Investigating the influence of code generating technologies on learning process of novice programmers in higher education computer science course

Albina Shynkar
a.shynkar@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

The advent of code generation technologies has been widely influencing the education in general and computer science education in particular. In recent years many studies have investigated how code generation technologies fit in educational progress, discussed potential applications and implications of its use, proposed new platforms as the solution to common problems and argued about their efficacy. However, recent studies are lacking the knowledge about effects of code generating tools usage on novice programmer's motivation and engagement in the learning process as well as on further development of programming proficiency of individuals. To address this gap, this work investigates the related literature to examine the impact of code generation technologies on novice programmer's learning experience as well as reinforces the findings by exploring the opinions, experiences of novice students. The results of the research shed the light on the acceptance of such technologies within educational realm, effects of code generating tools on novice programming student's learning experience, namely increased engagement towards learning process, and provide the area for the discussions about further application of code generation tools in computer science education.

KEYWORDS

AI, Code Generation Tools, ChatGPT, OpenAI Codex, GitHub Copilot, Novice Programmers, Computer Science Education, Learning Experience

1 INTRODUCTION

Powered by recent advancements of Deep Learning, Large Language Models, like GPT-3 and its recent successor GPT-4 [25], started developing quite rapidly [5], which made it possible to automate code generation for real world programming tasks [33]. This has paved the way for releasing many AI-driven agents over the past years, that have been trained on millions of lines of code that is available on internet. It resulted in the release of such already well-known models like ChatGPT [24], OpenAI Codex [23], GitHub Copilot [10], DeepMind AlphaCode [12], Google PaLM [11], Microsoft CodeBERT [8], etc. [15]

Code generation tools are such models that can automatically generate source code or actual pieces of code based on input specifications, templates, or models [13]. Some examples may include plugins in integrated development environments (IDEs), free-access websites that provide interaction with such generators, etc. With their dramatic improvement over the past years, it is believed they have the potential to revolutionize the coding process and make programming more productive and accessible [18].

Recent studies show that in computer science education there are a lot of potential areas of code generation tools usage. [6] Such tools have many ways of application to help both educators and novice programmer students in computer science education process. Teachers, in particular, could provide a more effective way to teach programming, vastly save their time and consequently allow to focus on more advanced topics. They can also support differentiated instruction by allowing students to work at their own pace and complexity. However, teachers might need to invest more time and resources into learning and bringing these tools in their teaching practices. Regarding students, especially novice programmers, code generation tools may help making programming less intimidating and more accessible [18]. Students may use these technologies as assistant to generate boilerplate code or guide their coding experience. Novice programmers generally perceive code generating technologies positively as they can help reduce the time and effort required to write code, therefore reduce the cognitive load and complexity of tasks, which can increase student's interest in programming and reduce frustration [18].

The studies, related to the topic of this research, include investigation of code generation influence on different aspects of our lives and societal factors [17, 20], some focus on software development and programmers [2, 4, 7, 28, 33], others bring insight on impact on education [3, 16, 22, 26, 29, 31], educators [3, 15, 22, 27, 31, 32, 34], on student's learning and performance in general [16, 22, 26], on the effects of specific tools on student's learning and performance in particular [9, 21, 27, 29–31, 34]. However, they are either too general in relation to the specifications of this research, or they address only partial elements of what this research is aiming to investigate. We want to pull together the larger picture of code generation tools' impact on learning experience and tie it up from the novice programmers perspective. Furthermore, there is still limited information about the impact of code generating tools on learning experience of novice programmers, especially on such factors as student motivation and engagement in educational process that in particular is the specification of our research. That is due to the fact that learning experience is broad concept and all of its factors can not be studied within this research. The study may bring insights to other aspects of code generation tool's influence on students and their study process, such as programming skills or confidence development.

TScIT 39, July 7,2023, Enschede, The Netherlands
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Therefore, the objective of research is to investigate the impact of different code generation technologies on novice programmers and education as a whole, and on experiences of novice computer science students in particular. Examining literature in this area will help to unite the knowledge about recent findings and influences of code generation tools that have been reported in studies.

In order to understand the topic more in depth, we provide the literature overview of the usage of code generation tools in higher education by analysing collected studies that have been held in the field. Moreover, we address the objective by analysing the level of interest of novice programmers towards and feedback on using these tools, as well as by investigating learning outcomes of students who use these tools.

RESEARCH QUESTIONS

The above-mentioned objective leads us to the following research question that can be answered by investigating the following sub-questions:

Main RQ: What are possible influences of code generation tools usage in higher computer science education on novice programmer's learning experience factors?

(1) What does literature say about the use and influence of code generation tools in higher computer science education?
(2) What is the first hand learning experience of novice programmers with the use of code generation tools in higher computer science education?
(3) What do novice programmers say about the influence of code generation tools on their engagement and motivation in programming learning process?

With novice programmers we imply students in the beginning of their programming journey in computer science education courses. With learning experience we mean a multifaceted concept which includes the following factors: understanding of programming concepts and syntax, ability to solve programming problems, to work with complex code bases, motivation, engagement and self confidence with programming, awareness of best practices and coding standards, coding efficiency, effectiveness, debugging skills. In this paper we focus on the facet of the student's motivation and engagement in the process of learning programming.

Structure of the research

The structure of this study is as follows. The 2^{nd} section describes the methodology that is used under this research to answer the research questions. An extensive overview and analysis of the related works in the field of code generation tools usage of computer science courses at higher education insitutions that is given in the section 3. Section 4 discusses the process of interviews and surveys that were handled in order to investigate the experience of novice programmers with code generation tools during their studies. Next, section 5 addresses further discussions on the topic of the study. Finally, in section 6 we provide conclusions of this research.

2 METHODS OF RESEARCH

There are several steps for conducting this research. First of all, the literature overview and analysis will be performed to address current researches in the field of code generation in computer science education. Their findings about the usage of code generation tools and their effects on student's learning process allow to understand the basics about the field and draw a few conclusions based on previous researches in this area. In its turn, this will lead to answering first research question defined in the section 2 of this research. To gather relevant literature, we have used search engines such as Scopus, Google Scholar and IEEE. With such terms as "code generation tools / technologies", "novice programming" and "higher education" and all the ways of combining those we have found several researches that have done investigations in this area.

