Engineering and Technology Journal e-ISSN: 2456-3358

Volume 10 Issue 11 November-2025, Page No.-7939-7946

DOI: 10.47191/etj/v10i11.25, I.F. – 8.482

When Students Asked ChatGPT Instead of Me: Investigating Generative AI in Programming Education Through NLP and Pedagogical Analytics

Milena Nikolić[1], Marina Marjanović[2]

[1,2]The Academy of Applied Technical and Preschool Studies, Singidunum University

ABSTRACT: This study investigates the growing dependence of students on popular generative artificial intelligence tools such as ChatGPT, GitHub Copilot, Jupyter AI, Google Bard, and more for coding assignments, project development, exam preparation, and conceptual learning in computer science higher education. Based on empirical data from various undergraduate and graduate-level programming courses at the Academy of Applied Technical and Preschool Studies in Serbia, the research applies natural language processing (NLP) and builds a machine learning model to examine student engagement and predict potential educational outcomes. A Python-based comprehensive framework integrates CodeBERT method for semantic similarity and plagiarism detection, TF-IDF with cosine similarity for benchmark comparisons, XGBoost for rubric-based classification, and DBSCAN clustering methods for code anomaly detection. Sentiment analysis further captures student attitudes toward frequent AI use. Rather than limiting the use of such approaches, this paper introduces a scalable solution for AI-aware assessment and curriculum design, encouraging responsible and ethical usage of modern generative technologies. The results support an innovative and future-ready model for education in the era of artificial intelligence.

KEYWORDS: Intelligent Systems, Programming Education, Pedagogical Analytics, NLP, Data Science, CodeBERT, XGBoost, Sentiment Analysis, DBSCAN Clustering

I. INTRODUCTION

Generative artificial intelligence (GAI) has quickly become a transformative force in programming education. Platforms including ChatGPT, GitHub Copilot, Jupyter AI, Google Bard, and OpenAI Codex are widely embedded in student practice, supporting various coursework, project development, exam preparation, and overall conceptual understanding [1]. Their availability has reshaped traditional help-seeking pathways, with students increasingly turning to AI systems rather than instructors or peers. Learners adopt distinct prompting and interaction strategies such as repeat-edit, scaffolding, copy-paste, and exploratory prompting, which illustrate the varied ways AI is used in programming contexts [2].

When carefully integrated into the coursework, GAI can improve assignment completion rates, enhance correctness, and strengthen general computational thinking, while also encouraging student motivation and confidence. However, unguided use risks fostering superficial learning, weakening debugging ability, and raising academic integrity concerns, while strongly influencing how learners approach reflection and cognitive skills [3]. This duality underscores the urgent need for pedagogical strategies that balance the efficiency of AI with the adoption of deeper problem-solving skills.

Alongside these considerations, research in computer and engineering education suggests that GAI can easily become a constructive learning companion when its use is toward objective reasoning. Embedding models like ChatGPT into programming assignments promotes higher-order thinking when students are guided to treat AI outputs as objects for analysis and evaluation instead of ready-made solutions [4]. This perspective encourages the idea that the incorporation of AI into programming education should be grounded on pedagogical practices that cover reasoning, design decisions, and ethical engagement, ensuring that efficiency is balanced with cognitive development.

Educational researchers increasingly argue for rethinking pedagogy and student assessment approaches. Rather than focusing only on the code output, assignments are expected to emphasize key reasoning processes, design choices, and reflective engagement with AI-assisted solutions. Analytical techniques that combine natural language processing with learning analytics are applied to cluster interaction patterns, analyze student prompts, and simulate behaviors to provide monitoring, personalization, and ethical oversight [5]. These insights point toward a broader reconfiguration of computer science education where AI is neither uncritically embraced nor prohibited but instead integrated in structured ways that maximize advantages while mitigating risks.

Building on the foundation of this work, the present study examines the integration of GAI in programming education through empirical data collected from the last two years of teaching courses at the Academy of Applied Technical and Preschool Studies in Serbia. A Python-based framework is proposed that leverages natural language processing and

machine learning techniques, specifically CodeBERT, TF-IDF with cosine similarity, followed by XGBoost and DBSCAN, to detect reliance patterns, predict student outcomes through the semester, and guide the implementation of an AI-aware curricula. By combining technical modeling with pedagogical reflection, this work contributes to the development of an ethical and scalable system for computer science education in the era of artificial intelligence.

