Detecting ChatGPT-Generated Code in a CS1 Course

Muntasir Hoq[1], Yang Shi[1], Juho Leinonen[2], Damilola Babalola[1], Collin Lynch[1] and Bita Akram[1]

[1]North Carolina State University, United States
[2]The University of Auckland, New Zealand

Abstract

The emergence of ChatGPT has raised concerns about students potentially using it for cheating. Computer Science (CS) educators are becoming worried because of the potential short and long-term adverse effects it might have on students. However, it is unclear to what extent ChatGPT-generated code can be distinguished from student-written code in introductory programming courses. In this work, we analyze how well student-written and ChatGPT-generated code can be automatically distinguished. We use an openly available dataset of student program solutions for CS1 assignments and have ChatGPT generate code for the same assignments. We evaluate the performance of both traditional machine learning models, such as SVM and XGBoost, as well as Abstract Syntax Tree-based deep learning models, such as code2vec and ASTNN, in distinguishing between student and ChatGPT code. The results suggest that both traditional machine learning models and AST-based deep learning models can be very effective in detecting whether a student or ChatGPT wrote the code in an educational setting with accuracies higher than 90%.

Keywords

ChatGPT, large language model, student programming code analysis, introductory programming course, plagiarism detection, ethical consideration of LLMs

1. Introduction

Plagiarism is a common problem in introductory programming courses [1]. Previous work has found, for example, that students might resort to plagiarism due to struggling [2] and might be confused about what constitutes plagiarism in programming [3, 4, 5]. In the context of programming, plagiarism can take various forms, such as copying code from the internet (e.g., StackOverflow), sharing solutions between students, and contract cheating.

Recently, a new possible type of plagiarism has emerged: using powerful, large language model-based AI models and tools such as ChatGPT[1] and GitHub Copilot[2] to create solutions to programming exercises. While these tools might help professional programmers develop code more efficiently[3] and can be used by instructors to create educational resoruces [6, 7], programming educators have raised concerns around potential student over-reliance on these

AIED2023 Empowering Education with LLMs - the Next-Gen Interface and Content Generation Workshop, June 03–07, 2023, Tokyo, Japan
mhoq@ncsu.edu (M. Hoq); yshi26@ncsu.edu (Y. Shi); juho.leinonen@auckland.ac.nz (J. Leinonen); djbabalo@ncsu.edu (D. Babalola); cflynch@ncsu.edu (C. Lynch); bakram@ncsu.edu (B. Akram)

[1]https://openai.com/blog/chatgpt
[2]https://github.com/features/copilot
[3]https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

models [8]. Students using such models without attributing the created code to the model might be considered a new type of plagiarism. Recent work has found that up to 80% of introductory programming problems can be successfully solved by state-of-the-art AI models [9, 10] and that this performance is better than the performance of average students [10]. Similar performance has been observed for more complex data structures and algorithms-level exercises [11].

One challenge related to plagiarism or student over-reliance on AI models is that it might be difficult to detect. Programming plagiarism detection tools that have been traditionally used in introductory programming courses such as MOSS[4] and JPlag [12] are based on comparing student submissions to each other. However, recent AI models do not work deterministically, and thus traditional methods for detecting plagiarism in introductory programming might be infeasible for detecting AI-created code. Some recent efforts aim to detect AI-generated content using AI models, expecting to mitigate the uncertainties of advanced AI. However, they only achieve mediocre performance that is far from adaptable in educational use [13] when applied to essays, and they are not specific to programming education. While there are methods that focus on the process of writing code [2, 14, 15], these require tailored IDEs and have thus not been widely adopted.

In this work, we study the automatic detection of ChatGPT-created programs for introductory programming exercises. We use a publicly available dataset of student-written programs in a CS1 course[5] and use ChatGPT to create programs for the same exercises. We evaluate different classification methods for identifying code as AI-generated or student-written. Our research question is: "How well can ChatGPT-created programs be distinguished from student-created programs in CS1 courses?"

