International Journal of Computer Applications (0975 – 8887)
Volume 187 – No.71, January 2026

EZ Coder: A Hybrid AI-Powered Mentorship Framework for Integrated Developer Education

Arun K.H.
Assistant Professor
Dept. of Information Science and Eng.
Acharya Institute of Technology
Bengaluru, India

Rakshith Gowda M.
Dept. of Information Science and Eng.
Acharya Institute of Technology
Bengaluru, India

Thushar Raj S.G.
Dept. of Information Science and Eng.
Acharya Institute of Technology
Bengaluru, India

Vishal M. Bharadwaj
Dept. of Information Science and Eng.
Acharya Institute of Technology Bengaluru, India

Vishnu M.T.
Dept. of Information Science and Eng.
Acharya Institute of Technology Bengaluru, India

ABSTRACT

IDEs (Integrated Development Environments) have become efficient systems which support team work and debugging in coding. However, present IDEs do not help in understanding the core concepts over time and majorly focus on particular tasks. As a result of this, programmers, specifically the ones who are at a beginner or an intermediate level tend to have a messy learning experience, limited grasp over core concepts of programming and get distracted very often. This paper introduces EZ Coder, a hybrid AI-powered mentorship framework designed as a VS Code extension. EZ Coder changes the IDE into an interactive learning environment. Unlike the typical AI assistants that help as passive tools, EZ Coder works as an active mentor by adding three major features into the IDE directly. These features include a personalised roadmap generator which works as an adaptive learning engine, an in-editor code visualizer to increase the understanding of the programs and an AI chatbot which gives context-aware feedback based on Abstract Syntax Tree (AST) analysis of code structure. An AI inference system is being used by the framework. Cloud-based large language models are combined with fast local models for real-time feedback generation and teaching insights. An ongoing, evidencebased learning cycle tracks the programmer's behavior and updates skill levels using Bayesian reasoning. This cycle prioritizes relevant learning actions without disturbing the workflow. Evaluation of a prototype with programming tasks show that EZ Coder provides extremely accurate and relevant feedback, reduces task completion time and offers much more meaningful guidance than general AI assistants and standard linters.

General Terms

Artificial Intelligence, Software Engineering, Programming Education, Human-Computer Interaction, Intelligent Systems

Keywords

IDE-Integrated learning, Abstract Syntax Tree (AST), Adaptive Learning, AI-Powered Mentorship, Code Visualization, Hybrid AI Inference

1 INTRODUCTION

Integrated Development Environments (IDEs) have become complex platforms, fully supporting activities like debugging and version control. Nevertheless, most contemporary IDEs remain heavily focused on tasks associated with code execution and troubleshooting. Notably, this has left a pedagogical divide between code authoring and code-related concepts in programming, especially in programmers at a beginner and intermediate skill-levels. Programming is a cyclic process centered around concepts. Unfortunately, most coding processes nowadays involve programmers halting a related activity to search for answers in various sources, resulting in a shallow learning attitude related to a solution or a subject.

At the same time, AI-powered developer assistants, such as smart code completion tools or large language model-driven chat interfaces, have further increased code-writing speed by minimizing syntactical mistakes in code. Nevertheless, these tools have been found to largely operate as passive helper tools, providing results to the users while not involving them in problem-solving or logical process of code implementation. Although promising studies are conducted in the areas of intelligent tutoring systems, adaptive learning systems and code visualizations have been seen to aid in better conceptualization, most of these tools currently exist as separate systems that are not well-integrated into current IDE-centric developer workflows.

To address these challenges, we introduce EZ Coder, a hybrid AI-based mentoring system that combines coding, learning and conceptual understanding directly in an integrated development environment (IDE) via VS Code extension. Contrary to traditional assistants, EZ Coder acts as an active mentoring system using intelligent feedback that combines ASTbased feedback analysis, adaptive learning pathways modeled from continuous behavioral analysis and code visualization analysis directly in the coding environment to help programmers understand program flow. A hybrid inference architecture combines local low-latency inference and cloud-based inference with selective large language models for detailed conceptual learning analysis. The experimental results bring out that incorporating mentoring directly into the coding process can improve feedback accuracy and reduce cognitive overload and switching activity while enhancing productivity directly in the development process.

2 PROBLEM STATEMENT

Even though significant advancements have been made in Integrated Development Environments (IDEs) and AI assisted programming tools, a fundamental gap still exists between software development and conceptual learning. Current development workflows focus on task completion and offer limited understanding of why certain solutions work or how programming concepts should be applied across diverse contexts. This poses a significant challenge for novice and intermediate developers, who are still forming foundational mental models of programming concepts.

A major challenge arises as developers frequently perform context switching to consult external resources such as tutorials, video lectures and discussion forums which disrupts active coding sessions. This repeated context switching increases cognitive load, disrupts problem solving continuity and weakens the association between theory and practice. As a result, learning becomes reactive and solution driven rather than systematic and concept oriented.

