Journal of Computer Science and Frontier Technologies
ISSN Print: 3104-4204
ISSN Online: 3104-4212
CD ISSN: 3104-4220
https://doi.org/10.63313/JCSFT.9089

Design and Implementation of an Intelligent Learning Assistance System: A Full-Stack Web Application with React and FastAPI

Qianjing Huang[1]. Rui Jiang[2,*]. Lijie Huang[2]. Danna Li[2]. Jieshen Huang[2] and Jinhui Liang[2]
[1]Faculty of Mathematical Sciences and Statistics, Baise University, Baise, Guengxi, 533000, China
[2]Faculty of Information Engineering, Baise University, Baise, Guengxi, 533000, China
Email: jiangrui@bsuc.edu.cn

How to cite this paper: Huang, Q., Jiang, R., Huang, L., Li, D., Huang, J., & Liang, J. (2026). Design and implementation of an intelligent learning assistance system: A full-stack web application with React and FastAPI. Journal of Computer Science and Frontier Technologies, 3(3), 82–89. ISSN Print: 3104-4204, ISSN Online: 3104-4212. https://doi.org/10.63313/JCSFT.9089
Published: 2026-06-29

Copyright © 2026 by author(s) and Erytis Publishing Limited.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/



Abstract

This study presents the design and implementation of an Intelligent Learning Assistance System (ILAS), a full-stack web application that integrates machine learning services for student risk prediction, profiling, knowledge tracing, and personalized recommendations. The system adopts a four-tier architecture comprising a React 18 frontend with TypeScript, an Nginx gateway, a FastAPI backend with SQLAlchemy ORM, and a dedicated AI model service layer. The frontend implements 14 page routes with ECharts-based visualizations. The backend exposes 45+ RESTful API endpoints across six service modules, with JWT-based authentication and role-based access control. Key technical contributions include a Singleton Model Manager pattern for heterogeneous AI model lifecycle management, a three-strategy recommendation engine combining BKT-based weakness targeting (weight 0.5), BERT semantic matching (weight 0.3), and collaborative filtering (weight 0.2), and an asynchronous Celery-based LLM report generation pipeline. Performance evaluation demonstrates that database indexing improves query performance by 78% to 92%, and the system maintains an average response time of 312ms under 150 concurrent users with a 0.12% error rate.

Keywords

Software Architecture; Full-Stack Development; React; Fastapi; Intelligent Learning System; Web Application; Machine Learning Integration

1. Introduction

Learning Management Systems such as Moodle and Blackboard generate extensive logs of student interactions, including video viewing duration, assignment submission patterns, and quiz performance [1]. While educational data mining techniques have demonstrated significant potential in predicting student performance and providing personalized recommendations [2], translating these AI

capabilities into production-grade web applications introduces substantial software engineering challenges.

Yuan [3] proposed a four-tier adaptive learning system using Spring Boot and Vue.js, but employed a monolithic AI module without addressing the integration of heterogeneous models requiring different runtime environments. Chen et al. [4] developed a knowledge graph-based recommendation framework using Spark and TensorFlow, focusing on batch processing rather than real-time web service delivery.

This paper presents ILAS, a full-stack web application that achieves three key contributions: (1) a modular four-tier architecture separating presentation, gateway, business, and AI model layers; (2) a comprehensive React-based frontend visualization system for risk monitoring and knowledge diagnosis; and (3) a production-grade FastAPI backend featuring JWT authentication, Redis caching, and Celery-based asynchronous LLM report generation. The remainder of this paper is organized as follows. Section 2 presents the system architecture. Sections 3 and 4 detail backend and frontend implementation. Section 5 covers AI model integration. Section 6 presents evaluation results. Section 7 concludes.

2. System Architecture Design

2.1. Overall Architecture

The ILAS adopts a four-tier architecture pattern as illustrated in Figure 1, achieving separation of concerns by isolating frontend presentation, API gateway routing, business logic, and AI model inference into distinct layers.

