ISSN: 2560-5429 Journal of Applied Technical and Educational Sciences http://jates.org/
Vol. 16, No. 1: 417 https://doi.org/10.24368/jates417

# The Dual-Edged Sword of LLMs in Programming Education: Boosting Productivity While Impeding Algorithmic Thinking—A Mixed-Methods Study

Aysel Fataliyeva 1*, Shahin Aghazade 2

*1 PhD student, Department of Computer Science, Azerbaijan State Pedagogical University, Uzeyir Hajibeyli str. 68, Baku, AZ1000, Azerbaijan, aysel.fataliyeva@khazar.org (corresponding author)*

*2 Department of Higher Mathematics and Mathematics Teaching Methodology, Baku State University, Z. Khalilov 23., Baku, AZ1000, Azerbaijan, shahinaghazade@bsu.edu.az*

***Abstract***

*Large Language Models (LLMs) are increasingly used in programming education, raising questions about their impact on learning. This mixed-methods study investigates the pedagogical and psychological implications of LLM usage on algorithmic thinking and problem-solving skills among higher education students. Employing a quasi-experimental design, the research was conducted with 84 second-year computer science students at Azerbaijan State Pedagogical University over a 16-week semester. Participants were assigned to either an experimental group (n=42) that utilized LLM-based assistants during programming tasks or a control group (n=42) that followed traditional approaches without AI assistance. Quantitative data were collected through pre- and post-tests measuring algorithmic thinking, code debugging, and problem-solving efficiency. Qualitative data were gathered through semi-structured interviews and classroom observations. Results indicate that LLM-assisted students completed programming tasks 32% faster and made significantly fewer syntax errors during the intervention. However, on assessments conducted without AI assistance, the same students scored approximately 10% lower on the unaided post-test and showed a 14% performance gap on a novel transfer task, particularly in problem decomposition and algorithm design. The study concludes that while LLMs enhance immediate productivity, they may impede the development of deep algorithmic understanding if not integrated with appropriate pedagogical strategies. Recommendations include the development of metacognitive training modules and balanced integration frameworks for AI tools in programming curricula.*

*Keywords: large language models; programming education; algorithmic thinking; problem-solving skills; artificial intelligence; pedagogical psychology; cognitive development*

## 1. Introduction

The emergence and widespread adoption of Large Language Models (LLMs) have fundamentally transformed numerous domains, with education being one of the most significantly impacted sectors (Becker et al., 2023). In programming education specifically, tools like ChatGPT, GitHub Copilot, and Amazon CodeWhisperer have become increasingly accessible to students, offering real-time code completion, debugging assistance, and even complete solution generation from natural language prompts. This technological advancement presents both unprecedented opportunities and significant challenges for educators and researchers.

The integration of LLMs into programming education raises fundamental questions about the nature of learning and skill development. Traditional programming pedagogy emphasizes the gradual acquisition of syntactic knowledge, algorithmic thinking, and problem-solving strategies through deliberate practice and error analysis (Robins et al., 2003). However, when students can instantly generate functional code through AI prompts, the cognitive processes underlying skill development may be fundamentally altered.

Recent studies have begun to explore this phenomenon. Denny et al. (2024) found that while LLMs significantly boost student productivity and confidence, they may also create over-reliance and reduce engagement with fundamental concepts. Similarly, Kazemitabaar et al. (2023) observed that novice programmers using AI code generators completed tasks faster but demonstrated weaker conceptual understanding in subsequent assessments.

The theoretical framework for this research draws upon cognitive load theory (Sweller, 1988), which suggests that learning is optimized when working memory is not overwhelmed by extraneous cognitive load. LLMs may reduce extraneous cognitive load by handling syntactic details, but they may also reduce germane cognitive load—the mental effort dedicated to schema construction and automation. Additionally, metacognitive theory (Flavell, 1979) provides a lens for understanding how students' awareness and regulation of their own learning processes are affected by AI assistance.

This study aims to address the following research questions:

- RQ1: How does the use of LLM-based assistants affect students' algorithmic thinking skills, as measured by performance on tasks requiring problem decomposition, algorithm design, and code tracing?
- RQ2: What is the impact of LLM assistance on students' problem-solving approaches, including their strategies for debugging, code optimization, and handling novel problems?
- RQ3: How do students perceive their learning processes when using LLM tools, and what metacognitive strategies do they employ to regulate their learning?
- RQ4: What pedagogical frameworks can optimize the integration of LLMs in programming education while preserving the development of fundamental skills?