Moreover, the existing AI based coding assistants are not designed to adapt their explanations and code complexity based on learner proficiency. While they provide syntactically correct suggestions and rapid fixes, they lack personalization and fail to identify recurring conceptual gaps.

Another limitation is the absence of active mentorship within professional development environments. Most tools function as passive assistants and rarely guide learners towards structured skill development or visualize code structures that supports deeper comprehension of control flow and program logic.

3 LITERATURE SURVEY

Research on improving programming education and developer productivity has enhanced in many related areas, including intelligent tutoring systems, AIassisted coding tools, Code visualization techniques, and adaptive learning platforms. All of these methods face particular challenges in leaning to program. However, they also have mostly developed separately, leading to scattered solutions that do not directly weave together educational support with professional development workflows.

3.1 Intelligent Tutoring Systems in Programming Education

Most of the early works done in programming education mainly focused on Intelligent Tutoring Systems (ITS), which focused to mimic one-on-one human tutoring system through rule-based feedback, creating student models, and guided problem solving. These systems demonstrated clear improvements in learning outcomes by adjusting instruction based on student performance. Adaptive learning paths which use probabilistic skill models, have improved concept mastery and retention. This is especially true when Bayesian inference models learner uncertainly and progression [12].

However, ITS platforms usually operate in controlled settings like classrooms or learning portals because they do not integrate so well with many IDEs, hence their usefulness decreases when learners move into real world software development settings.

3.2 Chatbot-Based and AI-Assisted Programming Tools

The development of natural language processing has seen the rise of learning assistants that use chatbots as well as coding tools that use AI. Such tools provide explanations, code generation, as well as debugging help through chatbots. Current literature has identified the challenges as well as the opportunities available through the application of large language models in programming [9], [11].

Code completion tools that rely on AI to aid in programming are especially concerned with speeding up the programming process. Though these tools work very well in minimizing syntactic errors and code repetition, these tools usually aim to maximize plausibility instead of pedagogical order [16]. Thus, students may find the right answers to problems despite not understanding the concept.

3.3 Structural Code Analysis and AST-Based Techniques

In view of the limitations associated with the analysis of plain text at the source code level, various research studies have examined the structure-aware representation of code using the paradigm of Abstract Syntax Trees (ASTs) in computer programming. Based on ASTs, it becomes easier to understand control flow and function boundaries in code. Code representation Learning Survey studies have highlighted ASTs as a fundamental construct for semantic analysis in code analysis [15].

There are also visual tools that rely on AST for improving the understanding of the code structure and flow. These visual tools and techniques that depend on AST have been found useful for beginner programmers because they aid in the understanding of the complex programming concepts. Most of the techniques that use AST are only tools that perform only analysis. They do not utilize learning capabilities.

3.4 Adaptive Learning Systems and Personalized Guidance

Adaptive learning systems aim to personalize content by modeling learner behavior and dynamically adjusting instruction. Bayesian knowledge tracing and probabilistic skill modeling are some of the techniques which have proven to be effective in identifying conceptual gaps and prioritizing necessary learning material[12]. However, these systems are implemented within online learning platforms and stay disconnected from active coding environments.

Recent work highlights the need for reducing cognitive load through context aware tool support in software development, arguing that learning interventions must closely align with the developer's current task and environment to be efficient [10].This insight emphasizes the need for adaptive learning mechanisms within the IDE rather than alongside it.

3.5 IDE-Integrated Learning-Oriented Tooling

Recent research has begun exploring IDE based program analysis tools specifically designed for learning support. Such tools facilitate to bridge the gap between development and education by leveraging in editor context to provide relevant feedback and explanations[14]. Moreover pedagogical discussions stress about the importance of guiding learners toward understanding rather than replacing cognitive effort to automation[13].

Even though these approaches move closer to authentic development workflows, they often lack a unified framework that integrates structural code understanding, adaptive learning logic and AI driven reasoning.

3.6 Research Gap and Motivation

The current state of literature displays visible fragmentation of methods. Chatbot systems also offer accessibility but they lack structural understanding. AST based systems offer structural understanding but do not offer adaptivity. LLM-based systems offer fluent explanation but they lack pedagogical orientation. Adaptive learning systems function independently of real-world development environments. Current systems offer no integrated functionality of these methods within a single framework. The need for a cohesive system centered on mentorship because of this gap is what EZ Coder is poised to fill by using AST code understanding, hybrid inference using AI, and improves a users skills by offering personalized mentorship on a constant, uninterrupted basis.

4 PROPOSED METHODOLOGY

Unlike standalone educational platforms the EZ Coder framework is designed as a continuous, in situ mentorship system that integrates learning and development within the Integrated Development Environment (IDE). EZ Coder operates in real time alongside the developer, forming a closed loop system that continuously observes coding behavior and delivers optimal feedback. At a high level, EZ Coder follows a cyclic operational model in which developer actions trigger analysis, inference and learning adaptation, which results in a feedback that is contextualized to the current coding task.