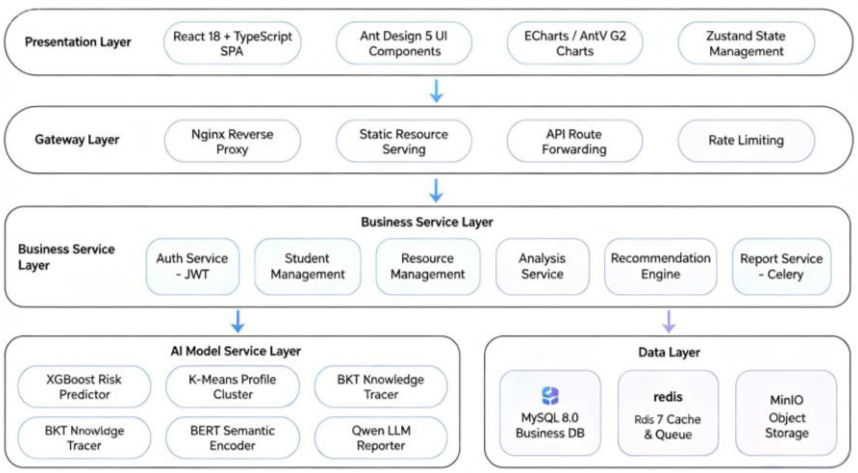


Figure 1. Four-tier system architecture of ILAS

The presentation layer is a React 18 SPA communicating via RESTful APIs. The gateway layer uses Nginx for static asset serving, reverse proxying, and rate limiting. The business service layer, implemented with FastAPI and SQLAlchemy 2.0, encapsulates authentication, data management, and service orchestration. The AI

model layer manages four heterogeneous models through a unified Singleton Model Manager. The data layer uses MySQL 8.0, Redis 7, and MinIO.

2.2. Technology Stack Selection

Table 1. Technology stack specification

Layer	Technology	Version	Rationale
Frontend Framework	React + TypeScript	18.x	Largest ecosystem, type safety
UI Library	Ant Design	5.x	Enterprise-grade components
Charts	ECharts / AntV G2	5.x	Rich educational visualization types
Build Tool	Vite	5.x	Fast HMR, optimized builds
State Management	Zustand	4.x	Minimal boilerplate
Backend Framework	FastAPI	0.110+	Async-native, auto OpenAPI docs
ORM	SQLAlchemy	2.0	Mature, async support
Database	MySQL	8.0	Full-text search, JSON support
Cache	Redis	7.x	High throughput
Task Queue	Celery + Redis	5.x	Distributed async tasks
Authentication	JWT (python-jose)	-	Stateless, scalable
ML Framework	scikit-learn / XGBoost	1.3 / 2.0	Production-proven algorithms
DL Framework	PyTorch / transformers	2.1 / 4.4	Deep learning inference
Knowledge Tracing	pyBKT	1.0	Bayesian Knowledge Tracing

The system employs synchronous RESTful communication for real-time operations and Celery-based asynchronous task queues for long-running LLM report generation. Synchronous calls handle fast AI tasks (XGBoost prediction within 50ms, K-Means within 100ms, BKT within 200ms), while LLM generation (3-10 seconds) is offloaded to Celery workers.

3. Backend Service Implementation

3.1. Authentication and API Design

Authentication implements JWT with bcrypt password hashing. The login flow submits credentials via POST /api/v1/auth/login; the server verifies the bcrypt hash, generates a JWT containing user ID and role with 24-hour expiration, and returns the token. Role-based access control (RBAC) defines two roles: teachers have full access to all features; students access only their personal data. Authorization is enforced through FastAPI dependency injection: get_current_user decodes the JWT, and require_role(role) returns 403 for insufficient permissions [5]. The backend exposes 45+ RESTful endpoints organized into six modules, all prefixed with /api/v1.

