UNSW
SYDNEY

COMP9900 - Info Tech Project
F16A-AVACADO

(P14) AI Comment Moderation - RAG and Classification modelling - Project Proposal

Xuanzhi Liu
Scrum Master
z5455855@ad.unsw.edu.au

Xiaotong Zhang
Project Owner
z5491457@ad.unsw.edu.au

Yinglong Cui
Technicial Design
z5497698@ad.unsw.edu.au

Zhongwei Yang
Technicial Design
z5441208@ad.unsw.edu.au

Hang Pan
Technicial Design
z5598515@ad.unsw.edu.au

Jianhong Liu
Scrum Master
z5460416@ad.unsw.edu.au

March 10, 2025

Background

To support the moderation and management of student comment data, staff are currently tasked with manually checking thousands of student responses. This process includes identifying inappropriate content, classifying feedback, and ensuring that valuable insights are accurately captured. As artificial intelligence (AI) and large-scale language modeling (LLM) technologies evolve, they offer promising opportunities to improve the efficiency and consistency of comment review. AI-powered systems can automatically analyze qualitative feedback, flag potentially sensitive or inappropriate content, categorize reviews into relevant topics, and even generate summary reports or actionable recommendations. By integrating these tools, organizations can simplify the review process while maintaining accuracy and reducing the workload of manual moderators.

Problem Domain

This project focuses on comment moderation and management based on large-scale text in the context of higher education feedback(Wen et al. 2014). Each term, students submit a massive volume of open-ended comments regarding a course's "Areas for Improvement" or a tutorial's "Best Area", covering a wide range of content related to teaching quality, the learning experience, assessment criteria, or even containing inappropriate or potentially harmful language. Due to the sheer volume of comments, relying solely on traditional manual review on a comment-by-comment basis poses significant challenges in terms of both efficiency and consistency(Brunk et al. 2019).

Problems Being Solved

Existing manual Comment Moderation systems suffer from the following problems:

- Large volume of comments, high moderation cost.

  Tens of thousands of student feedback comments are scattered across different courses and semesters, often requiring multiple staff members to quickly read, categorize, and filter them. Among these comments, many must be further analyzed to identify potentially sensitive words, inappropriate language, or harmful content(Riehle et al. 2020).

- Difficulty ensuring consistent and accurate moderation.

  Because human reviewers may interpret and judge comments differently, the same type of feedback can receive inconsistent results. Simultaneously, when faced with a massive influx of comments, balancing moderation efficiency and quality becomes increasingly challenging(Gorwa et al. 2020).

- Lack of structured data and visualization.

  Educational feedback is often stored haphazardly in survey systems or spreadsheets, without a unified structure or clear presentation. As a result, conducting an in-depth data analysis or tracking historical records later can be quite difficult.

Related Projects

There are already a number of language analysis and moderation tools on the market for different application scenarios:

- OpenAI Content Moderation.

  OpenAI provides a set of content moderation strategies based on a large language model to filter inappropriate remarks, discriminatory words, and personal privacy information in online texts or chats.(Palla et al. 2025) The policy is often used in chat rooms, comment sections, and other real-time application scenarios. It has the advantage of strong modeling capabilities, based on the GPT family(Brown et al. 2020, Palla et al. 2025), and is able to understand a wide range of complex sentences and semantics. The model is fast in development and upgrading, adapts to newly emerged web languages, and can cover different languages, which can better support internationalization scenarios. However, because of their dependence on external APIs, colleges and universities may have data compliance risks when it comes to sensitive comments transmitted to the cloud(Radford et al. 2019). And due to the payment model or speed limit restrictions, budget and call efficiency need to be considered when facing large-scale comment review(Chen et al. 2014).

- Jigsaw Perspective API.

  The Perspective API was developed by Jigsaw (a Google subsidiary) to detect and evaluate harmful speech or inappropriate content in text(Lees et al. 2022, Hosseini et al. 2017). It uses machine learning models to identify hate speech, threats of violence, personal attacks, etc.that may be contained in comments, so that potentially offensive content can be automatically flagged or filtered in forums, social media, or comment sections. Its strength lies in providing a quantitative score for 'toxicity' or 'insult', which helps developers to quickly filter or flag content based on thresholds. This detection feature can be easily embedded into your own system by simply calling a RESTful API(Ehsan et al. 2022), reducing development costs. However, the model focuses on general-purpose online speech, whereas misclassification may occur in educational, academic, or professional scenarios, requiring additional customization or fine-tuning for domain terminology(Pozzobon et al. 2023, Nogara et al. 2023, Mihaljević & Steffen 2023). Similarly, uploading comment data to a cloud-based model for testing requires an assessment of data security compliance, especially when sensitive student information is involved in a university setting.

- AWS Comprehend.

  AWS Comprehend is a natural language processing service provided by Amazon that supports sentiment analysis, topic modeling, and entity recognition of text, and can be combined with custom classifiers to detect or filter inappropriate content(Kadaskar & Kamthe 2024). In general-purpose scenarios, it helps developers quickly categorize and analyze massive amounts of text. It has the strength of allowing users to fine-tune domain-specific training data to enhance the accuracy of recognizing domain terms or sensitive items. It is also adaptable to internationalized teaching environments, as official support

for multiple languages is constantly being expanded. Its limitation is that calling and training custom classifiers may incur high costs and is less friendly to projects with limited budgets or one-time requirements (Bagai 2024, Ivan et al. 2024).

Based on the projects mentioned above, three common weaknesses can be summarized for the projects that currently exist in the market:

- Possible misjudgment of specialized vocabulary in the field of education.
- High cost of model prices.
- Protection of students' personal privacy needs to be emphasized.

Brief Summary

This project is a low-cost, professional and privacy-protected course evaluation moderation solution based on the whole process of "retrieval + pre-trained model + adaptive classification + manual review" formed by RAG(Lewis et al. 2020) and classification model. It not only solves the problem of manually reviewing a large number of students' comments, but also solves the shortcomings of existing products in the market.

User Stories

US-1: Import and Store Student Feedback Data

User Story: As a system administrator, I want to import and store student feedback data efficiently into a structured database, enabling effective moderation and analysis.

Acceptance Criteria:

- Successfully import and store feedback data.
- Logs should be accessible for monitoring and export.
- Data should be validated before storage to ensure consistency.

US-2: Automated Ingestion of Student Comment Data

User Story: As a system administrator, I want the system to automatically ingest and store student comment data into a structured database, enabling efficient moderation and analysis.

Acceptance Criteria:

- Data is successfully stored in a structured format.
- Admin can view logs and export data from the admin interface.
- The system should process new data in near real-time.

US-3: Retrieval of Similar Historical Comments

User Story: As a system, when a new student comment arrives, I want to retrieve top-N similar historical comments and pass the results to an LLM for classification, so that the new comment can be automatically categorized with appropriate labels and confidence scores.

Acceptance Criteria:

- Successfully retrieve top-N similar historical comments.
- LLM generates accurate labels and confidence scores for comments.
- Retrieval should be performed within an acceptable time threshold.

US-4: Moderation Interface for Flagged Comments

User Story: As a moderator, when the confidence score for a classification is low or indicates a potentially risky comment, I want to view flagged comments and similar historical comments directly from the moderation interface, confirm or override classifications, and save results, so I can complete the moderation process.

Acceptance Criteria:

- Interface displays flagged comments and similar historical comments clearly.
- Allows manual override of classifications.
- Saves confirmed results to the database.
- Provides an intuitive interface for reviewing flagged content efficiently.

US-5: Export Moderation Datasets for Analysis

User Story: As a data analyst, I want the system to provide the functionality to export both pre- and post-moderation datasets, so that I can perform audits and further analyses externally.

Acceptance Criteria:

- Data exports correctly in standard formats (CSV/JSON).
- Moderation logs are included in exports.
- Exported data should preserve all relevant metadata.

US-6: Intuitive UI for Moderation

User Story: As a moderator, I want an intuitive user interface to easily navigate through comments and classifications, ensuring efficient moderation.

Acceptance Criteria:

- The UI should be intuitive and responsive.
- Users should be able to easily navigate through comments and classifications.
- The interface should support filtering, sorting, and searching functionality.

US-7: Access to Historical Moderation Logs

User Story: As a moderator, I want the system to provide historical moderation logs for audit purposes and continuous improvement.

Acceptance Criteria:

- Historical moderation logs should be accessible and clearly displayed.
- An audit trail should be implemented for tracking moderation actions.
- Logs should include timestamps and user information for accountability.

US-8: Role-Based Access Control

User Story: As a system administrator, I want role-based access control for different users, ensuring secure and appropriate access.

Acceptance Criteria:

- Role-based access control is implemented.
- Different roles (Admin, Moderator, Viewer) should be clearly defined and manageable.
- Access levels should restrict actions based on user roles.

US-9: System Monitoring and Alerts

User Story: As a developer, I want system monitoring tools integrated to quickly detect and resolve issues.

Acceptance Criteria:

- Effective monitoring tools are integrated.
- Real-time alerts should be generated for system failures or anomalies.
- Logs should provide clear diagnostics for troubleshooting.

