How and Why Novice Students Use LLM-Based Tools to Solve CS1 Coding Tasks

Christian Garmann Sørli[0009−0007−0544−1349] and
Trond Aalberg[0000−0001−5593−0860]

Norwegian University of Science and Technology, Trondheim, Norway
trond.aalberg@ntnu.no
chrisgso@ntnu.no

Abstract. As Large Language Models (LLMs) gain popularity in programming education, understanding how novice programmers use these tools and their motivations is essential. Through 16 sessions with participant observation and a questionnaire, we investigate factors driving novice students to use AI tools in CS1 coding assignments. Furthermore, we examine *how* the students engage with AI tools. A thematic analysis examines (a) what learners are asking from AI tools, (b) prompt characteristics, and (c) how learners use AI-generated responses in terms of learning, manual modifications, and verification. Our analysis identifies key motivations that drive novices to use AI tools, including time-saving and task management, ease of access, aided learning, and cognitive miserliness of the user. Furthermore, our analysis reveals diverse engagement patterns, including a novel "AI Conceptual-Hybrid" approach, where students use AI to clarify concepts and task requirements but write code independently. Additionally, we offer insights on how inexperienced learners utilize AI, emphasizing self-regulation, indications of over-reliance, and possible opportunities for improving AI-assisted learning in computing education.

Keywords: Generative AI · Programming Education

1 Introduction

Artificial intelligence (AI) tools, particularly large language models (LLMs) such as OpenAI's ChatGPT, are increasingly being explored for their potential to support education, including programming and computer science. These tools offer functionalities such as generating code, providing explanations, and assisting with debugging. Although their application in education is still in its early stages, an increasing body of research highlights a positive correlation between their use and enhanced student learning outcomes [12, 1, 19]. Data from the Norwegian national student survey 2024, Studiebarometeret, reveal that 81% of Norwegian students have utilized AI in their academic work [7]. The survey further highlighted that technical and natural sciences students are among the most frequent users, with 78% reporting occasional or frequent use of AI tools.

Similarly, a study conducted by Saari et al. (2023) reported comparable findings, indicating that nearly two-thirds of students have already employed AI tools to support their programming education [18].

However, the extent to which LLMs can effectively enhance programming education is still an open question. Their role in addressing challenges faced by novice programmers, such as conceptual misunderstandings, debugging difficulties, and the cognitive demands of learning to code, requires further investigation. Additionally, there are concerns about over-reliance on AI-generated solutions, the potential for misuse, and the accuracy of the responses provided by these tools. The overarching goal of this study is to gain insights into how AI tools, powered by large language models, influence the learning experience in introductory programming education. The study seeks to understand the interplay between student engagement with AI tools and their learning strategies, aiming to inform the effective integration of such tools in computer science education. In this study, we use the term positive intention to describe a learner's perceived motivation or willingness to use AI tools. The project has been guided by the following research questions:

RQ 1 *What factors generate positive intention to use AI tools in assignments?*
RQ 2 *How do novices engage with the AI tools in assignments?*
RQ 3 *What coding approaches do novices use when they have access to AI tools?*

A thematic analysis is performed on data collected through an exploratory case study conducted within an introductory programming course at The Norwegian University of Science and Technology (NTNU). Data collection involved participant observations across 16 sessions, complemented by a post-study questionnaire. The research design emulates the typical conditions under which students complete their CS1 assignments, thereby mitigating potential biases associated with the observational context. Participation in the study was voluntary. The students were instructed to approach their coding assignments as they would independently in exercise sessions. This included their optional use of AI-based tools, reflecting authentic engagement with the tasks.

2 Background

Large Language Models are a subset of AI technologies capable of generating diverse and contextually relevant text, including programming code in any language. The number and quality of tools and services released annually have shown a consistent upward trend [14]. With recent advances, such as agentic coding tools and the ability to integrate LLMs with third party tools and services with the use of the Model Context Protocol, the way we create code, and consequently how we develop coding competence, is about to change drastically.

