ISSN: 2688-8645

Design and Implementation of a Collaborative Service System for Campus Self-Study Resources

Chengyue Yang, Jianzhen Chen, Quan Shi, Yisong Liu, Yi Yang

School of Computer Science and Artificial Intelligence, Wuzi University, Beijing, 101100, China

Abstract

To address the information asymmetry of classroom occupancy, scattered course schedules, and the low efficiency of finding study spaces on campus, this paper designs and implements a campus study-resource collaborative service system based on a WeChat Mini Program. The system adopts an integrated architecture combining the Mini Program and cloud development. Cloud functions and a cloud database support timetable management, crowd-sourced classroom-status reporting, vacant-classroom queries, study-space recommendation, and study-record management. A rule-based recommendation method is constructed by considering course schedules, the freshness of crowd-sourced reports, users' historical study behavior, and classroom capacity. Recommendation reasons are generated from the activated rules to improve interpretability. Classroom status is collected through voluntary user reports, while the latest report for each user-room pair is retained for status updates. Functional and performance tests are conducted in a self-built simulation environment. All 58 test cases covering cloud functions, page interactions, and boundary/error scenarios pass. With 12 classrooms, 12 timetable records, and 30 study records, the recommendation logic is called 20 times in each of two scenarios: without filters and with floor 3 and a minimum capacity of 80. The median local computation times are 0.08 ms and 0.04 ms, respectively, and the maximum observed times are 0.66 ms and 1.95 ms, respectively. These measurements reflect computation in a local in-memory simulation environment only; they exclude network round trips, cloud-function cold starts, and actual cloud-database I/O, and therefore do not represent end-to-end response times. The results indicate that the system supports basic campus study-space querying, collaborative status reporting, and rule-based recommendation, and provides a low-cost implementation approach for campus learning-space services.

Keywords

WeChat Mini Program; Smart Campus; Crowd Sensing; Rule-based Recommendation; Study Space.

1. Introduction

University students have a continuous need for self-study spaces during their daily studies; however, the actual occupancy status of classrooms exhibits significant temporal variability. Traditional academic administration systems primarily provide course scheduling information and struggle to directly reflect dynamic situations such as temporary occupancy, early class endings, or student self-study sessions; consequently, students typically rely on personal experience, on-site inspections, or inquiries to peers to find available classrooms, which incurs a high cost for information acquisition. Meanwhile, course information is scattered across various channels, including webpages of the academic administration system, screenshots, and class groups, leaving room for improvement in the mobile query experience. Recommendation systems can provide ranked results based on user preferences and the characteristics of candidate objects. Existing research has discussed basic recommendation system architectures

ISSN: 2688-8645

and collaborative filtering methods [1-5]. However, recommendations for campus self-study spaces are subject to strong spatio-temporal constraints; furthermore, during the initial deployment phase, systems often lack sufficient user-classroom interaction data, making the direct application of collaborative filtering susceptible to the "cold start" problem and data sparsity. Therefore, this paper adopts explainable rule-based recommendation as the initial solution and integrates crowd-sourced classroom status reporting with timetable information to form a collaborative service workflow tailored for campus self-study scenarios. The primary contributions of this paper include: designing a lightweight system architecture comprising a mini-program, cloud functions, and a cloud database; implementing functions for timetable management, classroom status reporting, map visualization, and learning records; developing a rule-based recommendation method that synthesizes comprehensive course status, reporting timeliness, historical preferences, and capacity characteristics; and validating system functionality and interface performance within a simulation environment. It should be noted that the performance results presented in this paper are derived from simulation environments and do not equate to the long-term operational performance in a real-world campus setting. Based on existing literature, research closely related to this paper can be broadly categorized into three types. The first type comprises campus information services or resource management systems based on WeChat mini-programs; these focus on the engineering implementation of functions such as timetable queries, campus navigation, lost-and-found services, and venue requests, and typically do not involve collaborative perception or recommendation ranking of real-time classroom status [11-13]. The second type focuses on the intelligent identification and scheduling of classroom resources—for example, using visual models or device-side data to determine classroom occupancy status. While these approaches offer high perception accuracy, they rely on camera deployment or dedicated hardware, which consequently increases deployment costs and privacy constraints. The third type concerns recommendation system research within smart campus scenarios; systematic mapping efforts have already delineated the recommendation objects, contextual signals, and method types in this field, with collaborative filtering and hybrid recommendation methods accounting for a relatively high proportion. However, lightweight, interpretable solutions tailored for the strong spatiotemporal constraint of "classroom-level idle status" remain relatively scarce [14-15]. The aforementioned work provides functional references and a methodological background for this paper, while also leaving room for specific engineering trade-offs: in campus environments lacking historical user–classroom interaction data and without the hardware infrastructure for visual perception, is it possible to complete the closed loop of "status acquisition ightarrow candidate screening ightarrow interpretable ranking" at a low cost? This paper does not seek to propose a new model at the recommendation algorithm level; instead, it organizes crowd-sourced status reporting, timetable constraints, and manual rule-based scoring into a single operational, interpretable, and maintainable service workflow, and evaluates its functional completeness and interface response performance within a simulation environment. This positioning also determines the scope of the contributions of this paper: engineering integration and scenario adaptation are given priority, while algorithmic innovation does not serve as the foundation for the paper's central argument.

