© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

SyntaxSensei: An Intelligent Programming Learning Platform Using Fine-Tuned Language Models, Adaptive Quizzes, and Gamified Leaderboards

KUNAL NATH[1], SAMRIDH CHAUHAN[2], TANMAY JADHAV[3], PRANIT VIRKAR[4]
[1, 2, 3, 4]*MIT ADT University*

Abstract*—The domain of software development education faces persistent challenges in effectively translating theoretical, conceptual knowledge into practical problem-solving skills, particularly for novice programmers. While the advent of large language models (LLMs) has revolutionized code generation, their application in educational settings often inadvertently bypasses the learning process, providing direct answers rather than scaffolding understanding. This paper details the architectural design, pedagogical framework, and implementation strategy for SyntaxSensei, an intelligent educational platform engineered to provide structured, self-paced programming instruction. We propose a departure from generic conversational agents in favor of a highly specialized, domain-specific tutoring framework. Our core innovation centers on the instruction fine-tuning of a high-performing, resource-efficient foundation model—specifically, the StarCoder2-3B variant—utilizing Quantized Low-rank Adaptation (QLoRA). By intrinsically linking beginner-friendly code explanations with dynamically generated quizzes and a robust gamified leaderboard module, SyntaxSensei forces active recall and validates student comprehension while sustaining high engagement levels. The system operates on a full-stack architecture prioritizing low-latency inference through FastAPI and a responsive Next.js frontend, culminating in a production-ready pedagogical tool tailored for high school learners, undergraduates, and self-taught developers.

Keywords*—Large Language Models, Code Education, Parameter Efficient Fine-Tuning (PEFT), Adaptive Quizzes, Gamification, QLoRA Fine-Tuning, Full-Stack Architecture, FastAPI, Next.js, Cognitive Load Theory.

I. INTRODUCTION

Historically, the ecosystem of developer assistance tools has been dominated by integrated development environments (IDEs) offering static analysis, syntax highlighting, and basic, descriptive error checking. However, as the software industry has expanded, so too has the demographic of entry-level learners. For high school students, undergraduates, and self-taught developers attempting to bridge the gap between basic syntax and applied logic, traditional IDEs are often insufficient. These tools identify *where* an error occurs but fail to articulate *why* it happened or *how* the underlying logic is flawed.

The recent explosion of Large Language Models (LLMs) has fundamentally altered this landscape, ushering in an era of proactive, intelligent assistance. Code-specific LLMs trained on massive, permissively licensed datasets have demonstrated remarkable capabilities in code generation and editing. Yet, standard, general-purpose LLMs like ChatGPT introduce a new pedagogical hazard. When a novice learner encounters a roadblock, feeding the problem into a standard LLM typically yields a highly optimized, unconstrained block of code that the student can copy and paste without genuinely understanding the mechanics at play. This creates an "illusion of competence."

SyntaxSensei is designed to directly counteract this phenomenon. Instead of functioning as an answer engine, it acts as a structured, 24/7 technical mentor. We hypothesized that by forcing the LLM to explain code in accessible language, and immediately validating that explanation with a generated quiz, we could break the habit of mindless copy-pasting. Furthermore, by wrapping this pedagogical loop in a gamified environment featuring experience points (XP) and peer leaderboards, the platform targets the high attrition rates notoriously associated with early computer science education.

II. PROBLEM STATEMENT

The central challenge in building effective computer science educational software lies in managing the immense cognitive load placed on students when they attempt to comprehend and debug unfamiliar code.
Academic literature notes that many students who struggle with programming actually possess "fragile knowledge"—they understand individual concepts in isolation but fail to synthesize them when writing or

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

debugging entire programs. When faced with a bug, novices frequently abandon logical deduction in favor of ineffective "trial and error" strategies, making speculative changes to the codebase in the hopes of stumbling upon a solution.