US-10: API Documentation and Schema Definitions

User Story: As a developer, I want clear documentation on system APIs and schema definitions, so I can quickly integrate new modules.

Acceptance Criteria:

- Complete and clear documentation should be provided.
- APIs should be tested and verified for correctness.
- Schema definitions should be version-controlled and well-documented.

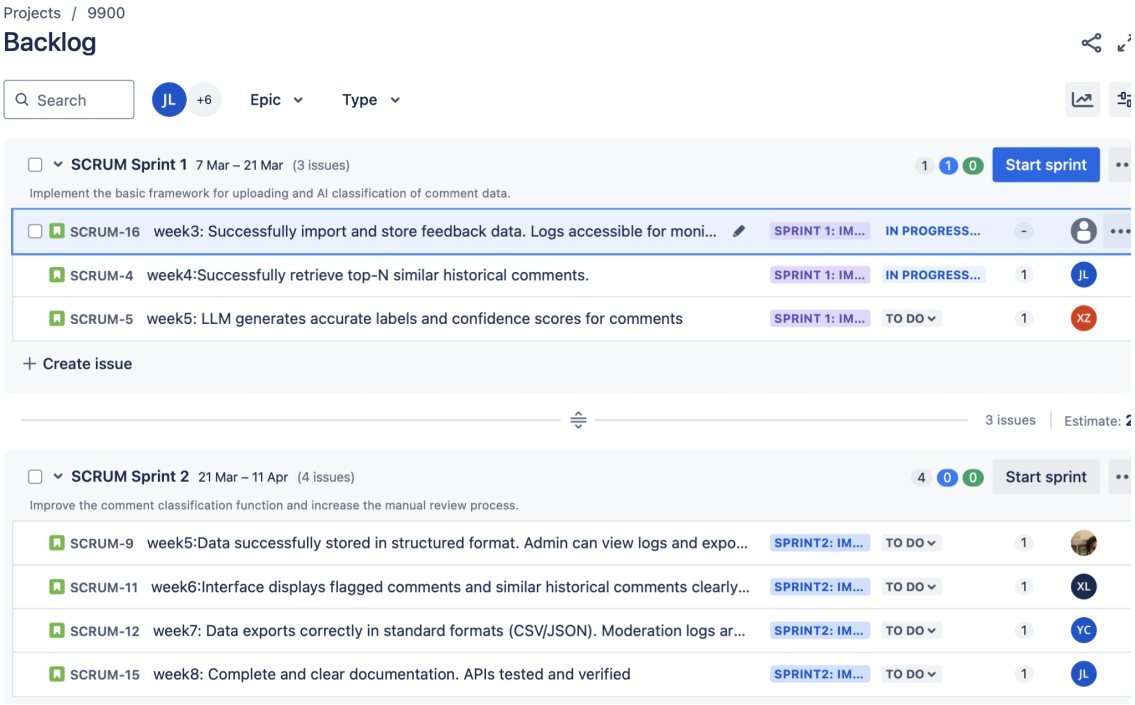


Figure 1: Jira Backlog page 1

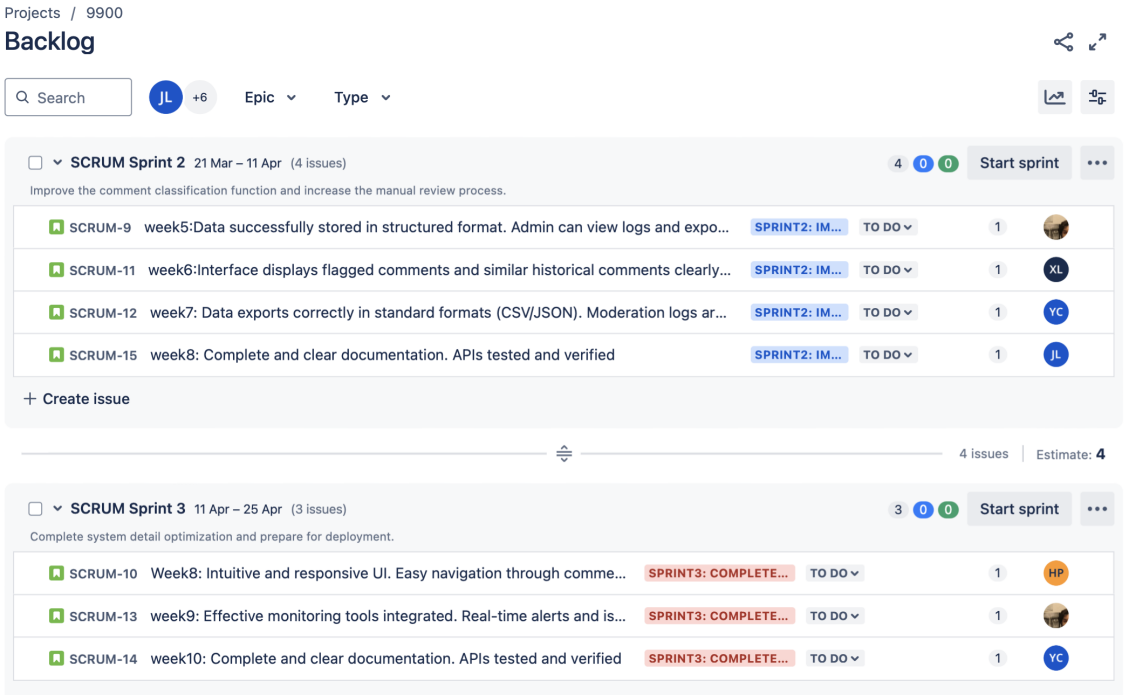


Figure 2: Jira Backlog page 2

Sprint

First Sprint (Week 3-Week 5)

Main goal: Implement the basic framework for uploading and AI classification of comment data.

Week 3: Develop a comment data upload interface built with React and Gradio on the front end. Ensure support for JSON and CSV file formats, and integrate the API built with FastAPI on the back end for efficient data processing. Week 4: Use LangChain and vector databases (such as FAISS) to build an efficient comment data index to ensure fast response to vector retrieval requests. At the same time, integrate RAG and LLM frameworks for preliminary comment classification to automatically mark sensitive content. Week 5: Test the entire data upload and automatic classification workflow and solve any interface call and data flow problems. Collect test results and fix the main problems found.