The effectiveness of LLMs can be influenced by a range of factors, including the model's hyperparameters, prompt design, and the diversity of training data [2]. Among these, prompt engineering has emerged as a critical approach for the utility and accuracy of LLMs, particularly in educational contexts. In recent

work, Denny et al. emphasize that the ability to craft effective prompts capable of generating accurate solutions has become an essential skill for students [5]. Key components of effective prompt design [2] includes:

- Providing comprehensive instructions within the prompt helps LLMs understand the task requirements and generate more relevant responses.
- Clarity and precision in prompts are essential to avoid ambiguity and ensure the LLM delivers accurate outputs.
- Role-Prompting involves assigning a specific role to the LLM to align its responses with the user's expectations.
- Because LLMs are non-deterministic, it is often beneficial to ask AI several times, and then choose the best of multiple responses.

LLMs hold significant potential to enrich higher education by fostering autonomous, collaborative, and interactive learning environments [16]. In computing education, LLMs offer unique opportunities to support novice learners in several ways:

- Personalized feedback: LLMs provide tailored explanations and suggestions, adapting to the individual needs of learners [21].
- Guidance without direct solutions: If used to offer incremental guidance instead of full solutions, these tools can encourage critical thinking and problem-solving skills [10].
- Practical applications: LLMs assist students in generating code from natural language specifications, explaining code functionality, generating comments and documentation, creating and debugging tests, and providing feedback on written code [9].

Recent studies highlight the transformative impact of AI tools in programming education. Yilmaz and Yilmaz [21] found that students who used ChatGPT demonstrated significant improvements in computational thinking, programming mastery, and motivation compared to a control group. The study suggested that AI tools help learners save time and mental resources otherwise spent on repetitive tasks like debugging, enabling them to focus on higher-level problem-solving and algorithmic thinking. Kazemitabaar et al. [8] explored how novices utilize AI tools to write code and identified four distinct strategies: AI Single Prompt, where students rely on AI to produce full solutions in a single request; AI Step-by-Step, in which students broke the problem up into smaller components and utilized AI to create each component; Hybrid, which involves students writing some of the code themselves and using AI tools to produce other parts; and Manual coding, where learners wrote the code themselves. LLM-based tools designed for an educational context is another topic for research e.g. CodeHelp [10], which was designed to provide guidance without offering direct answers and findings indicate that such tools promote independent problem-solving by providing constant, anxiety-reducing support to learners, enhancing the overall quality of assistance available. These findings align with prior research on intelligent tutoring systems, which suggest that personalized feedback and immediate support

contribute to improved learning outcomes compared to traditional methods like lectures or static textbooks [13]. The advantages of integrating LLMs in education may also extend beyond individualized feedback and assistance. Enhanced information accessibility and interactive engagement foster a more inclusive and motivating learning environment [1].

Despite the potential benefits AI tools can offer computing education, there are also challenges. Prejudices, biases, factual inaccuracies ("hallucinations"), and other deceptive responses are known hazards that might mislead and confuse students [10]. Other prevalent worries include the possibility of cheating [9] and excessive dependence on AI technologies [10]. Students might utilize AI tools to develop solutions to assignments without truly grasping the underlying principles, or in the worst case not even realizing that the resulting code is incorrect [9]. This is especially problematic when assignments have precise descriptions that AI can readily complete [9]. Students can also use these tools to produce code with simpler constructs, which appears less AI-generated [9], making it more challenging to detect that the code was not authored by the student.

3 Method

To explore how novices use AI tools, we have analyzed data collected from a study in which novice programming students solved programming tasks in the CS1 course TDT4109 "Information Technology, Introduction" at the Norwegian University of Science and Technology. The course gives an introduction to basic procedure-oriented programming in Python and includes programming assignments in Jupyter notebooks. The study consisted of sixteen 50-minute in person sessions with 16 individual students (4 females and 12 males), held over 11 days in October 2024. During the observations, each student spent 40 minutes working on a programming assignment of their choice, specifically focused on tasks they had not previously attempted. The session concluded with a 10-minute questionnaire that captured students' general experiences and perspectives on the use of AI tools in programming education, including perceived benefits and limitations from the students' viewpoints. The sessions were conducted independently of the CS1 course TDT4109, and participation did not provide any academic advantages or influence students' course evaluations or grades. Each participant received a 250 NOK gift card as compensation. All of the students were first-year students, and none of them reported having had any university programming courses before this CS1 course. In terms of language, all participants were Norwegian speakers.

