PyAI: an LLM-Driven AI Mentor for Personalized and Optimized Learning

Spoorthi Gumgol[1], Saanika Wani[2], Shreeya Rajurikar[3], Piyush Kale[4], Prof. Priyanka Deshpande[5]
[1,2,3,4]*Dept. of Artificial Intelligence and Data Science PES's Modern College of Engineering Affiliated to Savitribai Phule Pune University Pune, India*
[5]*Guide, Dept. of Artificial Intelligence and Data Science PES's Modern College of Engineering Affiliated to Savitribai Phule Pune University Pune, India*

***Abstract*—This paper presents the design and implementation of PyAI, an AI-driven, gamified programming education platform built for beginner-to-intermediate learners, with a primary focus on students aged 8–16. PyAI combines a React-based single-page application, a Supa base-hosted backend for authentication and persistent content storage, an interactive coding environment for hands-on Python practice, and an in-application AI Mentor character that delivers contextual feedback and motivational guidance during the learning process. The platform delivers structured Python courses through a hierarchical content model (courses, units, topics, lessons, and tests) stored in Supa base and retrieved dynamically at runtime. A skill-assessment evaluation module places students into ap-propriate learning tiers before enrollment. An adaptive progress system implemented in React Context tracks per-lesson accuracy using a rolling average and adjusts the difficulty label assigned to each student's next attempt. A separate Mentor Dashboard, powered by the Recharts library, gives instructors a real-time view of class-level XP accumulation, topic mastery, and individual student progress. Preliminary evaluation with a student cohort shows improvements in task completion rate and self-assessed comprehension over a static learning alternative. This work demonstrates the feasibility of building a functional, deployable AI-assisted coding education platform using a modern frontend stack with a managed backend-as-a-service provider.**

***Index Terms*—AI Mentor, Coding Education, Supa base, Adaptive Learning, Gamification, React, Intelligent Tutoring, Python Education, PyAI.**

I. INTRODUCTION

Programming education has become an essential skill in the modern technological era due to the rapid growth of software development, Artificial Intelligence, and data-driven applications. As industries increasingly rely on computational systems and intelligent technologies, the demand for program-ming knowledge has expanded across multiple academic and professional domains. However, many students face significant difficulties while learning programming concepts, particularly in understanding logical problem-solving, debugging errors, and writing optimized code. Traditional classroom teaching methods and several existing online coding platforms often provide limited personalized guidance, making it difficult for learners to identify mistakes and improve their conceptual understanding effectively.

Recent advancements in Artificial Intelligence (AI) and Large Language Models (LLMs) have created new opportunities for transforming programming education through intelligent and adaptive learning systems. Modern LLMs such as GPT-4, LLaMA, and CodeT5 are capable of interpreting code, generating explanations, identifying logical errors, and assisting users through human-like interaction. These capabilities have enabled the development of AI-powered educational platforms that can provide real-time guidance, contextual feedback, and personalized learning experiences.

To address the limitations of conventional coding education platforms, PyAI is proposed as an AI-driven smart learning platform designed to provide intelligent mentorship and inter-active coding assistance for

students between the ages of 8 and 16 years. The system integrates a web-based coding environment, a secure sandboxed code execution engine, and an LLM-powered AI mentor capable of analyzing user-submitted code and generating adaptive feedback. Unlike traditional coding systems that primarily focus on output correctness, PyAI emphasizes conceptual understanding by offering contextual hints, logical error analysis, optimization suggestions, and personalized explanations based on the learner's interaction history and proficiency level.

The architecture of PyAI follows a modular and scalable design consisting of frontend interaction services, backend communication modules, AI processing layers, database management systems, and secure execution environments. The platform is designed to support real-time code evaluation, intelligent feedback generation, and adaptive learning work-flows that improve user engagement and learning efficiency. Additionally, the integration of analytics and performance tracking mechanisms enables the system to monitor coding behavior, evaluate learner progress, and recommend personalized learning paths.

The implemented system demonstrates the practical application of Artificial Intelligence in educational technology by combining intelligent tutoring mechanisms with interactive programming environments. Preliminary testing and user evaluation indicate that the platform improves debugging assistance, enhances coding comprehension, and provides a more engaging learning experience compared to traditional static learning methods. Through intelligent automation, contextual interaction, and adaptive assistance, PyAI aims to contribute to the advancement of AI-assisted programming education and intelligent tutoring systems.

II. PROBLEM DEFINITION AND OBJECTIVES

A. Problem Definition

Students learning programming at the beginner level face three recurring challenges: (1) identifying the logical cause of a bug rather than just its symptom, (2) understanding how to improve code efficiency once a solution is found, and (3) sustaining motivation when feedback is purely correctness-based and impersonal. Traditional e-learning platforms address output correctness through automated test cases but provide no explanation of why an approach is wrong or how it could be improved. This gap reduces learning depth and increases dropout rates in introductory programming courses [1].

A further practical constraint in student-facing platforms is deployment simplicity. Server-side code execution introduces security, scalability, and infrastructure complexity that is non-trivial to manage in a student or small-team project context.

B. Objectives

The specific objectives guiding PyAI's development are:

- To implement a structured, gamified Python learning platform with course, unit, topic, lesson, and test content managed through a cloud database.
- To provide an interactive coding environment that sup-ports Python practice and output verification within the learning platform.
- To implement an AI Mentor character that delivers contextual hints, motivational feedback, and error-aware guidance inline within lessons and assessments.
- To implement a client-side adaptive progress system that tracks per-lesson rolling accuracy and adjusts assigned difficulty in response to student performance.
- To build a Mentor Dashboard that gives instructors a visual, analytics-driven overview of class and individual student progress.
- To implement user authentication and persistent profile storage through Supa base, ensuring student progress and identity are maintained across sessions.
- To evaluate the system's educational utility through user testing with student participants.

