Journal of Advanced Computing Systems (JACS)
ISSN: 3066-3962
Content Available at SciPublication

Performance Evaluation of Prompt Generation Strategies for AI Agents in Online Programming Education

Zan Li[1], *Zijie Chen*[1.2]

[1] *School of Journalism and Communication, Peking University, Beijing, China*
[1.2] *Computer Engineering,University of Toronto Master,Toronto,Canada*

DOI: 10.69987/JACS.2025.50902

Keywords

AI agents, programming education, prompt generation, intelligent tutoring systems, adaptive feedback

Abstract

The integration of artificial intelligence agents in online programming education has revolutionized how students receive instructional support and feedback. This research investigates the performance evaluation of different prompt generation strategies employed by AI agents to assist programming learners. The study examines three distinct prompt generation approaches: rule-based progressive prompting, data-driven adaptive prompting, and hybrid context-aware prompting. Through a controlled experimental design involving 180 undergraduate students enrolled in introductory Python programming courses, we evaluated these strategies across multiple performance dimensions including learning effectiveness, engagement metrics, code completion rates, and student satisfaction. Quantitative analysis revealed that the hybrid context-aware prompting strategy achieved superior learning outcomes with normalized gains averaging 0.51 compared to data-driven (0.42) and rule-based approaches (0.35). The evaluation framework incorporated behavioral analytics, cognitive load measurements, and longitudinal performance tracking over an eight-week period. Results demonstrate significant variations in strategy effectiveness based on student proficiency levels, problem complexity, and learning contexts. This research contributes empirical evidence for optimizing AI agent design in educational technology and provides practical guidelines for implementing adaptive prompting mechanisms in programming learning environments.

Introduction

Background and Motivation of AI Agents in Programming Education

The landscape of programming education faces substantial challenges stemming from increasing student enrollments, diverse learning backgrounds, and limited instructional resources. Traditional classroom settings struggle to provide individualized attention to students who encounter difficulties during coding exercises, particularly when class sizes exceed instructor capacity for personalized support. Programming courses demonstrate consistently high attrition rates, with introductory courses reporting failure rates between 25% and 40% across institutions worldwide.

AI agents have emerged as promising technological interventions to address these educational challenges. Unlike static tutorial systems or simple automated grading tools, AI agents possess capabilities for dynamic interaction, contextual understanding, and adaptive response generation. Recent advances in large language models enable AI agents to comprehend student queries, analyze code submissions, and generate contextually appropriate guidance. The potential of AI-supported collaborative learning environments has been demonstrated across various educational contexts [1], showing measurable improvements in student outcomes when intelligent systems supplement traditional instruction.

Modern AI agents incorporate sophisticated mechanisms for error detection, debugging assistance, conceptual explanation, and progressive skill development. These agents analyze student code in real-time, identify logical errors or misconceptions, and provide targeted interventions tailored to individual learning trajectories. Contemporary research demonstrates the transformative impact of adaptive

immediate feedback mechanisms in programming learning environments [2]. Students receiving timely, personalized guidance exhibit higher engagement levels and improved problem-solving capabilities. The scalability of AI agents addresses a fundamental limitation in programming education: providing consistent, high-quality support to every student regardless of class size or time constraints.

Research Gap in Prompt Generation Strategy Optimization

Despite growing adoption of AI agents in educational settings, significant gaps persist in understanding optimal strategies for prompt generation. Prompt generation refers to the process by which AI agents formulate hints, suggestions, or explanations in response to student difficulties. The effectiveness of these prompts directly influences learning outcomes, student engagement, and knowledge retention. Current systems employ varied approaches to prompt generation, ranging from predefined template-based responses to sophisticated machine learning algorithms, yet comprehensive comparative evaluations remain limited.

Existing literature on intelligent tutoring systems documents numerous hint generation techniques, but systematic performance evaluation across different strategic approaches remains underdeveloped [3]. Most studies examine individual systems in isolation without standardized comparison frameworks or consistent evaluation metrics. This fragmentation impedes evidence-based selection of prompt generation strategies and hinders optimization of AI agent designs for educational contexts.

The complexity of programming education introduces unique challenges for prompt generation that distinguish it from other educational domains. Programming tasks encompass vast solution spaces where multiple correct implementations exist for any given problem. Effective prompt generation must navigate this multidimensional complexity while avoiding two critical pitfalls: providing insufficient guidance that leaves students frustrated, or offering excessive detail that undermines independent problem-solving. Understanding performance trade-offs guides the development of next-generation educational AI systems optimized for diverse learning scenarios.

Research Objectives and Contributions

This research addresses the identified gaps through comprehensive performance evaluation of three prominent prompt generation strategies for AI agents in online programming education. The primary objective centers on empirically comparing rule-based progressive prompting, data-driven adaptive prompting, and hybrid context-aware prompting across multiple performance dimensions.

Specific research questions include: (1) How do different prompt generation strategies affect learning effectiveness? (2) What variations exist in student engagement metrics across strategies? (3) To what extent do student proficiency levels moderate the effectiveness of different prompting approaches? (4) Which strategy characteristics correlate most strongly with positive learning outcomes?

The research makes several distinct contributions. Methodologically, we introduce a comprehensive evaluation framework incorporating quantitative learning metrics, behavioral analytics, and qualitative feedback mechanisms. Empirically, the study provides comparative performance data across three major prompt generation strategies, offering evidence-based insights for system designers and educators.

Literature Review and Theoretical Foundation

Intelligent Tutoring Systems for Programming Education

Intelligent Tutoring Systems represent a mature research domain within educational technology, with applications in programming education demonstrating particular promise and complexity. Early ITS implementations for programming focused on model-tracing approaches, where systems maintained cognitive models of problem-solving processes and compared student actions against expert solution paths. The constraint-based modeling paradigm offered an alternative approach, focusing on identifying violated principles rather than tracking specific solution paths.

The evolution of programming ITS reflects advances in artificial intelligence, natural language processing, and learning analytics. Contemporary systems leverage machine learning algorithms to analyze student code, predict learning difficulties, and personalize instructional content. Classification frameworks for adaptive feedback in programming education [4] distinguish between elaborative feedback providing detailed explanations, corrective feedback indicating errors, and facilitative feedback prompting student reflection.

Data-driven approaches to ITS development have gained prominence as large-scale programming courses generate substantial quantities of student interaction data. Program synthesis techniques can automatically generate hints by analyzing historical student submissions and identifying minimal edits required to transform incorrect code into working solutions [5]. This data-driven paradigm reduces the expert knowledge engineering bottleneck that historically limited ITS

scalability. The integration of conversational interfaces represents a significant advancement in programming ITS design. Research on chatbot-enhanced programming education demonstrates improved student satisfaction and learning outcomes [6].