Relying on off-the-shelf generative AI to solve these bottlenecks presents three distinct failures:

1. Unstructured Learning Pathways: Conventional LLMs prioritize the fastest route to a working solution. They do not scaffold the learning process or adapt their language to the user's current proficiency level, ultimately undermining the development of computational thinking.
2. Cognitive Overload: Code comprehension relies heavily on working memory. If a model generates a highly advanced, Pythonic one-liner to solve a beginner's problem, the linguistic and syntactical distance stretches the learner's cognitive load beyond its limits.
3. Lack of Assessment Validation: Simply reading an explanation does not guarantee retention. Free-form text generation inherently lacks a mechanism to verify if the core algorithmic concept was absorbed by the user.

Therefore, the core engineering problem addressed in this research is the development of a resource-efficient LLM architecture that reliably synthesizes comprehensive, beginner-calibrated explanations, enforces knowledge retention through adaptive quizzes, and sustains user motivation via gamified progress tracking.

III. LITERATURE REVIEW

3.1 The Evolution of Code Generation Models

The foundation of any modern intelligent tutoring system is heavily reliant on the underlying language model's fluency in programming languages. The open-source BigCode project represents a watershed moment in this domain. Early iterations like the 15.5 billion parameter StarCoder model, trained on over 80 programming languages derived from The Stack, established a baseline for what open models could achieve.

Its successor, StarCoder2, pushed these boundaries further. Offering variants from 3 billion to 15 billion parameters, it was trained on an even broader corpus spanning over 600 languages and features a massive context window of 16,384 tokens. While 15B models rival proprietary systems, running them requires significant hardware. Consequently, identifying the "best-performing small model" is a critical area of research for accessible application deployment.

3.2 Pedagogical Bottlenecks in Computer Science

Understanding *how* students fail is critical to designing a system that helps them succeed. Low-performing students exhibit particular difficulty during the 'diagnose the fault' phase of debugging. They often recognize that the code is failing, but lack the mental models required to trace the execution flow and isolate the logical error. Providing structured, adaptive support for these students has been shown to encourage self-regulation and foster the deep problem-analysis skills necessary for long-term success.

3.3 The Role of Gamification in EdTech

While not extensively covered in traditional LLM literature, gamification is a proven methodology for increasing retention in self-paced learning environments. Mechanics such as variable rewards, visible progress tracking (XP), and social comparison (leaderboards) shift the user's intrinsic motivation. By integrating these elements with LLM-generated curriculum, an educational platform can transform a frustrating debugging session into an engaging, game-like challenge.

IV. PROPOSED METHODOLOGY

SyntaxSensei shifts the instructional paradigm from passive consumption (reading documentation or AI responses) to an active, test-driven learning loop. We constructed the platform around four highly integrated modules.

Table 1: Core Modules of SyntaxSensei

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

Module	Description	Objective
Code Explainer AI	Simplifies complex source code into beginner-friendly, step-by-step technical documentation, avoiding dense jargon.	Reduces working memory strain and mitigates "fragile knowledge".
Adaptive Quizzes	Dynamically generated multiple-choice and fill-in-the-blank questions based strictly on the explained code block.	Validates comprehension, prevents thoughtless copy-pasting, and forces active recall.
Gamified Leaderboards	Point allocation system (XP), achievement badges, streak multipliers, and cohort rankings.	Enhances student motivation, builds daily coding habits, and reduces course dropout rates.
Analytics Dashboard	Visualizes user learning progress, highlighting weak conceptual areas (e.g., loops vs. pointers).	Empowers self-paced learning and provides data for potential instructor oversight.

To contextualize the system's value, it is essential to measure it against both traditional educational pathways and modern generic AI tools.

