Learning AI Coding Style for Software Plagiarism Detection

Sri Haritha Ambati(✉), Natalia Stakhanova, and Enrico Branca

Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
zus978@usask.ca, natalia@cs.usask.ca

Abstract. Software plagiarism is the reuse of software code without proper attribution and in violation of software licensing agreements or copyright laws. With the popularity of open-source software and the rapid emergence of AI Large Language Models such as ChatGPT and Google Bard, the concerns of plagiarized AI-generated code have been rising. Code attribution has been used to aid in the detection of software plagiarism cases. In this paper, we investigate the authorship of AI-generated code. We analyze the feasibility of code attribution approaches to verify authorship of source code generated by AI-based tools and investigate scenarios when plagiarized AI code can be identified. We perform an attribution analysis of an AI-generated source code on a large sample of programs written by software developers and generated by ChatGPT and Google Bard tools. We believe our work offers valuable insights for both academia and the software development community while contributing to the research in the authorship style of the fast-growing AI conversational models, ChatGPT and Bard.

Keywords: plagiarism detection · code attribution · AI-generated code

1 Introduction

Software packages are no longer built from scratch. They are assembled from open-source and third-party components. Today over 80% of any software code is open source [10]. The rapid increase in open-source software has also given rise to a significant concern: software plagiarism which is reusing someone else's code in a way that violates the terms of its original license.

With the rise of the popularity of open-source software and rapid development in AI, the concerns of safeguarding intellectual property and maintaining the integrity of the open-source software industry became critical. To reduce the manual effort in identifying software plagiarism cases, the community leveraged the code authorship attribution techniques. Code author attribution is rooted in stylometry, a field that originated in literature and is commonly used to determine the author of a disputed text by analyzing their unique linguistic style,

H. Duan et al. (Eds.): SecureComm 2023, LNICST 568, pp. 467–489, 2025.
https://doi.org/10.1007/978-3-031-64954-7_24

including vocabulary, sentence length, and use of verbs. The underlying assumption of stylometric techniques is that authors tend to use consistent linguistic patterns unconsciously, which can be used to identify their literary works.

Instead of analyzing linguistic style, code author attribution aims to determine the developer of a computer program by examining its source or binary code. A program's source code frequently has traits and characteristics that reveal the individual coding style of the developer and can be used to identify them. These peculiarities, or stylistic patterns, might be subtle habits in the usage of syntax and control flow, or they can be as simple as artifacts in comments and code layouts. For example, although using for-loops might be more acceptable, a proponent of grammar would, for example, advocate for while-loops.

Code author attribution approach has strong implications for security. For example, it gives additional support to law enforcement in computer incident investigations [14,32]. It provides alternative means to establish the identity of a programmer suspected of writing malicious code [15,41], software theft [13,31], and other scenarios where software ownership needs to be established.

Numerous studies explored source code attribution focusing on human-generated code [1,3,8,15]. With the rapid development of AI and the emergence of Large Language Models (LLMs) such as ChatGPT and Google Bard, the concerns of plagiarized AI-generated code have been rising. The release of ChatGPT and Google Bard was met with curiosity from the general public, but it also caused a frenzy among practitioners who were concerned about the potential for plagiarism [30]. Many of these concerns are well-founded, as these AI-based LLM tools support software developers in writing code [33] and in automatic bug fixing [26,39].

In this work, we focus on code attribution in the presence of AI-generated code. We examine what constitutes a coding style of AI-generated code and examine the possibility of establishing its authorship. We explore the ability of code attribution approaches to verify authorship of source code generated by AI-based LLMs and investigate scenarios when plagiarized AI code can be identified. Our research is the first of its type to examine human and AI-generated source code attribution accuracy.

In this study, we adopt OpenAI ChatGPT and Google Bard to produce an AI-generated code and explore the effectiveness of five source code attribution approaches, i.e., three state-of-the-art benchmarks studies: Caliskan et al. [15], Ding et al. [18], Kothari et al. [31], and two recently introduced language-oblivious code attribution methods: Zafar et al. [44], Abazari et al. [1].

In our analysis, we use programming solutions from the Google Code Jam, an annual programming competition, where participants are given a set of problems' descriptions and asked to provide solutions. We select a set of programs written by human developers to solve contest problems for our analysis. These problem descriptions and the partial code written by human developers are then used to generate the corresponding programming solutions by ChatGPT [35] and Google Bard for our analysis [23].

Our findings indicate that similar to code written by human developers, AI-generated code can be accurately differentiated from human developers in both small-scale (up to 40 authors) and large (up to 800 authors) scenarios. Yet, while this identification is expected to be attributed to the inherited coding style present in AI tools similar to human developers, our findings show that this is not the case. The AI code attribution is due to consistently similar syntactically and structurally code constructs produced by AI tools and the general inability of AI tools to produce diverse code. This becomes critical in attributing the plagiarized AI code as attribution techniques fail to provide the necessary support. For example, providing code written by humans as an example input to AI distorts the commonly used AI tool's coding constructs leading to less accurate AI authorship attribution. We discuss the challenges we faced and lessons learned in this process. Our research provides critical insights for both software developers looking to enhance their code and the academic community that is concerned about plagiarism.

The rest of this work is organized as follows, Sect. 2 discusses Large Language Models, their uses in software development for code generation, and code attribution. The data collection, processing, and attribution techniques used in our work are outlined in Sect. 4. Section 5 details the technical information of the experiments. Finally, in Sect. 6, we provide a conclusion to our work.

2 Related Work and Background

Large Language Models(LLMs) are machine learning models that use deep learning algorithms to solve a variety of natural language processing (NLP) tasks. OpenAI ChatGPT [35], GPT-2 [38], GPT-3 [12], GitHub Copilot [21] and Google Bard [23] are popular examples of LLM-based tools.

ChatGPT is built on a Generative Pretrained Transformer (GPT-3.5 series) using reinforcement learning from human feedback (RLHF) [16]. It consists of a language model, trained using supervised learning, and a reward model to collect and train using comparison data based on human preferences. The language model is then optimized using reinforcement learning against the reward model [36]. The latest release of ChatGPT is based on GPT 4. Its earlier versions, GPT-2 [38], GPT-3 [12], and GPT-Neo [20,43] models have been evaluated on their capacity to produce testable code from natural language. Abram et al. [25] observed that the models' overall performance is considered below standard even though their ability to produce code is increasing. Weisz et al. [42] investigated whether these imperfect generative models are still beneficial for usage by software developers and concluded that an AI model that produced imperfect output can be considered a useful aid. Although it is conceivable for software developers to use faulty models, it is unclear whether doing so will boost output or alter working methods.

From the viewpoint of a software engineer, ChatGPT is regarded as a convenient tool that can generate software code in response to posed natural language questions. In the context of software development processes, it is an unexplored

model and is prone to potential dangers [11,19]. Recent studies and results suggest that assisting in engineering education [26], software programming [9,19], and testing [39] are some of the key areas in ChatGPT's focus. Although ChatGPT necessitates continual human supervision and interaction, it can deliver well-scripted programming solutions, reducing the time and effort required of a developer [9]. Some studies have concentrated on testing and debugging with ChatGPT in addition to producing the source code [26,39]. As opposed to state-of-the-art automatic bug-fixing solutions, ChatGPT permits a consultation with a software tester for stepwise detection and rectification of defects.

Google Bard [23] utilizes a lightweight model called Language Model for Dialogue Applications (LaMDA). LaMDA [40] is constructed on a transformer architecture, which is a neural network architecture. There are many comparative studies that assess the capabilities of ChatGPT and Bard. The study by Destefanis et al. [17] involved selecting problem statements from five different categories and assessing their correctness. The observations indicate that, in the context of the research, ChatGPT demonstrated relatively stronger performance. However, the authors also note that neither of the models consistently achieved accurate results. Another study [29] compared the ability of both models along with human evaluators, assessing the complexity of the writing prompts. The evaluation standards set by humans were not met by either of the models, leading the authors to propose that they should be used with caution and under human supervision. Despite their shortcoming, both models are extensively used for the generation of code. In this study, we explore the ability of code attribution approaches to verify the authorship of source code generated by AI-based LLMs.

