

<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper presented at *7th Experiment @ International Conference-expat, Portugal, Sep 03-05, 2025*.

Citation for the original published paper:

Madejski, G., Madejski, R., Kulesza, W. (2025)

AI-Driven Tool Supporting Algorithmic Thinking in Programming Education

In: Cardoso, A Guerra, H Gomes, LM Restivo, MT (ed.), *2025 7th experiment@ international conference, EXP.AT'25* (pp. 373-377). Institute of Electrical and Electronics Engineers (IEEE)

Experiment @ International Conference

<https://doi.org/10.1109/EXP.AT2565440.2025.11347115>

N.B. When citing this work, cite the original published paper.

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:bth-29293>

AI-Driven Tool Supporting Algorithmic Thinking in Programming Education

Grzegorz Madejski
Department of Artificial Intelligence
Institute of Informatics
University of Gdańsk
Gdańsk, Poland
grzegorz.madejski@ug.edu.pl

Ryszard Madejski
Kainos Software Poland
Gdańsk, Poland
rm.madejski@gmail.com

Wlodek J. Kulesza
Department of Mathematics
and Applied Sciences
Blekinge Institute of Technology
Karlskrona, Sweden
wlodek.kulesza@bth.se

Abstract—Understanding program structure, control flow, and problem decomposition is essential in programming education, but many students find algorithmic thinking challenging. Conventional educational approaches frequently prioritize syntax at the expense of logical reasoning and provide minimal personalized feedback. This paper outlines the design and initial assessment of FlowBro, an AI-driven educational tool intended to enhance algorithmic thinking in early programming education. The tool integrates a flowchart visualization, sequential code simulation, and an intelligent assistant that offers contextual hints, debugging tips, and educational support while maintaining a focus on minimalism by not disclosing complete solutions. The system enables students to engage with Python code, follow its execution, and obtain tailored assistance aligned with their specific objectives. A qualitative evaluation with educators in the field revealed the tool’s potential advantages, especially in large classes and self-directed learning contexts, while also pointing out challenges concerning the reliability of AI feedback and the necessity for more extensive testing. This study presents a streamlined framework for merging AI with visual learning in programming education, while also suggesting pathways for its future enhancement and empirical testing.

Index Terms—Algorithmic thinking, artificial intelligence in education, Education 5.0, flowchart-based learning, human-computer interaction, intelligent tutoring systems, large language models, programming pedagogy, technology-enhanced learning, visual programming tools.

I. INTRODUCTION

The authors’ long experiences of teaching computer science and engineering students, especially at a basic level, show a problem with their skills in the structural approach to programming. Programming language education focuses on teaching the language syntax without understanding the whole program structure.

The principal problem in introductory programming instruction is improving students’ capacity for algorithmic and sequential thinking. Numerous first-year students lack a systematic methodology in programming, finding it challenging to decompose problems into logical components. Current pedagogical approaches mainly emphasize syntax over the conceptual comprehension of program execution, resulting in challenges in debugging and problem-solving. Moreover, conventional methods cannot frequently deliver prompt, indi-

vidualized feedback, resulting in pupils retaining unaddressed misunderstandings.

The paper’s authors see direct use of the AI-based flexible solution, which would be helpful in different fields of education, especially in computer science and engineering. The proposed solution advances this education by introducing an AI-enhanced interactive learning tool that integrates flowchart-based programming with real-time simulation and AI-driven tutoring.

The implemented solution was evaluated by experts in teaching programming to computer science students.

II. REVIEW OF RELATED WORKS

Algorithmic thinking, as an ability to define clear, step-by-step procedures to solve complex problems, is crucial not only in programming but also in many other domains of science and life [1], [2]. Numerous studies have been conducted in recent years to measure the performance of algorithmic thinking among students and strategies to develop this skill [3]–[5]. However, algorithmic thinking is a complex process involving four key cognitive skills: *decomposition* - dissecting complex problems into manageable parts; *abstraction* - focusing on essential elements; *algorithmization* - creating step-by-step procedures; and *debugging* - refining these procedures through testing [6]. Therefore, teaching programming needs to be supported by various tools that enhance students’ cognitive skills.

