

RESEARCH ARTICLE

Designing Intelligent Tutoring Systems for Personalized Programming Education

Altmisheva Iroda Mavlyanovna

Independent Researcher, Gulistan State Pedagogical Institute, Uzbekistan

VOLUME: Vol.06 Issue03 2026

PAGE: 265-275

Copyright © 2026 European International Journal of Pedagogics, this is an open-access article distributed under the terms of the Creative Commons Attribution-Noncommercial-Share Alike 4.0 International License. Licensed under Creative Commons License a Creative Commons Attribution 4.0 International License.

Abstract

This article presents a methodological framework for designing intelligent tutoring systems (ITS) for personalized programming education in higher education. The study addresses a persistent problem in programming courses: students enter with heterogeneous prior knowledge, progress at different speeds, and require different kinds of feedback when solving coding problems. The proposed ITS model integrates five core layers: learner diagnosis, student modeling, domain modeling, pedagogical decision-making, and adaptive feedback. The methodology combines pedagogical analysis, learning analytics, diagnostic programming tasks, student modeling, and a quasi-experimental design with control and experimental groups. Model-based results indicate that a personalized ITS can improve programming competence by strengthening theoretical understanding, algorithmic thinking, code-writing quality, debugging skills, and reflective self-regulation. The article also provides a conceptual model, assessment rubrics, experimental tables, and implementation recommendations for institutions planning to introduce ITS-supported programming instruction. The contribution of the study lies in connecting intelligent tutoring architecture with programming-specific competence indicators and in proposing a practical design model that supports personalized learning paths, real-time feedback, and teacher-facing analytics.

KEY WORDS

Intelligent tutoring system, personalized programming education, adaptive feedback, student modeling, learning analytics, programming competence, code assessment, higher education.

1. INTRODUCTION

1.1. Relevance of the problem

Programming education has become one of the central components of higher education programs related to computer science, software engineering, information systems, data analytics and digital transformation. However, introductory and intermediate programming courses remain difficult for many students because programming requires the simultaneous development of conceptual knowledge,

algorithmic reasoning, syntactic accuracy, debugging strategies and problem-solving habits. In heterogeneous classrooms, some students quickly understand syntax but cannot design algorithms independently, while others can reason about a solution but struggle to transform it into executable code.

Traditional programming instruction often relies on lectures, common laboratory tasks and summative assessment. This

model does not always identify the exact source of student difficulty. A wrong program may result from a misunderstanding of the problem statement, an incorrect algorithm, a syntactic mistake, a faulty condition, an incomplete loop or a misconception about data structures. Therefore, feedback must be more granular than simply marking a solution as correct or incorrect. Intelligent tutoring systems are promising because they can diagnose learner actions, adapt task difficulty, deliver hints and generate feedback at the moment when the learner needs it.

Recent work on ITS in programming education emphasizes the importance of tailoring tutoring interventions to different levels of programming experience. Fodouop Kouam (2024) shows that ITS usefulness and satisfaction can vary depending on learners' programming proficiency, which means that the design of ITS must account for beginners, intermediate learners and advanced students rather than treating the class as a uniform group. Similarly, Lai et al. (2025) emphasize the role of real-time personalized feedback and coding behavior analysis in intelligent tutoring systems for coding and learning. These findings indicate that designing an ITS for programming education is not only a technical task but also a pedagogical design problem.

The present article proposes a design model for an intelligent tutoring system that supports personalized programming education. The model is built around five competence dimensions: theoretical programming knowledge, algorithmic thinking, code-writing quality, debugging and error correction, and reflective self-regulated learning. The system aims to help students move through individualized pathways in which diagnostic results, learning analytics and automated feedback continuously update the next learning task.

1.2. Literature review

The literature on intelligent tutoring systems has developed around the idea that computer-based learning environments can emulate some functions of a human tutor: diagnosing the learner, selecting tasks, providing hints and monitoring progress. A recent systematic review by Letourneau et al. (2025) reports that ITS interventions generally show positive effects on learning and performance, although the magnitude of effects varies by design, duration and comparison condition. This means that ITS should not be viewed as a universal solution, but as a design-sensitive technology that must be aligned with pedagogy, content and learner characteristics.

