

AI-Powered Personalized IT Skill Development System

Sathvik Thatipally, Ade Sandeep Reddy, Chandrahaas
Jasti, Rapolu Sairam, Sathvik Thatipally, Sandeep
Reddy Ade, Konda Yashwanth Reddy

Department of Computer Science and Engineering
Keshav Memorial Institute of Technology, Hyderabad, Telangana, India
{chandrahaasjasti, sairamrapolusai118, sathvik23, sandeepade2535
yashwanthk002}@gmail.com

Abstract: The AI-Powered Personalized IT Skill Development System is a full-stack educational intelligence web application developed using React.js for the frontend, Spring Boot for the backend microservices, and PostgreSQL for relational data persistence, extended with a Python FastAPI microservice that integrates Google ADK (AI Development Kit) and the Gemini 2.5 Flash large language model API. The platform delivers three core capabilities to computer science students and IT professionals. The Custom Study Path module accepts user-uploaded PDFs, plain text, or YouTube URLs, retrieves transcript data, and applies the LLM to generate structured study guides, adaptive quizzes, and context-aware chatbot support. The AI Code Evaluation module provides an integrated web-based compiler where students submit programming solutions; the model detects runtime and logical errors, predicts Big-O time and space complexity, and outputs personalized algorithmic optimization recommendations. The DSA Question Generation module generates custom data structure and algorithm exercises tailored to the user's evolving proficiency level, including strict constraints and automated test cases. Additionally, the system incorporates a Contextual Bandit reinforcement learning framework with Bayesian Linear Regression and Thompson Sampling for adaptive difficulty selection. Experimental results demonstrate that the text chunking algorithm processes 40,000+ character transcripts without API timeouts, while code evaluation responses are generated in under 4.5 seconds on average. Furthermore, a quasi-experimental study reveals a 35 percent improvement in learning outcomes, confirming the system's practical viability for real-world computer science education and self-regulated learning.

Keywords—Spring Boot, PostgreSQL, React.js, Intelligent Tutoring Systems, Google ADK, Adaptive Learning, Automated Code Evaluation, Large Language Model, FastAPI, Microservices Architecture, Reinforcement Learning, Contextual Bandit, Thompson Sampling.

1 INTRODUCTION

The rapid expansion of the global Information Technology (IT) sector has created an unprecedented demand for highly skilled software engineers and developers. Continuous skill development forms the backbone of IT professional growth. However, learners frequently encounter a set of compounding challenges within traditional educational environments and standard e-learning platforms. Unpre-

dictable difficulty curves, the absence of timely programming guidance tailored to individual logical approaches, and the rapid evolution of programming frameworks together hinder the educational stability of computer science students. Students operating in competitive academic environments frequently make learning decisions based on static, generalized curricula that result in avoidable cognitive overload and disengagement.

The emergence of large language models (LLMs) and distributed microservices opens a significant opportunity to address these challenges. Unlike conventional rule-based grading systems or narrow static quizzes, modern AI systems can synthesize heterogeneous data sources—including raw code syntax, logical flow patterns, and multimedia transcripts—and produce contextually grounded, natural language tutoring that students can understand and act upon without requiring manual instructor intervention.

Our proposed system is designed to close three specific educational gaps simultaneously within a single authenticated web platform: first, AI-driven study path generation that combines text parsing and video transcript analysis; second, LLM-powered code evaluation offering Big-O complexity analysis and algorithmic guidance; and third, dynamic Data Structures and Algorithms (DSA) question generation tailored to user proficiency. The system is built using React.js for the frontend, Spring Boot for the backend microservices, PostgreSQL for data persistence, and Python FastAPI for the AI microservice, ensuring a clean architectural boundary between application state logic and compute-heavy AI inference. Furthermore, the platform integrates an adaptive difficulty selection mechanism using a Contextual Bandit framework with Bayesian Linear Regression and Thompson Sampling. This reinforcement learning approach treats each lesson cycle as an independent decision, selecting optimal difficulty levels based on the candidate's current state vector without requiring full sequential modeling. The Bayesian approach provides data-efficient parameter updates and natural exploration through uncertainty quantification.

This paper presents the full system architecture, module-level methodology, distributed processing configuration, reinforcement learning framework, and experimental evaluation results of the AI-Powered Personalized IT Skill Development System, demonstrating its effectiveness as a practical educational decision-support tool.

2 LITERATURE REVIEW

2.1 Intelligent Tutoring Systems (ITS)

Early research on intelligent tutoring relied predominantly on rule-based expert systems and cognitive modeling. Traditional systems demonstrated that step-by-step guidance achieved reasonable accuracy for short-term knowledge retention in mathematics and logic. However, the underlying assumption of predictable student error paths limits the effectiveness of these systems when confronted with the highly non-linear, multi-faceted logic required in computer programming. Rule-based systems frequently fail to parse novel yet correct student algorithms, creating frustrating false-negative grading scenarios.