4.1 System Architecture

EZ Coder adopts a layered architecture and is implemented as an IDE integrated framework composed of four logical layers , as summarized in Table 1.

◦ **Presentation Layer:**

At the interface level, developer interactions such as code edits and cursor movements are captured using the VS Code Extension API. As this layer operates entirely within the IDE, it eliminates the need of external context switching.

◦ **Application Layer:**

This layer establishes communication between the IDE and the backend services. It handles event routing, enforces latency and determines whether feedback should be generated through local inference or escalated to cloud based reasoning.

◦ **Service Layer:**

This layer contains the Code Analysis Engine, the Hybrid AI

Inference Engine , the Adaptive Learning Engine which together form the analytical core of the EZ Coder.

◦ **Data Layer:**

This layer stores user profiles, learning history and configuration settings. Data is primarily stored locally using lightweight storage formats to preserve privacy and offline functionality.

4.2 Core Functional Cycle

EZ coder functions through a four stage functional cycle (see Figure 1) that stays active throughout the developer's coding session. The stages of the cycle are as follows:

◦ **Context Acquisitions and Event Capture:**

The EZ coder Extension continuously monitors developer activities and captures relevant events.

◦ **Structural Code Analysis:**

The captured code is parsed to generate Abstract Syntax Tree (AST). ASTs encode hierarchical and semantic relationships which enables identifying control flow structures.

◦ **Hybrid AI Inference:**

The Hybrid AI Inference Engine handles common patterns and frequently observed issues using locally available lightweight models for low latency feedback. More complex scenarios are escalated to cloud based large language model.

Table 1: System Components and Technologies in EZ Coder

Component	Description
Presentation Layer	**Technology / Mechanism:** VS Code Extension API, WebView **Functional Role:** Captures developer interactions (code edits, cursor events) and renders inline feedback, adaptive learning roadmaps, and code visualizations within the IDE.
Code Analysis Engine	**Technology / Mechanism:** Tree-sitter, AST Parsers **Functional Role:** Transforms source code into Abstract Syntax Trees (ASTs) to enable structural and semantic analysis of control flow, function boundaries, and conceptual constructs.
Local Inference Module	**Technology/Mechanism:** ONNX Runtime (Lightweight Models) **Functional Role:** Performs low-latency, privacy preserving analysis for common errors, style issues, and frequently observed conceptual mistakes in real time.
Cloud Inference Module	**Technology / Mechanism:** Large Language Model (LLM) via Secure API **Functional Role:** Provides deep pedagogical explanations, extended reasoning, and cross-context analysis for complex or low-confidence cases.
Adaptive Learning Engine	**Technology / Mechanism:** Bayesian Inference, Probabilistic Skill Model **Functional Role:** Maintains learner proficiency estimates, identifies conceptual gaps, and dynamically reprioritizes personalized learning roadmaps based on observed behavior.
Data Management Layer	**Technology / Mechanism:** Local Storage (JSON-based Profiles) **Functional Role:** Stores user profiles, probabilistic skill models, learning history, and configuration settings while preserving user privacy and offline functionality.

◦ **Feedback Synthesis and Learning Adaptation**

The inference output is converted into actionable feedback. The Adaptive Learning Engine simultaneously updates the user's skill model based on the observed behavior and completes the feedback loop.