The study was designed as an exploratory case study conducted in a real-world learning environment, closely mirroring the typical conditions under which students complete their assignments. Data were collected in a standard classroom that participants regularly used for studying, thereby enhancing the authenticity of the observations. To create a familiar learning situation, student assistants acted as observers, seated beside participants as they worked. This arrangement replicated the common scenario of a student working with an assistant

nearby and helped reduce potential observer effects. The observers followed a semi-structured observation guide containing questions aimed at gaining deeper insights into the students' thoughts, motivations, and feelings as they progressed. Any instances of AI tool usage were documented, noting the context, and students were asked to think aloud and explain their reasoning for each usage. Observed tasks were of similar complexity across sessions, though the specific topic varied based on each student's assignment choice, provided it was a topic they had not previously tackled. The assignments spanned topics such as "Lists and Word Processing," "Dictionaries, Sets, and File I/O," and "Recursion and Compound Programs". The questionnaire was based on the concepts from the Technology Acceptance Model [4] adapted for the context of this study.

We utilized non-parametric statistical methods for all the Likert scale questionnaire data, considering the ordinal nature of Likert scale responses and the limited sample size. Specifically, we employed the One-Sample Wilcoxon signed-rank test to test if the median response is significantly different from the neutral midpoint, i.e. "Neutral" on our Likert scale. For the rest of the data collected, we employed a thematic analysis to identify factors influencing novice students' use of AI tools and to discern patterns in how they engage with these tools while completing CS1 coding assignments. This approach provided a systematic method for interpreting qualitative data from both observational notes and open-ended questionnaire responses, revealing recurring themes related to students' motivations, challenges, and interactions when using AI tools.

To address RQ1, we built upon the pivotal factors influencing positive intentions to use AI tools identified by Niloy et al. [15]: *Time Saving and Task Management*, *Ease of Access*, *Aided Learning, Inseparability of Content*, *Technical Knowledge of the Program*, and *Cognitive Miserliness of the User*. We developed a coding framework using these as predefined themes, ensuring consistency by precisely defining each factor's scope in alignment with Niloy et al.'s analysis. This structured approach enabled us to systematically investigate students' motivations and intentions for using AI tools, providing a robust basis for thematic analysis. For RQ2 and RQ3, we utilized the codebook developed by Kazemitabaar et al. [3, 8]. This provided a foundational framework for categorizing and interpreting data related to how students engage with AI tools during coding tasks. However, since our study incorporates new data collection methods - specifically introspective and retrospective think-aloud protocols to delve deeper into students' thought processes - we iteratively updated the codes and themes. When findings emerged that did not align with the existing categories in the codebook, we refined and expanded the framework to ensure that all relevant insights were captured.

The data analysis followed established thematic analysis procedures: *Data Familiarization* – reviewing all observational notes and questionnaire responses; *Coding* – applying deductive codes from Niloy et al. [15] for RQ1 and Kazemitabaar et al. [8] for RQ2-3, with minimal modifications for new findings; *Theme Identification* – organizing codes into broader patterns of AI tool use; *Theme Review* – ensuring coherence and representativeness across participants; *Defining Themes*

– creating clear descriptions aligned with research questions; and *Interpretation* – analyzing relationships between themes and students' AI tool usage patterns. This analysis provided a nuanced understanding of the participants' attitudes, motivations, and usage patterns with AI, which we discuss further in the results section.

4 Results and Discussion

From the 16 observations, only 1 was completed without using AI tools, while the remaining involved using AI tools at least once. A majority of students (87.6%) disagreed or strongly disagreed that using AI tools in programming education is wrong, with a One-Sample Wilcoxon signed-rank test confirming a statistically significant lean toward ethical acceptance ($p<0.001$). Regarding usage frequency, six students (37.5%) consulted an AI tool multiple times per task, six students (37.5%) used it in some but not all tasks and three students (18.8%) relied on AI assistance for only a few tasks. The observed AI tool usage corresponds well with the self-reported usage from the questionnaire. All 16 students reported using AI tools in programming occasionally, with two students (12.5%) selecting 'rarely', four (25%) selecting 'sometimes', eight (50%) selecting 'often', and two (12.5%) reporting 'always' using such tools. Our findings align with this study reported in [6], as students in our research not only frequently utilize AI tools in programming tasks but also express strong confidence in the benefits of AI tools. The results show that a significant majority (87.6%) agreed or strongly agreed that AI tools have positively influenced their learning and understanding of programming concepts (see figure 1). This sentiment was supported by a One-Sample Wilcoxon signed-rank test, which confirmed a statistically significant lean toward agreement ($p = 0.001$). Additionally, 75% agreed or strongly agreed that they expect their trust in AI tools to increase over time ($p = 0.003$). In terms of practical application, 81.3% ($p = 0.004$) reported that using AI tools has made them better at solving programming problems independently, without relying on help from peers, teaching assistants, or academic staff. Similarly, 87.6% ($p = 0.001$) agreed or strongly agreed that AI tools have enabled them to solve a greater number of programming problems overall.

