Industry Science and Engineering Vol. 3 No. 6, 2026

Research and Practice of Online Course Management System Based on Big Data and Full-Stack Technology

Jiahui Yao[1], Yi Wang[1], Qingfeng Zhou[2], Jiangang Zhang[2]
[1]*College of Artificial Intelligence and Big Data*, *Henan University of Technology*, *Zhengzhou*, *Henan*, China
[2]*iFLYTEK Co., Ltd.*, *Hefei*, *Anhui*, *China*

Abstract: To address the shortcomings of traditional online education platforms in data-driven decision-making, personalized learning support, and system stability, this paper designs and implements an Online Course Management System (OCMS) that integrates big data analytics and full-stack development. From the perspective of big data products, a data pipeline based on Flume, Kafka, Spark, and ClickHouse is constructed to collect, clean, compute, and perform multi-dimensional analysis of learning behavior data, and generate visual dashboards for course operations and student profiles using ECharts. Furthermore, a front-end and back-end separation architecture is adopted, with the front-end built on Vue 3+TypeScript and the back-end on Spring Boot, implementing core teaching functions such as JWT-based permission isolation, course resource management, random exam paper generation, and automated grading. Finally, to ensure system quality, functional, interface, and performance tests are conducted, and a full-chain containerized deployment and monitoring are realized using Docker Compose. Practical results demonstrate that the system provides personalized teaching and learning experiences, data-driven support for platform operation decisions, and exhibits good stability and maintainability.

Keywords: Online Course Management System; Big Data Analytics; Spring Boot; Vue 3; Software Testing; Docker

1. Introduction

Against the background of in-depth integration of the Internet and education, online education has become an important part of teaching in colleges and universities. However, traditional course management systems usually focus on the simple release and viewing of resources, lacking in-depth mining of students' learning behaviors, personalized learning path recommendation, and stability guarantee under high-concurrency scenarios [1]. Meanwhile, how to use big data technology to analyze massive learning behavior data so as to optimize teaching resources and improve teaching quality has become a research hotspot in the field of educational informatization [1].

For this reason, this paper designs and implements an Online Course Management System (OCMS). The system is developed from three dimensions:

(1) Big data product design: Build a complete pipeline for data collection, transmission, calculation and visualization, conduct multi-dimensional analysis on learning behavior data, and provide data support for teaching decision-making.

(2) Front-end and back-end system development: Adopt the Spring Boot+Vue 3 technology stack to implement core business functions such as permission management, course resource management and online examination.

(3) Testing, operation and maintenance guarantee: Conduct functional, interface and performance tests, and realize one-click containerized deployment and monitoring via Docker Compose to ensure the system is stable and reliable.

(4) Subsequent chapters will elaborate on the design and implementation of the big data analysis system, the development of core front-end and back-end functions, system testing and operation and maintenance strategies, and present the operation results.

2. Design and Implementation of Big Data Analysis System

2.1 Big Data Requirement Analysis

The data of this system comes from business

databases (users, courses, scores), front-end buried-point logs (video playback, pause, click and other behaviors) and static course data. In terms of data scale, thousands to tens of thousands of learning behavior log records are generated daily, requiring support for offline statistics and near-real-time queries. Core big data requirements include: statistics of learning duration and activity, analysis of score distribution and knowledge point mastery, trend analysis of course popularity and completion rate, and identification and early warning of at-risk students.

Timeliness requirements: Offline calculation (T+1) for learning duration statistics and score ranking, and second-to-minute-level response for learning progress saving and homework reminders.

2.2 Overall Big Data Architecture

Based on the above requirements, a layered data pipeline is designed with data flowing upward from the bottom, divided into six layers:

(1) Data Source Layer: Business databases, front-end buried points, server logs, course CSV files.

(2) Transmission Layer: Kafka message queue with topics such as learn_behav and exam_data to decouple data production and consumption.

(3) Collection Layer: Flume collects log and file data and pushes it to Kafka.

(4) Computation Layer: Spark performs data cleaning, feature engineering and aggregation calculation.