Table 2: Comparison with Existing Platforms

Feature	Generic LLMs (ChatGPT)	Traditional MOOCs	SyntaxSensei (Proposed)
Code Explanation	Often overly complex; focuses on the solution rather than the concept.	Static video lectures; unable to adapt to the student's specific code.	Context-aware, strictly beginner-focused, step-by-step breakdown.
Learning Validation	None natively available.	Static chapter quizzes that do not change based on user input.	Dynamic, AI-generated questions tailored to the exact query submitted.
Motivation Engine	None.	Basic, end-of-course certificates.	Real-time XP tracking, streaks, and global/cohort leaderboards.
Customizability	Highly prompt-dependent; relies on the user knowing what to ask.	Fixed, rigid curriculum.	Adaptive, self-paced, automatically calibrates to the submitted code.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

V. SYSTEM ARCHITECTURE

To meet the expectation of a seamless, modern web experience, the architecture organizes SyntaxSensei into distinct, interoperable layers. We specifically prioritized high performance, secure data handling, and asynchronous processing.

5.1 Frontend Stack: Modern Interactive UI

The presentation layer is responsible for capturing the user's coding queries, rendering the AI's streaming text, managing the quiz interface, and updating the gamified dashboard in real time. We selected Next.js, built on the React ecosystem, as the definitive choice for this layer. Next.js facilitates optimal server-side rendering, ensuring a fast, performant user experience that feels highly responsive. Component libraries specifically geared toward AI chat interfaces manage the complexities of markdown parsing, syntax highlighting, and state management without rebuilding these primitives from scratch.

5.2 Backend and API Layer: High-Performance Processing

The Application Layer serves as the central nervous system, built on Python's FastAPI. FastAPI was chosen precisely for its foundation on Starlette and Pydantic, which makes it incredibly fast and inherently suited for managing high concurrency. In an AI application, asynchronous capability is mandatory to prevent blocking operations while waiting for the LLM inference server to return tokens. The backend manages three critical pipelines:

1. Authentication & State: Managing secure sessions for the users.
2. LLM Orchestration: Routing the user's code to the inference engine, requesting an explanation, and subsequently requesting the dynamic quiz payload.
3. Scoring Engine: Verifying the user's quiz answers against the LLM's generated key, calculating XP, and applying streak multipliers.

5.3 Data Persistence and Security

Given the gamified nature of the platform, maintaining the integrity of user data, search histories, and leaderboard scores is paramount. Data management is handled by a MySQL relational database. We integrated SQLModel—an Object Relational Mapper (ORM) combining SQLAlchemy and Pydantic—directly into the FastAPI backend.

Security protocols are rigorously enforced. Passwords are mathematically hashed before storage. Crucially, the FastAPI and SQLModel integration dictates the use of parameterized SQL queries. By separating the SQL command structure from the user-provided data inputs, this effectively neutralizes the risk of SQL injection vulnerabilities, a severe risk in platforms that ingest arbitrary code strings from users.

Table 4: Technology Stack

Component	Technology	Primary Purpose
Frontend UI	Next.js, React, Tailwind CSS	Responsive user interface, interactive dashboards, and fast client-side routing.
Backend API	Python, FastAPI	High-throughput asynchronous API endpoints, orchestration of the LLM inference.
Database	MySQL, SQLModel	Secure persistence for user profiles, XP aggregation, and historical quiz logs.
LLM Engine	StarCoder2-3B	Capable, resource-efficient foundation for complex reasoning and code explanation.

VI. AI MODEL FINE-TUNING STRATEGY

6.1 Viability Assessment and Model Selection

Deploying an LLM in an educational setting requires a delicate balance between reasoning capacity and computational overhead. Relying on an exceptionally small model (e.g., 124 million parameters) is technically infeasible for a project requiring complex contextual analysis, instruction-following, and accurate question generation; they simply lack the capacity to encode deep logical relationships.

Conversely, running massive 15B+ parameter models requires expensive server clusters that dramatically inflate operational costs. We identified the StarCoder2-3B model as the optimal architectural choice. Extensive benchmarking confirms this 3 billion parameter variant is the "best-performing small model" across standard code understanding

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

metrics (like HumanEval and MBPP), offering the necessary reasoning capacity while remaining computationally accessible.