Over the past two decades, code attribution has been at the center of research attention. A comprehensive overview of the existing techniques is provided by Kalagutkar et al. [27]. Initially, the early studies in this area focused on detecting instances of plagiarism and therefore focused on determining the feasibility of source code attribution. The early studies mostly experimented with programming language-dependent features. For example, *Ding et al.* [18] followed a statistical approach to extract features from Java programs by capturing the programming style, layout, and structure. They created a set of 56 features from 40 groups and employed a one-way analysis of variance, significance levels, and Pearson's correlations to exclude irrelevant features. The final attribution was performed with canonical discriminant analysis (CDA). *Caliskan et al.* [15] used a fuzzy parser to parse the source code and produced a feature set containing syntactic features derived from AST along with layout and lexical features. These include characteristics like the typical line length, the branching factor, the proportion of characters with and without white space, the maximum depth of any node in the AST, etc. Term frequencies (TF) and term frequency-inverse document frequency (TF-IDF) statistics for specific terms, AST node bigrams, and other tree elements are also added to the hand-designed features. This yields a significant number of raw features, which are subsequently reduced to a fea-

ture set represented by 120,000 dimensions using an information gain approach embedded within a Random Forest (RF) classifier.

Kothari et al. [31] used entropy to identify the 50 most significant metrics for each author. Utilizing Shannon's information entropy, these metrics in each profile are filtered to find the most effective traits for author recognition using the Voting Feature Interval and the Bayes classifiers.

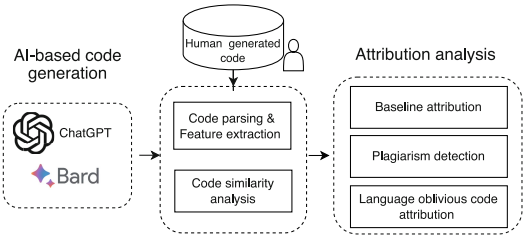


Fig. 1. The flow of the analysis

The recent studies primarily explore language-oblivious source code attribution [1,4,5,44]. For example, the *Zafar et al.* [44] approach represented the source code files as character-level vectors that are passed a Convolutional Neural Network (CNN) framework. This ensures that source codes authored by the same author are close to each other while those written by different programmers are mapped farther apart. The embedding vectors of all source code files are subjected to a k-nearest neighbor classifier that uses Manhattan distance to perform attribution.

Similarly to Zafar et al. [44], *Abazari et al.* [1] ignored language-specific constructs transforming code into a gray-scale image viewed as a 2D matrix. The top consistently occurring n-grams from the matrix are generated using a sliding window approach and filtered for information gain. These selected n-grams are combined to form a 2D matrix that is used to extract spatial patterns representing the author's coding style. This includes Gray-Level Co-occurrence Matrix(GLCM) features and Local Binary Patterns(LBP). GLCM analyzes the frequency of neighboring pixels at selected distances and orientations over the entire image, while LBP measures a local representation of image texture by comparing each pixel with its surrounding neighboring pixels located within a distance d and testing whether the surrounding points are greater or less than the central pixel value are denoted as 1 and others as 0. The resulting values are stored in an 8-bit binary array (or a corresponding decimal number), which is referred to as an LBP value. The attribution is then carried out utilizing RF classification on the obtained patterns.

Besides source code attribution, numerous studies focused on attribution of binary code [6,7,22,28].

3 Learning AI Coding Style

Detecting the authorship of AI-generated code requires an understanding of what constitutes an AI coding style. The challenge in this context is deriving a quantifiable and distinctive coding style, unique to AI tools and easily distinguishable from human software developers. In this analysis, we focus on the characteristics of code generated by AI tools and then explore the ability of code attribution approaches to differentiate AI-generated code. The flow of the analysis is given in Fig. 1.

Table 1. Datasets

Dataset	Year	Number of authors	Number of samples	Samples per author	Avg. LOC per sample
Baseline set	2018–2020	42	292		
Human		40	252	5–8	51.71
AI		2	40	20	51.97
Plagiarized set	2018–2020	42	5,796		
Human		40	252	5–8	51.71
AI		2	5,544	2,774	49.73
GCJ (training set)	2008–2020	2,362	69,549		
Human		2,360	66,991	10–28	65.76
AI		2	2,558	1,279	48.41
GCJ (test set)	2008–2020	42	864		
Human		40	264	5–10	57.26
AI		2	600	300	49.31
GCJ Java	2008–2020	802	31,798		
Human		800	30,548	18–43	58.19
AI		2	1,250	625	51.81
GCJ C	2008–2020	802	11,029		
Human		800	10,071	8–21	52.69
AI		2	958	479	48.25
GCJ Python	2008–2020	802	26,815		
Human		800	25,865	12–31	52.48
AI		2	950	475	50.46

3.1 Data

To provide reliable comparative results, this analysis requires the presence of source code written by both human developers and AI responding to the same coding task. Although several attribution studies employed GitHub open code repository, after manual analysis of repositories we realized that many lacked code descriptions or provided an incomplete or incoherent summary of the problem solved in the corresponding code.

The only viable source of data that provided a reliable and clearly defined problem statement with the corresponding code was the Google Code Jam (GCJ) [24]. GCJ is an annual international programming competition, where participants are given a set of problems and asked to provide solutions within a given time frame. Since the majority of the existing studies employ data

extracted from the GoogleCodeJam programming competition, for our analysis we also assembled the dataset containing code from GCJ competitions with authors that have code written in more than one language. From the GCJ competitions run between 2008 and 2018, we randomly selected 14,518 authors with source code files written in 15 programming languages. We further extracted code from this set written in Java, Python, and C.

The GCJ set was complemented with code samples generated by AI tools producing three distinct sets of data: GCJ Java, GCJ Python, and GCJ C for each programming language. We randomly selected 113 programming task descriptions and requested ChatGPT and Bard to generate code in a specified programming language. This resulted in 3158 AI-generated samples. Note that all including not parsable AI-generated solutions were included in this analysis. We refer to this set as *GCJ set*.

Table 2. Plagiarism scenarios AI-based code generation

Version	Scenario description
V1	Description and first 10% lines of the code
V2	Description and first 20% lines of the code
V3	Description and first 30% lines of the code
V4	Description and first 40% lines of the code
V5	Description and first 50% lines of the code
V6	Description and random 10% lines of the code
V7	Description and random 20% lines of the code
V8	Description and random 30% lines of the code
V9	Description and random 40% lines of the code
V10	Description and random 50% lines of the code
V11	Description and entire code to rewrite

To generate the *GCJ test set*, we randomly selected 40 authors from the resulting GCJ set and 100 programs generated by each AI model for each of the programming languages: Java, Python, and C. The rest of the GCJ was used for training, we refer to it as *GCJ training set*.

We contracted an alternative set referred to as *baseline set* for our baseline experiments. The selected authors participated in the competition during 2018–2020 and wrote code in Java programming language. The resulting set included 20 GCJ programming task descriptions. Similarly, for each of these tasks, we requested AI tools to generate the corresponding solution. To ensure that working code is included in this set, we checked and filtered those that were not parsable by the Java parser. We observed that ChatGPT 3.5 is only capable of producing 3000 words and Bard has even less capacity, i.e., it refused to generate code of more than 200 lines. To maintain a proper analysis and to minimize the

chances of incomplete code generation from both models, we further narrowed the selection of GCJ files to those containing up to 200 lines. We filtered files containing less than 5 lines of code.