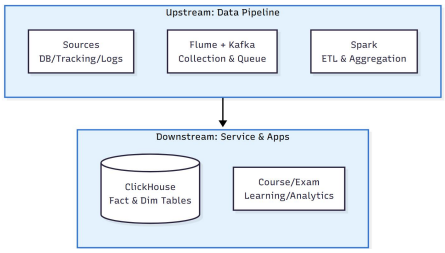

Figure 1. Overall Big Data Architecture

(5) Query Layer: ClickHouse columnar storage, building fact tables and dimension tables to support fast OLAP queries.

(6) Application Layer: Visual dashboards, student portraits, AI assistants, etc.

The overall architecture is shown in Figure 1. For detailed component selection and data processing workflows, refer to typical big data platform construction methods [2].

2.3 Deployment of Big Data Components and Data Processing

(1) Cluster Environment and Component Selection: A three-node cluster (node1~node3) is built. The versions and selection basis of each component are shown in Table 1.

(2) Data Collection and Transmission：Flume's SpoolDirSource monitors the /course_data/raw/ directory to collect CSV files and writes data to Kafka's course_data_topic, with interceptors configured to add timestamps and host information; Flume's TailDirSource is used to collect front-end buried-point logs in real time and transmit them to Kafka's user_behavior_topic. Meanwhile, DataX is configured with JSON tasks to incrementally synchronize user and course tables in MySQL to the Hive ODS layer.

(3) Data Processing and Calculation: Spark is used for offline data cleaning and aggregation. First, abnormal values in the data are processed and missing values are filled for data cleaning; then derived features such as learning efficiency and participation score are calculated for feature engineering; subsequently, indicators such as the number of students, average scores and average learning duration are counted by age, score level and learning duration dimensions for aggregation analysis. Finally, cleaned data is written to HDFS in Parquet format, and aggregation results are written to the ClickHouse ADS layer.

(4) Data Query and Visualization：A star schema consisting of dimension tables (student table dim_student, course table dim_course, grade table dim_grade) and a learning fact table fact_student_learning is built in ClickHouse. Materialized views are created based on this schema to precompute score summaries, and SQL queries are written to support visual dashboards. The front-end calls statistical interfaces and uses ECharts to render 8 charts such as a pie chart of course category distribution, a scatter plot of learning duration-score correlation, and a line chart of activity trend, providing intuitive data-driven decision support for administrators and teachers [3].

Table 1. Selection and Versions of Big Data Components

Component	Version	Selection Rationale
Flume	1.9.0	Lightweight, easy to deploy, supports breakpoint resume transmission, well integrated with Kafka

Kafka	2.8.0	High throughput, message persistence, suitable for log collection scenarios
Spark	3.2.0	Fast in-memory computing, unified batch-stream processing, MLlib facilitates subsequent portrait construction
ClickHouse	22.8 LTS	Columnar storage, high compression ratio, second-level query for billion-level data
DataX	3.0.0	Heterogeneous data source synchronization, simple configuration, supports ClickHouse writing

3 Front-End and Back-End Development of Online Course System

3.1 System Functional Requirements

Based on the goals of big data analysis, the system needs to provide core functions for three types of users:

(1) Administrators: User management, RBAC permission configuration, big data dashboard viewing.

(2) Teachers: Course/chapter tree maintenance, teaching resource upload, question bank management, random test paper generation and exam release.

Students: Course browsing and selection, video learning (breakpoint playback), online examination, score viewing, AI assistant Q&A.

3.2 Front-End and Back-End Technology Selection and Architecture

A B/S front-end and back-end separation architecture is adopted, with technology selection shown in Table 2. Similar technology stacks have been successfully applied in multiple online course systems [4].

