribute ChatGPT codes, we inspect the codes' labels manually to understand any patterns associated with the misclassification. As highlighted in the partial results depicted in Figure 3, we found that ChatGPT codes from the naive approach may have mixed styles and patterns, causing them to be identified inaccurately as belonging to multiple authors by the code authorship model, rendering the attribution impractical. For example, the non-ChatGPT code authorship model identifies the first code of the first challenge as belonging to author "A1" (non-ChatGPT author). However, for the second and third challenges, it recognizes the first code of each challenge as belonging to authors "A9" and "A3" (non-ChatGPT authors), respectively, as shown in the same figure.

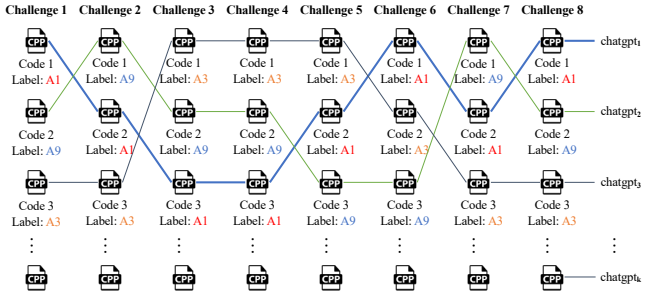


Figure 4: The feature-based approach we used for attributing codes authored by ChatGPT.

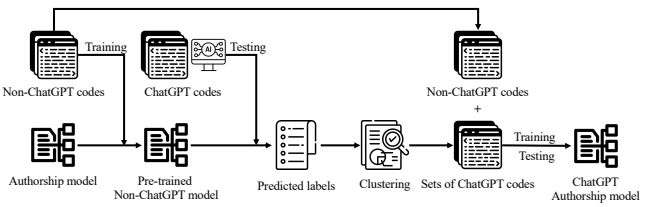


Figure 5: An overview of ChatGPT's code authorship.

Takeaway. As a result, we conclude that even if we create a set with those codes from this synchronized code generation and train a code authorship attribution model, the authorship model cannot reorganize them as belonging to a single author since the model could not capture the common features among those codes, as they seem to resemble the collective features in the non-ChatGPT codes.

3.6 Distilling ChatGPT Labels

To address issues with current methods of identifying the authorship of ChatGPT codes, we propose a "feature-based" approach that exploits the insights we highlighted in understanding the misclassification cases. Our approach extracts stylistic patterns from ChatGPT codes, grouped by their predicted labels from a pretrained code authorship model trained exclusively on nonChatGPT authors— While a jointly trained model can achieve similar results, it would leave some ChatGPT codes out (8.3% in our previous results) by correctly classifying them into the ChatGPT label. This approach allows us to gather ChatGPT codes with similar stylistic features (labels) to group code sets for training a new authorship attribution model. Thus, the model can learn common features from them to accurately classify them once those features are included in the training of the common ChatGPT vs non-ChatGPT code samples.

Justification. The key insight we use as a guiding principle in this design is that it might be easier (computational within the convergence limits or the training epoch time) to train a model to distinguish between two labels that have slight differences to produce accurate inferences of them rather than distinguishing a mixed label from a large number of other labels (*e.g.*, 204 other labels). This insight, indeed, is validated through our experimental work. To evaluate the effectiveness of our approach, we compare it to the naive approach that assumes users will behave in a predictable way (*e.g.*, choosing the first response from AI assistants).

Feature-based Approach (FBA). Rather than naively training a model on both ChatGPT and non-ChatGPT codes together, our feature-based approach begins by training a model solely on non-ChatGPT codes (*e.g.*, 204 authors in the case of the preliminary results). We then use this model to infer labels for the ChatGPT codes, associating them with the original non-ChatGPT labels (*e.g.*, "A1" through "A9" in Figure 4). Upon obtaining those (predicted labels), we group codes that have the same label into the same group and give it a unique identifier (*e.g.*, where codes predicted in the inference step as "A1" are labeled as $\mathsf{chatgpt}_1$, those predicted as "A9" are given the label $\mathsf{chatgpt}_2$). In this grouping, we use the pretrained model with non-ChatGPT codes as a similarity oracle. This similarity oracle will consolidate labels in the generated ChatGPT codes to a smaller set of codes that exhibit the same stylistic characteristics. Given our understanding (based on the failure of the synchronized attribution approach) that ChatGPT is an asynchronous system, the consolidation of codes and reduction are justified.

3.7 ChatGPT's Code Attribution

Now that we have ChatGPT codes grouped into stylistically similar codes using the model pretrained using exclusively non-ChatGPT codes, as shown in Figure 5, we retrain the model to include those grouped ChatGPT codes with their corresponding labels. For clarity, we refer to the non-ChatGPT code labels that were used to group the ChatGPT codes as the target labels. The resulting jointly trained model on ChatGPT and non-ChatGPT codes will be capable of ChatGPT code authorship attribution. In the next section, we evaluate this expectation through granular and holistic attribution, as well as attribution against the target labels.

4 Evaluation and Discussion

To evaluate our feature-based approach in attributing codes generated we start with challenges from GCJ 2017, 2018, and 2019 [62]. We selected eight (8) challenges from each GCJ competition and obtained their respective problem statements, sample inputs, and outputs, as well as any limitations associated with them. The following is an example of one of the challenges from GCJ 2017.

ChatGPT allows the users to input the problem statement and constraints as a prompt and produces codes that solve the problem as deserved. Upon obtaining the solutions, we used the GCJ interface to validate the correctness of the produced solution to that problems—We note that not all codes generated by ChatGPT were successfully validated by GCJ, and we queried ChatGPT a sufficient number of times to produce the required number of solutions to the given challenge using the approach in section 3.5.

Overall, we generated a total of 4,000 codes for GCJ 2017 and 1,600 codes for GCJ 2018 and 2019 by asking questions to ChatGPT (*i.e.*, Write C++ code with this statement and requirements). In the following, we analyzed the stylistic patterns (labels) of ChatGPT codes using a pretrained authorship model trained with non-ChatGPT codes. Then, we made sets of ChatGPT code based on their styles using our feature-based approach to train a new authorship model

Table 4: Datasets. non-ChatGPT is used to train non-ChatGPT code authorship models while vs. human is used for the binary classification (ChatGPT vs. Human). A stands for authors, C for challenges, and T for total.