Secondly, in order to answer last two research questions that with literature overview will help to answer main research question, survey and interviews were conducted with first year computer science student's participation. Interviews and surveys were chosen as the method of investigation with the aim of gathering first-hand experience of novice programmers with code generation tools during their studies. The lists of the questions were prepared at the earliest stage of the study. These lists can be seen in Appendix A section, where Survey questions and Interviews questions sub sections contain survey and interviews questions accordingly.

Following university procedures for user studies, we prepared informed consents for both survey and interviews. After the agreement of ethical committee, the next stage of the study, that is the recruitment of the participants, has started. The students of the first year computer science course are the target population defined for this study. There were several techniques used in order to engage these potential participants, namely promotion of the survey in social medias and through announcements on educational platforms in the current course the students are following.

Thirdly, we reviewed the quantitative data from the survey using descriptive analysis of statistics to summarise collected data. The qualitative data from the interviews is analysed using thematic analysis to identify common themes and patterns in the data. Last but not least, we compare gathered information to answer the main research question. This includes a comparison of research findings from literature overview and outcomes of the conducted study involving surveys and interviews.

3 CODE GENERATION TOOLS IN EDUCATION

The topic investigated within this research is code generation tools usage by novice programmers and its effect on their learning experience, engagement and motivation in particular. It lays on the intersection of different areas that have been previously explored. The main goal of this literature overview is to illuminate findings of the studies done in such areas as: possible applications and implications of AI and code generation in education; their usage in introductory programming; their influence on both educators and students; overview of works on particular code generation tools and findings towards the influence on novice programmers and their learning experience. Several studies that we found show interest in the intersection of the topics we define here. We also present a brief overview of articles collected for this literature overview in literature summary table in Appendix A section.

3.1 AI Applications in education

The concept of computer programs capable of generating other computer applications dates back several decades, but has only recently gained much attention [19]. Nowadays, generation technologies made impact on different spheres of life, including education and programming education in particular. To understand this influence we take a closer look at already existing ways to apply generation tools in education system as well as at possible ideas of further applications. It sheds the light how these tools fit in student's learning and can help to investigate its further impact on it.

Looking at recent works in the area we see generation technologies having vast variety of applications along with resulting implications in educational realm. We further review the works that suggest ways of applications of both AI and code generation tools in different aspects of education and discuss their conclusions.

3.1.1 Artificial intelligence. The systematic review of empirical research on literature from 2011 to 2020 of Ouyang et al. (2022) focuses on AI in online higher education in particular. They distinguish four main functions of applications of AI among the reviewed articles. The majority of them (66%) address predictions of student performance in online higher education context. They suggest applications as prediction of dropout risks, student academic performance, student satisfaction about online courses. Several articles (22%) fall to factor of resource recommendation and report improvement of learning quality of students, their satisfaction levels and programming capabilities in e-learning context.

Couple of articles (6%) concentrate on automatic assessment with the help of generation technologies, examples include formative assessment of programming performances and problem-solving capacities, automated essay evaluation. Their results indicate enhancement of learning interest, positive attitudes, degree of technology acceptance and problem solving amid students. As for automated essay evaluation, the study reports improved writing performance of students. Other two articles (6%) address the last function, that is improvement of learning experience of students. One example of such work presents interactive learning with VR tools that made learning process more engaging and motivating compared to traditional methods to learn. AI-based virtual assistants, chatbots, such as ChatGPT [24] can provide instant support and guidance to students. They can answer frequently asked questions, provide study materials and information on a variety of educational topics.

Recent work of Holmes et al. (2023) discusses such fields of applications in education as learning in collaborative environments, continuous examination assessment, moreover, states that AI can play a role of learning companion and teacher assistant. Regarding collaborative environments, AI-powered collaboration tools can facilitate collaboration between students by manipulating the content of the code actively, providing real-time feedback and encouraging knowledge sharing. This stimulates collaborative problem solving and enhances the overall learning experience. In addition,the work discusses the use of AI in continuous test analysis, where AI-powered tools can help automate coding exercises and evaluate student submissions, enabling timely and objective assessment. This approach allows teachers to provide feedback immediately and monitor student progress closely, which promotes dynamic and flexible assessment process. The study confirms that AI can be a valuable learning partner and teacher's assistant. AI capabilities can provide personalized instruction, flexible learning styles, and content tailored to individual students. Such support system may help students to navigate complex design concepts, refine, and improve their skills at their own pace.

Ouyang et al.(2022) present some thoughts based on their review results regarding the ability of AI-powered systems to guide students in their learning practices. They suggest that to achieve a high quality of prediction, assessment or recommendation, AI-enabled systems or models should first take into consideration students' diverse characteristics from both learning processes and summative performances, and secondly use advanced AI algorithms to achieve precision of the outcome in order to improve students' learning motivation, engagement and performance.

3.1.2 Code generation tools. The study of Mosterman (2006) explores new teaching opportunities with introduction of automatic code generation in engineering education, specifically in the domain of embedded control systems. Researcher highlights that the advent of automatic code generation technologies shifts in the required skill set for software engineers. Traditionally, software engineers were responsible for manually writing code to implement algorithms in embedded control systems. However, the role of engineers changes from being code producers to code reviewers. Mosterman's work warns software engineers increasingly rely on automatic code generators to produce the necessary algorithms. As a result, the emphasis in engineering education shifts from teaching students writing write code from scratch to teaching them how to effectively review and validate automatically generated code. This change in focus acknowledges growing importance of understanding and assessing code generated by these tools, ensuring its correctness, efficiency, and adherence to design requirements. By studying the implications of automatic code generation in engineering education, Mosterman's work highlights the need for educators to adapt their curriculum to reflect this shift. It becomes crucial to provide students with the skills and knowledge to critically evaluate automatically generated code, identify potential issues or improvements, and ensure the overall quality of the generated software.

The work of Becker et al. (2023) provides the overview of three tools for code generation, namely Codex, AlphaCode and Amazon CodeWhisperer [1] and discusses their potential and challenges within computing education community. The opportunities they identify are: production of new learning resources, generation of programming exercises, code explanations, illustrative examples, code solutions for learning purposes, code reviews of solutions.