II. LITERATURE REVIEW

Recent studies confirm that students all over the world have rapidly adopted GAI tools in programming courses, often preferring them over traditional sources of help like instructors or office hours [6]. Distinct prompt-use clusters have been identified, illustrating that learners employ varied interaction styles when engaging with systems like ChatGPT and GitHub Copilot [7]. For example, some students rely on rapid-fire prompting to obtain instant solutions, while others use iterative refinement to gradually improve code quality or depend heavily on AI for debugging assistance. These behaviors indicate that GAI is influencing not only the speed of task completion but also the depth of conceptual learning.

The research base between 2022 and 2025 documents opportunities and risks together. Positive outcomes contain measurable improvements in task completion, assignment correctness, and computational thinking when GAI is used under guided conditions [8]-[9]. Moreover, several studies report benefits in motivation, self-efficacy, and retention, with retention increases of up to 25 percent, then repeated mistakes decrease by 30 percent, and student satisfaction gains of 20 percent [10]-[11]. In benchmarking studies, GPT-4 achieved program repair rates of 88 percent and strong performance in generating relevant explanations, efficiently approaching the proficiency of a human tutor [12]. Similarly, CodeBERT and OpenAI Codex were employed to generate programming exercises and explanations that students rated highly for novelty, readiness, and usefulness [13].

Despite these gains, risks remain significant. Unguided or heavy dependence on AI has been primarily associated with shallow learning gains [14]. Similar concerns are raised in broader discussions of AI in education, which caution that students may rely on surface-level outputs at the expense of deeper engagement with problem-solving processes and disciplinary knowledge. Essential ethical challenges like plagiarism, fairness, and bias are widely documented as well, with at least five studies identifying academic integrity as a central issue [15]-[16]. In addition, privacy, transparency, and inclusivity remain unresolved, raising equity constraints around who secures advantages from these tools.

Pedagogical adaptations are also increasingly recognized as crucial. Research indicates that traditional summative assessment models are inadequate in contexts where AI solutions are readily available. Instead, scholars recommend designing assignments that emphasize reasoning, problem decomposition, and reflective evaluation of AI outputs [17]. Many classroom studies confirm that supervised integration produces more positive learning outcomes than unguided inclusion. Evidence further suggests that students benefit most when AI use is explicitly framed as a learning aid rather than a substitute for problem-solving. As a result, the role of instructors is shifting from being the main source of answers to becoming facilitators who teach prompt literacy, critical evaluation of model outputs, and higher-order design skills.

To encourage these pedagogical transformations, NLP and educational data analytics have been increasingly applied. Clustering analyses of student prompts reveal consistent interaction patterns that can be linked to learning behaviors. Simulation frameworks like CoderAgent demonstrate how synthetic learners can be utilized to explore personalization and adaptive scaffolding. Large-scale analytics pipelines at the institutional level have been deployed as well to detect bottlenecks in student progress and measure pre- and post-utilization effects of AI use [18]. A detailed overview of these studies, along with models used, tasks assigned, datasets, and performance highlights, is presented in Table I.

Overall, the literature showcases that GAI can act as a catalyst for improved programming education by enhancing feedback and supporting personalization. However, critical limitations remain. Most studies are strongly restricted to single programming courses, short time frames, or narrow institutional contexts. There is also not enough evidence of conceptual learning and skill transfers, and low consistency across evaluation metrics. Addressing described problems is necessary to ensure that AI integration improves short-term performance and sustains long-term learning outcomes. The present study responds to this gap by combining natural language processing, machine learning, and real classroom data to build a replicable and pedagogically informed model for AI-aware assessment and curriculum design.

Table I. A brief overview of previous findings related to GAI in programming education.