Dataset	Lang.	non-ChatGPT			ChatGPT			vs. human		
		A	C	T	A	C	T	A	C	T
GCJ 2017	C++	204	8	1,632	8	500	4,000	8	200	3,200
GCJ 2018	C++	204	8	1,632	8	200	1,600	8	200	3,200
GCJ 2019	C++	204	8	1,632	8	200	1,600	8	200	3,200

and examine its accuracy. The accuracy is the percentage of the correctly attributed codes out of the total number of codes in the given labels. We also made sets of ChatGPT code using the naive approach for comparison.

4.1 Experimental Setup and Goal

Experiment Setup. Our experiments were carried out on a workstation running Ubuntu 20.04.5 LTS and using Nvidia RTX A6000 48GB GPU and an Intel Core i7-8700K CPU.

Model Setup. We utilized GPT-3.5 as the GPT model for our experiment, which was the latest freely available version as of May 2023 [1] and two distinct classification models as proposed by Caliskan-Islam *et al.* [29] and Abuhamad *et al.* [30]. For implementation, we used the code provided by Quiring *et al.* [37], [63] and adhered to their setup and hyperparameters to ensure fair results.

Non-ChatGPT Dataset. In this work, we made use of challenge statements from the GCJ 2017, 2018, and 2019. Thus, we created three different non-ChatGPT datasets, one for each GCJ competition, to train the code authorship models for ChatGPT code feature extraction. As we utilized the implemented codes for classification models from [37], we adhered to their dataset configurations, adding two extra datasets as confirmation purposes. Specifically, we obtained the GCJ 2017 as same with [37] and 2018 and 2019 datasets for extra experiments from Github of GCJ dataset [64] and selected 204 authors for each competition, with C++ codes of eight challenges per author. Therefore, each dataset contained a total of 1,632 codes, with each author having eight codes, as shown in Table 4. Given that the goal is to only perform code authorship attribution of ChatGPT codes, we did not see the value in conducting experiments with additional languages besides C++.

ChatGPT Dataset. We used a total of 24 challenges, eight from each year of GCJ 2017, 2018, and 2019. For the challenges from GCJ 2017, we created 500 codes per challenge to examine the diversity of the stylistic patterns of ChatGPT. Moreover, for confirmation purposes, we generated 200 additional codes for each challenge from GCJ 2018 and 2019. Consequently, our dataset for ChatGPT codes includes a total of 4,000 (=8 $\times$ 500) codes for GCJ 2017 and 1,600 (=8 $\times$ 200) codes for GCJ 2018 and 2019, as presented in Table 4. Prompts are crucial when using LLMs, and different prompts may alter the model's response even with the same input. When generating code with ChatGPT, we designed our prompts to reflect the typical users unfamiliar with prompt engineering. We kept the prompts simple, focusing on problem statements, to explore the diverse styles that emerge from straightforward prompts that might be commonly used by users, as following: "Can you write a C++ code with the given problem statement, constraints, and sample input/output?". Moreover, we opted for the web client of ChatGPT instead of the API since typical users are less likely to use the API or adjust parameters.

Binary-classification dataset. We conducted binary classification experiments using the GCJ datasets from 2017, 2018, and 2019 and the ChatGPT codes we generated using GCJ challenges. Each dataset consisted of two distinct classes (labels): ChatGPT and human. Both classes contained 1,600 code samples, spread across 8 challenges with 200 code samples per challenge, resulting in a total of 3,200 code samples. Moreover, we combined these datasets from the three years and carried out an experiment using the combined dataset. However, in this combined dataset, we reduced the number of challenges per year from 8 to 5, resulting in a total of 6,000 code samples. We made this decision in order to achieve a balance in the number of codes within each dataset. If we had retained 8 challenges, the combined dataset would have had a total of 9,600 codes, causing an imbalance when compared to the other datasets. The datasets are in Table 4.

Experiments. Initially, we trained the code authorship model using the non-ChatGPT datasets of GCJ 2017, 2018, and 2019, respectively, to obtain pretrained models. Using these pretrained models, we extracted the features (based on the predicted labels) from each ChatGPT code. We then analyzed the patterns and styles of the ChatGPT codes to observe ChatGPT's stylistic patterns. Next, we created six sets of ChatGPT codes from GCJ 2017 and 2018 and two sets of ChatGPT codes from GCJ 2019 that exhibited similar features (having the same predicted labels) across all eight challenges by utilizing our feature-based approach. For evaluation, we examined the accuracy of code authorship attribution over the combined dataset, the six target authors—those who were used to group the ChatGPT codes—, and our approach. Those results are in addition to the naive approach results shown in section 3.5.

Binary Classification Experiments. We employed the challenge-level accuracy for our feature-based experiments. Initially, we excluded one challenge and train the code authorship attribution model using the remaining challenges. We subsequently tested the trained model using the challenge that had been omitted. Using this approach, we carried out a total of four distinct experiments. The initial three experiments focused on individual GCJ datasets containing ChatGPT codes. The final experiment had the combined dataset and its corresponding ChatGPT codes.

4.2 Results and Discussion

Baseline Attribution. As part of the baseline analysis, we trained and tested Caliskan-Islam *et al.*'s method [29] as a code authorship attribution model with 1,632 codes obtained from 204 non-ChatGPT authors from the GCJ 2017, 2018, and 2019 dataset. Using Caliskan-Islam *et al.*'s method, we attained accuracies rate of 90.43%, 80.5%, and 85.8%, as demonstrated in Table 5 as baselines for the 204 authors.

1. It is worth noting that while a higher version, GPT-4o, exists as of August 2024, the service automatically reverts to a lower model (GPT-3.5) once access to GPT-4o is exhausted.

Table 5: Base: GCJ 2017–2019's accuracy results of the 204 authors (A; baseline) and 6 target authors (T; used to group the ChatGPT codes) from GCJ 2017–2019 non-ChatGPT dataset. 2019 had two target authors. The accuracy is expressed as a percentage. The comparison is for two schemes.