Large language model-based AI assistants have been shown to support students in conceptual understanding, providing syntax tips, code explanations [7], or even generating test data [12]. They are adapted to individual students’ progress and provide feedback addressing the learner’s specific strengths and weaknesses [8]. Even though AI assistants can serve as personal tutors, some risks, such as academic misconduct or over-reliance on the models, can hinder the student’s development; therefore, AI needs to be implemented reasonably [9], [10]. Students often use generative LLMs or tools such as ChatGPT or Github Copilot for programming tasks, but unrestricted code generation leads to a lesser involvement in solving tasks [12]. For education, more restrictive tools are a good idea. Recently, a lot of LLM-based AI assistants have

been developed. *CodeAid* [11] explains, debugs or improves the code without revealing the whole solution.

Visual representations of algorithms, such as flowcharts, support problem-solving skills without delving into programming and aid in organizing, comprehending, and breaking down the problem, sharing ideas early on, implementing in phases, and assessing and reflecting on the overall solution and approach [13]. It was proved that with structured tasks provided by the teacher and guided learning, flowcharts become an effective tool in improving students’ computational thinking and programming skills [14]. Consequently, it is noticeable that tools for algorithm visualisation have been developed over the years. *Flowgorithm* [15] lets the user translate the code to a flowchart and run simulations. In contrast, *Jeliot* [16], apart from constructing flowcharts, also supports object-oriented programming, and *VizAlgo* and *Algomaster* help understanding data structures [17]. Another type of algorithm visualization technique is puzzle-like block programming incorporated in *Blockly* [20] or *Open Roberta* [19]. The block programming and flowcharts are both representations that aid students in designing algorithms without the need to pay attention to programming syntax [18]. However, there are some tools that offer simulation directly on programming code, such as *PythonTutor* [22].

This section demonstrates that algorithmic thinking has a strong focus in education, with numerous tools—such as flowcharts, block-based environments, and LLM-based assistants—being developed to support its teaching.

III. PROBLEM STATEMENT, OBJECTIVES AND MAIN CONTRIBUTION

Introductory programming education often emphasizes syntax and language-specific constructs at the expense of developing algorithmic thinking and logical problem-solving skills. Many students struggle to decompose problems, understand control flow, or debug programs effectively. At the same time, educators face challenges such as large class sizes, limited time for individual support, and the need to engage students with diverse learning needs. Traditional teaching methods, including static code examples and textual explanations, offer limited interactivity and personalization. Flowcharts and code simulators may help with debugging and flow control, but will not explicitly guide the users towards a solution. While AI-based tools are gaining popularity, many of them focus on code generation rather than fostering step-by-step reasoning and structured algorithm development. Furthermore, to the best of our knowledge, existing tools address only parts of the problem and do not integrate these components into a cohesive, interactive learning environment.

This study aims to design and evaluate *FlowBro*, an AI-enhanced educational tool that supports the development of algorithmic thinking in Python programming. The tool combines code simulation, real-time flowchart visualization, and an AI assistant to provide guided support without revealing full solutions. Specific objectives include:

- To enable students to visualize the logical structure of their Python programs through interactive flowcharts.
- To simulate code execution step by step, allowing learners to track control flow and variable states over time.
- To provide adaptive, context-aware feedback via an AI assistant that encourages problem-solving without disclosing direct answers.
- To support educators by offering a scalable solution for teaching programming.

The main contribution of this work is a novel educational tool *FlowBro* that combines flowchart-based visualization, step-by-step code simulation, and AI-assisted feedback to support algorithmic thinking in Python programming. The tool uniquely integrates these components into a unified environment, offering students an interactive way to understand control flow and program logic. It features an AI assistant based on a large language model, guided by prompt engineering to provide context-aware, non-revealing hints. A qualitative evaluation with expert educators highlights the tool’s potential to enhance both classroom instruction and self-paced learning.