III. RELATED WORK

Intelligent Tutoring Systems (ITS) have been studied since the 1980s for their potential to replicate one-on-one human tutoring in digital environments [5]. Earlier systems relied on rule-based engines and Bayesian Knowledge Tracing (BKT)

[2] to model student knowledge states and adapt feedback accordingly. While effective in narrow domains, such systems lack the linguistic flexibility

needed for open-ended program-ming tasks.

The emergence of LLMs has substantially expanded ITS capability. Research by Smith and Li [1] demonstrates that AI tutors can improve student motivation and learning out-comes in STEM subjects. Nguyen and Patel [9] identify prompt design and session context management as critical determinants of LLM tutor quality. Work on CodeT5 [7] and related code-specific pre-trained models establishes that fine-tuned transformers achieve strong performance on code understanding and generation tasks.

Gamification has been shown to improve engagement and retention in educational technology platforms. Duolingo's streak and XP mechanics [11] are among the most studied examples of gamified progress systems for language learning, and their applicability to programming education is increasingly explored in the literature.

Commercial platforms such as LeetCode and Codecademy provide automated test-based grading but deliver no qualitative feedback on logic or efficiency. PyAI differentiates itself by combining browser-native execution with an AI Mentor char-acter, gamified XP and streak mechanics, and an instructor-facing analytics dashboard capabilities not simultaneously present in existing platforms targeting this age group.

IV. SYSTEM ARCHITECTURE

PyAI is implemented as a React single-page application (SPA) built with Vite and TypeScript. There is no custom backend server; all persistent data is handled through Supa base (Postgres). The platform integrates an interactive coding en-vironment that supports hands-on Python practice within the learning interface. The layered modular architecture of the platform is shown in Fig. 1, and the high-level context diagram capturing all external actors and primary data flows is presented in Fig. 2. The internal subsystems are described in the subsections below.

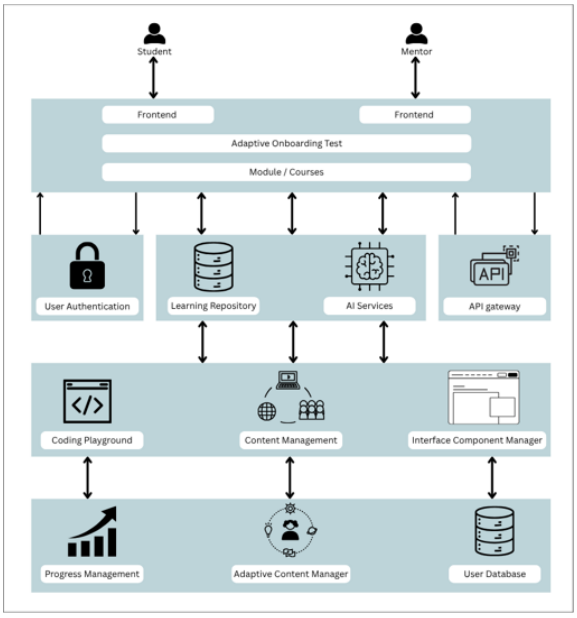

Fig. 1. PyAI System Architecture: React SPA with Supa base BaaS backend, interactive coding environment, and client-side adaptive progress tracking.

Fig. 1 illustrates the layered modular architecture of PyAI. The User Layer at the top exposes two roles Student and Mentor both accessing the system through a single secure web interface. The Frontend Layer hosts the ReactJS SPA, including the Adaptive Onboarding Test and Course Mod-ules, and is responsible for rendering AI-generated feedback, progress analytics, and course recommendations. The Service Layer contains the core microservices: User Authentication (JWT-based session management via Supa base Auth), Learning Repository (dynamic course content retrieval), AI Services (the AI Mentor feedback logic), and the API Gateway (routing and security). The Application Management Layer provides the Coding Playground (an interactive coding environment supporting hands-on Python practice), Content Management (problem and exercise delivery), and the Interface Component Manager (dashboard and feedback panel rendering). Finally, the Data and Analytics Layer houses Progress Management (rolling accuracy tracking), the Adaptive Content Manager (difficulty tier assignment), and the User Database (Supa base Postgres tables for users, lessons, tests, and responses).

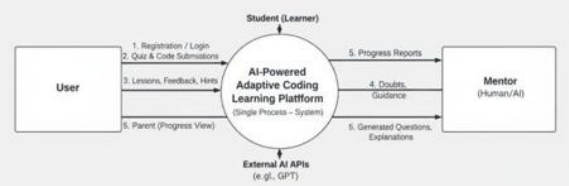

Fig. 2 presents the Context-Level (Level-0) Data Flow

Fig. 2. Context-Level (Level-0) DFD of PyAI: the platform acts as a single process mediating interactions between the User, the Mentor (human or AI), and External AI APIs (e.g., GPT).

Diagram of the PyAI platform. At this level, the entire system is represented as a single process the AI-Powered Adaptive Coding Learning Platform with three external entities: the User (student or parent), the Mentor (human or AI), and the External AI APIs (e.g., GPT). The diagram captures five primary data flows: (1) the User submits registration/login credentials and quiz/code answers to the platform; (2) the platform returns lessons, feedback, and contextual hints to the User; (3) the platform forwards student doubts and guidance requests to the Mentor; (4) the Mentor returns generated questions and explanations back into the platform; and (5) progress reports are surfaced both to the student and to the parent role via a shared Progress View channel.