3.2 Code generation tools in introductory programming

Introductory and novice programming have been the focus of lots of works in computer education area [3]. Most recent comprehensive analysis of literature overview over introductory programming studies by Luxton-Reilly et al. (2018) reported decent number of papers corresponding to category of either measuring or improving student's engagement. However, none of them related to the topic of code generating tools. In this section we bring together the topics of introductory programming and code generating technologies and give a brief overview on the works made in this area.

Li et al. (2022) note that automatic code generation could make programming more accessible and help educate new programmers. By automating certain aspects of code creation, these tools can lower barriers to entry and enable individuals with limited programming experience to engage more effectively in learning.

Studies of Kazemitabaar (2023), Finnie-Ansley (2022) and Becker et al. (2023) highlight valuable role of OpenAI Codex as a supportive assistant for students, that fosters their learning journey as they navigate programming practices. These studies demonstrate potential benefits of AI-powered code generation tools integration in introductory programming education, promoting greater comprehension, productivity, and overall learning outcomes improvement.

3.3 Impact on teachers and students

When talking about applications of code generation tools in programming education we also have to take into account its influence on educators and novice programmers in particular, as that is the important factor of the acceptance of technology and, what is more interesting for us, its impact on learning experience of students. Some works divide possible applications of artificial intelligence in education in three categories: teacher supporting, student supporting and student teaching(system supporting) [14, 31]. Within our research we pay more attention to student- and system-facing side of the question. Therefore, we take a closer look at the situation from the perspective of view of both teachers and programmer students and try to identify all possible advantages and pitfalls of its use as reported by related studies that we found.

3.3.1 Educators. Several studies have emphasized the potential of AI-generated code technologies to support teachers in automating exercise generation and providing textual explanations of the code. Sarsa et al. (2022) and Marwan et al. (2019) have specifically highlighted the benefits of these tools in assisting educators in creating a wide range of programming exercises and offering detailed explanations, thereby enhancing student learning experiences [21, 29]. Becker et al. (2023) states code generation tools open doors for new pedagogical approaches. As an example, these tools not only facilitate generation of exercises but can make clear explanations to them, surpassing what teachers themselves could produce. By utilizing these technologies, teachers can save time, effort and be able to provide valuable resources to their students. Such capabilities of code generation tools offer educators access to a diverse set of resources that can greatly enrich student learning experiences, providing them with comprehensive support and guidance throughout their programming journey. The advent of these tools has raised concerns about readiness of teachers to handle the significant influence on educational practices [3]. The researchers therefore emphasize the need for urgent review and modification of instructional approaches and traditional practices in response to the advancements in code generation technology.

3.3.2 Students. Regarding novice programmers, code generating tools can help them with solving programming tasks [34], resolving programming bugs [30], assist with software engineering projects [16], generating exercises with explanations and illustrative examples of programming constructs and algorithmic patterns or mapping problem to solution [3], preparing for examinations.

Becker et al. (2023) address code generation tools ability to capture relevant relationships between variables in problem statement as the helping hand for students to learn to communicate algorithmic problems clearly. This may encourage student to focus on understanding topic more in depth. The input for tools needs to be precisely formulated as suggestions provided by code generation tools are sensitive to changes in problem statement itself.

Some studies identify the benefits of using code generating technologies for students as improved performance and learning [21, 26], reduced cognitive load [3]. Works of Marwan et al. (2019) and Ouyang et al. (2022) report students increased productivity, consistency in the code quality and reduced errors when using those tools. Recent review of empirical studies reports an improvement of academic performance of students, online engagement and participation when introducing AI applications in educational process [26]. Additionally, these tools can assist students in generating exemplar solutions for programming exercises and code reviews of solutions. Researchers point out on the variety of solutions these tools can propose, so that educators could introduce students to the diversity of ways that a problem can be solved.

However, identified drawbacks of code generation tools usage include a possibility of students to have lack of understanding of the fundamental programming concepts. Several studies discuss the issue of bias, bad habits and over-reliance [3, 6]. Becker et al. (2023) raises concerns about its suitability for novices, given that public code is often contributed by professionals, thus may not reflect the expected quality for beginners and desires of educators.

Talking about bias, which exists in the data used for training of AI models that power code generation tools, educational community has to consider error prompt of generated code. To give an example, the analysis of the solutions generated by AlphaCode revealed that 11% of Python solutions were syntactically incorrect and 35% of C++ solutions failed to compile [18]. Codex also may suggest solutions that appear to be correct at the first glance, but may not actually accomplish the task intended by the user[6] . Researchers warn novices about issues with potential over-reliance on the tool rather than developing independent coding skills [3]. These potential pitfalls highlight the importance of caution and critical thinking when using such tools, especially for novice programmers. Balancing the use of these tools with developing basic programming and problem-solving skills is important to ensure student's learning progress and understanding of theoretical part.

3.4 Particular code generation tools

There is a variety of studies on the use of a particular code generating tool in introductory programming. We further continue the discussion of the impact of code generation tools on student learning experience by taking a look at these works.

3.4.1 ChatGPT. Recently Tajik and Tajik (2023) published a comprehensive examination of potential applications of ChatGPT in higher education. The researchers state that students can definitely benefit from ChatGPT, however, they need guidance on how to work with GPT to fully understand a specific topic by thinking logically and assessing GPT's output. As of educators, the study states the integration of GPT in teaching practices can significantly reduce workload of teachers and allow them to better focus on

all individual students' needs. Researchers name vast amount of limitations of GPT, such as: inability to reason physical and social world and to understand emotions and idioms; lack of reasoning and self-awareness and knowledge of ethics/morality, transparency, reliability; presence of bias, errors and discrimination in data; issues with robustness and security; and plagiarism question being raised. Tajik and Tajik conclude that the potential of GPT and other LLM tools for positive impact on education cannot be ignored, however it is important to acknowledge and address their limitations.

Surameery et al. (2023) investigate ChatGPT performance in solving programming bugs The researchers state ChatGPT can assist in solving programming bugs through debugging assistance, bug prediction and explanation. Its code analysis, knowledge representation, and natural language generation capabilities make it well-suited for these tasks. However, important to note that ChatGPT is not a perfect solution, as the quality of outputs depends on training data and system design. To ensure optimal results, it should be used alongside other debugging tools and techniques for validation and bug-free code. Researchers conclude by combining the strengths of ChatGPT with other tools, developers can enhance their understanding, identify and fix bugs more effectively.