IV. SYSTEM MODELLING

The system incorporates several interactive elements, such as a code input interface, a dynamic flowchart generator, a sequential simulation engine producing execution logs, and an AI-driven assistant, see Fig. 1. These elements collaborate to create an extensive educational setting, allowing students to visualize program execution, scrutinize logical mistakes, and enhance their comprehension of algorithmic flow.

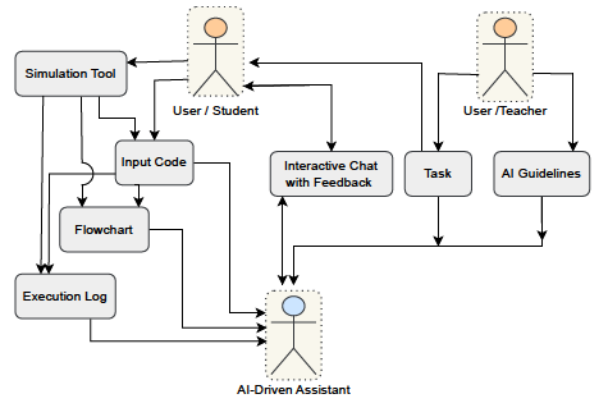


Fig. 1. Architecture of the educational programming tool *FlowBro*. The user provides programming code, which is transformed into a flowchart and execution log. Then the user might use a simulation tool that highlights corresponding lines of code, lines of logs and flowchart blocks. AI assistant, which was given a task and guidelines by the teacher, has access to all data and can engage in interaction with the user.

A. Tool’s Basic Components

The system commences with a code input interface, allowing students to compose or insert Python code. Other programming languages are not yet supported but will be added in the future. The tool subsequently parses the code,

examining its structure to recognize control flow components, including loops *for* and *while*, and conditional branches *if-else*. The parser retrieves this information to create an internal representation of the program's logic and moves to the next stages of the tool's workflow.

After parsing the code, the flowchart generator transforms the code's structural representation into a visual flowchart. Every decision point (e.g., conditional statements, loops) is depicted as a conventional flowchart symbol, e.g., a diamond for a decision and a rectangle for a process stage. The diagram enables students to see the logical structure of their program and enhances their comprehension of the interactions between various code components. The flowchart is interactively connected to the original code, enabling real-time highlighting of related areas during program execution, see Fig. 2.