2. Overall System Design

2.1. Requirement Analysis

Campus self-study environments offer a high degree of autonomy and flexibility. When selecting a study location, students typically need to comprehensively consider factors such as the classroom's location, opening hours, seating capacity, current occupancy, and their personal course schedules. Traditional methods of accessing study resources primarily rely on offline

ISSN: 2688-8645

inquiries, verbal communication among peers, or campus notices; these approaches suffer from issues such as fragmented information, delayed updates, and a lack of transparency regarding resource utilization. Consequently, when students seek a suitable study space, they often expend additional time on searching and comparison. Furthermore, the lack of an effective correlation between classroom availability and course schedules can lead to underutilized resource information. To address these scenarios, the system should provide support centered on resource queries, resource recommendations, and the recording of learning information. First, regarding resource queries, the system should be capable of displaying basic classroom information—including the classroom name, floor level, seating capacity, and current availability—and should support filtering by criteria such as floor level or capacity, thereby helping students quickly locate a study space that meets their specific needs. Second, regarding resource recommendations, the system should integrate the classroom's availability, course schedules, and user-defined filters to perform a comprehensive ranking of available study resources, providing students with actionable recommendations. The recommendation process should prioritize information timeliness, avoiding the use of resource statuses that have already changed as the basis for current recommendations. Furthermore, given that students may have ongoing learning and the need to document their learning activities at different times, the system should also provide a learning record management feature that enables users to record study duration and related information, facilitating the review of individual learning progress. For self-study resource information, the system should provide a corresponding data maintenance and update mechanism to ensure that data—including classroom information, course schedules, and learning records—is managed in a unified manner. In light of the above requirements, the system should include functions for displaying and filtering self-study resource information, rule-based collaborative resource recommendations, learning record management, and basic data maintenance. Additionally, the system should offer good usability and maintainability; its interface should be concise and intuitive, and its data processing logic should be as clear as possible, thereby meeting the basic needs for resource queries and collaborative use within campus self-study environments.