	Caliskan-Islam *et al.*						Abuhamad *et al.*					
	GCJ 2017		**GCJ 2018**		**GCJ 2019**		**GCJ 2017**		**GCJ 2018**		**GCJ 2019**	
c	A	T	A	T	A	T	A	T	A	T	A	T
1	89.2	100	66.7	33.3	91.7	100	85.3	100	52.0	16.7	88.2	50.0
2	89.2	100	79.9	83.3	88.7	100	79.4	83.3	76.0	33.3	89.2	50.0
3	87.7	83.3	89.7	83.3	78.9	0.0	87.7	100	84.8	83.3	80.4	50.0
4	90.2	100	91.2	100	89.7	50.0	84.8	83.3	83.3	66.7	86.3	0.0
5	88.2	100	77.0	50.0	91.7	100	89.2	100	71.6	16.7	90.7	100
6	93.1	100	81.9	66.7	86.3	100	86.8	66.7	74.5	33.3	81.4	50.0
7	91.7	100	79.0	66.7	78.9	50.0	81.4	83.3	66.7	33.3	79.4	50.0
8	94.1	83.3	84.8	66.7	80.9	50.0	85.3	66.7	75.0	16.7	81.9	50.0
a	90.4	95.8	80.5	68.8	85.8	68.8	85.0	85.4	73.0	37.5	84.7	50.0

Table 6: FBA: Caliskan-Islam *et al.*'s method [29] with GCJ 2017–2019's accuracy results. For CGJ 2017 and 2018, 210 authors and 6 target authors, whereas 206 authors and 2 target authors are used for GCJ 2019.

	GCJ 2017				**GCJ 2018**				**GCJ 2019**			
c	A	T	(G)	(H)	A	T	(G)	(H)	A	T	(G)	(H)
1	88.6	100	83.3	100	65.7	33.3	66.7	100	91.7	100	100	100
2	88.1	100	83.3	100	80.0	33.3	66.7	83.3	92.2	50.0	100	100
3	88.1	100	66.7	83.3	87.1	66.7	33.3	66.7	78.6	0.0	100	100
4	87.6	66.7	100	100	88.6	66.7	66.7	100	90.3	0.0	100	100
5	87.6	66.7	66.7	66.7	76.2	50.0	50.0	66.7	93.2	50.0	100	100
6	90.5	66.7	83.3	83.3	80.0	50.0	50.0	50.0	87.9	100	100	100
7	91.0	83.3	83.3	100	71.0	33.3	66.7	66.7	79.6	50.0	100	100
8	93.3	83.3	83.3	100	82.4	50.0	66.7	66.7	78.6	0.0	100	100
a	89.3	83.3	81.3	91.7	78.9	47.9	58.3	75.0	86.5	43.8	100	100

Moreover, we evaluated the accuracy of the model on the six target authors (for GCJ 2017 and 2018) and two target authors (for GCJ 2019) who were employed in the ChatGPT code grouping process. Ideally, we would want the code authorship attribution method to distinguish those target labels from the ChatGPT labels.

We also calculated the baseline accuracy using Abuhamad *et al.*'s approach [30] for the 204 authors, and the results are displayed in Table 5. These findings showcased similar accuracy levels when compared to the results obtained in the Caliskan-Islam *et al.* [29] case, with accuracies of 85%, 73%, and 84.7% for GCJ 2017, 2018, and 2019, respectively. Moreover, we assessed the model's accuracy on six target authors (for GCJ 2017 and 2018) and two target authors (for GCJ 2019) using Abuhamad *et al.*'s method, producing accuracies of 85.4%, 37.5%, and 50%.

ChatGPT Attribution. We first conducted experiments to evaluate the classification accuracy of the jointly trained authorship attribution model on ChatGPT and non-ChatGPT codes by employing Caliskan-Islam *et al.*'s method [29]. In total, the used datasets for this model pertain to 210 authors (six ChatGPT sets and 204 non-ChatGPT authors) for the GCJ 2017 and 2018, and 206 authors (two ChatGPT sets and 204 non-ChatGPT authors) for the GCJ 2019, respectively. The average accuracy rates of these 210-author and 206-author datasets were 89.3%, 78.9%, and 86.5% as shown in Table 6. For the target authors with the GCJ 2017 dataset, the accuracy rate was 83.3%, which is 12.5% lower than the baseline accuracy. In contrast, our feature-based approach yielded an accuracy rate of 81.3%, while the naive approach's accuracy rate was only 8.3% for the granular attribution as explained in section 3.5. Moreover, for holistic accuracy, our evaluation yields 91.7% accuracy, in comparison to 29.2% in the naive case.

Table 7: FBA: Abuhamad *et al.*'s method [30] with GCJ 2017–2019's accuracy results. For CGJ 2017 and 2018, 210 authors and 6 target authors, whereas 206 authors and 2 target authors are used for GCJ 2019.

	GCJ 2017				**GCJ 2018**				**GCJ 2019**			
c	A	T	(G)	(H)	A	T	(G)	(H)	A	T	(G)	(H)
1	85.2	83.3	16.7	33.3	51.0	16.7	50.0	66.7	87.4	50.0	50.0	100
2	78.1	83.3	50.0	83.3	74.8	33.3	33.3	50.0	89.8	50.0	100	100
3	85.7	100	33.3	33.3	80.5	83.3	33.3	50.0	79.1	50.0	100	100
4	82.4	83.3	33.3	33.3	83.8	50.0	33.3	66.7	84.5	0.0	50.0	50.0
5	85.7	100	16.7	16.7	71.0	16.7	33.3	33.3	90.3	100	100	100
6	83.3	50.0	33.3	66.7	76.2	33.3	50.0	83.3	80.1	50.0	0.0	50.0
7	79.0	66.7	33.3	50.0	63.8	33.3	50.0	83.3	78.6	50.0	50.0	50.0
8	83.8	50.0	16.7	33.3	73.8	16.7	50.0	50.0	80.6	0.0	0.0	100
a	82.9	77.1	29.2	43.8	71.8	35.4	41.7	60.4	83.8	43.8	56.3	81.3

We noticed a similar trend when examining the GCJ 2018 dataset. The authors' target accuracy was 47.9%, marking a 20.9% decrease compared to the baseline. Nevertheless, our feature-based method produced higher accuracy rates of 58.3% and 75% for granular and holistic attributions, respectively. In contrast, the naive approach resulted in accuracies of 0% for both granular and holistic attributions. Thus, these improved accuracies represent a substantial enhancement. In the GCJ 2019 dataset experiment, we achieved a perfect accuracy of 100% for both granular and holistic attributions with a 25% target author accuracy decrease. These accuracies represent a substantial improvement, with a 62.5% increase for granular attributions and a 25% increase for holistic attributions compared to the results of the naive experiment, which yielded accuracies of 37.5% and 75%.