Screenshots

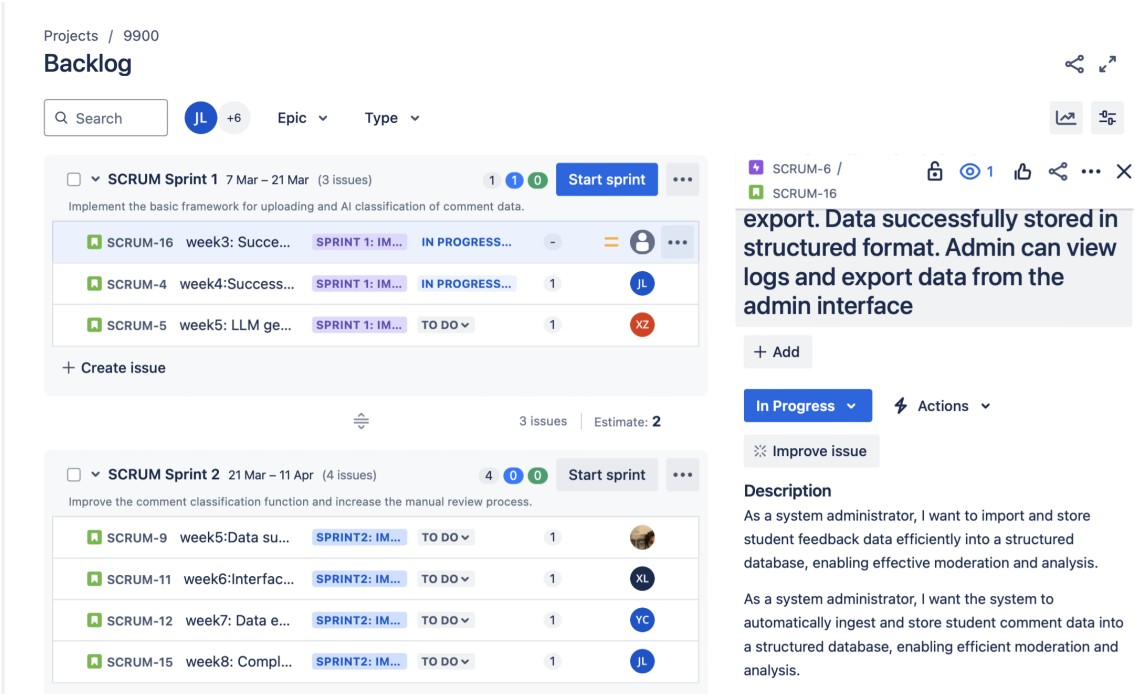


Figure 3: Week 3 user story for first sprint

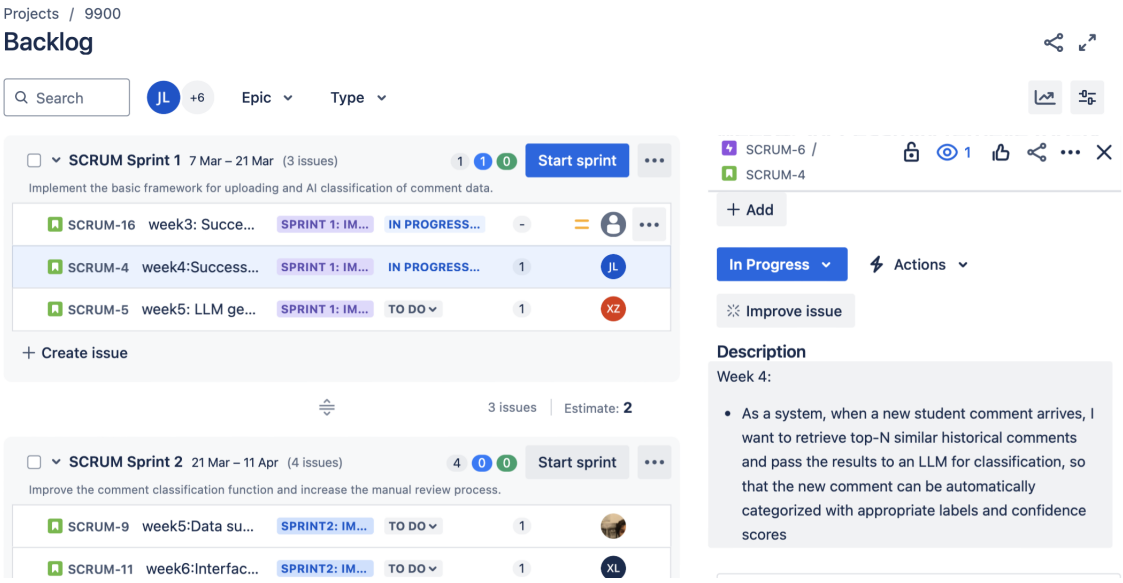


Figure 4: Week 4 user story for first sprint

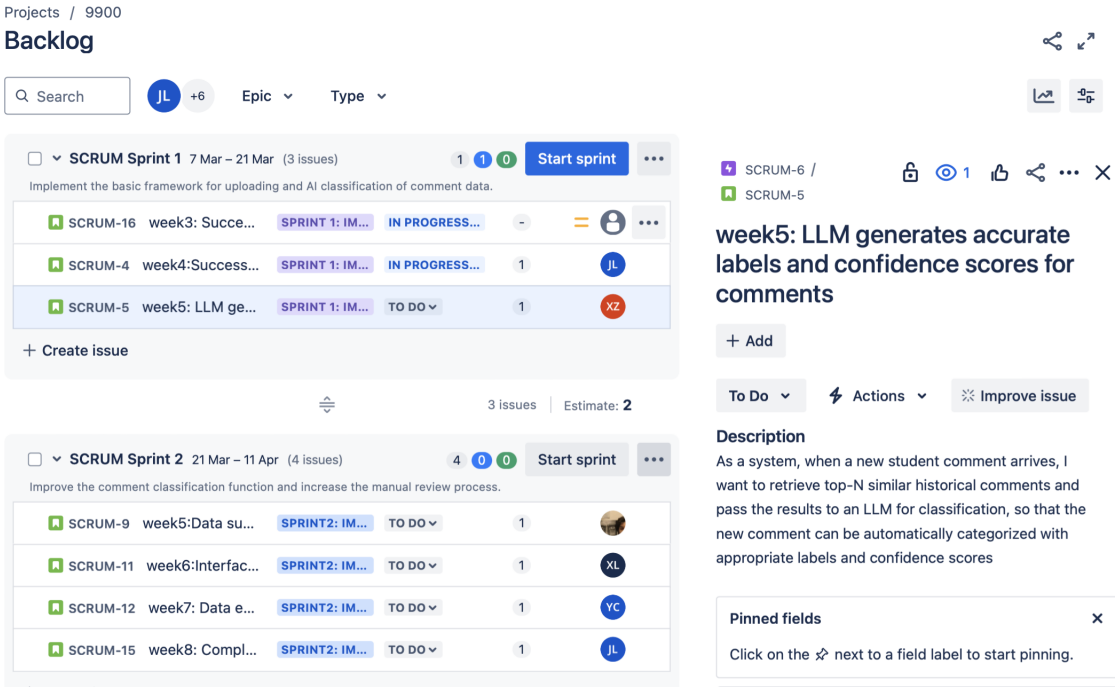


Figure 5: Week 5 user story for first sprint

Second Sprint (Week 5-Week 8)

Main goal: Improve the comment classification function and increase the manual review process.

Week 5 (Continued): Develop and test the file processing interface to display the upload and processing progress. Ensure that the system can correctly read and classify newly uploaded comments. Week 6: Implement a data review option in the system, allowing users to reset imported data, view system-generated classification results, and enter manual review as needed. Week 7: Enhance the manual review interface so that reviewers can see low-confidence comments and make manual adjustments to classification results. Ensure that the interface is intuitive and supports quick review and processing. Week 8: Conduct comprehensive testing of the entire system's front-end to back-end workflow to ensure data accuracy and system stability. Prepare system documentation and operation manuals.

Third Sprint (Week 8-Week 10)

Main goals: Complete system detail optimization and prepare for deployment.

Week 8: Optimize the user interface and interaction design, especially the intuitiveness and operability of the review interface. Ensure that all user stories are supported, such as data import, automatic retrieval classification, and manual review. Week 9: Conduct comprehensive system testing, including security testing and stress testing. Ensure that the system can run stably under high load. Week 10: Prepare for system deployment, conduct final user training and document preparation. Ensure that all functional modules are completed as required by the project and can run stably in the production environment.

Three milestones

Figure 6: Timeline visualization

Technical Design

System Architecture Diagram

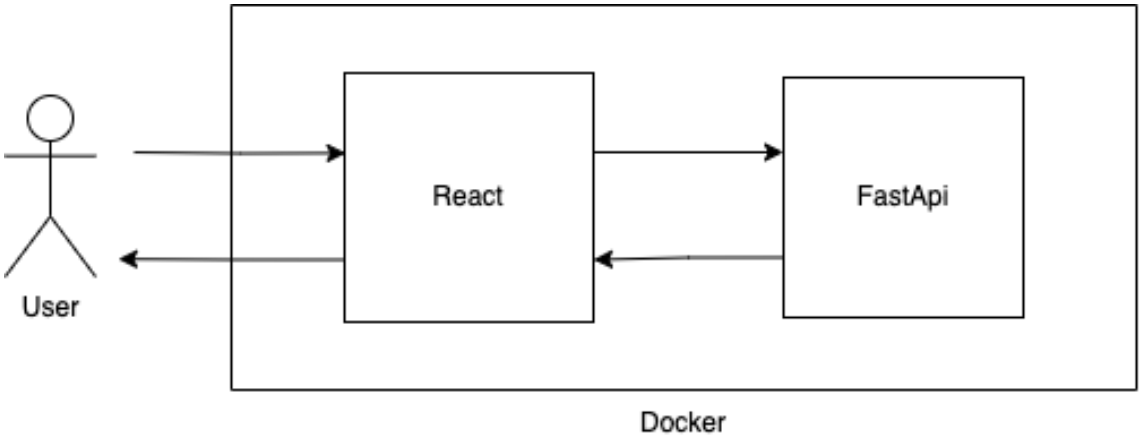


Figure 7: Web architecture diagram.

