AI-POWERED PERSONALIZED LEARNING ASSISTANT FOR FRONTEND DEVELOPMENT EDUCATION

Khalilova Sarvinoz
Student, Japan Digital University, Tashkent, Uzbekistan
Academic Supervisor: Lochinbek Boboev, PhD
Head of the IT Department, Japan Digital University, Tashkent, Uzbekistan

ABSTRACT

This article investigates an AI-powered personalized learning assistant model designed for students studying frontend development (HTML, CSS, JavaScript, React) at higher education institutions in Uzbekistan. The relevance of the study stems from the fact that students face considerable difficulty in independently identifying syntactic, logical, and asynchronous errors in their code, as well as errors related to responsive design and the prop/state relationships between React components, which reduces the overall effectiveness of practical learning. To address this problem, the article proposes an "AI Assistant" system that analyzes student code in real time and, instead of providing a ready-made solution, offers personalized, step-by-step guidance on how to correct the identified error. The study describes the system's four-layer architecture, a three-stage experimental process, and a set of performance indicators. The expected results indicate that the proposed model reduces debugging time, increases independent error-correction ability, and deepens students' understanding of component-based architecture.

Keywords: *Frontend Education, React, AI Assistant, Personalized Learning, Debugging, Component Architecture.*

I. INTRODUCTION

In the context of the modern digital economy, frontend development technologies — HTML, CSS, JavaScript, and React — constitute the core toolkit for building web applications and occupy a central place in computer science

curricula at higher education institutions. However, the rapid evolution and multi-layered nature of frontend technologies present significant pedagogical challenges for students. Research has shown that students learning web development for the first time struggle to understand computational concepts such as notation, hierarchy, paths, nesting, parameters, and decomposition when writing HTML, CSS, and JavaScript code [1].

In particular, the results of a large-scale analysis indicate that in introductory web development courses, students' help-seeking behavior is primarily related to development, technology, content, and design issues, with HTML and CSS syntax errors constituting one of the most frequently encountered difficulties [2]. Similarly, a systematic analysis of HTML and CSS syntax errors shows that students commonly encounter recurring error types such as unclosed tags, incorrectly nested elements, and inconsistencies in CSS selectors, underscoring the need for dedicated pedagogical support tools [3].

When learning component-based libraries such as React, students face a new level of difficulty — namely, understanding the distinction between the concepts of “props” and “state,” as well as the unidirectional nature of data flow. Practical observation and teaching experience confirm that, for beginner-level students, the most challenging concept is precisely the distinction between props and state and how they are passed through the component hierarchy [4]. In addition, the “prop drilling” problem that arises when passing data between components, and the incorrect use of the “key” attribute when rendering list elements, represent error categories that students frequently encounter but find difficult to diagnose independently [5].

The potential of artificial intelligence, particularly large language models (LLMs), for addressing these problems has been actively explored in recent research. For example, a large-scale study evaluating the debugging capabilities of open-source LLMs found a significant gap between these models' coding ability and their debugging ability — that is, a model's ability to write code does not

guarantee that it can understand and fix an error [6]. At the same time, it has been demonstrated that a language model can be trained to independently identify its own errors through a “Self-Debugging” approach, using code execution results and natural-language explanations, confirming that a pedagogical strategy similar to “rubber duck debugging” can also be effective within AI systems [7]. Another study proposed the “Automated Scientific Debugging” (AutoSD) model, which requires an LLM to proceed through the stages of hypothesis formulation, verification, and conclusion, thereby providing the developer (or student) with clear, well-grounded explanations [8].

Research on the pedagogical application of artificial intelligence in the context of programming education shows that, instead of providing ready-made code, the “scaffolding” principle can be implemented by offering students step-by-step hints and guiding questions, which preserves the student's independent reasoning activity [9; 10]. Furthermore, interactive chatbot systems developed using LLMs in the learning process have been shown to have a positive effect on the development of students' computational thinking [11].

In the higher education system of Uzbekistan, the integration of artificial intelligence technologies into the educational process has also been identified as an important task at the level of state policy. The Strategy of the Republic of Uzbekistan for the Development of Artificial Intelligence Technologies until 2030 designates digital education and workforce training as priority areas [12]. At the same time, the results of a nationwide survey indicate a generally positive attitude toward artificial intelligence within higher education institutions, although significant gaps remain in terms of digital competencies and infrastructure [13]. It is further noted that a growing gap exists between institutional policy on the use of generative artificial intelligence in distance education and students' actual practical use of such tools [14].

The above analysis indicates that the primary difficulty faced by students in frontend development education is not writing code itself, but independently

identifying and understanding errors within it. The aim of this study is therefore to develop an “AI Assistant” pedagogical model that analyzes HTML, CSS, and React code in real time and provides students with logical hints and direction rather than ready-made answers, and to scientifically substantiate its applicability within the context of higher education in Uzbekistan.