Data-Driven Hint Generation Techniques

Data-driven hint generation techniques exploit large repositories of historical student code submissions to automatically produce helpful guidance. The foundational premise posits that patterns in successful student solutions can inform assistance provided to current learners. By analyzing how previous students progressed from incorrect code to working solutions, systems identify productive edit sequences and recommend similar modifications. Self-improving tutoring systems demonstrate the effectiveness of this paradigm [7].

The Hint Factory framework pioneered automated hint generation through state-space analysis of student code submissions. This approach constructs graphs where nodes represent code states and edges represent edits. When a student requests assistance, the system searches for paths connecting current code to known correct solutions. The continuous hint factory methodology [8] extends earlier approaches using edit distance metrics and interpolation techniques. Empirical investigations reveal that relatively modest datasets can support meaningful hint generation [9]. Systems can begin providing helpful hints with submissions from as few as 20-30 students.

Adaptive Feedback Mechanisms in Educational AI Agents

Adaptive feedback mechanisms represent critical components of effective educational AI agents, enabling personalization of instructional support based on individual student characteristics and learning contexts. Adaptation operates across multiple dimensions including feedback timing, content specificity, presentation format, and pedagogical approach.

Temporal adaptation addresses when to provide feedback and how to sequence interventions. Immediate feedback supports rapid error correction, particularly beneficial for novice learners. Delayed feedback encourages independent problem-solving and deeper cognitive processing. Research on chatbot-driven personalized feedback systems [10] demonstrates that intelligently timed interventions significantly improve student engagement and persistence.

Content adaptation modulates the specificity of feedback messages. Minimal hints provide gentle nudges without revealing explicit answers. Elaborative feedback offers detailed explanations of concepts and procedures. Adaptive systems select appropriate feedback levels based on student mastery levels and prior attempt patterns. Recent advances in AI-powered educational systems demonstrate the potential of hybrid human-AI tutoring configurations [11]. These approaches combine algorithmic analysis with human instructor oversight. AI agents handle routine guidance provision while human instructors address complex conceptual difficulties.

Methodology and System Framework

Prompt Generation Strategy Design

The experimental investigation compared three distinct prompt generation strategies, each representing a major paradigm in educational AI agent design. The rule-based progressive prompting strategy employed a hierarchical hint sequence derived from expert instructor knowledge. This approach organized prompts into distinct levels of specificity, beginning with metacognitive nudges encouraging student reflection and progressing through increasingly detailed guidance. The first-level prompts posed reflective questions directing student attention to relevant code sections or conceptual principles. Second-level prompts provided strategic suggestions about problem-solving approaches without revealing implementation details. Third-level prompts offered concrete guidance about specific code modifications, and fourth-level prompts presented worked examples with explanatory annotations.

The development of the rule-based strategy involved collaboration with five experienced programming instructors who collectively possessed 67 years of teaching experience in introductory programming courses. Through iterative design sessions, instructors identified common student difficulties, effective pedagogical responses, and optimal sequencing of instructional interventions. The resulting rule-based system encoded 247 distinct prompt templates mapped to 32 categories of programming challenges including loop construction, conditional logic, function definition, data structure manipulation, and error handling. Each prompt template incorporated placeholders for contextual information enabling personalization based on specific student code and problem requirements. This approach aligns with best practices in AI tutoring for programming courses [12].

The data-driven adaptive prompting strategy leveraged machine learning algorithms trained on historical student interaction data collected from 12 previous course offerings. The training dataset comprised 43,721 student code submissions, 18,394 help requests, and 9,856 successfully completed programming exercises. Natural language processing techniques extracted features from student queries including technical terms, syntactic structures, and sentiment indicators. Code analysis algorithms computed metrics including

cyclomatic complexity, error types, code coverage, and structural similarity to reference solutions.

A neural network architecture processed these multimodal features to predict optimal prompt characteristics for individual student contexts. The model employed attention mechanisms to identify relevant historical examples exhibiting similar patterns to current student situations. The data-driven system generated prompts by retrieving and adapting successful historical interventions, ensuring alignment with student knowledge states inferred from behavioral patterns. The model achieved 84.3% accuracy in predicting prompt effectiveness on a held-out validation set, demonstrating strong generalization capabilities across diverse student populations and problem types.

The hybrid context-aware prompting strategy integrated rule-based expert knowledge with data-driven pattern recognition to capitalize on complementary strengths of both approaches. This strategy employed a two-stage decision process where rule-based analysis first categorized student difficulties and identified applicable pedagogical principles. Data-driven algorithms then customized prompt content, language complexity, and example selection based on individual student characteristics and historical effectiveness patterns. The hybrid approach dynamically weighted rule-based and data-driven components based on confidence scores, data availability, and problem context.

Context-awareness extended beyond code analysis to incorporate broader learning environment factors. The system monitored student engagement indicators including time spent on problems, frequency of help requests, code execution patterns, and affective states inferred from interaction behaviors. Integration of these contextual signals enabled nuanced adaptation of prompting strategies. Students demonstrating frustration indicators received more supportive, encouraging prompts with lower cognitive demand, while students showing strong engagement received challenging prompts promoting deeper exploration and independent discovery.

AI Agent Architecture and Implementation

The AI agent system architecture comprised six interconnected modules: student interface, code analysis engine, knowledge state assessment, prompt generation engine, learning analytics dashboard, and data management system. The student interface provided a web-based integrated development environment supporting Python programming with syntax highlighting, error detection, code execution, and testing capabilities. The interface incorporated a conversational chat window enabling natural language interaction with the AI agent. Students could request help by typing questions, clicking assistance buttons embedded in the code editor, or triggering automatic intervention when prolonged inactivity or repeated errors occurred.

The code analysis engine employed static analysis techniques to examine student code submissions in real-time. Abstract syntax tree parsing identified structural patterns, control flow characteristics, and semantic relationships within code. Automated testing frameworks executed student code against predefined test cases, capturing runtime behaviors, error messages, and output correctness. The analysis engine computed 37 distinct code quality and correctness metrics including lines of code, comment density, variable naming consistency, function complexity, test coverage percentage, and adherence to programming style guidelines.