Table 2. Front-end and Back-end Technology Selection

Category	Technology Stack	Description
Front-end	Vue3 + TypeScript + Element Plus	Responsive framework, strong type support
Back-end	SpringBoot+MyBatis	RESTful API, lightweight ORM
Database	MySQL 8 + Redis 7	Business data persistence, hotspot caching
Object Storage	MinIO	Teaching video and document storage
AI Integration	LangChain4j + QWEN API	Intelligent Q&A and learning suggestions

3.3 Implementation of Key Modules

(1) Permission and Authentication：

The system adopts JWT + Interceptor + Custom Annotation to implement unified permission control. The back-end defines custom annotations @RequireLogin and @RequirePermission to mark interface access permissions. The AuthInterceptor parses JWT tokens in the preprocessing stage of requests, obtains user identity and permission codes, stores them in ThreadLocal, and verifies them against the permissions declared by interface annotations. The front-end implements global request processing via Axios interceptors, automatically adding the Authorization: Bearer <token> header to requests, completing unified front-end and back-end authentication and ensuring secure interface access.

(2) Course and Resource Management:

Teachers can perform CRUD operations on courses and maintain infinite-level chapter trees (via the parent_id field). Videos, documents and other resources are received via MultipartFile, stored in MinIO, and resource paths and associated course/chapter IDs are recorded in the database. Resource access is securely controlled via pre-signed URLs.

(3) Intelligent Examination System:

The system supports teachers to configure test paper generation strategies based on question type, difficulty, quantity and score. The back-end randomly selects questions from the question bank according to the strategy, removes duplicates, automatically generates standardized test papers, and persists test paper information in exam_paper and exam_paper_question tables. Students can load test papers online, and the front-end controls exam timing via a countdown component, automatically submitting answers when time expires. The back-end receives student answers, automatically grades objective questions by comparing with preset answers, updates exam status and total score in the exam_attempt table, and generates exam_answer detailed answer records.

(4) Data Visualization Dashboard:

The back-end executes SQL aggregation queries by associating student learning fact tables with various dimension tables, encapsulating and returning multi-dimensional statistical data at one time. The front-end uses ECharts combined with computed properties for efficient and responsive chart rendering. For example, in the scatter plot of learning duration and score correlation, visual mapping components map

learning scores to color gradients, intuitively showing their positive correlation and providing visual support for teaching analysis.

(5) Xiao Yao AI Assistant:

A Retrieval-Augmented Generation (RAG) process is built based on the LangChain4j framework. Course knowledge base documents are vectorized and stored in a vector database. When a user asks a question, the system first retrieves matching relevant knowledge text fragments, then generates accurate and reliable answers by combining the large model API. It also provides conversation session management, persisting complete chat records in ai_chat_session and ai_chat_message tables.

4 Technical Schemes for Core System Functions

4.1 RBAC Permission Management

RBAC (Role-Based Access Control) is a classic permission management model with the core idea of a three-level association mechanism of "User-Role-Permission". By assigning permissions to roles and roles to users, centralized management and flexible allocation of permissions are realized, avoiding management chaos caused by directly assigning permissions to users and reducing permission maintenance costs, which is suitable for complex system scenarios with multiple users, roles and permissions [5,6]. Compared with traditional Discretionary Access Control (DAC), the RBAC model has clear permission isolation, strong scalability and easy auditability, perfectly meeting the differentiated permission requirements of administrators, teachers and students in this online course management system.

The core elements of the RBAC model include four parts: User, Role, Permission, Resource. The association relationships are shown in Figure 2: many-to-many between users and roles, many-to-many between roles and permissions, and one-to-one/one-to-many between permissions and resources.

Combined with the Spring Boot+Vue 3 technology stack of the system, the overall architecture of RBAC permission management is designed and implemented, divided into three layers: front-end permission control layer, back-end permission verification layer and data persistence layer, realizing full-process control of front-end display control, back-end interface verification and data persistent storage. The architecture is shown in Figure 3. The core implementation process of system RBAC permission management is as follows: First, administrators configure the association between roles and permissions, and assign roles to users in the background, with all association data persisted in MySQL. Middle tables (user-role association table, role-permission association table) realize many-to-many relationships. Second, when users log in, the back-end generates a JWT token containing user ID and role ID and returns it to the front-end, which stores the token in localStorage for subsequent authentication. Third, the front-end intercepts all route requests via route guards, parses role information from the token, and dynamically renders accessible menus and buttons for corresponding roles to prevent unauthorized page access. Finally, when users initiate interface requests, the front-end automatically carries the JWT token in the request header. The back-end parses the token via a permission interceptor, obtains user roles and corresponding permission lists, and verifies access permissions against the custom @RequirePermission annotation. If verified, the interface logic is executed; otherwise, a "permission denied" prompt is returned.