II. MAIN PART

2.1. Practical Analysis of the Problem

A key problem observed in teaching frontend development courses is that, although students write code, they often fail to correctly interpret the error message displayed in the browser console (for example, “Cannot read property of undefined” or “Warning: Each child in a list should have a unique key prop”). In such situations, students frequently copy the error message into ChatGPT and receive a ready-made corrected version of the code, without understanding the actual root cause of the error — for instance, the component's re-render cycle or the timing of an asynchronous request.

Furthermore, when implementing responsive design (media queries, flexbox, and grid systems), students often observe the visual outcome but are unable to analyze why a particular CSS rule fails to work at a given screen size. This constitutes an implicit problem for students, requiring deep knowledge of CSS syntax and cascading specificity rules [3].

2.2. Architecture of the “AI Assistant” System

To address this problem, this study proposes an “AI Assistant” system with a four-layer architecture. The system's core principle is that it analyzes a student's HTML, CSS, or React code in real time but never provides fully complete, ready-to-run code. The structure of the system architecture is presented in Table 1.

Table 1. Architecture of the “AI Assistant” system

Layer	Function	Technical Description
1. Code Analysis Layer	Reads HTML, CSS, and JavaScript/React code in real time	Errors are identified at the Abstract Syntax Tree (AST) level from the code editor (Monaco / VS Code extension) through syntax analysis, ESLint rules, and CSS validation.
2. Error Classification Layer	Categorizes the detected error	Errors are divided into four types: syntactic (e.g., unclosed tag), logical (incorrect condition), asynchronous (Promise/async-await sequencing issues), and component-level (prop/state) errors.
3. Pedagogical Dialogue Layer (LLM)	Provides hints rather than ready-made answers	Based on the error context, a large language model (LLM) formulates a Socratic question or guiding hint, e.g., “Check the dependency array inside your useEffect.”
4. Adaptive Profile Layer	Tracks the student’s knowledge level	An individual error history is maintained for each student, and the complexity of the hints is adapted accordingly (personalization).

The third layer of this architecture — the pedagogical dialogue layer — represents the most significant innovative aspect of the model. Here, the large language model operates in a manner similar to the “Self-Debugging” approach: it analyzes the execution outcome of the error and explains it in natural language, without providing a direct answer [7]. This approach also aligns with the

hypothesis-verification-conclusion structure applied in the AutoSD model, whereby the system progressively substantiates why the error occurred [8].

2.3. Three-Stage Experimental Process

A three-stage experimental process has been designed to evaluate the effectiveness of the proposed "AI Assistant" model, as shown in Table 2.

Table 2. Three-stage experimental process

Stage	Name	Content of Activity	Expected Outcome
1	Initial diagnostics	Students complete a React-component or responsive-page creation task without AI assistance; the number and type of errors are recorded.	An initial debugging profile is established for each student.
2	AI Assistant-guided practice	When a student makes an error, the system does not provide corrected code but only a hint regarding the location and cause of the error; the student corrects the code independently.	Students develop the ability to independently identify and correct errors.
3	Final assessment	A new, more complex responsive/React task is completed without AI assistance; the results are compared with Stage 1.	Improvements in debugging speed, error count, and independent problem-solving are measured.

This three-stage structure corresponds to the quasi-experimental designs applied in previous studies, in which experimental and control groups are measured separately and their results compared [9]. Such an approach makes it possible to objectively evaluate not only a student's ability to write code but also their ability to independently analyze and correct it.

2.4. Expected Performance Indicators

The expected differences between the traditional teaching method (a control group working independently or relying solely on tools that provide ready-made code) and the experimental group using the “AI Assistant” model are summarized in Table 3.

Table 3. Expected performance comparison between the control and experimental groups

Indicator	**Traditional Method (Control Group)**	**With AI Assistant (Experimental Group)**
Average error-identification time	High (long)	Significantly reduced
Share of independently corrected errors	Low	High
Level of understanding of prop/state errors	Limited	Expanded (cause-and-effect understanding)
Student motivation and self-confidence	Moderate	High
Critical code-analysis skills	Declining trend	Steady growth

The indicators presented in the table show that the “AI Assistant” model transforms students from passive consumers of code into active problem-solvers.

This, in turn, offers an effective response to concerns that obtaining ready-made answers from LLMs may negatively affect critical thinking and debugging skills [6; 9]. The positive impact of LLM-based interactive systems that foster computational thinking on learning outcomes has also been confirmed in the existing literature [11].