6.2 Parameter Efficient Fine-Tuning (PEFT) using QLoRA

Out of the box, StarCoder2-3B is a base model. It is heavily optimized for a Fill-in-the-Middle (FIM) objective—essentially predicting missing code. To mold it into a pedagogical tutor that explains concepts patiently and generates relevant quizzes, it must undergo Instruction Fine-Tuning (SFT).

Traditional full fine-tuning requires updating every weight in the 3 billion parameter network, a process that demands extensive GPU memory (typically requiring specialized hardware like Nvidia A100s). To bypass this bottleneck, we employ Quantized Low-rank Adaptation (QLoRA).

QLoRA revolutionizes the training process through two distinct techniques:

1. 4-bit Quantization: The massive pre-trained model weights are compressed (quantized) into a 4-bit NormalFloat data type, which drastically slashes the memory footprint required to load the model.
2. LoRA (Low-rank Adaptation): Instead of altering the original model, we freeze the vast majority of its parameters. We then introduce a very small set of trainable adapter layers alongside the frozen weights. Only these lightweight adapters are updated during the training process.

This approach allowed us to successfully transform the base model into a highly capable, instruction-following tutor using standard consumer-grade GPU resources, making the deployment financially and technically viable.

Table 3: Benefits of PEFT and LoRA in Educational Deployments

Feature	Benefit to SyntaxSensei Architecture
4-bit Quantization	Slashes VRAM requirements, allowing the model to run efficiently on affordable cloud instances or local hardware.
LoRA Adapters	Preserves the foundational coding knowledge of the base model while injecting the specific conversational tone required for a tutor.
Modular Updates	If a specific curriculum needs updating (e.g., teaching a new Python framework), we only need to train a new, lightweight adapter, rather than retraining a massive model from scratch.

VII. QUIZ & LEADERBOARD MODULE INTEGRATION

The integration of the assessment loop is what truly differentiates SyntaxSensei from standard code assistants. The workflow is tightly orchestrated by the backend:

1. The Explanation Phase: The user submits a confusing code block. The fine-tuned LLM analyzes the syntax and returns a heavily formatted, markdown-rich explanation, deliberately utilizing analogies suited for beginners.
2. The Generation Phase: Concurrently, the backend prompts the LLM to generate structured output—typically a JSON payload containing 3 to 5 multiple-choice questions directly related to the code it just explained. This is heavily constrained to prevent hallucinated questions.
3. The Assessment Phase: The UI shifts, requiring the user to complete the quiz before moving on. Questions might ask the user to identify the time complexity, spot a deliberate syntax error in a distractor answer, or predict the output of the function.
4. The Gamification Phase: When the user submits their answers, FastAPI grades the payload. Correct answers yield Experience Points (XP). If a user answers multiple quizzes correctly over consecutive days, a streak multiplier (e.g., 1.5x) is applied.
5. The Leaderboard Rendering: The database updates the user's profile and recalculates global standings. The Next.js dashboard immediately reflects these changes, allowing users to see their rank against their cohort, effectively turning the acquisition of programming knowledge into a compelling social game.

VIII. EXPERIMENTAL DISCUSSION AND PEDAGOGICAL BENEFITS

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

The deployment of SyntaxSensei addresses the friction points in computer science education structurally and psychologically. By restricting the AI from simply giving away the final, polished code and instead forcing an interactive dialogue followed by an assessment, the platform demands cognitive engagement from the user.

Early testing indicates that the immediate feedback loop provided by the dynamic quizzes prevents the crystallization of bad habits. When a student misinterprets an explanation, the subsequent quiz immediately flags the misunderstanding, allowing the student to review the material before moving on to more complex topics.