We also conducted experiments using Abuhamad *et al.*'s method [30] under identical settings. Overall, their technique yielded lower accuracy rates compared to Caliskan-Islam *et al.*'s, although it outperformed the naive approach. The average accuracy rates across these experiments were 82.9%, 71.8%, and 83.8%, as depicted in Table 7. For the target author using the GCJ 2017 dataset, the accuracy rate stood at 77.1%, which is 8.3% below the baseline. In contrast, our feature-based approach achieved higher scores of 29.2% and 43.8% for granular and holistic attributions, respectively, while the naive approach only produced 8.3% and 31.3%. For the GCJ 2018 dataset, the target author's accuracy reached 35.4%, which is 2.1% lower than the baseline. Our feature-based approach, however, delivered remarkable accuracy rates of 41.7% and 60.4% for granular and holistic attributions, whereas the naive approach only yielded 2.1% and 6.3%. This demonstrates a substantial improvement of 39.6% and 54.1% in accuracy. In the GCJ 2019 dataset experiment, the target author's accuracy rate was 43.8%, which is 6.2% lower than the baseline. In contrast, our feature-based approach achieved impressive accuracies of 56.3% and 81.3% for the granular and holistic attributions, respectively. These figures represent a significant improvement of 31.3% and 25% over the naive approach.

Based on these results, we observe that our method delivers high accuracy in both settings for ChatGPT authorship attribution, outperforming the naive approach. While this comes with a minor degradation in target authors' attri-

```cpp
#include<bits/stdc++.h>
using namespace std;
long long t, n, cnt;
string str;
bool flag;
int main() {
    cin >> t;
    for(int i = 1; i <= t; i++) {
        cin >> n;
        str = to_string(n);
        flag = false;
        for(int j = str.length() - 1; j > 0; j--) {
            if(str[j] < str[j - 1]) {
                flag = true;
                str[j - 1]--;
                for(int k = j; k < str.length(); k++) str
                    [k] = '9';
            }
        }
        if(str[0] == '0') {
            str = str.substr(1, str.length() - 1);
        }
        cout << "Case #" << i << ": " << str << endl;
    }
    return 0;
}
```

(a) Source code generated by ChatGPT in the first time.

```cpp
#include <iostream>
using namespace std;
bool isTidy(long long n) {
    int prev = n % 10;
    n /= 10;
    while (n > 0) {
        int digit = n % 10;
        if (digit > prev) {
            return false;
        }
        prev = digit;
        n /= 10;
    }
    return true;
}
long long findTidy(long long n) {
    while (!isTidy(n)) {
        n--;
    }
    return n;
}
int main() {
    int t;
    cin >> t;
    for (int i = 1; i <= t; i++) {
        long long n;
        cin >> n;
        cout << "Case #" << i << ": " << findTidy(n) <<
            endl;
    }
    return 0;
}
```

(b) Source code generated by ChatGPT in the second time.

Figure 6: Examples of ChatGPT codes under the same problem statement, inputs and outputs, and limitations generated in the first and second time.

bution accuracy, the trade-off is justified by the substantial improvement in ChatGPT attribution.

The Number of Styles. One of the initial observations we made in section 3.5 is that ChatGPT is an agglomeration of styles, causing the limited code authorship success. Given that our ability to detect ChatGPT would be limited by the number of styles it is capable of exhibiting, it is essential to examine this number with rigorous evaluation. To this end, we utilized a dataset consisting of 204 non-ChatGPT authors

Table 8: The number of styles of GCJ 2017 ChatGPT code dataset under different numbers of maximum ChatGPT codes. A total of 8 challenges are used with 500 generated ChatGPT codes per challenge. → highlights the number of styles mapped in the specific challenge, and a stands for the average number of styles.

→	100	200	300	400	500
1	14 (+13)	14 (+0)	15 (+1)	15 (+0)	15 (+0)
2	20 (+19)	23 (+3)	24 (+1)	25 (+1)	25 (+0)
3	12 (+11)	16 (+4)	16 (+0)	16 (+0)	16 (+0)
4	13 (+12)	15 (+2)	16 (+1)	16 (+0)	16 (+0)
5	18 (+17)	19 (+1)	22 (+3)	22 (+0)	22 (+0)
6	17 (+16)	22 (+5)	24 (+2)	25 (+1)	25 (+0)
7	18 (+17)	20 (+2)	22 (+2)	23 (+1)	23 (+0)
8	19 (+18)	24 (+5)	27 (+3)	27 (+0)	27 (+0)
a	15.37	2.75	1.62	0.37	0

Table 9: The number of styles of GCJ 2018 ChatGPT code dataset under different numbers of maximum ChatGPT codes. A total of 8 challenges is used with 200 generated ChatGPT codes per challenge.

GCJ 2018			GCJ 2019		
→	100	200	→	100	200
1	14 (+13)	19 (+5)	1	13 (+12)	14 (+1)
2	35 (+34)	43 (+8)	2	8 (+7)	9 (+1)
3	16 (+15)	20 (+4)	3	10 (+9)	15 (+5)
4	18 (+17)	22 (+4)	4	4 (+3)	6 (+2)
5	29 (+28)	36 (+7)	5	2 (+1)	2 (+0)
6	27 (+26)	36 (+9)	6	5 (+4)	6 (+1)
7	10 (+9)	10 (+0)	7	1 (+0)	2 (+1)
8	16 (+15)	23 (+7)	8	4 (+3)	6 (+2)
a	19.62	5.5	a	4.87	1.62

to train an authorship attribution model, thereby enabling it to identify 204 distinct styles (labels). Using this pretrained authorship model, we evaluated ChatGPT codes to assess their stylistic patterns across each challenge, using the same oracle model used in our feature extraction. We then report the total number of styles (labels) and the discovery of new styles as we increase the number of ChatGPT codes. The latter indicates whether the number of styles reaches a stationary state as we grow the number of codes.

For the initial experiment, we used the 4,000 C++ codes dataset. From Table 8, the maximum number of predicted labels was 27 for challenge 8, while the minimum was 15 for challenge 1, indicating that although there is a limitation, ChatGPT can generate source codes in up to 27 styles. Moreover, the mean number of predicted labels has the highest growth rate at 100 codes, with 15.37 styles on average.