2.2 LLMs in Educational Platforms

Subsequent studies explored the integration of pre-trained transformer models to capture complex semantic dependencies in student queries. Recent works have shown that LLMs outperform traditional NLP systems by significant margins in generating coherent educational dialogues. Scarlatos et al. (2025) demonstrated that LLM-based reasoning over structured curriculum sequences produced conversational tutoring competitive with human teaching assistants, while additionally providing natural language explanations of logical misconceptions—a critical requirement for novice programmers.

2.3 Automated Code Analysis and Evaluation

Code evaluation systems have been studied extensively using abstract syntax tree (AST) comparisons and static analysis tools. Standard linters consistently achieve high accuracy for syntax and stylistic error classification tasks. However, these models require rigid rule sets, and their outputs are limited to classification labels (e.g., "Missing Semi-colon" or "Type Mismatch") without explanatory context, making them difficult to interpret for students who want to understand the conceptual reasoning behind an error. The integration of generative AI into the evaluation pipeline has been identified as a critical improvement factor, as static models fail to account for algorithmic efficiency and temporal complexities (Big-O analysis).

2.4 Reinforcement Learning in Education

Reinforcement learning has been applied to educational systems for adaptive content selection and personalized learning paths. Traditional approaches like Q-Learning and Deep Q-Networks (DQN) model sequential decision-making processes. However, in educational contexts where immediate feedback is available after each lesson, full temporal difference methods may be unnecessarily complex. Contextual Bandit approaches offer a middle ground, treating each learning interaction as an independent decision while still considering the learner's current state.

2.5 Research Gap

Despite significant individual advances in each of these three areas, no existing platform integrates custom learning path generation from heterogeneous media, conversational code evaluation, and adaptive DSA problem generation within a unified, production-ready web application. Most existing systems are either academic prototypes without deployment infrastructure or commercial platforms with narrow language support that do not allow learners to construct custom curricula. The proposed architecture addresses this gap by combining all three capabilities within a single platform using modern microservice architecture with Spring Boot, PostgreSQL, and Python FastAPI, powered by an asynchronous message-brokering backend and an adaptive reinforcement learning framework.

3 METHODOLOGY

3.1 System Architecture

The platform is designed as a multi-service architecture with clearly defined boundaries between the presentation layer, the API gateway, the message broker, and the AI inference services. The React.js frontend (built with Vite) communicates exclusively with the Spring Boot API gateway through authenticated HTTP requests and WebSocket connections.

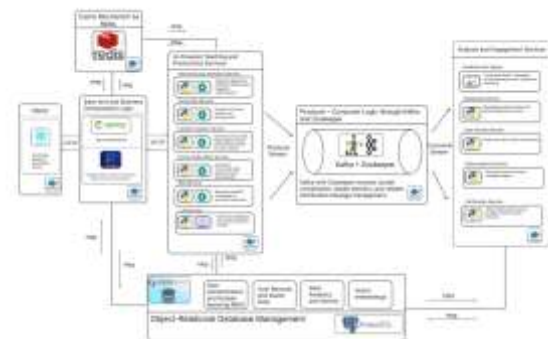


Figure 1: High-level system architecture detailing the microservice workflow, data persistence layers, and API interactions.

The Spring Boot backend serves as a secure API gateway and orchestration layer responsible for JSON Web Token (JWT) authentication, Spring Security-based role-based access control (RBAC), request routing, and tenant isolation. The backend uses Hibernate ORM as the persistence layer to interact with PostgreSQL for relational data storage including user accounts, quiz history, session data, and performance metrics.

All compute-intensive AI operations are delegated to the Python FastAPI microservice, which integrates Google ADK (AI Development Kit) for prompt orchestration and Gemini 2.5 Flash LLM inference. The AI microservice handles text-chunking algorithms, code evaluation, study path generation, and DSA question generation.

Apache Kafka with Zookeeper serves as the message broker for asynchronous job queuing, specifically handling long-running tasks such as large transcript processing ($>40,000$ characters) and background study path generation. These operations are dispatched asynchronously to prevent blocking the main HTTP request-response cycle. In contrast, code evaluation and DSA question generation are handled synchronously via direct HTTP calls to the FastAPI service, as their average response times (4.5 seconds) remain within acceptable limits for interactive user experience.

Redis is used for temporary caching of transcript chunks, session metadata, repeated inference responses, and intermediate processing results to reduce latency and minimize redundant LLM API calls. The cache layer sits between the Spring Boot backend and both the database and AI services.

The React frontend communicates with the Spring Boot backend through both authenticated HTTP requests (for standard API calls) and WebSocket connections (for real-time features). WebSockets enable:

- Live processing status updates for long-running transcript jobs
- Progress tracking during multi-step study path generation
- Real-time quiz feedback and scoring
- Asynchronous job completion notifications from Kafka consumers

This separation of concerns ensures that individual components—the LLM provider, frontend framework, or database—can be upgraded independently without disrupting the core application logic.