2.2. System Architecture

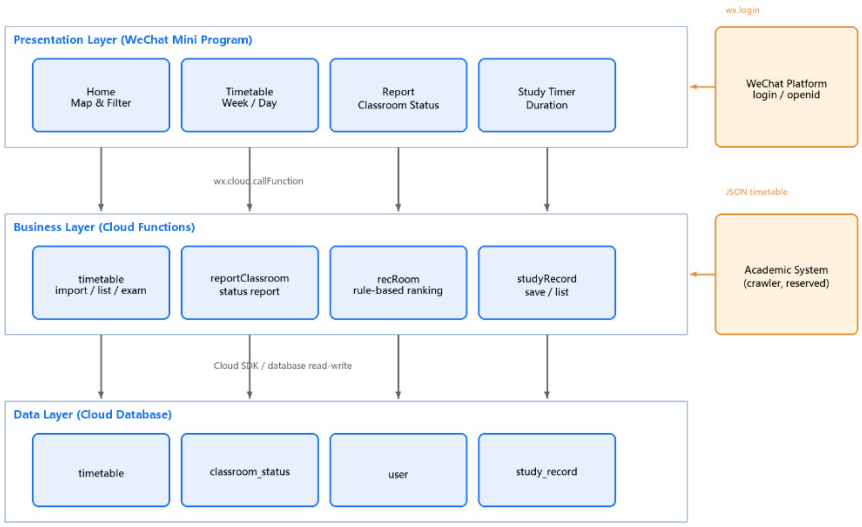

Figure 1. Overall System Architecture Diagram

The system adopts a three-tier architecture consisting of a presentation layer, a business layer, and a data layer. The presentation layer is implemented as a WeChat Mini Program, providing interactive pages for the home page, class schedule, status reporting, and study timer; the business layer comprises cloud functions that handle the class schedule, classroom status,

ISSN: 2688-8645

recommendations, and study records; and the data layer uses a document database in WeChat Cloud Development to store the class schedule, classroom status, user information, and study records. The Mini Program calls cloud functions via wx.cloud.callFunction, while the cloud functions access the database through the Cloud Development SDK. The system leverages the WeChat login session to retrieve the user's openid in the cloud, thereby avoiding the direct transmission of user identifiers from the frontend. The overall system architecture is shown in Figure 1.

The system backend primarily consists of cloud functions such as timetable, reportClassroom, recRoom, and studyRecord, which handle timetable import and query, classroom status reporting, self-study room recommendations, and study record management, respectively. The database is organized into four business-related collections: timetable, classroom_status, user, and study_record. This design separates page interaction from core business logic, facilitating subsequent independent maintenance and scalability.

2.3. Data Organization and Interfaces

The Schedule Collection stores information such as user identifiers, course names, instructors, days of the week, session numbers, classrooms, week numbers, and exam markers; the Classroom Status Collection stores reported users, classroom IDs, building, floor, capacity, status, and update times; the User Collection maintains WeChat user identifiers and nicknames; and the Learning Records Collection stores classroom information, start times, end times, and study duration. The original system design established query indexes for schedules based on openid, created lookup indexes for classroom status based on both openid and roomId, and set up filter indexes for fields such as status and floor. The Cloud Function interface adopts a unified return structure containing a status code, a notification message, and business data. The Schedule Module provides import functionality, list queries, and exam information queries; the Classroom Status Module receives building, classroom, and status information; the Recommendation Module accepts filtering criteria such as floor and minimum capacity, and returns a recommended list; and the Learning Records Module provides save functionality, list queries, and a summary of today's study duration.

3. Classroom State and Recommendation Methods Based on Crowdsourced Perception

3.1. Crowdsourced Collection of Classroom Status

The system collects classroom status through a reporting interface provided by a mini-program. After a user selects a building, classroom, and current status and submits the request, the server writes the data based on the user's identity and the classroom identifier. The original system employed a method of overlapping updates through repeated reports from the same user for the same classroom, thereby reducing redundant records generated by consecutive reports from the same user. During the recommendation phase, deduplication is performed based on classroom identifiers, with priority given to retaining newer status records; subsequently, idle classrooms are filtered based on the status field. Crowdsourced reporting offers the advantages of low cost and ease of deployment, but a single report may suffer from false positives, latency, or individual variability. The original system mitigated the impact of expired information on ranking results by recording update times and incorporating report freshness into the recommendation scoring. Currently, the implementation has not yet established a user reputation scoring system or a multi-user voting fusion mechanism; therefore, there is room for further improvement in the credibility of status data. The fusion of crowdsourced data credibility and quality control have been discussed in the context of mobile sensing and spatial state estimation; common approaches include multi-source voting, reputation weighting, and

ISSN: 2688-8645

time decay. In the present work, only temporal freshness is used as a quality adjustment mechanism, representing a minimal implementation within this research direction; future work could introduce the aforementioned mechanisms without altering the system architecture.