Study	Models/Tools Used	Tasks Addressed	Dataset / Context	Key Findings
Prather et al. (2023)	GPT-3.5, GPT-4, GitHub Copilot	Code generation, interpretation, teaching material creation	Undergraduate / Global	Highlighted opportunities and risks; GPT-4 achieved 51.5% avg. score; concerns about overreliance and misconduct
Xie (2024)	ChatGPT	Assignment completion, correctness, learning outcomes	Introductory Java / University	Guided use improved assignment completion rates and correctness
Cambaz & Zhang (2024)	Codex, GPT-3/3.5/4, Copilot	Code generation, tutoring, feedback	Introductory Python / Undergraduate	Identified performance variability; need for scaffolding and monitoring
Mboya et al. (2025)	GPT-3, GPT-4, CodeGeex	Personalized learning, feedback, tutoring	Universities / Kenya	Reported 25% retention gain, 30% fewer mistakes, 20% higher satisfaction
Boguslawski et al. (2024)	ChatGPT, LLMs	Motivation, debugging, complex projects	Undergraduate / Graduate / Germany	77% frequent use; improved autonomy and competence; risks of uncritical adoption
Phung et al. (2023)	ChatGPT 3.5/4	Program repair, hints, explanations	Introductory Python	GPT-4 achieved 88% program repair, 84% explanation, close to human tutor
Sarsa et al. (2022)	Codex, GPT-3, CodeBERT	Exercise generation, explanations	Introductory programming / University	Students rated exercises as 75% sensibleness, 81.8% novelty, 76.7% readiness

III. METHODOLOGY

A. Data Sources

As already mentioned, the empirical data for this study was collected from three undergraduate and graduate-level courses taught at the Academy of Applied Technical and Preschool Studies in the city of Niš, Serbia. The courses were Fundamentals of Programming, Software Engineering and Big Data Analytics. Each course spanned twelve weeks and required student submissions at regular intervals [19]-[21].

In Fundamentals of Programming course, approximately 120 assignments were submitted every two weeks. These tasks emphasized foundational C programming concepts and direct applications from class material and real-world exercises. Examples contained writing functions to simulate banking transactions, generating statistics from student records, and managing file operations.

In Software Engineering course, 60 Java assignments were submitted once per week. These projects introduced object-oriented design concepts, with exercises such as developing class hierarchies for e-commerce, implementing scheduling systems, or designing modules for simple management app.

In Big Data Analytics final-year students submitted 30 Python assignments every week. The assigned tasks focused on data preprocessing, analysis, and visualization methods. Representative examples covered parsing large CSV files for hotel and Twitter data, implementing sentiment analysis on text corpora, and generating dashboards for observations.

All assignments were designed to handle two categories of exercises. The first category consisted of direct coding tasks aligned with lecture topics. For example, students were asked to write recursive functions in C to compute factorials or Fibonacci numbers, to implement Python scripts for basic statistical calculations such as mean, median, and variance, and to design object-oriented class hierarchies in Java that applied design patterns to model entities such as students, courses, or bank accounts. The second category emphasized tasks that simulated common real-life scenarios, requiring students to transfer their knowledge and skills into practical, contextualized solutions. In this group, students developed C programs to manage a library system with borrowing and returning functionalities, built console Java applications for ride-sharing platforms that incorporated design patterns such as Singleton for global configuration, Factory Method for generating vehicle or driver objects, and Observer for updating ride status, and wrote Python programs to parse social media data and perform sentiment analysis. Other assignments involved designing systems that automatically generate timetables for exam registration and implementing data visualization dashboards in Python to display trends in booking specific hotels using public datasets.

In total, almost 4,500 student submissions were collected across courses, providing a rich and diverse collection that captures meaningful variations in coding styles, evolving problem-solving strategies, and dynamic interactions with generative AI tools observed over time.

B. Data Preprocessing

All code submissions were standardized in the beginning to ensure consistency across the dataset. This procedure involved unifying indentation styles, removing extraneous whitespace, and correcting encoding inconsistencies where applicable. Submissions that failed to compile or included incomplete fragments were flagged and retained separately to avoid skewing semantic or structural analysis. Likewise, broken, corrupted or otherwise invalid submissions were cleared at this stage, so they did not participate in clustering.