In programming education, Crow, Luxton-Reilly and Wuensche (2018) reviewed intelligent tutoring systems for programming and found that most systems include interactive programming exercises, while the range of supplementary features such as plans, quizzes, examples and worked solutions varies substantially. This is important because programming learners require different forms of support: conceptual explanations, algorithmic scaffolds, code examples, debugging guidance and reflective prompts. A well-designed ITS should therefore combine multiple forms of support rather than relying only on automated code checking.

Feedback is one of the most important components of ITS for programming education. Keuning, Jeuring and Heeren (2018) synthesized automated feedback generation for programming exercises and showed that feedback may target syntax, style, test cases, logic, performance, misconceptions and next steps. In an ITS, feedback should be formative, actionable and adaptive. It should tell students not merely what is wrong but why it is wrong and what strategy can help them correct the solution.

Personalized learning research supports the claim that adaptive systems can improve academic performance, engagement and learning when they are implemented with clear pedagogical purposes. du Plooy, Casteleijn and Franzsen (2024) reviewed personalized adaptive learning in higher education and concluded that such systems have the potential to positively affect academic performance and student engagement, though technological and implementation limitations must be considered. This is consistent with the idea that ITS effectiveness depends on content quality, diagnostic accuracy, usability and teacher integration.

Learning analytics and student modeling provide the data foundation for personalization. Ifenthaler and Yau (2020) describe learning analytics as a means of supporting study success by identifying patterns in learner activity and enabling timely support. In programming education, learning analytics can include attempted solutions, compilation errors, runtime errors, time on task, hint requests, test-case failures, revision frequency and code quality indicators. These data can be transformed into a student model that guides the next tutoring intervention.

A growing body of recent work connects AI and personalized learning. Fortuna et al. (2025) mapped AI in personalized learning across tertiary and higher education and reported that AI-supported systems increasingly employ learner

modeling, adaptive recommendation and feedback mechanisms. Farhood et al. (2025) similarly emphasize the role of AI-based personalization in helping students meet unique educational needs. Within programming education, these ideas must be operationalized through code-specific diagnostics, algorithmic reasoning indicators and feedback that supports independent debugging.

1.3. Research aim and questions

The aim of the study is to design and justify a methodological model for an intelligent tutoring system that supports personalized programming education in higher education. The study is guided by the following research questions: (1) What components should an ITS include to personalize programming education? (2) How can student modeling and learning analytics be used to select adaptive programming tasks and feedback? (3) What competence indicators can be used to evaluate student progress? (4) What model-based effects may be expected when such an ITS is implemented in

a programming course?

1.4. Scientific novelty

- A programming-specific ITS design model is proposed, integrating learner diagnosis, student modeling, domain modeling, pedagogical decision-making and adaptive feedback.
- A competence-oriented assessment framework is developed for theoretical knowledge, algorithmic thinking, code quality, debugging and reflective learning.
- A methodology for generating personalized programming learning pathways is formulated through the combination of diagnostic assessment, learning analytics and recommendation mechanisms.
- A teacher-facing analytics component is included to support evidence-based instructional decisions rather than replacing the instructor.

Table 1. Pedagogical problems and ITS-based design responses

Problem in programming education	ITS design response	Expected pedagogical effect
Different prior knowledge	Initial diagnostic tasks and learner classification	Students receive tasks aligned with their readiness level
Frequent syntax and logic errors	Automated error detection and adaptive hints	Students correct errors earlier and learn debugging strategies
Weak algorithmic reasoning	Scaffolded algorithm design tasks before coding	Students improve decomposition and solution planning
Low self-regulation	Progress dashboard and reflective prompts	Students monitor their own learning process
High teacher workload	Teacher analytics and automated formative feedback	Instructor focuses on targeted intervention

2. METHODS

2.1. Research design

The study uses a design-based and quasi-experimental methodological logic. The design-based part develops an ITS model aligned with programming competence indicators. The quasi-experimental part describes how the model can be implemented with a control group and an experimental group in a higher education programming course. Because the article

is intended as an IMRAD-based design study, the numerical results presented in the Results section are model-based sample indicators that should be replaced by real institutional data before journal submission.

The design process included four stages: analysis of programming learning difficulties, definition of competence indicators, construction of the ITS architecture and development of a trial evaluation plan. The model assumes

that programming competence grows when students receive tasks matched to their ability, formative feedback that explains errors, and opportunities for repeated practice and reflection.