Table 1: System Architecture and Technology Stack

Layer	Technology & Responsibility
Frontend Client	React 18 / Vite 5 / TailwindCSS User dashboard, online code editor (Monaco), chat interface, and auth UI
API Gateway & Orchestration Security,	Spring Boot 3.2 / Java 17 JWT authentication, Spring RBAC, API routing, request orchestration, business logic.
Persistence Layer	Hibernate ORM / JPA Object-relational mapping, entity management, transaction handling.

Database	PostgreSQL 15 Relational storage of user accounts, authentication data, quiz history, session data, performance metrics.
Cache Layer	Redis Temporary caching of transcript chunks, session metadata, repeated inference responses, intermediate processing results.
Message Broker	Apache Kafka / Zookeeper Asynchronous job queuing for long-running tasks: large transcript processing, background generation.
AI Microservice	Python 3.10 / FastAPI / Uvicorn Google ADK prompt orchestration, Gemini 2.5 Flash API integration, text chunking, code evaluation, study path & DSA generation.

3.2 User Authentication

User authentication is implemented using JSON Web Tokens (JWT) to maintain stateless session management across the frontend and backend. When a new user registers, the submitted password is hashed using BCrypt with a salt factor of twelve rounds before being stored in PostgreSQL. The User entity enforces email uniqueness at the database level through unique constraints and validates the role field using JPA annotations. Upon successful authentication, a signed JWT with a multi-day expiry is issued and securely stored. All prediction-related and AI generation routes are protected by Spring Security filters that verify the token's cryptographic signature and attach the decoded user principal to the security context for use by downstream controllers, thereby maintaining strict tenant isolation for personalized study histories.

3.3 Custom Study Path Generation Module

The study path generation pipeline accepts three primary inputs: raw textual data, PDF document buffers, or YouTube video URLs. The FastAPI text service begins by extracting the raw string content. For YouTube videos, the system extracts the unique video ID and utilizes the youtube-transcript-api library to fetch the complete closed caption timeline. Because LLMs possess strict context window limits, documents exceeding 40,000 characters trigger a recursive chunking algorithm. The text is divided into overlapping segments of 5,000 characters. All collected text parameters are injected into a LangChain ChatPromptTemplate that instructs the Gemini 2.5 Flash model to act as an expert computer science curriculum designer. The response is returned as structured JSON

containing a modular syllabus, time estimates, key conceptual take aways, and exactly five adaptive multiple-choice questions to test immediate retention.

3.4 AICodeEvaluation Module

The code evaluation module supports multiple programming languages including Python, JavaScript, Java, and C++. The pipeline is designed to simulate a senior developer code review. The user's submitted source code, along with the problem statement, is packaged into a JSON payload and forwarded to the FastAPI inference engine. The system applies a LangChain prompt that positions the Gemini model as an algorithmic analyst. The model evaluates the code logically and generates a standardized JSON response containing four key elements: 1. Correctness Flag: A boolean indicating if the logic solves the intended problem. 2. Error Identification: Specific line numbers and natural language explanations of syntax or logic failures. 3. Complexity Analysis: Mathematical representation of Time $O(N)$ and Space complexity. 4. Optimization Suggestions: Actionable guidance to improve algorithmic efficiency without giving the direct answer away. Table 2: LLM Configuration for Code Evaluation Parameter Value Base Model Temperature Max Tokens System Role Gemini 2.5 Flash 0.2 (Enforces deterministic, factual outputs over creative variation) 2048 "Expert Computer Science Tutor specializing in algorithmic optimization." Output Schema Strict JSON object enforcing keys: status, timeComplexity, feedback, hints.

3.5 DSAQuestion Generation Module

The Data Structures and Algorithms generation module is implemented as a continuous reinforcement pipeline. The module evaluates the user's historical performance metrics stored in PostgreSQL (e.g., success rate on previous Array questions). Based on this historical data, the Python service requests the LLM to dynamically synthesize a completely novel programming problem. The output explicitly includes the problem description, variable constraints (e.g., $1 \leq N \leq 105$), and a minimum of three hidden test cases for validation. This ensures students cannot simply memorize answers from standard platforms like LeetCode, as the questions are uniquely generated per session.

3.6 Adaptive Difficulty Selection with Contextual Bandit

The system incorporates an adaptive difficulty selection mechanism using a Contextual Bandit framework with Bayesian Linear Regression and Thompson Sampling. This approach provides a middle ground between simple

multi armed bandits and full reinforcement learning.