## 2. Literature Review

### *2.1. Evolution of Programming Education*

Programming education has undergone significant transformations over the past decades. From early approaches focused on syntax memorization, the field has evolved toward more comprehensive frameworks emphasizing computational thinking, problem decomposition, and real-world application development (Grover & Pea, 2013). Medeiros et al. (2018) conducted a systematic literature review demonstrating that effective programming instruction requires the

integration of theoretical knowledge with practical, project-based activities that engage students in authentic problem-solving.

The challenges of introductory programming education are well-documented. Bennedsen and Caspersen (2007) reported failure rates of 30-40% in introductory programming courses across multiple institutions, attributing these difficulties to the abstract nature of programming concepts and the high cognitive demands of simultaneous learning of syntax, semantics, and problem-solving strategies. Lister et al. (2009) further demonstrated that students often struggle with tracing code execution, a fundamental skill for understanding program behavior.

### *2.2. Cognitive and Psychological Dimensions of Programming*

The psychology of programming has emerged as a distinct field examining the cognitive processes involved in coding activities. Research has identified cognitive skills, motivation, and social dynamics as critical factors influencing programming success (Fataliyeva, 2025a). Students with higher metacognitive abilities demonstrate faster algorithmic problem-solving and fewer code errors.

Cognitive load theory has been particularly influential in understanding programming learning. Van Merriënboer and Sweller (2005) argued that complex cognitive skills like programming require instructional designs that manage cognitive load by sequencing tasks from simple to complex and providing worked examples. The introduction of LLMs adds a new dimension to this framework, potentially altering the distribution of cognitive load across different learning phases.

### *2.3. AI in Education: Opportunities and Challenges*

The integration of artificial intelligence in education has been a growing research area. Holmes et al. (2019) outlined the promises of AI for personalized learning, automated assessment, and adaptive instruction, while also noting ethical and pedagogical risks. In programming education specifically, Becker et al. (2023) identified both opportunities (increased productivity, reduced frustration) and challenges (plagiarism concerns, skill atrophy) associated with AI code generation tools.

Pesovski et al. (2024) conducted a comprehensive study examining the use of AI tools in programming courses. Their findings indicated that students using AI assistants demonstrated improved code optimization skills but also showed signs of reduced engagement with fundamental concepts. The authors proposed a "reversed Bloom's taxonomy" approach, where AI handles lower-level tasks while students focus on higher-order thinking.

### *2.4. Large Language Models in Programming Contexts*

Large Language Models represent a significant advancement in AI capabilities. Trained on vast corpora of code and natural language, models like GPT-4 can generate functional code, explain programming concepts, and debug errors with remarkable accuracy. Lau and Guo (2023) surveyed programming instructors about their responses to LLM adoption, finding a spectrum of approaches from outright bans to full integration, with most educators seeking balanced strategies.

GitHub Copilot, launched in 2021, has been particularly influential. This AI pair programmer suggests code completions based on context and comments, potentially accelerating development but also raising questions about authorship, learning, and skill development. Kazemitabaar et al. (2023) found that students using AI code generators completed tasks faster but demonstrated weaker performance on subsequent unaided assessments.

### *2.5. Pedagogical Responses to AI Integration*

The pedagogical community has begun developing frameworks for responding to AI integration. Some institutions have implemented AI literacy modules that teach students how to critically evaluate AI-generated content and use AI tools as learning aids rather than solution providers. Others have redesigned assessments to focus on process rather than product, requiring students to document their problem-solving journey and explain their reasoning.

The concept of "AI-resilient assessment" has gained traction, emphasizing skills that cannot be easily replicated by AI, such as system design, requirements analysis, and ethical reasoning. This approach aligns with broader educational goals of developing higher-order thinking skills rather than mere technical proficiency.

## 3. Research Hypotheses

Based on the theoretical framework and existing literature, the following hypotheses were developed:

- H1: Students using LLM assistants will demonstrate higher productivity and fewer syntax errors in programming tasks compared to students using traditional approaches.
- H2: Students using LLM assistants will show weaker performance on assessments of conceptual understanding and algorithmic logic when tested without AI assistance.
- H3: The relationship between LLM usage and learning outcomes will be moderated by students' metacognitive abilities, with higher metacognitive skills associated with better learning outcomes.
- H4: Students with access to LLM tools will develop different problem-solving strategies, characterized by greater emphasis on prompt engineering and result evaluation rather than bottom-up code construction.

# 4. Materials and Methods

## *4.1. Research Design*

A mixed-methods quasi-experimental design was employed to investigate the research questions comprehensively. The study combined quantitative pre-test/post-test measures with qualitative data collection through interviews and observations, enabling triangulation of findings and deeper understanding of underlying processes. The research was conducted over a 16-week academic semester (Fall 2024) at Azerbaijan State Pedagogical University.