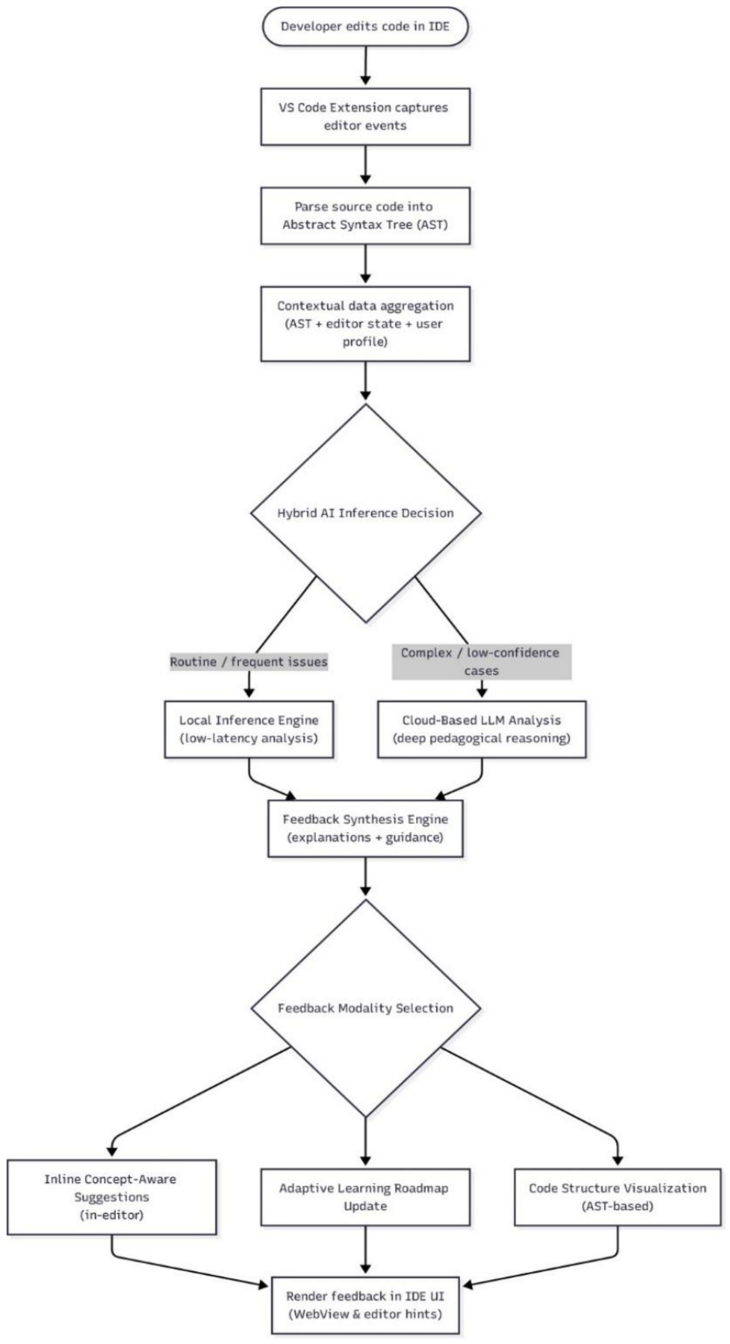
Developer edits code in IDE
VS Code Extension captures editor events
Parse source code into Abstract Syntax Tree (AST)
Contextual data aggregation (AST + editor state + user profile)
Hybrid AI Inference Decision
Routine / frequent issues
Complex / low-confidence cases
Local Inference Engine (low-latency analysis)
Cloud-Based LLM Analysis (deep pedagogical reasoning)
Feedback Synthesis Engine (explanations + guidance)
Feedback Modality Selection
Inline Concept-Aware Suggestions (in-editor)
Adaptive Learning Roadmap Update
Code Structure Visualization (AST-based)
Render feedback in IDE UI (WebView & editor hints)

Figure 1: High-level operational workflow of EZ Coder illustrating IDE event capture, AST-based analysis, hybrid AI inference, and multi-modal feedback delivery.

5 IMPLEMENTATION

The EZ Coder system is implemented as a Visual Studio Code extension using a modular and scalable technology stack. The architecture is designed to support real-time user interaction, AI-assisted code generation, and adaptive learning features while carefully considering privacy, latency, and IDE performance constraints. Each component is optimized to operate efficiently within the Visual Studio Code environment while enabling continuous learning and user feedback.

5.1 IDE Extension and UI Layer

EZ Coder is developed as a Visual Studio Code extension with full TypeScript support. The Visual Studio Code Extension API provides access to editor state, open files, cursor movement, and user input, allowing EZ Coder to integrate seamlessly into the developer's workflow.

All user-facing components—including inline coding assistance, adaptive learning roadmaps, and code structure visualizations—are rendered using VS Code WebView Panels. These panels enable rich, interactive content to be displayed directly inside the IDE without requiring users to switch contexts or leave the editor.

Event listeners and message-passing mechanisms are carefully scoped and optimized to minimize performance overhead. This design ensures responsive interactions even during complex operations such as real-time AI inference and large-scale code analysis.

5.2 Code Parsing and Structural Analysis

The Code Analysis Engine is the central part of EZ Coder's reasoning capability in that it is responsible for performing source code structural analysis based on Abstract Syntax Trees (ASTs). The source code obtained from the IDE is parsed based on the Tree-sitter parsing framework. The framework supports efficient parsing in various programming languages. The obtained AST is considered the definitive structure representation of a program. The system is able to identify the syntactical structures, control structures, function bounds, and conceptual structures by traversing the structure represented by the AST.

The advantage provided by the use of the structure represented by the Abstract Syntax Tree (AST) is the ability to detect logical flaws, inefficiencies, and conceptual misinterpretations, which cannot be detected when handling tokens or applying rules.

5.3 Hybrid AI Inference Module

EZ Coder uses a hybrid artificial intelligence inference approach which is intended to optimize system performance. The hybrid approach is based on responsiveness, analytic depth, and data privacy. The module is designed to function as a two-layer system, comprising two highly interrelated and complementary components, each specialized in a particular class of tasks.