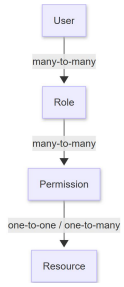

Figure 2. Schematic Diagram of the RBAC Model

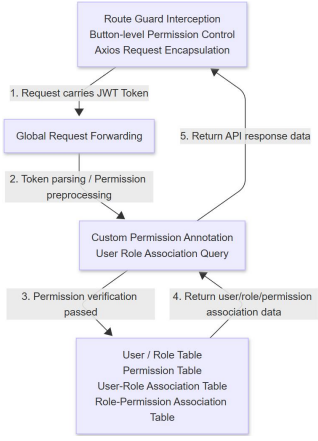

Figure 3. Implementation Architecture of System RBAC Permission Management

4.2 Video File Chunked Upload, Breakpoint Resume and Breakpoint Playback

Teaching videos in online course management systems are usually large in size (hundreds of MB to several GB) with complex upload scenarios (unstable networks, browser refreshes). Traditional one-time upload methods are prone to upload failures, progress loss and long time consumption, seriously affecting user experience. Meanwhile, students need breakpoint playback when watching videos, i.e., no need to restart from the beginning next time. To address this, chunked upload, breakpoint resume and breakpoint playback for video files are designed and implemented based on MinIO object storage, Vue 3 and Spring Boot.

Core Technical Principles:

(1) Chunked upload

Split large videos into fixed-size chunks (2MB, configurable) and upload them one by one. After all chunks are uploaded, merge them into a complete video, reducing single upload data volume and mitigating network fluctuation impacts [7].

(2) Breakpoint resume

Record uploaded chunk information to resume uploading only incomplete chunks after interruptions. The core lies in chunk ID + upload status record [7].

(3) Breakpoint playback

Record viewing progress (timestamp) in the database and load videos from the timestamp next time. The core lies in progress record + video positioning loading.

The overall architecture is divided into three layers: front-end upload layer, back-end processing layer and storage layer, as shown in Figure 4.

Core function implementation process:

(1) Chunked upload process

The front-end calculates the video file's MD5 as a unique identifier, splits it into 2MB chunks indexed sequentially. It first queries the back-end for uploaded chunk indices to avoid duplicates, then uploads chunks one by one with MD5, index and size. The back-end verifies chunk integrity and stores them temporarily in MinIO. After all chunks are uploaded, the front-end sends a merge request. The back-end merges chunks into a complete video, stores it in MinIO's formal directory, deletes temporary chunks and updates the video path in the database.

(2) Breakpoint resume process

After an interruption, re-uploading the same video recalculates MD5 and queries uploaded chunks. The back-end returns uploaded indices, and the front-end only uploads missing chunks. Real-time progress tracking and back-end status updates enable breakpoint recovery via MD5 and chunk index [7]. The detailed workflow of chunked upload and breakpoint resume is illustrated in Figure 5.

(3) Breakpoint playback process

The front-end listens to playback progress every 30 seconds and sends requests with user ID, video ID and timestamp. The back-end stores progress in the video_watch_progress table. Next time, the front-end queries the latest timestamp and positions the video via currentTime. Figure 6 shows the flowchart of this breakpoint playback process.

The system uses MultipartFile to receive chunks, MinIO's Java SDK for temporary storage/merging, and Axios for concurrent upload control. Breakpoint playback uses video.js for progress listening and MySQL for storage, solving large video upload pain points and enhancing user experience.