2.5. Practical Recommendations for the Context of Higher Education in Uzbekistan

Implementing the “AI Assistant” system within frontend-related programs at higher education institutions in Uzbekistan requires several conditions. First, faculty members must undergo specialized training in React, component-based architecture, and LLM-based pedagogical tools, since, according to national survey results, the current level of digital competency remains insufficient [13]. Second, it is advisable to introduce the system into practical coursework gradually, beginning with pilot projects before extending its application to broader curricula. Third, clear methodological guidelines for using the system must be developed, which will help narrow the gap between institutional policy and students' actual practical use of such tools [14].

III. CONCLUSION

This study examined the theoretical foundations of an artificial-intelligence-based personalized learning assistant — the “AI Assistant” model — for frontend development education (HTML, CSS, JavaScript, React). The findings indicate that students' primary difficulty lies not in writing code itself but in independently identifying and understanding the syntactic, logical, asynchronous, and component-level (prop/state) errors within it. Using artificial intelligence without oversight, solely to obtain ready-made code, may hinder the development of these skills [6].

The proposed four-layer “AI Assistant” architecture — comprising code analysis, error classification, pedagogical dialogue, and adaptive profiling layers — addresses this problem by guiding students through logical hints rather than providing ready-made answers. The three-stage experimental process (diagnostics, guided practice, and final assessment) provides a foundation for objectively evaluating the model's practical applicability and effectiveness.

For future research, it is recommended that the “AI Assistant” model be tested under real conditions at higher education institutions in Uzbekistan, that its effectiveness be quantitatively measured across various frontend technologies (Vue, Angular, TypeScript), and that student and instructor attitudes toward the system be studied in greater depth. In conclusion, artificial intelligence in frontend education should not be integrated as a substitute for the student, but as a “digital mentor” that strengthens the student's independent debugging and critical-thinking abilities.

REFERENCES

1. Park, T., Wiedenbeck, S. (2011). Learning web development: Challenges at an earlier stage of computing education. Proceedings of the Seventh International Workshop on Computing Education Research (ICER '11), ACM. https://dl.acm.org/doi/10.1145/2016911.2016937
2. Park, T., Wiedenbeck, S. (2011). Learning web development: Challenges at an earlier stage of computing education (PDF version). ResearchGate. https://www.researchgate.net/publication/221209607_Learning_web_development_Challenges_at_an_earlier_stage_of_computing_education
3. Park, T., Dorn, B., Forte, A. (2015). An Analysis of HTML and CSS Syntax Errors in a Web Development Course. ACM Transactions on Computing Education, 15(1), 1–21. https://dl.acm.org/doi/10.1145/2700510

4. A Case for Learning Functional Components First: Teaching Students Props and State in React. Codementor. https://www.codementor.io/@alyssahope/a-case-for-learning-functional-components-first-njix8bc9t
5. How to Learn the Hard Parts of React — and Tips to Conquer Them. freeCodeCamp, 2024. https://www.freecodecamp.org/news/hard-parts-of-react/
6. Majdoub, Y. et al. (2024). Debugging with Open-Source Large Language Models: An Evaluation. Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24). arXiv preprint. https://arxiv.org/html/2409.03031v1
7. Chen, X., Lin, M. et al. (2023). Teaching Large Language Models to Self-Debug. arXiv preprint arXiv:2304.05128. https://arxiv.org/abs/2304.05128
8. Kang, S. et al. (2024). Explainable Automated Debugging via Large Language Model-Driven Scientific Debugging (AutoSD). Empirical Software Engineering, Springer. https://link.springer.com/article/10.1007/s10664-024-10594-x
9. Closing the Loop: An Instructor-in-the-Loop AI Assistance System for Supporting Student Help-Seeking in Programming Education. arXiv preprint, 2025. https://arxiv.org/pdf/2510.14457
10. CodeExemplar: Example-Based Scaffolding for Introductory Programming in the GenAI Era. arXiv preprint, 2026. https://arxiv.org/pdf/2603.23830
11. MazeMate: An LLM-Powered Chatbot to Support Computational Thinking in Gamified Programming Learning. arXiv preprint, 2025. https://arxiv.org/pdf/2511.03727
12. Uzbekistan's strategy for the development of artificial intelligence technologies until 2030. Digital Watch Observatory, 2025. https://dig.watch/resource/uzbekinstans-strategy-for-the-development-of-artificial-intelligence-technologies-until-2030

13. The Future of Higher Education in Uzbekistan: The Potential and Challenges of Integrating Artificial Intelligence (National Survey). Perspectives of Higher Education Development, 2025. https://erasmus.uz/journal/index.php/files/article/view/129
14. Generative AI Governance in Distance Higher Education: Aligning Uzbekistan's Digital Learning Methodology with International Standards. 2026. https://www.pstjournal.uz/index.php/pst/article/view/125
15. Scaffolding Metacognition in Programming Education: Understanding Student-AI Interactions and Design Implications. arXiv preprint, 2025. https://arxiv.org/pdf/2511.04144