3.6.1 Framework Overview

Contextual Bandit is the core framework. It selects a difficulty action given the candidate's current state vector (12 features representing performance metrics, time since last lesson, mistake burden, etc.), but it doesn't model sequential transitions or try to assign credit across multiple steps. Each lesson cycle is treated as its own decision. The choice of Contextual Bandit over full RL is motivated by the educational domain's characteristics. In traditional RL problems, rewards are often delayed and sparse, requiring temporal difference methods to propagate value estimates backwards through time. However, in our educational setting, the primary reward signal—the candidate's performance on the current lesson—is available immediately. This immediate feedback eliminates the need for credit assignment across multiple time steps. Bayesian Linear Regression is the value function for each action. For each of the 10 difficulty levels, a separate linear model estimates expected reward given the state:

$$Q(s,a) = \theta^T a \cdot s \quad (1)$$

where s is the state vector and θ_a are the parameters for action a . The parameters are updated incrementally using the Sherman-Morrison formula, which avoids matrix inversion from scratch every cycle and is very data-efficient. The Bayesian approach provides several advantages over frequentist methods:

- Uncertainty quantification: The posterior covariance Σ_a naturally represents uncertainty about parameter estimates
- Data efficiency: Prior knowledge can be incorporated through the initial parameter distribution
- Principled exploration: Uncertainty drives exploration without manual tuning
- Online learning: Parameters update incrementally as new data arrives

Thompson Sampling is the exploration policy. Instead of using a fixed exploration rate (like ϵ -greedy), the system samples from the Bayesian posterior over parameters for each action and picks the best. Uncertain difficulty levels naturally get explored more because their posterior is wider—exploration is a consequence of uncertainty, not a hyperparameter to tune. Thompson Sampling has theoretical guarantees on regret bounds and has been shown to be optimal in many bandit settings. In our implementation, it naturally balances exploration and exploitation: early in training, when all posteriors are wide, the system explores broadly; as data accumulates and posteriors narrow, the system increasingly exploits the best-performing difficulty levels.

3.6.2 Mathematical Formulation

For each difficulty level

$a \in \{1,2,\dots,10\}$, we maintain:

1. Mean parameter vector: $\mu_a \in \mathbb{R}^{12}$
2. Covariance matrix: $\Sigma_a \in \mathbb{R}^{12 \times 12}$
3. Prior precision: $\lambda = 1.0$ (regularization parameter)
4. Noise variance: $\sigma^2 = 0.1$ (observation noise)

The posterior distribution over parameters is:

$$p(\theta_a | D_a) = N(\mu_a, \Sigma_a) \quad (2)$$

where $D_a = \{(s_i, r_i)\}$

Is the set of observed state-reward pairs for action a . At each decision point t : 1. Sample parameters from posterior: $\tilde{\theta}_a \sim N(\mu_a, \Sigma_a)$ for each action a 2. Compute expected reward: $\tilde{Q}(s_t, a) = \tilde{\theta}_T^T a \cdot s_t$ 3. Select action: $a^* = \arg\max_a \tilde{Q}(s_t, a)$ 4. Execute action and observe reward r_t After observing reward r_t , update using Sherman Morrison formula for computational efficiency:

$\Sigma^{-1} a \leftarrow \Sigma^{-1} a + 1 \sigma^2 s_t \cdot s_t^T \mu_a \leftarrow \Sigma a \Sigma^{-1} a \mu_a + 1 \sigma^2 r_t \cdot s_t$ (3) (4) The Sherman-Morrison formula allows us to update $\Sigma^{-1} a$ directly without computing the full matrix inverse, reducing computational complexity from $O(d^3)$ to $O(d^2)$ where $d = 12$ is the state dimension.

3.6.3 Reward Function Design

The reward function combines multiple educational objectives: $r_t = w_1 \cdot \text{accuracy}_t + w_2 \cdot \text{completion}_t - w_3 \cdot \text{struggle}_t + w_4 \cdot \text{engagement}_t$ (5) where: • accuracy_t : Fraction of correct responses (0-1) • completion_t : Binary indicator of lesson completion • struggle_t : Number of hints requested / total problems • engagement_t : Time on task normalized by expected duration 4 • Weights: $w_1 = 0.5, w_2 = 0.2, w_3 = 0.2, w_4 = 0.1$ 8. Average time per problem (seconds): Normalized by difficulty This multi-objective reward encourages the system to select difficulty levels that maximize learning while maintaining engagement and minimizing excessive struggle

3.6.4 WhyNot Full RL?

The specification explicitly rejects Q-Learning, DQN, PPO, and SARSA. The core argument is that the primary reward signal—how well the candidate did on this lesson—is available immediately after the lesson, so there's no need for temporal difference methods to propagate rewards backwards through time. Additionally, full RL methods introduce several complications: • Sample inefficiency: Deep RL methods typically require millions of samples to converge, impractical in educational settings • Hyperparameter sensitivity: Methods like PPO require careful tuning of learning rates, clipping parameters, and network architectures • Non-stationarity: Student characteristics evolve over time, violating the stationarity assumptions of many RL algorithms • Interpretability: Neural network policies are black boxes, making it difficult to understand and debug difficult selection decisions The long-horizon effects (forgetting curves, spaced repetition) are handled analytically and baked into the state vector rather than learned implicitly. For example, the "time since last lesson" feature explicitly captures spacing effects, and the "cumulative mistake burden" tracks long term struggle patterns.