Confirmation. The insight in this experiment is interesting, although it requires confirmation. We used the GCJ 2018 and 2019 ChatGPT dataset comprising 1,600 C++ codes, with 200 codes per challenge, respectively. We used the same method for counting the styles in the initial experiment and found that the maximum number of predicted labels with GCJ 2018 was 43 for challenge 2, while the minimum was 10 for challenge 7, as shown in Table 9.Furthermore, in the context of GCJ 2019, the highest number of predicted labels reached 15 for challenge 3, while the lowest was 2 for challenges 5 and 7, as illustrated in Table 9. Compared to GCJ 2017, ChatGPT produced source codes with more varying styles for GCJ 2018 problems but still fewer than the total number of styles (*i.e.*, 204 styles in the non-ChatGPT codes; *i.e.*,

the maximum possible labels). We found that the average number of predicted labels is slightly more than that in the 2017 dataset. However, both datasets show slowing growth, highlighting the stationary nature of the number of styles as the number of codes grows. In the context of GCJ 2019, compared to GCJ 2017, it is worth noting that ChatGPT generated fewer styles, with the exception of almost one additional style observed for challenges 5 and 7. As a result, these experiments highlight ChatGPT's limited style range.

Stylistic Differences and ChatGPT's Transformations. Figure 6 shows examples of ChatGPT codes for the same question generated in two consecutive times. In the first code (Figure 6a), ChatGPT implemented the program in a single "main" function, while in the second (Figure 6b), the code was divided into three functions: "isTidy", "findTidy", and "main". Due to these stylistic differences, the code authorship attribution methods label the codes differently.

Diversity of ChatGPT Styles. We analyzed how often each label was utilized to examine the variety of coding styles in ChatGPT codes. Our investigation of the GCJ 2017 ChatGPT dataset, as presented in Figure 7, indicates that out of 500 codes, up to four labels (styles) are prevalent. For example, in challenge 1 (shown in Figure 7a), the label "A58" appeared 244 times, while the average number of ChatGPT codes per style is 33.33 (=500/15). Moreover, in challenge 8 (Figure 7h), four labels—"A57", "A77", "A148", and "A202"—appeared 63, 79, 60, and 61 times, respectively, while the average number of codes per style was 18.51 (=500/27). We confirm the same pattern with the GCJ 2018 ChatGPT dataset, as illustrated in Figure 8, revealing that up to three labels are dominant in most cases. For instance, in challenge 3 (Figure 8c), the label "A113" appeared 88 times, while the mean number of codes per label (style) is 10 times (=200/20). Moreover, in challenge 5 (Figure 8e), the labels "A23", "A124", and "A133" appeared 27, 18, and 22 times, respectively, while the average number of ChatGPT codes per style was only 5.55 (=200/36).

In the case of GCJ 2019, more pronounced results are evident, as depicted in Figure 9. For most challenges (2-8), a single label dominates. For instance, in Challenge 2, the label "A39" appeared 120 times, while the average number of codes per style stands at 22 (=200/9). In Challenge 1, four labels, namely "A9", "A39", "A106", and "A196", exhibit similar frequencies, with occurrences of 28, 53, 28, and 50 times, respectively. This is notable given that the mean number of codes per style for Challenge 1 is 14 (=200/14). Thus, we confirmed the same pattern with the GCJ 2017 and 2018 with the GCJ 2019 experiments.

Takeaway. These findings suggest that although ChatGPT has the potential to generate source codes with as many as 27, 43, and 15 different styles, it tends to use only a small number of styles very frequently, making it feasible to classify ChatGPT codes based on their predominant styles with very high accuracy, even when the authorship attribution does not work well for less dominant styles.

Binary Classification. Given the conventional approach for classifying text snippets, which distinguishes between those that belong to the ChatGPT and those that do not (*e.g.*, "GPTZero" [24] and "ZeroGPT" [25]), our classification task naturally can take on a binary form, specifically, ChatGPT versus non-ChatGPT (human) codes, rather than classifying individual authors. Generally, the binary classification accuracy experiments yielded notably high levels of accuracy as shown in Table 10 and Table 11. The lowest accuracy was observed from the individual experiment with GCJ 2018, which is 79.1%. The highest accuracy was achieved in the individual experiment with GCJ 2017, which is 90.1%. In the combined dataset experiment, we obtained average challenge-level and average accuracy metrics, which are listed under 'All.' Additionally, we calculated challenge-level and average accuracy for each dataset separately and provided them under their respective dataset names. Utilizing only five challenges to construct the combined dataset to achieve balance led to a slight deviation in the outcomes. However, these results fell within a similar range as those obtained from the individual experiments, resulting in an average accuracy of 87% when using the combined dataset.

Limitation. Our experiment had some notable limitations that should be taken into consideration. One such limitation was the relatively small number of ChatGPT sets used in the experiment. Specifically, we only observed six labels for GCJ 2017 and 2018 and two labels for GCJ 2019 that appeared across the eight challenges, which led us to create six sets and two sets of ChatGPT codes for the accuracy experiment. Consequently, the experiment results may have been impacted by this small sample size, and further research may be required to validate the findings with a larger number of ChatGPT sets. Another limitation of our experiment was that it was conducted solely on C++ code format, which may not represent other programming languages. Therefore, the results may vary when applied to other programming languages. Future research should consider exploring the effectiveness of our approach in other programming languages to gain a more comprehensive understanding of its applicability.

Future Works. In this work, we conducted experiments to attribute ChatGPT-generated code by employing machine learning- and deep learning-based models. Given that ChatGPT can produce code in various styles and also understand codes, it is essential to consider ChatGPT as well as other LLMs as tools for attribution. LLMs are extensively employed across various tasks, including understanding, explaining, generating, and troubleshooting code, as evidenced by their remarkable performance in programming assignments [65], [66]. Therefore, we need to investigate the potential of LLMs as attribution tools.

5 Conclusion

Despite the many possible good uses of ChatGPT, it poses many risks. Given those risks, there is an apparent need to detect contents generated by ChatGPT. The literature has demonstrated various methods for detecting natural language contents generated by ChatGPT, although ChatGPT code authorship attribution is vastly unexplored. This paper demonstrated that the straightforward, naive approach to utilizing off-the-shelf code authorship attribution is limited and produces poor attribution accuracy. We found that the poor accuracy is a result of the agglomeration of styles exhibited in the codes generated by ChatGPT, and introduce