The resulting dataset consisted of 252 files from 40 human authors and 40 files from AI tools solving 20 different programming problems. Although the recommended dataset parameters for reliable attribution results are dependent on the feature set, a small pool of 10 authors was shown to be sufficient for code attribution by Abazari et al. [2].

For plagiarism detection analysis, we created an additional *plagiarism dataset* mimicking several plagiarism scenarios listed in Table 2. A portion of each author's code in the original dataset was seeded to ChatGPT and Bard with the corresponding problem description, the generated responses were captured and stored in each dataset under the corresponding name. We used a Java parser on all data sets to ensure that only working solutions were included.

Our Python script reads the program files and measures the line count in each file before selecting the first few lines or a random number of lines depending on the dataset. The lines are selected consecutively for datasets V1-V5, and arbitrarily for datasets V6-V10. These lines are appended to the task description taken from the GCJ competition, as well as two additional statements directing each AI model to utilize the description and code that constitute the final question. The prompt presented to each of these models is a combination of a piece of code chosen sequentially along with the problem statement to provide the problem's solution in Java programming language. With this request, ChatGPT and Bard generated code shown in Fig. 2 are grouped and organized under the author ChatGPT and Bard respectively. The resulting dataset contains code from 40 authors along with the code generated by ChatGPT and Bard.

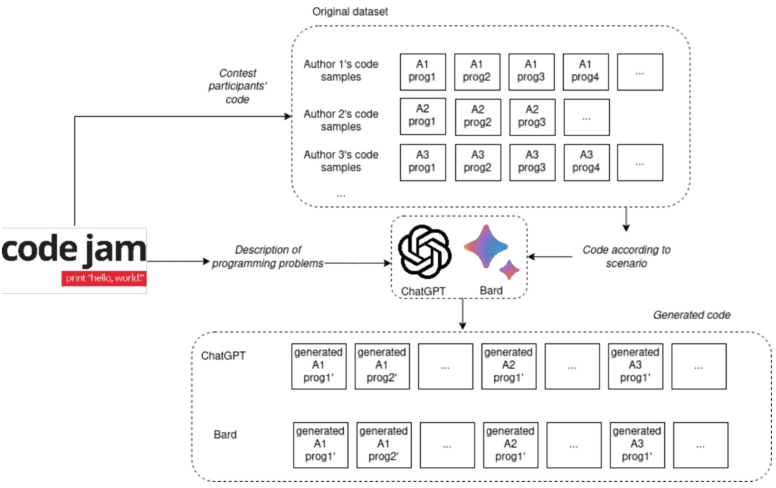


Fig. 2. Plagiarized dataset generation

The generated code versions have solutions generated by both ChatGPT and Bard based on the problem description along with a portion of code from each program of each author.

The details of these datasets are provided in Table 1.

3.2 Data Parsing and Feature Extraction

One of the most critical steps in code attribution is feature engineering, an ability of an approach to accurately and efficiently represent patterns characteristic to a developer across their works. In our study, we adopt five state-of-the-art code attribution approaches developed by Caliskan et al. [15], Zafar et al. [44], Ding et al. [18], Kothari et al. [31], and Abazari et al. [1].

All these approaches focus on source code attribution and employ different feature sets that cover layout (format or layout metrics that describe the appearance of the code), lexical (programming style metrics including the frequency of n-grams), or syntactical (representing the code structure) features. Caliskan et al. [15], Ding et al. [18], and Kothari et al. [31] methods have been widely used and their feature sets are often seen as benchmark feature sets [2]. Zafar et al. [44] and Abazari et al. [1] are the recently proposed language-oblivious and obfuscation-resistant methods. We implemented each method for the attribution analysis.

For each of these approaches, we derive the corresponding feature sets and employ several classification algorithms for attribution analysis.

Following an existing practice in code attribution studies, we selected 6 commonly used machine learning classification algorithms to assess the accuracy of attribution [2]: Random Forest (RF), Decision Tree (DT) Classifier, Extra Trees (XG) Classifier, Neural Network (NN) Classifier, Gaussian Naive Bayes (GNB), and Linear Discriminant Analysis (LDA). RF is a classification technique, composed of a huge number of independent decision trees that interact together as an ensemble. Each individual tree in RF produces a class prediction and the class that is selected by most trees becomes the accuracy of the model. We chose this model as it calculates the importance of each feature and does not overfit when a significant number of features are supplied.

DT is a tree-like structure with each node representing a feature, the branch indicating the decision rule, and the leaves signifying the outcome. We adopted this as it accurately handles heavy input. XG Classifier is a decision tree-based ensemble learning approach. A random subset of the data is used to train each decision tree, and the thresholds for the splitting of each node are chosen at random. It can reduce the variance and overfitting that are naturally observed in ensemble-based learning algorithms.

NN classifiers learn and predict using artificial neural networks that are modeled after the structure and function of the human brain. They are made up of layers of linked neurons that process and transfer information. The input features are passed from one neuron to the next until the final neuron gives the outcome.

When the features acquired are continuous, the GNB algorithm is implemented, and it models the conditional probability of each feature using a Gaussian distribution. It needs less data for training and is quick and efficient. LDA is a classification problem that explores a linear combination of features to differentiate the classes in the input feature set. It is less susceptible to outliers.

Experimental Parameters. We implemented the selected attribution approaches and the analysis system in the Python language. For classification, we used Python modules available in the scikit-learn library [37].

A summary of the classification algorithm parameters is given in Table 3. For hyperparameter optimization, we used the grid search approach. 5-fold cross-validation was employed to measure the accuracy of the machine learning models. It shuffles the dataset randomly and splits into 5 groups, where 4 folds are used for training and the remaining for testing. Hence 5–8 programs for each author are chosen to ensure at least one program is present in all folds for training and testing.

The results are evaluated using a commonly used metric in attribution studies indicating the attribution accuracy or the *accuracy* in short. We used weighted-average accuracy defined as a percentage of code samples correctly attributed to the corresponding authors over the total number of samples, where the accuracy of each class is weighted by the number of samples from that class. All experiments were performed on an Ubuntu 22.04 Operating system equipped with 128 GB of RAM and 24 CPU cores.

Table 3. The parameters of the classification algorithms.

Algorithm	Hyperparameters
RF	n_estimators=100, criterion='entropy', max_features='log2', min_samples_leaf=1, min_samples_split=2
DT	max_depth=100
XG	n_estimators=100, criterion='entropy', max_features='log2', min_samples_leaf=1, min_samples_split=2
NN	max_iter=10000, learning_rate='adaptive', solver='adam', hidden_layer_sizes=(100,)
GNB	var_smoothing = 1e–9
LDA	solver='svd', shrinkage=None

3.3 Similarity Analysis

The characteristics of the generated code might influence the results of the analysis. For instance, identical code samples originating from the same authors can introduce biases in classification, emphasizing repetitive style features and simplifying attribution performance. Code samples may exhibit slight variations

rather than being completely identical. This frequently occurs due to code reuse, where software developers utilize code they have previously written. We investigate the presence of duplicated and similar code in our input datasets.

We use Moss [34], a state-of-the-art code similarity detection system to measure similarity between programs generated for each programming task in our data.

Table 4 shows the characteristics of the AI-generated code compared to code written by human developers. The immediate observation is the diversity of code characteristics of programs produced by human developers and significantly less variability among AI code samples. For example, Bard consistently uses more variables and conditional statements than human developers.