3.2. Candidate Classroom Screening

The recommendation process first reads the floor and minimum capacity constraints set by the user, then queries the classroom status data. The system deduplicates the results by classroom number and retains the most recent status; it subsequently filters for classrooms with a "available" status. If the user has specified floor or capacity constraints, the candidate set is further narrowed. Once the candidate classrooms have been identified, the system reads the course schedule for the current day and the user's recent learning records, which serve as input for the subsequent rule-based scoring.

3.3. Multidimensional Rule Scoring

The recommendation process is illustrated in Figure 2: after deduplication, availability screening, and conditional screening, candidate classrooms proceed to rule-based scoring and ranking; the rule scores are provided in Table 1.

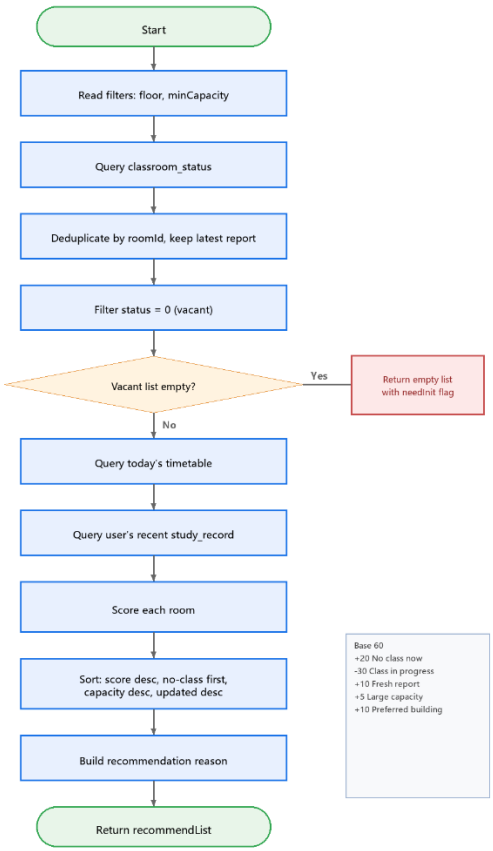

Figure 2. Recommended Flowchart

Let the candidate classroom be denoted as r, and let its comprehensive score be denoted as S (r). The original system employs a combination of a base score and score adjustments based o n rules; the scoring expression is

$$S(r)=60+20I[-class(r)]-30I[class(r)]+f_{fresh}(r)+f_{hist}(r)+f_{cap}(r)$$

ISSN: 2688-8645

Here, class(r) indicates whether there is currently a class in the classroom, while I[] is the indi cator function; f_fresh(r), f_hist(r), and f_cap(r) represent the scores corresponding to reporte d freshness, user historical preferences, and classroom capacity, respectively. The specific rule s are shown in Table 1.

Table 1. Recommendation Rules and Scoring

dimension	Trigger conditions	value
Classroom engagement	No classes currently	+20
Classroom engagement	Classes currently available	-30
Report freshness	No more than 30 min from the reporting deadline	+10
Report freshness	More than 30 min and no more than 120 min	+5
Report freshness	Exceeds 120 min but does not exceed 360 min	0
Report freshness	Over 360 min	-5
Historical preferences	Users frequently use this classroom.	+10
Historical preferences	Users frequently visit this building (not this classroom).	+5
capacity	Capacity shall not be less than 100	+5
capacity	Capacity:60-99	+3

When the overall scores are identical, the system ranks the results by priority of availability, descending capacity, and descending submission time; that is, classrooms with more recent submissions are ranked higher. To improve the interpretability of the results, the system generates recommendation reasons based on the matched rules—for example, "currently no classes available," "recently submitted," "the building you frequently use," or "large-capacity classrooms" —and returns these recommendations to the frontend along with the results. One frequently discussed advantage of rule-based recommendation is that its decision-making process can be traced item by item; a local rule match can serve as an explanatory basis without relying on post-hoc approximations. By using matched rules to generate recommendation reasons, this paper directly leverages this property; however, it should be noted that the weights of the rules are still set manually and have not yet been learned or calibrated using real-world behavioral data. This approach does not require reliance on large volumes of historical interaction data, making it suitable for early-stage system deployment.