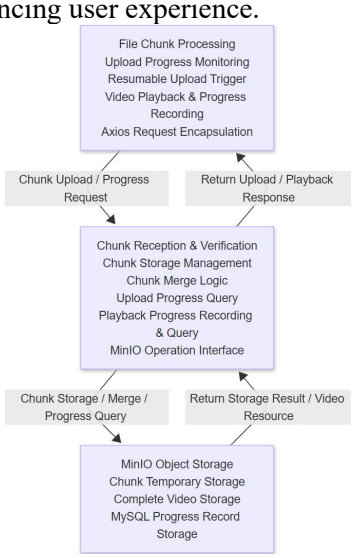

Figure 4. Architecture of Video Chunked Upload, Breakpoint Resume and Breakpoint Playback

4.3 RAG Vector Retrieval Implemented via LangChain4j

RAG (Retrieval-Augmented Generation) integrates information retrieval and text generation. Before generating answers, it retrieves relevant knowledge fragments from external knowledge bases, inputs them as context to large models, and generates accurate,

reliable answers, addressing issues like outdated knowledge, hallucinations and poor scenario adaptation in large models [8,9]. The Xiao Yao AI Assistant builds a RAG system via LangChain4j, combining QWEN and vector databases for course-specific intelligent Q&A.

LangChain4j is a lightweight, extensible Java framework for large models, encapsulating complete RAG processes (vector database integration, document embedding, retrieval strategy configuration, large model calls) [8]. The core RAG workflow: Document Loading & Preprocessing → Document Embedding & Storage → Query Embedding → Vector Retrieval → Context Assembly → Large Model Generation.

The overall architecture is shown in Figure 7.

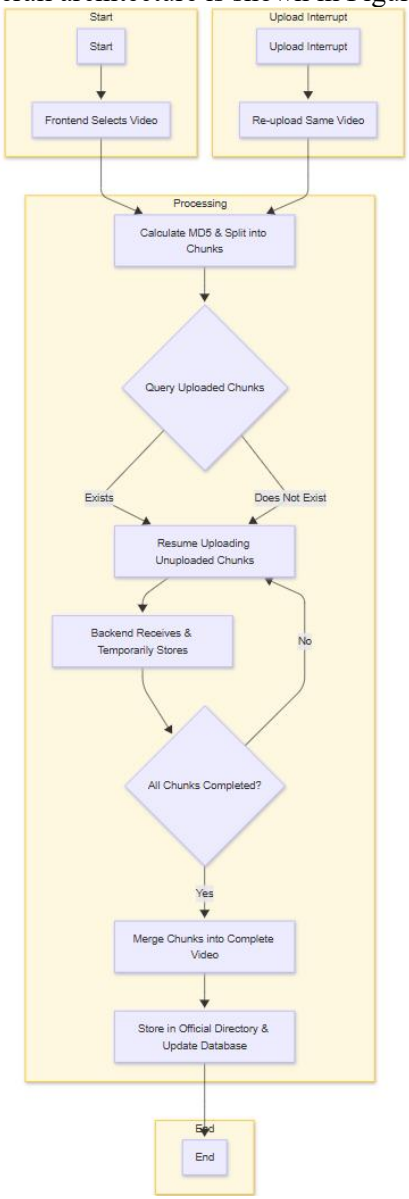

Figure 5. Flowchart of Video Chunked Upload and Breakpoint Resume

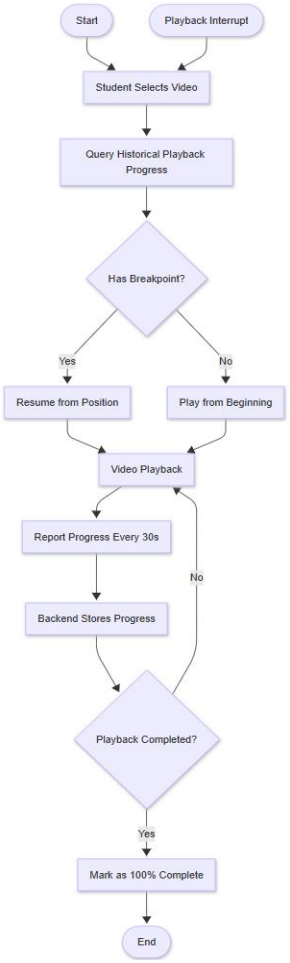

Figure 6. Flowchart of Video Breakpoint Playback

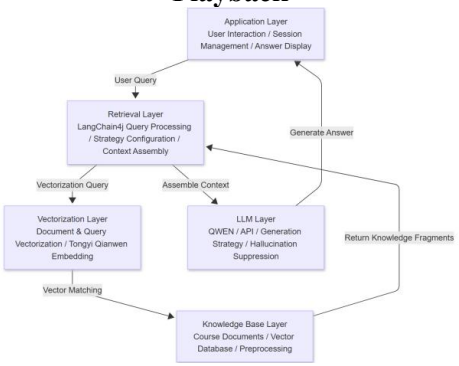

Figure 7. Architecture of RAG Vector Retrieval Based on LangChain4j

Core implementation details:

(1) Knowledge base construction & preprocessing

Load course documents (PPT, PDF, notes, exercises) via LangChain4j's DocumentLoader, split into 500-character fragments, remove irrelevant content and extract core information [9].

(2) Document embedding & storage

Use QWEN's Embedding model to convert fragments into 768-dimensional vectors, storing

vectors, text fragments, document IDs and tags in Milvus (lightweight, efficient for small-scale knowledge bases) [8].

(3) Query processing & vector retrieval

Convert user queries into vectors, retrieve Top5 relevant fragments via cosine similarity, and assemble context [9].

(4) Context assembly & large model generation

Assemble fragments, queries and prompts, send to QWEN API, format answers and store conversations in MySQL tables (ai_chat_session, ai_chat_message) for session management [8]. The complete RAG workflow is presented in Figure 8.

Flowchart.

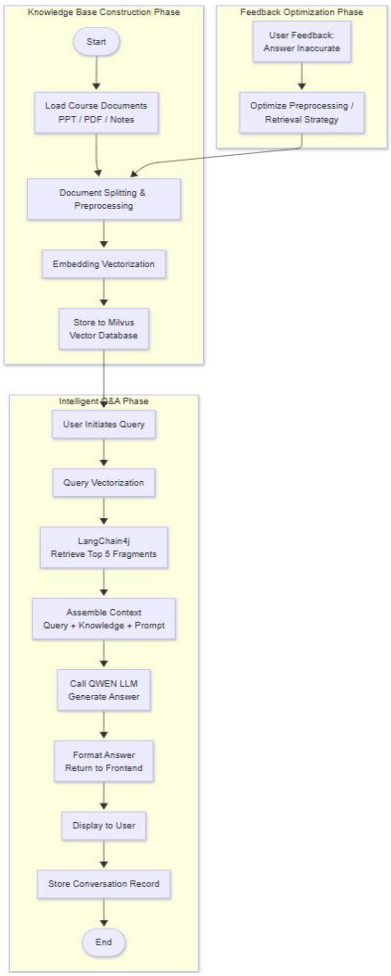

Figure 8. Flowchart of RAG Vector Retrieval Based on LangChain4j

LangChain4j simplifies RAG development by orchestrating embedding, retrieval, assembly and model calls. Vector retrieval ensures relevance and accuracy, while QWEN's generation capabilities deliver course-specific answers, addressing large model knowledge gaps [9].

5. System Testing, Operation and Maintenance

5.1 Testing Strategy and Design

A layered testing strategy is adopted, including functional testing, interface automation testing, performance testing and defect management. 35 test cases are designed covering core modules (login, course management, exams, dashboards) with P0/P1/P2 priorities.

5.2 Functional and Interface Testing

Manual Functional Testing: 91.43% pass rate in Chrome, 6 defects identified (missing duplicate course check, no auto-submit on exam timeout, etc.).

Interface Automation Testing: 15 Postman scripts for core interfaces, 8 Python+Requests scripts for login, course list and AI dialogue. Fiddler simulates weak networks/abnormal parameters to verify back-end fault tolerance.

5.3 Performance Testing

JMeter simulates 50 concurrent users for login and course list interfaces. Results in Table 3 show average response time <100ms, 0% error rate, ~20 requests/second throughput, meeting requirements.

Table 3. Performance Test Results (50 Concurrency)

Interface	Avg. Response Time (ms)	99% Response Time (ms)	Error Rate
/api/auth/login	97	123	0%
/api/course/list	49	56	0%

5.4 Defect Management and Repair

Defects are tracked via ZenTao (New → Active → Assigned → Resolved → Closed). A typical "500 error on exam submission" defect was fixed by resolving a null pointer exception and verified in regression tests.