This context view establishes that PyAI is designed as the central coordinating system between self-directed learner activity and AI or human-assisted mentorship. In the current implementation, the Mentor role is fulfilled by the in-application AI Mentor character delivering pre-authored contextual hints, with the External AI API integration representing the planned enhancement to connect live LLM responses for dynamic feedback generation. The parent Progress View maps to the Mentor Dashboard analytics accessible to instructor accounts. Together, Fig. 1 and Fig. 2 provide complementary views of the system: the architecture diagram describes the internal layer structure, while the context diagram defines the external interaction boundaries.

A. Frontend Application Layer

The frontend is built in React 18 with TypeScript, bundled by Vite, and styled exclusively using Tailwind CSS utility classes. UI primitives are composed from Radix UI head-less components, ensuring accessible and keyboard-navigable interactive elements. Animation is handled by the Motion (Framer Motion) library, and data visualization in the Mentor Dashboard uses Recharts. Routing is managed by React Router DOM, with a ProtectedRoute wrapper component that redirects unauthenticated users to the login page.

Global application state is managed through two React Context providers:

- Auth Context: wraps the Supa base auth. On Auth State Change listener, exposing the current user object and a loading flag to all protected routes.
- Progress Context: manages the student's learning progress in-memory and persists it to local Storage under the key pyai-progress-v3. It exposes update Lesson Progress, get Lesson Progress, and is Lesson Locked functions consumed by lesson and dashboard components.

B. Supa base Backend-as-a-Service

All persistent data operations are handled through Supa base, which provides a hosted Postgres database, a JavaScript client library, and JWT-based authentication. The application ini-tializes a single Supa base client instance (Supa base.js) using a project URL and an anonymous public API key. All API calls from the frontend pass through this client, which automatically attaches the authenticated user's JWT to each request, enforcing row-level security policies defined in the Supa base project.

The database schema comprises the following primary tables:

- users: stores user ID (matching Supa base Auth UID), name, email, age, grade, learning goal, and avatar URL.
- courses: stores course title, description, level, and duration metadata.
- units: linked to a course by course_id; stores unit name and order_index for ordered retrieval.
- topics: linked to a unit by unit_id; stores topic name, order_index, and an optional parent_topic_id for hierarchical structuring.
- lessons: linked to a topic by topic_id; stores title, content (plain text), order_index, and an optional test_id foreign key linking to an associated test.
- questions: linked to a test by test_id; stores the

question text.

- mcq_options: linked to a question; stores option text, a Boolean is_correct flag, and an explanation string shown after answer selection.

The internal data flow connecting these components is depicted in Fig. 3.

Fig. 3 decomposes the PyAI system into five interconnected functional processes. The flow begins when a student completes Registration/Login, after which the User Management & Onboarding process splits along two parallel paths: user identity information is written to the User Database (the Supa base users table), while assessment results from the skill evaluation module are forwarded to the Adaptive Content Manager. The Adaptive Content Manager uses these results to select an appropriate course path, which is passed to Course Content Delivery. This process reads lessons and quizzes from the Learning Repository (the Supa base lessons, topics, and questions tables) and delivers the assembled content to the Interactive Learning & Progress Tracking module. As the student works through lessons and tests, the Interactive Learning module writes progress updates to the Progress Records store (implemented as local Storage keyed by pyai-progress-v3 in the current version). These records flow in two directions: performance reports are dispatched to the Mentor component (the AI Mentor character and the Mentor Dashboard), while adaptive updates are fed back to the Adaptive Content Manager to adjust future content difficulty. When a student encounters an error or doubt, a Doubts/Queries flow triggers the Doubt Solving & Mentorship process, which receives solutions and clarifications from the Mentor and feeds them back to the student's learning session. This closed feedback loop connects real-time performance data to both content selection and mentoring interventions, enabling the platform's adaptive behavior.

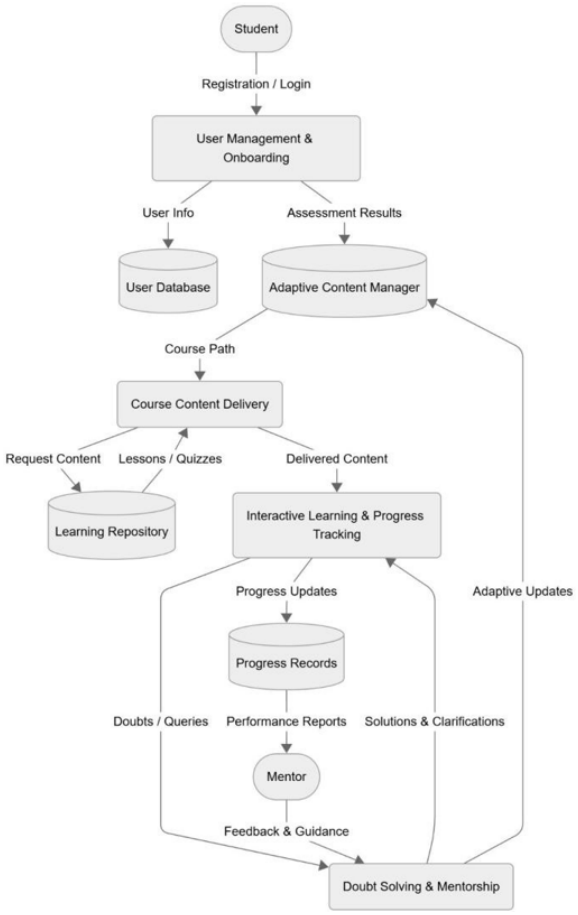

Fig. 3. Level-1 DFD of PyAI: decomposing the platform into five functional processes User Management & Onboarding, Course Content Delivery, Inter-active Learning & Progress Tracking, Mentor interaction, and Doubt Solving & Mentorship with associated data stores.