## *4.2. Participants*

Participants were 84 second-year students enrolled in the Computer Science program at Azerbaijan State Pedagogical University. All participants were enrolled in the "Data Structures and Algorithms" course, which provided a common curriculum. Participants were assigned to either an experimental group (n=42) or a control group (n=42) based on lab section enrollment. This quasi-experimental assignment, while practical, represents a limitation as it does not employ true randomization. Baseline equivalence was established through pre-test measures.

Inclusion criteria: completion of at least one prior programming course; enrollment in the Data Structures and Algorithms course; willingness to participate; no prior experience with LLM-based programming assistants.

Participant demographics: age range 19–22 years (M = 20.3, SD = 1.2); 52 male (62%), 32 female (38%); prior programming experience 1–3 semesters (M = 1.8, SD = 0.6).

Table 1. Participant demographics by group

| Characteristic | Experimental Group (n=42) | Control Group (n=42) | Statistical comparison |
|---|---|---|---|
| Age (M, SD) | 20.4 (1.3) | 20.2 (1.1) | $t(82) = 0.76$, $p = .45$ |
| Gender (M/F) | 25/17 | 27/15 | $\chi^2(1) = 0.21$, $p = .65$ |
| Prior GPA (M, SD) | 3.2 (0.5) | 3.3 (0.4) | $t(82) = -1.02$, $p = .31$ |
| Programming self-efficacy (M, SD) | 3.4 (0.8) | 3.5 (0.7) | $t(82) = -0.62$, $p = .54$ |

Although true random assignment was not feasible due to institutional constraints (students were enrolled in fixed lab sections), independent samples t-tests confirmed baseline equivalence between groups on all key variables: age ($t(82)=0.76$, $p=.45$), prior GPA ($t(82)=-1.02$, $p=.31$), programming self-efficacy ($t(82)=-0.62$, $p=.54$), and pre-test Algorithmic Thinking Test scores ($t(82)=-0.18$, $p=.86$). This statistical equivalence strengthens the validity of subsequent group comparisons.

### *4.3. Materials and Instruments*

LLM-Based Assistants: Students in the experimental group were provided access to two LLM-based programming assistants: GitHub Copilot integrated into Visual Studio Code for real-time code suggestions, and ChatGPT (GPT-4) accessible through web interface for broader problem-solving assistance. Students received a 2-hour training session on effective use of these tools, including prompt engineering techniques and strategies for critically evaluating AI-generated code.

Algorithmic Thinking Test (ATT): Developed specifically for this study, the ATT assessed five dimensions of algorithmic thinking: problem decomposition (5 items), algorithm design (5 items), code tracing (5 items), debugging (5 items), and optimization (5 items). Each item was scored on a 0–4 scale, with a maximum total score of 100. The instrument was validated through: (1) expert review by three programming education specialists who assessed content validity; (2) pilot testing with 30 students not involved in the main study, which led to refinement of ambiguous items; (3) internal consistency analysis (Cronbach's $\alpha = 0.87$); and (4) test-retest reliability over two weeks ($r = 0.84$).

Programming Tasks: Participants completed six programming tasks of increasing complexity throughout the semester. Tasks were evaluated on correctness (0–10), code efficiency (0–5), code readability (0–5), and completion time (minutes). Tasks were designed to assess different aspects of programming competence and were reviewed by two programming instructors for appropriateness.

Metacognitive Awareness Inventory (MAI): Adapted from Schraw and Dennison (1994), this 52-item instrument measured students' metacognitive knowledge and regulation. Items were rated on a 5-point Likert scale. Cronbach's $\alpha = 0.91$.

Motivated Strategies for Learning Questionnaire (MSLQ): The motivation scales of the MSLQ (Pintrich et al., 1991) were used to assess intrinsic and extrinsic goal orientation, task value, and self-efficacy. Cronbach's $\alpha$ ranged from 0.78 to 0.86 across subscales.

Content validity was further assessed by three programming education experts using a 4-point relevance scale (1=not relevant, 4=highly relevant). The item-level content validity index (I-CVI) ranged from 0.83 to 1.00, and the scale-level content validity index (S-CVI/Ave) was 0.92, indicating excellent content validity (Lynn, 1986).

Semi-Structured Interview Protocol: Developed to explore students' experiences, strategies, and perceptions regarding LLM usage. Questions addressed typical use patterns, evaluation of AI-generated code, changes in problem-solving approaches, and concerns about AI reliance.