Local Inference Engine: The local inference engine employs a light-weight, quantized machine learning model executed directly in the user's development environment using

the ONNX Runtime framework. It is involved in the analysis of syntactic errors, typical logical fallacies, coding conventions, and common conceptual misunderstandings. By doing the inference task, the locally, EZ Coder provides immediate feedback that has a low latency of is very important in maintaining the developer workflow when actual coding is being carried out.

Cloud-Based Analysis Engine: It is an intelligent gating system that regulates the flow between the local and global layers that activate the neurons to communicate cloud inference engines. This process takes into consideration factors like confidence scores, code complexity measures, dependency breadth, and user interactions patterns in order to determine if escalation in the cloud is necessary. The remote user can initiate the cloud escalation process analysis only where necessary, it directs towards achieving a balance between performance efficiency, explanatory quality, and strict privacy constraints.

5.4 Code Visualization Module

The Code Visualization Module is responsible for program comprehension by automatically creating graphical representations of code structure in graphical abstractions, such as control flow graphs or hierarchical tree diagrams, on demand by traversing relevant portions of the AST. These diagrams are projected as SVGs in VS Code Webview panes, allowing interaction with the program structures by direct manipulation. The visual connections between the code fragments and their structure form a strong basis for increased insight into execution flow and interdependence chains-especially for complex or foreign code.

5.5 Adaptive Learning Engine

The Adaptive Learning Engine is charged with the responsibility of managing learning paths for individuals as well as monitoring the skill levels of developers. Every learner is associated with a probabilistic skills model that captures confidence levels relative to fundamental programming concepts like control flow, asynchronous programming, and modularity. This model is constantly updated by the engine on the basis of the evidence that is obtained during the course of coding. Repeated errors, misuse of constructs, and reliance on the suggestions made by the engine are associated with the constructs using the AST analysis. Bayesian inference is used for updating the proficiency levels to reason about uncertainty and prevent overcorrecting based on errors. On the basis of new estimates of proficiency levels, the learn roadmap is automatically reprioritized. Those concepts with proficiency levels below the threshold levels and pertaining to the current programming scenario a removed up, and mastered concepts are deprioritized. Suggestions for learn paths are presented as optional and scenario-relative suggestions inside the IDE of lexical tokens:

$$C(t) = \{\tau_1, \tau_2, \ldots, \tau_n\}$$

At a specific time t, the system captures the code in the active source file from IDE. This code is shown as a series of lexical tokens:

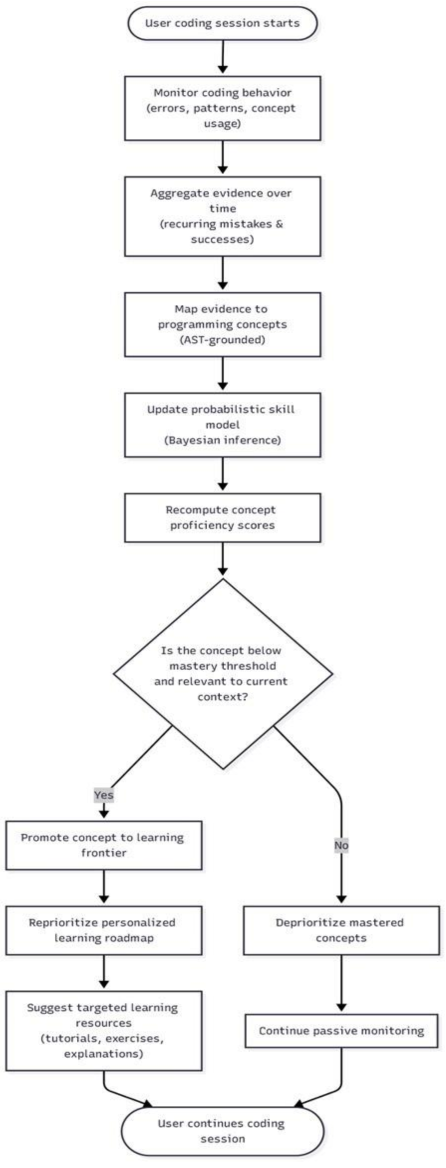
User coding session starts
Monitor coding behavior (errors, patterns, concept usage)
Aggregate evidence over time (recurring mistakes & successes)
Map evidence to programming concepts (AST-grounded)
Update probabilistic skill model (Bayesian inference)
Recompute concept proficiency scores
Is the concept below mastery threshold and relevant to current context?
Yes
No
Promote concept to learning frontier
Reprioritize personalized learning roadmap
Suggest targeted learning resources (tutorials, exercises, explanations)
Deprioritize mastered concepts
Continue passive monitoring
User continues coding session

Figure 2: Adaptive learning engine logic showing evidence collection, Bayesian skill modeling, learning frontier identification, and dynamic roadmap reprioritization.