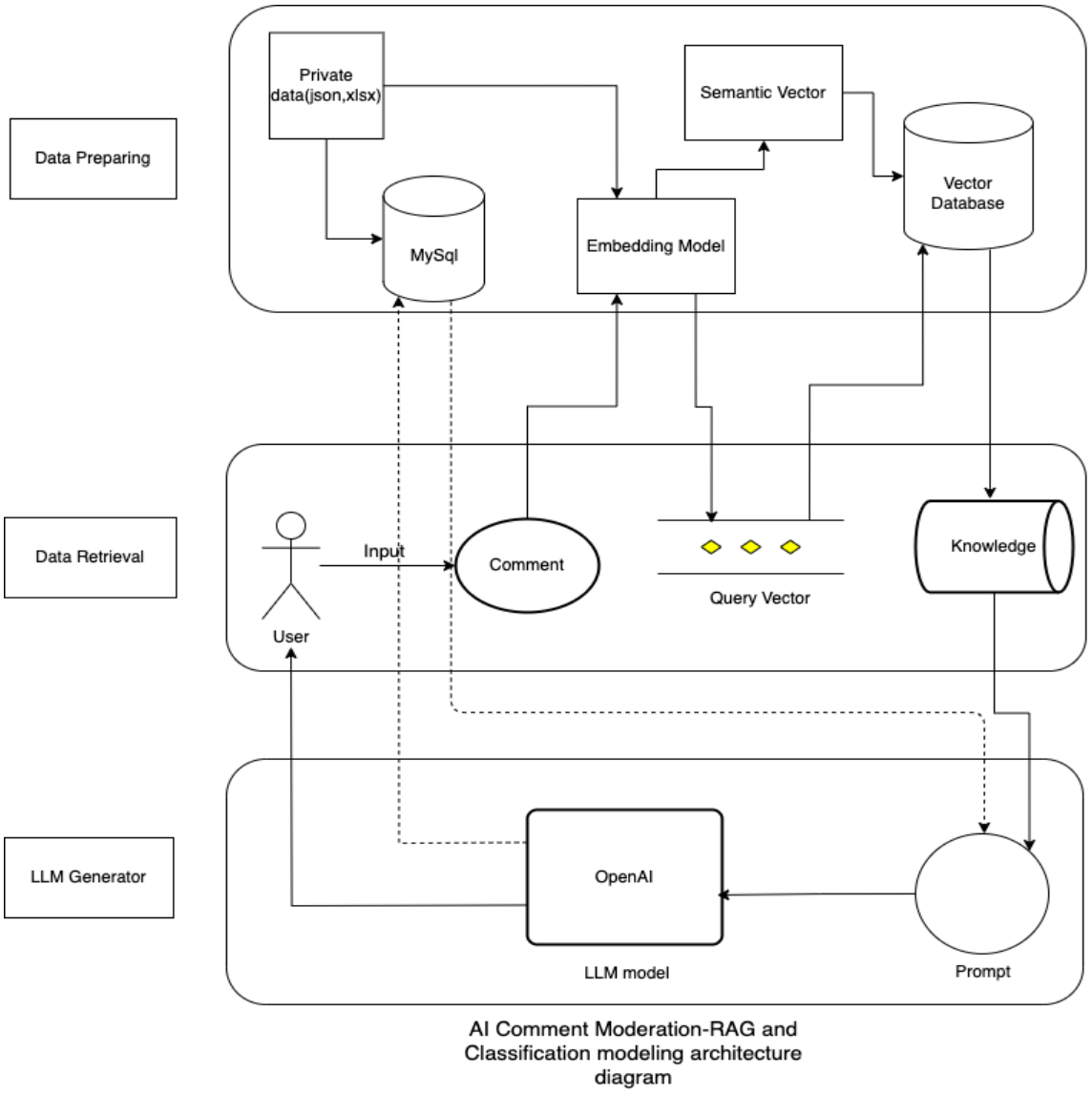


Figure 8: System architecture diagram.

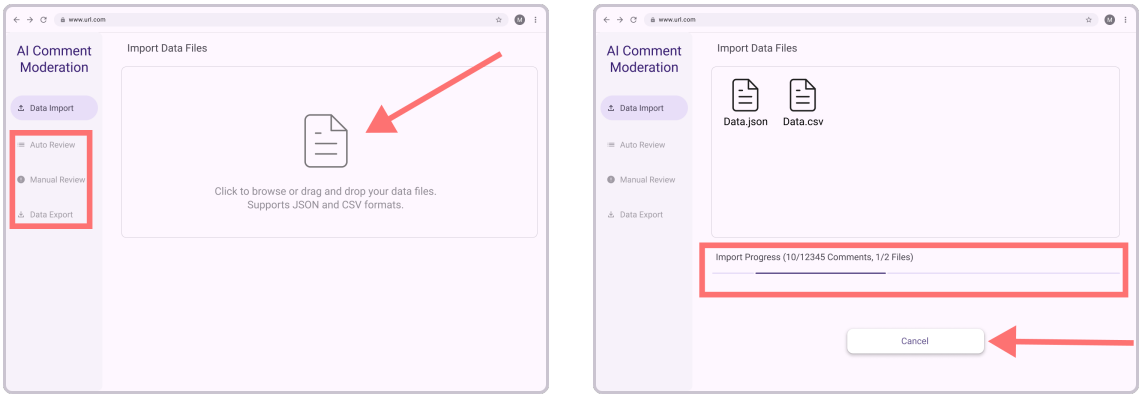
System Storyboard

Figure 9: Overall system UI storyboard showing the AI comment moderation interface with all components and workflow paths.

Figure 9 illustrates the complete UI interaction framework, which is fundamentally built around the user journey. By seamlessly integrating 21 functional interfaces with 24 interaction paths, the system forms a hierarchical workflow that covers the entire data lifecycle—from the initial data import to the final data export and back to the starting page. In addition to the primary task flow, several branch interaction nodes enable users to execute personalized settings and extended operations via intuitive components like hover panels on key pages, creating a composite interaction network with clearly defined primary and secondary paths.

The Storyboard's path planning is meticulously aligned with users' cognitive habits, employing a linear progression structure to ensure smooth and coherent operations. Each interface clearly marks interaction hotspots and provides concise feedback instructions. In the following section, we will analyze the logical connections and state transition mechanisms between each page based on the user's operational sequence.

User Journey



(Step 1) (Step 2)

Figure 10: Initial data import workflow: Step 1 (left) and Step 2 (right) interfaces.

- **Step 1: Comment Data Upload Interface**
  The initial interface for uploading and importing comment data. At this stage, the other three options in the left navigation bar are disabled and displayed with high transparency (as shown by the red box in Figure 10, Step 1). Users can click the "Select Files" button or drag and drop files for multi-file uploads. The system primarily supports JSON and CSV formats (as indicated by the red arrow in Figure 10, Step 1).

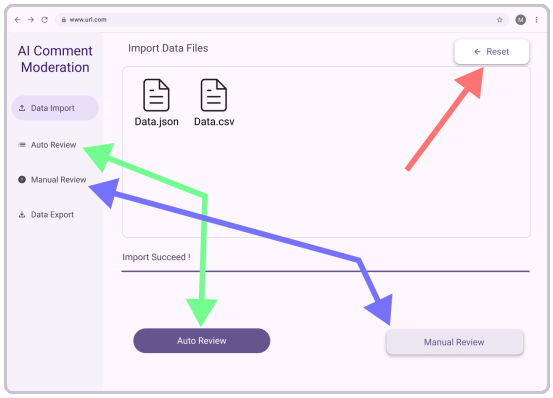
- **Path (1): File Upload Interaction**
  The user uploads files by clicking to select or by dragging and dropping them, which transitions to Step 2.

- **Step 2: File Processing Interface**
  This interface displays the upload progress, showing the number of comments read and categorized, as well as the current file being processed (highlighted by the red box in Figure 10, Step 2). The other navigation options remain disabled.

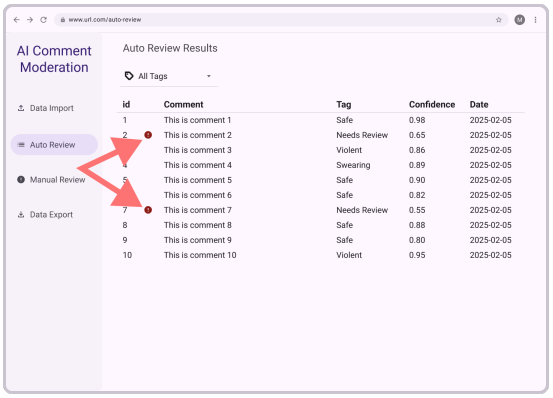
- **Path (2): Upload Cancellation**
  The user clicks the "Cancel" button (indicated by the red arrow in Figure 10, Step 2) to abort the import and recognition tasks and return to the Step 1 interface.

- **Path (3): Automatic Transition to Step 3**
  After comment import and AI classification, the system automatically transitions to the Step 3 interface.

- **Step 3: Data Review Options**
  In this interface, users can choose to reset the imported data, view system-generated classification results, or proceed to manual review for comments with low confidence.



(Step 3)



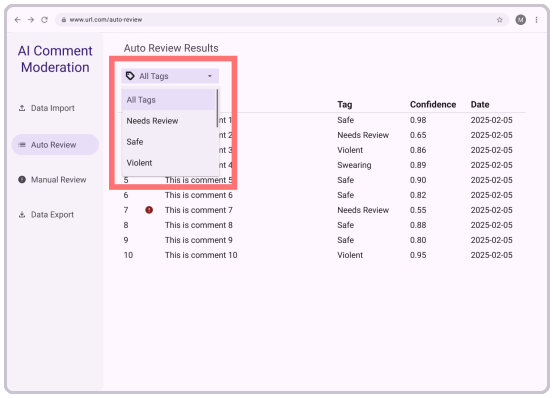
(Step 4)

Figure 11: Data review and classification options following AI processing.

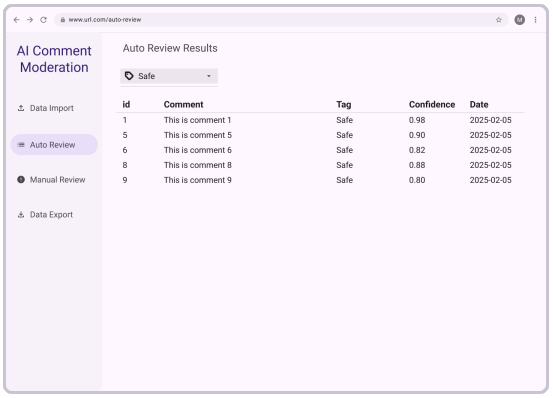
- **Path (4): Reset Data**
  The user clicks the "Reset" button at the top right (as shown by the red arrow in Figure 11, Step 3) to clear the imported data and return to the Step 1 interface.