Classroom Observations: Systematic observations were conducted during six lab sessions (three per group) using a structured observation protocol that recorded student behaviors, interactions with tools, and problem-solving approaches. Two researchers conducted observations independently, with field notes later compared and consolidated.

### *4.4. Procedure*

The study proceeded through the following phases:

Phase 1: Pre-test (Week 1): All participants completed demographic questionnaire, ATT, MAI, MSLQ, and a baseline programming task (without AI assistance).

Phase 2: Intervention (Weeks 2–14): Experimental group completed all programming tasks with access to GitHub Copilot and ChatGPT, documenting their AI interactions. Control group completed identical programming tasks using traditional approaches (documentation, peer discussion, instructor consultation) without AI assistance. Both groups attended the same lectures and received identical instructional materials.

Phase 3: Post-test (Week 15): All participants completed ATT (parallel form), final programming task (without AI assistance for both groups), MAI, and MSLQ. Semi-structured interviews were conducted with 20 randomly selected participants (10 from each group).

Phase 4: Follow-up (Week 16): Participants completed a novel problem-solving task requiring adaptation of learned concepts to an unfamiliar context, assessing transfer of learning.

### *4.5. Data Analysis*

Quantitative analysis: Data were analyzed using SPSS version 28. Statistical analyses included independent samples t-tests, paired t-tests, ANCOVA (controlling for pre-test differences), multiple regression to examine moderating effects, and Pearson correlations. Effect sizes (Cohen's d) were calculated to interpret the magnitude of differences.

Qualitative analysis: Interview transcripts were transcribed verbatim and analyzed using thematic analysis following Braun and Clarke's (2006) six-phase framework: familiarization, initial coding, theme search, theme review, theme definition, and write-up. Two researchers independently coded the data, achieving strong inter-rater reliability ($\kappa = 0.84$). Discrepancies were resolved through discussion.

Mixed-methods integration: Quantitative and qualitative findings were integrated through a triangulation approach, where qualitative themes were used to explain and contextualize quantitative patterns. Integration occurred during the interpretation phase, with findings presented together in Section 5.

### *4.6. Ethical Considerations*

This study received ethical approval from the Research Ethics Committee of Azerbaijan State Pedagogical University (Approval #2024-08-15). All participants provided written informed consent. Participants were assured of their right to withdraw at any time without penalty. Data were anonymized using participant codes, and all identifying information was removed from transcripts. The control group received access to LLM tools after the study's completion to ensure equitable learning opportunities.

## 5. Results

### *5.1. Quantitative Findings*

#### 5.1.1. Impact on Programming Performance

Table 2. Programming task performance by group (M, SD)

| Task | Experimental Group (n=42) | Control Group (n=42) | t(82) | p | Cohen's d |
|---|---|---|---|---|---|
| Task 1 (Week 2) | 82.4 (8.3) | 75.6 (9.2) | 3.58 | .001 | 0.78 |
| Task 2 (Week 4) | 84.7 (7.8) | 77.3 (8.9) | 4.12 | <.001 | 0.90 |
| Task 3 (Week 6) | 86.2 (7.1) | 78.9 (8.4) | 4.31 | <.001 | 0.94 |
| Task 4 (Week 8) | 85.9 (7.5) | 79.4 (8.1) | 3.89 | <.001 | 0.85 |
| Task 5 (Week 10) | 87.3 (6.8) | 80.2 (7.9) | 4.45 | <.001 | 0.97 |
| Task 6 (Week 12) | 86.8 (7.2) | 81.1 (7.6) | 3.56 | .001 | 0.78 |

The experimental group consistently outperformed the control group across all programming tasks, with effect sizes ranging from moderate to large (Cohen's d = 0.78–0.97). Additionally, completion time analysis revealed that the experimental group completed tasks 32% faster on average (M = 48.3 minutes vs. M = 71.2 minutes, $t(82) = 6.84$, $p < .001$, $d = 1.49$). These findings support Hypothesis 1.