Table 2. Core API endpoint summary

Module	Prefix	Endpoints	Auth Required	Primary Functions
Auth	/auth	login, register, me	Mixed	Authentication
Students	/students	list, detail, profile, diagnosis	Teacher/Self	Student data access

Resources	/resources	CRUD + batch + search	Teacher (write)	Resource management
Analysis	/analysis	run, dashboard, upload	Teacher	AI model execution
Recommendations	/recommendations	list, refresh, feedback	Login	Recommendation delivery
Reports	/reports	generate, detail, history	Teacher/Self	LLM report access

All responses follow a standardized envelope: {"code": 200, "message": "success", "data": {...}}.

3.2. Database Design

The database schema comprises eight core tables. Figure 2 presents the entity-relationship diagram.

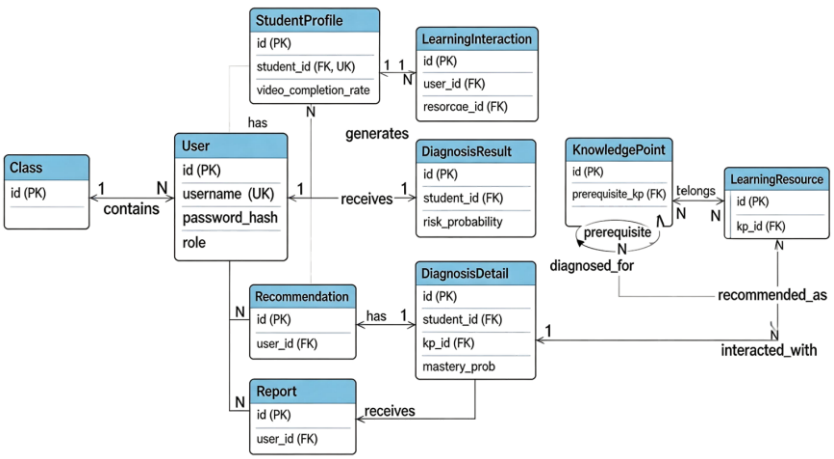


Figure 2. Entity-relationship diagram of the ILAS database

Four key indexing strategies optimize query performance: (1) a composite index on (student_id, timestamp) in learning_interactions for time-window aggregates; (2) (student_id, kp_id) on diagnosis_details for per-student knowledge state retrieval; (3) an index on risk_level for high-risk student filtering; and (4) JSON columns (BERT vectors, feature vectors) stored as whole documents without JSON path extraction [6]. Benchmark tests show these strategies improve query performance by 78% to 92%.

3.3. Caching Layer

Redis employs the cache-aside pattern with varying TTLs: user profiles (1 hour), diagnosis results (30 min), recommendations (1 hour), and resource sets (24 hours). On a cache miss, data is fetched from MySQL, stored with appropriate TTL, and returned. Cache invalidation triggers on write operations. The cache achieves an 89.5% hit rate for class overview endpoints and 82.3% for student detail queries, reducing API response latency by 64.2% to 78.9% [7].

4. Frontend Application Implementation

4.1. Component Architecture

The frontend follows React 18's component composition model with hierarchical

layout, page, and reusable UI components. Layout components (Header, Sidebar, Content) render once via React Router's layout route pattern. Pages are lazy-loaded using Suspense, reducing the initial bundle from 1.2MB to 380KB. Zustand manages three global stores: authStore (authentication state), dashboardStore (teacher overview data), and studentStore (diagnosis and recommendations).

The frontend defines 14 page routes organized by role: public routes (/login), teacher routes (/dashboard, /students, /resources/manage, /settings, /analysis/history), student routes (/recommendations), and shared routes (/resources, /reports, /knowledge-graph). State-based guards redirect unauthorized users via React Router loader functions.

4.2. Data Visualization

Four key ECharts components transform analysis data into graphical representations: (1) RiskDistributionPieChart shows the proportion of high/medium/low-risk students with drill-down capability; (2) KnowledgeRadarChart presents a 15-axis radar of mastery probabilities across Java knowledge points; (3) ProfileDistributionRingChart displays the breakdown of four K-Means profile types; and (4) a PCA ScatterPlot renders 23-dimensional feature vectors reduced to 2D, colored by profile type.