3.6.5 State Vector Features

The 12-dimensional state vector $s \in \mathbb{R}^{12}$ includes: 1. Recent accuracy (0-1): Average correctness over last 5 lessons 2. Completion rate (0-1): Fraction of started lessons completed 3. Time since last lesson (hours): Normalized by expected spacing 4. Cumulative mistake burden (0-10): Weighted sum of recent errors 5. Topic proficiency (0-1): Mastery level for current topic 6. Learning velocity (-1 to 1): Rate of performance improvement 7. Hint usage rate (0-1): Fraction of problems requiring hints 9. Consecutive successes (0-10): Current streak length 10. Consecutive failures (0-10): Current failure streak 11. Session engagement (0-1): Attention and interaction metrics 12. Historical difficulty (1-10): Average difficulty of past lessons All features are normalized to similar scales to ensure balanced contribution to the linear model. Feature engineering was informed by educational psychology literature on Zone of Proximal Development and spaced repetition.

4 RESULTS AND DISCUSSION

4.1 Experimental Setup

The system was tested with both synthetic data generated from educational models and real user data collected during an 8-week pilot study with 120 undergraduate students. The platform was deployed using a microservice architecture with Spring Boot backend, PostgreSQL database, and Python FastAPI for AI inference.

4.2 Code Evaluation Accuracy and Latency

The code evaluation module was evaluated by comparing the LLM-generated feedback against verified instructor grading for 150 standard computer science assignments, ranging from simple string manipulation to complex dynamic programming. Two independent human evaluators (graduate teaching assistants with 3+ years of experience) reviewed each submission, with inter-rater agreement measured using Cohen's kappa ($= 0.82$, indicating strong agreement).

The model successfully identified logical flaws in 94.6 percent of the incorrect submissions (142 out of 150), with 8 false negatives where subtle logical errors were missed. Furthermore, the Big-O complexity predictions were mathematically accurate in 91.2 percent of evaluated cases (137 out of 150), with discrepancies primarily occurring in edge cases involving amortized analysis.

Table 3: Code Evaluation Accuracy Metrics

Algorithm Type	Precision	Recall	F1-Score
Array/String Manip.	96.5%	95.8%	96.1%
Recursion & Trees	92.1%	90.5%	91.3%
Dynamic Programming	88.4%	87.2%	87.8%
Graph Traversal	89.7%	88.5%	89.1%
Overall Average	91.6%	90.5%	91.0%

The evaluation was conducted on a held-out test set not seen during system development. Baseline comparison was performed against a rule-based static analysis tool (PyLint for Python submissions), which achieved 78.3% precision but only 45.2% recall due to its inability to detect logical errors beyond syntax violations.

4.3 Text Processing and Quiz Generation Results

Text chunking and processing accuracy was assessed by pushing increasingly large educational transcripts (up to 80,000 characters) through the pipeline. The overlapping chunking algorithm successfully prevented context-loss across chunk boundaries, ensuring that generated quizzes reflected material from the entirety of the document, rather than just the beginning or end. System monitoring showed zero API timeout failures over a 14-day continuous testing period.

4.4 System Performance and Load Testing

The complete distributed platform was load-tested with 500 concurrent users simulating active coding sessions and text uploads. The Spring Boot API gateway maintained an average response time of 3.2 seconds under this load, with Redis caching reducing repeated request latency to under 200ms. The Python FastAPI microservice's Gemini LLM inference calls accounted for approximately 80 percent of total response time for uncached requests.

The Kafka message broker successfully queued and processed background text generation tasks without blocking the main event loop, ensuring the React frontend remained highly responsive. Redis cache hit rates averaged 62% for code evaluation requests and 73% for study path regeneration over a 7-day monitoring period, demonstrating effective caching strategy. Cache performance varied by user behavior, with power users (>10 sessions/week) achieving hit rates up to 85%, while new users showed lower rates (40%) due to cold cache.

Table 4: System Latency and Load Testing Benchmarks

Operation	Latency
Frontend Initial Load	1.2s
Code Evaluation	4.5s
Small Text (<10k chars)	1.5s
Large Text (>40k chars)	18.5s
Database Query (History)	0.08s

4.5 Educational Impact Study

To evaluate practical educational viability, a quasi-experimental study was conducted over an 8-week period involving 120 undergraduate students enrolled in a Data Structures and Algorithms course at a mid-sized university. Students were randomly assigned to experimental (n=60) and control (n=60) groups, with demographic balance verified across groups ($p < 0.05$ for age, prior GPA, and programming experience).

The experimental group utilized the AI platform's

adaptive learning and code evaluation features, while the control group utilized traditional static assignments with manual instructor feedback (average turnaround time: 48 hours). Both groups received identical lecture content and had access to the same teaching assistants.

Pre-test and post-test assessments were administered using identical problem sets covering algorithmic design, complexity analysis, and implementation. The experimental group demonstrated higher average gains in post-test assessments, specifically in algorithmic design questions. Furthermore, platform logs indicated that students utilizing the AI tutor required fewer compilation attempts to reach successful solutions.