2. Background

Cheating and plagiarism are more common issues in introductory-level courses than in higher-level courses, i.e., ones attended by graduate students [16]. Studies have shown that cheating is common among struggling students, aided with wide tolerance despite awareness of university policy [16, 17]. Moreover, the rise of online courses made the scenario even worse [18, 19].

As artificial intelligence and natural language processing technologies have advanced, distinguishing between human and machine-generated text has become increasingly important. Several academic studies have proposed several methods to detect machine-generated text. In a previous study [20], a method for detecting computer-generated text (CGT) in academic papers was proposed. The method involves extracting topic-related text features, stemming tokens to avoid confusion, and scoring features. These features were used to train a nearest neighbor classifier binary classification model due to its simplicity on small datasets. The methodology was evaluated using various datasets and demonstrated high accuracy in detecting CGT in different domains, such as online reviews, social media posts, and scientific articles. For detecting programming plagiarism, Huang et al. [21] introduced a novel code plagiarism detection approach using the XGBoost incremental learning algorithm. The method involves extracting 11 code features through static analysis, filtering out weaker features, and training an

[4] https://theory.stanford.edu/~aiken/moss/
[5] https://codeworkout.cs.vt.edu/

XGBoost model to identify plagiarism. The approach demonstrated high accuracy in detecting code plagiarism, surpassing existing techniques. It is proposed as an effective tool for plagiarism detection in academic and software industry scenarios. Kechao et al. [22] proposed a student program plagiarism detection approach using the CloSpan data mining algorithm. It mines comparable code segments, computes similarities between programs, and generates a plagiarism report. Experimental results showed improved precision and detection efficiency compared to the MOSS tool, providing more detailed statistical information and visualizing comparable code fragments. However, with the rise of the ChatGPT model, these methods need to be at least validated, as the similarity-based methods may not apply in cases when the code is automatically generated as compared with copied from other students or resources.

3. Method

3.1. Dataset

We use the student-written code from a publicly available dataset obtained from the CodeWorkout[6] platform. The CodeWorkout dataset contains student code from an introductory programming course in Java. There are 50 programming problems. The programs are graded with the ratio of test case passing. We use the first 10 problems from the Spring 2019 semester in our experiment. Uncompilable submissions are removed from the dataset as uncompilable code can not be parsed into ASTs. Incorrect submissions are also removed from the dataset since the intermediate states of a problem-solving process. Moreover, ChatGPT correctly solves programming problems from introductory programming courses, and we do not want our detection process biased towards detecting correct and incorrect codes. The programming problems include different introductory Java programming concepts, such as methods, variable declaration, data types, conditionals, strings, etc. Students in this course were given a prompt with the problem statement for each assignment with a function prototype.

3.2. ChatGPT-generated Code

A dataset comprising programming code generated by ChatGPT is created for the purpose of this study. To create this dataset, we present ChatGPT with the problem statements of the first ten problems. We used the GUI to interact with ChatGPT (March 2023). We also provide the ChatGPT prompt with the function prototype given to the students with the problem statement in the CodeWorkout platform to ensure that ChatGPT has the same information as students had when they were constructing their programs. In order to maintain uniformity and balance in the dataset, we collect 300 ChatGPT-generated codes for each assignment, which are used for comparison with the student-written codes. The correctness of ChatGPT-generated code is manually examined to ensure that we only include the correct programs. To generate each instance of a solution to a specific problem, we regenerate the response of ChatGPT to get different solutions. The characteristics of both datasets are provided in Table 1.