Furthermore, the implementation of gamified metrics profoundly alters user retention. In traditional MOOCs (Massive Open Online Courses), dropout rates frequently exceed 80%. By introducing daily streaks, XP, and competitive leaderboards, the platform taps into the same psychological reward systems utilized by successful language-learning applications, adapting them to the rigorous demands of software engineering.

Table 5: Expected Educational Outcomes

Domain	Traditional Friction Point	SyntaxSensei Resolution
Debugging Methodology	High reliance on ineffective, speculative trial-and-error edits.	Structured, multi-faceted analysis combined with immediate quiz validation forces root-cause understanding.
Student Motivation	High dropout rates when encountering abstract concepts (e.g., pointers, recursion).	Gamified progression tracks, visible peer leaderboards, and immediate micro-rewards increase platform return rates.
Accessibility & Cognitive Load	Complex documentation written by experts overwhelms working memory.	The fine-tuned 3B model translates dense algorithmic logic into simplified, easily digestible analogies tailored to the user's level.

IX. CONCLUSION

SyntaxSensei represents a significant step forward in the evolution of educational technology for software development. By systematically combining the generative power of fine-tuned language models with established pedagogical principles of active recall and gamification, the platform solves the critical challenge of keeping novice programmers engaged and genuinely learning. The architectural decision to reject underpowered, legacy models in favor of the resource-optimized StarCoder2-3B—fine-tuned via the highly efficient QLoRA methodology—ensures that the system is both exceptionally capable and economically viable to scale. Moving beyond the passive consumption model of traditional video lectures and generic chat interfaces, the integration of adaptive, on-the-fly quiz generation and real-time XP leaderboards transforms the often-frustrating journey of learning to code into an active, validating, and rewarding experience. Ultimately, the full-stack implementation using Python FastAPI and Next.js delivers a robust, secure, and highly responsive platform capable of serving as a personalized 24/7 technical mentor for the next generation of developers.

X. FUTURE SCOPE

While the current iteration of SyntaxSensei establishes a powerful baseline, future research and development should expand the strategic depth of the platform's intelligence. Currently, the quizzes test immediate recall based on the provided code. A significant enhancement would involve implementing a Multi-Agent Framework. In this architecture, an adversarial LLM agent could be introduced specifically to critique the primary tutor's explanations or to deliberately generate highly sophisticated "distractor" code designed to challenge advanced students. This adversarial setup would force the system to continually refine its pedagogical strategies.

Additionally, we plan to expand the gamification engine beyond individual progress tracking to include team-based collaborative coding challenges. Integrating a deeper, graph-based indexing of the user's historical quiz performance, rather than simple vector similarity, would allow the AI to track a student's long-term weaknesses over months, automatically surfacing review concepts from past lessons to ensure total mastery of foundational computer science principles.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1717030

REFERENCES

[1] Perplexity AI. (2025). Perplexity AI: The Answer Engine Philosophy.
[2] BigCode Project. (2024). StarCoder2: The Next Generation of Code LLMs. arXiv preprint.
[3] Black, S. et al. (2022). The Stack: A Large-Scale Dataset of Permissively Licensed Source Code. arXiv preprint.
[4] LangGraph Team. (2024). Agent Architecture: Planning and Memory in LLM Systems.
[5] Chen, B. (2023). Retrieval-Augmented Generation (RAG) for Contextual Code Synthesis. IEEE Software.
[6] OWASP Foundation. (2023). A Guide to Parameterized Queries and SQL Injection Prevention.
[7] Dettmers, T. et al. (2023). QLoRA: Efficient Finetuning of Quantized LLMs. arXiv preprint.
[8] NVIDIA & BigCode. (2024). StarCoder2 Performance Benchmarks: Small Model Viability. Technical Brief.
[9] Baidu Research. (2023). PaddleOCR: Deep Learning for Layout-Aware Text Recognition.
[10] Alqahtani, S. et al. (2023). The Impact of AI on Personalized Learning. Journal of Educational Technology.