International Journal of Advanced Scientific Research in Engineering Trends and Technology

Volume 1 Issue 5, Nov 2026
www.ijengt.com

AlgoAdaptive: An Intelligent Programming Practice Platform for Learner Weakness Identification Using Machine Learning

Swathi H Y
Adarsh Ramanagouda Malawade, Amrit Kumar Kamble, Mithun P Y, Pavan M
Department of CSE(AI&ML), Malnad College of Engineering, Hassan, Karnataka, India
Email: hys@mcehassan.ac.in , all other coauthors are students

***Abstract*—**Beginning programmers often receive immediate correctness feedback from coding platforms but limited evidence about the concepts that repeatedly cause difficulty. This paper presents AlgoAdaptive, a web-based programming practice platform that combines a structured fundamentals curriculum, automated code evaluation, learner analytics, explainable mastery estimation, sequential struggle prediction, and adaptive problem recommendations. The platform is designed to use a learner's submission history as evidence rather than infer weakness from a single incorrect attempt. Its active analytics pipeline extracts 25 point-in-time behavioral features from interactions preceding a prediction cutoff. A Ridge regression model estimates the learner's prospective success rate over the next three to five attempts; the continuous estimate is clipped to the unit interval and mapped to pedagogical mastery bands. A separate PyTorch Deep Knowledge Tracing model based on a gated recurrent unit (GRU) estimates next-attempt success probability from an ordered interaction sequence. Model explanations combine linear feature contributions with deterministic code-diagnostic evidence, while recommendation logic prioritizes unsolved problems from concepts requiring practice. The training corpus contains 178,915 simulated submissions from 1,500 synthetic learners generated using a two-parameter Item Response Theory simulator. Student-level partitioning and temporal feature/label separation were verified to reduce identity overlap and target leakage. On 4,075 held-out test instances from 232 simulated learners, the Ridge model achieved MAE 0.1978, RZ 0.3598, derived macro-F1 0.5484, expected calibration error 0.0176, and Spearman correlation 0.8503 with simulated latent ability. The DKT model achieved ROC-AUC 0.7202, accuracy 0.6727, log loss 0.6037, Brier score 0.2082, and ECE 0.0095. Independent acceptance testing also verified 393 backend tests, 65 frontend tests, and ten end-to-end user journeys. These findings establish functional and simulation-based evidence, not validated effectiveness on real learners. The paper details the architecture, methodology, evaluation, operational limitations, and requirements for a future real-student pilot.

***Index Terms*—**adaptive learning, programming education, learner analytics, weakness detection, knowledge tracing, Ridge regression, gated recurrent unit, explainable artificial intelligence, automated code evaluation.

I. INTRODUCTION

Programming competence develops through repeated attempts, debugging, conceptual transfer, and reflection. A conventional online judge can determine whether a submitted program satisfies test cases, but a verdict such as Accepted or Wrong Answer does not by itself explain the learner's underlying conceptual difficulty. A beginner may repeatedly misuse loop boundaries, misunderstand conditional expressions, or struggle with function parameters while receiving feedback that is limited to output correctness. A learning platform can become more useful when it connects submission outcomes with a concept model and recommends a meaningful next exercise.

Knowledge tracing provides a foundation for estimating a learner's evolving knowledge from a sequence of interactions. Early Bayesian approaches model changes in mastery as practice occurs [1], while recurrent neural approaches learn temporal patterns from interaction sequences [2]. In parallel, explainability methods can help expose which observed features contribute to a model output [3]. These ideas motivate an integrated system that links practice, prediction, explanation, and recommendation.

The main contributions of this work are fourfold: (1) a point-in-time, concept-aware weakness estimation pipeline that uses historical programming behavior rather than a single submission; (2) an evidence-gated decision layer that distinguishes insufficient evidence from demonstrated weakness; (3) a complementary dual-model design combining tabular mastery estimation with sequence-based next-attempt prediction; and (4) an end-to-end implementation that connects code execution, analytics, explanations, and adaptive recommendations, together with independent functional and operational verification. These contributions are evaluated as a system and as predictive components on a controlled synthetic cohort.