2.2. Participants and context

The proposed experiment is designed for undergraduate students enrolled in an introductory or intermediate programming course. Participants are divided into a control group that studies through traditional classroom instruction and an experimental group that studies through the ITS-supported personalized learning environment. Both groups receive the same core curriculum topics, but the experimental group receives adaptive programming tasks, automated feedback, personalized hints and progress monitoring through the ITS.

2.3. Intelligent tutoring system architecture

The ITS architecture consists of six interacting components: a domain model, a student model, a pedagogical model, an assessment model, a feedback model and a teacher analytics dashboard. The domain model represents programming concepts, tasks, solution strategies and test cases. The student model stores diagnostic results, task performance, misconceptions, code submissions, hint requests and learning progress. The pedagogical model selects tasks and interventions. The assessment model evaluates code, algorithms and debugging behavior. The feedback model delivers individualized hints and explanations. The teacher dashboard presents aggregated and individual analytics for instructional decision-making.

TELLIGENT TUTORING SYSTEM MODEL FOR PERSONALIZED PROGRAMMING EDUCATI

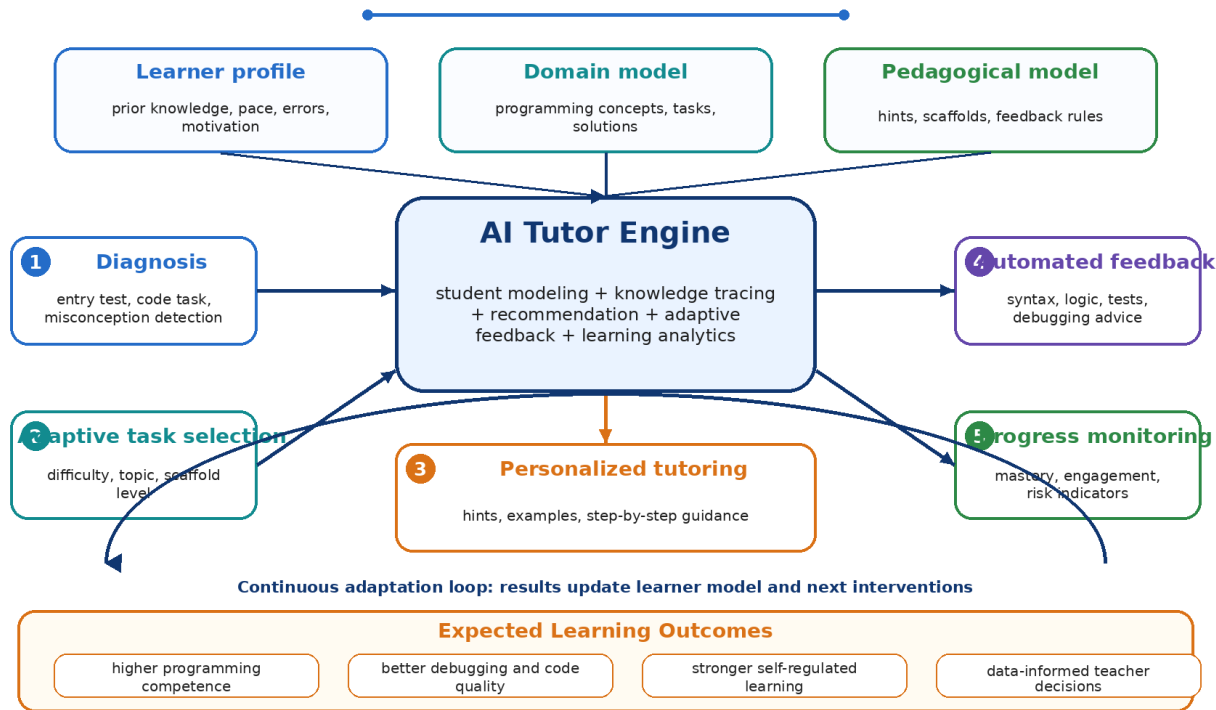


Figure 1. Intelligent tutoring system model for personalized programming education

Table 2. Core components of the proposed ITS architecture

ITS component	Main function	Programming-specific data used
Domain model	Represents topics, task types, solution strategies and constraints	Syntax rules, algorithms, test cases, expected outputs, style criteria
Student model	Maintains the learner	Diagnostic scores,

	profile and mastery estimates	submissions, errors, time on task, hints used
Pedagogical model	Selects next task, hint level and learning path	Mastery level, error history, difficulty progression
Assessment model	Evaluates code and problem-solving processes	Compilation, unit tests, algorithm trace, code quality indicators
Feedback model	Generates individualized formative responses	Error type, misconception pattern, debugging strategy
Teacher dashboard	Supports data-informed teaching decisions	Group risk map, progress trends, weak topic analysis

2.4. Programming competence indicators

The ITS evaluates programming competence as a multidimensional construct. The goal is not to reduce competence to a single test score, but to capture the learner's

progress across conceptual, algorithmic, practical, debugging and reflective dimensions. Each component is assigned observable indicators that can be measured through tests, coding tasks, logs and reflective prompts.