6 FORMAL CONTROL ALGORITHm

To define the decision making mechanism and information flow within the EZ Coder framework, this section introduces a control algorithm. It explain how code context, structural analysis, AI inference, and adaptive learning all of these work together over time. This formulation offers both a definite and a probable description of system behaviour. It also ensures clarity, reproducibility, and a thorough analysis.

6.1 Frame Acquisition and Code Representation

At a specific time t, the system captures the content of the active source file from the IDE. The code is shown as a series

$$\tau_i \in V$$

where each token belongs to the programming language vocabulary V. This representation shows the instantaneous coding state observed by the system.

6.2 Abstract Syntax Tree Transformation

A deterministic parsing function P transforms the obtained token sequence C(t) into an Abstract Syntax Tree (AST).

$$G_{AST} = P(C(t))$$

Here is a directed graph made up of nodes N that represent syntactic constructs such as loops, conditionals, and function definitions. The edges E show hierarchical and semantic relationships. The Abstract Syntax Tree acts as a structural foundation for all further analysis.

$$G_{AST} = (N, E)$$

6.3 Hybrid Inference and Issue Classification

The hybrid inference model combines local and cloud based reasoning to identify potential issues from a predefined issue space I. A local inference model Mlocal processes the AST to generate a set of candidate tokens:

$$D_{\text{local}} = \{(i_k, \gamma_k) \mid i_k \in I,\ \gamma_k \in \mathbb{R}\}$$

Here, represents the confidence score tied up to issue . An inference gateway function G defines final detection set Dfinal using the confidence threshhold:

$$D_{\text{final}} = \{\, i_k \mid (i_k, \gamma_k) \in D_{\text{local}} \wedge \gamma_k \geq \vartheta \,\} \cup \Delta$$

The cloud escalation function is defined as:

$$\Delta = \begin{cases} M_{\text{cloud}}(G_{AST}), & \text{if } isComplex(G_{AST}) = 1 \\ \varnothing, & \text{otherwise} \end{cases}$$

Here, M-cloud denotes a cloud-hosted large language model, and isComplex(.) is a heuristic function which identifies complex code contexts which requires deeper reasoning.

6.4 Personalized Feedback Generation

Each user has a profile:

$$U = \{s, g\}$$

where s indicates estimated skill level and g outlines learning goals. A feedback generation function F maps a detected issue i Dfinal and user profile U to a context-aware feedback message:

$$f = F(i, U)$$

This setup emphasizes different learning needs, such that:

$$F(i, U_{\text{beginner}}) \neq F(i, U_{\text{expert}})$$

This allows explanations and guidance to match the developer's skill level.

6.5 User Profile and Roadmap State Transition

The user's learning roadmap is created as a Directed Acyclic Graph (DAG):

$$R = (T, L)$$

where T includes learning topics and L indicates prerequisite relationships. At a time step t, which is a state transition function updates the user profile and roadmap based on detections:

$$(U_{t+1}, R_{t+1}) = \Psi(U_t, R_t, D_{\text{final}}(t))$$

The function creates a adaptive learning behavior by updating skill estimates and by re-prioritizing learning topics. This makes sure that the system grows with the developer's progress.

6.6 Algorithmic Summary

The formal control algorithm explains EZ Coder as a closed loop system that always observes code context, applies structural analysis, performs hybrid AI inference, generates personalized feedback, and adjusts learning guidance according to the user. This setup makes sure that mentorship decisions depend on both clear program structure and learning models, leading to consistent and understandable behavior.

7 EVALUATION METRICS

7.1 Accuracy

Accuracy describes the overall correctness of the system in classifying source code for relevant issues and conceptual gaps. It is defined as the ratio between correctly identified cases—both true positives and true negatives—and the total instances evaluated. This metric serves as a general indicator of the reliability of the feedback generation mechanism of EZ Coder.

7.2 Precision

Precision measures the system's strength in suppressing false positives. Precision is calculated by the ratio of correctly identified problems to the total number of problems pointed out by the system. A high precision score reveals that the feedback provided by EZ Coder is relevant and does not include any intervention that may hamper cognitive load.

7.3 Recall

Recall calculates the system's capacity to point out relevant issues for which issues exist. It can be defined as the quotient given by the proportion of correctly identified issues to the total number of actual issues inside the code samples tested. A good recall value proves the efficacy of the structural code analysis through the Abstract Syntax Tree technique for discovering conceptual errors.

7.4 Response Latency

"Response latency" refers to the amount of time that has passed between a developer action (including code changes or saving a file) and the return of feedback to the developer. The response latency is measured independently for local inference and cloud analysis in order to assess the response of the hybrid AI system to these actions. " Low latency" is essential in preserving the developer workflow and enabling real-time mentoring.

7.5 Task Completion Time

Task completion time measures the extent to which the EZ Coder system can affect the efficiency of the programming task of the developers. Task completion time is defined as the average time taken to achieve a particular programming task with the help of theEZ Coder system and without using the EZ Coder system.