AlgoAdaptive is a browser-based platform focused on programming fundamentals rather than an advanced data-structures-and-algorithms curriculum. It provides a bank of 500 language-agnostic problems distributed across seven foundational concepts. The application records attempts and verdicts, executes submitted programs, updates learner analytics, and provides a tutor and adaptive practice recommendations. Its weakness-identification pipeline is designed to avoid premature labeling: a concept is not assessed until a minimum amount of evidence has accumulated.

The principal contribution of this work is an implementation-level account of a complete programming practice workflow and its audited analytics pipeline. The paper reports model metrics with explicit provenance, separates active models from archived legacy artifacts, and documents operational risks. It does not claim that synthetic evaluation proves educational impact or real-world predictive validity.

©IJENGT: All Rights are Reserved

II. PROBLEM DEFINITION AND OBJECTIVES

A. Problem Definition

A learner's programming difficulty is not fully represented by a single failed submission. Incorrect output can result from a syntax mistake, an incomplete understanding of a concept, a test-case edge condition, or an execution issue. A useful support system must aggregate evidence across problems, distinguish insufficient evidence from demonstrated weakness, and offer next steps that are relevant to the learner's current curriculum position.

B. Objectives

- Provide a structured and progressively organized programming fundamentals practice environment.
- Execute submitted code and return meaningful verdicts while preserving submission history.
- Estimate concept-level mastery from behavioral evidence observed before a defined prediction time.
- Model sequential interaction patterns to estimate the probability of success on a subsequent attempt.
- Explain predictions through feature contributions and deterministic code diagnostics.
- Recommend an appropriate unsolved problem from the learner's concept-specific progression.
- Enforce user-level data isolation and avoid exposing one learner's submissions to another.

C. Scope

The system is a learning-support prototype for individual programming practice. It is not an accredited assessment instrument, a substitute for an instructor, or evidence that a learner has a fixed ability. The current machine-learning benchmark is based on synthetic learner simulations. Consequently, predictions should be interpreted as experimental estimates that require validation with a larger, consented real-student cohort.

III. RELATED WORK AND TECHNICAL FOUNDATIONS

A. Knowledge Tracing

Knowledge tracing estimates latent knowledge from a learner's observed responses over time. Bayesian Knowledge Tracing represents knowledge as a latent state and updates the probability of mastery after practice [1]. Deep Knowledge Tracing introduced recurrent neural networks to model sequences of educational interactions without requiring a manually specified state transition for every skill [2]. AlgoAdaptive adopts complementary approaches: a tabular, point-in-time mastery regressor and a sequence model for next-attempt outcome estimation.

B. Item Response Theory

Item Response Theory (IRT) models the probability of a response as a function of learner ability and item characteristics. In a two-parameter logistic (2PL) model, item difficulty and discrimination influence the probability of a correct response [4]. The project uses a 2PL-based simulator to generate synthetic learner profiles and submission outcomes. This provides controlled data for development and evaluation, but simulated behavior cannot reproduce every feature of human learning, motivation, fatigue, collaboration, or strategy.

C. Regularized Regression

Ridge regression adds an L2 penalty to the ordinary least-squares objective. The penalty constrains coefficient magnitude and can improve stability when input variables are correlated [5]. AlgoAdaptive uses a Ridge regressor for continuous prospective success-rate estimation. The model is relatively compact and its linear structure supports direct decomposition of feature contributions.

D. Sequential Neural Models

A gated recurrent unit maintains a hidden representation across a sequence and uses gates to regulate information flow. In the project's DKT implementation, a PyTorch GRU receives encoded interaction information and produces a next-attempt success estimate. The model is a genuine sequential neural network; it should not be conflated with the archived legacy scikit-learn multilayer perceptron.

E. Prediction Explanations

SHAP provides a unified framework for attributing a model prediction to input features [3]. The active mastery workflow uses linear feature contributions to provide reasons associated with a prediction. Such attributions describe the model's dependence on observed inputs; they do not prove that a feature caused a learner's difficulty. The platform also incorporates deterministic abstract syntax tree (AST) diagnostics, such as detected code-pattern warnings, as additional evidence.

F. Research Gap Addressed

The implementation addresses a practical integration gap between code execution and learner support. Instead of treating the judge as an isolated verdict service, it connects attempts to concept analytics and recommendations. The research contribution is therefore system-oriented: it documents the integration and evaluates the active prediction components on a controlled synthetic cohort.