Table 3. Programming competence indicators used in the ITS

Competence component	Observable indicators	Assessment evidence
Theoretical knowledge	Correct use of concepts, operators, data types and control structures	Concept test, code explanation tasks
Algorithmic thinking	Problem decomposition, sequence construction, condition/loop logic	Pseudocode, flowchart, trace tasks
Code-writing quality	Correctness, readability, modularity and efficiency	Code submissions, unit tests, style checks
Debugging competence	Identifying, explaining and correcting syntax, runtime and logic errors	Debugging tasks, revision logs
Reflective self-regulation	Analyzing one's solution, using feedback, planning next learning steps	Reflection prompts, dashboard use, improvement history

2.5. Personalized tutoring algorithm

1. Step 1 - diagnosis: the learner completes a concept test, algorithm task and coding task.
2. Step 2 - student modeling: the system creates a learner profile including mastery, misconceptions, pace and preferred support level.
3. Step 3 - task recommendation: the system selects the next task according to mastery level and target competence component.

4. Step 4 - tutoring interaction: the learner attempts the task and may request hints or examples.
5. Step 5 - automated assessment: the system analyzes correctness, code style, test results and error patterns.
6. Step 6 - adaptive feedback: the system provides a hint, explanation, debugging suggestion or enrichment task.
7. Step 7 - path updating: the student model and learning pathway are updated based on the latest evidence.
8. Step 8 - teacher intervention: dashboard indicators

notify the instructor when a learner needs human support.

2.6. Instruments and assessment rubric

The evaluation plan includes a diagnostic test, programming laboratory tasks, automated code analysis, student activity

logs, a programming competence rubric and a satisfaction questionnaire. The rubric uses three levels: low, medium and high. The system also stores process data such as number of attempts, error types, hints used, time on task and improvement between attempts.

Table 4. Competence level rubric used for overall programming achievement

Score range	Level	Interpretation
0-50	Low	Student requires intensive guidance; concepts and solution strategies are unstable
51-75	Medium	Student solves standard tasks but needs support in unfamiliar or complex problems
76-100	High	Student independently analyzes, codes, debugs and reflects on solutions

Table 5. Data sources used in the proposed ITS evaluation

Data source	Data collected	Use in personalization
Diagnostic test	Concept mastery and initial misconceptions	Initial grouping and task difficulty selection
Coding submission	Code, errors, test results, style indicators	Automated feedback and competence update
Learning log	Time, attempts, hint requests, revisions	Engagement and self-regulation analysis
Reflection prompt	Student explanation of solution and error correction	Reflective competence assessment
Teacher observation	Qualitative notes on persistence and collaboration	Contextual interpretation of analytics

3. RESULTS

3.1. Initial diagnostic profile

The initial diagnostic phase is designed to identify students' readiness for personalized tutoring. In the model-based sample, most learners are located in the low and medium

competence levels, indicating a need for structured tutoring and adaptive feedback. The diagnostic profile also reveals that code-writing and debugging are weaker than theoretical knowledge, a pattern frequently observed in programming courses where students can recognize concepts but cannot always apply them independently.

Table 6. Initial diagnostic distribution of programming competence

Competence level	Control group initial (%)	Experimental group initial	Interpretation
------------------	---------------------------	----------------------------	----------------

		(%)	
High	17	18	Students can solve non-routine programming tasks with limited support
Medium	46	45	Students can complete standard tasks but need scaffolding
Low	37	37	Students have unstable foundations and require guided practice

3.2. ITS-supported learning outcomes

After the implementation of the ITS-supported personalized learning pathway, the experimental group shows greater improvement than the control group in the model-based

sample. The largest change is observed in the reduction of low-level learners and the increase in high-level learners. This suggests that adaptive task sequencing and individualized feedback may help learners progress from basic task completion to more independent problem solving.