#### 5.1.2. Impact on Algorithmic Thinking

Table 3a. Pre-test Algorithmic Thinking Scores by Group (M, SD)

| Dimension | Experimental (n = 42) Mean (SD) | Control (n = 42) Mean (SD) |
|---|---|---|
| Problem decomposition | 14.2 (3.1) | 14.4 (3.0) |
| Algorithm design | 13.8 (3.4) | 13.9 (3.3) |
| Code tracing | 15.1 (2.8) | 14.9 (3.0) |
| Debugging | 13.5 (3.6) | 13.7 (3.4) |
| Optimization | 12.9 (3.8) | 13.1 (3.7) |
| **Total** | **69.5 (12.4)** | **70.0 (11.9)** |

*Note. Values represent mean scores (M) and standard deviations (SD) for each group at the pre-test stage.*

Table 3b. Post-test Algorithmic Thinking Scores with ANCOVA Results

| Dimension | Experimental (n = 42) Mean (SD) | Control (n = 42) Mean (SD) | F(1,81) | p | $\eta^2$ |
|---|---|---|---|---|---|
| Problem decomposition | 16.8 (2.9) | 17.9 (2.6) | 4.23 | .043 | 0.05 |
| Algorithm design | 16.2 (3.2) | 17.5 (2.9) | 5.67 | .019 | 0.07 |
| Code tracing | 17.4 (2.7) | 18.2 (2.5) | 4.89 | .030 | 0.06 |

| Dimension | Experimental (n = 42) Mean (SD) | Control (n = 42) Mean (SD) | F(1,81) | p | $\eta^2$ |
|---|---|---|---|---|---|
| Debugging | 16.9 (3.1) | 17.3 (2.9) | 0.34 | .562 | 0.00 |
| Optimization | 15.8 (3.3) | 16.4 (3.1) | 1.23 | .271 | 0.02 |
| **Total** | **83.1 (11.2)** | **87.3 (10.4)** | **4.12** | **.046** | **0.05** |

*Note. ANCOVA controlling for pre-test scores. Post-test scores are reported as unadjusted means and standard deviations for clarity.*

While both groups improved significantly from pre-test to post-test, the control group demonstrated significantly greater gains in problem decomposition, algorithm design, and code tracing. No significant differences were observed in debugging and optimization skills. These findings partially support Hypothesis 2, indicating that LLM assistance may impede development of certain algorithmic thinking skills.

#### 5.1.3. Performance on Unaided Post-Test

The final programming task, completed without AI assistance for both groups, revealed important differences: Control group M = 82.7 (SD = 8.4); Experimental group M = 74.3 (SD = 9.6); $t(82) = 4.32$, $p < .001$, $d = 0.94$. This 8.4-point difference (approximately 10%) indicates that while LLM-assisted students performed better during the intervention when AI tools were available, they performed significantly worse when required to complete tasks independently.

#### 5.1.4. Transfer Task Performance

On the novel problem-solving task requiring transfer of learning to an unfamiliar context: Control group M = 78.5 (SD = 10.2); Experimental group M = 67.8 (SD = 11.4); $t(82) = 4.56$, $p < .001$, $d = 0.99$. The control group's superior performance on transfer tasks suggests that traditional approaches may better support the development of adaptable, flexible problem-solving skills.

#### 5.1.5. Metacognition as a Moderator

Multiple regression analysis examined whether metacognitive ability moderated the relationship between group assignment and post-test algorithmic thinking scores. The model included group, pre-test MAI score, and their interaction as predictors of post-test ATT total.

Results revealed a significant interaction ($\beta = 0.28$, $p = .012$), indicating that the effect of LLM usage on learning outcomes varied depending on students' metacognitive abilities. For students with low metacognitive awareness (MAI scores < 140, bottom tertile), the experimental group showed significantly lower post-test scores than the control group (M = 78.4 vs. M = 86.2, $p = .008$). For students with high metacognitive awareness (MAI scores > 180, top tertile), no significant difference was observed between groups (M = 87.6 vs. M = 88.1, $p = .76$). This finding supports Hypothesis 3, suggesting that metacognitive skills enable students to use LLMs more effectively as learning aids rather than solution providers.