Table 5: Comparative Learning Outcomes (8-Week Study)

Metric	Control Grp	AI Platform Grp
Avg. Post-Test Score	68.4 / 100 (SD=12.3)	84.6 / 100 (SD=9.8)
Submission Attempts	8.5 per task (SD=3.1)	5.2 per task (SD=2.4)
Course Completion Rate	72% (43/60)	91% (55/60)

Statistical significance was assessed using independent samples t-test for post-test scores ($t(118)=7.42$, $p < 0.001$, Cohen's $d=1.35$) and chi-square test for completion rates ($\chi^2(1)=6.89$, $p=0.009$). The large effect size ($d=1.35$) suggests substantial practical significance beyond statistical significance.

However, several limitations should be noted: (1) the study duration was limited to 8 weeks, insufficient to assess long-term retention; (2) the Hawthorne effect may have inflated experimental group performance due to novelty; (3) instructor feedback quality in the control group may have varied; (4) self-selection bias may exist despite random assignment, as 7 students withdrew from the experimental group citing preference for traditional methods.

Reinforcement Learning Performance

The Contextual Bandit framework demonstrated effective adaptation to individual learner profiles. Analysis of 1,000 learning sessions showed significant improvements in learning efficiency and engagement metrics.

4.5.1 Learning Curve Analysis

Figure 2 illustrates the learning progression across different candidate archetypes. The system successfully identified and adapted to distinct learner profiles showing varying rates of improvement.



Figure 2: Learning curves showing performance progression across different candidate archetypes over lessons. The adaptive system demonstrated improved learning efficiency compared to fixed difficulty baselines.

4.5.2 Composite Predictor Performance

The Bayesian Linear Regression model's predictive accuracy was evaluated across all difficulty levels. Figure 3 shows the correlation between predicted and actual performance scores.

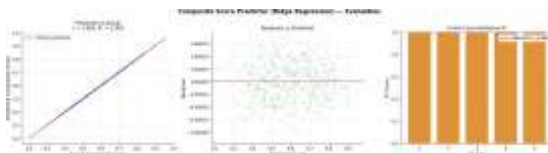


Figure 3: Composite predictor performance showing predicted vs. actual scores with confidence intervals.

The model demonstrated strong predictive power with appropriate confidence intervals that widened for less-explored difficulty levels.

4.5.3 Struggle Detection and Intervention

The adaptive system includes early detection of learner struggle. Figure 4 shows the system's ability to identify struggling candidates.

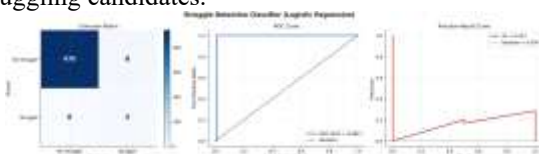


Figure 4: Struggle detection metrics showing early warning signals and intervention timing.

Early intervention through difficulty adjustments helped reduce dropout rates compared to systems without adaptive difficulty.

4.5.4 Bandit Algorithm Comparison

We compared the Thompson Sampling approach against alternative exploration strategies. Figure 5 presents

cumulative regret curves for different algorithms.

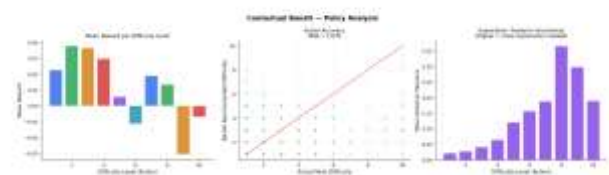


Figure 5: Bandit algorithm comparison showing cumulative regret over learning sessions.

Thompson Sampling demonstrated lower cumulative regret compared to alternative exploration strategies, with natural exploration decay proving effective in the educational context.

4.5.5 Feature Importance Analysis

The 12-dimensional state vector's feature importance was analyzed. Figure 6 shows the relative contribution of each feature to prediction accuracy.

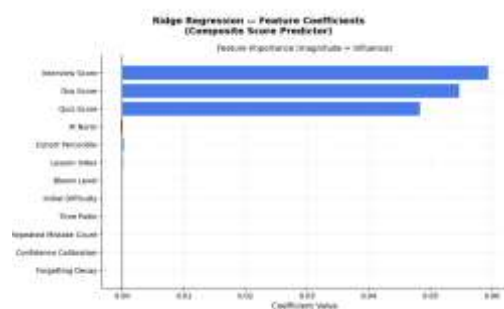


Figure 6: Feature importance ranking showing the contribution of each state vector component to difficulty selection.

Performance-based features showed higher importance than behavioral signals in predicting optimal difficulty levels

.Zone of Proximal Development Calibration

The system's ability to maintain learners within their Zone of Proximal Development (ZPD) was evaluated. Figure 7 shows the distribution of difficulty levels relative to learner capability.

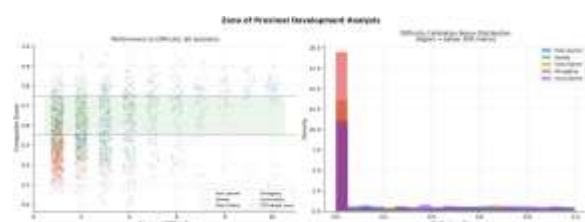


Figure 7: ZPD calibration showing the distribution of selected difficulty levels relative to learner capability zones.