Table 7. Experimental group changes after ITS-supported instruction

Level	Experimental initial (%)	Experimental final (%)	Change
High	18	39	+21
Medium	45	48	+3
Low	37	13	-24

3.3. Component-based competence development

The component-level results show that the strongest growth appears in debugging competence and reflective self-regulation. This is pedagogically important because

personalized programming education should not only increase correct answers but also improve how learners diagnose and repair mistakes. The ITS therefore functions as a formative environment in which repeated attempts, hints and explanations help students learn from errors.

Table 8. Component-based growth in the experimental group

Component	Initial score (%)	Final score (%)	Growth
Theoretical programming knowledge	54	77	+23
Algorithmic thinking	47	76	+29
Code-writing quality	44	75	+31
Debugging competence	40	73	+33
Reflective self-regulation	38	70	+32

3.4. Control-experimental comparison

The control and experimental groups both improve, but the experimental group demonstrates a larger average increase. In the control group, growth can be attributed to normal

instruction and practice. In the experimental group, the greater gain is associated with diagnostic personalization, adaptive task selection, automated feedback and progress monitoring.

Table 9. Comparison of control and experimental group mean scores

Group	Initial mean (%)	Final mean (%)	Mean gain
Control group	46	60	+14

Experimental group	45	75	+30
--------------------	----	----	-----

3.5. Statistical analysis

To determine whether the difference between groups is statistically meaningful, a Student t-test can be applied to final competence scores. In the model-based sample, the

calculated t value exceeds the critical threshold and the p-value is below 0.05, indicating a statistically significant difference between the groups. Before submission to a journal, these sample indicators must be replaced by actual experimental data from the institution.

Table 10. Statistical summary of the model-based experiment

Statistical indicator	Value	Interpretation
t-calculated	3.34	Difference between groups is meaningful in the sample model
p-value	0.001	$p < 0.05$, statistically significant
Effect interpretation	Moderate to strong	ITS-supported personalization shows practical value
Data status	Model-based sample	Replace with real experiment results before submission

3.6. Teacher-facing analytics output

A distinctive result of the proposed design is the teacher-facing analytics layer. The dashboard does not replace the instructor; instead, it identifies which students are stuck,

which topics generate the most errors and which learners need enrichment. This helps teachers plan micro-interventions, group remediation and individualized consultation.

Table 11. Teacher-facing analytics indicators and instructional actions

Analytics indicator	Meaning	Teacher action
High error frequency in loops	Students misunderstand iteration logic	Provide mini-lesson and visual tracing task
Many hint requests	Task may be too difficult or explanation insufficient	Lower scaffold level or give worked example
Low time on task + low score	Possible disengagement or guessing	Contact student and assign guided practice
High score + low challenge	Student may need enrichment	Recommend advanced coding challenge
Repeated same error	Misconception is persistent	Provide targeted feedback and one-to-one support

4. DISCUSSION

4.1. Interpretation of findings

The proposed model demonstrates how intelligent tutoring systems can be designed for personalized programming education by connecting AI components with pedagogical functions. The strongest contribution of the model is that it views programming learning as a process rather than a final output. A program that fails a test case is not only a wrong

answer; it is evidence of the learner's current reasoning, misconception and strategy. Therefore, each code submission becomes a source of learning analytics that can guide the next tutoring action.

The component-level results support the claim that an ITS should not focus only on automated grading. Automated grading can tell whether a solution passes tests, but intelligent tutoring requires diagnostic feedback and adaptive scaffolding. In the proposed design, feedback is linked to error

type and competence component. For example, a syntax error receives a different response from a logic error in loop conditions, and a poor algorithmic strategy receives a different intervention from a formatting problem.

The teacher-facing dashboard is also crucial. Some AI systems are presented as fully autonomous tutors, but programming education requires human pedagogical judgment, especially when students are anxious, disengaged or repeatedly unsuccessful. The proposed ITS supports the teacher by providing structured evidence, while the teacher remains responsible for motivation, contextual explanation and ethical decision-making.

4.2. Comparison with previous research

The results are consistent with Fodouop Kouam (2024), who emphasized that ITS should be optimized for learners with different programming backgrounds. The proposed model responds to this requirement by classifying learners through diagnosis and then adapting task difficulty, hint level and feedback type. It also aligns with Lai et al. (2025), who highlight the value of personalized real-time feedback in coding and learning environments.