IV. SYSTEM OVERVIEW AND ARCHITECTURE

AlgoAdaptive follows a client–server design. The frontend is implemented using React 18, TypeScript, Vite, and Tailwind CSS [6]. The backend uses Python FastAPI and exposes versioned API routes [7]. The audited development configuration used a Vite frontend on port 5175 and a FastAPI backend on port 8000. The persistence layer is SQLite (aiml_platform.db) [8]. The code-execution workflow is handled by the backend judge and queue components.

TABLE I. SYSTEM ARCHITECTURE LAYERS AND IMPLEMENTATION RESPONSIBILITIES.

Layer	Implementation / responsibility
Presentation	React 18, TypeScript, Vite, Tailwind CSS; practice library, editor, dashboard, insights, assistant and submission views.
API and application services	FastAPI; authentication, problem catalog, submission, judge, analytics, recommendations and assistant endpoints.
Execution	LocalExecutor and judge queue; temporary work directories, time and output limits, process termination and environment sanitization.
Persistence	SQLite database for users, problems, submissions, progress, analytics and recommendations.

©IJENGT: All Rights are Reserved

Layer	Implementation / responsibility
Analytics models	Ridge mastery champion (scikit-learn) and DKT GRU champion (PyTorch), stored as versioned model artifacts.

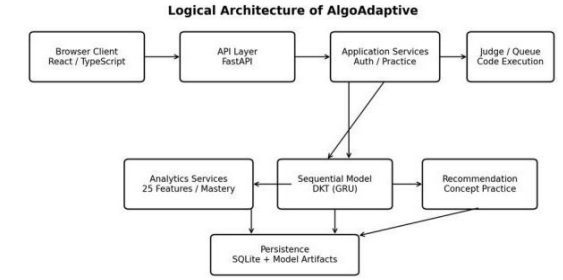


Fig. 1. Logical architecture of AlgoAdaptive derived from the audited implementation: browser client, API layer, application services, judge/queue, analytics, sequential prediction, recommendation, and persistence/model artifacts.

A. Practice and Submission Flow

A learner selects a problem, reads the statement and submits code through the editor. The backend evaluates the submission and records the verdict. The active submission queue invokes mastery snapshot computation and resolves prior DKT predictions against actual outcomes. Updated analytics can then be requested through the weakness endpoint. Recommendation services use the resulting concept status and the learner's completion history to select an unsolved problem.

B. Curriculum

TABLE II. DISTRIBUTION OF QUESTIONS ACROSS FOUNDATIONAL CONCEPTS.

Concept area	Question count
Syntax / Basics	248
Operators	77
Conditionals	50
Loops	40
Functions	40
Strings	25
Arrays	20
Total	500

The audited question bank contains 500 active questions: 333 Easy, 157 Medium, and 10 Hard. The independent curriculum scan found zero banned advanced DSA-topic violations. This scope is intentional: the platform aims to support a beginner's transition into programming logic.

The audited curriculum is distributed across seven foundational concepts: Syntax/Basics (248), Operators (77), Conditionals (50), Loops (40), Functions (40), Strings (25), and Arrays (20), for a total of 500 questions. This distribution emphasizes introductory programming logic rather than advanced data-structures-and-algorithms content.

C. Authentication and Data Isolation

The acceptance audit verified user-scoped ownership checks on submission, judge, assistant, user, and recommendation operations. Requests attempting to access another user's submission are rejected through ownership filtering or authorization errors. Seven authorization-isolation regression tests passed within the broader backend suite. These controls are important because learner histories and submitted source code are private user data.

V. DATASET CONSTRUCTION AND FEATURE ENGINEERING

A. Synthetic Corpus

The active Phase 7 analytics corpus contains 178,915 simulated submissions from 1,500 synthetic students interacting with 150 problems. Student cohorts were partitioned into 1,033 training students, 235 validation students, and 232 test students. The evaluation script reported 4,075 test instances across the 232 held-out synthetic learners. The dataset card identifies the cohort as generated through a 2PL IRT simulator.

TABLE III. STUDENT-LEVEL PARTITIONING REPORTED BY THE DATASET AUDIT.