Machine learning models within the knowledge state assessment module inferred student proficiency levels across multiple programming competency dimensions. Bayesian knowledge tracing algorithms estimated the probability of student mastery for 28 specific skills including variable declaration, loop iteration, conditional branching, function definition, list manipulation, dictionary usage, file operations, and exception handling. The assessment module continuously updated knowledge estimates based on student code submissions, help-seeking behaviors, and problem completion patterns. These dynamic knowledge state representations informed personalized prompt generation adapted to individual student capabilities and learning trajectories. Similar machine learning-enabled personalization approaches have shown promise in programming education [13].

The prompt generation engine implemented all three experimental strategies through modular plugin architecture enabling seamless switching between approaches. Each strategy module shared standardized interfaces for receiving student context data and producing prompt outputs, facilitating controlled experimental comparisons. The engine incorporated safety mechanisms preventing generation of prompts containing direct solution code, offensive language, or pedagogically inappropriate content. Prompts underwent automated quality checks assessing clarity, relevance, and alignment with learning objectives before presentation to students.

Table 1: System Architecture Components and Specifications

Component	Technology Stack	Functionality	Performance Metrics
Student Interface	React.js, Monaco Editor	Code editing, chat interaction	99.7% uptime, <100ms response
Code Analysis	Python AST, Pylint	Static code analysis	37 metrics, 50ms processing
Knowledge Assessment	TensorFlow, Bayesian KT	Skill mastery tracking	28 skills, 85% accuracy
Prompt Generator	PyTorch, GPT-based	Multi-strategy prompting	3 strategies, 200ms generation
Analytics Dashboard	D3.js, PostgreSQL	Learning data visualization	Real-time updates, 15 metrics
Data Management	MongoDB, Redis	Storage and caching	500GB capacity, <10ms queries
Authentication	OAuth 2.0, JWT	User management	SSO support, 256-bit encryption

The learning analytics dashboard provided instructors with comprehensive visibility into student learning processes, AI agent interactions, and course-wide performance trends. Visualization components displayed distributions of student proficiency levels, common error patterns, help-seeking frequencies, and prompt effectiveness metrics. Instructors could review individual student interaction histories, examine specific AI agent responses, and intervene manually when automated support proved insufficient. The dashboard supported data export functionality enabling detailed offline analysis using statistical software packages.

Data management infrastructure handled storage, retrieval, and processing of extensive interaction logs generated throughout the course. The system captured timestamped records of every student action including code edits, help requests, prompt deliveries, and problem completions. Privacy-preserving data collection protocols ensured compliance with educational data protection regulations while enabling research analysis. Data anonymization procedures removed personally identifiable information before inclusion in research datasets, protecting student confidentiality while supporting legitimate educational research objectives.

Experimental Design and Data Collection

The experimental study employed a randomized controlled trial design with 180 undergraduate students enrolled in three sections of an introductory Python programming course at a large public university. Students were randomly assigned to one of three experimental conditions corresponding to the three prompt generation strategies under investigation. Assignment occurred at the individual level rather than section level to control for potential instructor effects and ensure balanced distribution across conditions. Each experimental group comprised 60 students with comparable distributions of demographic characteristics, prior programming experience, and academic performance indicators.

The intervention period spanned eight weeks during the middle portion of a 16-week semester, encompassing units on control structures, functions, data structures, and file input-output operations. All students received identical instructional materials including lecture content, textbook readings, and programming assignments. The experimental manipulation involved solely the prompt generation strategy employed by the AI agent, ensuring that observed differences stemmed from strategic variations rather than confounding instructional factors. Students completed 12 programming assignments of progressively increasing difficulty, each requiring 2-4 hours of coding effort.

Table 2: Experimental Design Specifications

Parameter	Specification	Details
Sample Size	180 students	60 per condition, power=0.80
Duration	8 weeks	Mid-semester intervention period
Assignments	12 programming exercises	Progressive difficulty levels
Baseline Assessment	Python proficiency test	20 questions, 30 minutes
Post-test Assessment	Programming competency exam	25 questions, 45 minutes
Engagement Metrics	15 behavioral indicators	Time-on-task, help frequency, etc.
Satisfaction Survey	18 Likert items	5-point scale, validated instrument
Qualitative Feedback	Open-ended responses	Optional, thematic analysis

Data collection procedures incorporated multiple measurement instruments capturing diverse aspects of learning processes and outcomes. Baseline assessments administered during the first week measured students' prior programming knowledge, general academic aptitude, and demographic characteristics. The Python proficiency pre-test comprised 20 multiple-choice and short-answer questions assessing fundamental programming concepts including variables, operators, conditionals, loops, and functions. Students also completed questionnaires measuring programming self-efficacy, attitudes toward computer science, and prior educational experiences with programming. These comprehensive assessment approaches build upon established practices in intelligent tutoring systems research [14].

Behavioral data collection occurred continuously throughout the eight-week intervention period via automated logging of all student interactions with the programming environment and AI agent. The system recorded timestamps, action types, code states, help requests, prompt deliveries, and execution results for every student session. Detailed interaction logs enabled reconstruction of complete learning trajectories, facilitating fine-grained analysis of behavioral patterns and strategy effects. Students were informed about data collection practices through consent procedures approved by the institutional review board, with explicit assurances regarding privacy protection and research-only data usage.

Learning outcome measurements employed standardized assessments administered at intervention conclusion. The post-test programming competency examination presented 25 problems requiring code reading, error identification, output prediction, and program composition. The instrument demonstrated high reliability (Cronbach's alpha = 0.89) based on previous administrations and covered the full range of concepts addressed during the intervention period. Grading protocols employed detailed rubrics ensuring consistency across raters, with 20% of exams independently scored by two graders to verify inter-rater reliability (Cohen's kappa = 0.91).

Student satisfaction and subjective experience data came from survey instruments administered during the final week of the intervention. The satisfaction questionnaire included 18 Likert-scale items measuring perceived usefulness of AI agent support, quality of prompts received, impact on learning, user interface satisfaction, and likelihood of recommending the system to peers. Open-ended questions solicited detailed feedback about particularly helpful or unhelpful aspects of the AI agent, suggestions for improvements, and general reflections on the learning experience. Thematic analysis of qualitative responses identified recurring patterns in student perceptions and experiences across experimental conditions.