Following the described normalization, tokenization and parsing were applied using efficient language-specific tools. For example, Java code was processed with ANTLR while Python files were treated using the built-in tokenize module. This allowed main identifiers, keywords, and operators to be extracted into structured token sequences that could later be mapped to numerical feature spaces.

Natural language components, such as student comments embedded in the code and reflective notes submitted with assignments, were carefully preprocessed as well. Standard techniques were applied, such as lowercasing, punctuation removal, stopword filtering, and lemmatization, to create a clean textual representation [22]. This step guaranteed that natural language processing elements could be meaningfully aligned with programming constructs during analysis.

Feature extraction process combined two complementary strategies. TF-IDF vectorization was used to capture lexical distributions within comments, and CodeBERT embeddings provided representations of source code and accompanying text. This approach supported analysis of submissions at the syntactic and semantic levels, improving the model's ability to detect plagiarism, similarity, or conceptual overlap [23].

Finally, metadata related to submissions was encoded as numerical features, including variables such as submission frequency, time intervals between assignments, and code length. By incorporating temporal and behavioral features, the dataset was enhanced with information that revealed engagement patterns and possible overreliance on AI tools.

After preprocessing was performed, the raw submissions were transformed into vectors, embeddings, and metadata, ensuring that both code syntax and semantic meaning were preserved for upcoming machine learning and NLP tasks. Table II presents the final number of records retained for courses after all preprocessing techniques were applied.

Table II. The number of records before and after preprocessing.

Course	Initial Records	Final Records
Fundamentals of Programming (C)	2,323	2,103
Software Engineering (Java)	1,413	1,297
Big Data Analytics (Python)	697	652

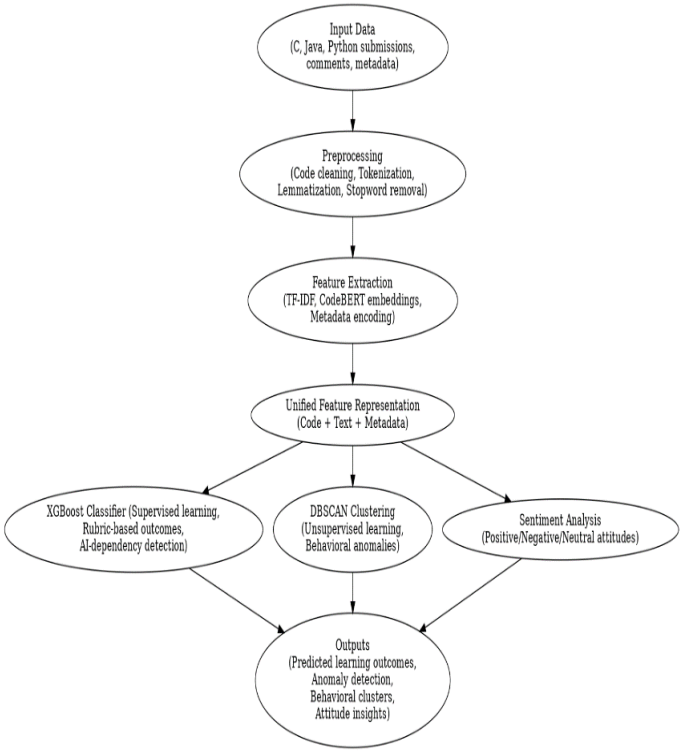

Figure I. Hybrid model architecture combining XGBoost classification, DBSCAN clustering, and sentiment analysis.

C. Model Selection

The proposed model architecture incorporates several complementary components that collectively address key structural, semantic, and behavioral aspects of programming activities among students. The feature space was built from three dimensions: CodeBERT embeddings provided deeper contextual representations of source code and comments, TF-IDF vectors captured lexical patterns in natural language segments, and metadata features like submission frequency, posting times, and code length added a behavioral layer. These diverse inputs were concatenated into a unified high-dimensional representation, shown in Figure I, enabling the system to capture lexical patterns, semantic context, and behavioral signals within a single analytical framework.