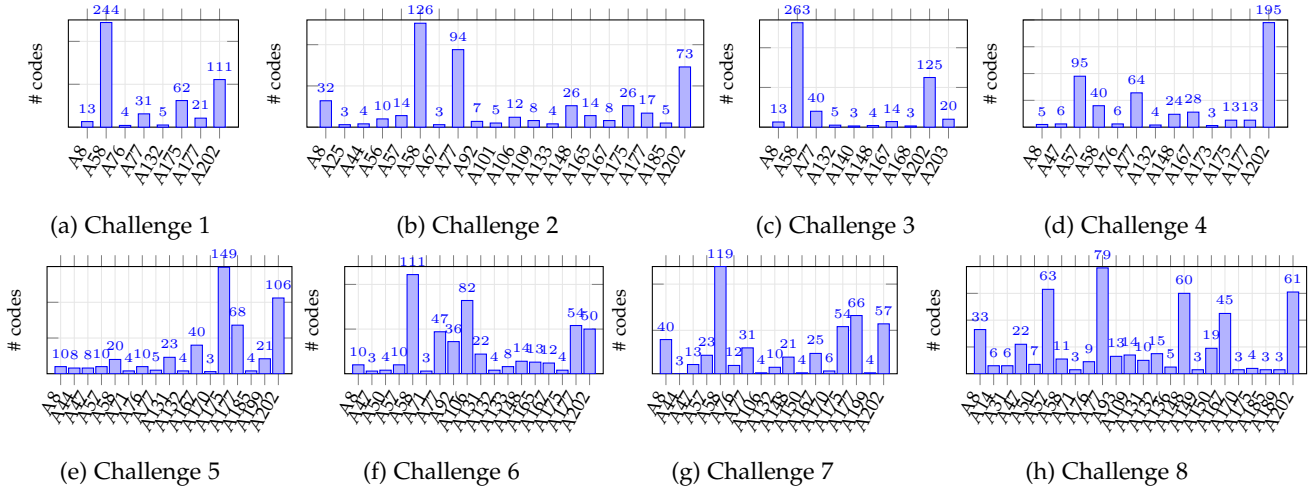


Figure 7: The diversity of ChatGPT styles of eight challenges from CGJ 2017 ChatGPT dataset (a total of 4,000 codes, 500 generated codes per challenge). The x-axis signifies the styles (coded), and the y-axis signifies the number of codes. The figures filter all styles with less than two codes (which were 7, 5, 6, 3, 5, 7, 6, 3 in challenges 1 through 8, respectively).

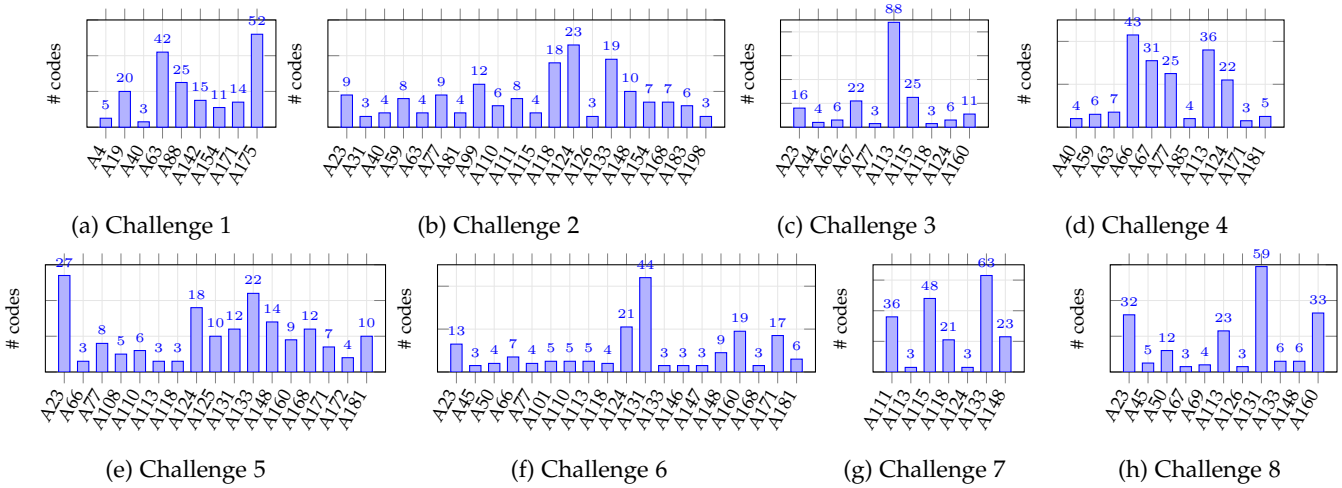


Figure 8: ChatGPT styles diversity of eight challenges from the CGJ 2018 ChatGPT dataset (a total of 1,600 codes, 200 generated codes per challenge). The x-axis signifies the styles (coded) and the y-axis signifies the number of codes. The figures filter all styles with less than two codes (10, 23, 10, 11, 19, 17, 3, and 12 in challenges 1 through 8, respectively).

Table 10: Binary classification accuracy results for GCJ 2017–2019. A model is built and tested for each year individually.

c	2017	2018	2019
1	92.4	60.0	98.5
2	91.6	63.5	99.3
3	96.0	97.0	55.8
4	96.0	76.0	98.0
5	80.3	66.5	97.5
6	89.7	89.5	77.0
7	88.9	93.0	50.0
8	85.5	87.5	91.0
a	90.1	79.1	83.4

Table 11: Binary classification results for GCJ 2017–2019. The model is built using the combined years, and the results reported are for individual year's testing accuracy results (c stands for challenge).

c	2017	2018	2019	All
1	95.5	76.5	97.0	89.7
2	91.8	56.8	92.3	80.3
3	92.8	96.0	85.5	91.4
4	90.3	95.3	97.5	94.3
5	62.0	83.3	93.0	79.4
a	86.5	81.6	93.1	87.0

a novel approach for code authorship attribution by grouping ChatGPT-generated code while capitalizing on their common characteristics using a pretrained authorship attribution model as a similarity oracle. Initially, we analyzed the stylistic features (predicted labels) of ChatGPT code by utilizing a pretrained authorship model that had been trained using non-ChatGPT author dataset. This analysis revealed that ChatGPT tends to generate source codes with various styles but a limited number of styles, with certain styles being more frequently used than others. Following this observation, the ChatGPT codes were grouped into sets based on their labels or features. Using these sets of ChatGPT code, a new code authorship attribution model was developed, which was trained on a dataset including ChatGPT code. The resulting model, specially trained with GCJ 2017, was able to achieve a classification accuracy of 81.3% (granular attribution) and 91.7% (holistic accuracy), which is significantly higher than the accuracy achieved by the naive approach (8.3% and 29.2%, respectively). Furthermore, the authorship model, which was trained for binary