[6] https://codeworkout.cs.vt.edu/

Table 1
Dataset properties

Dataset	CodeWorkout	ChatGPT
Language	Java	Java
# programs	3162	3000
# problems	10	10
Class	1	0
min code length	4	3
max code length	83	27
mean code length	17	10

3.3. Automatically Distinguishing ChatGPT and student code

To detect student-written codes and ChatGPT-generated code, we use both different traditional machine learning techniques and more recent neural methods. The traditional methods include Support Vector Machine (SVM) [23, 24, 25] and Extreme Gradient Boosting (XGBoost) [21, 24, 25]. The more recent methods include code2vec [26], and ASTNN [27].

code2vec [26] is an attention-based neural network model that uses a code representation technique to learn vector embeddings for programming code tokens, such as function and variable names and their surrounding code context. The model was first introduced for predicting method names, given their implementation for analyzing professionally-written Java codes, and is also known for representing programming code information in educational domains such as bug detection [28, 29], performance prediction [30, 31] and skill representation [32, 33].

The code2vec model comprises two encoders and one attention mechanism: a path-based encoder and an attention-based classifier. The path-based encoder takes a code snippet (represented as an abstract syntax tree, AST) as input and represents it as a set of paths connecting two leaf nodes. These paths are then encoded into a fixed-length vector with the attention mechanism by calculating the importance of different paths. It allows for exploring the most important paths and structures of the programming code, which provides insights into understanding the associated task. Finally, the model learns to predict the label of the code snippet from the code vectors. In this study, we use code2vec to classify programming codes into student-written codes and ChatGPT-generated codes based on their learned vector embeddings.

ASTNN [27] is an Abstract Syntax Tree-based Neural Network that uses an AST-based neural network approach to code classification. The model takes the Abstract Syntax Tree (AST) of a code snippet as input and generates a vector representation of the code's structure. It has demonstrated strong performance in various code classification tasks, including code correctness prediction, detecting code patterns, identifying plagiarism, and detecting code clones [27, 34, 35, 36, 37, 38]. The basic idea is to split an AST into different statement trees and generate vectorial code representations. These statement trees are then embedded into statement tree vectors, and a bidirectional recurrent neural network is used to encode the statements with their sequential dependency. Further feature extraction layers then process the embeddings before the final identification on student-written or ChatGPT-generated code. ASTNN is designed to capture the high-level structure of the code and is, therefore, well-suited

for classifying code snippets. We also developed an ASTNN model for the AI code identification task in our experiments for comparisons.

4. Experiments

Experiments are designed to distinguish between student-written codes and ChatGPT-generated codes. To detect the author of a code, we perform a binary classification task, where we identify if a piece of code is being written by a student (class 1) or generated using ChatGPT (class 0). We use accuracy, precision, recall, and F1-score as the evaluation metrics. Utilizing a variety of evaluation metrics enables a complete understanding of model strengths and weaknesses [39].

We also use different traditional Machine Learning models, including SVM and XGBoost, as the baselines to compare with the AST-based deep learning models. For these models, we use TFIDF [31] to embed and vectorize the programming codes for the inputs of these traditional ML models. We use 10-fold cross-validation to tune the hyperparameters of the traditional ML models. For SVM, the kernel is set to '*poly*' from the set {*'linear', 'poly', 'rbf'*}, C to 10 from the set {0.1, 1, 10}. For XGBoost, we set the value of *max_depth* to 10 from the set of {3, 6, 10}, *gamma* to 1 from the set {1, 5, 9}, and *n_estimator* to 180.

We performed a manual search to tune the hyperparameters of the code2vec and ASTNN models. The dataset is split in a 3:1:1 ratio for training, validating, and testing in the experiments. The best hyperparameters are selected using the validating dataset, while the results are reported on the testing dataset. For the code2vec model, we set the embedding size of the model to 128 from a set of {64, 128, 256} and padded the context path sequence for each code submission to 100. For the ASTNN model, we set the embedding size to 128 from a set of {64, 128, 256}. The maximum epoch is set to 200 with a patience of 50 to prevent overfitting. To generate the ASTs from the programming codes, an open-source tool called javalang[7] is used. javalang provides a lexer and a parser for Java programming language.