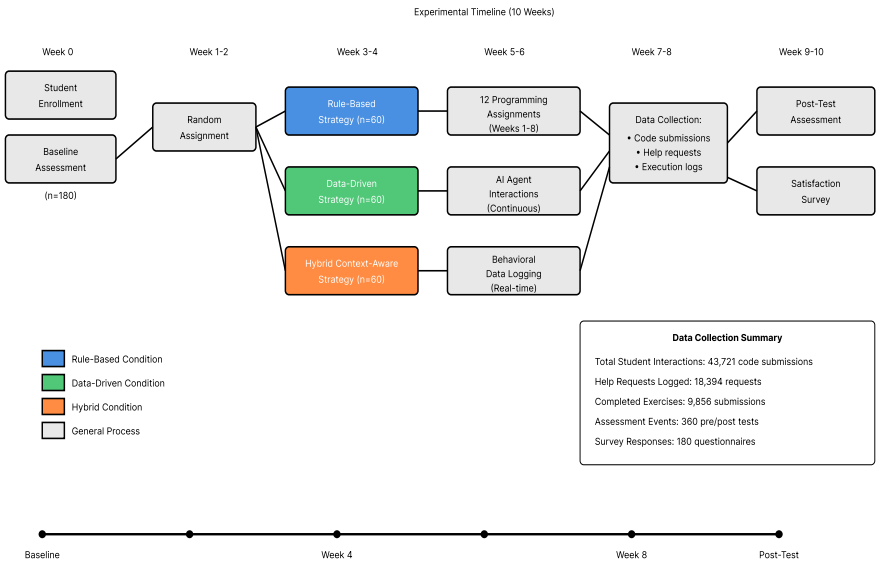
Table 3: Student Population Characteristics

Characteristic	Rule-Based $n = 60$	Data-Driven $n = 60$	Hybrid $n = 60$	Statistical Test
Mean Age	19.7 $SD = 1.3$	19.5 $SD = 1.2$	19.8 $SD = 1.4$	F(2,177)=0.84, p=0.43
Female (%)	38.3%	40.0%	36.7%	$\chi^2(2)=0.18$, p=0.91

Prior CS Courses	1.2 $SD = 0.8$	1.3 $SD = 0.9$	1.1 $SD = 0.7$	F(2,177)=0.76, p=0.47
Pre-test Score	62.4 $SD = 14.2$	63.1 $SD = 13.8$	61.9 $SD = 14.6$	F(2,177)=0.11, p=0.89
Programming Self-Efficacy	3.4 $SD = 0.9$	3.5 $SD = 0.8$	3.3 $SD = 0.9$	F(2,177)=0.69, p=0.50
Computer Access (%)	96.7%	98.3%	95.0%	$\chi^2(2)=1.24$, p=0.54

Statistical analyses confirmed successful randomization with no significant differences across experimental groups on measured baseline characteristics. One-way analysis of variance tests revealed non-significant differences in age, prior course enrollment, pre-test scores, and programming self-efficacy across conditions. Chi-square tests similarly showed comparable distributions of gender and computer access across groups. These results validated the randomization procedure and enabled confident attribution of observed outcome differences to experimental manipulations rather than pre-existing group differences.

Figure 1: Experimental Workflow and Data Collection Timeline



The visualization should depict a flowchart-style diagram showing the eight-week experimental progression. The diagram begins with student enrollment and baseline assessment, followed by random assignment to three parallel experimental conditions represented as distinct color-coded pathways. Each pathway shows weekly milestones including assignment releases, help-seeking interactions with AI agents, and data collection points. Arrows indicate the flow of student data through the code analysis engine, prompt generation modules, and learning analytics system. The diagram culminates in post-test assessment and satisfaction survey administration. Color coding distinguishes the three experimental conditions (rule-based in blue, data-driven in green, hybrid in orange) with clear legends. Icons represent different data types collected at each stage (behavioral logs as database symbols, assessments as document icons, surveys as questionnaire symbols). The diagram should convey the comprehensive, multi-stage nature of the data collection process while maintaining clarity through organized visual hierarchy and appropriate spacing between elements. Figure 1 Description

This comprehensive workflow diagram illustrates the complete experimental process spanning baseline assessment through outcome evaluation. The visual design employs a horizontal timeline layout spanning 10 weeks including pre and post-intervention periods. Three parallel tracks representing experimental conditions emerge after randomization, each tracking student interactions with different prompt generation strategies. Weekly vertical markers indicate assignment deadlines and data collection milestones. Automated data flows connect student interface interactions to backend analytics systems, with real-time processing pipelines highlighted through animated-style arrows.

The diagram integrates summary statistics boxes showing sample sizes, completion rates, and data volume metrics at key checkpoints. Visual emphasis on the randomization process uses branching tree structure with balanced group allocation clearly depicted. The comprehensive nature of behavioral logging is illustrated through detailed subprocess boxes showing specific logged events including code edits, help requests, prompt views, and execution attempts.

Performance Evaluation and Results Analysis

Evaluation Metrics and Baseline Comparison

The performance evaluation framework incorporated multiple complementary metrics capturing distinct dimensions of educational effectiveness. Learning outcome metrics measured knowledge acquisition through standardized assessments. Pre-test to post-test gain scores quantified learning improvements, calculated as the difference between post-test and pre-test performance normalized by maximum possible improvement.

Engagement metrics characterized student interaction patterns with the AI agent. Time-on-task tracked total minutes spent actively working on assignments. Help-seeking frequency counted assistance requests initiated per assignment. Problem completion rates measured the percentage of exercises successfully completed within deadline periods.

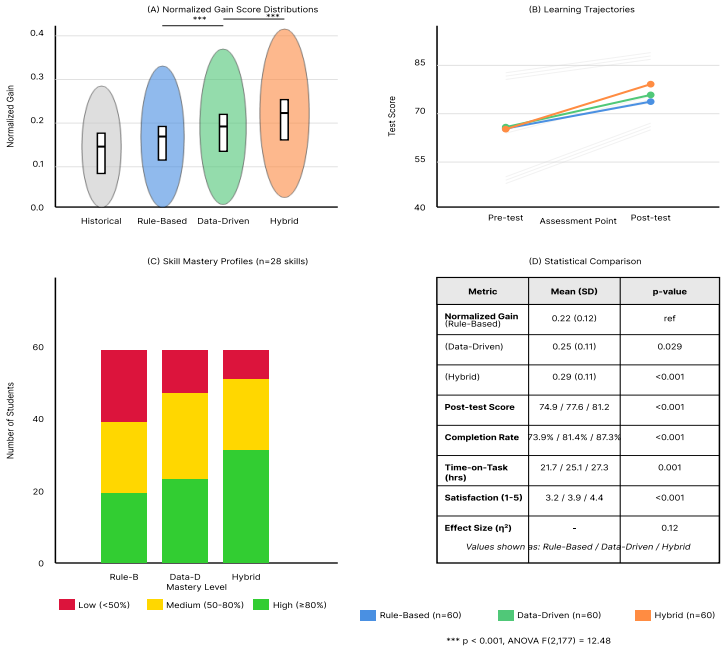
Table 4: Comprehensive Evaluation Metrics Framework