For supervised learning, XGBoost approach was selected as the primary classification model. Its gradient-boosted decision trees are highly effective for structured educational data, and the model is known for strong predictive accuracy, robustness against overfitting, and the ability to produce interpretable feature importance scores. Within this study, XGBoost was utilized to classify submissions against rubric-aligned criteria, identify potential overreliance on generative AI, and predict student outcomes across assignments [24].

For unsupervised learning, DBSCAN was adopted owing to its effectiveness to uncover irregular cluster structures and detect anomalies in noisy student submission datasets [25]. Unlike k-means or other centroid-based methods, DBSCAN does not require predefining the number of clusters and it is well suited for detecting behavioral outliers, such as sudden spikes in activity that may indicate excessive AI tool usage.

D. Model Training and Evaluation

Model training and evaluation was carried out in Python using a fusion of Scikit-learn (GridSearchCV, StratifiedKFold), HuggingFace Transformers (AutoModel, AutoTokenizer with pretrained CodeBERT embeddings) and the XGBoost library, enabling the powerful combination of traditional machine learning models and transformer-based embeddings. The processed student submissions were presented as matrices integrating lexical, semantic, and behavioral insights. This consolidated representation formed the input layer for both supervised and unsupervised learning modules.

To ensure reliable and generalizable model performance, a stratified cross-validation strategy was applied, balancing submissions across all assignment categories and difficulty levels. Data partitioning was designed to preserve temporal consistency, preventing information leakage between weeks and confirming that the evaluation mirrored real classroom dynamics. Hyperparameter tuning for XGBoost model was performed through a grid search, optimizing key parameters such as learning rate, maximum tree depth and the number of estimators. Early stopping mechanisms were integrated into the training cycle to mitigate overfitting and preserve model generalizability. For DBSCAN clustering, the epsilon radius and minimum sample threshold value were adjusted based on empirical testing, including silhouette scores with domain expertise on student coding behaviors to distinguish meaningful patterns from noise. DBSCAN was suitable in this context as it identifies clusters of arbitrary shape and labels low-density points as anomalies, making it effective for capturing irregular and atypical coding patterns.

The training phase thus merged the expressive strengths of transformer-derived embeddings with the interpretability offered by gradient boosting. This hybrid approach allowed the classifier to detect plagiarism-like similarities, identify evidence of AI-assisted code generation, and predict grading outcomes, while the clustering exposed latent behavioral structures and anomalies in submission patterns.

Model evaluation involved quantitative and qualitative analyses. For the supervised classification, performance was assessed using accuracy, precision, recall, and F1-scores to provide a balanced view of predictive capability. Clustering validity was examined through silhouette coefficients as well as manual inspection of cluster cohesion and separation.

IV. EXPERIMENTAL RESULTS

The experimental evaluation was designed to assess the effectiveness of the proposed architecture in capturing both semantic and behavioral properties of student submissions. Results are reported through classification, clustering, and sentiment analysis, providing a comprehensive overview of student engagement with different coding assignments.

The initial runs of the XGBoost classifier did not produce particularly strong results, with accuracy and recall values fluctuating below 80 percent. However, after fine-tuning of hyperparameters and optimization of feature integration, the model achieved stronger predictive performance across all courses. Using stratified cross-validation techniques, the average assignments accuracy reached 91.4 percent, with precision at 0.900, recall at 0.895, and F1-scores averaging 0.897 for more runs. Performance was slightly higher in the Fundamentals of Programming and Software Engineering courses, where assignments were notably more structured and guided by predefined grading criteria. For example, F1-scores reached 0.921 in Java projects that integrated design patterns like Singleton and Observer, reflecting the model's ability to capture learning outcomes. In contrast, Big Data Analytics showed a bit lower result, with accuracy averaging 88.3 percent and F1-scores around 0.865, due to the open-ended nature of Python assignments and diverse coding and analytical strategies among students. These findings suggest opportunities for continued refinement of the model and its adaptation to more instructional contexts. Table III presents a summary of evaluation metrics, showing that the classifier achieved high accuracy and balanced performance across courses while still noting challenges in less constrained tasks.