5. Results

To investigate how well student-written and ChatGPT-generated codes can be distinguished in an introductory programming course, we perform a classification task using SVM, XGBoost, code2vec, and ASTNN model. In the experiments, we randomly selected 4000 programs in the training set and 1083 programs in the validation and test set. The testing results of the experiment are shown in Table 2.

From Table 2, one can see that AI models can distinguish between student-written codes and ChatGPT-generated codes. All identification accuracies and F1 scores are higher than 90%, which means that distinguishing is accurate with any of the models analyzed in the experiments. SVM has the lowest performance compared to other models, and XGBoost outperforms SVM. Table 2 also shows that code2vec outperforms ASTNN in terms of accuracy, recall, and f1-score with the value of 0.95, 0.95, and 0.95, respectively. However, the precision of ASTNN is the highest, with a value of 0.99. This implies that the correctly predicted positive instances by

[7]https://github.com/c2nes/javalang

Table 2
Performance comparison of different models

Model	Accuracy	Precision	Recall	F1-score
SVM	0.90	0.90	0.90	0.90
XGBoost	0.91	0.91	0.91	0.91
code2vec	**0.95**	0.95	**0.95**	**0.95**
ASTNN	0.92	**0.99**	0.87	0.92

ASTNN over all the predicted positive instances is 99%. However, from the recall value, we can see that ASTNN has a lower recall value of 0.87. Therefore, it predicts fewer positive instances over all the actual positive instances than code2vec (recall value: 0.95).

From these experimental results, we can conclude that code2vec performs better than ASTNN in classifying student-written codes and ChatGPT-generated codes. In general, the AST-based models have better performance than the traditional ML models in terms of accuracy, precision, recall, and F1-score. However, the performance of the traditional ML models is competitive compared to the AST-based models though traditional ML models deal with codes as textual data, whereas AST-based models try to encode the syntactic and semantic information of the codes. This indicates that there are enough textual differences between student-written codes and ChatGPT-generated codes. When looking into the differences between student-written code and ChatGPT code (see e.g. Figure 1), we found that ChatGPT uses more advanced operations like the ternary operator while students typically use nested if-else statements instead. This was supported by looking further into the attention weights of the code2vec model for the caughtSpeeding problem (Fig 1) and observing that the most important features to detect student versus ChatGPT code relied on paths related to using ternary versus nested if-else.

6. Discussion

The emergence of advanced AI tools (such as ChatGPT) could cause adverse effects in introductory programming such as students using AI tools to solve homework without understanding the generated code. However, in this paper, we found evidence to assure computing education practitioners that ChatGPT-generated code can be detected in introductory programming. We found that ChatGPT-generated code submissions could be detected automatically with a relatively high performance (95% accuracy). The ChatGPT code submissions also share patterns that human instructors could identify such as using more advanced programming constructs than the typical student would use. While it could still be worthwhile to investigate how students interact with such tools, they are effortless to detect by 1) an automated tool (as our experiment suggests) and 2) instructor inspections.

In addition, we also went a step further on preliminary explorations of the code generated by different prompts. Through simple attempts, we found that even if students attempt to make simple modifications to the prompts for ChatGPT to generate code that mimics novice programming code, the code generated is still distinguishable from students' own written code. For example, when we tried to add the sentence "Write the code as a novice programmer", or "Act as a novice programmer while programming", etc., the code generated is still structurally

```java
public int caughtspeeding(int speed, boolean isBirthday) {
    if(isBirthday) {
        if(speed <= 65) {
            return 0;
        }
        else if(66 <= speed \&\& speed <= 85) {
            return 1;
        }
        else {
            return 2;
        }
    }
    else {
        if(speed <= 60) {
            return 0;
        }
        else if(61 <= speed \&\& speed <= 80) {
            return 1;
        }
        else {
            return 2;
        }
    }
}
```

Student-written solution

```java
public int caughtSpeeding(int speed, boolean isBirthday) {
    int limit1 = isBirthday ? 65 : 60;
    int limit2 = isBirthday ? 85 : 80;
    if (speed <= limit1) {
        return 0;
    } else if (speed <= limit2) {
        return 1;
    } else {
        return 2;
    }
}
```

ChatGPT-generated solution

Figure 1: Student-written and ChatGPT-generated solution for the problem caughtSpeeding

different from actual students' programming code (e.g., the ChatGPT-generated code will still use ternary operators). While more solid experiments are still yet to be designed to validate this finding systematically, this preliminary result suggests that our findings on detecting the AI-generated code remain promising across different scenarios.