3.4.2 OpenAI Codex. The study of Finnie-Ansley (2022) discuss that the way of teaching introductory programming has the possibility to be change dramatically in the next years. The researchers find the evidence that Codex performs better when given input that is precisely defined and succinctly written, and in those tasks which do not require adaptation of already existing code. The study confirms the capability of Codex to solve tasks beyond CS1 problems. The impact of question complexity on its performance in programming education is uncertain. The researchers advice educators to work on the ways to adapt their classroom practices and use this freely available technology to enhance student learning effectively.

Broad investigation of Chen et al. (2021) on Codex brings up that the tool helps users familiarize themselves with new code bases, allows them to have draft implementations, supports in education and exploration, allows non-programmers to write specifications and reduces context switching for experienced programmers. However, they also identify risks: over reliance on the tool, misalignment of generated code and user's intentions, argue about bias, security implications, influence on economic and labor market impacts.

The work of Sarsa et al. (2022) examines the use of Codex for code explanations and their assessment, generation of programming exercises. Their findings confirm that large language models, like OpenAI Codex, are performing well in tasks even with limited or no task-specific examples as input. It suggests such models offer significant potential for programming course designers, although the challenges should be acknowledged as well. The analysis of this study demonstrated impressive results in generating sensible programming exercises with sample solutions and automated tests, despite minor accuracy and quality issues that can be easily addressed. Generated code explanations were also promising. Sarsa et al. concludes with high expectations for continuous evolution of models like Codex and foresees only their improvement over time.

3.4.3 GitHub Copilot. Bird et al. (2022) bring insights on usage of AI-powered pair programming with Copilot. They state that AI-powered developer tools like Copilot represent a significant shift in the software development landscape, with the potential to assist developers in various tasks beyond just writing code. These tools can enhance code review, suggest fixes for defects and build failures, automatically generate tests, refactor complex code bases, and even generate code comments and documentation. However, the challenge lies in creating a user experience that effectively assists developers without hindering their workflow. Researchers say trust in AI-powered tools is crucial, as developers need confidence that the tools are making right decisions and not introducing vulnerabilities or performance issues. Understanding the dynamics between developers and AI tools, tracking AI-generated code throughout the software development life cycle, and developing provenance tools to track the origin of generated code are important areas of further research. These considerations are essential for leveraging AI-powered tools effectively and making informed decisions in software development organizations.

Recent work of Wermelinger (2023) on GitHub Copilot compares it with "most performant Codex model, that can solve typical CS1 problems" and argues if the tool can solve simple programming problems of introductory programming courses. The results of the study show that the Copilot clearly fails to be correct with responses. While sometimes a deeper understanding of the problem is demonstrated, it often leads to incorrect or unnecessary suggestions that resemble variations on the same topic. Communication with Copilot can also be difficult to achieve appropriate answers, and require specific instructions. Although Copilot is a code similar to human-written code, it requires students to have a strong grasp of linguistic syntax and semantics in order to process that proposal. Copilot's explanations provide a low level of understanding of of the law but may have omitted important parts. Students must learn to write clear documents, understand them without running code, and handle errors independently. The study defines strength of Copilot as saving time for students by completing and evaluating code scripts with fewer syntactic errors and allowing teachers to focus on higher-order thinking. The researchers suggest that teachers should consider using more challenging problems to promote algorithmic thinking and debugging skills of students, and grading methods may need to be adjusted to accommodate partial solutions. The study concludes Copilot is a useful tool for solving introductory programming problems, but it does not replace the necessary skills and understanding needed in programming education.

3.4.4 DeepMind AlphaCode. Extensive study on AlphaCode by Li et al. (2022) describes code generation tools as the ones having potential for a positive, transformative influence on society, and wide range of applications including computer science education. The researchers state AlphaCode is capable of innovative solutions to previously unseen programming problems and performs at levels comparable to weaker competitors. The study of Becker et al. (2023) also found that simplifying the problem description significantly increase the success rate of AlphaCode. Detailed analysis of the work of Li et al. shows that AlphaCode does not simply copy previous solutions or exploit problem weaknesses, but highlights its ability to solve complex problems that require deeper understanding.

4 SURVEY AND INTERVIEWS

4.1 Survey

4.1.1 General familiarity. We gathered 39 survey responses and those show interesting insights and evoke some discussion. All of the respondents have shown familiarity with ChatGPT(GPT-4), which is not surprising, as the platform has been extremely popular lately. ChatGPT has been also marked as mostly used, as 77% of people mentioned they have tried this tool on practice.

Talking about the tools made for code generation, the vast amount of respondents know about GitHub Copilot (69,2%) with 25,6% of people who tried it on practice, OpenAI Codex (30,8%) and Tabnine (15,4%), with 25,6% and 12,8% used on practice respectively. The least known tools appear to be DeepMind AlphaCode (7,7%), Microsoft CodeBERT and Google PaLM with 5,1% each. These three tools have never been used on practice by survey respondents.

Only 6 people(15,4%) tried code generation tools before they started university computer science course. Among those people two were quite familiar with code generation tools, only one have always used them in coding practices. Half of students who had previous experience with such tools marked ChatGPT as most useful one. Tabnine and Copilot as well as "none of the tools" response all share a second place in the question about usefulness.

4.1.2 First year computer science course. Out of all 39 responses, 21 (53,8%) have used code generation tools during their first year of computer science course. Among tools ChatGPT (GPT-4) still takes absolute lead with all of respondents using it during first academic year. GitHub Copilot is second in the list being used by 7 people (33,3%), which is followed by Tabnine (23,8%) and Copilot (14,3%).

Regarding code generation tools that have been used specifically within student's coursework, ChatGPT is still in lead with 66,6% of students using it for study-related tasks. Second most popular answer is using none of the tools (28,6%). Copilot takes third place with (23,8%), while Codex and Tabnine received only 1 vote each (4,8%). Among respondents 9,5% of people have used code generation tools frequently, 61,9% used them few times at least.

Next survey question showed in which programming tasks specifically students used these tools. Most votes went for project work and exam preparation (57,1% each), assignments (47,6%), labs (33,3%) and group work (23,8%). There were 3 people who have not used any tool for any type of coursework, that is 14,2% of all respondents. Some rare answers included usage for example code (4,8%).