The percentage of similarity among datasets varies significantly. Both AI tools appear to generate *consistently similar* code samples. For example, both

Table 4. Program Characteristics of Baseline dataset

	Number of variables						Number of methods						Number of comments					
	Human Authors		ChatGPT		Bard		Human Authors		ChatGPT		Bard		Human Authors		ChatGPT		Bard	
	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average
Q1	7–28	14.35	8-29	17.45	5–28	12.61	1–8	3	0–11	4.08	0–10	2.06	4–27	10.95	1–30	10.62	1–28	8.37
Q2	2–26	11.23	4–19	11.66	4–26	9.63	0–7	2.35	1–6	2.35	0-8	1.59	1–19	5.46	0–11	5.03	0–27	5.64
Q3	7–21	13.58	7–32	16.48	4–24	14.57	1–11	2.94	1–8	2.78	1–12	2.56	1–48	5.93	1–19	6.63	0–23	6.62
Q4	4–30	13.57	11–19	14.9	3–29	13.02	0–11	3.14	1–3	1.63	0–11	3.15	3–24	11	3–15	8.45	0–16	7.26
Q5	5–21	8.84	4–16	8.36	3–20	8.18	0–8	2.19	0–8	1.99	0–8	1.65	3–48	6.65	1–14	4.28	1–29	6.35
Q6	4–21	8.66	3–30	9.22	3–25	10.22	1–7	1.85	0–10	1.89	0–10	1.98	4–30	8.04	0–25	9.37	0–40	9.96
Q7	14–28	16.9	8–23	16.2	8–25	13.61	0–5	2.18	1–7	2.4	0–7	1.61	2–27	6.8	0–13	5.27	0–31	7.45
Q8	1–24	18.22	4–22	10.97	6–23	16	0–3	2.17	1–6	2.16	1–6	2.91	4–9	5.87	1–8	4.52	6–15	8.58
Q9	6–26	16.1	8–34	15.05	6–37	13.57	0–8	2.25	0–7	1.74	0–8	1.52	1–11	4.69	1–28	5.39	1–24	6.38
Q10	5–33	9.6	5–22	9.76	3–25	9.63	0–8	2.37	0–8	1.72	0–8	1.64	5–33	9.6	3–25	6.56	1–28	9.5
Q11	7–25	15.31	5–24	14.29	5–25	12.98	1–11	4.28	1–9	3.1	1–13	3.27	5–14	8.04	3–29	8.45	1–24	9.26
Q12	4–16	12.56	4–16	13.27	9–17	13.16	1–8	2.37	0–1	0.83	1–8	3.5	3–10	6.71	3–23	7.26	0–14	7.08
Q13	5–26	12.14	16–22	18.17	6–28	14.58	0–4	2.13	1–2	1.33	1–2	1.08	5-15	5.84	4–23	12.67	4–17	8.5
Q14	4–13	6.27	10–19	11.28	10–20	15.67	2–6	2.89	0–8	2.84	1–6	3.33	2–6	4.97	2–17	6.26	1–9	3.83
Q15	5–19	15.19	7–20	13.5	4–20	15.17	0–3	2.17	2–3	2.8	1–5	2.67	1–4	2.47	3–9	6.6	1–13	6.33
Q16	4–15	9.5	4–19	12.16	8–19	14.67	0–7	3.5	1–3	2.18	1–7	3.75	1–5	3	1–21	6.29	3–12	6.17
Q17	4–9	6.43	3–16	10.17	7–14	10.5	0–2	1.26	1–7	3.06	1–3	1.83	2–3	2.81	0–18	7.61	3–15	8.42
Q18	6–14	10.75	5–16	8.22	6–16	10.42	0–1	0.58	2–5	4.12	1–4	2.08	5–16	8.07	1–12	8.24	0–13	6
Q19	4–17	11.58	9–19	13.02	12–21	15.75	1–5	2.97	1–7	3.53	1–4	1.58	2–5	3.59	1–17	7.31	0–10	5.33
Q20	4–15	16.27	5–16	8.27	8–15	12.72	1–4	2.42	2–6	4.91	1–5	2.18	5–13	4.67	2–11	8.26	1–16	5.34

	Number of loops						Number of conditionals						Similar lines of code					
	Human Authors		ChatGPT		Bard		Human Authors		ChatGPT		Bard		Human Authors		ChatGPT		Bard	
	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average	Range	Average
Q1	2–7	4.6	2–12	4.81	1–8	3.86	2–9	5.07	2–10	5.76	1–11	4.04	4–37%	23%	21–69%	41%	19–72%	39%
Q2	3–8	4.56	3–9	5.25	1–10	4.27	0–7	3.4	1–6	2.99	0–10	2.95	10–49%	31%	15–85%	57%	26–69%	42%
Q3	2–16	5.71	2–12	5.24	1–14	5.8	1–7	4.34	2–18	6.28	0–9	3.98	2–35%	26%	5–42%	35%	17–51%	38%
Q4	1–9	3.29	2–6	4.45	0–11	3.15	0–3	1.71	0–5	1.9	0–16	7.26	7–21%	17%	10–71%	41%	21–75%	51%
Q5	1–10	2.79	1–5	2.39	0–16	3.37	0–6	1.74	0–4	1.24	0–9	2.52	6–33%	15%	22–89%	57%	21–83%	48%
Q6	1–14	3.53	0–10	3.42	1–13	4.23	0–8	1.73	0–10	3.57	0–12	3.77	2–15%	9%	12–67%	38%	17–62%	43%
Q7	6–16	7.43	3–13	7.18	2–12	5.71	0–7	1.86	0–8	3.44	0–7	2.34	10–31%	15%	2–19%	12%	7–24%	19%
Q8	1–9	7.6	2–6	4.57	2–9	6	0–3	2.4	0–4	1.97	1–6	2.5	7–42%	24%	17–91%	61%	25–95%	71%
Q9	3–11	7.15	2–13	6.68	2–15	6.31	0–6	2.88	2–7	2.71	0–9	3.24	9–34%	22%	31–98%	70%	42–95%	53%
Q10	2–8	5.33	1–12	5.05	0–12	3.43	0–8	2.03	0–7	1.14	0–10	3.4	4–26%	14%	9–67%	32%	18–57%	36%
Q11	2–7	4.29	1–8	4.22	1–12	4.82	1–7	3.81	1–9	3.46	1–15	4.02	6–47%	21%	7–69%	39%	15–52%	29%
Q12	2–5	3.12	1–5	3.81	3–10	6.5	0–3	1.86	0–13	5.27	0–7	3.58	2–25%	12%	19–82%	46%	23–91%	54%
Q13	2–9	4.23	3–7	4.75	0–9	4.75	0–9	4.17	2–11	6.08	1–13	5.17	3–14%	8%	13–81%	39%	18–87%	42%
Q14	3–5	4.07	3–8	4.31	3–9	5.42	0–4	2.48	0–5	2.7	1–4	2.58	5–23%	12%	12–71%	34%	21–64%	45%
Q15	2–5	3.09	3–6	4.4	1–11	5.5	0–7	2.91	2–6	4.2	1–13	6.42	7–19%	11%	7–59%	28%	12–71%	38%
Q16	2–4	3	2–7	4.06	4–8	5.75	0–5	2.5	1–9	5.14	1–6	3.92	3–24%	16%	24–79%	48%	30–83%	51%
Q17	1–3	2.67	1–9	6.17	3–8	4.75	0–3	2.57	2–6	3.19	2–6	4	4–31%	22%	31–82%	53%	27–73%	48%
Q18	2–8	4	2–6	3.92	3–9	5.25	0–5	2.97	0–9	4.03	1–9	4.67	10–35%	20%	29–75%	47%	24–78%	41%
Q19	2–11	6.47	1–7	4.38	5–13	9	0–7	4.27	2–6	3.61	1–10	6.25	7–15%	9%	37–95%	62%	42–99%	69%
Q20	2–10	5.17	0–8	5.48	0–7	4.58	0–6	3.15	1–7	4.28	2–7	3.79	5–28%	14%	15–61%	37%	24–71%	56%

tools often generate samples that reach 95–99% similarity. This is not the case among human developers, i.e., code between developers and within the same developers rarely reach 49% similarity.