- **Path (5): Automatic Review Transition**
  Alternatively, the user clicks the "Auto Review" button on the left navigation bar or at the bottom of the page (as indicated by the green double arrow in Figure 11, Step 3) to enter the Step 4 interface.

- **Step 4: Comment List Display**
  This interface displays a list of all imported comments along with their IDs, brief content, AI-assigned classification tags, confidence levels, and comment dates. For comments with confidence below a preset threshold, an error icon appears beside the ID (as shown by the double arrow in Figure 11, Step 4).



(Step 5)



(Step 6)

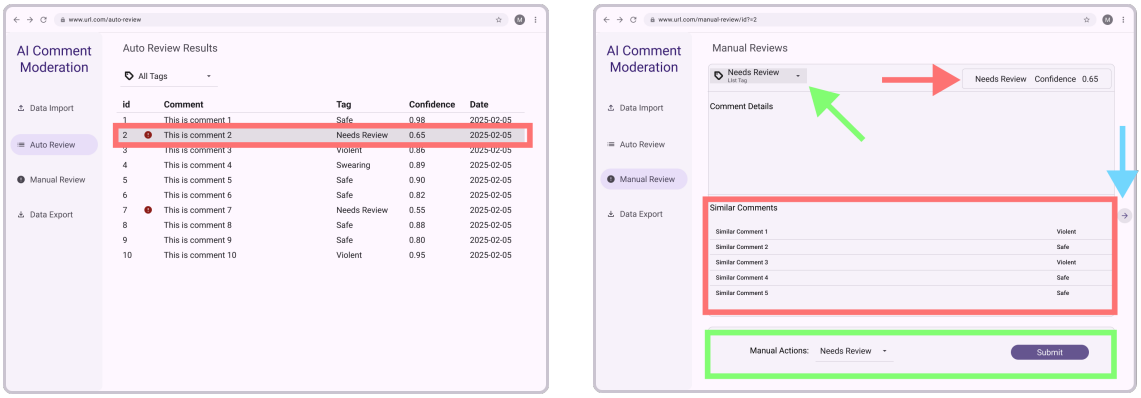
Figure 12: Tag filtering process for comments.

- **Path (6): Initiating Tag Selection**
  From the Step 4 interface, the user clicks the selector at the top left of the list to move to the Step 5 interface.

- **Step 5: Tag Selection for Filtering**
  On this page, the user selects the desired tag using the selector to filter the comments (as highlighted by the red box in Figure 12, Step 5).

- **Path (7): Tag-Based Filtering Transition**
  The user clicks on a specific tag from the list, prompting the system to transition to the Step 6 interface where only comments with the selected tag are displayed.

- **Step 6: Filtered Comment Display**
  This interface shows only the comments that match the selected tag.



(Step 7) (Step 8)

Figure 13: Detailed comment review and manual review entry.

- **Path (8): Comment Selection**
  The user clicks a specific comment from the list in Step 4 to enter the Step 7 interface.

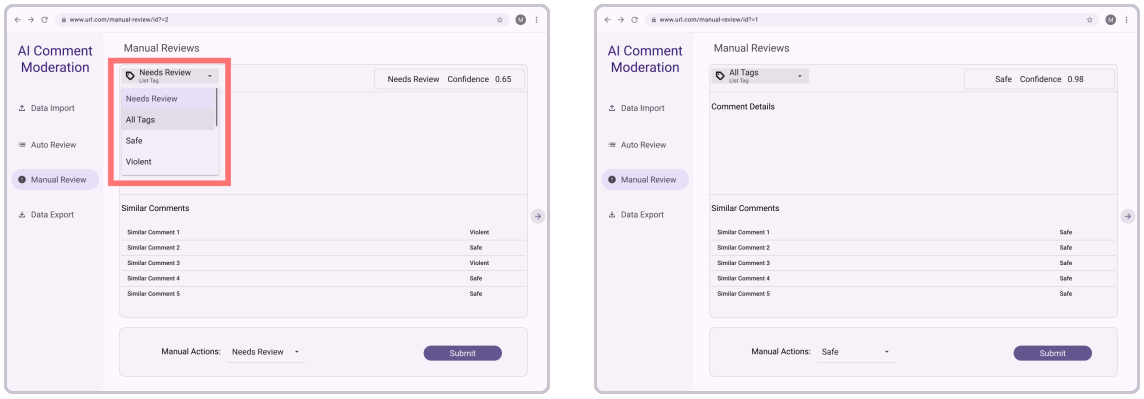
- **Step 7: Comment Selection for Detailed View**
  On the Step 7 page, the user selects the comment for which detailed information or manual review is desired (as indicated by the red box in Figure 13, Step 7).

- **Path (9): Manual Review via Double-Click**
  The user double-clicks the selected comment to enter the manual review interface, which also displays other comments sharing the same tag (as shown by the green arrow in Figure 13, Step 8).

- **Path (10): Direct Manual Review Access**
  Alternatively, the user may click the "Manual Review" button on the Step 3 interface (as indicated by the blue double arrow in Figure 13, Step 3) to access the manual review interface.

- **Step 8: Comprehensive Comment Details**
  In this interface, complete comment details are displayed. The upper left shows the system's classification and confidence (as indicated by the red arrow), the area below lists 3 to 5 similar comments with their classifications (highlighted

by the red box), and the bottom provides a submission area for manual review decisions (marked by the green box). Additionally, navigation arrows allow sequential browsing of comments within the same tag group (with only the forward arrow visible when at the smallest ID, as indicated by the blue arrow).

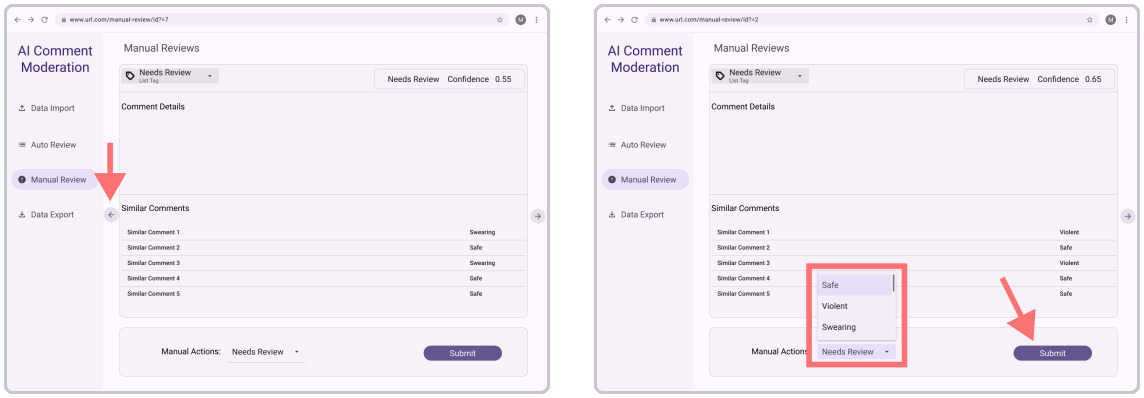


(Step 9) (Step 10)

Figure 14: Tag selection within the manual review interface.

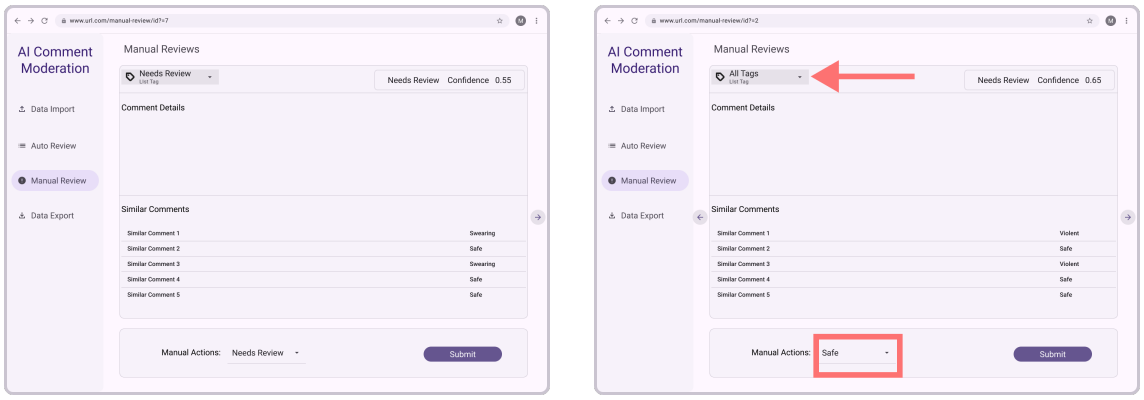
- **Path (11): Transition to Tag Selection**
  Within the manual review interface (Step 8), the user clicks the selector at the upper left of the comment details to enter the Step 9 interface.
- **Step 9: Tag Choice for Manual Review**
  The user views and selects the comment tag they wish to review on the Step 9 page (as highlighted by the red box in Figure 14, Step 9).
- **Path (12): Navigation via Tag Selection**
  After selecting a tag, the user clicks it to transition to the Step 10 interface.
- **Step 10: Detailed Tag-Based Comment Review**
  This interface displays the list of comments associated with the chosen tag, allowing sequential review of detailed comment information and tag verification.