All the 15 users of AI tools used chat-based versions, with only one of them also utilizing auto-complete functionality, specifically through GitHub Copilot, to directly generate code within their editor throughout every task. For the chat-based tools, AI-usage involves writing a prompt and pressing the send button. Among the AI tools, ChatGPT was the most widely used, with 12 students (80%) utilizing it, while GitHub Copilot, Microsoft Copilot, and Aria Browser AI from Opera were each used by one student.

We define the problem-solving process as a sequence of actions until the completion of a task showing the correct output. These actions include manually editing code, AI tool usage, searching in search engines, consulting other sources, and executing the code. For clarity, we denote participants as Pn in the following analysis.

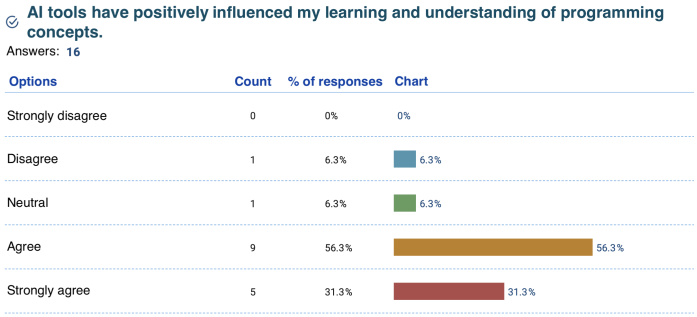

Fig. 1. Questionnaire responses for: *'AI tools have positively influenced my learning and understanding of programming concepts.'*

4.1 RQ1: Factors generating positive intention to use AI tools

According to the Technology Acceptance Model, users' perceptions of a technology's usefulness and ease of use significantly influence their intentions to adopt the technology [4]. As already presented, students expressed strong perceived usefulness of AI tools. Additionally, students reported that the tools were very easy to use. The majority (81.3%) agreed or strongly agreed that they adapted to using AI tools for programming without major challenges. A One-Sample Wilcoxon signed-rank test confirmed a significant tendency toward agreement ($p = 0.001$). Similarly, a majority (75.1%, $p = 0.003$) agreed or strongly agreed that they felt comfortable using AI tools within a short amount of time. In general, we found that that students extensively used AI tools, perceiving them as highly useful and easy to use.

Focusing on the specific context of solving CS1 coding assignments, this study identified 48 instances where students articulated factors that generated a positive intention to use AI tools during the problem-solving process. This study reveals that novice programmers are motivated to use AI tools for reasons including time-saving and task management (37.5%, $n = 18$), ease of access (6.3%, $n = 3$)), aided learning (35.4%, $n = 17$), and cognitive miserliness of the user (20.8%, $n = 10$). Our findings align well with the conclusions of Niloy et al. [15], also in terms of no evidence that Technical Knowledge of the Program significantly impacts students' intention to use AI tools.

The emphasis on time savings and task management as the predominant motivator corroborates previous research, including assertions made by Rutner and Scott [17], Yang et al. [20], Lu et al. [11], and Niloy et al. [15]. These findings underscore the value of efficiency as a key driver in adopting AI tools among novice programmers. Ultimately, this study contributes by exploring what motivates students to use AI tools in the specific context of introductory programming education.

4.2 RQ2: How do novices engage with the AI tools?

What are users asking AI tools for? In terms of engagement, we first analyzed what users are asking the AI tools and found that learners primarily utilized AI tools to address points of confusion, such as resolving syntax errors, clarifying conceptual misunderstandings, and creating, completing, or refining code solutions. This study identified 40 instances of learner queries to AI tools, with 13 cases (32.5%) involving follow-up queries. The queries were categorized into four primary themes:

Debugging Assistance (40%, $n = 16$): The most common type of query was related to debugging, ie. the prior situation was some sort of bug. Participants often copied their code or error messages into the AI tool and sought clarification. For instance, P3 repeatedly input their code, asking, *"What's wrong? Short answer."* Similarly, P6 and P13 copied error messages directly and queried, *"What does this error mean?"* or *"What does the error message say?"* In contrast, P1 simply pasted their code into the AI tool without providing additional instructions, leaving the tool to infer the issue.