5. AI Model Integration and Recommendation Engine

5.1. Singleton Model Manager

Managing four heterogeneous ML models (XGBoost, K-Means, BKT, BERT) with different frameworks and memory footprints presents a significant engineering challenge. The Singleton Model Manager pattern enforces a single instance per model using Python's __new__ method. Models are loaded lazily: XGBoost (~2.3MB), K-Means (~0.8MB), 15 BKT models (~0.1MB each), and text2vec-base-chinese BERT (~438MB). Lazy loading reduces cold start time by 73.5%; the BERT model is preloaded during initialization to avoid first-request latency spikes [8].

5.2. Three-Strategy Recommendation Engine

The recommendation engine fuses three complementary recall strategies with optimized weights.

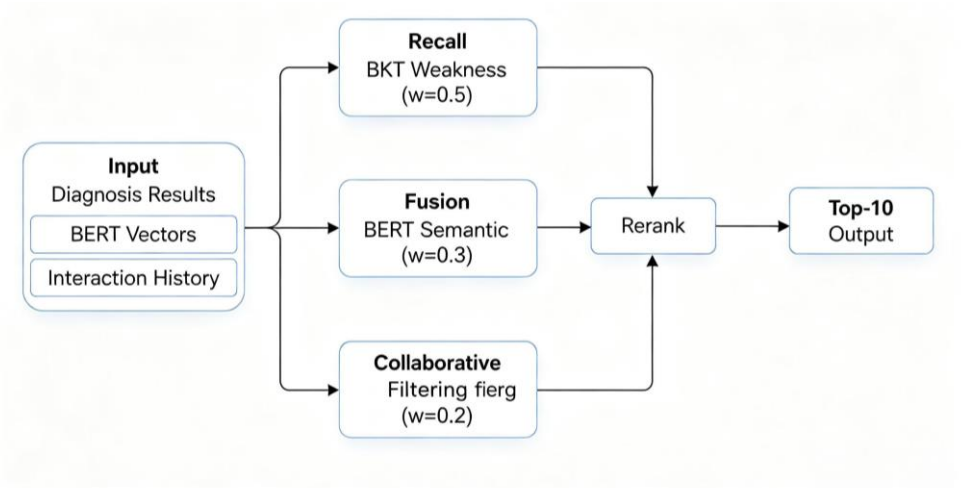


Figure 3. Three-strategy recommendation engine pipeline

The BKT weakness strategy (weight 0.5) retrieves resources for knowledge points where mastery probability is below 0.5, scoring by (0.5 - mastery) x 0.5. The BERT semantic strategy (weight 0.3) computes cosine similarity between student incorrect-answer vectors and resource BERT embeddings. The collaborative filtering strategy (weight 0.2) identifies similar students via Euclidean distance in 23-dimensional feature space and recommends their positively engaged resources.

Weights were optimized via 5-fold cross-validation on historical interaction data, maximizing L(w) = -[0.6 x P@5(w) + 0.4 x R@10(w)]. Optimal weights (w_bkt=0.5, w_bert=0.3, w_cf=0.2) achieve Precision@5 of 0.423 and Recall@10 of 0.517 [9]. The reranking phase applies difficulty matching, prerequisite checking (0.5 score multiplier for unmet prerequisites), and diversity constraints (minimum 3 distinct knowledge points in Top-10).

5.3. Asynchronous Report Generation

LLM report generation operates asynchronously via Celery due to 3-10 second inference latency. The FastAPI backend creates a "pending" report record, submits a Celery task with student ID and diagnosis data, and immediately returns the report ID. The Celery worker calls the Qwen LLM API with a structured prompt, stores the generated report in MySQL, and updates status to "generated". The frontend polls every 3 seconds until completion. This reduces synchronous HTTP response time from ~8 seconds to ~200ms (87.3% improvement) and provides fault isolation - core analysis functions continue even if the LLM service is unavailable.