There are many potential next steps for instructors after detecting students' cheating. As a data-driven model detector, we acknowledge that there is no evidence to be presented to students showing that they committed cheating. However, many mechanisms could be designed to prevent cheating. For example, a cheating alarm system could be created to prevent students from submitting possibly AI-generated source code. When the submitted file is likely to have been generated by AI, instructors could request a re-submission automatically or ask the students for an additional check. Since possibly cheating students may face challenges in learning, the system could also serve as a detector of learning difficulty, and students who trigger the alarms frequently could be targeted by possible support interventions to learn related concepts.

Even though we consider using ChatGPT as potential cheating in this work, a suggestion we make is that schools and teachers should consider teaching students how to use AI ethically and efficiently. AI-based tools are double-edged swords, which can be over-relied on but might also help students learn concepts [40, 41]. It remains an effort and active area in the computing education domain to teach students when and how [42] to seek help from AI-driven tools.

6.1. Limitations

While this study provides an extensive exploration into detecting code generated by ChatGPT, some limitations should be considered. One of the main limitations of this study is that the dataset used for ChatGPT-generated code has a small number of variations on a small set of

original problems (10 problems). The prompts from the introductory programming course considered were straightforward and short, limiting the variety of code structures and syntax that ChatGPT would produce. It could cause the detecting model to be overfitted to specific kinds of variations, and future work should study other types of problems to verify whether our findings can be generalized, e.g., to more complicated problems. In addition, our current research makes an assumption that novice students know how to use ChatGPT to generate code. While there is little existing research (see e.g. [43, 44] for some preliminary results) systematically investigating how students interact with large language models and tools based on them such as ChatGPT, chances are that many students have not heard about it or do not have the ability to work with such tools.

7. Conclusion

We investigated the possibility of differentiating between student-written code and code generated by ChatGPT, a large language model developed by OpenAI, in the context of an introductory programming course. This study uses the CodeWorkout dataset with additional ChatGPT-generated data for the identification task. The study uses AST-based deep learning models, such as code2vec and ASTNN, and compares them with traditional machine learning, such as SVM and XGBoost, on their performance. The results show that all four models achieved high levels of accuracy of 90% and above, with code2vec achieving the best accuracy of 95%.

References

[1] I. Albluwi, Plagiarism in programming assessments: a systematic review, ACM Transactions on Computing Education (TOCE) 20 (2019) 1–28.

[2] A. Hellas, J. Leinonen, P. Ihantola, Plagiarism in take-home exams: help-seeking, collaboration, and systematic cheating, in: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, 2017, pp. 238–243.

[3] G. Cosma, M. Joy, Towards a definition of source-code plagiarism, IEEE Transactions on Education 51 (2008) 195–200.

[4] M. Joy, G. Cosma, J. Y.-K. Yau, J. Sinclair, Source code plagiarism—a student perspective, IEEE Transactions on Education 54 (2010) 125–132.

[5] M. Joy, J. Sinclair, R. Boyatt, J.-K. Yau, G. Cosma, Student perspectives on source-code plagiarism, International Journal for Educational Integrity 9 (2013).

[6] S. Sarsa, P. Denny, A. Hellas, J. Leinonen, Automatic generation of programming exercises and code explanations using large language models, in: Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1, 2022, pp. 27–43.