Explanations (30%, n=12): Learners sought conceptual understanding (n=9), code walkthroughs (n=1), best practices (n=1), and practice examples (n=1). This usage pattern, not previously documented in Kazemitabaar et al.'s codebook [3], aligns with findings that 80% of technology students use AI for explanations [7]. Notably, P11 requested *"examples to practice on to learn the concepts thoroughly,"*, P3 asked *"in which cases does the function return nothing?"* while P11 asked *"explain this with simpler words, I do not understand what they are asking for"*. All demonstrating an interest in using AI tools as a learning resource for self-guided practice.

Direct Solutions (22.5%, n=9): Queries in this category focused on obtaining complete solutions ($n = 6$) or solutions to sub-problems ($n = 3$). For example, P2 pasted task descriptions directly into the AI tool, while P14 uploaded screenshots of their tasks. P3 used follow-up queries to discuss subproblems and eventually requested, *"Show with code based on what I sent."* Meanwhile, P15 leveraged GitHub Copilot's auto-complete functionality to generate several sections of the solution incrementally.

Syntax and Semantics (7.5%, n=3): A smaller subset of queries focused on understanding the syntax and semantics of specific constructs. For instance, P4 asked, *"Can the function 'startswith()' in Python take more than one value?"*. Similarly, P8 sought guidance on Python basics with the queries *"How to add a value to a list in Python?"* and *"How to retrieve values from a dictionary in Python?"*.

These findings highlight the varied ways learners interact with AI tools to support their programming tasks. Debugging assistance emerged as the most prevalent use case, reflecting the utility of AI tools in addressing immediate obstacles. Additionally, learners demonstrated a significant reliance on AI tools for conceptual understanding, direct solutions, and clarifying syntactical details, underscoring the multifaceted role of AI tools in novice programming education.

How are learners asking AI tools? In terms of *how* learners are asking the AI tools we found that their prompts exhibited diverse crafting patterns, varying in specificity and intent. The contents of the textual prompts generally included one or more of the following components: copied student code, copied error messages, and textual queries. The majority (51%) of prompts, including both initial and follow-up, were crafted as textual queries alone. Additionally, 15% contained both code and a textual query, 9% included both code and an error message, 6% combined an error message with a textual query, and 12% consisted solely of copied code with no accompanying query. Examples of textual-only prompts include P6's query, *"explain how to write a general recursive function in python"* and P3's follow-up question, *"how do I get it to check the whole string? Answer short, without code"*. Prompts combining code and textual queries included P14's *"<code>How to add content to a shopping list"* and P3's follow-up prompt, *"why not this? <a bit of their code>."* Prompts that combined code and error messages were direct copies of both elements, while prompts that combined error messages with textual queries were variations of P6's question, *"what does this error mean: <the error message>."*

Some learners demonstrated an ability to shape the AI tool's behavior by seeking personalized and context-specific responses, effectively engaging in role-prompting. For instance, P3 explicitly requested short, non-code answers on multiple occasions, including three queries that specifically emphasized brevity. Similarly, P9 guided the AI to respond in Norwegian, while P11 sought simplified explanations tailored to their level of understanding. These actions align closely with the concept of role-prompting, as described by Chen et al. [2], and by doing so, the students influenced the model's behavior to ensure outputs were more relevant and aligned with their individual needs, better leveraging the AI as a personalized assistant.

Five out of 40 initial queries (12.5%) were classified as vague, with participants (P1, P7, P9) pasting code without explanation, contrasting with effective prompt engineering principles that emphasize clarity [2]. Vague prompts were typically followed up by more detailed queries. P1 followed up by combining code and error messages, while P9 refined the query incrementally. With 32.5% of all prompts involving follow-ups, this pattern suggests that learners adopt trial-and-error approaches, iteratively improving their prompts to align with best practices like "being clear and precise" and "trying several times" [2].