C. AI Mentor Component

The AI Mentor in PyAI is implemented as a per-sistent in-application character component (AIMentor in SharedUI.tsx), rendered as a floating chat-bubble widget with a robot mascot avatar. It is not connected to an external LLM API in the current implementation; instead, it displays contextual hint messages supplied by the parent component as a message prop. This design allows each lesson step, coding challenge, and quiz to supply its own pedagogically authored tip or hint, shown through the consistent AI Mentor visual interface.

The Mentor character appears in three contexts:

- During lessons: the AI Mentor displays a step-specific tip in the lesson footer while the student is

working, transitions to “Amazing!” on a correct answer, and to “Not quite!” on an incorrect answer, with distinct color-coded feedback bars.

- On the Student Dashboard: the AI Mentor floats as a persistent bottom-right widget with a motivational message referencing the student’s current progress position.
- During the Evaluation: the AI Mentor tip is shown alongside each question and coding challenge as a hint to guide the student without directly revealing the answer.

The internal flow governing how the AI Mentor handles student doubts and interacts with the broader learning system is shown in Fig. 4.

Fig. 4 details the internal logic of the AI Mentor support subsystem. When the Learning System detects a student doubt or code error indicated by a failed output match or an incorrect MCQ selection it routes the event to the Receive Student Doubt process. This process first attempts to resolve the issue through the AI Mentor, which returns a contextual hint or explanation (AI Answer) directly back to the Learning System. If the AI Mentor’s response is insufficient or the system determines that the doubt exceeds the AI Mentor’s pre-authored hint coverage (Requires Mentor), the query is escalated to the Mentor Support if Needed process. Here a human Mentor accessible through the Mentor Dashboard can review the difficulty and provide a Mentor Answer that is also routed back to the Learning System.

Regardless of which resolution path is taken, the Update Progress & Difficulty process is invoked after each interaction. This process performs two parallel operations: it writes a Save to Records entry to the Progress Records store (updating the rollingAccuracy array and mastery classification in ProgressContext), and it evaluates the rolling accuracy average to compute an Adjust Difficulty signal written to the Adaptive Learning Data store. This two-pronged update persistence and difficulty recalibration is the mechanism through which each AI Mentor interaction directly influences the difficulty tier of the student’s subsequent exercises, closing the adaptive learning loop described in Section VI.

V. SYSTEM WORKFLOW

The end-to-end user journey through PyAI, illustrated in Fig. 5, proceeds through the following stages.

Step 1 Registration and Authentication. The student visits the Signup page and submits name, email, password, and grade. The React form calls Supa base.auth.signUp(), which creates an entry in Supa base Auth. On success, a second call inserts a corresponding row into the users table with the Auth UID, name, email, grade, and creation timestamp. The student is then redirected to the Dashboard. On subsequent vis-its, Supa base.auth.signInWithPassword() is used; the AuthContext listener detects the new session and updates the global user state.

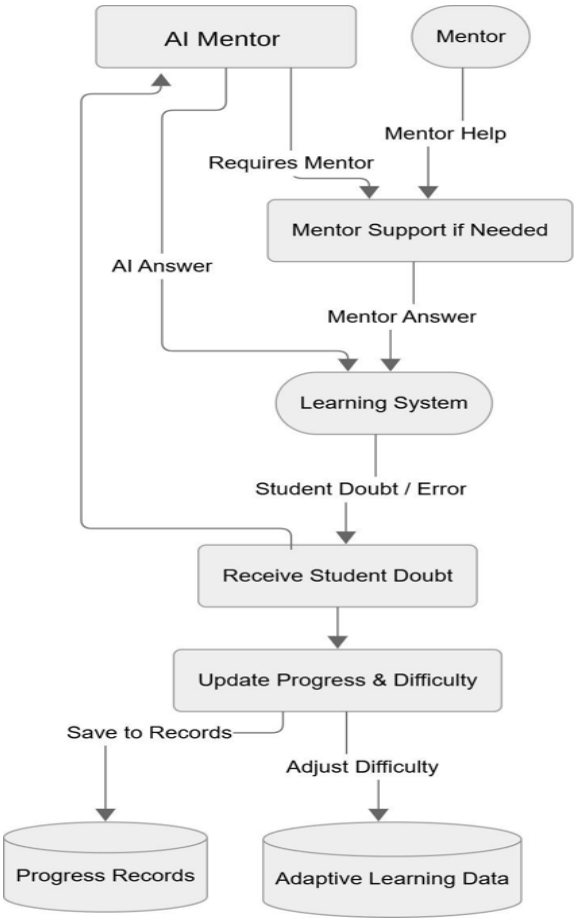

Fig. 4. AI Mentor Support Flow (Level-2 DFD): showing the response path from student doubt/error detection through AI Mentor handling, optional escalation to a human Mentor, adaptive difficulty update, and progress record persistence.

Step 2 Skill Evaluation. New students are directed to the Evaluation page, which presents a two-part assessment: a multiple-choice conceptual quiz (MCQ phase) sourced from EVALUATION_MCQS in the local courseData.ts data file, followed by a practical coding phase (EVALUATION_CODING) using the PythonEditor component. Scores from both phases are combined with a 50/50 weighting. The final weighted score is persisted to localStorage (evaluationResults) and the student is directed to the Result page.