Partition	Learners	Share (approx.)
Training	1,033	68.9%
Validation	235	15.7%
Test	232	15.5%
Total	1,500	100%

B. Point-in-Time Feature Extraction

For each observation, the feature cutoff is the timestamp of the current interaction. Features are constructed using only interactions strictly earlier than that cutoff. The audited extractor produces 25 features representing behavioral evidence such as error ratios, solve times, attempt velocity, hint usage, and prerequisite mastery. The feature vector is validated and sanitized to the expected shape of (1, 25) before inference.

The target is prospective rather than contemporaneous: it is calculated from the learner's next three to five submissions. The target is the future accepted-attempt rate, defined as accepted attempts divided by attempts, and constrained to [0, 1]. This arrangement is intended to prevent the model from using future outcomes as input when predicting them.

C. Leakage Prevention

Two safeguards were verified. First, the student split is grouped by student identifier, with assertions that training, validation, and test ID sets have empty intersections. Second, temporal feature extraction uses interactions before the cutoff, while the label uses subsequent interactions. The design reduces two common evaluation errors: the same learner appearing in both training and test sets, and future outcome information entering the predictor's input.

©IJENGT: All Rights are Reserved

D. Data Provenance and Limitations

All training examples for the active champions are synthetic. The acceptance audit reported only a small number of genuine pilot interactions in the live database, far below the project's stated minimum threshold of 500 real submissions for candidate promotion. Therefore, the reported scores characterize performance on the simulator's data distribution. They must not be presented as validated accuracy for real students.

VI. MACHINE-LEARNING METHODOLOGY

A. Ridge Mastery Predictor

The active mastery artifact is mastery_champion_mastery_v1.0.joblib, implemented as scikit-learn Ridge regression. The evaluation audit identifies the champion regularization setting as $\alpha = 2.0$. For feature vector x, the model produces a continuous estimate $\hat{y}$ of future solve rate. The serving path clips the output to [0, 1] before downstream interpretation. In general form, Ridge estimates coefficients by minimizing: $J(\beta) = \Sigma_i (y_i - x_i^T\beta)Z + \alpha\|\beta\|Z$

The predicted value is converted into pedagogical bands using fixed thresholds: Weak for $\hat{y} < 0.40$, Developing for $0.40 \leq \hat{y} \leq 0.70$, and Strong for $\hat{y} > 0.70$. The same boundaries are applied to prospective ground-truth success rates when deriving classification metrics. These are application decision bands, not universal educational standards.

B. Deep Knowledge Tracing Struggle Predictor

The active sequential artifact is dkt_champion_dkt_v1.0.pt. It contains a PyTorch DKTModel using a GRU with a hidden dimension of 64 and multi-head embeddings. Its sequence inputs encode interaction properties including concept, difficulty, verdict, tag, attempts, hints, time, and target/query information. At inference, the service constructs the learner's ordered sequence and appends a query step for the problem being evaluated. The model estimates the probability of acceptance on the query problem; struggle probability is represented as one minus that probability.

The DKT model complements the Ridge predictor. The Ridge model estimates a concept-level prospective success rate from an engineered snapshot, whereas DKT models sequential response patterns for a next-step prediction. Their scores should not be described as interchangeable outputs.

C. Evidence Gating

The weakness endpoint does not label a learner when evidence is insufficient. The verified gate requires at least three attempts and at least two distinct problems for the concept. Below this threshold, the API returns an insufficient-data state and reports no active concepts evaluated. This behavior prevents a new learner from being classified as weak solely because of one or two early mistakes.

D. Explanation and Recommendation

The mastery service combines linear feature-contribution information with deterministic AST-based diagnostic warnings. Examples in the audit include nested-loop complexity patterns and repeated failure on edge-case tests. These signals are presented as explanatory evidence, not as causal diagnoses. The recommender ranks concepts by weakness status and selects an unsolved problem from the relevant pedagogical ladder. The result is an actionable practice suggestion rather than only a numeric score.



Fig. 2. Evidence-gated weakness-detection workflow.