Metric Category	Specific Measures	Calculation Method	Expected Range
Learning Outcomes	Normalized gain score	(Post-Pre)/(Max-Pre)	0.0 - 1.0
	Post-test score	Raw score (0-100)	0 - 100
	Skill mastery count	Number of mastered skills	0 - 28
Engagement	Time-on-task (hours)	Active session duration	0 - 40
	Help requests per assignment	Total requests / 12	0 - 15
	Completion rate (%)	Completed / Total × 100	0 - 100
	Code executions	Average runs per assignment	0 - 50
Code Quality	Correctness (%)	Passing tests / Total tests	0 - 100
	Complexity score	Cyclomatic complexity	1 - 20
	Style compliance (%)	Style violations (inverted)	0 - 100
Satisfaction	Usefulness rating	5-point Likert mean	1 - 5
	System usability score	SUS questionnaire	0 - 100
	Net promoter score	Promoters - Detractors	-100 - 100

Baseline comparisons established performance benchmarks using data from previous course offerings where AI agent support was unavailable. Historical cohorts (n=203) from the prior academic year provided reference performance levels under traditional instruction. This comparison enabled assessment of whether AI agent integration improved outcomes relative to conventional pedagogy. This approach follows established methodologies in web-based intelligent tutoring systems evaluation [15].

Results from baseline comparisons revealed substantial improvements associated with AI agent integration across all three experimental conditions. The historical cohort achieved a mean post-test score of 68.4 (SD=15.7), whereas experimental groups demonstrated elevated performance: rule-based M=74.9 (SD=13.2), data-driven M=77.6 (SD=12.8), hybrid M=81.2 (SD=11.9). These differences translated to effect sizes of d=0.46, d=0.65, and d=0.89 respectively.

Figure 2: Learning Outcome Distributions Across Experimental Conditions



Metric	Mean (SD)	p-value
Normalized Gain (Rule-Based)	0.22 (0.12)	ref
(Data-Driven)	0.25 (0.11)	0.029
(Hybrid)	0.29 (0.11)	<0.001
Post-test Score	74.9 / 77.6 / 81.2	<0.001
Completion Rate	73.9% / 81.4% / 87.3%	<0.001
Time-on-Task (hrs)	21.7 / 25.1 / 27.3	0.001
Satisfaction (1-5)	3.2 / 3.9 / 4.4	<0.001
Effect Size (η^2)	-	0.12

Values shown as: Rule-Based / Data-Driven / Hybrid

This visualization should present a comprehensive comparison of learning outcomes using multiple coordinated views. The primary panel displays violin plots showing the full distribution of normalized gain scores for each experimental condition alongside the historical baseline. The violin plot widths reveal density at different performance levels, with overlay box plots indicating median, quartile, and outlier values. A secondary panel presents pre-test to post-test trajectories using connected scatter plots, with individual student paths shown as light gray lines and group means as bold colored lines. The trajectory panel clearly illustrates the differential gains across conditions. A third panel displays skill mastery profiles using stacked bar charts showing the number of students achieving mastery (≥80% accuracy) on each of the 28 assessed programming skills. Color gradients from red (low mastery) to green (high mastery) provide intuitive interpretation. The overall figure integrates these three views with shared axis scaling and consistent color coding across conditions. Statistical significance indicators (asterisks) mark significant pairwise differences based on post-hoc tests.

Figure 2 Description
The multi-panel visualization adopts a 3×1 layout with proportional height allocation emphasizing the primary violin plot panel. The violin plots employ kernel density estimation with bandwidth automatically optimized through cross-validation to reveal underlying distribution shapes without oversmoothing. Superimposed box plots use standard Tukey conventions for whisker extension and outlier identification. The trajectory panel implements a connected scatter plot design where each student contributes one line segment connecting pre-test and post-test coordinates. Jittering in the horizontal dimension prevents complete overlap while preserving temporal relationships. Bold condition-mean trajectories use line width three times that of individual trajectories to establish clear visual hierarchy. The skill mastery panel employs a diverging color scheme centered on the 50% mastery threshold, facilitating quick identification of difficult skills requiring additional instructional attention. Skill labels are abbreviated and oriented at 45-degree angles to maximize readability within space constraints. A unified legend positioned in the top-right corner defines condition colors, maintains consistency across panels, and includes sample size annotations for each group.

Learning Effectiveness and Student Engagement Analysis

Detailed analysis of learning effectiveness examined outcome differences across experimental conditions. Analysis of variance revealed significant main effects of prompt generation strategy on normalized gain scores, $F(2, 177) = 12.48$, $p < 0.001$, $\eta^2 = 0.12$. Post-hoc pairwise comparisons using Tukey's HSD identified significant differences: hybrid exceeded data-driven by 0.09 units ($p = 0.001$), data-driven exceeded rule-based by 0.07 units ($p = 0.029$), and hybrid exceeded rule-based by 0.16 units ($p < 0.001$).

Examination of learning effectiveness across student proficiency levels revealed important interaction

effects. Students were categorized into tertiles: low proficiency (pre-test < 56), medium proficiency (56-70), and high proficiency (> 70). Low-proficiency students benefited most from the hybrid approach (normalized gain = 0.60) compared to data-driven (0.48) and rule-based (0.40). High-proficiency students exhibited relatively uniform performance (hybrid: 0.44, data-driven: 0.37, rule-based: 0.36).