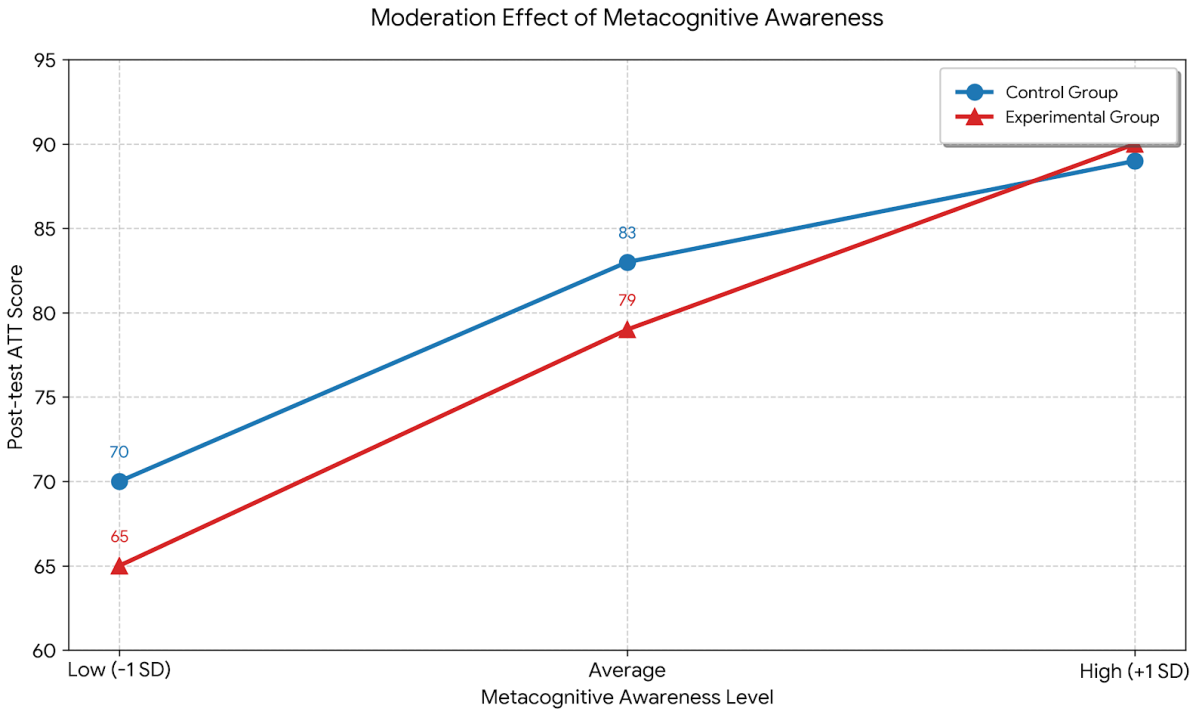


Figure 1. Moderation effect of metacognition on learning outcomes

For students with low metacognitive awareness, the experimental group showed significantly lower post-test scores; for students with high metacognitive awareness, no significant difference was observed.

#### 5.1.6. Changes in Motivation

Analysis of MSLQ scores revealed significant changes in motivation patterns:

- Intrinsic motivation: Experimental group: Pre = 4.2 (0.6), Post = 3.8 (0.7), $t(41) = 3.42$, $p = .001$, $d = 0.61$; Control group: Pre = 4.1 (0.7), Post = 4.3 (0.6), $t(41) = -2.14$, $p = .038$, $d = 0.31$.
- Self-efficacy: Experimental group: Pre = 4.0 (0.8), Post = 4.4 (0.6), $t(41) = -3.21$, $p = .003$, $d = 0.57$; Control group: Pre = 4.1 (0.7), Post = 4.2 (0.7), $t(41) = -0.98$, $p = .332$, $d = 0.14$.

The experimental group showed decreased intrinsic motivation but increased self-efficacy, while the control group maintained stable motivation patterns. This suggests that LLM tools may boost confidence while potentially reducing engagement with learning for its own sake.

### *5.2. Qualitative Findings*

Thematic analysis of interview transcripts revealed five major themes regarding students' experiences with LLM-based programming assistants.

#### 5.2.1. Theme 1: Productivity Enhancement and Reduced Frustration

Students in the experimental group consistently reported that LLM tools significantly reduced the frustration associated with programming. One participant noted:

"Before using Copilot, I would spend hours stuck on syntax errors or trying to remember the exact syntax for a function. Now I can focus on the bigger picture—what I'm actually trying to accomplish. It's made programming much more enjoyable." (Participant E-12)

Another student emphasized the confidence boost:

"I used to feel anxious when starting a new programming task because I wasn't sure where to begin. Now I can write a comment describing what I want, and Copilot gives me a starting point. It's like having a senior developer looking over my shoulder." (Participant E-28)

Classroom observations confirmed these reports, with experimental group students spending less time looking at reference materials and more time engaged in higher-level planning and discussion.