A close manual analysis of AI-generated code confirms our observations. To facilitate diversity, we instructed AI tools to produce "alternative version", "different version" of code, "optimized solution", "using recursion", "using class and objects", "without using loops", "without using conditional statements", "with standard libraries" and "without using standard libraries". For each request, we started a new chat. With each request that forced AI tools to produce different code, their response time gradually increased (sometimes up to 1 h). The results were discouraging. Out of 9 requests to produce different code, ChatGPT produced only 2 code samples that were 14% and 37% similar to the originally generated version, the rest exhibited up to 80% similarity.

Most of the produced AI code incorporates the same code constructs (e.g., loop, data structure use) with the same variable and structure names.

We suspect that the cumulative coding style of all the developers' code was utilized to train the AI LLMs algorithm. According to OpenAI, among other sources, the training data for ChatGPT was sourced from public repositories, and the model was refined through human feedback. As a result, it is highly likely that the code generated by ChatGPT comprises elements used by various developers. Similarly, Bard is also trained on extensive text and code available on the web. Although AI is expected to use elements from other developers, just as software developers commonly use constructs learned/borrowed from other sources, we were surprised to see little flexibility in generating code. Indeed, this finding is another indication of the need for authorship verification of AI-generated code.

Table 5. Benchmark results

Classification Algorithm	Kothari et al. [31]	Zafar et al. [44]	Ding et al. [18]	Caliskan et al. [15]	Abazari et al. [1]
Random Forest	100%	81%	73.3%	63.21%	84.83%
Decision Tree	96.46%	65.38%	72.82%	60.6%	15.35%
Extra Trees Classifier	100%	62.19%	71.97%	56.1%	21.23%
Neural-Network	94.1%	42%	72.3%	41.16%	31.75%
Gaussian Naive Bayes	100%	59.21%	52.72%	48.67%	22.75%
Linear Discriminant	51.74%	30.08%	60.72%	8.7%	14%

4 Attributing AI Code

4.1 Baseline Analysis Results

We performed a series of experiments to assess the attribution accuracy of human-written code across different attribution approaches in order to establish a baseline for our study using the baseline human dataset, i.e., without AI-generated data. Table 5 presents the results of the baseline experiment.

While several algorithms produced comparable results, the RF was more successful in classifying the original human authors and the AI model's code. We therefore only show the results with the Random Forest classifier in the remaining portion of the study. Overall Kothari et al.'s [31] approach was able to achieve the most accurate attribution with 100% accuracy as the features selected by this approach are highly correlated. We decided not to include these results in the paper due to space constraints, but they are still 100% accurate for both human authors and AI-generated code. Followed by Zafar et al.'s [44] method at 81% attribution accuracy, Ding et al.'s [18] feature at 73.3% and Caliskan et al.'s [15] at 63.21%. These results are consistent with the results reported by other attribution methods [1,2].

4.2 Attributing AI Code

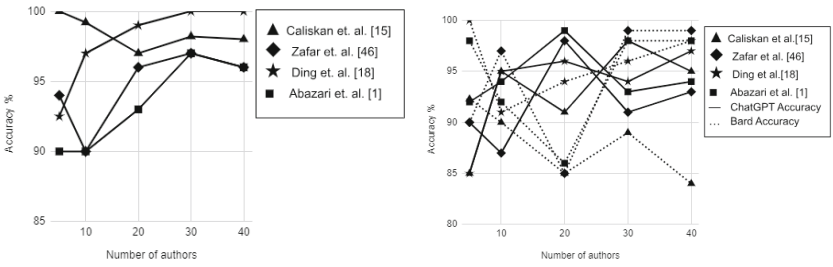
The goal of these experiments is to attribute AI-generated code given only descriptions of the programming problems. This scenario addresses the question of AI code authorship, i.e., whether an AI-based LLM is able to generate code with its own coding style.

Our findings show that the AI code authorship is clearly distinguishable when programs are generated based on an explanation of the programming task without code input from human developers (Fig. 3).

The results show consistently high accuracy across all five attribution approaches. For example, both Ding et al.'s [18] and Calikan et al.'s [15] approaches achieved greater than 88% accuracy regardless of the number of authors in the dataset.

The highest accuracy of 99.98% was achieved with Ding et al.'s [18] method. Similarly, regardless of the quantity of information provided to ChatGPT and the number of authors, Kothari et al.'s [31] features could attribute AI code with 100% accuracy for all datasets[1].

There is a slight difference in performance when smaller numbers of authors are used in experiments. For example, in the ChatGPT case, Zafar et al.'s [44]



(a) Human developers (b) AI tools

Fig. 3. Accuracy of attributing authors, ChatGPT and Bard (Baseline set)

[1] Due to space limitations we decided to omit these results from the paper.

and Abazari et al.'s [1] methods achieved over 70% accuracy for groups of 20, 30, and 40 authors. Zafar et al.'s [44] accuracy dropped to 62.5% with only 5 authors by including ChatGPT responses.

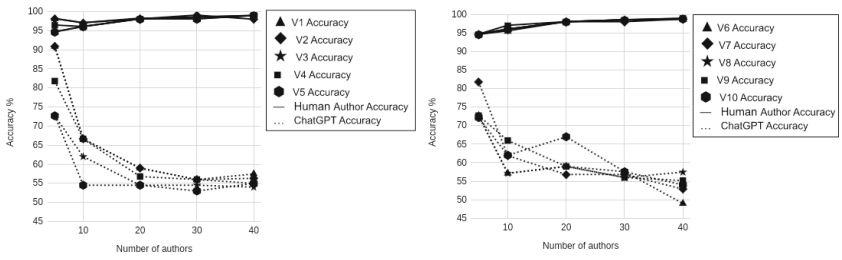
Overall, the results show that it is possible to establish authorship of AI-generated code produced from a simple description of a task and without any accompanying code. In other words, it is feasible to accurately differentiate AI-generated code from code produced by human developers. The code similarity analysis gives more insight into this finding. While the AI code characteristics are different from human code characteristics, the consistency of AI code constructs makes it easier to recognize in any scenario.

5 Plagiarism Detection

It appears from our findings that code attribution approaches are able to characterize AI-generated code and distinguish it from code written by other developers. However, in real-world scenarios, the plagiarized code is often integrated into existing software packages partially or in its entirety or copied with small adjustments. To determine the feasibility of detecting plagiarized AI code, we provide findings from two perspectives:

- Attributing AI-generated code produced based on partial code from human developers (scenarios V1-V10). Since partial code is provided as a seed in these cases, we explore (1) whether it is possible to recognize AI participation in generating the final code sample and (2) whether the random selection of input impacts the detection of plagiarism.
- Attributing AI-generated code produced based on provided complete code (scenario V11). In essence, this scenario aims to answer the question of whether it is possible to attribute a plagiarized AI-generated code if the code initially written by humans is rewritten by AI tools.

These experiments use a plagiarized dataset and their results are shown in Figs. 4, 5, 6 and 7 (Figs. 8, 9, 10 and 11).



(a) (b)

Fig. 4. ChatGPT attribution accuracy using Caliskan et al.'s [15] approach

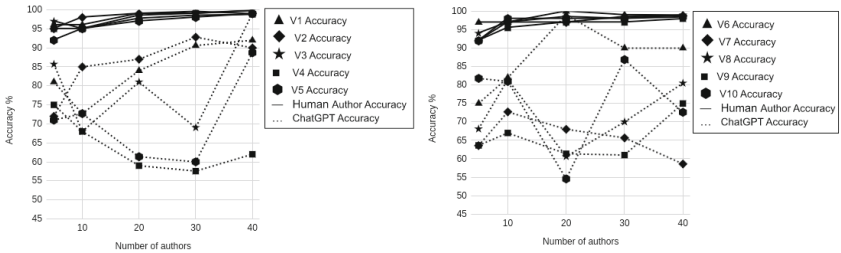


Fig. 5. ChatGPT attribution accuracy using Ding et al.'s [18] approach

Is it Feasible to Establish Authorship of Plagiarized AI Code? As opposed to the results obtained with the baseline dataset, the feasibility of attributing ChatGPT and Bard code in these cases is less evident. The accuracy of code attribution for both AI tools decreases with the increase in the number of authors for all approaches though at a different pace. For example, in the case of ChatGPT code, Caliskan et al.'s [15] approach are able to determine code authorship with 82% accuracy for 5 authors (V4 set), while hardly reaching 55% with 40 authors. The results follow a similar trend for other attribution methods quickly diminishing for 40 authors and generally only reaching 49–55% accuracy.