We can say that majority of students (76,2%) who completed the survey feel that code generation tools use in their practices make programming fundamentals and concepts learning less difficult. Vast amount of people (47,6%) feel such tools also increase their ability and skills of problem solving in programming tasks introduced by their computer science program. The same percentage of people feel code generation tools make it more fun to learn. 42,9% report increased interest in topic learned when using tools, and only 38,1% feel the same about engagement in learning programming.

The survey also have an open question that relates to code generation tool's impact on ability to learn programming fundamentals and concepts. Most respondents reported positive effect, tools have helped them to understand better some concepts that are harder to grasp, as they explain terms, pieces of code, methods, classes or libraries, functionality behind them, and detected small mistakes in code. Some students outline code generation tools ability to give lots of related information, inspiration and "personal learning experience due to their flexibility". Overall, respondents reveal increased understanding and algorithmic thinking, enhanced learning and saved time "to read unclear, far too technical documentation or explanation using concepts that I do not know" and to debug.

4.1.3 Motivation, engagement, confidence. Regarding code generation tool's increase of motivation, the answers are bit ambiguous: one third (33,3%) of respondents agree that usage of code generation tools increase motivation towards learning, other third (33,3%) stays neutral, last third completely (9,5%) or disagree (23,8%). Results of question about the increase of student's engagement were slightly different: most of respondents (38,1%) would indicate agreement with the statement, 4,8% would strongly agree, 33,3% would again stay neutral, other 28,6% would completely or disagree.

With further open questions about factors of influence on student's motivation and engagement we could distinguish five main sources. First one (not by amount of answers) is the end-goal, profession, job, money. Second is other people as the source: teamwork, friends in course, teachers. Third, how interesting the project, assignment, practical, course is. Fourth is self-development: gaining more knowledge and skills, getting better at programming, learning new things that of their interest. Finally, fifth - desire to build, coding process itself, "applying theory and creating something that has a purpose", "feeling of my implementations being used", "challenging myself with difficult problems and finding solutions".

The survey questions about abilities to write the code have shown in general that respondents feel less confident when using code generation tools. Important to mention that for students who are initially pretty confident in their programming abilities it does not make big difference when involving code generation tools to their practices. However, for people who generally have lack of confidence in programming, it only gets worse when involving code generation. Thus, we see 4,8% of people disagreeing in the first case, increasing to 19% in the second case when involving code generation. In the end only 26,8% of respondents feel that code generation tools positively affect their confidence in programming abilities.

4.1.4 Conclusions. Overall, 57,1% of respondents who had experience with code generation tools in first year of computer science course would positively rate this experience. 14,3% would strongly agree to the statement, 19% would stay neutral and only two people (9,6%) would strongly or disagree about its positive impact.

We also asked participants to compare writing code manually with using code generation tools in terms of learning outcomes and practical applications. The majority considers writing code manually feels better as there are still a lot of issues associated with code generation tools. As they report, automatic code generation often results in "code that is hard to understand, which makes debugging extremely difficult and ultimately does not teach you how to solve certain problems", or such code that "doesn't have its intended purpose". Respondents also warn about possible over-reliance on tools, which shows their awareness that such scenario could happen. However, students say code generation tools suit great when being used for inspiration, solving trivial tasks or when applied

once they know fundamentals already. They suggest starting using them when you already have a great knowledge basis of fundamentals and concepts so that they can assist you further in learning deeper. Thus, when using code generation tools as extra source of help, respondents emphasize saved time for debugging, "when needing to write short, simple, repetitive methods for a project, like making "getters" and "setters" ... , or writing code to test a part of the system", they also point out on "reduced stress/boredom of writing repetitive code, fixing small syntax mistakes, searching the right function and more small tasks". Overall, students still prefer to solve complex problems manually, "both to learn and to have some challenge", but see code generation tools as a great assistant in further knowledge development.

The question regarding further introduction of code generation tools in computer science course showed most students (52,4%) would like more interaction with them. Among those who support the idea, 14,3% strongly agree, 38,1% would agree, others would stay neutral (14,3%), disagree(23,8%) and strongly disagree(9,5%).

This can be explained by latter questions of the survey. We have asked respondents about the main benefits and pitfalls of using code generation tools that they can distinguish from their personal learning experience. Common benefits include help when stuck on problem, have doubts, in need to check answers, debug, explain or improve the code, enhance the knowledge. Respondents report code generation tools are "excellent learning aid" and can effectively save time, increase their productivity and efficiency, give inspiration and are flexible and personalisable for learning experience. Regarding disadvantages, lots of respondents again warn about over-reliance on the tool, saying it "can become addictive or overused", "make people lazy" to write code manually, "can prevent from understanding new concepts". Moreover, they say generated code is often hard to understand or does not have clear goal which makes its further usage harder. Some report reduced interest in becoming proficient in fundamentals, "make [students] feel more confident in things [they] know little about". Students suggest to use code generation wisely, either for time saving on "doing boring stuff you already know" or for personal code and knowledge improvement.

4.2 Interviews

We have conducted interviews with five first year computer science students. The answers have showed us that all students know about code generation tools and use them frequently for different learning purposes. We have explored their initial expectation and impression of the use of such technologies and found out in most cases the expectations were not high, but results were quite impressive. Students did not expect code generation tools to be that smart and did not believe they would help them as much.

Regarding the experience with code generation tools in first year of computer science course, respondents state these tools have been a great learning assistant and been used extensively for coding assignments, projects, exam preparation, understanding theoretical part of the course, personal coding practices. Students report these tools are of great help when in need of making easy methods, quick functionality or logic, building components or finding little bugs. We have also asked students about any benefits or pitfalls regarding their learning process when using code generation tools. Among advantages they mention reduced time spent on problem or bug, as code generation tools can detect issue in code, on learning overall, as they explain how certain things in code work very effectively. Another positive thing is making easier to understand some topics, parts of code, methods or classes. Students report such tools provide fresh perspective towards the problem approach. In general, students consider code generation tools being great supplement for learning. As of disadvantages, the main one that students mention is possibility of over-reliance on the tool, they say it "could be bad for lazy people" or "can make people lazy". This, of course, depends on person and their intentions of such tool's usage.