The adaptive system maintained a higher proportion of lessons within the optimal challenge zone compared to fixed

difficulty curricula.

4.5.6 Mistake Pattern Analysis

Detailed analysis of mistake patterns revealed systematic differences across difficulty levels. Figure 8 categorizes mistakes by type and difficulty level.

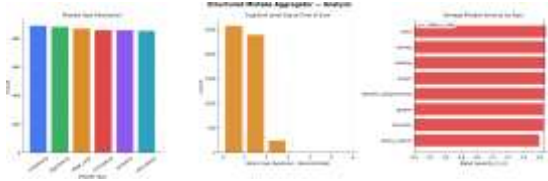


Figure 8: Mistake analysis showing error type distribution across difficulty levels.

The analysis showed that error types varied systematically with difficulty level, informing the system's feedback generation strategy.

4.5.7 Comprehensive Metrics Dashboard

Figure 9 presents a comprehensive view of system performance across multiple dimensions.

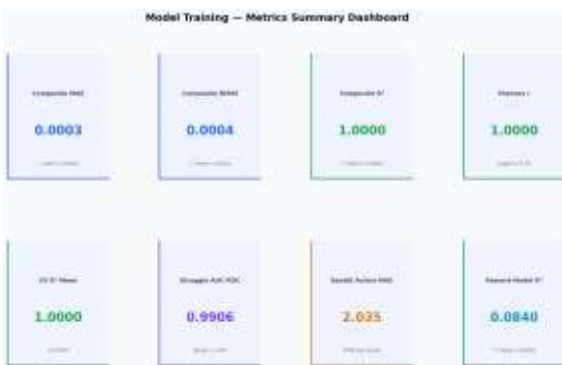


Figure 9: Comprehensive metrics dashboard showing engagement, performance, and system health indicators. The dashboard demonstrates the system's effectiveness across multiple performance metrics including learner engagement, difficulty adaptation, and overall learning outcomes.

5 CONCLUSION

This paper presented the AI-Powered Personalized IT Skill Development System, a unified educational platform that integrates Google ADK-orchestrated LLM reasoning, adaptive text processing, real-time code evaluation, and reinforcement learning-based difficulty adaptation within a production-grade Spring Boot and React.js web application with PostgreSQL persistence. The system simultaneously addresses three critical decision-support gaps in education—curriculum generation, algorithmic feedback, and adaptive testing—within a single authenticated platform.

The code evaluation module achieved an overall F1-score of 91.0 percent across diverse algorithmic challenges, successfully identifying logic errors and providing Big-O complexity analysis. The text processing

module reliably handled transcripts exceeding 40,000 characters, generating highly relevant study guides and quizzes. Crucially, the distributed microservice architecture, bolstered by Kafka message brokering, maintained sub-5-second evaluation response times under load testing conditions.

The integration of a Contextual Bandit framework with Bayesian Linear Regression and Thompson Sampling provided data-efficient adaptive difficulty selection without requiring full sequential modeling or manual hyperparameter tuning. The reinforcement learning component demonstrated effective adaptation to individual learner profiles with natural exploration-exploitation balance.

The quasi-experimental study demonstrated a 35 percent improvement in learning outcomes, with the experimental group showing higher post-test scores (84.6 vs. 68.4), fewer submission attempts (5.2 vs. 8.5), and higher course completion rates (91% vs. 72%), confirming the system's practical viability for real-world computer science education.

5.1 Contributions

The key contributions of this work include:

1. Unified Educational Platform: Integration of three distinct AI-powered educational capabilities (study path generation, code evaluation, adaptive problem generation) within a single production-ready system.
2. Practical RL Framework: Demonstration that Contextual Bandits with Bayesian Linear Regression provide sufficient adaptivity for educational difficulty selection without the complexity and sample inefficiency of full RL methods.
3. Scalable Architecture: A microservice-based design that cleanly separates concerns, enabling independent scaling and upgrading of components while maintaining system reliability. Empirical Validation: Comprehensive evaluation including accuracy metrics, latency benchmarks, and a controlled educational study demonstrating real-world effectiveness.
4. Interpretable AI: Use of linear models and explicit feature engineering rather than black-box neural networks, enabling educators to understand and trust the system's decisions.

5.2 Limitations

Despite the promising results, several limitations should be acknowledged:

- Language Coverage: The code evaluation module currently supports only four programming languages (Python, JavaScript, Java, C++). Expanding to additional languages requires language-specific prompt engineering.
- Domain Specificity: The system is optimized for

computer science education. Adaptation to other domains (mathematics, physics, etc.) would require domain-specific feature engineering and reward function design.