The model is also consistent with broader systematic reviews of ITS and AI-supported personalized learning. Letourneau et al. (2025) report generally positive but variable effects of ITS, suggesting that design quality matters. du Plooy et al. (2024) describe personalized adaptive learning as promising in higher education but dependent on careful implementation. These conclusions support the present article's emphasis on pedagogical design, not only technical automation.

In relation to automated feedback research, the article extends the logic of Keuning et al. (2018) by integrating feedback into a larger tutoring architecture. Feedback is not treated as a separate tool but as a component connected to the student model, domain model and teacher analytics. This integration is important for personalized programming education because a hint should depend on the learner's current level, prior errors and learning trajectory.

4.3. Practical implications

For universities, the model provides a practical blueprint for developing ITS-supported programming courses. The first implication is that ITS implementation should begin with competence indicators and diagnostic tasks, not with technology alone. Institutions need to define what programming competence means in their curriculum and how each component can be measured.

The second implication is that instructors should be trained to interpret learning analytics. A dashboard is useful only when teachers can translate indicators into pedagogical action. Therefore, professional development should include data-informed teaching, formative feedback strategies and ethical use of AI in education.

The third implication is that ITS should be implemented gradually. A realistic first step is to introduce automated feedback for selected programming topics, then add student modeling, adaptive task selection and teacher analytics. This reduces technical risk and allows instructors to evaluate the pedagogical value of each component.

4.4. Limitations

- The statistical values in this article are model-based sample indicators and must be replaced by real experimental data before journal submission.
- Automated feedback quality depends on the quality of test cases, error classification and the pedagogical clarity of hints.
- Student modeling may be inaccurate if the system uses insufficient data or if students work outside the platform.
- Privacy, data security and algorithmic fairness must be considered when collecting learner data.
- The system requires stable technical infrastructure, teacher training and continuous updating of programming tasks.

4.5. Ethical and implementation considerations

Table 12. Ethical and implementation considerations for ITS design

Risk	Potential problem	Mitigation strategy
Privacy risk	Learner logs and code submissions contain sensitive learning data	Use consent, anonymization and secure storage
Algorithmic bias	Recommendations may favor	Audit outcomes by

	certain learner profiles	subgroup and allow teacher override
Over-automation	Students may rely on hints without thinking independently	Use graduated hints and reflection prompts
Teacher deskilling	Instructors may over-trust dashboard indicators	Provide analytics literacy and pedagogical training
Limited transparency	Students may not understand why tasks are recommended	Explain recommendation logic in learner-friendly language

5. CONCLUSION

This article designed and justified an intelligent tutoring system model for personalized programming education. The proposed model integrates learner diagnosis, student modeling, domain representation, adaptive task recommendation, automated formative feedback and teacher-facing analytics. It addresses the central challenge of programming education: students differ in prior knowledge, learning speed, misconceptions and support needs, while traditional instruction often provides the same task sequence to all learners.

The model-based results indicate that ITS-supported personalization can improve programming competence across theoretical knowledge, algorithmic thinking, code-writing quality, debugging competence and reflective self-regulation. The strongest value of the ITS lies in its capacity to transform programming errors into learning opportunities through timely, actionable and adaptive feedback.

The scientific contribution of the article is the programming-specific integration of ITS architecture and competence-based assessment. The practical contribution is a design blueprint that universities can use when planning AI-supported programming courses. Future research should validate the model with real students, compare different feedback strategies, develop fairness-aware student modeling, and examine the long-term effects of ITS-supported learning on programming independence and professional readiness.