Significantly less stable and predictable results are obtained with Ding et al.'s [18] feature set for both AI tools, e.g., its ChatGPT accuracy of 82% with 5 authors (V10) plummeted to 55% for 20 authors and then climbed back to 87% for 30 authors. This behavior is similar across other scenarios for ChatGPT. For Bard code, the performance is more stable. Ding et al.'s features' sensitivity to lines of code in the code sample was reported by other studies [2]. Thus, we attribute this fluctuating behavior to the varying lengths of code present in ChatGPT-generated samples. We can observe, for instance, that ChatGPT-generated code has significantly varying lengths (11–200 lines of code).

The other obvious trend in our results is diminishing accuracy in identifying each AI tool as an author, from datasets V1-V5 and V6-V10, which corresponds to the larger amount of original code offered to each AI tool with problem description. The code generated for these datasets appears to blend with the style of the original authors and hence cannot be attributed to AI tools. This is visible in Figs. 4, 5, 6, 7.

Generally, attribution accuracy obtained on V1 and V2 datasets is higher than the performance on V5 data. In other words, providing more code as an example to the AI tool seems to lead to a less pronounced AI coding style, i.e., it becomes more challenging to recognize code as AI-generated. Note that the results with V6-V10 in this regard are less conclusive. It appears that providing more overall, but randomly extracted code is likely to result in a more unified coding style, hence, is more likely to be attributed to an AI tool. We observe that this differs slightly across attribution methods. For example, Caliskan et al.'s [15] approach allows to achieve 66% accuracy when ChatGPT can rely on

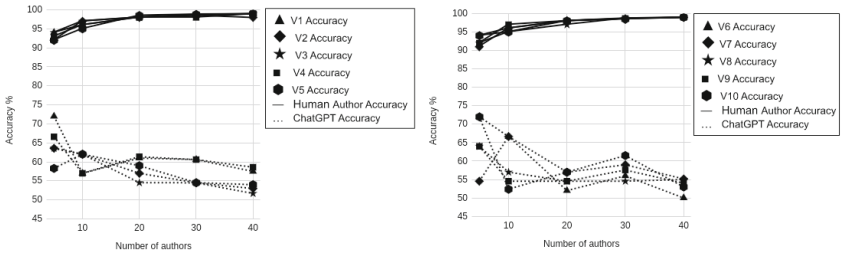


Fig. 6. ChatGPT attribution accuracy using Zafar et al.'s [44] approach

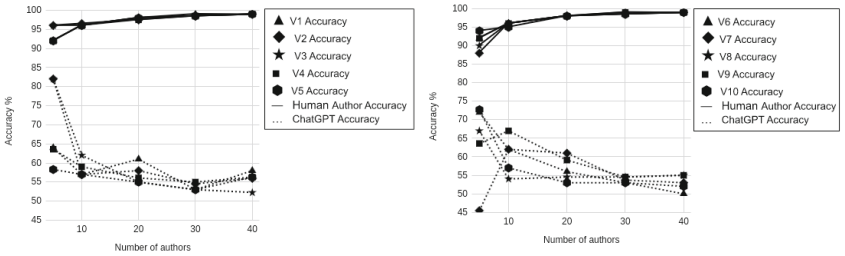


Fig. 7. ChatGPT attribution accuracy using Abazari et al.'s [1] approach

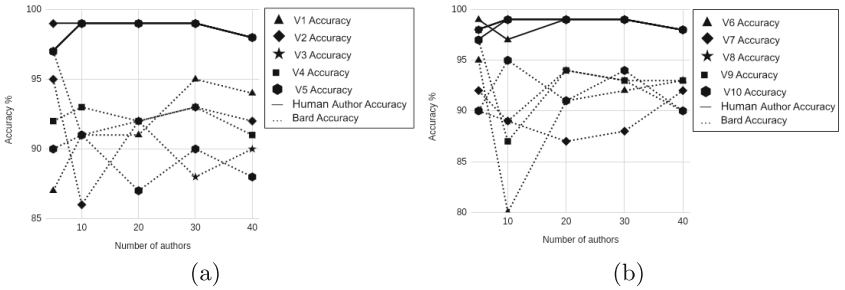


Fig. 8. Bard attribution accuracy using Caliskan et al.'s [15] approach

50% of code written by humans (V10), the accuracy drops to 56% with the input which provides 20% of original code as an example (20 authors, V7).

It is interesting to see how the difference in generated code characteristics is manifested in attribution accuracy. The accuracy of Bard code is generally more consistent and more recognizable compared to ChatGPT code. Detecting ChatGPT use is challenging, i.e., the accuracy does not reach 60% (40 authors) except Ding et al.'s [18] feature set. Bard accuracy stays fairly high, i.e., around 90% in most cases. Hence, in spite of both AI tools producing consistently similar

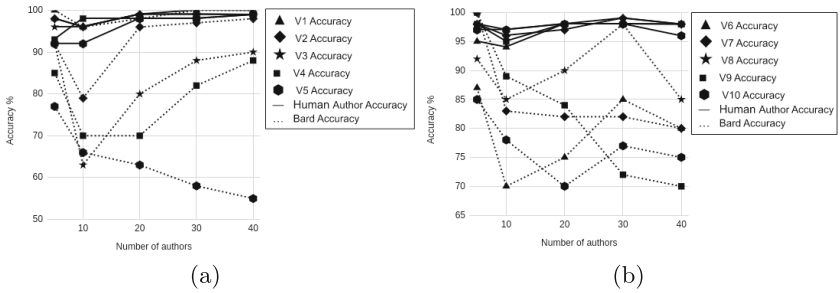


Fig. 9. Bard attribution accuracy using Ding et al.'s [18] approach

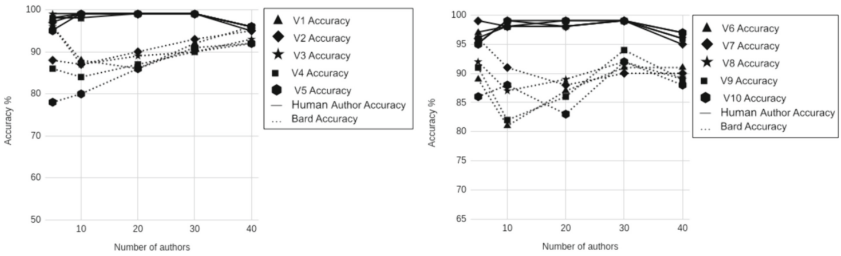


Fig. 10. Bard attribution accuracy using Zafar et al.'s [44] approach

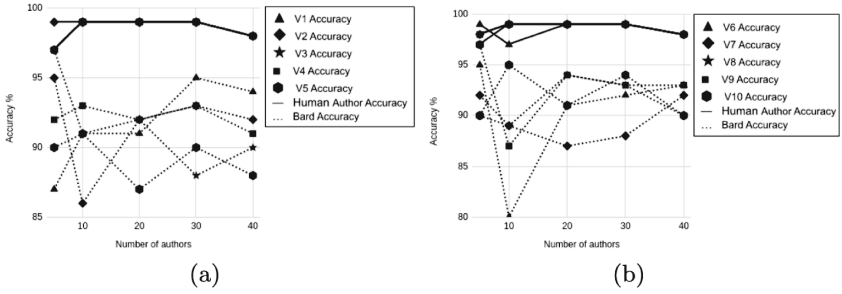


Fig. 11. Bard attribution accuracy using Abazari et al.'s [1] approach

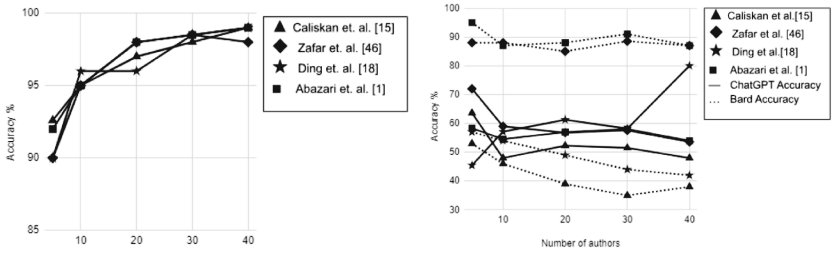
code, a tendency of ChatGPT to produce a slightly diverse code leads to less recognizable code in plagiarism scenarios, consequently making it less detectable.