5.5 Containerized Deployment and Monitoring

Multi-stage Dockerfiles build back-end (OpenJDK 17 JRE) and front-end (Nginx) images. docker-compose.yml orchestrates MySQL, Redis, MinIO, backend and frontend with custom networks and persistent volumes. Nginx configures try_files for Vue Router History mode 404 errors and /api/ proxy to the back-end.

Zabbix monitors server CPU, memory, disk and

container status with email alerts for CPU>80%, ensuring operational observability [10].

6. Operation Results and Analysis

6.1 Front-end and Back-end Function Demonstration

Deployed at http://150.158.2.239, the system supports student course browsing/selection, breakpoint video playback, exam countdown and auto-grading. Teachers create courses, upload videos and configure random tests. Administrators manage users and view big data dashboards.

6.2 Big Data Analysis Results

ClickHouse queries show:
Score distribution: A (18%), B (32%), C (35%), D/F (15%).
Learning duration-score correlation: 0.67 positive correlation, high-duration groups score 23 points higher.
Activity trends: Peaks at start/end of semesters.Visual dashboards provide intuitive insights for teaching intervention.

6.3 Testing and Deployment Results

Docker Compose starts all 5 containers successfully. Postman test pass rate 100%, JMeter stress tests error-free. The system is responsive under 50 concurrency, with consistent environments via containerization.

7. Conclusion and Outlook

This paper completes full engineering practice from three dimensions: big data product design, front-end/back-end development, testing and operation/maintenance. Key achievements:
(1) Built a Flume+Kafka+Spark+ClickHouse pipeline for learning behavior analysis and visualization, supporting multi-dimensional learning analytics.
(2) Developed a comprehensive OCMS via Spring Boot+Vue 3, implementing permission isolation, course management, random testing, auto-grading and AI assistant integration.
(3) Conducted systematic testing and containerized deployment, verifying functional correctness, interface stability and concurrency performance for production readiness.
Future Work:
(1) Introduce real-time frameworks (e.g., Flink) for second-level learning progress updates and real-time early warnings.
(2) Enhance mobile adaptation and explore cross-platform applications.
(3) Implement personalized course recommendation via collaborative filtering.
(4) Build comprehensive monitoring via Prometheus+Grafana.

References

[1] Liu J, Zhao W, Chen M. Research on the Application of Big Data Technology in Online Education Platforms. Journal of Data Acquisition and Processing, 2023, 38(4): 789-798.
[2] Wang H, Li M, Zhang L. Construction of Big Data Analysis Platform Based on Spark and ClickHouse. Big Data Research, 2023, 9(2): 102-110.
[3] Zhao Q, Wang L, Chen M. Application and Optimization of ECharts in Big Data Visualization Dashboards. Computer Engineering and Design, 2023, 44(6): 1789-1796
[4] Wang F, Li J, Zhang M. Design and Implementation of Online Course Management System Based on Spring Boot+Vue. Computer Technology and Development, 2022, 32(7): 156-161.
[5] Zhang H, Li L, Wang H. Design of Permission Management System Based on RBAC Model. Application Research of Computers, 2022, 39(5): 1489-1492.
[6] Li G. Computer System Security and Permission Management Technology. Beijing: Publishing House of Electronics Industry, 2021: 123-145.
[7] Zhang L, Li J, Wang H. Implementation of Large File Chunked Upload and Breakpoint Resume Based on MinIO. Computer Engineering and Applications, 2023, 59(12): 189-196.
[8] Chen M, Liu M, Zhao Y. Research on RAG Technology in Intelligent Q&A Systems Based on LangChain4j. Computer Science, 2024, 51(3): 201-208.
[9] Li L, Zhang Q, Wu D. Optimization and Application Practice of Retrieval-Augmented Generation (RAG) Technology. Journal of Artificial Intelligence, 2024, 40(2): 321-330.
[10] Chen Y, Zhou M, Li J. Practice of Microservice Containerized Deployment Based on Docker Compose. Software Engineering, 2023, 26(8): 45-50.