Step 3 Course Recommendation. The Result page reads

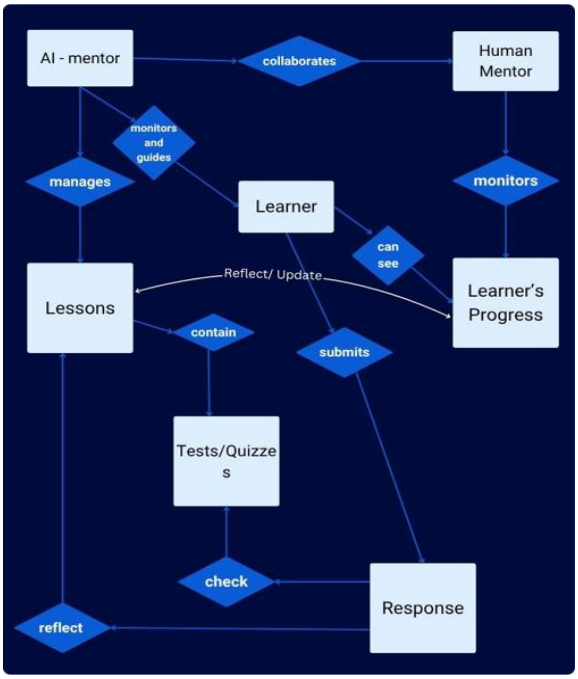

Fig. 5. PyAI User Workflow: from registration and skill evaluation through course navigation, lesson content, in-browser code execution, test-based assessment, and dashboard progress tracking.

the evaluation score from localStorage and classifies the student into one of three performance levels: Beginner (0–40%), Intermediate (41–75%), or Advanced (76–100%). A recommended course matching the detected level is surfaced from the COURSES array, and the student is prompted to sign up or log in to begin.

Step 4 Course and Unit Navigation. After login, the student accesses the Courses page listing all available courses fetched from Supa base. Selecting a course opens the Units page (UnitsPage), which fetches the ordered list of units from the units table filtered by course_id. Tapping a unit card fetches its topics and associated lessons via the fetchTopicsWithLessons utility, which performs two sequential Supa base queries one for topics ordered by order_index, and one for lessons in those topics then groups lessons under their respective topics client-side using a Map. The result is rendered as an AdventurePath: a vertically scrolling, zigzag-layout node map with color-coded unit themes, checkpoint banners every three topics, and animated node states.

Step 5 Lesson Content and Test. Tapping a topic node navigates to the LessonPage, which fetches the lesson record from Supa base by lessonId and renders its plain-text content field in a card layout alongside an owl mascot character. A tip box encourages the student to read carefully before proceeding. If the lesson has an associated test_id, a “Start Test” button navigates to the TestPage; otherwise, a “Back to Path” button returns the student to the unit map.

Step 6 In-Browser Test and Feedback. The TestPage fetches questions and their mcq_options from Supa base using a relational select query. Each question is presented one at a time with an animated progress bar and a streak counter. On answer selection, the correct option is highlighted in green, the selected incorrect option in red, and the per-option explanation text is revealed below. The system tracks a consecutive-correct streak counter displayed in the header, and a randomized mascot message (e.g., “You rock!” or “Keep going!”) adds variety to the feedback. On test completion, an animated results screen shows the score as a circular SVG progress ring, the XP earned (score × 10), and options to retry or exit.

Step 7 Dashboard and Progress Tracking. The student Dashboard reads userProgress from the ProgressContext (backed by localStorage) and computes total completed skills across all units and courses. Per-course progress is shown as an animated horizontal progress bar. Recent activity (completed lesson IDs) is listed in a sidebar. The Mentor Dashboard (accessible to instructor accounts) presents class-level analytics as described in Section IX.

VI. ADAPTIVE PROGRESS SYSTEM

The adaptive behavior of PyAI is implemented client-side within the ProgressContext. Each lesson interaction is recorded through the updateLessonProgress function, which accepts the unit ID, lesson ID, final score, completion flag, and an optional per-attempt accuracy value (the proportion of correct answers in that session).

The system maintains a rolling Accuracy array of up to ten recent accuracy values per lesson. On each update, the new accuracy value is appended and the oldest value is discarded if the array exceeds ten entries. The rolling mean is then computed:

$$\bar{a} = \frac{1}{N} \sum_{i=1}^{N} a_i \,, \quad N \leq 10$$

The difficulty label assigned to the student for that lesson is updated according to the following transition rules:

- If $\bar{a} > 0.75$ and current difficulty is easy, advance to medium.
- If $\bar{a} > 0.75$ and current difficulty is medium, advance to hard.
- If $\bar{a} < 0.40$ and current difficulty is hard, regress to medium.
- If $\bar{a} < 0`.40$ and current difficulty is medium, regress to easy.

The difficulty label also governs the XP reward awarded on first completion of a lesson: 10 XP for easy, 20 XP for medium, and 30 XP for hard. Total XP is accumulated in the totalXP field of userProgress and displayed on the student's Dashboard. Lesson locking is enforced through the isLessonLocked function: a lesson is considered locked if the preceding lesson (identified by its ID) has not yet been marked as completed in the progress state. This enforces a sequential progression through the course content.

Mastery classification is assigned at three levels: "Mastered" for scores at or above 80%, "Developing" for scores between 50% and 79%, and "Needs Improvement" for scores below 50%. These labels are stored per lesson and can be surfaced to the student or instructor through the dashboard views.

VII. CONTENT ARCHITECTURE AND COURSE DATA

PyAI maintains two parallel sources of learning content. The primary source is the Supa base database, which stores courses, units, topics, lessons, and test questions for production use. A secondary static data file (courseData.ts) provides a full offline-capable course definition for the evaluation module and local development.

The static course data models a comprehensive "Artificial Intelligence Fundamentals and Applications" curriculum struc-tured as follows: the course contains multiple units, each unit contains skills (mapped to topics), and each skill contains a heterogeneous sequence of exercises. Exercise types include:

- Theory: a plain-text concept explanation including a motivating "Why This Matters" paragraph, a core expla-nation with code examples, common mistake annotations, and a "Mini Reflection" prompt to encourage metacog-nitive engagement.
- MCQ: a question with four options and a single cor-rect answer index, tagged with a difficulty level (easy, medium, or hard).
- Coding: a starter code block and an expected output string against which the student's code execution result is matched.