3.4. Algorithm Complexity and Scalability

Let the number of candidate classrooms be n. Deduplication, filtering, and item-by-item scoring are primarily linear operations; the time complexity of the sorting step is O(n log n), resulting in relatively low overall computational overhead. A natural subsequent approach would be, after accumulating a sufficient amount of user–classroom behavioral data, to transform learning duration, visit frequency, and recent visits into implicit feedback, and to combine this with collaborative filtering and existing rule-based scores. This direction already has a relatively mature research foundation; however, it does not fall within the scope of the work completed in this paper, and this paper makes no prediction regarding its effectiveness.

ISSN: 2688-8645

4. System Implementation, System Implemention

4.1. Class Schedule Management

The timetable module supports batch import, weekly and daily view displays, as well as the extraction and notification of examination information. (The timetable page is shown in Figure 3(a)). Timetable data are isolated by user openid, and only the current user's data are returned during queries. The original system reserved an interface for data collection from academic administration systems and conducted offline validation of the timetable parsing process; however, because the page structures of academic administration systems vary across different schools, automated data collection capabilities must still be configured in accordance with the specific school environment and should not be described as being compatible with all academic administration systems.

4.2. Classroom Status Reporting and Map Display

Users complete the reporting process by selecting a building, classroom, and status (as shown in Figure 3(b)). The homepage generates map markers based on classroom status (as shown in Figure 3(c)), displays the spatial distribution of available classrooms, and supports filtering by floor and capacity. The map markers use the GCJ-02 coordinate system; the original paper documents the coordinate calibration for buildings such as the campus library, teaching buildings, and experimental teaching centers.

4.3. Recommendations and Learning Records

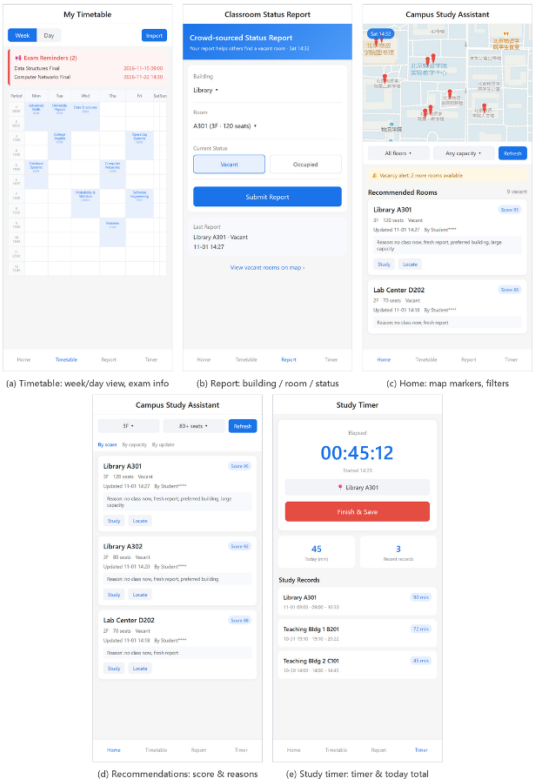

Figure 3. Screenshot of the system's key functional pages

(a) Class schedule page; (b) Status reporting page; (c) Home page map and filtering; (d) Recommended results page; (e) Study time tracking and logging page. Note: Interface data is sourced from the simulation test environment; field names and control captions are subject to the actual code

ISSN: 2688-8645

The recommended cloud function performs candidate query completion, deduplication of the latest status, course status determination, rule-based scoring, and generation of recommendation reasons (the recommendation results are shown in Figure 3(d)). After the user selects a classroom to begin self-study, the page records the start time; once the end time has been recorded, the start and end times are submitted, after which the backend calculates the study duration and saves the record. The study log module supports a historical list and a summary of today's study duration, as shown in Figure 3(e). At present, the recommended cloud function only performs rule scoring and reason generation; it does not involve model training or online learning. Consequently, its behavior can be fully reproduced using the rule table and input data.