Overall, these experiments lead to the following conclusions: (1) Providing longer sequential code as an example input distorts the AI tool's coding constructs leading to less accurate AI authorship attribution. This pattern, however, may be altered by offering each tool a random code instead of sequential. (2) Regardless of the amount of input data, the accuracy of attributing ChatGPT-generated code decreases with the increase in the number of authors. (3) Detecting plagiarized code generated by Bard appears to be straightforward for most methods except Ding et al.'s [18]. *(4) The attribution of code written by human developers*

remains consistently accurate across all attribution features, all scenarios, and regardless of the number of analyzed authors.

Is it Possible to Attribute a Rewritten by AI Code? Frequently, in instances of plagiarism, it becomes necessary to modify the original code in order to make it look distinct from its initial form. For these experiments, we utilize the V11 dataset, which was created by supplying a problem description along with the entire source code of each program written by a human developer. It is interesting to note that in these cases, ChatGPT constructed longer files than originally provided.

The attribution results of all five approaches are demonstrated in Fig. 12. It is apparent that human coding style remains the same and is thus easily attributable However, recognizing ChatGPT becomes more challenging as its generated responses attempt to emulate the provided original code hence introducing noise to classification analysis that consequently fails to recognize AI tool. Three approaches, Caliskan et al.'s [15], Zafar et al.'s [44], and Abazari et al.'s [1] show low accuracy from 48–59% with 10 authors to 48%-54% with 40 authors. An exception is Ding et al.'s [18] method that achieves 45% accuracy attributing ChatGPT's responses in the presence of 5 authors and 80% with 40 authors.



(a) Human developers (b) AI tools

Fig. 12. Accuracy of attributing authors, ChatGPT and Bard (scenario V11). Note the results are shown separately for clarity.

Interestingly, Bard code stays mostly consistent with Zafar et al.'s [44] and Abazari et al.'s [1] approach falling slightly below 90% accuracy.

Overall, providing the complete source code for modification by each model results in attribution that is either equally or more inaccurate than instances where only portions of the original code are presented as input examples. Hence, providing code as input to AI tools forces ChatGPT to change its coding style, perhaps aiming to imitate the style of original developers.

6 AI Code Attribution in Real-World

In numerous practical situations, code that necessitates authorship identification or verification is often present in various languages and may exist in incomplete and non-compilable fragments. In order to assess the practicality of AI code identification in real-world scenarios, we designed a series of experiments utilizing the larger GCJ dataset. The attribution methods were trained using a diverse GCJ train dataset encompassing programs generated by human developers and AI tools in multiple programming languages. Subsequently, they were tested on the GCJ test set. Similarly, for individual languages, the methods were trained on subsets of GCJ Java, GCJ Python, and GCJ C sets and tested on the rest of these sets (70%/30% split). In all cases, testing sets deliberately excluded any code samples from human developers or AI tools used during the training phase.

The results are shown in Table 6. The results confirm our initial observations. The AI-generated code is clearly distinguishable from code produced by developers with 95% - 97% accuracy even in the presence of multiple languages. It appears that this approach can be effectively leveraged in real-world scenarios to detect the presence of AI code.

Table 6. Attribution accuracy (Abazari et al. [1])

Dataset	Attribution accuracy	
	Human authors (Aver.)	AI tools
GCJ Java	99.8%	95.7%
GCJ Python	99.8%	97.5%
GCJ C	99.8%	95.6%
GCJ	99.8%	96.2%

7 Lessons and Their Implications

It appears from our findings that both models are able to produce code that follows some coding style that can characterize AI-generated code and distinguish it from code written by other developers. Our experience generating code with AI LLMs revealed several limitations:

- *Incomplete code.* One of the commonly encountered issues is unfinished code. It was common for AI tools to abruptly stop code generation producing inconclusive results. An example is shown in code listing 1.1. Although we have not used these incomplete pieces in our analysis, this limits the use of AI tools in their current state in daily software practices.
- *Incorrect code.* We also observed some minor syntactical and semantic errors such as missing parenthesis, incomplete import statements, and incorrect naming conventions, e.g., using symbols in variable names, class names, using

reserved programming language keywords (e.g., 'this', 'throw') as variable names, and giving only a few solution methods without following the proper structure of the class, and interface definition. This effectively produced not compilable code that required additional correction by a human developer to make code usable. We have not corrected these errors.
- *Limited code size.* OpenAI guidelines suggest that there is a limit of 3000 words on the output generated by ChatGPT. Bard also has a fairly small limit on its responses. In practice, this limits the type of programming tasks that can be produced by AI tools.
- *Inefficient code.* We observed that although the ChatGPT code was functional, it lacked the efficiency present in code written by human programmers. Frequently, ChatGPT produced longer solutions than the original developers. For example, as Table 7 shows programs generated by ChatGPT were longer than the original code written by human developers for the equivalent tasks.
- *Simple solutions.* We observed that both tools had challenges producing solutions for programming tasks with higher complexity. Both tools responded with messages indicating the complexity of the problem and the inability of an AI model to provide a solution to it. We attempted to force the tools to produce code based on their understanding of the task, this only yielded results in some cases, i.e., tools generate a few lines of code.

Table 7. LOC distribution across code samples in the baseline and plagiarized datasets

Average LOC	Baseline Dataset		Plagiarized dataset										
	Human	AI	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
11–20	5	0	0	0	1	1	2	2	0	1	0	2	0
21–30	30	5	33	40	29	21	22	32	31	29	30	22	26
31–40	32	3	53	40	47	55	45	51	46	38	45	60	47
41–50	54	6	67	70	61	54	58	56	69	65	61	58	46
51–60	59	2	51	46	50	44	62	53	51	60	44	46	54
61–70	35	2	26	32	35	40	33	30	29	24	48	35	34
71–80	22	1	10	14	17	23	18	18	16	25	9	13	28
81–90	10	1	7	6	10	13	8	8	6	7	12	12	14
91–100	2	0	3	4	1	0	3	0	3	1	3	4	3
101–200	3	0	2	0	1	1	1	2	1	2	0	0	0
Total	252	20	252	252	252	252	252	252	252	252	252	252	252

- *'TODO'.* Unexpectedly, AI tools often generated solutions with portions of the program lacking implementation. In these cases, the structure of the code portion would be present and marked with a 'TODO' message. In all these cases, we were forced to regenerate code.
- *Similar code.* AI tools produced consistently similar code regardless of the programming task. Our efforts to force tools to generate different programming code were often not fruitful.

Overall, AI tools promise to enhance productivity and improve code quality, yet with their current limitations they are more likely to provide average-quality code.