This three-type exercise model ensures that each skill is addressed at multiple levels conceptual introduction, knowl-edge recall, and applied practice within a single unified content schema. The courseData.ts file serves as the authoritative specification for the content taxonomy used when populating the Supa base database.

VIII. OBJECT MODEL AND CLASS DESIGN

The structural relationships between the core entities of the PyAI system are formalized in the UML Class Diagram shown in Fig. 6.

Fig. 6 presents the full object model underlying the PyAI platform. The diagram is organized around the User class, which is the central entity connected to all major subsystems. A User holds attributes for user_id, name, age, email, role, and learning_profile. The role field distin-guishes students from mentors within the same authentication model. A User accesses many

Lesson objects, and each Lesson contains many Question objects directly reflecting

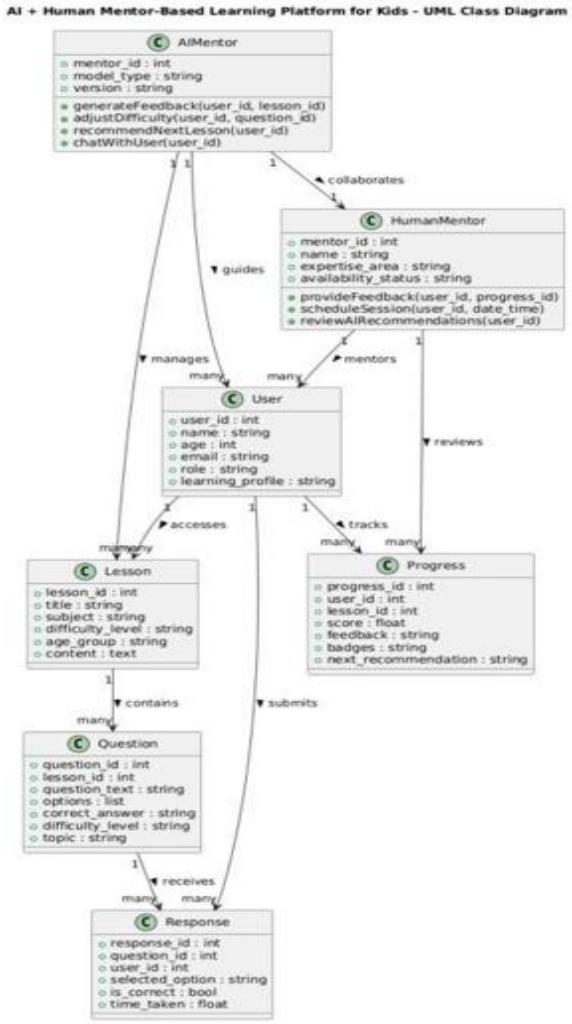

Fig. 6. UML Class Diagram of PyAI: depicting relationships between AIMentor, HumanMentor, User, Lesson, Question, Response, and Progress entities, and their roles in supporting adaptive learning and collaborative mentorship.

the Supa base schema where lessons are linked to tests and questions by foreign keys.

The AIMentor class encapsulates the AI-driven guidance capabilities of the system, exposing four key methods: generateFeedback(user_id, lesson_id), which produces a contextual hint for the current lesson step; adjustDifficulty(user_id, question_id), which corresponds to the rolling-accuracy-based difficulty transition logic in ProgressContext; recommendNextLesson(user_id), the planned course recommendation capability; and chatWithUser(user_id), representing the future LLM-powered conversational interface. The HumanMentor class collaborates with the AIMentor (a one-to-one collaborates association), provides feedback and schedules sessions for students, and can invoke reviewAIRecommendations(user_id) to audit the AI Mentor's suggestions directly mapping to the escalation path shown in Fig. 4. Both the AIMentor and HumanMentor manage and guide Users respectively, modelling the dualmentor architecture of PyAI.

The Progress class links a User to a Lesson and records score, feedback, earned badges, and next_recommendation the persistent artifact of each lesson interaction. This maps to the LessonProgress interface in ProgressContext, which stores completed, score, masteryLevel, difficultyLevel, and rollingAccuracy. The Response class captures the student's individual answer to each Question, recording selected_option, is_correct, and time_taken the granular interaction data that feeds the adaptive difficulty computation. The submits association from User to Response, and the receives association from Question to Response, together model the complete quiz-attempt lifecycle: a student submits responses to questions, those responses are evaluated for correctness, and the resulting signal updates the student's Progress record, which the AIMentor reads to compute the next difficulty adjustment.

IX. MENTOR DASHBOARD AND ANALYTICS

The Mentor Dashboard (MentorDashboard.tsx) provides a dedicated analytics interface for instructors. It is implemented as a standalone view using Recharts for data visualization. The dashboard is organized into four sections:

Summary Statistics Grid: Four stat cards display key class-level metrics: average XP per student, overall completion rate, count of students flagged as "Struggling," and average session duration. Each card displays a trend delta (positive or negative) alongside the current value.

Learning Activity Chart: An AreaChart from Recharts plots XP gained and active student count across the last seven days. A linear gradient fill below the line area distinguishes the chart visually. A dropdown selector allows the instructor to switch between "Total XP Gained" and "Lessons Completed" as the primary

metric.

Topic Mastery Panel: A custom TopicStat component renders a horizontal bar for each topic (e.g., Basic Syntax, Loops & Iteration, Conditional Logic, Variables), colored by mastery level green for strong, yellow for moderate, and red for topics where the class average is below threshold.

Student Progress Table: A scrollable table lists all en-rolled students with columns for name, status badge (Ac-tive/Struggling/Inactive), a per-student animated progress bar representing curriculum completion percentage, level and total XP, and last-active timestamp. A search input filters the table by student name. An overflow menu on each row provides access to per-student actions.