4.4. Critical Exception Handling

Records from the original system testing and defect remediation indicate that, during development, the following issues were prioritized for resolution: the failure of the latest reported data when the status field was used in queries; the failure to overwrite older records when timestamps were identical; incorrect mapping between floor drop-down options and actual floor values; empty data upon first page load; and unclear error notifications in cloud functions. The corresponding remediation measures included retrieving the latest reported data first and then filtering by status; adjusting the time comparison criteria; establishing a mapping for floor values; adding a check for empty data during initialization; and refining error notifications.

5. System Testing and Result Analysis

5.1. Test Environment and Methods

Since the WeChat Developer Tools are difficult to drive directly from the command line, the original system built a Node.js-based emulation test environment, used in-memory databases to simulate common cloud database operations, and emulated wx-server-sdk and certain Mini Program APIs. The tests covered cloud function business logic, page interactions, boundary exceptions, and static consistency checks. The original test environment consisted of Node.js 24, an in-memory emulation database, and self-developed Mini Program API mocks; cloud functions used real business code.

Table 2. Test Environment

project	configure
Java runtime environment	Node.js 24
database	Memory Simulation Database
Mini Program API	In-house developed mock
Cloud Functions	Original code, unmodified

5.2. Functional and Abnormality Testing

The test cases cover scenarios including timetable import and query, user data isolation, classroom status reporting, duplicate reporting coverage, recommendation result filtering, map marker generation, page view switching, learning record storage, and failures in cloud functions. The original paper recorded a total of 58 test cases; in addition, one set of static consistency checks was conducted, and all test results passed. Some boundary and exception test results are shown in Table 3.

ISSN: 2688-8645

Table 3. Boundary and Abnormality Test Results (Excerpt)

Test Scenario	expected result	result
Home page: duplicate trigger onShow	Auto-fill only once	Pass through
Repeated loading of the class schedule	Automatic import performed only once	Pass through
Home Cloud Function call failed	Error prompt displayed; loading state reset	Pass through
Course Schedule Cloud Function call failed	Error prompt displayed; loading state reset	Pass through
Failed to submit to the cloud function in the classroom	Error prompt displayed and button state reset	Pass through
Learning timer save failed	Show error prompt while preserving the timer state	Pass through
Cloud function returns an error code	Display error messages without triggering automatic import	Pass through
No results found	Do not trigger auto-fill	Pass through
The vacancy state has changed	Silent initial load; prompt upon state change	Pass through

5.3. Interface Performance Testing

5.3.1. Definition of the Test Scope

This section evaluates the local execution time of the recRoom recommendation cloud functio n in a Node.js simulation environment, with a focus on processing stages such as candidate dat a retrieval, filtering, rule scoring, and sorting. The test database is a memory-based simulation database rather than a real cloud database used by WeChat Cloud Development. The timing re sults do not include the network round-trip time from the Mini Program client to the cloud, th e cloud function's cold start time, or the I/O latency of a real cloud database; therefore, this pa per defines them as algorithmic and business logic computation time, does not treat them as e nd-to-end response times, and does not use them to estimate user request latency in a real de ployment environment.

5.3.2. Testing Environment

Table 4. Test Environment Configuration

project	configure
CPU	Intel Core i9-14900HX, 32threads
internal storage	31.8 GB
OS	Windows 10.0.26200 x64
At runtime	Node.js v24.21.0
database	Memory-based simulation database (non-cloud database)
Timing method	process.hrtime

5.3.3. Data Initialization Method

All test data is written through the cloud function's own interfaces, ensuring that the data initialization path aligns with the business call path. The initialization sequence is as follows: call recRoom.init to write data for 12 classrooms; call timetable.import to write 12 timetable records; and call studyRecord.save to write 30 study records. Once data initialization is complete, the recommendation interface is warmed up three times; then, under two conditions—without any filtering criteria and with "floor = 3 and minimum capacity = 80" —it is called consecutively 20 times each, and the duration of each call is recorded.