Listing 1.1. Incomplete AI-generated code

```
public static int[] getCuts(int[] arr, int numCuts, int target) {
    int[] cuts = new int[numCuts + 1];
    int chipsSoFar = 0;
    int chipsForPiece = target * (numCuts + 1);
    for(int i=0; i < arr.length; i++) {
    chipsSoFar += arr[I];
    if(chipsSoFar == chipsForPiece) {
        cuts[0] = -1;
        break;
    }
```

8 Conclusion

Code attribution is critical in many security contexts. The demands of code plagiarism detection and attribution of intellectual property theft raise questions about the adequacy of existing techniques in offering the necessary support. In this research, we investigated the ability of code attribution techniques to verify the authorship of source code generated by the AI-based LLMs and to reveal cases of code plagiarism. Our work provides a practical roadmap for the verification of code authorship in real-world scenarios.

References

1. Abazari, F., Branca, E., Novikova, E., Stakhanova, N.: Language and platform independent attribution of heterogeneous code. In: Li, F., Liang, K., Lin, Z., Katsikas, S.K. (eds.) SecureComm 2022. LNCS, vol. 462, pp. 173–191. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-25538-0_10
2. Abazari, F., Branca, E., Ridley, N., Stakhanova, N., Dalla Preda, M.: Dataset characteristics for reliable code authorship attribution. IEEE Trans. Dependable Secure Comput. **20**(1), 506–521 (2023)
3. Abuhamad, M., AbuHmed, T., Mohaisen, A., Nyang, D.: Large-scale and language-oblivious code authorship identification. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, pp. 101–114. Association for Computing Machinery, New York (2018)
4. Abuhamad, M., AbuHmed, T., Mohaisen, A., Nyang, D.: Large-scale and language-oblivious code authorship identification. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 101–114 (2018)
5. Abuhamad, M., Rhim, J.S., AbuHmed, T., Ullah, S., Kang, S., Nyang, D.: Code authorship identification using convolutional neural networks. Futur. Gener. Comput. Syst. **95**, 104–115 (2019)
6. Alrabaee, S., Debbabi, M., Wang, L.: CPA: accurate cross-platform binary authorship characterization using lda. IEEE Trans. Inf. Forensics Secur. **15**, 3051–3066 (2020)

7. Alrabaee, S., Karbab, E.M.B., Wang, L., Debbabi, M.: BinEye: towards efficient binary authorship characterization using deep learning. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11736, pp. 47–67. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29962-0_3
8. Alsulami, B., Dauber, E., Harang, R., Mancoridis, S., Greenstadt, R.: Source code authorship attribution using long short-term memory based networks. In: Computer Security - ESORICS 2017, pp. 65–82 (2017)
9. Avila-Chauvet, L., Mejía, D., Acosta Quiroz, C.O.: Chatgpt as a support tool for online behavioral task programming. SSRN 4329020 (2023)
10. BlackDuck: 2017 open source security and risk analysis report. Technical report, BlackDuck Inc., 800 District Ave., Suite 201 Burlington, MA 01803-5061 (2017)
11. Borji, A.: A categorical archive of chatgpt failures (2023)
12. Brown, T., et al.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020)
13. Burrows, S., Tahaghoghi, S.: Source code authorship attribution using n-grams. In: ADCS 2007 - Proceedings of the Twelfth Australasian Document Computing Symposium (2007)
14. Burrows, S., Uitdenbogerd, A.L., Turpin, A.: Comparing techniques for authorship attribution of source code. Softw. Pract. Exp. **44**(1), 1–32 (2014)
15. Caliskan-Islam, A., Harang, R., Liu, A., Narayanan, A., Voss, C., Yamaguchi, F., Greenstadt, R.: De-anonymizing programmers via code stylometry. In: 24th USENIX Security Symposium (USENIX Security 2015), pp. 255–270. USENIX Association, Washington, D.C. (2015)
16. Christiano, P.F., Leike, J., Brown, T., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017)
17. Destefanis, G., Bartolucci, S., Ortu, M.: A preliminary analysis on the code generation capabilities of gpt-3.5 and bard ai models for java functions (2023)
18. Ding, H., Samadzadeh, M.H.: Extraction of java program fingerprints for software authorship identification. J. Syst. Softw. **72**, 49–57 (2004)
19. Doglio, F.: The rise of chatgpt and the fall of the software developer - is this the beginning of the end? (2023). https://tinyurl.com/3mxrfmjh
20. Gao, L., et al.: The pile: an 800 gb dataset of diverse text for language modeling. arXiv preprint arXiv:2101.00027 (2020)
21. GitHub: Github copilot blog (2023). https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/
22. Gonzalez, H., Stakhanova, N., Ghorbani, A.A.: Authorship attribution of android apps. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, pp. 277–286. Association for Computing Machinery, New York (2018)
23. Google: Bard chat (version 2023). https://bard.google.com/. Accessed 21 Mar 2023
24. Google: Google code jam archive (2023). https://codingcompetitions.withgoogle.com/codejam/archive. Accessed Mar 2023
25. Hindle, A., Barr, E.T., Su, Z., Gabel, M., Devanbu, P.: On the naturalness of software. In: Proceedings of the 34th International Conference on Software Engineering, ICSE 2012, pp. 837–847. IEEE Press (2012)
26. Jalil, S., Rafi, S., LaToza, T.D., Moran, K., Lam, W.: Chatgpt and software testing education: Promises & perils (2023)

27. Kalgutkar, V., Kaur, R., Gonzalez, H., Stakhanova, N., Matyukhina, A.: Code authorship attribution: methods and challenges. ACM Comput. Surv. (CSUR) **52**(1), 1–36 (2019)
28. Kalgutkar, V., Stakhanova, N., Cook, P., Matyukhina, A.: Android authorship attribution through string analysis. In: Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, pp. 1–10. Association for Computing Machinery, New York (2018)
29. Khademi, A.: Can ChatGPT and bard generate aligned assessment items? a reliability analysis against human performance, **6**(1) (2023)
30. Khalil, M., Er, E.: Will chatgpt get you caught? rethinking of plagiarism detection. arXiv preprint arXiv:2302.04335 (2023)
31. Kothari, J., Shevertalov, M., Stehle, E., Mancoridis, S.: A probabilistic approach to source code authorship identification. In: Fourth International Conference on Information Technology (ITNG 2007), pp. 243–248 (2007)
32. Lange, R.C., Mancoridis, S.: Using code metric histograms and genetic algorithms to perform author identification for software forensics. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO 2007, pp. 2082–2089. Association for Computing Machinery, New York (2007)
33. Liu, J., Xia, C.S., Wang, Y., Zhang, L.: Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation (2023)
34. Moss: A system for detecting software plagiarism (2023). https://theory.stanford.edu/~aiken/moss/
35. OpenAI: Openai blog (version 2022). https://openai.com/blog/chatgpt. Accessed 13 Feb 2023
36. Ouyang, L., et al.: Training language models to follow instructions with human feedback. Adv. Neural. Inf. Process. Syst. **35**, 27730–27744 (2022)
37. Pedregosa, F., et al.: Scikit-learn: machine learning in Python (2011)
38. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)
39. Sobania, D., Briesch, M., Hanna, C., Petke, J.: An analysis of the automatic bug fixing performance of Chatgpt. In: IEEE/ACM International Conference on Software Engineering (2023)
40. Thoppilan, R., et al.: Lamda: language models for dialog applications (2022)
41. Ullah, F., Wang, J., Jabbar, S., Al-Turjman, F., Alazab, M.: Source code authorship attribution using hybrid approach of program dependence graph and deep learning model. IEEE Access **7**, 141987–141999 (2019)
42. Weisz, J.D., et al.: Perfection not required? human-AI partnerships in code translation. In: 26th International Conference on Intelligent User Interfaces, pp. 402–412 (2021)
43. Xu, F.F., Alon, U., Neubig, G., Hellendoorn, V.J.: A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, pp. 1–10 (2022)
44. Zafar, S., Sarwar, M.U., Salem, S., Malik, M.Z.: Language and obfuscation oblivious source code authorship attribution. IEEE Access **8**, 197581–197596 (2020)