VII. EXPERIMENTAL DESIGN AND EVALUATION

A. Evaluation Protocol

The independent machine-learning evaluation was reproduced using the saved active artifacts and evaluation script. The test set comprised 4,075 instances from 232 synthetic learners not used in training or validation. The evaluation included a majority baseline, a rule-based baseline, multinomial logistic regression, and the Ridge champion. The DKT model was evaluated separately using binary next-attempt prediction metrics. No claim is made that the test set represents a random sample of real programming students.

B. Metrics

Mean absolute error (MAE) is the average absolute difference between predicted and observed future success rates. RZ measures the proportion of variance explained relative to a constant-mean reference. Macro-F1 is the unweighted average of per-class F1 scores after continuous outputs are mapped to mastery bands. Expected calibration error (ECE) summarizes the difference between predicted confidence and observed frequency across confidence bins. Spearman's rank correlation measures the association between predicted ordering and the simulator's latent ability ordering. For DKT, ROC-AUC measures ranking quality across thresholds, log loss evaluates probabilistic predictions, and the Brier score measures squared probability error.

C. Mastery Model Results

TABLE IV. REPRODUCED MASTERY EVALUATION ON THE SYNTHETIC HELD-OUT COHORT. DASHES INDICATE METRICS NOT REPORTED IN THE SUPPLIED AUDIT.

Model / baseline	Macro-F1	MAE	R^2	ECE
Majority baseline	0.1956	—	—	—
Rule-based baseline	0.4772	0.2487	−0.1200	0.1145
Multinomial logistic regression	0.5551	—	—	—
Ridge champion ($\alpha = 2.0$)	0.5484	0.1978	+0.3598	0.0176

©IJENGT: All Rights are Reserved

The Ridge champion improves substantially over the rule-based baseline on MAE, RZ, and calibration. Its derived macro-F1 is 0.5484, slightly below the reported multinomial logistic regression macro-F1 of 0.5551. Accordingly, it should not be described as the best model on every metric. The Ridge model has lower MAE and ECE than the rule baseline and a positive RZ, while the logistic regression has the higher reported macro-F1.

The Ridge champion's Spearman correlation with the simulator's latent ability parameter was 0.8503. This indicates strong rank association within the simulated evaluation environment. It does not demonstrate that the platform can recover a real student's underlying ability with the same correlation.

D. DKT Results

TABLE V. SEQUENTIAL STRUGGLE PREDICTION RESULTS FROM THE MODEL AUDIT.

Metric	DKT GRU	Baseline
ROC-AUC	0.7202	0.5000 (global mean)
Accuracy	0.6727	0.5820
Log loss	0.6037	0.6796
Brier score	0.2082	0.2433
Expected calibration error	0.0095	—

The DKT model outperformed the stated baseline on ROC-AUC, accuracy, log loss, and Brier score. Its ECE was 0.0095. These results support the usefulness of the sequence model on the simulator-generated test sequences. External validation is needed before interpreting the outputs as reliable estimates for real learners.

E. Runtime Artifact Verification

The audit loaded the Ridge artifact and confirmed its 25-feature input shape. A deterministic in-memory inference check returned a finite value within the valid mastery interval after the service's output handling. The DKT state dictionary was loaded using PyTorch's weights-only loading path; a sequence inference check returned a valid acceptance probability. These checks establish artifact integrity and executable inference, not generalization.

F. Product and Acceptance Tests

TABLE VI. ACCEPTANCE-TEST EVIDENCE RECORDED ON 31 OCTOBER 2026.

Test category	Verified result	Interpretation
Backend automated tests	393 passed; 3 warnings	Test suite completed successfully.
Frontend tests	65 passed across 12 files	Frontend unit/smoke tests passed.
End-to-end journeys	10 of 10 passed	Signup, login, judge, tutor, gating, recommendations, dashboard, logout and expiry.
Authorization isolation	7 of 7 targeted tests passed	Cross-user access controls verified.
Load test	120/120 requests successful	100 dashboard and 20 submission requests; latency varied by endpoint.

The load test measured 100% success across 120 requests. Dashboard requests at concurrency 20 achieved 17.90 requests per second, with p95 latency of 3,064.96 ms and p99 of 3,252.06 ms. Submission requests at concurrency 8 achieved 14.07 requests per second, with p95 latency of 518.00 ms and p99 of 561.99 ms. The audit attributed elevated dashboard tail latency to SQLite write-lock contention during recommendation regeneration. A successful request rate therefore should not be interpreted as uniformly low latency.