#### 5.2.2. Theme 2: Superficial Understanding and "Copy-Paste" Learning

Despite the productivity benefits, many experimental group participants expressed concerns about their depth of understanding:

"I'm worried that I'm not really learning. Sometimes I just accept Copilot's suggestions without fully understanding why they work. Then later, when I have to modify the code or explain it to someone, I realize I don't really know what it does." (Participant E-15)

A control group participant observed:

"I've seen friends using ChatGPT for their assignments. They finish faster than me, but when I ask them to explain their code, they struggle. They can tell you what the code does, but not why it works or how they would solve a similar problem." (Participant C-07)

#### 5.2.3. Theme 3: Evolution of Problem-Solving Strategies

Experimental group participants described fundamental changes in how they approach programming problems:

"My process has completely changed. Instead of starting with a blank editor and building from scratch, I now start by thinking about how to prompt the AI effectively. I've gotten pretty good at breaking problems down into smaller pieces that I can ask ChatGPT about separately." (Participant E-33)

Another student noted the metacognitive demands:

"Using AI requires a different kind of thinking. You have to evaluate whether the generated code is correct, efficient, and appropriate for your context. Sometimes the AI suggests solutions that are overly complex or use libraries I'm not familiar with. I have to make those judgments." (Participant E-19)

#### 5.2.4. Theme 4: The Importance of Metacognitive Strategies

Students who reported using LLMs effectively described specific strategies they employed:

"I never just copy-paste. I always read through the generated code carefully and try to understand each line. If something is unclear, I ask ChatGPT to explain it. Sometimes I even ask for multiple alternative solutions and compare them." (Participant E-08)

Another high-performing student explained:

"I use the AI as a teacher, not just a solution provider. When I get stuck, I ask it to explain the concept rather than just give me the answer. Then I try to implement it myself. If I get stuck again, I compare my solution with what the AI suggests and learn from the differences." (Participant E-41)

#### 5.2.5. Theme 5: Concerns About Long-Term Skill Development

Both groups expressed concerns about the long-term implications of AI reliance:

"I worry that when I graduate and start working, I won't be able to program without AI assistance. What if I'm in an interview and they ask me to code on a whiteboard? Or what if I'm somewhere without internet access?" (Participant E-22)

A control group participant reflected:

"I've deliberately chosen not to use AI for my assignments because I want to develop strong fundamentals. I'm concerned that if I become dependent on AI now, I'll never develop the problem-solving skills I need as a professional." (Participant C-14)

## 6. Discussion

### *6.1. Interpretation of Findings*

From a cognitive load perspective, the findings suggest that while LLMs reduce extraneous load, they may inadvertently reduce germane load—the mental effort dedicated to constructing mental schemas. This trade-off explains why LLM-assisted students struggled with transfer tasks, which require well-automated schemas.

The results of this study provide a nuanced picture of LLM integration in programming education. Consistent with previous research (Becker et al., 2023; Denny et al., 2024), LLM-

assisted students demonstrated higher productivity and fewer errors during the intervention phase. This finding supports Hypothesis 1 and aligns with cognitive load theory predictions that reducing extraneous cognitive load allows students to focus on higher-level aspects of programming.

However, the superior performance of the control group on the unaided post-test and transfer task raises important concerns. The 10% performance gap on unaided tasks and 14% gap on transfer tasks suggest that LLM assistance may impede the development of deep, transferable understanding. This finding supports Hypothesis 2 and echoes concerns raised by Kazemitabaar et al. (2023) about the potential for AI tools to create performance that appears competent in context but lacks underlying conceptual foundations.

The differential impact on various algorithmic thinking dimensions is particularly instructive. LLM-assisted students showed comparable gains in debugging and optimization skills—areas where AI tools provide direct feedback—but significantly smaller gains in problem decomposition, algorithm design, and code tracing. These latter skills require the kind of active cognitive engagement that may be reduced when AI provides ready-made solutions.

### *6.2. The Moderating Role of Metacognition*

The moderating effect of metacognitive ability emerged as a key finding. Students with high metacognitive awareness were able to use LLM tools effectively without compromising their learning outcomes, while students with low metacognitive awareness showed significant learning deficits. This finding extends our understanding of how individual differences interact with educational technology and suggests that metacognition may be particularly critical in AI-augmented learning environments.

### *6.3. Implications for Pedagogical Practice*

The findings suggest several implications for programming educators:

- Balanced integration frameworks: Educators should develop balanced frameworks specifying when AI assistance is appropriate, such as permitting AI for initial exploration but restricting it during assessments of fundamental skills.
- Metacognitive training: Curricula should explicitly teach metacognitive strategies for AI use, including techniques for critically evaluating AI outputs and monitoring comprehension.
- Process-oriented assessment: Assessment practices should shift from evaluating final products to evaluating processes, requiring students to document AI interactions and explain their reasoning.
- Targeted skill development: Instructional time should be strategically allocated to develop skills most compromised by AI assistance (problem decomposition, algorithm design) through activities that minimize AI use.

- Differentiated instruction: Students with weaker metacognitive skills may benefit from more structured guidance and gradual introduction to AI tools.