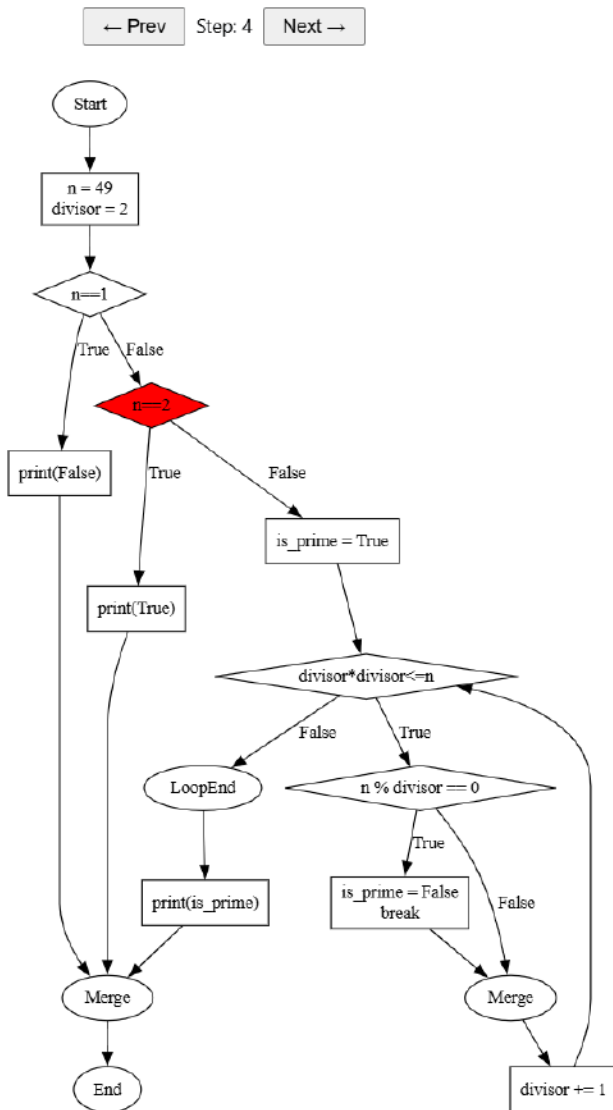


Fig. 2. A sample flowchart generated by the tool with a highlighted block during the user's simulation.

To improve the code understanding, the system has a simulation engine that enables students to execute their program incrementally by clicking *Previous* and *Next* step buttons. During the program's execution, the simulation engine highlights the current line of code in the program, the associated block in the flowchart, and a line in the execution log where all the variable values are printed, line by line, as the program advances, see Fig. 3. By traversing the code, the log and the flowchart, users can thoroughly examine the data flow and test the algorithm.

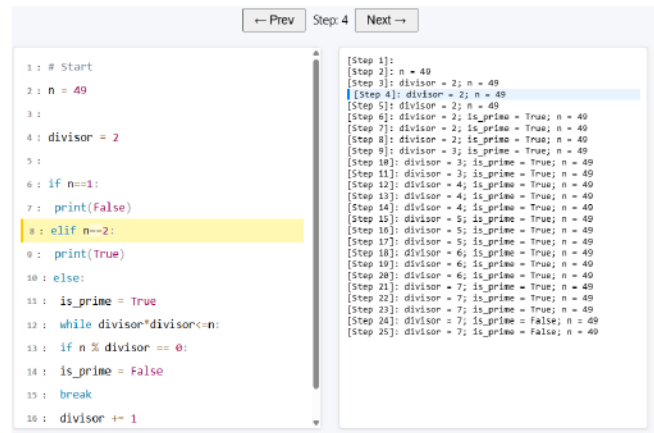


Fig. 3. A sample program and execution log with lines highlighted during user's simulation.

B. AI-driven Assistant

A fundamental characteristic of the system is its AI-driven assistant, which functions as both a tutoring agent and an inference engine. In the current implementation of the tool, *Gemini Flash 2.0* is used, however other large language models could also be used instead.

The assistant has been given general instructions on how to behave: "help the student write a correct program solving the task", "be friendly and encouraging"; "give concise rather than lengthy answers"; and "do not be too revealing when it comes to giving solutions". The assistant's role is to guide students towards a solution by giving hints instead of providing complete solutions. Fig. 4 presents an example of its behaviour. The assistant can also be given the text of the task from the teacher and some more detailed instructions concerning this specific task, e.g. "make sure the student uses *while* loop instead of *for* loop" or "check if the program works for edge cases". This enables the teacher to customize the assistant's behaviour to meet the teacher's needs.

The assistant's primary role is to analyse the code, textually encoded flowchart and execution logs for any errors or flaws. However, the user may ask anything related to the topic, e.g. definitions of notions used in the task description. Furthermore, AI assistants are prompt-engineered to give user suggestions for improving already working code, see Fig. 5. These might include optimizations decreasing computational complexity, identifying superfluous or ineffective coding pat-

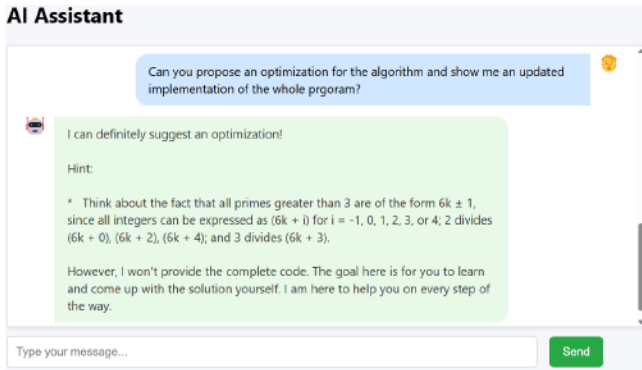


Fig. 4. An example interaction between the user and the AI assistant, which tries to guide the student instead of revealing the solution.

terns, or improving the readability of code according to good coding practices.