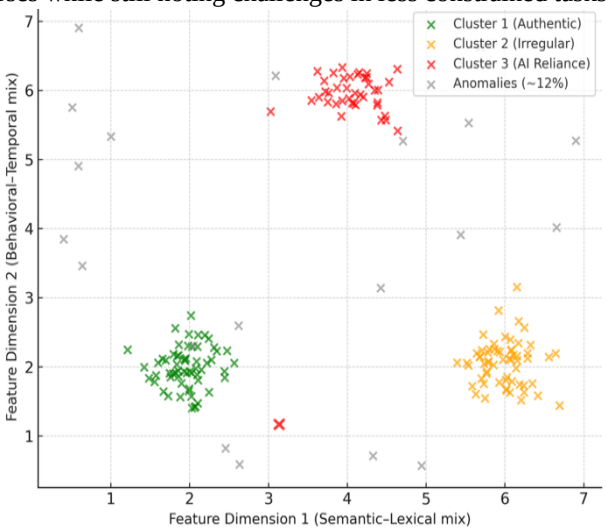

Figure II. DBSCAN clustering for student submissions.

Table III. Model performance metrics across courses.

Course	Accuracy	Precision	Recall	F1 Score
Fundamentals of Programming (C)	92.5%	0.910	0.900	0.906
Software Engineering (Java)	93.1%	0.920	0.921	0.921
Big Data Analytics (Python)	88.3%	0.870	0.861	0.865
Average (All courses)	91.4%	0.900	0.895	0.897

Feature importance analysis revealed that the CodeBERT embeddings were the strongest predictors, highlighting that semantic patterns in code and comments directly influenced classification outcomes. The way students wrote, structured, and explained submission codes carried significant weight in distinguishing authentic work from AI-assisted submissions. Behavioral metadata, including submission timestamps and assignment complexity, proved to be the next most relevant contributors. Unusually fast completions on complex tasks often signaled possible reliance on AI tools, while irregular submission intervals pointed to inconsistent engagement.

DBSCAN clustering revealed distinct behavioral groups among students, as illustrated in Figure II. Approximately 12 percent of submissions were flagged as anomalous (shown in gray), usually defined by high similarity to AI-generated templates, excessive resubmissions, or short completion times inconsistent with assignment length and complexity. The silhouette score reached 0.67, indicating meaningful cluster separation, as confirmed by manual inspection.

The clustering process produced three main groups based on semantic embeddings, lexical features, and behavioral metadata. The first cluster (green colored on graph) involved students with consistent and authentic implementation styles, where submission frequency and code length aligned with expected patterns. The second cluster (orange colored) contained students who showed irregular engagement, such as rapid completions of complex tasks or uneven submission intervals, suggesting intermittent adoption of AI utilities. The third cluster (red colored), which was proportionally small, consisted of students producing unusually long and well-structured code early in the semester, pointing to possible external help or heavy reliance on AI-generated solutions.

Course-specific differences were evident across clusters. In Fundamentals of Programming assignments, anomalies reflected copy-paste behaviors in repetitive C exercises. In Software Engineering course, flagged anomalies centered on Java design pattern implementation closely resembling AI-generated snippets. In Big Data Analytics projects, clusters exposed dependencies on external tutorials and pretrained libraries, particularly in climate and hotel data visualization, followed by sentiment analysis tasks for Twitter posts.

V. CONCLUSION

This study investigated the increasing reliance of students on generative artificial intelligence tools such as ChatGPT, GitHub Copilot, Jupyter AI, and Google Bard in programming education. Using data collected from three undergraduate and graduate-level programming courses at the Academy of Applied Technical and Preschool Studies in Serbia, we built an analytical framework that incorporated natural language processing, supervised and unsupervised machine learning, and pedagogical analytics. The system combined CodeBERT for semantic similarity and plagiarism detection, TF-IDF for deep lexical analysis, XGBoost for rubric-based classification, DBSCAN for anomaly detection, and sentiment analysis for interpreting reflective notes.