- **Path (13): Navigate to the Next Comment**
  The user clicks the "Next" button within the manual review interface to move to the Step 11 page.
- **Step 11: Display of the Next Comment**
  The interface displays the next comment in the current tag group for manual review.
- **Path (14): Edit Tag Decision**
  The user clicks the tag selector located in the manual review area at the bottom (as shown by the red box in Figure 15, Step 12) to enter the Step 12 interface.



(Step 11) (Step 12)

Figure 15: Sequential comment navigation during manual review.



(Step 13) (Step 14)

Figure 16: Submission and post-submission review process.

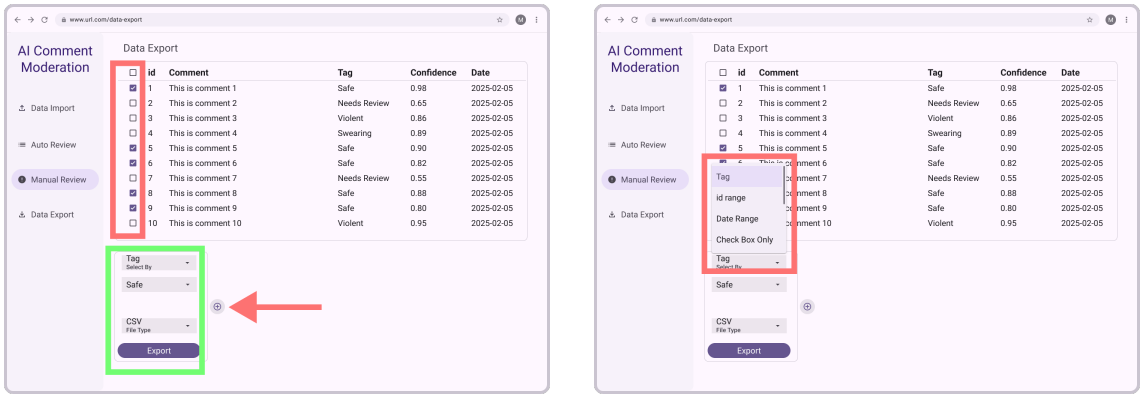
- **Step 12: Submission of Tag Decision**
  In this interface, after selecting the desired tag, the user clicks the "Submit" button on the right to save the decision (as indicated by the red arrow in Figure 15, Step 12).

- **Path (15): Tag Decision Submission**
  After selecting the classification tag, the user clicks the "Submit" button to transition to the Step 13 interface.

- **Step 13: Comment Update After Submission**
  If the submitted tag differs from the original, the comment is removed from the current tag list and the next comment is automatically displayed.

- **Path (16): Transition for Sole Comment**
  If the current tag list contains only a single comment, clicking "Submit" causes the system to transition to the Step 14 interface.

- **Step 14: Confirmation and Default Tag Setting**
  In this interface, the comment remains displayed while the selected tag is saved as the default option (as highlighted by the red box in Figure 16, Step 14), and the tag filter resets to "All Tags" (as indicated by the red arrow).



(Step 15) (Step 16)

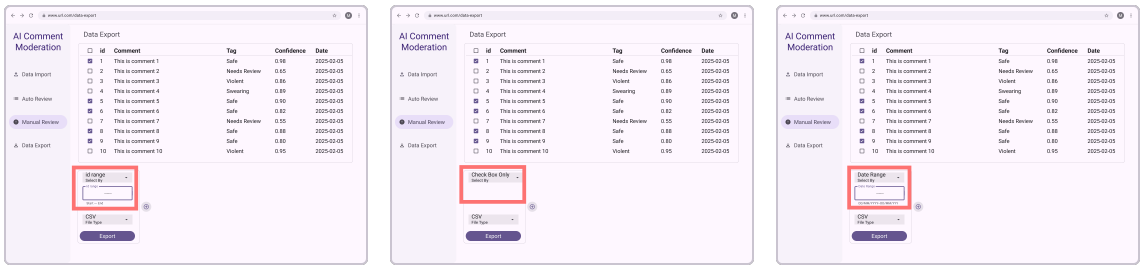
Figure 17: Data export initiation and filter configuration.

- **Path (17): Initiate Data Export**
  The user clicks the "Data Export" button in the left navigation bar, transitioning to the Step 15 interface.

- **Step 15: Data Export and Filtering Options**
  In this interface, users can filter comments using checkboxes preceding the list (as shown by the red box in Figure 17, Step 15) or use the filter panel at the bottom (as indicated by the green box). The plus button adjacent to the panel allows the addition of more filtering conditions (as marked by the red arrow).

- **Path (18): Filter Panel Interaction**
  The user clicks the condition selector in the filter panel to transition to the Step 16 interface.

- **Step 16: Selection of Filtering Conditions**
  In this interface, users may choose a filtering condition based on tag, ID range, date range, or solely via the checkboxes in the list (as highlighted by the red box).

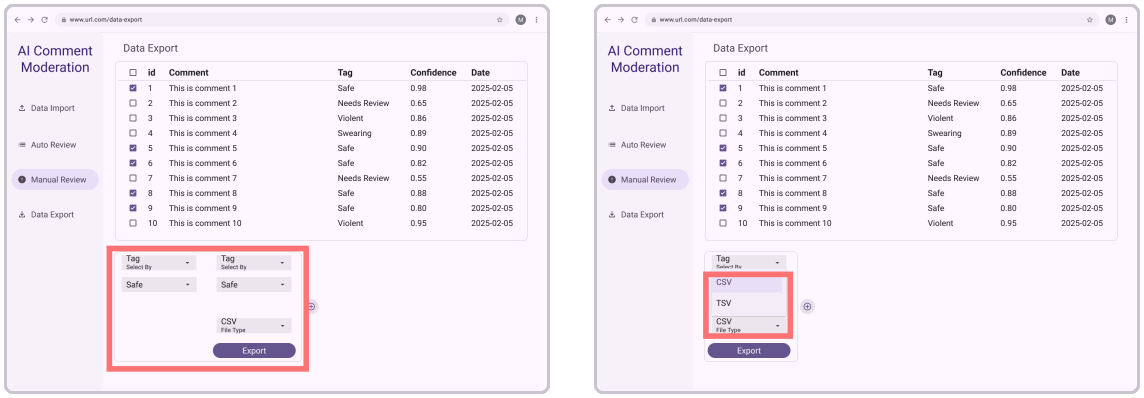


(Step 17) (Step 18) (Step 19)

Figure 18: Filtering criteria selection interfaces.

- **Path (19): ID Range Filtering**
  The user selects "ID Range" as the filtering criterion from the panel, transitioning to the Step 17 interface.

- **Step 17: ID Range Input**
  In this interface, users can enter the starting and ending IDs to filter comments within a specific range (as highlighted by the red box in Figure 18, Step 17).
- **Path (20): Checkbox-Based Filtering**
  The user opts for "Checkbox Only" as the filtering criterion, transitioning to the Step 18 interface.
- **Step 18: Checkbox Filtering**
  In this interface, comments can be filtered solely by selecting checkboxes next to each comment (as indicated by the red box in Figure 18, Step 18).
- **Path (21): Date Range Filtering**
  The user selects "Date Range" as the filtering criterion, transitioning to the Step 19 interface.
- **Step 19: Date Input for Filtering**
  In this interface, users can enter the start and end dates for comment creation to filter comments within the specified date range (as indicated by the red box in Figure 18, Step 19).



(Step 20) (Step 21)

Figure 19: Export configuration and output format selection.

- **Path (22): Adding Additional Filtering Criteria**
  The user clicks the plus button adjacent to the filter panel to add a new filtering condition, transitioning to the Step 20 interface.
- **Step 20: Multi-Criteria Filtering Setup**
  In this interface, users can configure multiple filtering conditions to select the comments for export (as highlighted by the red box in Figure 19, Step 20).
- **Path (23): File Format Selection**
  The user clicks the output file format selector within the panel, transitioning to the Step 21 interface.
- **Step 21: Output File Format Selection**
  In this interface, users can choose the desired output file format (e.g., CSV, TSV, etc.) as indicated by the red box (Figure 19, Step 21).

- **Path (24): Confirm Export and Return**
  Once the export is confirmed, the system automatically returns to the Step 1 interface.