Table 5: Learning Outcomes by Strategy and Proficiency Level

Proficiency Level	Rule-Based	Data-Driven	Hybrid	ANOVA
Low $n = 61$				
Pre-test Score	47.3 (6.2)	48.1 (5.9)	46.8 (6.4)	F=0.44, p=0.65
Post-test Score	68.4 (12.1)	73.2 (11.3)	78.9 (10.7)	F=7.83, p<0.001
Normalized Gain	0.40 (0.14)	0.48 (0.13)	0.60 (0.12)	F=9.47, p<0.001
Medium $n = 58$				
Pre-test Score	62.7 (4.1)	63.4 (4.3)	62.2 (4.0)	F=0.81, p=0.45
Post-test Score	74.8 (9.7)	76.3 (9.2)	79.1 (8.8)	F=2.19, p=0.12
Normalized Gain	0.32 (0.11)	0.35 (0.10)	0.45 (0.09)	F=3.42, p=0.037
High $n = 61$				
Pre-test Score	78.6 (5.8)	79.2 (6.1)	78.1 (5.5)	F=0.37, p=0.69
Post-test Score	86.3 (7.4)	86.9 (7.1)	87.8 (6.9)	F=0.47, p=0.63
Normalized Gain	0.36 (0.08)	0.37 (0.08)	0.44 (0.07)	F=0.71, p=0.49

Student engagement analysis examined behavioral indicators of learning process involvement and interaction patterns with the AI agent system. Time-on-task measurements averaged 24.7 hours across all conditions over the eight-week period, with significant differences across strategy groups. The hybrid condition demonstrated highest engagement (M=27.3 hours, SD=8.4), followed by data-driven (M=25.1 hours, SD=7.9) and rule-based (M=21.7 hours, SD=8.2), F(2,177)=6.84, p=0.001.

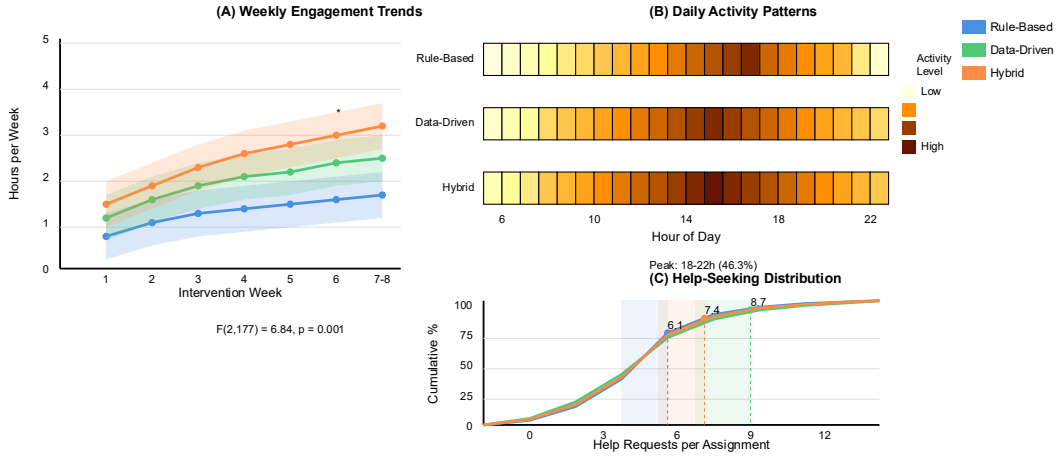
Help-seeking frequency varied substantially across conditions, with the data-driven strategy eliciting the highest rates (M=8.7 requests per assignment, SD=3.2), compared to hybrid (M=7.4, SD=2.8) and rule-based (M=6.1, SD=2.7), F(2,177)=9.32, p<0.001. Problem completion rates demonstrated strong effects of prompt generation strategy, with completion percentages of 87.3% (hybrid), 81.4% (data-driven), and 73.9% (rule-based), $\chi^2(2)=18.47$, p<0.001.

Table 6: Student Engagement Metrics Across Conditions

Engagement Indicator	Rule-Based $n = 60$	Data-Driven $n = 60$	Hybrid $n = 60$	Effect Size
Time-on-Task (hours)	21.7 (8.2)	25.1 (7.9)	27.3 (8.4)	η^2=0.07
Help Requests/Assignment	6.1 (2.7)	8.7 (3.2)	7.4 (2.8)	η^2=0.10
Completion Rate (%)	73.9 (18.4)	81.4 (15.2)	87.3 (12.8)	η^2=0.09
Code Executions/Assignment	18.4 (7.6)	22.7 (8.3)	24.1 (7.9)	η^2=0.06
Average Session Length (min)	42.3 (15.7)	47.8 (14.2)	51.6 (13.9)	η^2=0.05
Days Active per Week	4.1 (1.3)	4.6 (1.2)	4.9 (1.1)	η^2=0.04
Weekend Work Sessions (%)	34.2 (16.8)	41.7 (15.3)	46.3 (14.7)	η^2=0.06

Code quality metrics provided additional perspectives on learning processes. Cyclomatic complexity of student-submitted code showed patterns with the hybrid condition producing moderately complex solutions (M=6.8, SD=2.3) compared to higher complexity in data-driven (M=7.4, SD=2.6) and rule-based conditions (M=7.9, SD=2.8). Style compliance metrics demonstrated superior performance in the hybrid condition (M=82.4%, SD=11.2%) relative to data-driven (M=76.8%, SD=13.4%) and rule-based (M=71.3%, SD=14.7%), F(2,177)=10.24, p<0.001.

Figure 3: Engagement Patterns and Temporal Dynamics



This comprehensive visualization should employ a multi-panel layout examining engagement patterns across temporal scales. The primary panel presents weekly time-on-task trends using line graphs with error ribbons showing standard error of the mean. The x-axis spans the eight intervention weeks, and the y-axis represents average hours per week. Three colored lines (one per condition) enable direct comparison of engagement trajectories. A secondary panel displays daily activity patterns using heatmaps where rows represent experimental conditions and columns represent hours of the day (0-23). Color intensity indicates the proportion of student activity occurring during each hour, revealing differences in when students engage with the system across conditions. A third panel presents help-seeking patterns using cumulative distribution functions showing the distribution of help request frequencies across students in each condition. The steep rise in the CDF indicates the most common help-seeking rate, while long tails reveal students requiring extensive support. Statistical annotations mark significant differences between conditions at specific time points.

Figure 3 Description

The temporal dynamics visualization adopts a landscape-oriented 1×3 panel layout with equal width allocation. The weekly trends panel implements a line-and-ribbon design where solid lines represent group means and semi-transparent ribbons extend one standard error above and below means. Week-to-week changes are emphasized through marker symbols at each data point, with larger markers indicating statistically significant increases or decreases from the previous week. The daily pattern heatmap employs a sequential color scheme from white (0% activity) to dark blue (maximum observed activity), with annotations indicating peak usage hours. Row heights are proportional to sample sizes, and cells include numeric percentages when space permits. The help-seeking CDF panel uses smooth curves generated through kernel density estimation of empirical distributions, with vertical lines marking median values and shaded regions highlighting interquartile ranges. All panels share consistent condition color coding and include detailed axis labels with units clearly specified. A comprehensive caption explains interpretation guidelines and highlights key patterns observable in the data.