The results demonstrated that the model achieved strong predictive accuracy, distinguishing authentic work from AI-assisted submissions and uncovering irregular behaviors related to notable overreliance on generative tools. DBSCAN clustering revealed three distinct behavioral groups, where anomalies reflected copy-paste practices in C programming, AI-driven design pattern replication in Java, and reliance on external libraries in Python analytics. Moreover, sentiment analysis and student reflections further highlighted positive attitudes toward AI guidance and critical concerns related to fairness, critical thinking, and sustainable learning.

Overall, the hybrid approach effectively merged semantic embeddings, lexical features, and behavioral metadata to capture authentic engagement, anomalous patterns, and key risks of dependency. This combination of supervised and unsupervised techniques showcasing the benefits of aligning advanced analytics with pedagogical reflection, offering a more dependable model for evaluating student learning in AI-powered contexts.

While these results strongly indicate that generative AI is transforming programming education, further research is required to strengthen and generalize the framework. Multi-institutional studies might be helpful to validate applications beyond a single academic setting, and longitudinal research is needed to determine whether this type of learning fosters long-lasting knowledge acquisition. Developing a uniform evaluation criterion for AI-assisted learning could further enhance consistency and comparability across institutions. Furthermore, incorporating explainable AI methods such as SHAP (used for global feature importance) and LIME (used to provide local explanations of individual predictors) would enhance transparency, giving instructors better insights into model decisions and enabling more actionable feedback.

Taken together, this study demonstrates that prohibiting generative modern AI tools is neither feasible nor beneficial for education. Instead, a more balanced approach can be achieved through integration supported by guided feedback, clear ethical policies, and continuous monitoring. By uniting technical analysis with reflective pedagogy, the proposed solution highlights opportunities and risks of generative AI, pointing toward a future-ready model of computer science education that fosters meaningful learning while equipping students for an AI-driven professional environment.

REFERENCES

1. H. Güner and E. Er, "AI in the classroom: Exploring students' interaction with ChatGPT in programming learning," Educ. Inf. Tech., vol. 30, pp. 12681–12707, 2025, doi: 10.1007/s10639-025-13337-7.
2. K. Fuchs, "Exploring the opportunities and challenges of NLP models in higher education: Is ChatGPT a

blessing or a curse?," Front. Educ., vol. 8, p. 1166682, May 2023, Frontiers Media SA, doi: 10.3389/feduc.2023.1166682.