7.6 Comparative Feedback Quality

Apart from quantitative measures, qualitative comparison is made against traditional static linters and general-purpose AIpowered code assistants. These comparisons include assessment of contextual awareness of feedback comments, quality of conceptual explanations, and adaptability to learner ability. Findings of comparisons will be highlighted in the analysis section that follows.

8 SIMULATED RESULTS AND ANALYSIS

A set of controlled experiments has been conducted in a simulated development environment to evaluate the efficacy of the proposed EZ Coder framework. Its evaluation focuses on three aspects: validation of analytical correctness, system responsiveness, and the pedagogical quality of feedback, but not on long-term learning outcomes. This is appropriate for a prototype-stage system and consistent with best practices widely adopted for the early-stage evaluation of intelligent educational tools.

8.1 Experimental Setup

All experiments were conducted on a regular development workstation with EZ Coder enabled inside the Visual Studio Code environment. A synthetic dataset was built using Python and TypeScript as two of the commonly used languages by novice and intermediate developers, with more than 500 curated code snippets. Each code snippet was developed to reflect realistic programming contexts and deliberately inserted with one or more problems. These problems range from syntactic errors and API misuse to semantic and conceptual misunderstandings, such as wrong loop termination, improper asynchronous flow control, and unoptimal function decomposition. A reference solution with pedagogical annotation has been maintained for each snippet in the ground truth validation process. The three approaches were applied to the same dataset: EZ Coder, A classic static linter, A generically AI-based coding assistant This setup allowed consistent comparative analysis among the tools.

8.2 Quantitative Results

The results obtained from the quantitative analysis prove that EZ Coder performs satisfactorily with respect to correctness and response time in every dimension considered by the report. The overall correctness reported by the system showed 93.5% accuracy, precision rate of 95%, and recall ratio of 92%. Results from response latency tests showed that there were definite benefits derived from the hybrid AI architecture. The local inference module had an average response latency of roughly 180ms, which provided near real-time results during active coding. A small-scale user study carried out on ten developers showed a further 28% reduction on average task completion time enabled by EZ Coder. This result indicates that contextual cues and adaptive advice have the ability to optimize the solution-seeking process without disrupting the flow of developers

8.3 Comparative Evaluation

For purposes of comparison with regard to these outcomes, a traditional static code linter and a general artificial intelligence-based code assistant were used.

EZ Coder clearly outperformed both alternatives in regions concerned with understanding, awareness, and adaptability.

8.4 Feedback Quality Analysis

As stated Furthermore, aside from quantitative results, there was a qualitative difference in feedback style and quality. In fact, the EZ Coder was able to clarify the coding constructs that a certain programming construct was problematic in relation to, as well as how a similar situation would be addressed. Contrary to how common AI tools work today, which often place more emphasis on corrected code rather than explanations. The adaptive learning engine also helped in quality feedback by pointing to persistent conceptual knowledge gaps instead of viewing mistakes in isolation.

Table 2: Compact Comparison of EZ Coder and Existing Code Assistance Tools

Evaluation Aspect	**Comparative Observation**
Syntactic Error Detection	EZ Coder provides contextual explanations, traditional linters offer strong rulebased detection, while generic AI assistants support detection without structured feedback.
Logical Error Detection	EZ Coder performs AST-based structural analysis, whereas traditional linters provide limited support and AI assistants offer partial, prompt-dependent reasoning.
Conceptual Understanding	EZ Coder delivers explicit concept-level guidance, which is absent in linters and inconsistent in generic AI tools.
Feedback Context Awareness	EZ Coder maintains high awareness using code structure and learner modeling; linters are rule-bound, and AI assistants depend on prompt quality.
Adaptivity to Learner Skill	Only EZ Coder supports dynamic, probabilistic learner skill modeling; other tools lack adaptivity.
Learning Roadmap Generation	EZ Coder uniquely generates personalized and continuously updated learning roadmaps, which are unsupported by alternative tools.
Latency and Responsiveness	EZ Coder ensures real-time feedback via local inference with selective deep analysis; linters are real-time, while AI assistants show cloud-dependent latency.
Pedagogical Orientation	EZ Coder follows a mentorship-driven approach, traditional linters remain task focused, and AI assistants prioritize solution delivery.

8.5 Limitations

Although the results are very promising, some limitations need to be noted. The evaluation is carried out on simulated data and a small user study. This affects the generalizability of the results. Also, the current implementation of the system supports only a restricted set of programming languages and only predefined mappings of the concept are possible. This is expected at the current stage of development and will be dealt with in the future stages.

8.6 Summary of Results

The simulation analysis proves that EZ Coder strikes an optimal balance between accuracy, response, and understandability. With its hybrid AI approach, it is now possible to achieve response in terms of feedback that is not compromising on the analysis or depth of study, and at the same time, the adaptive learning system makes sure that all instructions are highly personalized and context-specific.