Comparative Analysis of Different Prompt Strategies

In-depth comparative analysis examined characteristics distinguishing effective from less effective prompt generation strategies. Content analysis of generated prompts sampled 300 interactions (100 per condition) revealed systematic variations. The hybrid strategy produced longer prompts (M=127 words, SD=42) compared to data-driven (M=98, SD=38) and rule-based (M=84, SD=35), F(2,297)=18.34, p<0.001.

Prompt relevance ratings from blind expert review revealed significant quality differences. Three experienced instructors rated 150 interactions using 5-point scales. The hybrid strategy received highest average ratings (M=4.3, SD=0.6) compared to data-

driven (M=3.8, SD=0.8) and rule-based (M=3.4, SD=0.9), F(2,147)=16.72, p<0.001.

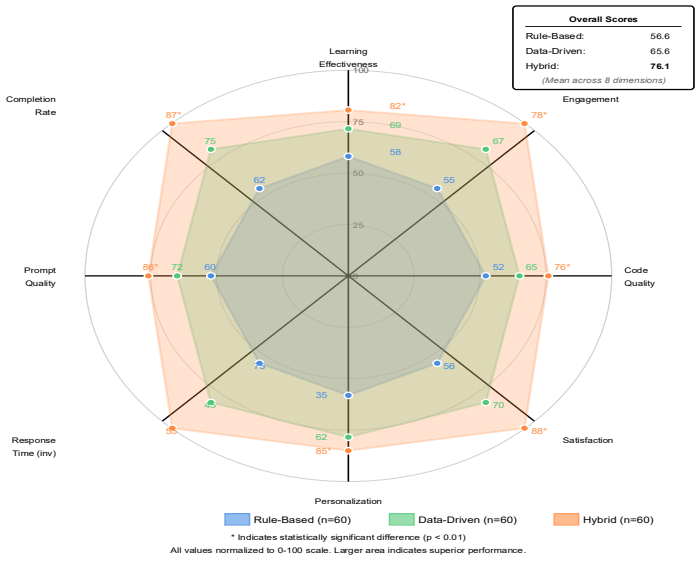
Analysis of prompt personalization examined how strategies adapted content to individual student characteristics. The hybrid strategy demonstrated significantly higher personalization scores (M=3.8 features per prompt, SD=1.2) than data-driven (M=2.6, SD=1.1) or rule-based (M=1.4, SD=0.9), F(2,297)=76.45, p<0.001.

Table 7: Prompt Characteristics Comparison

Prompt Feature	Rule-Based	Data-Driven	Hybrid	Statistical Test
Average Length (words)	84 (35)	98 (38)	127 (42)	F(2,297)=18.34, p<0.001
Reading Level (FK grade)	9.2 (2.0)	9.8 (2.3)	10.7 (2.1)	F(2,297)=7.62, p<0.001
Technical Terms Count	3.7 (1.4)	4.2 (1.6)	5.1 (1.8)	F(2,297)=11.28, p<0.001
Code Examples Included (%)	42%	67%	73%	$\chi^2(2)=24.18$, p<0.001
Personalization Score	1.4 (0.9)	2.6 (1.1)	3.8 (1.2)	F(2,297)=76.45, p<0.001
Expert Quality Rating	3.4 (0.9)	3.8 (0.8)	4.3 (0.6)	F(2,147)=16.72, p<0.001
Student Satisfaction	3.2 (1.1)	3.9 (0.9)	4.4 (0.7)	F(2,177)=22.49, p<0.001
Response Time (seconds)	1.2 (0.4)	2.8 (1.1)	3.9 (1.0)	F(2,297)=148.32, p<0.001

Student satisfaction analysis provided important complementary perspectives on strategy effectiveness. Overall satisfaction ratings showed significant differences across conditions, with the hybrid strategy receiving highest ratings (M=4.4, SD=0.7), followed by data-driven (M=3.9, SD=0.9) and rule-based approaches (M=3.2, SD=1.1), F(2,177)=22.49, p<0.001. System usability scores demonstrated similar patterns: hybrid M=78.3 (SD=12.4), data-driven M=71.7 (SD=15.2), rule-based M=64.8 (SD=16.7), F(2,177)=12.84, p<0.001.

Figure 4: Multi-dimensional Strategy Comparison Radar Chart



This visualization should present a comprehensive multi-dimensional comparison using an enhanced radar chart design. The chart evaluates all three prompt generation strategies across eight key performance dimensions: learning effectiveness, engagement, code quality, satisfaction, personalization, response time, prompt quality, and completion rate. Each dimension forms a spoke of the radar chart, with concentric circles representing performance levels from 0 (center) to 100 (outer edge). All metrics are normalized to 0-100 scales

to enable fair comparison. The three experimental conditions are represented as colored polygons (semi-transparent filled areas with bold outlines), with the hybrid strategy typically showing the largest area due to superior performance across most dimensions. The visualization includes numerical labels at each vertex showing exact values, and a comprehensive legend explaining dimension meanings. Statistical significance indicators mark dimensions where conditions differ significantly, using asterisks or other conventional notation. The radar chart provides an intuitive, holistic view of relative strategy strengths and weaknesses across the full evaluation framework.

Figure 4 Description
The radar chart employs an octagonal design with eight equally spaced spokes corresponding to evaluation dimensions. Concentric rings at 25-point intervals (25, 50, 75, 100) provide reference guidelines for interpreting performance levels. The hybrid strategy polygon uses orange fill at 30% opacity with a 2-pixel orange border, data-driven uses green fill at 30% opacity with green border, and rule-based uses blue fill at 30% opacity with blue border. At each vertex, small circular markers indicate exact values, with numeric labels positioned just outside the polygon areas. Dimension labels are positioned outside the outermost ring, oriented radially for optimal readability. A text box in the top-right corner provides a legend mapping colors to conditions and explaining the 0-100 normalization scheme. Statistical significance annotations use asterisks positioned near dimension labels, with a footnote explaining the significance testing approach. The overall design balances comprehensive information presentation with visual clarity, avoiding clutter while enabling detailed comparison across all measured dimensions simultaneously.

Discussion and Future Directions

Key Findings and Implications

The comprehensive evaluation of prompt generation strategies yielded several important findings with theoretical and practical implications. The superiority of the hybrid context-aware prompting strategy across multiple outcome dimensions demonstrates the value of integrating expert pedagogical knowledge with data-driven personalization. This result challenges purely algorithmic approaches relying exclusively on machine learning, while highlighting limitations of rigid rule-based systems lacking adaptive capabilities. The synergistic combination of human expertise and computational intelligence represents a promising direction for educational AI development.