3. J. Beltrán and E. Veiga-Zarza, "Evaluating the use of large language models in programming courses: a comparative study," EDULEARN Proc., vol. 1, pp. 1761–1768, 2025, doi: 10.21125/edulearn.2025.0530.
4. A. Konak and C. J. S. F. Clarke, "Augmenting critical thinking skills in programming education through leveraging ChatGPT: Analysis of its opportunities and consequences," in Proc. 2023 Fall Mid Atlantic Conf.: Meeting Our Students Where They Are and Getting Them Where They Need to Be, Ewing, NJ, USA, Oct. 2023, doi: 10.18260/1-2--45117.
5. S. Folvarochna, Using learning analytics to identify student challenges in programming education. B.Sc. thesis, Dept. Comput. Sci. Inf. Technol., Fac. Appl. Sci., Ukrainian Catholic Univ., Lviv, Ukraine, 2025.
6. G. Fenu, R. Galici, M. Marras, and D. Reforgiato, "Exploring student interactions with AI in programming training," in Adjunct Proc. 32nd ACM Conf. User Modeling, Adaptation and Personalization, New York, NY, USA: Assoc. Comput. Mach., 2024, pp. 555–560, doi: 10.1145/3631700.3665227.
7. B. Ma, L. Chen, and S. Konomi, "Exploring student perception and interaction using ChatGPT in programming education," in Proc. 21st Int. Conf. Cogn. Explor. Learn. Digital Age (CELDA), 2024, doi: 10.33965/celda2024_202408l005.
8. J. Xie, "Improving introductory Java programming education through ChatGPT," J. Comput. Sci. Coll., vol. 40, no. 3, pp. 140–150, Oct. 2024.
9. R. Yilmaz and F. G. K. Yilmaz, "The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation," Comput. Educ.: Artif. Intell., vol. 4, p. 100147, 2023, doi: 10.1016/j.caeai.2023.100147.
10. F. M. Mboya, G. M. Wambugu, A. M. Oirere, E. O. Omuya, F. M. Musyoka, and J. W. Gikandi, "Enhancing personalized learning in programming education through generative artificial intelligence frameworks: A systematic literature review," Int. J. Adv. Trends Comput. Sci. Eng., vol. 14, no. 2, pp. 514–522, 2025, doi: 10.30534/ijatcse/2025/051422025.
11. S. Boguslawski, R. Deer, and M. G. Dawson, "Programming education and learner motivation in the age of generative AI: Student and educator perspectives," Inf. Learn. Sci., vol. 126, no. 1/2, pp. 91–109, 2025, doi: 10.1108/ILS-10-2023-0163.
12. T. Phung, V. A. Pădurean, J. Cambronero, S. Gulwani, T. Kohn, R. Majumdar, and G. Soares, "Generative AI for programming education: Benchmarking ChatGPT, GPT-4, and human tutors," in Proc. 2023 ACM Conf. Int. Comput. Educ. Res.–Vol. 2, Aug. 2023, pp. 41–42, doi: 10.1145/3568812.3603476
13. S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in Proc. 2022 ACM Conf. Int. Comput. Educ. Res. – Vol. 1, Aug. 2022, pp. 27–43, doi: 10.1145/3501385.3543957.
14. S. Yazdani, M. Najimi, and M. Ahmadzadeh, "The paradox of generative AI in programming education," in EDULEARN Proc., 2025, pp. 7775–7784, doi: 10.21125/edulearn.2025.1927.
15. D. Franklin, P. Denny, D. A. Gonzalez-Maldonado, and M. Tran, Generative AI in computer science education: Challenges and opportunities. Cambridge, U.K.: Cambridge Univ. Press, 2025.
16. J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Albluwi, M. Craig, and J. Savelka, "The robots are here: Navigating the generative AI revolution in computing education," in Proc. 2023 Working Group Reports Innov. Technol. Comput. Sci. Educ., 2023, pp. 108–159, doi: 10.1145/3623762.3633499.
17. D. Cambaz and X. Zhang, "Use of AI-driven code generation models in teaching and learning programming: A systematic literature review," in Proceed. 55th ACM Techn. Sympos. Comput. Sci. Educ. vol. 1, Mar. 2024, pp. 172–178, doi: 10.1145/3626252.3630958.
18. Y. Zhan, Q. Liu, W. Gao, Z. Zhang, T. Wang, S. Shen, et al., "CoderAgent: Simulating student behavior for personalized programming learning with large language models," arXiv preprint, 2025. doi: 10.48550/arXiv.2505.20642.
19. The Academy of Applied Technical and Preschool Studies, Lecture notes on Fund. of Programming, Serbia, 2023-2024.
20. The Academy of Applied Technical and Preschool Studies, Lecture notes on Software Engineering, Serbia, 2023-2024.
21. The Academy of Applied Technical and Preschool Studies, Lecture notes on Big Data Analytics, Niš, Serbia, 2023-2024.
22. K. M. G. S. Karunarathna and R. A. H. M. Rupasingha, "Learning to use normalization techniques for preprocessing and classification of text documents," Int. J. Multidiscip. Stud., vol. 9, no. 2, pp. 69–81, 2022.
23. P. T. Nguyen, J. Di Rocco, C. Di Sipio, R. Rubei, D. Di Ruscio, and M. Di Penta, "Is this snippet written by ChatGPT? An empirical study with a CodeBERT-based classifier," arXiv preprint, 2023, doi: 10.48550/arXiv.2307.09381.
24. A. Asselman, M. Khaldi, and S. Aammou, "Enhancing the prediction of student performance based on the

machine learning XGBoost algorithm," Interact. Learn. Environ., vol. 29, no. 3, pp. 3360–3379, 2021, doi: 10.1080/10494820.2021. 1928235.

25. H. Du, S. Chen, H. Niu and Y. Li, "Application of DBSCAN clustering algorithm in evaluating students' learning status," in Proc. 17th Int. Conf. Comput. Intell. Security, Chengdu, China, 2021, pp. 372–376, doi: 10.1109/CIS54983.2021.00084.