ISSN: 2688-8645

5.3.4. Test Result

Table 5. Stepwise computation time of the recRoom interface (unit: ms)

Call Sequence Number	No screening	Floor 3 + capacity 80
1	0.12	0.04
2	0.16	1.95
3	0.12	0.07
4	0.10	1.18
5	0.66	0.07
6	0.14	0.05
7	0.10	0.04
8	0.07	0.04
9	0.10	0.04
10	0.06	0.04
11	0.05	0.05
12	0.05	0.06
13	0.05	0.03
14	0.05	0.04
15	0.07	0.03
16	0.30	0.04
17	0.10	0.42
18	0.06	0.04
19	0.06	0.03
20	0.05	0.03

Table 6. Time statistics under the two conditions (unit: ms)

spectacle	minimum	median	average value	P95	maximum
No screening	0.05	0.08	0.12	0.30	0.66
Floor 3 + capacity 80	0.03	0.04	0.21	1.18	1.95

In the unfiltered scenario, the median across 20 calls was 0.08 ms, the mean was 0.12 ms, and the maximum was 0.66 ms. In the filtering scenario with floor 3 and minimum capacity 80, the median was 0.04 ms, the mean was 0.21 ms, and the maximum was 1.95 ms. The two higher values of 1.18 ms and 1.95 ms in the filtering scenario occurred at the beginning of the call sequence; after excluding these two observations, the average execution time for the remaining 18 calls was approximately 0.06 ms. This exclusion result is intended solely to describe the observed distribution and does not replace the formal statistical values derived from all 20 calls.

5.3.5. Interpretation of Result

The median values in both scenarios are below 0.1 ms, indicating that, under current small-scale memory simulation data and local execution conditions, the computational overhead of the recommended business logic is relatively low. In the filtering scenario, the mean exceeds the median, primarily due to the influence of two higher observed values in the early stage; this indicates that fluctuations in the execution time of a small number of calls can significantly affect the mean of a small sample. Since the test sample consists of only 20 executions per scenario and excludes network latency, cold starts, and cloud database I/O, these results should be used solely to compare local computational overhead under the current simulation environment; they should not be interpreted as cloud function response times or actual user experience latency. Subsequent multi-round testing should still be conducted in a real cloud

ISSN: 2688-8645

development environment, and end-to-end metrics under concurrent load, cold start, and network and database access conditions should be reported.

5.4. Discussion of Results

Based on functional test results, the system has completed core workflows—including class schedules, status reporting, recommendations, and learning records—and has handled several common abnormal scenarios. Based on performance test results, rule computation and sorting overhead remains low under the current candidate scale. However, validation at this stage still has limitations: first, the volume of test data is relatively small, and large-scale concurrency and long-term stability testing have not yet been conducted; second, the simulation environment cannot fully account for the operational differences between the WeChat client, cloud networks, and real-world cloud databases; and third, there is currently no continuous real-world campus operational data available to evaluate the accuracy of crowdsourced reporting, recommendation hit rates, and user satisfaction. Consequently, the conclusions of this paper are limited to the system's functional implementation and validation within the simulation environment. Compared with classroom resource systems that rely on visual perception, this approach replaces hardware-based recognition with crowdsourced reporting; while this sacrifices some state accuracy, it achieves a significant reduction in deployment costs. Compared with smart campus systems focused on learning-resource recommendation, the candidate objects in this study are not courses or materials, but physical spaces with a clearly defined temporal and spatial occupancy status; therefore, rule design must prioritize "whether there is currently a class" and "reporting freshness." This distinction constitutes both the specific nature of the scenarios addressed in this paper and the fundamental constraint on the choice of methodology.