Overall, the findings highlight considerable variability in students' prompt crafting skills, which had a direct impact on the relevance and quality of AI-generated outputs. This underscores the critical need to equip learners with effective strategies for designing prompts to maximize their interactions with AI tools, thereby enhancing both learning outcomes and problem-solving capabilities. The importance of this need is reinforced by the students' own expectations, as a majority (68.8%; $p = 0.012$) agreed or strongly agreed that programming education should incorporate training in the effective use of AI tools. Although some effective prompt engineering techniques were observed, many prompts contrasted with these principles, lacking clarity and precision, and no advanced

methodologies were identified. Techniques such as Few-Shot Prompting [2] for illustrating programming patterns and Chain-of-Thought (CoT) [2] prompting for debugging step-by-step could have significantly benefited learners, fostering deeper understanding and better problem-solving strategies.

How are learners utilizing AI responses? Finally, we analyzed how the students *engaged and utilized* AI-generated responses to learn concepts, adapt solutions, and ensure correctness. Participants displayed distinct approaches in utilizing, modifying, and verifying AI-generated code. Analysis revealed 38 unique instances in which participants actively engaged with and utilized AI-generated responses. While participants received more than 38 responses overall, some were excluded from this analysis because they were quickly discarded by the students, likely due to perceived irrelevance or inapplicability. Interestingly, most participants (81.3%) agreed or strongly agreed that they generally possess a good ability to assess the usefulness of AI-generated code, with a One-Sample Wilcoxon signed-rank test confirming significant agreement ($p = 0.002$). However, since this study does not primarily focus on the relevance or accuracy of AI responses or participants' ability to identify irrelevance, only instances demonstrating active engagement—such as learning new concepts or adapting AI-generated solutions—were included in the analysis. These instances were grouped into three primary themes:

Made no changes to code (39.5%, $n = 15$): In 12 instances, participants focused on understanding the underlying concepts provided by AI without making any modifications to the code. Additionally, in 3 instances, participants engaged with AI through follow-up prompts to seek clarification or gain deeper insights into the concepts, still refraining from altering the code. This pattern suggests a focus on learning over immediate application.

Direct utilization of AI-generated code (36.8%, $n = 14$): where participants incorporated the AI-generated code or portions of it directly into their solutions and verified its correctness through execution and runtime behavior, relying on this as their primary measure of validity. This integration was achieved in two ways: by manually typing out the code (10 instances) or by copying and pasting it directly (4 instances). While some participants took time to carefully read and reflect on the AI-generated code before running it, others prioritized immediate execution. This variation highlights differing levels of engagement with the conceptual aspects of the code before testing its functionality, and this finding is also prevalent in the questionnaire data. For the statement *'I carefully assess the quality of generated code from AI tools'*, the student responses almost spread equally across the different options, except for "Strongly disagree", which no student reported (see figure 2). At the same time, the majority of students (68.8%) agreed or strongly agreed that they generally trust the solutions that AI tools provide. A One-Sample Wilcoxon signed-rank test confirmed that responses significantly leaned toward agreement ($p = 0.012$). While these results might intuitively lead us to assume that some students trust AI-generated code to such an extent

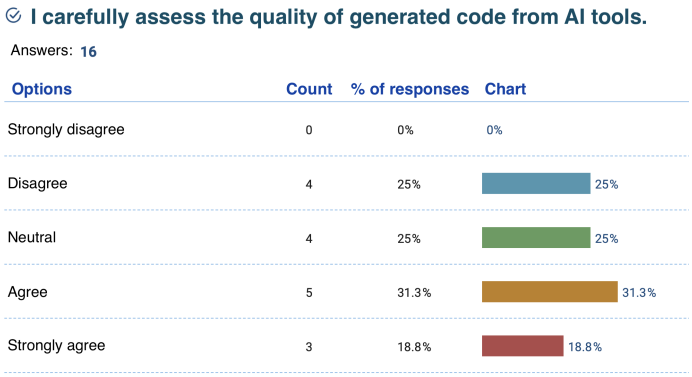

Fig. 2. Questionnaire responses for: *'I carefully assess the quality of generated code from AI tools'*

that they neglect to assess its quality, this study does not support such a claim. The Spearman's rho correlation coefficient was calculated as -0.269 ($p = 0.314$, two-tailed), indicating no significant correlation between the variables. However, given the small sample size ($n = 16$), these findings should be interpreted with caution, as the limited data may reduce the generalizability and statistical power of the results.

Internalization and distinct modifications (23.6%, $n = 9$): In these instances, participants internalized the concepts presented in the AI-generated responses and implemented their own unique solutions. These modifications were distinct from any generated code included in the AI response, reflecting a deeper understanding and the ability to adapt learned concepts creatively.