[7] P. Denny, S. Sarsa, A. Hellas, J. Leinonen, Robosourcing educational resources–leveraging large language models for learnersourcing, arXiv preprint arXiv:2211.04715 (2022).

[8] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, E. A. Santos, Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, 2023, pp. 500–506.

[9] P. Denny, V. Kumar, N. Giacaman, Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, 2023, pp. 1136–1142.
[10] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, J. Prather, The robots are coming: Exploring the implications of openai codex on introductory programming, in: Australasian Computing Education Conference, 2022, pp. 10–19.
[11] J. Finnie-Ansley, P. Denny, A. Luxton-Reilly, E. A. Santos, J. Prather, B. A. Becker, My ai wants to know if this will be on the exam: Testing openai's codex on cs2 programming exercises, in: Proc. of the 25th Australasian Computing Education Conf., 2023, pp. 97–104.
[12] L. Prechelt, G. Malpohl, M. Philippsen, et al., Finding plagiarisms among a set of programs with jplag., J. Univers. Comput. Sci. 8 (2002) 1016.
[13] G. Rosalsky, E. Peaslee, This 22-year-old is trying to save us from chatgpt before it changes writing forever, NPR. Archived from the original on January 18 (2023).
[14] J. Leinonen, K. Longi, A. Klami, A. Ahadi, A. Vihavainen, Typing patterns and authentication in practical programming exams, in: Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, 2016, pp. 160–165.
[15] K. Longi, J. Leinonen, H. Nygren, J. Salmi, A. Klami, A. Vihavainen, Identification of programmers from typing patterns, in: Proceedings of the 15th Koli Calling Conference on Computing Education Research, 2015, pp. 60–67.
[16] J. Sheard, S. Markham, M. Dick, Investigating differences in cheating behaviours of it undergraduate and graduate students: The maturity and motivation factors, Higher Education Research & Development 22 (2003) 91–108.
[17] J. Sheard, M. Dick, S. Markham, I. Macdonald, M. Walsh, Cheating and plagiarism: Perceptions and practices of first year it students, in: Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education, 2002, pp. 183–187.
[18] S. Dendir, R. S. Maxwell, Cheating in online courses: Evidence from online proctoring, Computers in Human Behavior Reports 2 (2020) 100033.
[19] M. Jha, S. J. Leemans, R. Berretta, A. A. Bilgin, L. Jayarathna, J. Sheard, Online assessment and covid: Opportunities and challenges, in: Australasian Computing Education Conference, 2022, pp. 27–35.
[20] A. Lavoie, M. Krishnamoorthy, Algorithmic detection of computer generated text, arXiv preprint arXiv:1008.0706 (2010).
[21] Q. Huang, G. Fang, K. Jiang, An Approach of Suspected Code Plagiarism Detection Based on XGBoost Incremental Learning, Proceedings of the 2019 International Conference on Computer, Network, Communication and Information Systems (CNCI 2019) (2019).
[22] W. Kechao, W. Tiantian, Z. Mingkui, W. Zhifei, R. Xiangmin, Detection of plagiarism in students' programs using a data mining algorithm, in: Proceedings of 2012 2nd International Conference on Computer Science and Network Technology, 2012, pp. 1318–1321.
[23] A. Eppa, A. Murali, Source code plagiarism detection: A machine intelligence approach, in: 2022 IEEE Fourth International Conference on Advances in Electronics, Computers and Communications (ICAECC), 2022, pp. 1–7. doi:10.1109/ICAECC54045.2022.9716671.
[24] M. Hoq, P. Brusilovsky, B. Akram, Analysis of an explainable student performance prediction model in an introductory programming course, in: Proceedings of the 16th International Conference on Educational Data Mining (EDM) 2023, 2023.