Amid challenges of code generation tools usage respondents often mention their "output is only as good as input", meaning that users have to really make effort in order to explain the task in single prompt. Therefore, students need to learn how to communicate the problem properly so that the algorithm understands what is needed. However, it could be seen as a positive outcome as well. In addition, respondents often mention tools being of limited knowledge, so that they can sometimes not help as much as user wants or give solution that does not match user's intentions.

Regarding code generation tool's impact on understanding of programming concepts and syntax, all students report positive effects, they are trying to use tools in a smart way taking as much benefits as possible. Respondents say they often asked for explanation of some hard concepts that were hard to understand. Moreover, students report saved time with use of code generation tools, compared to traditional learning methods. Overall tools made student's learning much faster, increased effectiveness and productivity, helped to explore many possible approaches to the problem. Respondents do not report much impact on confidence in their programming abilities. Only one student said that the confidence actually increased due to the fact that "you still have to have knowledge to understand [generated code] and the functionality behind it".

As this research explores the impact on such elements of learning as motivation and engagement, we have asked students how code generation tools influence these factors. We found out that the use of code generation does not influence students motivation much, as many say, due to completely different sources of motivation towards programming learning. However, we can clearly identify some positive influence on student's engagement in learning process. Respondents say code generation helps them not to waste time on basic functionality that they have mastered already or on "tiring time consuming tasks". Because code generation tools give many examples and explanations, they tend to positively impact problem-solving skills of respondents, decrease time on googling encountered problems, "it is almost like having mentor by my side" they say. Code generation tools speed up programming process much, helping students create things much faster. All these factors have very positive impact on student's curiosity, joy towards and engagement in learning process as they report.

Interviews show students feel more accomplishment when writing code or solving programming tasks on their own, they also prefer learning new things with manual writing and use code generation tools on later stages for saving time and broadening knowledge. Due to the fact that code generation tools are not perfect yet, they can often give wrong or bad answers and students prefer to switch to traditional problem approaches in such situations.

Talking about impact on student's ability to develop programming proficiency, all respondents state code generation tools are much helpful, for basic tasks especially. As students say, tools increased efficiency, decreased time for debugging, solving tasks and understanding concepts. Some respondents mention code generation "saves" brain from thinking and if being overused can lead to slowed development of problem-solving and topic understanding.

We asked respondents whether they would recommend code generation tools use to other novice programmers. The common conclusion is that such technologies are of most help when already having sufficient knowledge basis on the topic and at least some experience. Respondents advice that students should understand how and when to use tools to achieve best learning outcomes for themselves. This also includes comprehension of consequences of over-reliance on technologies and cheating with their help.

Regarding further integration of code generation tools in educational realm, we can conclude from students responses that such technologies should be introduced and not only the way how they work, but also how they can be used most effectively should be studied. "It would be stupid to ban [code generation tools], for lots it is a helping tool, not an answer tool", as one respondent says. Overall, they can bring lots of advantages, including improvement of student's engagement towards learning process. This can happen "if all students will understand real goal of [code generation tools] and use it in smart way, not for cheating".

5 FURTHER DISCUSSION

From the literature overview we found that code generation tools have wide range of applications and opportunities in introductory programming course. Those include wider range of learning resources, solutions, illustrative examples, and better assistance for both students and teachers in their educational and learning practices. The benefits of their use and effects on novice programmers include increased engagement, productivity, comprehension, as well as saved time and better overall learning outcomes. Still, it is arguable if code generation tools could be involved in novice programmer's educational process without any associated risks.

From survey we discovered that code generation tools have more effect on student's engagement rather than motivation towards programming learning. This might be because the source of motivation is broad and different for individuals. Overall, majority of respondents report positive experience with code generation tools during first year computer science course and use it as helping hand in their practices. Most of students would like code generation tools to be better introduced and involved in education.

Interviews showed us students prefer using code generation tools when they are confident in their basic knowledge on topics, so that they help them to broaden their understanding and problem approaches. Students say code generation tools can be great assistant and speed up programming and learning process. When being aware of risks of the use of such technologies, students can benefit from them much and teachers, therefore, might use these benefits to improve the quality of the course.

However, research findings rise questions about ethics of the use of code generation technologies in computer science higher education. As these tools become more accessible and commonly used, concerns regarding their potential impact on plagiarism cases involving novice programmers can arise. There is a risk of student's over-reliance on tools, thus, plagiarism cases or the submission of unoriginal work can increase. As discussed by Becker et al. (2023), code generation tools enable negative academic practices, allowing students to transfer their teaching to technologies without usual risks associated with traditional practice. This can hinder the detection of fraud and lead to short-term grades being prioritized over true learning. Furthermore, AI-generated code detection is difficult, especially when it closely resembles code suggestions provided by standard IDEs [4]. This blurs the distinction between human and machine contributions and calls into question the notion of plagiarism. Such code reuse can cause licensing problems, as AI models can inadvertently use code that requires proper attribution and compliance [3]. To eliminate the risks, educators and institutions must carefully consider how to address these issues, ensure students understanding of ethical implications code generation tools usage, and promote academic integrity within their programming assignments or projects. Proper guidelines and policies may need to be implemented to prevent and address cases of plagiarism associated with the use of these technologies.

5.1 Limitations

First of all, due to the multifaceted nature of the learning experience, which could not be fully studied within the scope of this research, we had to focus on such factors as student's engagement and motivation. Therefore, the work can provide relevant insights only regarding those factors. Furthermore, the possibility of bias arises as we rely on collected data from surveys and interviews, where participants could give socially desirable or possibly biased answers. Furthermore, analyzed data is of limited sample size, specific time frame and geographical focus, which may limit the generalization of our findings. It is important to acknowledge these limitations when interpreting results of this study and future research should aim to address these barriers to knowledge expansion.

6 CONCLUSION

Code generation technologies are developing rapidly and it is hard to ignore their prevalence among aids for learning within computer science courses. They have a great potential to bring benefits in educational realm if being approached and used in a smart way. The work of Becker et al. (2023) presents such position: AI-powered code generation has the potential to bring both opportunities and challenges to introductory programming and related courses, as well as to educators and students. We agree with this statement and can conclude from gathered information that code generation tools have all the potential to positively influence on student's engagement towards learning process. However, further researches should carefully consider associated risks that have been found with the help of literature overview, survey and interviews, as well as take into account warnings of researchers in order to find the best way and moment to integrate code generation tools into educational practices of computer science courses.