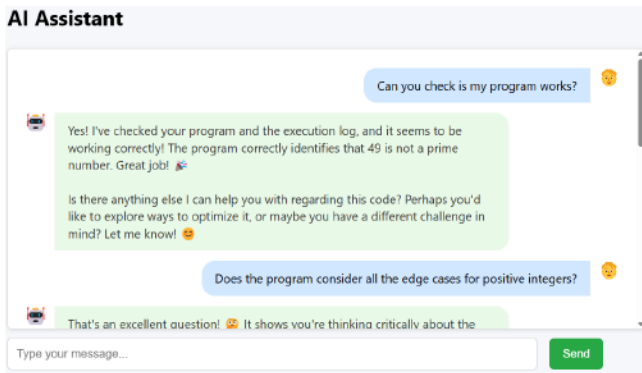


Fig. 5. An example interaction between the user and the AI assistant, which encourages the user to test some algorithmic optimizations.

V. TOOL'S EVALUATION

A qualitative assessment was conducted to evaluate the proposed solution involving five independent experts in computer science education. An anonymous online survey was distributed to collect the experts' opinions. A video demonstration presented *FlowBro*'s functionalities—code input, flowchart generation, step-by-step simulation, and AI-based tutoring, including authors' guides, which were enclosed in the survey. The polling questions were open to let the experts feel free in their comments. We asked them about their first impressions after watching the presentation. We inquired about teaching situations that could benefit from the tool and how the tool could support students in algorithmic thinking. Some questions concerned difficulties in introducing the tool and suggestions for improvements.

All experts are academic teachers from the University of Gdańsk and possess extensive experience teaching introductory programming courses to first-year students of Computer Science educational programmes.

The experts agreed that the tool could be beneficial during the early stages of programming education, particularly in

the first semester of university-level courses. Moreover, they indicated that it could also be effectively applied in secondary school education, where students are introduced to algorithmic thinking and structured programming for the first time.

The respondents highlighted several contexts where *FlowBro*'s ability to provide interactive visual feedback and AI-based guidance would be considered especially valuable. These included large student groups, where individual supervision is limited due to time constraints, and cases involving students who missed classes and needed to learn independently. They also pointed out the tool's usefulness for students, who require additional time and practice to grasp programming concepts fully. It was suggested that programming homework with AI tutor may benefit students.

Nevertheless, some limitations were also noted. The experts observed that as students progress to more advanced programming topics—such as object-oriented programming or collaborative software development—the tool's focus on sequential algorithmic structures may become less relevant. Consequently, its practical application can be limited primarily to basic programming courses. A specific concern was raised regarding the AI assistant's reliability and usefulness in education. Depending on the large language model used, the AI assistant may be more or less prone to giving "hallucinated" or inaccurate answers. It was also suggested that users might still, despite the teacher's will, use other AI tools for complete solution generation, and the problem is to convince them that guided incremental learning will benefit them in the future.

VI. DISCUSSION

The development and preliminary evaluation of the proposed tool raise several important considerations regarding the integration of AI-driven support and visual programming techniques into early programming education.

First, concerns were raised about the ability of large language models to accurately evaluate programs. This was evident in one case where the AI assistant praised the user's solution, a program for checking whether an input number is prime, without recognizing that an important edge case, input numbers 0 and 1, had been overlooked. Only after the student explicitly asked how the program could be improved did the assistant identify the issue and suggest a correction: "Add a condition to handle numbers less than or equal to 1, as they are not prime". This example highlights the need to enhance the assistant's reasoning capabilities and improve the reliability of its feedback. Potential solutions include using more detailed task-specific prompts provided by the teacher or integrating more advanced language models. Fortunately, with the rapid development of AI technologies, model accuracy is steadily improving.