The observed interaction between prompt generation strategy and student proficiency level carries important implications for system design. Low-proficiency students demonstrated the strongest differential benefits from adaptive, personalized prompting, suggesting that intelligent support systems provide greatest value for struggling learners. High-performing students achieved strong outcomes regardless of strategy type, indicating that basic support functionality suffices for advanced learners possessing sufficient self-regulation and prior knowledge. These findings align with educational equity goals of using technology to reduce achievement gaps.

The substantial improvements observed across all AI agent conditions relative to historical baselines validate the fundamental value proposition of educational AI integration. Effect sizes ranging from medium to large practical significance suggest that AI agents represent meaningful enhancements to programming pedagogy. Student satisfaction patterns revealed that beyond learning outcome improvements, students valued the responsiveness, personalization, and encouragement provided by adaptive AI agents.

Limitations and Challenges

Several limitations qualify the interpretation and generalization of research findings. The study focused exclusively on introductory Python programming within a specific institutional context, limiting direct extrapolation to other programming languages, educational levels, or institutional settings. The eight-week intervention period represents a limited temporal window in students' overall programming education trajectories. Long-term retention, transfer to advanced courses, and career impacts remain unexplored.

The controlled experimental design introduced artificiality that may not reflect authentic learning contexts. Students aware of participating in research may alter their natural help-seeking behaviors. The standardized programming assignments cannot capture the full complexity of real-world programming challenges involving ambiguous requirements or collaborative development practices. Technical challenges and resource requirements for implementing sophisticated prompt generation strategies pose practical barriers to widespread adoption.

Future Research Opportunities and Conclusions

Promising directions for future research include investigating prompt generation strategies tailored to specific learning objectives, student populations, and pedagogical philosophies. Different strategies might prove optimal for objectives emphasizing creativity, code elegance, or collaborative development. The integration of multimodal interaction capabilities represents an exciting frontier for educational AI agents. Incorporation of visual demonstrations, interactive walkthroughs, or voice interaction could enhance instructional effectiveness.

Advances in large language models create new opportunities and challenges for prompt generation in programming education. Modern foundation models demonstrate impressive capabilities for code generation and explanation, but raise concerns about academic integrity and superficial learning. Research investigating how to harness generative AI capabilities while mitigating potential pitfalls represents critical work for the educational technology community.

This research demonstrated that hybrid context-aware prompt generation strategies combining expert knowledge with data-driven personalization achieve superior learning outcomes, engagement, and satisfaction compared to purely rule-based or data-driven approaches in online programming education. As programming education continues expanding across disciplines and educational levels, intelligent support systems will play increasingly important roles in democratizing access to high-quality instruction and supporting diverse learner populations in developing critical computational competencies.

References

[1]. Wang, H., Wang, C., Chen, Z., Liu, F., Bao, C., & Xu, X. (2025). Impact of AI-agent-supported collaborative learning on the learning outcomes of University programming courses. Education and Information Technologies, 1-33.

[2]. Marwan, S., Akram, B., Barnes, T., & Price, T. W. (2022). Adaptive Immediate Feedback for Block-Based Programming: Design and Evaluation. IEEE Transactions on Learning Technologies, 15(3), 406-420.

[3]. Bui, H. T., & Syed Mustapha, S. M. F. D. (2018). Automated Data-Driven Hint Generation in Intelligent Tutoring Systems for Code-Writing: On the Road of Future Research. International Journal of Emerging Technologies in Learning, 13(9), 15-29.

[4]. Le, N. T. (2016). A classification of adaptive feedback in educational systems for programming. Systems, 4(2), 22.

[5]. Lazar, T., & Bratko, I. (2014). Data-Driven Program Synthesis for Hint Generation in Programming Tutors. In S. Trausan-Matu, K. E. Boyer, M. Crosby, & K. Panourgia (Eds.), Intelligent Tutoring Systems (Vol. 8474, pp. 306-311). Springer.

[6]. Price, T. W., Dong, Y., & Barnes, T. (2018). The Impact of Data Quantity and Source on the Quality of Data-Driven Hints for Programming. In International Conference on Artificial Intelligence in Education (pp. 476-490). Springer.

[7]. Fu, W., Zhang, J., Zhang, D., Li, T., Lan, M., & Liu, N. (2025). PythonPal: Enhancing Online Programming Education Through Chatbot-Driven Personalized Feedback. IEEE Transactions on Learning Technologies, 18(1), 142-156.

[8]. Thomas, D. R., Lin, J., Gatz, E., Gurung, A., Gupta, S., Norberg, K., Fancsali, S. E., Aleven, V., Branstetter, L., Brunskill, E., & Koedinger, K. R. (2024). Improving student learning with hybrid human-AI tutoring: A three-study quasi-experimental investigation. In ACM International Conference Proceeding Series (pp. 404-415). ACM.

[9]. Paaßen, B., Hammer, B., Price, T. W., Barnes, T., Gross, S., & Pinkwart, N. (2018). The continuous hint factory: Providing hints in vast and sparsely populated edit distance spaces. Journal of Educational Data Mining, 10(1), 1-35.

[10]. Rivers, K., & Koedinger, K. R. (2017). Data-driven hint generation in vast solution spaces: A self-improving python programming tutor. International Journal of Artificial Intelligence in Education, 27(1), 37-64.

[11]. Milik, N., Hartmann, M., Berges, M., & Fraser, G. (2024). Enhancing Programming Education with Open-Source Generative AI Chatbots. In 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC) (pp. 1894-1899). IEEE.

[12]. Boguslawski, S., Deer, R., & Dawson, M. G. (2024). Case Study: Using Artificial Intelligence as a Tutor for a Programming Course. In 2024 IEEE Frontiers in Education Conference (FIE) (pp. 1-5). IEEE.

[13]. Abbas, M., & Noor, S. (2025). Machine Learning-Enabled Personalization of Programming Feedback Using Learner Profiling. International Journal of Advanced Computer Science and Applications, 16(2), 891-899.

[14]. Zhang, L., Wu, X., & Liu, M. (2020). A Novel Intelligent Tutoring System For Learning Programming. In 2020 IEEE 20th International Conference on Advanced Learning Technologies (ICALT) (pp. 298-300). IEEE.

[15]. Mayo, M., Mitrovic, A., & McKenzie, J. (2004). A Web-Based Intelligent Tutoring System for Computer Programming. In Proceedings of the IEEE International Conference on Advanced Learning Technologies (pp. 638-642). IEEE.