Design details and solution demonstration

This article mainly analyzes the challenges currently faced by manual review of efficient student comments. The fundamental problem lies in the key point of "manual". To solve this core reason, machines need to replace manual work. This article introduces an automatic review solution for comments that replaces traditional manual work based on the two technologies of retrieval-augmented generation (RAG) and large language model (LLM). The following sections are: system architecture design details, the processing mechanism of combining RAG and LLM, possible limitations and corresponding solutions.

1. Layered architecture design

When considering the architectural design, we took into account scalability, flexibility, and development efficiency of team members, and decided to use a layered architecture for construction. At the same time, under the design of a layered architecture, it is easier to combine the vector index database with the cloud LLM for targeted technical optimization.

Front-end and back-end separation architecture

- **Front-end technology combination: React**

First, use React to build a user interface to display a list of comments to be reviewed. There is a "Review" button next to each comment, which triggers the review process when clicked.
Then use Gradio to quickly build an interactive interface for reviewers to view the classification results of comments (such as "positive", "negative", "need manual review"), and reviewers have the highest authority to manually modify the classification results.
In addition, this front-end page uses a single-page application (SPA) architecture in many places (Brunk et al., 2019). This architecture is very effective in reducing server load. For example, when the reviewer browses the comment list on the front-end interface, the page will not refresh frequently, but dynamically load data through AJAX requests. In addition, RESTful API technology is used in the front-end and back-end, which echoes the characteristics of the separation architecture: independence and maintainability during the development process. (Gorwa et al., 2020) A specific example is as follows: When the reviewer clicks "Next Page", the front-end requests new comment data through the RESTful API, and the page only updates the comment list. Such a page is more intuitive and improves the user experience.

- **Backend technology combination: FastAPI+LangChain**

As the name implies, FastAPI is faster than traditional methods and supports asynchronous calls, which makes this technology more handy in high-concurrency scenarios than traditional frameworks. (Kadaskar Kamthe, 2024). In fact, when we first designed the backend, we also considered Flask, but in the end we chose FastAPI because this project will call asynchronously in many places and there are many high-concurrency scenarios. In addition, FastAPI has another advantage: during development, it automatically generates swagger interface test documents to improve development efficiency.

LangChain is a framework designed specifically for LLM (Large Language Model) applications, and is particularly suitable for building complex RAG processes. It supports connecting to a variety of LLMs, databases, and tools, and can easily integrate external knowledge bases and vector retrieval functions. In reality, we simulated a LangChain usage scenario: In the university review system, we used LangChain to build an efficient RAG (Retrieval Augmentation Generation) process. The whole process starts with data loading. First, the comment data is extracted from the MySQL database, and then the comment content is converted into semantic vectors through OpenAI's Embedding API. Then, using LangChain's built-in FAISS vector database, comments similar to the current comment are quickly retrieved, and the components of the comment are further analyzed in combination with external knowledge bases such as the syllabus. In the classification stage, we call the OpenAI GPT model, combine the retrieved similar comments and external knowledge bases, and generate classification results (such as "negative reviews"). For comments with low confidence, the system will mark them as "need manual review" and send them to the reviewer for manual review to ensure the accuracy of the results.

In order to visualize the theory, we introduce a specific process:

Step 1: The reviewer sees a comment on the front-end interface: "The teacher of this course is very boring and the amount of homework is also very large." After clicking the "Review" button, the front-end sends the comment content to the backend through the Restful API.

Step 2: After receiving the comments, the backend first performs vector retrieval through the RAG framework to find similar comments (such as "The teacher's lectures are boring and there are too many homework assignments"), and then calls the OpenAI GPT model for classification to determine whether the comment is a "negative review."

Step 3: The backend returns the classification result (such as "negative review, confidence level 85%)", and the frontend displays the result.

Step 4: If the confidence level is lower than a certain percentage, the review will be automatically classified into the manual review category.

2. RAG and LLM review strategy

Why choose RAG?

The RAG framework has been used in different natural language fields in recent years. The technology has been proven in the market to improve the accuracy of text classification tasks and the interpretability of output results (Lewis et al., 2020)

This technology can combine external knowledge bases to improve the accuracy of decision-making when processing complex text data: Suppose there is a comment: "The exam for this course is too difficult and completely beyond the scope of the course syllabus." Traditional LLM may classify it as a "negative review", but combined with the RAG framework, the system can find similar comments through vector retrieval and make judgments based on external knowledge bases (such as the course syllabus). If the external knowledge base shows that the exam content is indeed beyond the scope of the syllabus, the system can classify the comment as "reasonable criticism" rather than "negative review", thereby improving the accuracy of classification. This is very suitable for our project, so we chose RAG technology and combined vector retrieval and LLM for comment classification.

Specific process

Step 1:Data loading and storage
Import data using JSON/CSV format and store it in MySQL to achieve efficient query and management of structured data

Step 2: Embedding represents calculation
The OpenAI Embedding API is used to generate high-quality semantic vector representations of comment data (Radford et al., 2019). For example, the comment "The lectures of this course are very boring and the amount of homework is also very large." is converted into a 1536-dimensional vector. Data index management and efficient block processing are performed through LangChain and LlamaIndex to improve the efficiency of subsequent retrieval and classification (Lees et al., 2022).

Step 3: Vector Similarity Retrieval
Use FAISS, ChromaDB or Qdrant to implement an efficient and accurate vector retrieval process (Palla et al., 2025). When using this technology: you may find the 5 most similar reviews to the current review. For example, the retrieved similar reviews may include: “The teacher’s lectures are boring and there is too much homework.” “The course content is boring and the homework load is too heavy.”

Step 4: LLM classification judgment
Call the OpenAI GPT model, combine it with the retrieved similar comments,

achieve accurate classification and labeling, and output clear classification labels and confidence information (the GPT model may output: "Negative review, confidence 85%"). (Brown et al., 2020).

Step 5: Manual review process design
For comments with low classification confidence or high sensitivity, a special manual review mechanism is designed to further improve the accuracy and reliability of the review results. That is, if the classification confidence is lower than a certain threshold (such as 85%), the system will mark the comment as "requires manual review" and send it to the reviewer for manual review.

3. Examples of limitations and their optimization strategies

3.1 Data Storage Optimization

Potential problem: In reality, student evaluation in colleges and universities is a massive task, often involving tens of thousands of data points. At this moment, a single MySQL server may face problems when processing unstructured data (such as comment content and its embedding vector): storage bottleneck and slow retrieval

Optimization measures:

Hybrid storage of data: We divide data into two categories: structured and unstructured. We store unstructured data (comment content and its embedding vector) in MongoDB; on the other hand, we keep structured data (such as user information, course information, etc.) in MySQL. In this way, the hybrid storage solution can significantly improve the overall performance of the system and effectively solve storage bottlenecks and slow retrieval.

3.2 LLM cost and delay issues

Example: Reduce LLM call frequency Assuming that the system needs to review 10,000 comments every day, if each comment calls the OpenAI GPT model, there is no doubt that this will make the cost very high. In order to reduce costs, the system can introduce miniGPT to cache regular queries.
For example, for the common comment "The teacher's lecture is very good", miniGPT can directly classify it as a "positive review" without calling the OpenAI GPT model. For high-risk or low-confidence comments (such as "The exam for this course is too difficult"), the system will call the OpenAI GPT model for detailed analysis.

3.3 Vector retrieval accuracy issues

Example: Optimizing vector search parameters If you find that the search results are not accurate enough when using FAISS for vector search, you can optimize the search performance by adjusting the nprobe parameter of FAISS (controlling the search accuracy).
For example, adjusting nprobe from the default value of 1 to 10 can significantly

improve the accuracy of retrieval results, but at the expense of increased computational overhead. We can find a balance point through repeated debugging, which can ensure accuracy and control computing costs. Of course, this will most likely require long-term experimentation and debugging.

4. Technical solution comparison and selection examples

Example: Comparison between cloud service API and local model

Cloud Service API (OpenAI):
Assuming that multilingual comments (such as Chinese, English, and Spanish) need to be reviewed somewhere, the OpenAI API can easily process multilingual data without additional development.
For example, for a Spanish review, "El profesor es muy aburrido" (The teacher is boring), the OpenAI API can automatically identify the language and classify it.

Local Model (LLaMA/BERT):
If data privacy requirements are extremely high, you can choose to deploy the LLaMA model locally. For example, if the review data of a school involves student privacy, local deployment can ensure that the data is not leaked.
However, local deployment requires professional hardware support (such as GPU) and has high maintenance costs.
Finally, after discussion among our team members, we chose the cloud API solution mainly for the following reasons: First, the project time was tight, and cloud services were deployed very quickly, allowing us to quickly test and iterate. Second, cloud services support data processing in multiple languages and multiple scenarios, which is just right for the complexity and diversity of university review data. In addition, using cloud services can greatly reduce the system maintenance cost and the complexity of model management, allowing us to focus more on the development of core functions.

Mapping between features and user stories

This section mainly explains the system functional modules and their corresponding user stories, visualizes the user story content, and ensures that the content of each user story meets the requirements of the demander.