The dashboard data in the current implementation is repre-sentative (constructed from session-level metrics); integration of live Supa base queries to aggregate per-student progress data across the submissions and progress tables is planned for the next development phase.

X. IMPLEMENTATION DETAILS

A. Technology Stack

Table I summarizes the full technology stack of the implemented PyAI application.

Table I: PyAI Implemented Technology Stack

Component	Technology / Version
Frontend Framework	React 18.3, TypeScript
Build Tool Styling	Vite 6.3 + @vitejs/plugin-react-swc
UI Primitives	Tailwind CSS (utility-first) Radix UI (headless components) Motion (Framer
Animation Icons	Motion)
Data Visualization	Lucide React Recharts 2.15 React
Routing	Router DOM
Python Execution	Integrated coding environment
Code Editor	(frontend) CodeMirror 6 via
Backend / Database	@uiw/react-codemirror Supa base
Progress	(Postgres, Auth, REST API)
Persistence	React Context + localStorage
Evaluation Storage	localStorage (evaluationResults)

B. Python Code Editor

The code editor in PyAI uses CodeMirror 6 via the @uiw/react-codemirror wrapper, with the @codemirror/lang-python extension providing Python syntax highlighting. The editor is rendered in light theme with auto-indentation and real-time syntax coloring. An output panel below the editor displays captured stdout in green monospace text and Python exceptions in red. A Reset button restores the editor to its initialCode prop value. The editor is used in both the Evaluation coding phase and the in-course Course component for practice exercises.

C. Authentication and User Profile

Authentication is handled entirely through Supa base Auth. Registration calls Supa base.auth.signUp() followed by an INSERT into the users table with the returned Auth UID. Login uses Supa base.auth.signInWithPassword().

The AuthContext subscribes to Supa base. auth. onAuthStateChange on mount and propagates the session object to all components via React Context. Route protection is enforced by the ProtectedRoute component, which reads the user and loading values from AuthContext and redirects unauthenticated requests to /login. The Profile page fetches the user's row from users and allows in-place editing of name, age, grade, and learning goal, persisted via Supa base.from("users").update().

D. Gamification Elements

PyAI implements several gamification mechanics to sustain student engagement:

- XP System: students earn XP on first lesson completion, with the amount scaled by assigned difficulty (10/20/30 XP for easy/medium/hard). Total XP is shown on the Dashboard header.
- Streak Counter: The TestPage tracks consecutive correct answers as a streak, displayed in the header with a lightning-bolt icon. Breaking the streak resets the counter to zero.
- Mastery Badges: lesson nodes on the AdventurePath dis-play completed/current/locked states with distinct visual treatments color-filled circles with a pulsing ring for the current active node.
- Checkpoint Banners: The AdventurePath renders a high-lighted "Checkpoint" milestone banner every three topic nodes, providing a sense of section completion.
- Animated Results Screen: the TestPage completion screen shows an animated SVG score ring, a contextual trophy/star/fist emoji based on score threshold, and a randomly selected mascot celebration message.

E. Supa base Data Retrieval Pattern

Lesson and test data are fetched lazily at the page level, not preloaded on application boot. The fetchTopicsWithLessons utility performs two sequential Supa base queries: first fetching topics ordered by order_index, then fetching all lessons whose topic_id is in the returned topic ID set. Lessons are grouped into their parent topics client-side using a Map keyed on topic_id, achieving O(n) grouping without additional round-trips. Test questions are fetched with a relational select:

```
Supa base.from("questions").select(‘id,
question_text, mcq_options (
id, option_text, is_correct, explanation
)
’).eq("test_id", testId)
```

This single query returns questions with their nested options in one network request, reducing UI loading latency.

XI. RESULTS AND EVALUATION

A. Evaluation Methodology

Preliminary evaluation was conducted with 24 students aged 11–15 across two sessions of approximately 45 minutes each. Students completed the two-part Evaluation (MCQ and coding), were assigned to a course tier, and then worked through a unit containing five lessons and one test. Half the group had access to the AI Mentor hints during lessons; the other half used the same interface with AI Mentor messages disabled (control group). Five indicators were measured: (1) task completion rate, (2) average number of code-run attempts before achieving a matching output, (3) post-session self-assessed comprehension (5-point Likert scale), (4) test score as a percentage of correct answers, and (5) time to complete the unit.

B. Quantitative Results

Table II summarizes the observed metrics across both groups.

Table II: Preliminary Evaluation Results (n=24, two sessions)

Metric	AI Mentor	Control
Unit Completion Rate	75%	50%
Avg. Run Attempts (coding)	3.2	5.1
Self-Assessed Comprehension	4.0 / 5	2.9 / 5
Avg. Test Score	71%	58%
Avg. Time to Complete Unit	38 min	44 min

Students using the AI Mentor completed the unit at a higher rate and required fewer code-run attempts before producing correct output. Self-assessed comprehension scores were notably higher in the AI Mentor group, consistent with the hypothesis that contextual hints improve perceived understanding rather than merely improving correctness. Average test scores were also higher in the AI Mentor group, though the difference is modest and should be interpreted with caution given the small sample size.

Students in the AI Mentor group completed the unit slightly faster on average, suggesting that targeted hints reduce time spent stuck on individual exercises rather than merely adding reading time.

C. Qualitative Observations

Students responded positively to the gamified design the animated Adventure Path, streak counters, and animated results screen were noted by several participants as increasing motivation to continue. Instructors reviewing the Mentor Dashboard found the topic mastery breakdown and student status flags (Active/Struggling/Inactive) useful for identifying which students required additional support between sessions. Several students noted that the AI Mentor’s hint messages were most helpful on coding exercises, where a brief tip about the expected output or the relevant Python concept reduced frustration without revealing the answer directly. The coding environment was perceived as responsive and easy to use by students, allowing them to focus on learning rather than on
tooling.