VIII. SECURITY, RELIABILITY, AND OPERATIONAL LIMITATIONS

A. Code Execution Isolation

The execution service uses temporary working directories, process-tree termination on timeout, output-size limits, memory monitoring, and environment sanitization. However, the audited Windows LocalExecutor launches native child processes under the host user account. It does not enforce Linux namespaces, cgroups, seccomp, or container-level isolation. Submitted code may retain access to files and network resources available to that host account, subject to operating-system and firewall permissions. The implementation is suitable for controlled local demonstration, but it is not an adequate security boundary for unrestricted public execution of untrusted code.

B. Database Concurrency

The system uses SQLite. Under concurrent dashboard activity, recommendation regeneration can execute a delete followed by batch inserts and a commit. SQLite's single-writer behavior can serialize these operations and increase tail latency. The audit observed p95/p99 dashboard latency of approximately three seconds under the tested concurrency. A production deployment with higher concurrency should evaluate PostgreSQL and transaction redesign.

C. Secret and Repository Hygiene

The active working-tree scan found no exposed secrets, but the Git-history scan found one historical Google API key committed in an earlier version of frontend/.env.example. It was removed from the current working tree, but remains in repository history. If the repository is exposed publicly, the key should be rotated in the provider console and repository history handled according to the institution's security policy.

D. Database Integrity Debt

The backup-and-restore audit confirmed a valid SQLite integrity check and exact row-count parity across 38 tables after restoration. It also found 165 orphaned foreign-key rows in legacy tables: 56 in user_concept_masteries, 56 in mastery_snapshots, and 53 in recommendations. These rows did not prevent the tested restore, but they represent data-cleanup work before strict foreign-key enforcement.

E. Legacy Components

The audit identified older AI endpoints whose services may fall back to rule-based heuristics because their configured model directory does not match the archived artifact location. The active submission and analytics paths use the Phase 7 champion services.

©IJENGT: All Rights are Reserved

Documentation and future maintenance should distinguish the active /analytics workflow from older /ai endpoints to avoid treating fallback output as champion-model inference.

F. Synthetic-Data Validity

Synthetic simulation is useful for controlled experiments, reproducible labels, and leakage testing. It also creates a domain gap: simulated response behavior is determined by the simulator's assumptions. Real learners may use hints differently, retry for different reasons, learn unevenly, or be affected by language familiarity and external constraints. The small real pilot sample is not sufficient for retraining or external validation. The current results therefore support a proof-of-concept, not a validated educational intervention.

IX. DISCUSSION

The results indicate that an end-to-end system can connect code practice, submission outcomes, concept-level analytics, sequential prediction, explanations, and adaptive recommendations. The evidence-gating mechanism is a significant product safeguard: it distinguishes insufficient evidence from a weak mastery estimate. This distinction is pedagogically important because early mistakes are common during initial exposure to a concept.

The model results should be interpreted as complementary rather than as a single leaderboard. Ridge regression has a useful continuous target and interpretable linear structure. Its derived macro-F1 is 0.5484, while the logistic regression baseline has a slightly higher macro-F1 of 0.5551. The Ridge model nevertheless has the reported lower MAE and better calibration than the rule-based baseline. The DKT model provides a sequence-based estimate and outperforms its reported global-mean baseline on the supplied metrics. A single combined accuracy is not reported because the two models address different prediction tasks.

The audit's data-leakage checks strengthen confidence that the reported synthetic evaluation is not trivially inflated by student overlap or future-feature leakage. However, leakage control does not remove simulator bias. An evaluation can be methodologically clean and still fail to generalize when the data-generating process differs from the real deployment population.

The application's functional verification is comparatively strong: 393 backend tests, 65 frontend tests, and ten end-to-end journeys passed in the acceptance audit. Yet quality is multidimensional. The code-execution isolation gap, SQLite tail latency, historical key exposure, and orphaned database rows are material limitations that must be addressed before public deployment.