### *6.4. Comparison with International Research*

The findings align with emerging international research while providing unique contributions. The productivity gains observed (32%) are comparable to those reported by Kazemitabaar et al. (2023). However, this study extends previous research by providing detailed analysis of which specific skills are most affected by AI assistance and by identifying metacognition as a critical moderating variable.

### *6.5. Limitations*

Several limitations should be considered when interpreting these findings:

- Sample characteristics: The sample consisted of second-year computer science students at a single institution, limiting generalizability to other populations.
- Quasi-experimental design: Group assignment was based on lab section enrollment rather than true randomization, limiting causal interpretation.
- Duration: The 16-week study remains insufficient to assess long-term effects on professional skill development.
- AI tool evolution: The specific tools used may evolve or be superseded, potentially altering their educational impacts.
- Task types: The programming tasks may not fully represent professional challenges.
- Measurement challenges: Assessing algorithmic thinking is inherently complex, and instruments may not capture all relevant dimensions.

## 7. Conclusion

This study investigated the impact of LLM-based assistants on algorithmic thinking development in programming education. Within the limitations of this single-institution, quasi-experimental study, several conclusions can be drawn:

First, LLM tools substantially increase programming efficiency and reduce syntax errors during learning. Second, LLM assistance differentially affects algorithmic thinking dimensions, with problem decomposition, algorithm design, and code tracing most compromised. Third, the effectiveness of LLM integration depends critically on students' metacognitive abilities. Fourth, LLM-assisted students show difficulty applying knowledge to novel problems. Fifth, LLM use is associated with increased self-efficacy but decreased intrinsic motivation.

As AI tools become increasingly ubiquitous, the question is no longer whether to integrate them, but how to integrate them effectively. Successful integration requires explicit instruction in metacognitive strategies, redesigned assessments emphasizing process over product, and differentiated approaches accounting for individual differences.

Future research should investigate long-term effects on professional outcomes, explore metacognitive training interventions, and examine how AI tools might be designed to better support learning.

**Acknowledgements**

The authors thank the students who participated in this study and colleagues at the Department of Computer Science, Azerbaijan State Pedagogical University, for their support.

**Ethics Statement**

This study received ethical approval from the Research Ethics Committee of Azerbaijan State Pedagogical University (Approval #2024-08-15). All participants provided written informed consent.

**Funding Statement**

This research received no specific grant from any funding agency.

**Declaration of Competing Interest**

The authors declare no competing interests.

**References**

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & 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* (Vol. 1, pp. 500–506).

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2), 32–36.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology, 3*(2), 77–101.

Denny, P., Prather, J., Becker, B. A., Finnie-Ansley, J., Hellas, A., Leinonen, J., Luxton-Reilly, A., Reeves, B. N., Santos, E. A., & Sarsa, S. (2024). Computing education in the era of generative AI. *Communications of the ACM, 67*(2), 56–67.

Fataliyeva, A. (2025a). Proqramlaşdırma bacarıqlarının inkişafında psixoloji ölçmələrin rolu: idrak, motivasiya və sosial dinamikaların təhlili. *Naxçıvan Müəllimlər İnstitutu. Elmi əsərlər, 2025*(1), 130–137.

Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American Psychologist, 34*(10), 906–911.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Holmes, W., Bialik, M., & Fadel, C. (2019). *Artificial intelligence in education: Promises and implications for teaching and learning*. Center for Curriculum Redesign.

Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D., & 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* (pp. 1–23).

Lau, S., & Guo, P. (2023). 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. In *Proceedings of the 2023 ACM Conference on International Computing Education Research* (pp. 106–121).

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin, 41*(3), 161–165.

Medeiros, R. P., Ramalho, G. L., & Falcao, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education, 62*(2), 77–90.

Pesovski, I., Vorkel, D., & Trajkovik, V. (2024). AI-powered education: Rethinking the way programming is taught using AI tools and reversed Bloom's taxonomy. In *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)* (pp. 1–6). IEEE.

Pintrich, P. R., Smith, D. A., García, T., & McKeachie, W. J. (1991). *A manual for the use of the Motivated Strategies for Learning Questionnaire (MSLQ)*. University of Michigan.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.

Schraw, G., & Dennison, R. S. (1994). Assessing metacognitive awareness. *Contemporary Educational Psychology, 19*(4), 460–475.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*(2), 257–285.

Van Merriënboer, J. J., & Sweller, J. (2005). Cognitive load theory and complex learning: Recent developments and future directions. *Educational Psychology Review, 17*(2), 147–177.

Lynn, M. R. (1986). Determination and quantification of content validity. Nursing Research, 35(6), 382-385.