6. Conclusion

This paper designs and implements a campus self-study resource collaborative service system based on a WeChat Mini Program. The system adopts a lightweight architecture composed of a Mini Program, cloud functions, and a cloud database to implement functions such as timetable management, crowdsourced classroom status reporting, idle classroom map visualization, rule-based recommendations, and learning record management. To address the issue of insufficient early-user behavioral data, a rule-based scoring method is constructed that integrates course status, reporting freshness, historical preferences, and classroom capacity; recommendation reasons are then generated based on rule-matching information. During functional testing, all 58 test cases passed. In a local memory simulation environment comprising 12 classrooms, 12 timetable records, and 30 learning records, when the filtering scenario with no filters and the floor with 3 floors and a minimum capacity of 80 was called consecutively 20 times, the median computation time for the recommendation business logic was 0.08 ms and 0.04 ms, respectively, with maximum observed values of 0.66 ms and 1.95 ms, respectively. These results reflect only the computation time of the local algorithm and business logic and do not include network round-trip latency, cloud function cold starts, or actual cloud database I/O; therefore, they cannot be used to infer the end-to-end response time in a real deployment environment. The system still faces challenges, including the lack of reputation assessment in status reporting, recommendation weights that depend on manual configuration, limited parsing and adaptability in the academic affairs system, and insufficient testing in real-world scenarios. Future work may involve introducing multi-user state fusion and a user reputation mechanism to calibrate recommendation weights using actual learning behavior, as well as conducting continuous operational, concurrency performance, and user experience evaluations in actual campus environments. The contributions of this paper are limited to system design, functional implementation, and preliminary validation within a simulation environment; it

ISSN: 2688-8645

does not claim that the proposed method exhibits novelty at the algorithmic level. The manually assigned nature of rule weights, the small scale of the test data, and the gap between the simulation environment and real-world deployment collectively define the applicable boundaries of the present conclusions.

References

[1] Resnick, P., & Varian, H. R. (1997). Recommender systems. Communications of the ACM, 40(3), 56–58.

[2] Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (pp. 43–52).

[3] Sarwar, B., Karypis, G., Konstan, J., et al. (2001). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web (pp. 285–295).

[4] Goldberg, D., Nichols, D., Oki, B. M., et al. (1992). Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12), 61–70.

[5] Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, 7(1), 76–80.

[6] Xiang, L. (2012). Practice of recommender systems. Posts & Telecom Press. (In Chinese)

[7] Li, H. (2012). Statistical learning methods. Tsinghua University Press. (In Chinese)

[8] WeChat Open Documentation. (n.d.). Mini program development guide. https://developers.weixin.qq.com/miniprogram/dev/framework/.

[9] WeChat Open Documentation. (n.d.). WeChat cloud development documentation. https://developers.weixin.qq.com/miniprogram/dev/wxcloud/basis/getting-started.html.

[10] Python Software Foundation. (n.d.). Requests: HTTP for humans. https://requests.readthedocs.io/.

[11] Wang, Q. Y., Wang, C., & Tong, Z. X. (2023). Development of campus guide system based on WeChat Mini Program. Computer Programming Skills & Maintenance, (11), 79–81. (In Chinese)

[12] Ma, E. K., Niyazibieke, M., Yakufu, A., et al. (2023). Design and implementation of campus information service system based on WeChat Mini Program. Journal of Jilin Institute of Chemical Technology, 40(3), 53–55. (In Chinese)

[13] Wang, J. Z., Lei, J. J., & Wang, Y. M. (2024). Intelligent campus resource management system based on WeChat Mini Program. Information & Computer, (13), 85–88. (In Chinese)

[14] Maruyama, M. H. M., Silveira, L. W., da Silva Júnior, E., et al. (2025). Recommender systems in smart campus: A systematic mapping. Knowledge and Information Systems, 67(3), 2063–2089.

[15] Dalla Vecchia, A., Marastoni, N., Oliboni, B., & Quintarelli, E. (2023). ICARE: An intuitive context-aware recommender with explanations. In Proceedings of the 31st Symposium of Advanced Database Systems (SEBD 2023) (pp. 623–631). Galzignano Terme, Italy.