X. LIMITATIONS AND FUTURE WORK

A. Current Limitations

- The active mastery and DKT training/evaluation data are synthetic; real-world predictive validity is not established.
- The live pilot contains too few genuine interactions to support reliable retraining or subgroup analysis.
- The local Windows execution process lacks container or kernel-level isolation for untrusted public workloads.
- SQLite write contention creates high dashboard p95 and p99 latency under the tested concurrency.
- A historical API key remains in Git history and requires rotation if exposed.
- Legacy AI endpoints and active champion endpoints must be clearly separated to avoid ambiguous model provenance.
- Orphaned foreign-key rows require cleanup before enabling strict referential integrity.
- Model outputs are estimates and should not be used for highstakes grading, student ranking, or punitive decisions.

B. Future Work

- Run a consent-based pilot across a diverse cohort and collect sufficient real submissions before retraining or promoting candidate models.
- Pre -register evaluation criteria and compare against simple baselines on a temporally separated real-student test cohort.
- Evaluate calibration, error rates, and performance across programming experience levels, languages, and accessibility contexts.
- Deploy code execution in disposable Linux containers or a hardened sandbox such as a namespace- and seccomp-based worker.
- Move high-concurrency persistence workloads to PostgreSQL and optimize recommendation transactions.
- Rotate historical credentials, review repository history, and establish automated secret scanning in continuous integration.
- Clean orphaned records and enable explicit foreign-key enforcement after a verified migration.
- Conduct a controlled educational study to measure whether adaptive recommendations improve learning outcomes, not merely prediction metrics.

Research validity statement—The reported predictive metrics are internally evaluated on a synthetic, student-disjoint test cohort with temporal feature/label separation. They should therefore be interpreted as evidence of model behavior under the simulator assumptions, not as evidence of validated learning gains or generalization to real students.

XI. CONCLUSION

This paper presented AlgoAdaptive, a programming practice and learner-support platform that integrates automated code evaluation with machine-learning-based mastery estimation, sequential struggle prediction, explanatory evidence, and adaptive recommendations. The active analytics pipeline uses 25 point-in-time features for Ridge-based prediction of future solve rate and a PyTorch GRU-based Deep Knowledge Tracing model for next-attempt prediction. The evaluation used 178,915 synthetic submissions from 1,500 simulated learners, with student-level train, validation, and test separation and temporal feature/label boundaries.

On the held-out synthetic cohort, the Ridge model achieved MAE 0.1978, RZ 0.3598, derived macro-F1 0.5484, ECE 0.0176, and Spearman correlation 0.8503 with simulated latent ability. The DKT model achieved ROC-AUC 0.7202, accuracy 0.6727, log loss 0.6037, Brier score 0.2082, and ECE 0.0095. Acceptance testing verified 393 backend tests, 65 frontend tests, and ten end-to-end journeys. These results demonstrate a functioning prototype and measurable performance on synthetic data. They do not establish effectiveness or accuracy for real learners. A real-student pilot, hardened execution isolation, database scalability work, and security remediation are necessary next steps.

©IJENGT: All Rights are Reserved

ACKNOWLEDGMENT

The author acknowledges the guidance of the project supervisor and faculty members of the Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), Malnad College of Engineering, Hassan. The author also acknowledges the researchers whose work on knowledge tracing, Item Response Theory, regularized regression, and explainable machine learning informed the technical foundations discussed in this paper.

REFERENCES

[1] 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, 1995.

[2] C. Piech, J. Bassen, J. Huang, S. Ganguli, M. Sahami, L. Guibas, and J. SohlDickstein, "Deep Knowledge Tracing," in Advances in Neural Information Processing Systems 28, 2015, pp. 505–513.

[3] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems 30, 2017, pp. 4765–4774.

[4] A. Birnbaum, "Some latent trait models and their use in inferring an examinee's ability," in Statistical Theories of Mental Test Scores, F. M. Lord and M. R. Novick, Eds. Reading, MA, USA: Addison-Wesley, 1968, pp. 395–479.

[5] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," Technometrics, vol. 12, no. 1, pp. 55–67, 1970.

[6] Meta Open Source, "React documentation," official project documentation.

[7] S. Ramírez, "FastAPI documentation," official project documentation.

[8] SQLite Consortium, "SQLite documentation," official project documentation.

[9] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in Proc. 34th Int. Conf. Machine Learning, 2017, pp. 1321–1330.

©IJENGT: All Rights are Reserved