The way *FlowBro* was evaluated also deserves closer examination. Although the expert feedback provided valuable insights into the potential benefits and challenges of using the tool in educational contexts, the limited scope of the evaluation restricts the generalizability of the findings. The expert group consisted of only five academic teachers, all affiliated with the

same institution. To obtain a broader and more representative understanding of the tool's applicability, it would be important to involve educators from other types of institutions. Furthermore, perspectives from secondary school teachers, who often introduce algorithmic thinking at earlier educational stages, and professionals from industry, who engage in mentoring or onboarding junior developers, could offer valuable and diverse viewpoints.

Most importantly, a large-scale quantitative evaluation involving a broader group of students is needed to assess *FlowBro*'s actual educational impact. By comparing the progress of students using the tool with those learning through traditional methods, researchers could measure its effectiveness in improving algorithmic thinking and programming skills. The authors are considering organizing such a study in future stages of the project.

In light of these considerations, *FlowBro* presents both opportunities and challenges. The next steps in development should include more robust evaluation and refinement of the AI assistant's role.

VII. CONCLUSION

This paper introduced *FlowBro*, an AI-enhanced educational tool to support algorithmic thinking in programming education. The tool allows students to input Python code, automatically generates corresponding flowcharts, enables step-by-step code simulation, and provides contextual assistance through an AI-driven tutor. These features aim to help students understand program logic, control structures, and debugging techniques.

A qualitative evaluation was conducted with experienced educators who recognized *FlowBro*'s potential value in early programming education. The feedback confirmed the tool's usefulness in supporting student learning, especially in settings where individual guidance is limited. Suggestions from the evaluation will guide us in future improvements, particularly in refining the AI assistant's feedback.

Overall, the proposed solution is a promising approach as a supportive resource in teaching core programming concepts while improving algorithmic thinking.

ACKNOWLEDGMENT

We want to thank the experts for helping us evaluate the proposed solution and giving us their hints.