APPENDIX A

Literature summary table

No	Author	Article name	Important findings
1	Becker et al. (2019)	50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research	"AI-generated code is on the way to being firmly part of the programming education landscape, but we do not yet know how to adapt our practices to overcome the challenges and leverage the benefits."
2	Bird et al. (2022)	Taking Flight with Copilot	AI-powered tools like Copilot have potential to assist developers in various tasks. The challenge lies in creating user experience that enhances productivity, establishes trust and understanding of AI-generated code' impact on software development life cycle.
3	Chen et al. (2019)	Evaluating Large Language Models Trained on Code	"By fine-tuning GPT on code from GitHub, we found that our models displayed strong performance on a dataset of human-written problems with difficulty level comparable to easy interview problems."
4	Finne-Ansley et al. (2022)	My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises	"Codex is able to solve most CS2 questions, performing similarly to students in the top quartile of the class that answered the same questions"
5	Holmes et al. (2023)	Artificial intelligence in education	"AI is increasingly being used in education and learning contexts. We can either leave it to others ... or engage in productive dialogue."
6	Kazemitabaar et al. (2023)	Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming	"AI code generators can significantly increase task completion, improve correctness score, reduce encountered errors, and task completion time."
7	Khmelevsky et al. (2023)	Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming	"Automatic software generation tools is a key component of higher education in Software Engineering."
8	Marwan et al. (2019)	An Evaluation of the Impact of Automated Programming Hints on Performance and Learning	"Learners who received textual explanations in addition to code hints perceived iSnap's support as significantly more useful"
9	Mosterman et al. (2022)	Automatic Code Generation: Facilitating New Teaching Opportunities in Engineering Education	"To provide engineering students with necessary skill set to be successful in the field of embedded control systems, it is important to familiarize and educate them on the technology of automatic code generation."
10	Ouyang et al. (2022)	Artificial intelligence in online higher education: A systematic review of empirical research from 2011 to 2020	"AI is proved to be positive to enhance online instruction and learning quality by offering accurate prediction, assessment and engaging students with online materials and environments."
11	Sarsa et al. (2022)	Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models	OpenAI Codex is capable of generating sensible, novel and readily applicable exercises.
12	Surameery and Shakor (2023)	Use Chat GPT to Solve Programming Bugs	"By combining the strengths of ChatGPT with the strengths of other debugging tools, developers can gain a more complete understanding of their code, and identify and fix bugs more effectively."
13	Tajik, E. and Tajik, F. (2023)	A comprehensive Examination of the potential application of Chat GPT in Higher Education Institutions	Application of ChatGPT brings lots of benefits in education as student-, teacher- and system-facing tool.
14	Wermelinger, M. (2023)	Using GitHub Copilot to Solve Simple Programming Problems	"Copilot is a useful springboard to productively solve CS1 problems."

Survey questions

Down below you can see list of questions for the conducted survey for this study. The questions were grouped to make it easier for the respondents to navigate. The types of the questions were also indicated in this list to give a general idea of the types of answers.

General familiarity with code generation tools

(1) Which code generation tools do you know about? Multiple choice
(2) Which code generation tools have you tried on practice? Multiple choice
(3) Have you used any code generation tools before you started computer science course? program Yes/no
   (a) How familiar were you with code generation tools before computer science course? Scale 1-4
   (b) Overall, how often did you use code generation tools compared to writing code manually? Scale 1-4
   (c) Which code generation tools did you find most useful? Multiple choice

Code generation tools usage in computer science course

(4) Have you used any code generation tools during your first year of computer science course? Yes/no
   (a) Which code generating tools have you had experience with during your first year of computer science course? Multiple choice
   (b) Which code generating tools have you had experience with within coursework? Multiple choice
   (c) In what specific programming tasks or projects did you use code generation tools? Multiple choice
   (d) How frequently do you use code generation tools when writing code for your coursework? Scale 1-5

Code generation tools impact on student Questions from this section are formulated in such way so that students can mark how the statement relates to them on scale from 1 to 5.

(5) In general, I am confident in my ability to write code.
(6) I am confident in my ability to write code with the use of code generation tools.
(7) I feel a sense of accomplishment when successfully generating code using code generation tools.
(8) I feel that code generation tools usage has positively affected my confidence in my programming abilities.
(9) I feel that code generation tools usage enhanced(made less difficult) my learning of programming fundamentals and concepts.
(10) I feel that code generation tools usage increases my ability and skills of problem solving in programming tasks introduced in my computer science course.
(11) I feel that the use of code generation tools increased my interest in topics that have been learned in my computer science course.
(12) I feel that the use of code generation tools increases my engagement in programming learning in my computer science course.
(13) I feel that the use of code generation tools increases my motivation in programming learning in my computer science course.
(14) I feel that the use of code generation tools made it more fun to learn programming topics introduced in my computer science course.
(15) Overall, I would positively rate my experience with code generation tools in my first year of computer science course.
(16) I wish there was more interaction with code generation tools introduced in my computer science program.
(17) Overall, I would recommend the use of code generation tools to other novice programmers for learning purposes.

Going deeper in impact All the questions in this section are open questions.

(18) What are the main factors that influence your motivation and engagement in programming learning process in your computer science course?
(19) How do you think code generation tools compare to writing code manually in terms of learning outcomes and practical application?
(20) How do you think using code generation tools has impacted your ability to learn programming fundamentals and concepts?
(21) What are the main benefits and pitfalls of using code generation tools for programming learning that you can distinguish from your personal learning experience?

Interviews questions

The list of questions for the interviews conducted within this research are provided below. The questions were grouped to make it easier for the respondents to navigate.

General familiarity with code generation tools

(1) What do you know about code generation tools in general?
(2) When was the last time you have used code generating tools?
(3) What was your initial impression or expectation of code generation tools before using them?

Code generation tools usage in computer science course

(4) Can you describe your experience using code generation tools during your first year of computer science education?
(5) In what specific programming tasks or projects did you use code generation tools?
(6) What were the advantages or pitfalls you observed when using code generation tools?
(7) Did you face any challenges or difficulties while using code generation tools? If yes, can you elaborate on those?
(8) How would you describe your overall experience with code generation tools during your first year of computer science course?
(9) What are the main factors that influence your motivation in the process of programming learning?
(10) What are the main factors that influence your engagement in the process of programming learning?