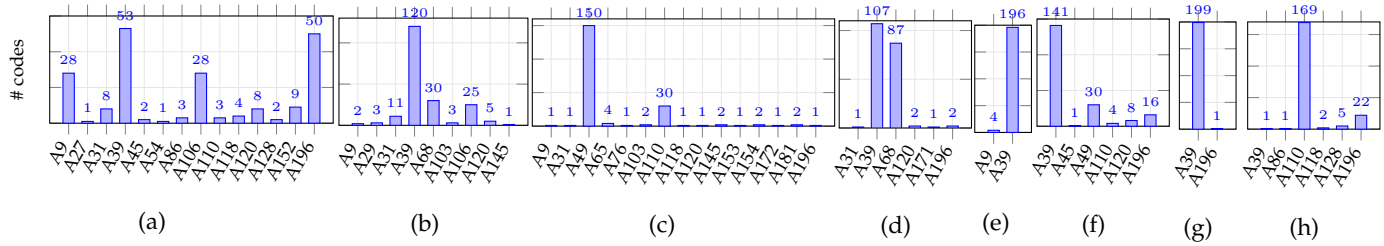


Figure 9: The diversity of ChatGPT styles of eight challenges (9a–9h) from CGJ 2019 ChatGPT dataset (a total of 1,600 codes, 200 generated codes per challenge). The x-axis signifies the styles (coded), and the y-axis is the number of codes.

classification (ChatGPT vs. Human), attained an accuracy of 87% when evaluated using a dataset consisting of 6K code samples. Our findings open a new direction in detecting chatbot-generated codes and can aid in applications where such a feature is desired, including academic misconduct and security analysis.

REFERENCES

[1] —, "ChatGPT," Online, 2023.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *NeurIPS*, 2020.

[3] —, "Document of ChatGPT," Online, 2023.

[4] L. Bishop, "A computer wrote this paper: What chatgpt means for education, research, and writing," *Research, and Writing (January 26, 2023)*, 2023.

[5] S. Biswas, "ChatGPT and the future of medical writing," p. 223312, 2023.

[6] R. Jürgen, T. Samson, and T. Shannon, "ChatGPT: Bullshit spewer or the end of traditional assessments in higher education?" *Journal of Applied Learning and Teaching*, vol. 6, no. 1, 2023.

[7] E. Shimony and O. Tsarfati, "Chatting our way into creating a polymorphic malware," https://shorturl.at/EQbqN, January 2023.

[8] M. Stockley, "Will chatgpt write ransomware? yes." https://shorturl.at/0GpVl, November 2023.

[9] A. Zacharakos, "How hackers can abuse chatgpt to create malware," https://shorturl.at/vyqKM, February 2023.

[10] Y. M. Pa Pa, S. Tanizaki, T. Kou, M. Van Eeten, K. Yoshioka, and T. Matsumoto, "An attacker's dream? exploring the capabilities of chatgpt for developing malware," in *Proceedings of the 16th Cyber Security Experimentation and Test Workshop*, 2023, pp. 10–18.

[11] E. Chatzoglou, G. Karopoulos, G. Kambourakis, and Z. Tsiatsikas, "Bypassing antivirus detection: old-school malware, new tricks," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, 2023, pp. 1–10.

[12] T. Susnjak, "ChatGPT: The End of Online Exam Integrity?" *CoRR*, vol. abs/2212.09292, 2022.

[13] A. Tlili, B. Shehata, M. A. Adarkwah, A. Bozkurt, D. T. Hickey, R. Huang, and B. Agyemang, "What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education," *Smart Learn. Environ.*, vol. 10, no. 1, p. 15, 2023.

[14] B. A. Anders, "Is using ChatGPT cheating, plagiarism, both, neither, or forward thinking?" *Patterns*, vol. 4, no. 3, p. 100694, 2023.

[15] B. D. Lund, T. Wang, N. R. Mannuru, B. Nie, S. Shimray, and Z. Wang, "ChatGPT and a new academic reality: Artificial Intelligence-written research papers and the ethics of the large language models in scholarly publishing," *J. Assoc. Inf. Sci. Technol.*, vol. 74, no. 5, pp. 570–581, 2023.

[16] O. Asare, M. Nagappan, and N. Asokan, "Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code?" *Empir. Softw. Eng.*, vol. 28, no. 6, p. 129, 2023.

[17] R. Khoury, A. R. Avila, J. Brunelle, and B. M. Camara, "How Secure is Code Generated by ChatGPT?" *CoRR*, vol. abs/2304.09655, 2023.

[18] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do Users Write More Insecure Code with AI Assistants?" *CoRR*, vol. abs/2211.03622, 2022.

[19] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," in *IEEE S&P*, 2022, pp. 754–768.

[20] S. Peleg and A. Rosenfeld, "Breaking Substitution Ciphers Using a Relaxation Algorithm," *Commun. ACM*, vol. 22, no. 11, pp. 598–605, 1979.

[21] O. SS, A.-K. AS, and A.-S. DM, "Using genetic algorithm to break a mono-alphabetic substitution cipher," in *IEEE ICOS*, 2010, pp. 63–67.

[22] W. Shahzad, A. B. Siddiqui, and F. A. Khan, "Cryptanalysis of four-rounded DES using binary particleswarm optimization," in *ACM GECCO*, 2009, pp. 2161–2166.

[23] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 14:1–14:39, 2010.

[24] —, "GPTZero," Online, 2023.

[25] ——, "AI Text Detector:ZeroGPT," Online, 2023.

[26] S. Choi, R. Jang, D. Nyang, and D. Mohaisen, "Untargeted code authorship evasion with seq2seq transformation," in *Computational Data and Social Networks - 12th International Conference, CSoNet 2023, Hanoi, Vietnam, December 11-13, 2023, Proceedings*, ser. Lecture Notes in Computer Science, vol. 14479. Springer, 2023, pp. 83–92. [Online]. Available: https://doi.org/10.1007/978-981-97-0669-3_8

[27] M. Abuhamad, T. AbuHmed, D. Mohaisen, and D. Nyang, "Large-scale and robust code authorship identification with deep feature learning," *ACM Trans. Priv. Secur.*, vol. 24, no. 4, pp. 23:1–23:35, 2021. [Online]. Available: https://doi.org/10.1145/3461666

[28] M. Abuhamad, C. Jung, D. Mohaisen, and D. Nyang, "SHIELD: thwarting code authorship attribution," *CoRR*, vol. abs/2304.13255, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.13255

[29] A. C. Islam, R. E. Harang, A. Liu, A. Narayanan, C. R. Voss, F. Yamaguchi, and R. Greenstadt, "De-anonymizing Programmers via Code Stylometry," in *USENIX Security Symposium*, 2015, pp. 255–270.