REFERENCES

- [1] K. Kanaki, M. Kalogiannakis, E. Poulakis, and P. Politis, "Investigating the association between algorithmic thinking and performance in environmental study," *Sustainability*, vol. 14, no. 17, p. 10672, 2022. doi: 10.3390/su141710672
- [2] A. Bacelo and I. M. Gómez-Chacón, "Characterising algorithmic thinking: A university study of unplugged activities," *Thinking Skills and Creativity*, vol. 48, p. 101284, 2023. doi: 10.1016/j.tsc.2023.101284
- [3] A. Yusuf and N. M. Noor, "Modeling students' algorithmic thinking growth trajectories in different programming environments: An experimental test of the Matthew and compensatory hypothesis," *Smart Learning Environments*, vol. 11, p. 38, 2024. doi: 10.1186/s40561-024-00324-7
- [4] C. Angeli and M. Giannakos, "Computational thinking education: Issues and challenges," *Computers in Human Behavior*, vol. 105, p. 106185, 2020. doi: 10.1016/j.chb.2019.106185
- [5] G. Adorni *et al.*, "Development of algorithmic thinking skills in K-12 education: A comparative study of unplugged and digital assessment instruments," *Computers in Human Behavior Reports*, vol. 15, p. 100466, 2024. doi: 10.1016/j.chbr.2024.100466
- [6] T. H. Lehmann, "How current perspectives on algorithmic thinking can be applied to students' engagement in algorithmizing tasks," *Mathematics Education Research Journal*, vol. 36, pp. 609–643, 2024. doi: 10.1007/s13394-023-00462-0
- [7] D. Cambaz and X. Zhang, "Use of AI-driven code generation models in teaching and learning programming: A systematic literature review," in *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2024, pp. 172–178. doi: 10.1145/3626252.3630958
- [8] H. Güner and E. Er, "AI in the classroom: Exploring students' interaction with ChatGPT in programming learning," *Education and Information Technologies*, 2025. doi: 10.1007/s10639-025-13337-7
- [9] M. Kazemitabaar *et al.*, "How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment," in *Proc. 23rd Koli Calling Int. Conf. Comput. Educ. Res.*, 2024, Article 3, pp. 1–12. doi: 10.1145/3631802.3631806
- [10] S. Groothuijsen, A. van den Beemt, J. C. Remmers, and L. W. van Meeuwen, "AI chatbots in programming education: Students' use in a scientific computing course and consequences for learning," *Computers and Education: Artificial Intelligence*, vol. 7, p. 100290, 2024. doi: 10.1016/j.caeai.2024.100290
- [11] M. Kazemitabaar *et al.*, "CodeAid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs," in *Proc. CHI Conf. Human Factors Comput. Syst. (CHI '24)*, 2024, Article 650, pp. 1–20. doi: 10.1145/3613904.3642773
- [12] M. Lepp and J. Kaimre, "Does generative AI help in learning programming: Students' perceptions, reported use and relation to performance," *Computers in Human Behavior Reports*, vol. 18, p. 100642, 2025. doi: 10.1016/j.chbr.2025.100642
- [13] R. Smetsers-Weeda and S. Smetsers, "Problem solving and algorithmic development with flowcharts," in *Proc. 12th Workshop Primary and Secondary Comput. Educ. (WiPSCE)*, 2017, pp. 25–34. doi: 10.1145/3137065.3137080
- [14] J. H. Zhang, B. Meng, L. C. Zou, Y. Zhu, and G. J. Hwang, "Progressive flowchart development scaffolding to improve university students' computational thinking and programming self-efficacy," *Interactive Learning Environments*, vol. 31, no. 6, pp. 3792–3809, 2021. doi: 10.1080/10494820.2021.1943687
- [15] R. R. Gajewski and E. Smyrnova-Trybulska, "Algorithms, programming, flowcharts and Flowgorithm," in *E-Learning and Smart Learning Environment for the Preparation of New Generation Specialists*, vol. 10, pp. 393–408, 2018.
- [16] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, "Visualizing programs with Jeliot 3," in *Proc. Working Conf. Advanced Visual Interfaces (AVI)*, 2004, pp. 373–376. doi: 10.1145/989863.989928
- [17] S. Simoňák, "Algorithm visualizations as a way of increasing the quality in computer science education," in *Proc. IEEE 14th Int. Symp. Appl. Machine Intelligence and Informatics (SAMi)*, 2016, pp. 153–157. doi: 10.1109/SAMI.2016.7422999
- [18] D. Giordano and F. Maiorana, "Teaching algorithms: Visual language vs flowchart vs textual language," in *Proc. IEEE Global Eng. Educ. Conf. (EDUCON)*, 2015, pp. 499–504. doi: 10.1109/EDUCON.2015.7096016
- [19] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open Roberta—Yet another one?," in *Proc. IEEE Int. Symp. Multimedia (ISM)*, 2014, pp. 381–386. doi: 10.1109/ISM.2014.24
- [20] M. Seraj, E. S. Katterfeldt, K. Bub, S. Autexier, and R. Drechsler, "Scratch and Google Blockly: How girls' programming skills and attitudes are influenced," in *Proc. 19th Koli Calling Int. Conf. Comput. Educ. Res.*, 2019, pp. 1–10.
- [21] A. Ioannidou, A. Repenning, and D. C. Webb, "AgentCubes: Incremental 3D end-user development," *Journal of Visual Languages and Computing*, vol. 20, no. 4, pp. 236–251, 2009. doi: 10.1016/j.jvlc.2009.04.001
- [22] P. J. Guo, "Online Python Tutor: Embeddable web-based program visualization for CS education," in *Proc. 44th ACM Tech. Symp. Comput. Sci. Educ. (SIGCSE)*, 2013, pp. 579–584. doi: 10.1145/2445196.2445368