REFERENCES

1. Crow, T., Luxton-Reilly, A., & Wuensche, B. (2018). Intelligent tutoring systems for programming education: A systematic review. *Proceedings of the Australasian Computing Education Conference*, 53-62. <https://doi.org/10.1145/3160489.3160492>
2. du Plooy, E., Casteleijn, D., & Franzsen, D. (2024). Personalized adaptive learning in higher education: A scoping review of key characteristics and impact on academic performance and engagement. *Heliyon*, 10, e39630. <https://doi.org/10.1016/j.heliyon.2024.e39630>
3. Fodouop Kouam, A. W. (2024). The effectiveness of intelligent tutoring systems in supporting students with varying levels of programming experience. *Discover Education*, 3, 278. <https://doi.org/10.1007/s44217-024-00385-3>
4. Fortuna, A., Prasetya, F., Samala, A. D., Rawas, S., Criollo-C, S., Kaya, D., Raihan, M., Andriani, W., Safitri, D., & Nabawi, R. A. (2025). Artificial intelligence in personalized learning: A global systematic review of current advancements and shaping future opportunities. *Social Sciences & Humanities Open*, 102114. <https://doi.org/10.1016/j.ssaho.2025.102114>
5. Farhood, H., et al. (2025). Artificial intelligence-based personalised learning in education: A systematic literature review. *Discover Artificial Intelligence*. <https://doi.org/10.1007/s44163-025-00598-x>
6. Ifenthaler, D., & Yau, J. Y. K. (2020). Utilising learning analytics to support study success in higher education: A systematic review. *Educational Technology Research and Development*, 68, 1961-1990. <https://doi.org/10.1007/s11423-020-09788-z>
7. Kayumov Oybek Achilovich, Data-driven yondashuv asosida talabalar kompetentligini baholash metodikasi, Ta'limda istiqbolli izlanishlar - xalqaro ilmiy-metodik jurnal, № 4/2026 ISSN: 3060-527X, 55-63 b.
8. Kayumov Oybek Achilovich, Kompetentligini baholash mezonlari va indikatorlarini ishlab chiqish, "Maktabgacha va maktab ta'limi" jurnali, ISSN: 3060-4613, 2026-yil,

aprel, 4(1)-son, 203-208 b.

9. Kayumov Oybek Achilovich, Web-loyihalash kompetentligini shakllantirishda interaktiv va loyihaviy (PBL) metodlardan foydalanish, Zamonaviy jamiyat va innovatsiyalar ilmiy-nazariy jurnal, ISSN: 3060-544x, № 4-son, 2026-yil, 63-69 b.
10. Kayumov Oybek Achilovich, Evaluating the Effectiveness of Adaptive Learning Systems in Developing Web Design Competence: An Empirical Study, International journal of pedagogics, ISSN: 2771-2281, Volume 06 Issue 03 (March) 2026, 362-367 b.
11. Kayumov Oybek Achilovich, Human–Ai Collaboration in Adaptive Learning Systems: A New Paradigm for Competence Development, American journal of social sciences and humanity research, ISSN: 2771-2141, Volume 06 Issue 03 (March) 2026, 126-131 b.
12. Kayumov Oybek Achilovich, Data-Driven Approaches to Enhancing Students' Web Design Skills Using Multimodal Learning Analytics, Current research journal of pedagogics, ISSN: 2767-3278, Vol.07 Issue03 2026, 46-52 b.
13. Kayumov Oybek Achilovich, AI Capital As A Predictor of Students' Digital and Web Design Competence in Higher Education, Journal of Social Sciences and Humanities Research Fundamentals, ISSN: - 2748-9345, Volume 06 Issue 03 (March) 2026, 94-99 b.
14. Kayumov Oybek Achilovich, A Conceptual Framework for Developing Web Design Competence Based on Multimodal Learning Analytics in Adaptive Digital Environments, European International Journal Of Pedagogics, ISSN: 2751-000X, Volume 06 Issue 03 (MARCH) 2026, 155-161 b.
15. Keuning, H., Jeuring, J., & Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. ACM Transactions on Computing Education, 19(1), Article 3. <https://doi.org/10.1145/3231711>
16. Lai, C. H., et al. (2025). Analysis of learning behaviors and outcomes for students with different knowledge levels: A case study of Intelligent Tutoring System for Coding and Learning (ITS-CAL). Applied Sciences, 15(4), 1922. <https://doi.org/10.3390/app15041922>
17. Letourneau, A., et al. (2025). A systematic review of AI-driven intelligent tutoring systems (ITS) in K-12 education. npj Science of Learning, 10, Article 37. <https://doi.org/10.1038/s41539-025-00320-7>
18. Messer, M., Brown, N. C. C., Kölling, M., & Shi, M. (2024). Automated grading and feedback tools for programming education: A systematic review. ACM Transactions on Computing Education, 24(1), 1-43.
19. Sharma, P., et al. (2022). Designing an intelligent tutoring system for computer programming in the Pacific. Education and Information Technologies, 27, 6197-6221. <https://doi.org/10.1007/s10639-021-10882-9>
20. Zawacki-Richter, O., Marin, V. I., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education. International Journal of Educational Technology in Higher Education, 16, 39. <https://doi.org/10.1186/s41239-019-0171-0>