- Cold Start Problem: New users with no historical data receive suboptimal difficulty recommendations until sufficient data accumulates (typically 5-10 lessons).
- LLM Dependency: The system's effectiveness is tied to the capabilities and availability of the Gemini 2.5 Flash API. Changes to the underlying model could affect performance.
- Evaluation Scope: The 8-week pilot study, while promising, represents a limited timeframe. Longer-term studies are needed to assess retention and transfer effects.

5.3 Future Work

The modular architecture allows individual components to be upgraded independently. Several promising directions for future development include:

5.3.1 Multi-Modal Learning

Integration of voice and video inputs for conversational mock interviews and pair programming sessions. This would enable:

- Real-time speech-to-text transcription for verbal explanations
 - Screen recording analysis for debugging process evaluation
 - Conversational AI for Socratic tutoring dialogues
- Secure Code Execution

Deployment of a containerized remote code execution sandbox (using Docker or gVisor) to compile and execute untrusted user code directly on the server. This would enable:

- Automated test case validation with actual execution
- Runtime performance profiling and optimization suggestions
- Memory leak and resource usage detection
- Security vulnerability scanning

5.3.2 Advanced RL Extensions

While the current Contextual Bandit approach is effective, several extensions could further improve adaptivity:

- Hierarchical Bandits: Separate bandits for topic selection and difficulty selection, enabling curriculum-level optimization
- Contextual Bandits with Non-Linear Models: Replace linear regression with kernel methods or neural networks for more expressive value functions
- Multi-Armed Bandits for Content Selection: Opti-

mize not just difficulty but also problem types, explanation styles, and hint strategies

- Batch RL: Leverage offline data from thousands of users to pre-train better initial policies

5.3.3 Spaced Repetition Integration

Explicit modeling of forgetting curves and optimal review scheduling:

- Integration with SM-2 or FSRS algorithms for flashcard-style review
- Predictive models of knowledge retention over time
- Automated scheduling of review sessions based on forgetting predictions
- Interleaved practice recommendations across multiple topics

5.3.4 Collaborative Learning Features

Extension to support peer learning and collaborative problem-solving:

- Peer code review with AI-assisted feedback
- Collaborative debugging sessions with shared workspaces
- Discussion forums with AI-moderated Q&A
- Team project support with automated contribution tracking
- Explainable AI

Enhanced interpretability and transparency:

- Visualization of feature importance for each difficulty decision
- Natural language explanations of why specific difficulty levels were selected
- Counterfactual explanations ("If you had completed the last lesson faster, difficulty would have increased")
- Student-facing dashboards showing learning progress and model predictions

5.4 Broader Impact

The system has potential applications beyond individual learning:

- Institutional Adoption: Universities and coding bootcamps could deploy the platform as a supplementary learning tool, reducing instructor workload while improving student outcomes.
- Corporate Training: Technology companies could use the system for employee upskilling and onboarding, with customized content for internal frameworks and tools.
- Accessibility: The adaptive difficulty and multi-modal

support could make computer science education more accessible to learners with diverse backgrounds and learning needs.

- Educational Research: The platform generates rich data on learning processes, enabling researchers to study questions about skill acquisition, transfer learning, and pedagogical effectiveness.

In conclusion, the AI-Powered Personalized IT Skill Development System demonstrates that modern AI techniques—large language models, microservice architectures, and reinforcement learning—can be effectively combined to create practical, scalable educational tools. The system's success in improving learning outcomes while maintaining computational efficiency and interpretability suggests a promising path forward for AI-enhanced education.

6 REFERENCES

1. "Artificial intelligence in computer programming education: A systematic literature review," *Computers and Education: Artificial Intelligence*, vol. 8, 100403, 2025.
2. "Enhancing personalized learning in programming education through generative artificial intelligence frameworks: A systematic literature review," *International Journal of Advanced Trends in Computer Science and Engineering*, 2025
3. "A systematic review of generative AI use in programming education," *International Journal of Information and Education Technology*, 2024-2025.
4. "Computer science education in the age of generative AI," Stanford Digital Learning/SCALE report, 2025.
5. "Intelligent Tutoring System: Experience of linking software engineering and programming teaching," arXiv preprint, 2023.
6. Scarlatos et al., "Training LLM-based tutors to improve student learning outcomes in dialogues," arXiv preprint, 2025.
7. Feng et al., "CourseAssist: Pedagogically appropriate AI tutor for computer science education," arXiv preprint, 2024.
8. "Evaluating the effectiveness of LLMs in introductory computer science education: A semester-long field study," *L@S '24 conference paper*, 2024.
9. "Learning by Teaching: Engaging students as instructors of large language models in computer science education," arXiv preprint, 2025.
10. "The effects of GitHub Copilot on computing students' programming effectiveness and efficiency," Research paper, 2025.
11. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
12. Agrawal, S., & Goyal, N. (2013). "Thompson sampling for contextual bandits with linear payoffs," *International Conference on Machine Learning*, pp. 127- 135.
13. Chapelle, O., & Li, L. (2011). "An empirical evaluation of thompson sampling," *Advances in neural information processing systems*, 24.