[30] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang, "Large-Scale and Language-Oblivious Code Authorship Identification," in *ACM CCS*, 2018, pp. 101–114.

[31] Z. Li, Q. G. Chen, C. Chen, Y. Zou, and S. Xu, "RoPGen: Towards Robust Code Authorship Attribution via Automatic Coding Style Transformation," in *IEEE/ACM ICSE*, 2022, pp. 1906–1918.

[32] M. Abuhamad, T. AbuHmed, D. Nyang, and D. Mohaisen, "Multi-χ: Identifying Multiple Authors from Source Code Files," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 3, pp. 25–41, 2020.

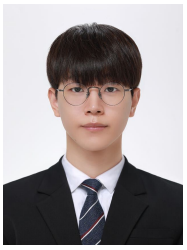
[33] J. Kothari, M. Shevertalov, E. Stehle, and S. Mancoridis, "A Probabilistic Approach to Source Code Authorship Identification," in *IEEE ITNG*, 2007, pp. 243–248.

[34] B. Steven and T. S. MM, "Source code authorship attribution using n-grams," in *Proceedings of the twelth Australasian document computing symposium, Melbourne, Australia, RMIT University*. Citeseer, 2007, pp. 32–39.

[35] V. Kalgutkar, R. Kaur, H. Gonzalez, N. Stakhanova, and A. Matyukhina, "Code Authorship Attribution: Methods and Challenges," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 3:1–3:36, 2019.

[36] M. Abuhamad, J. Rhim, T. AbuHmed, S. Ullah, S. Kang, and D. Nyang, "Code authorship identification using convolutional

neural networks," *Future Gener. Comput. Syst.*, vol. 95, pp. 104–115, 2019.
[37] E. Quiring, A. Maier, and K. Rieck, "Misleading Authorship Attribution of Source Code using Adversarial Learning," in *USENIX Security Symposium*, 2019, pp. 479–496.
[38] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design," *CoRR*, vol. abs/2303.07839, 2023.
[39] —, "GitHubCopilot," Online, 2023.
[40] ——, "OpenAICodex," Online, 2023.
[41] M. Chen, J. Tworek, H. Jun, Q. Yuan *et al.*, "Evaluating Large Language Models Trained on Code," *CoRR*, vol. abs/2107.03374, 2021.
[42] —, "DeepMindAlphaCode," Online, 2023.
[43] Y. Li, D. H. Choi, J. Chung, N. Kushman, J. Schrittwieser *et al.*, "Competition-Level Code Generation with AlphaCode," *CoRR*, vol. abs/2203.07814, 2022.
[44] N. Nguyen and S. Nadi, "An Empirical Evaluation of GitHub Copilot's Code Suggestions," in *IEEE/ACM MSR*, 2022, pp. 1–5.
[45] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather, "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming," in *ACM ACE*, 2022, pp. 10–19.
[46] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs?" *CoRR*, vol. abs/2112.02125, 2021.
[47] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *Comput. Secur.*, vol. 52, pp. 251–266, 2015. [Online]. Available: https://doi.org/10.1016/j.cose.2015.04.001
[48] H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, "Analyzing and detecting emerging internet of things malware: A graph-based approach," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8977–8988, 2019. [Online]. Available: https://doi.org/10.1109/JIOT.2019.2925929
[49] M. Beckerich, L. Plein, and S. Coronado, "Ratgpt: Turning online llms into proxies for malware attacks," *CoRR*, vol. abs/2308.09183, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.09183
[50] K. Jesse, T. Ahmed, P. T. Devanbu, and E. Morgan, "Large Language Models and Simple, Stupid Bugs," in *IEEE/ACM MSR*, 2023, pp. 563–575.
[51] H. Vasconcelos, G. Bansal, A. Fourney, Q. V. Liao, and J. W. Vaughan, "Generation Probabilities Are Not Enough: Exploring the Effectiveness of Uncertainty Highlighting in AI-Powered Code Completions," *CoRR*, vol. abs/2302.07248, 2023.
[52] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, ser. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.
[53] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[54] J. R. Quinlan, "Induction of Decision Trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[55] R. Alec, N. Karthik, S. Tim, and S. Ilya, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf
[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *NeurIPS*, 2017, pp. 5998–6008.
[57] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon, "Unified Language Model Pre-training for Natural Language Understanding and Generation," in *NeurIPS*, 2019, pp. 13 042–13 054.
[58] J. Salazar, D. Liang, T. Q. Nguyen, and K. Kirchhoff, "Masked Language Model Scoring," in *ACL*, 2020, pp. 2699–2712.
[59] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano, "Learning to summarize with human feedback," in *NeurIPS*, 2020.
[60] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep Reinforcement Learning from Human Preferences," in *NeurIPS*, 2017, pp. 4299–4307.
[61] C. Debby RE, C. Peter A, and S. J. Reuben, "Chatting and Cheating: Ensuring academic integrity in the era of ChatGPT," *Innovations in Education and Teaching International*, pp. 1–12, 2023.
[62] —, "Google Code Jam," Online, 2023.
[63] ——, "GitHub/Code-imitator," Online, 2023.
[64] ——, "Google Code Jam Dataset," Online, 2023.
[65] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *ACM MAPS@PLDI*, 2022, pp. 1–10.
[66] J. Leinonen, A. Hellas, S. Sarsa, B. N. Reeves, P. Denny, J. Prather, and B. A. Becker, "Using Large Language Models to Enhance Programming Error Messages," in *ACM SIGCSE*, 2023, pp. 563–569.



Soohyeon Choi received the MSc degree in computer science from South Dakota State University, Brookings, SD, USA, in 2021. He is currently working toward the PhD degree with the Department of Computer Science, University of Central Florida. His research interests include security, machine learning, authorship attribution, and programming language processing.

David Mohaisen (Senior Member, IEEE) received the MSc and PhD degrees from the University of Minnesota in 2012. He is currently a full professor at the University of Central Florida, where he directs the Security and Analytics Lab. From 2015 to 2017, he was an assistant professor at SUNY Buffalo, and from 2012 to 2015, he was a senior research scientist with Verisign Labs. His research interests span networked systems security, online privacy, and measurements. He has been an associate editor for the IEEE Transactions on Mobile Computing, IEEE Transactions on Cloud Computing, IEEE Transactions on Parallel and Distributed Systems, and IEEE Transactions on Dependable and Secure Computing. He is a senior member of ACM (2018) and IEEE (2015), a distinguished speaker of the ACM, and a distinguished visitor of the IEEE Computer Society.