Code generation tools impact on student

(11) How do you think code generation tools influenced your understanding of programming concepts and syntax?
(12) How did the use of code generation tools influence your engagement in the programming learning process? Why?
(13) How did code generation tools impact your efficiency and productivity in completing programming tasks?

(14) In what ways did code generation tools influence your motivation to learn programming? Please provide examples.
(15) Did using code generation tools make programming more or less enjoyable for you? Why?
(16) Did you feel a sense of accomplishment when successfully using code generation tools to generate code? Why or why not?
(17) How did code generation tools affect your problem-solving skills in programming tasks?
(18) Did the use of code generation tools impact your confidence in your programming abilities? If yes, how?
Going deeper in impact
(19) How do you think code generation tools compare to writing code manually in terms of learning outcomes?
(20) Were there any coursework tasks performing which you felt the need to switch from code generation tools to writing code manually? Why?
(21) Do you think using a code generation tool has influenced your ability to develop your programming proficiency? Why?
(22) Would you recommend the use of code generation tools to other novice programmers? Why or why not?
(23) Based on your experience, do you think code generation tools should be used extensively in computer science education? Why or why not?

REFERENCES

[1] Amazon (2023). CodeWhisperer.
[2] Barke, S., James, M. B., and Polikarpova, N. (2023). Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1):85–111.
[3] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A. (2023). 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*, pages 500–506, Toronto ON Canada. ACM.
[4] Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., and Gazit, I. (2022). Taking Flight with Copilot.
[5] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv:2005.14165 [cs].
[6] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs].
[7] Cheng, R., Wang, R., Zimmermann, T., and Ford, D. (2023). "It would work for me too": How Online Communities Shape Software Developers' Trust in AI-Powered Code Generation Tools. arXiv:2212.03491 [cs].
[8] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong (YIMING), M., Shou (), L., Qin, B., Liu, T., Jiang (), D., and Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages.
[9] Finnie-Ansley, J., Denny, P., Luxton-Reilly, A., Santos, E. A., Prather, J., and Becker, B. A. (2023). My AI Wants to Know if This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. In *Australasian Computing Education Conference*, pages 97–104, Melbourne VIC Australia. ACM.
[10] GitHub (2022). Copilot:Your AI pair programmer.
[11] Google (2022). Pathways Language Model (PaLM): Scaling to 540 Billion Parameters for Breakthrough Performance.
[12] GoogleDeepMind (2022). Competitive programming with AlphaCode:Solving novel problems and setting a new milestone in competitive programming.
[13] Herrington, J. (2003). *Code Generation in Action*. Manning Publications Co., USA.
[14] Holmes, W., Bialik, M., and Fadel, C. (2023). Artificial intelligence in education. In Duggal, P., editor, *Data ethics : building trust : how digital technologies can serve humanity*, pages 621–653. Globethics Publications.
[15] Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., and Grossman, T. (2023). 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*, pages 1–23. arXiv:2302.07427 [cs].
[16] Khmelevsky, Y., Hains, G., and Li, C. (2012). Automatic code generation within student's software engineering projects. In *Proceedings of the Seventeenth Western Canadian Conference on Computing Education*, pages 29–33, Vancouver British Columbia Canada. ACM.
[17] Kolides, A., Nawaz, A., Rathor, A., Beeman, D., Hashmi, M., Fatima, S., Berdik, D., Al-Ayyoub, M., and Jararweh, Y. (2023). Artificial intelligence foundation and pre-trained models: Fundamentals, applications, opportunities, and social impacts. *Simulation Modelling Practice and Theory*, 126:102754.
[18] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., Hubert, T., Choy, P., d'Autume, C. d. M., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Robson, E. S., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. (2022). Competition-Level Code Generation with AlphaCode. *Science*, 378(6624):1092–1097. arXiv:2203.07814 [cs].
[19] Manna, Z. and Waldinger, R. J. (1971). Toward automatic program synthesis. *Communications of the ACM*, 14(3):151–165.
[20] Manning, S., Mishkin, P., Hadfield, G., and Eisner, E. (2022). A Research Agenda for Assessing the Economic Impacts of Code Generation Models.
[21] Marwan, S., Jay Williams, J., and Price, T. (2019). 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*, pages 61–70, Toronto ON Canada. ACM.
[22] Mosterman, P. (2006). Automatic Code Generation: Facilitating New Teaching Opportunities in Engineering Education. In *Proceedings. Frontiers in Education. 36th Annual Conference*, pages 1–6, San Diego, CA, USA. IEEE.
[23] OpenAI (2022a). Codex.
[24] OpenAI (2022b). Introducing ChatGPT.
[25] OpenAI (2023). GPT-4 is OpenAI's most advanced system, producing safer and more useful responses.
[26] Ouyang, F., Zheng, L., and Jiao, P. (2022). Artificial intelligence in online higher education: A systematic review of empirical research from 2011 to 2020. *Education and Information Technologies*, 27(6):7893–7925.
[27] Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., and Santos, E. A. (2023). "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. arXiv:2304.02491 [cs].
[28] Ross, S. I., Martinez, F., Houde, S., Muller, M., and Weisz, J. D. (2023). The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 491–514, Sydney NSW Australia. ACM.
[29] Sarsa, S., Denny, P., Hellas, A., and Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research V.1*, pages 27–43, Lugano and Virtual Event Switzerland. ACM.
[30] Surameery, N. M. S. and Shakor, M. Y. (2023). Use Chat GPT to Solve Programming Bugs. *International Journal of Information technology and Computer Engineering*, (31):17–22.
[31] Tajik, E. and Tajik, F. (2023). A comprehensive Examination of the potential application of Chat GPT in Higher Education Institutions. preprint.
[32] Thompson, E., Whalley, J., Lister, R., and Simon, B. (2023). Code Classification as a Learning and Assessment Exercise for Novice Programmers.
[33] Vaithilingam, P., Zhang, T., and Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, New Orleans LA USA. ACM.
[34] Wermelinger, M. (2023). Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 172–178, Toronto ON Canada. ACM.