Importantly, participants did not adhere to a single approach when interacting with AI-generated responses. Many demonstrated adaptive behaviors, alternating between direct utilization and conceptual exploration depending on the task or context. For example, Participant P1 exhibited varying strategies: in some instances meticulously examining AI responses, investing time in understanding and articulating the concepts, in other cases quickly typing out the AI-generated code with minimal examination. This indicates that novices often balance between using AI-generated code directly and using it as a learning aid for conceptual understanding. Such behavior reflects elements of self-regulation, as participants adapted their strategies to align with their learning objectives, while also hinting at potential over-reliance on AI tools when conceptual engagement was de-prioritized in favor of expedient solutions.

4.3 RQ3: Novices coding approaches with access to AI tools

We categorized coding approaches on a task-by-task basis, as students often employed different strategies depending on the specific task. Throughout the

observations, students completed a total of 27 tasks. Notably, 31.3% of students (n=5) employed multiple coding strategies, representing 62.5% of the eight participants who completed more than one task. Our findings align closely with the coding strategies outlined in the study by Kazemitabaar et al. [8], which categorized novice approaches into four distinct strategies when using AI tools to write code: AI Single Prompt (18.5%, $n = 5$ *tasks*), AI Step-by-Step (3.7%, $n = 1$), Hybrid (40.7%, $n = 11$) and Manual coding (3.7%, $n = 1$).

While our data supports these strategies, it also reveals an additional coding approach not previously documented, which we term **"AI Conceptual-Hybrid"** (33.3%, $n = 9$). In this approach, students did not rely on AI to generate any portion of the code itself, therefore these cases do not fit into the Hybrid coding strategy. Instead, they used AI to clarify task requirements, understand underlying concepts, or resolve uncertainties at a conceptual level before independently writing the code. This distinct strategy emphasizes the role of AI as a conceptual aid rather than a direct code-writing tool. For example, participants employing the AI Conceptual-Hybrid strategy often asked AI for explanations of task objectives or sought insights into debugging principles without integrating AI-generated code into their final solutions. This behavior is consistent with survey results, which show that 18.8% of students reported "Always" using AI tools in an educational manner that does not directly provide answers. Additionally, 74.9% indicated using AI tools in this way at least occasionally, though not consistently. These findings highlight an innovative approach for learners to engage with AI tools, enabling them to enhance their problem-solving processes while retaining full ownership of the coding task. This emphasizes the importance of viewing AI tools not merely as code generators but as conceptual facilitators that support learning and comprehension.

5 Conclusion

This study explores the role of AI tools powered by large language models in the context of introductory programming education. The findings illustrate that novice programming students frequently utilize AI tools such as ChatGPT, motivated by factors such as time-saving and task management, ease of access, aided learning, and cognitive miserliness of the user. The majority of learners used AI to overcome immediate programming challenges, primarily asking for debugging assistance, conceptual explanations and direct solutions.

The analysis reveals a spectrum of coding strategies employed by students, ranging from passive reliance on AI tools to active, independent learning. Approximately one-fifth of the tasks completed by students in the study were generated entirely by AI. The most prevalent approach, however, was a hybrid strategy, wherein students authored portions of the code themselves while relying on AI to produce the remaining components. A further variation of this hybrid model also emerged, termed the "AI Conceptual-Hybrid" approach. In this strategy, students utilized AI not for direct code generation, but as a conceptual aid to clarify programming principles and deepen their understanding, while inde-

pendently implementing the code. These findings suggest that AI serves not only as a code-generation tool but also as a pedagogically valuable resource that can support and enhance the learning process.

Despite these opportunities, this study highlights notable challenges, such as the risk of over-reliance on AI and variability in students' abilities to craft effective prompts. Additionally, students expressed a clear expectation for formal instruction on the use of these technologies. These observations point to the potential benefits of integrating structured guidance on AI tool usage into programming curricula, particularly to better support novice learners and address the identified challenges.

A number of limitations were identified in this study that pave the way for future work. As participation in the study was voluntary, the sample may reflect a selection bias toward more motivated or engaged students, which could limit the generalizability of the findings. Future studies could address a larger, more diverse sample across multiple courses or institutions, examining longitudinal usage patterns, and incorporating additional quantitative measures to complement the qualitative findings. These avenues represent exciting opportunities to extend our understanding of how novice programmers engage with AI tools.