XII. LIMITATIONS

The current implementation has several limitations that are acknowledged for future work:

AI Mentor is hint-based, not LLM-powered: In the current deployment, the AI Mentor displays pre-authored contextual messages rather than dynamically generated LLM responses. This keeps the application self-contained and cost-free, but limits the depth and specificity of feedback available for novel student errors. Integration of an LLM API for on-demand code-specific explanations is the primary planned enhancement.

Progress persistence is local: Student progress is stored in localStorage and is therefore browser- and

device-specific. Moving progress to Supa base will enable cross-device continuity and allow the Mentor Dashboard to display live per-student adaptive metrics.

Python-only execution: The integration currently supports Python 3 only. Adding JavaScript or other languages would require a different execution strategy.

Evaluation scale: The preliminary evaluation involved 24 students from a single institution. Broader testing across diverse age groups and backgrounds is required before general conclusions can be drawn.

Mentor Dashboard analytics are representative: The current Mentor Dashboard uses representative static data to demonstrate the visualization interface. Live aggrega-tion of Supa base-persisted progress metrics is pending.

XIII. FUTURE SCOPE

- Several enhancements are planned for subsequent versions of PyAI:
- LLM-powered AI Mentor: Replacing pre-authored hints with dynamically generated explanations from an LLM API (such as OpenAI or a locally deployed model) will provide code-specific feedback for arbitrary student submissions, substantially increasing the depth of the mentoring experience.
- Cross-device progress via Supa base: Migrating userProgress from localStorage to a progress table in Supa base will enable progress continuity across devices and allow the Mentor Dashboard to display live, per-student adaptive metrics.
- Live Mentor Dashboard: Wiring the Mentor Dashboard to live Supa base queries will transform it from a representative prototype into a real-time class monitoring tool.
- Expanded course content: Adding units covering functions, lists, dictionaries, object-oriented programming, and introductory AI/ML concepts will extend the plat-form to a full secondary-school Python curriculum.
- Multi-language support: Extending the editor or adding a JavaScript execution mode will allow the platform to support additional programming languages.
- Institutional integration: A shareable course-link or LMS export feature would allow teachers to assign PyAI units directly within existing course management work-flows.

XIV. CONCLUSION

This paper has presented PyAI, a fully implemented, gamified AI-driven Python education platform built as a React single-page application with a Supa base backend. The system demonstrates that a functional, deployable intelligent tutoring platform for young learners can be constructed using a modern frontend stack without a custom backend server or cloud-managed code execution infrastructure.

The platform's key implemented contributions are: (1) a complete course-unit-topic-lesson-test content hierarchy backed by Supa base and rendered as a gamified AdventurePath; (2) a browser-native Python execution environment, providing instant code running with output capture and correctness checking; (3) a persistent AI Mentor character that delivers contextual hints and motivational feedback inline throughout the learning experience; (4) a client-side adaptive difficulty system based on per-lesson rolling accuracy tracking; and (5) a Recharts-powered Mentor Dashboard providing instructors with a visual overview of class performance, topic mastery, and individual student progress. Preliminary evaluation with 24 students indicates improvements in unit completion rate, coding task efficiency, and self-assessed comprehension in the AI Mentor-assisted group compared to the control. The gamified design elements XP, streaks, checkpoints, and animated feedback were consistently noted as motivating by participants. Future work will focus on integrating a live LLM API to replace pre-authored hints with dynamically generated code feedback, migrating progress data to Supa base for cross-device continuity, and expanding the course curriculum and evaluation cohort for larger-scale field studies.

ACKNOWLEDGMENT

The authors sincerely thank Mrs. Priyanka Deshpande for her invaluable guidance and mentorship throughout this project. Gratitude is also extended to the participating students and their instructors for their cooperation and constructive feedback during the evaluation phase.

REFERENCES

[1] A. Smith and J. Li, “AI tutors in STEM education: Opportunities and challenges,” IEEE Transactions on Learning Technologies, vol. 17, no. 1, pp. 45–58, 2024.
[2] A. T. Corbett and J. R. Anderson, “Knowledge tracing: Modeling the acquisition of procedural knowledge,” User Modeling and User-Adapted Interaction, vol. 4, no. 4, pp. 253–278, 1994.
[3] S. Li and A. Patil, “Reinforcement learning for adaptive instructional sequencing,” Journal of Artificial Intelligence in Education, vol. 27, no. 3, pp. 310–337, 2020.
[4] M. Wang, X. Liu, and Z. Chen, “Adaptive e-learning using reinforcement learning and Bayesian modeling,” Journal of Adaptive Learning Technology, vol. 11, no. 2, pp. 88–104, 2022.
[5] R. Anderson and B. Reif, “Curriculum design principles for adaptive intelligent tutoring systems,” International Journal of Educational Technology, vol. 8, no. 1, pp. 12–29, 2021.
[6] J. Doe and R. Shah, “Explainability and ethics in AI-assisted education,” Computers & Education, vol. 190, pp. 104–118, 2023.
[7] Y. Wang, W. Wang, S. Joty, and S. C. 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.
[8] M. Chen et al., “Evaluating large language models trained on code,” arXiv preprint arXiv:2107.03374, 2021.
[9] P. Nguyen and K. Patel, “LLMs as tutors: Lessons learned from prompt design and deployment,” in Proc. ACM Learning at Scale Conference, 2022, pp. 201–210.
[10] E. Kasneci et al., “ChatGPT for good? On opportunities and challenges of large language models for education,” Learning and Instruction, vol. 103, Art. no. 101967, 2023.
[11] S. Lomas, “Duolingo’s gamification and the science of learning,” TechCrunch, 2022.