Functional module 1: Data import and vector indexing

User Story ID: US-1

1. The data storage and index building
   system supports efficient storage of existing or newly added comment data in the database, and generates vector indexes at this time, which greatly improves the efficiency of similarity retrieval in the subsequent links.

2. Multi-format data import
   In order to meet the data loading requirements in different scenarios, the system is compatible with multiple data formats (such as JSON, CSV), which is more universal in actual situations.

3. Semantic vector generation and index optimization
   By calling the OpenAI Embedding API, the system converts the comment content into a high-dimensional semantic vector and stores the vector in a vector database. It uses the vector database (such as FAISS) to build an efficient index structure.

User story example:

Role : System Administrator

Requirements :
As a system administrator, I need to import students' comment data into the system and generate vector indexes for subsequent automatic classification and retrieval.

Scenario :

- An administrator uploads a CSV file containing 10,000 reviews.
- The system automatically parses the file and stores the comments in the MySQL database.
- The system calls the OpenAI Embedding API to generate a semantic vector for each comment and stores it in a vector database (such as FAISS).
- Administrators can view the data import status and vector index generation progress in the system.

Functional module 2: Automatic retrieval and classification (RAG + LLM)

User Story ID: US-2

1. After receiving new comments, the new comment processing and automatic classification system uses vector retrieval technology to quickly locate similar comments and combines it with the large language model (LLM) to achieve automatic classification.

2. RAG framework and external knowledge integration
   Using the Retrieval Augmented Generation (RAG) framework, the system can combine external knowledge bases (such as course outlines, historical reviews, etc.) to improve the accuracy of classification results.

3. Classification result generation and confidence assessment
   Call the OpenAI GPT model to conduct in-depth analysis of the comments, generate classification labels (such as "positive review", "negative review"), and output confidence scores.

4. Processing of low-confidence comments
   For comments with low classification confidence or high risk, the system automatically marks them as "requires manual review" and pushes them to the manual review interface.

User story example:

Role : Auditor

Requirements : As a reviewer, I hope that the system can automatically classify new comments and mark the comments that need manual review to improve review efficiency.

Scenario :

- A new review: "The lectures of this course are very boring and the amount of homework is very large." was submitted to the system.
- The system finds similar comments (such as "The teacher's lectures are boring and there are too many homework assignments") through vector retrieval.
- Combined with external knowledge bases (such as course syllabus), the system calls the OpenAI GPT model to generate classification results: "negative review, confidence level 85%".
- If the confidence level is lower than 85%, the system marks the review as "needs manual review".

Functional module 3: Manual review interface (low confidence/high risk)

User Story ID: US-3

1. Interface review function for low-confidence or high-risk comments, which is convenient for auditors to quickly view and process.
2. Multi-dimensional information display and manual adjustment
   Reviewers can view similar comments, automatic classification results and confidence scores in the interface, and manually adjust the classification results according to actual conditions.
3. The audit result preservation and traceability
   The system supports the preservation of the final audit results and records the complete audit process, providing a reliable basis for subsequent analysis and auditing.

User story example:

Role : Auditor

Requirements : As a reviewer, I want to be able to view the details of low-confidence comments through the interface and manually modify the classification results to ensure the accuracy of the review.

Scenario :

- After the reviewer logs into the system, he sees a list of comments awaiting review.
- Click on a review: "The exam for this course is too difficult and completely beyond the scope of the course syllabus."
- The system displays detailed information about the review, including similar reviews, classification results (such as "negative review, confidence level 65%"), and relevant content from external knowledge bases (such as course syllabus).
- The reviewer determines whether the comment is reasonable based on the information and manually modifies the classification result to "reasonable criticism".

Functional module 4: Version management and data export

User Story ID: US-4

1. Versioning management and historical backtracking
   The system performs versioning management on the final results of automatic classification and manual review, and supports viewing and backtracking of historical records.
2. Data export and archiving support
   The system provides flexible data export functions and supports exporting audit results in common formats such as CSV/TSV, which facilitates subsequent data analysis, archiving and sharing.

User story example:

Role : System Administrator

Requirements : As a system administrator, I want to be able to perform version management on the audit results and support data export for subsequent analysis and archiving.

Scenario :

- The administrator views the result of a certain audit task in the system.
- The system displays the version history of the task, including automatic classification results and manually reviewed modification records.

- When the administrator chooses to export the review results of the task, the system generates a CSV file containing information such as comment content, category label, confidence level, and reviewer.
- The administrator downloads the CSV file to the local computer for subsequent data analysis and archiving.

Functional module 5: Audit statistics and visualization

User Story ID: US-5

1. Provides dashboard function to display statistical information such as total audit volume, classification distribution, and confidence distribution.
2. Supports visual charts (such as bar charts and pie charts) to display audit results, helping administrators quickly understand the audit status.
3. Provides filtering function and supports viewing statistical data by dimensions such as time, course, category label, etc.

User story example:

Role : Administrator

Requirements : As an administrator, I want to be able to view the statistical information of the audit data through the dashboard and quickly understand the audit status through visual charts.

Scenario :

- After the administrator logs in to the system, he/she enters the audit statistics dashboard.
- The dashboard displays the following information:
- Total number of reviews: 10,000 comments.
- Category distribution: 60% positive reviews, 30% negative reviews, and 10% require manual review.
- Confidence distribution: high confidence (>80%) 70%, medium confidence (50%-80%) 20%, low confidence (<50%) 10%.
- Administrators can use the filtering function to view statistical data by time, course, category label, and other dimensions.
- The system supports exporting statistical results as charts (such as bar charts, pie charts) to facilitate report generation.

References

Bagai, R. (2024), 'Comparative analysis of aws model deployment services', *arXiv preprint arXiv:2405.08175* .

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020), 'Language models are few-shot learners', *Advances in neural information processing systems* **33**, 1877–1901.

Brunk, J., Mattern, J. & Riehle, D. M. (2019), Effect of transparency and trust on acceptance of automatic online comment moderation systems, *in* '2019 IEEE 21st Conference on Business Informatics (CBI)', Vol. 1, IEEE, pp. 429–435.

Chen, X., Vorvoreanu, M. & Madhavan, K. (2014), 'Mining social media data for understanding students' learning experiences', *IEEE Transactions on learning technologies* **7**(3), 246–259.

Ehsan, A., Abuhaliqa, M. A. M., Catal, C. & Mishra, D. (2022), 'Restful api testing methodologies: Rationale, challenges, and solution directions', *Applied Sciences* **12**(9), 4369.

Gorwa, R., Binns, R. & Katzenbach, C. (2020), 'Algorithmic content moderation: Technical and political challenges in the automation of platform governance', *Big Data & Society* **7**(1), 2053951719897945.

Hosseini, H., Kannan, S., Zhang, B. & Poovendran, R. (2017), 'Deceiving google's perspective api built for detecting toxic comments', *arXiv preprint arXiv:1702.08138* .

Ivan, S. C., Győrödi, R. Ş. & Győrödi, C. A. (2024), 'Sentiment analysis using amazon web services and microsoft azure', *Big Data and Cognitive Computing* **8**(12), 166.

Kadaskar, H. R. & Kamthe, V. R. (2024), 'An overview of aws', *International Journal of Scientific Research in Modern Science and Technology* **3**(7), 22–30.

Lees, A., Tran, V. Q., Tay, Y., Sorensen, J., Gupta, J., Metzler, D. & Vasserman, L. (2022), A new generation of perspective api: Efficient multilingual character-level transformers, *in* 'Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining', pp. 3197–3207.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T. et al. (2020), 'Retrieval-augmented generation for knowledge-intensive nlp tasks', *Advances in neural information processing systems* **33**, 9459–9474.

Mihaljević, H. & Steffen, E. (2023), 'How toxic is antisemitism? potentials and limitations of automated toxicity scoring for antisemitic online content', *arXiv preprint arXiv:2310.04465* .

Nogara, G., Pierri, F., Cresci, S., Luceri, L., Törnberg, P. & Giordano, S. (2023), 'Toxic bias: Perspective api misreads german as more toxic', *arXiv preprint arXiv:2312.12651* .

Palla, K., García, J. L. R., Hauff, C., Fabbri, F., Lindström, H., Taber, D. R., Damianou, A. & Lalmas, M. (2025), 'Policy-as-prompt: Rethinking content moderation in the age of large language models', *arXiv preprint arXiv:2502.18695* .

Pozzobon, L., Ermis, B., Lewis, P. & Hooker, S. (2023), 'On the challenges of using black-box apis for toxicity evaluation in research', *arXiv preprint arXiv:2304.12397* .

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. et al. (2019), 'Language models are unsupervised multitask learners', *OpenAI blog* **1**(8), 9.

Riehle, D. M., Niemann, M., Brunk, J., Assenmacher, D., Trautmann, H. & Becker, J. (2020), Building an integrated comment moderation system–towards a semi-automatic moderation tool, *in* 'International Conference on Human-Computer Interaction', Springer, pp. 71–86.

Wen, M., Yang, D. & Rose, C. (2014), Sentiment analysis in mooc discussion forums: What does it tell us?, *in* 'Educational data mining 2014', Citeseer.