[25] M. Hoq, M. N. Uddin, S.-B. Park, Vocal feature extraction-based artificial intelligent model for parkinson's disease detection, Diagnostics 11 (2021) 1076.
[26] U. Alon, M. Zilberstein, O. Levy, E. Yahav, code2vec: Learning distributed representations of code, Proceedings of the ACM on Programming Languages 3 (2019) 1–29.
[27] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, X. Liu, A novel neural source code representation based on abstract syntax tree, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 783–794.
[28] Y. Shi, K. Shah, W. Wang, S. Marwan, P. Penmetsa, T. Price, Toward semi-automatic misconception discovery using code embeddings, in: LAK21: 11th International Learning Analytics and Knowledge Conference, 2021, pp. 606–612.
[29] Y. Shi, Y. Mao, T. Barnes, M. Chi, T. W. Price, More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code., in: EDM'21, 2021, pp. 446–453.
[30] Y. Shi, M. Chi, T. Barnes, T. Price, Code-dkt: A code-based knowledge tracing model for programming tasks, in: In Proc. of the 15th Int. Conf. on Educational Data Mining, 2022, pp. 50–61.
[31] M. Hoq, P. Brusilovsky, B. Akram, SANN: A subtree-based attention neural network model for student success prediction through source code analysis, in: 6th Educational Data Mining in Computer Science Education (CSEDM) Workshop, 2022.
[32] Y. Shi, Interpretable code-informed learning analytics for cs education, in: Companion Proc. 13th Int. Conf. on Learning Analytics & Knowledge, 2023, pp. 180–187.
[33] Y. Shi, R. Schmucker, M. Chi, T. Barnes, T. Price, Kc-finder: Automated knowledge component discovery for programming problems., in: Proceedings of the 16th International Conference on Educational Data Mining (EDM) 2023, 2023.
[34] W. Wang, G. Li, B. Ma, X. Xia, Z. Jin, Detecting code clones with graph neural network and flow-augmented abstract syntax tree, in: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2020, pp. 261–271.
[35] C. Fang, Z. Liu, Y. Shi, J. Huang, Q. Shi, Functional code clone detection with syntax and semantics fusion learning, in: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2020, pp. 516–527.
[36] Y. Mao, Y. Shi, S. Marwan, T. W. Price, T. Barnes, M. Chi, Knowing both when and where: Temporal-astnn for early prediction of student success in novice programming tasks, in: In Proceedings of the 14th International Conference on Educational Data Mining (EDM) 2021, 2021.
[37] M. Lei, H. Li, J. Li, N. Aundhkar, D.-K. Kim, Deep learning application on code clone detection: A review of current knowledge, Journal of Systems and Software 184 (2022) 111141.
[38] M. A. Fokam, R. Ajoodha, Influence of contrastive learning on source code plagiarism detection through recursive neural networks, in: 2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC), IEEE, 2021, pp. 1–6.
[39] S. Sarsa, J. Leinonen, A. Hellas, et al., Empirical evaluation of deep learning models for knowledge tracing: Of hyperparameters and metrics on performance and replicability, Journal of Educational Data Mining 14 (2022).
[40] K. Holstein, B. M. McLaren, V. Aleven, Student learning benefits of a mixed-reality teacher

awareness tool in ai-enhanced classrooms, in: Artificial Intelligence in Education: 19th International Conference, AIED 2018, London, UK, June 27–30, 2018, Proceedings, Part I 19, Springer, 2018, pp. 154–168.

[41] B. Akram, W. Min, E. Wiebe, B. Mott, K. E. Boyer, J. Lester, Improving stealth assessment in game-based learning with lstm-based analytics, in: International Conference on Educational Data Mining, 2018, pp. 208–218.

[42] S. Marwan, J. Jay Williams, T. Price, An evaluation of the impact of automated programming hints on performance and learning, in: Proceedings of the 2019 ACM Conference on International Computing Education Research, 2019, pp. 61–70.

[43] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, E. A. Santos, "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers, arXiv preprint arXiv:2304.02491 (2023).

[44] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, T. Grossman, Studying the effect of ai code generators on supporting novice learners in introductory programming, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1–23.