References

1. Anissa M. Bettayeb, Manar Abu Talib, Al Zahraa Sobhe Altayasinah, and Fatima Dakalbab. Exploring the impact of ChatGPT: conversational AI in education. *Frontiers in Education*, 9, 2024.
2. Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review, 10 2023.
3. Codex learning analysis codebook. https://tinyurl.com/codex-analysis-codebook. Accessed: 2024-11-19.
4. Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly,*, 13:319–340, September 1989.
5. Paul Denny, Viraj Kumar, and Nasser Giacaman. Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM technical symposium on computer science education V. 1*, pages 1136–1142, 2023.
6. Aashish Ghimire and John Edwards. Generative AI adoption in classroom in context of Technology Acceptance Model (TAM) and the Innovation Diffusion Theory (IDT), 2024.
7. Gustavo Guajardo Ingebjørg Flaata Bjaaland, Mathias Meier Nilsen and Magnus Strand Hauge. Studiebarometeret 2024 – hovedtendenser. Technical report, NOKUT – Nasjonalt organ for kvalitet i utdanningen, 2025.
8. M. Kazemitabaar, X. Hou, A. Henley, B. J. Ericson, D. Weintrop, and T. Grossman. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli, Finland, 2024.
9. S. Lau and P. Guo. From 'ban it till we understand it' to 'resistance is futile': How university programming instructors plan to adapt as more students use AI

code generation and explanation tools such as ChatGPT and GitHub Copilot. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, pages 106–121, Chicago, IL, USA, 2023.
10. M. Liffiton, B. E. Sheese, J. Savelka, and P. Denny. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli, Finland, 2024.
11. Owen H. T. Lu, Anna Y. Q. Huang, Danny C. L. Tsai, and Stephen J. H. Yang. Expert-authored and machine-generated short-answer questions for assessing students learning performance. *Educational Technology & Society*, 24(3), 2021.
12. Wenhan Lyu, Yimeng Wang, Tingting Chung, Yifan Sun, and Yixuan Zhang. Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. In *Proceedings of the eleventh ACM conference on learning@ scale*, pages 63–74, 2024.
13. Wenting Ma, Olusola O. Adesope, John C. Nesbit, and Qing Liu. Intelligent tutoring systems and learning outcomes: A meta-analysis. *Journal of Educational Psychology*, 106:901–918, 2014.
14. Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models, 2024.
15. Ahnaf Chowdhury Niloy, Md Ashraful Bari, Jakia Sultana, Rup Chowdhury, Fareha Meem Raisa, Afsana Islam, Saadman Mahmud, Iffat Jahan, Moumita Sarkar, Salma Akter, Nurunnahar Nishat, Muslima Afroz, Amit Sen, Tasnem Islam, Mehedi Hasan Tareq, and Md Amjad Hossen. Why do students use ChatGPT? answering through a triangulation approach. *Computers and Education: Artificial Intelligence*, 6:100208, 2024.
16. Iris Cristina Peláez-Sánchez, Davis Velarde-Camaqui, and Leonardo David Glasserman-Morales. The impact of large language models on higher education: exploring the connection between AI and education 4.0. *Frontiers in Education*, 9, 2024.
17. S. Rutner and R. Scott. Use of artificial intelligence to grade student discussion boards: An exploratory study. *Information Systems Electronic Journal*, 20(4):4–18, 2022.
18. Mika Saari, Petri Rantanen, Mikko Nurminen, Terhi Kilamo, Kari Systä, and Pekka Abrahamsson. Survey of AI tool usage in programming course: Early observations. In Philippe Kruchten and Peggy Gregory, editors, *Agile Processes in Software Engineering and Extreme Programming – Workshops*, pages 182–191, Cham, 2024. Springer Nature Switzerland.
19. Rongxuan Wei, Kangkang Li, and Jiaming Lan. Improving collaborative learning performance based on llm virtual assistant. In *2024 13th International Conference on Educational and Information Technology (ICEIT)*, pages 1–6, 2024.
20. Albert Yang, Irene Chen, Brendan Flanagan, and Hiroaki Ogata. Automatic generation of cloze items for repeated testing to improve reading comprehension. *Educational Technology & Society*, 24:147–158, 06 2021.
21. R. Yilmaz and F. G. Karaoglan Yilmaz. The effect of generative artificial intelligence (ai)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Computers and Education: Artificial Intelligence*, 4:100147, 2023.