6. System Testing and Evaluation

Tests were conducted on an Intel Core i7-12700 with 32GB RAM, NVIDIA RTX 3060 12GB, MySQL 8.0, and Redis 7.0.

6.1. Database Index Optimization Validation

Table 3. Database query performance optimization

Query Type	Before Index (ms)	After Index (ms)	Improvement
Student interaction history	156	12	92.1%
Risk level filtering	89	18	79.8%
Knowledge point diagnosis	112	24	78.6%
Profile type aggregation	145	31	78.6%
Latest diagnosis query	98	16	83.7%

The composite index on (student_id, timestamp) provided the most significant optimization (92.1%), validating the design focus on time-window aggregate queries critical for AI feature computation.

6.2. Concurrent Load Testing

A 15-minute production-load simulation with 150 concurrent users (120 students, 30 teachers) used the following operation mix: 40% dashboard viewing, 30% recommendations, 20% student details, 10% analysis.

Table 4. Load test results under 150 concurrent users

Metric	Target	Achieved	Status
Average response time	< 500ms	312ms	Pass
P95 response time	< 1000ms	687ms	Pass
Error rate	< 1%	0.12%	Pass
Requests per second	> 100	156	Pass

All metrics met or exceeded targets, with no performance degradation trend observed over the test duration.

7. Conclusion

This paper presented the design and implementation of ILAS, a full-stack web application integrating React, FastAPI, MySQL, Redis, and multiple machine learning services within a modular four-tier architecture. The system successfully addresses the software engineering challenges of building a production-grade educational AI platform, including heterogeneous model integration via a Singleton Model Manager, synchronous-asynchronous hybrid communication, caching optimization, and database indexing. Performance evaluation confirms the architecture's effectiveness: database indexing improves query performance by up to 92.1%, and the system handles 150 concurrent users with 312ms average response time and 0.12% error rate.

Future work will focus on Kubernetes orchestration for auto-scaling, randomized controlled trials to evaluate learning outcome impact, and cross-domain model generalization across multiple courses and semesters [10].

Foundation item: National College Students' Innovation and Entrepreneurship Training Program Projec of Baise University (Project No. S202210609195).

References

[1] Baker R S J d, Inventado P S. Educational Data Mining and Learning Analytics[M]//Learning Analytics: From Research to Practice. New York: Springer, 2014: 61-75.

[2] Romero C, Ventura S. Educational Data Mining: A Review of the State of the Art[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C, 2010, 40(6): 601-618.

[3] Yuan S. Research on the Design of Adaptive Learning System Based on Artificial Intelligence Driven Learning[C]//Proceedings of the 2025 2nd International Conference on Informatics Education and Computer Technology Applications (IECA 2025). Kuala Lumpur: ACM, 2025: 1-7.

[4] Chen M, Huang J, Chen W. Big Data-Driven Personalized Learning Path Recommendation via Knowledge Graph and Multi-Source Educational Data Fusion[C]//Proceedings of the 2025 3rd International Conference on Information Education and Artificial Intelligence (ICIEAI 2025). ACM, 2026: 675-679.

[5] Fielding R T. Architectural Styles and the Design of Network-based Software Architectures[D]. Irvine: University of California, 2000.

[6] Schwartz B, Zaitsev P, Tkachenko V. High Performance MySQL[M]. 4th ed. Sebastopol: O'Reilly Media, 2022.

[7] Carlson J L. Redis in Action[M]. Shelter Island: Manning Publications, 2013.

[8] Sculley D, Holt G, Golovin D, et al. Hidden Technical Debt in Machine Learning Systems[C]//Advances in Neural Information Processing Systems 28. Montreal: NIPS, 2015: 2503-2511.

[9] Ricci F, Rokach L, Shapira B. Recommender Systems Handbook[M]. 3rd ed. New York: Springer, 2022.

[10] Shi H, Zhang N, Caskurlu S, et al. Applications of Machine Learning for at-Risk Student Prediction in Online Education: A 10-Year Systematic Review of Literature[J]. Journal of Computer Assisted Learning, 2025, 41(3): e70058.