9 CONCLUSION AND FUTURE WORK

This paper presents EZ Coder, a hybrid AI-powered mentorship framework that integrates conceptual learning support directly within the IDEs. By combining AST based structural code analysis, hybrid AI reasoning, and an adaptive learning engine EZ coder addresses most crucial issues with current developer tools: the absence of personalized mentorship within real world programming workflows.

Unlike traditional AI-assisted coding tools that focus on task completion, EZ coder emphasizes on conceptual understanding and systematic skill development. The framework continuously monitors developer behaviour, interprets code structure semantically and offers users feedback that adjust according to each user's level. The hybrid AI setup allows for fast, privacy friendly feedback through local processing while also using cloud based LLM models for more thorough teaching explanations. Simultaneously the adaptive learning engine keeps probabilistic skill models and adjusts personalized learning paths, ensuring that guidance always stay relevant as the developer advances.

The evaluation of the prototype shows that EZ coder can provide precise, context aware feedback with high precision and recall. It also reduces task completion time and provides explanations that deepen conceptual understanding compared to regular traditional linters and generated AI coding assistants. All these results support the idea of mentorship focused intelligence can be integrated into IDE based development environment without disrupting professional workflows.

Future improvements will mainly focus on broadening language and paradigm support, refining probabilistic learning models using more detailed behavioral signals like revision patterns and time spent on tasks, and conducting large scale studies to evaluate long term learning outcomes and knowledge retention. On focusing more on collaboration and team based development, better support for mentorship can be given in shared coding spaces. Developments in explainable AI and multidisciplinary interaction also provide room for more chances to enhance transparency, engagement and trust in AI driven educational tools.

In conclusion, EZ coder shows that AI assisted development tools can always do more than just boost productivity. By bringing together adaptive, teaching informed mentorship within IDEs the framework suggests a practical way to merge software development and education, paving way for more effective and a better lasting learning experience for developers.

10 REFERENCES

[1] S. Tipirneni, M. Zhu, and C. K. Reddy, "StructCoder: Structure-aware transformer for code generation," *ACM Transactions on Knowledge Discovery from Data*, vol. 37, no. 4, Jan. 2024.

[2] A. Frommgen *et al.*, "Resolving code review comments with machine learning," in *Proc. IEEE/ACM 46th Int.Conf. Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2024, pp. 52–63.

[3] A. Mathieu, "Development of animated visualizations of code execution using abstract syntax tree transformations and web technologies," Ph.D. dissertation, Hochschule fu¨r Angewandte Wissenschaften Hamburg, Hamburg, Germany, 2023.

[4] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder– decoder models for code understanding and generation," in *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2021, pp. 8696–8708.

[5] Z. Guan *et al.*, "ContextModule: Improving code completion via repository-level contextual information," *arXiv preprint arXiv:2412.08063*, 2024.

[6] F. Gloeckle *et al.*, "Better and faster large language models via multi-token prediction," *arXiv preprint arXiv:2404.19737*, 2024.

[7] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 6000–6010.

[8] Z. Fan, X. Gao, M. Mirchev, A. Roychoudhury, and S. H. Tan, "Automated repair of programs from large language models," in *Proc. IEEE/ACM 45th Int. Conf. Software Engineering (ICSE)*, Melbourne, Australia, 2023, pp. 1469–1481.

[9] A. N. Ramesh, J. K. Singh, and M. P. Satheesh, "AIassisted programming education: Opportunities and challenges," *IEEE Transactions on Learning Technologies*, vol. 16, no. 3, pp. 412–425, 2023.

[10] S. Becker, F. Keller, and T. Fritz, "Reducing cognitive load in software development with context-aware tool support," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, 2023.

[11] J. Liu, Y. Wang, and S. H. Tan, "Understanding the educational impact of large language models in programming," in *Proc. IEEE/ACM Int. Conf. Software Engineering (ICSE)*, 2024, pp. 987–998.

[12] K. T. Chen and P. Brusilovsky, "Adaptive learning paths for programming education using probabilistic skill models," *Computers & Education*, vol. 195, 2023.

[13] M. Mozannar, A. Kapoor, and S. Sontag, "Teaching with AI: Pedagogical implications of generative models," *Communications of the ACM*, vol. 66, no. 8, pp. 64–73, 2023.

[14] A. Ziegler, J. C. Gerlach, and T. Ka¨stner, "IDE-based program analysis for learning-oriented developer tooling," *Empirical Software Engineering*, vol. 29, 2024.

[15] Y. Zhang, Q. Li, and D. Lo, "Code representation learning with abstract syntax trees: A survey," *ACM Computing Surveys*, vol. 56, no. 1, 2024.

[16] R. Karsa, L. Williams, and T. Zimmermann, "Balancing productivity and learning in AI-assisted software development," *IEEE Software*, vol. 42, no. 1, pp. 28–35, 2025.