Perfect Studymate: A Learning Chatbot Powered by RAG Technology

Juyong Rhee[1], Yewon Chun[2], Jihee Hwang[3], Jorge Alcorta[4],

[1] Department of Chinese Language and Literature, SungKyunKwan University qpsvs0131@gmail.com
[2] Department of Statistics, SungKyunKwan University 1000daughther@g.skku.edu
[3] Department of Korean Language and Literature, SungKyunKwan University jihee336723@gmail.com
[4] Department of Software, SungKyunKwan University 2024319370@g.skku.edu

Abstract. Chatbots often confuse users by producing false responses (hallucinations). This is also a reason why users are reluctant to use chatbots for learning. In this report, we propose a RAG-based chatbot service named "Perfect Studymate" that allows students to experience fewer hallucinations. The user uploads the '.pdf' formatted resources used in courses to the chatroom, and the model embeds the uploaded data and stores it in a database. Then, when the user asks chatbot a question related to the data, the model extracts the context related to the question from the embedding database and reflects it in the answer. At the same time, an interactive conversation is implemented by utilizing the history of conversation between the user and the chatbot. Users can enjoy an improved learning experience by talking to chatbots with less hallucination based on the content of the course material they want.

Keywords: LangChain · RAG · LLM · Chatbot

1 Introduction

1.1 Motivation

There will be no student who has not used ChatGPT for learning. At the same time, there will be no student who has not experienced a low-quality answer generated from ChatGPT. Therefore, many students use chatbots for learning, but they do not trust the chatbot's response much. Furthermore, even if the chatbot generates a response consistent with ground truths, it may differ from the user's expectations in terms of terminology. These limitations have been pointed out as reasons why chatbots cannot be fully used for learning.

1.2 Proposal and Goals

The RAG (retrieval augmented generation) technology is designed to solve the hallucination problem of LLM model. The LLM model to which RAG technology is applied refers to highly relevant content from a database outside the LLM learning data source before generating a response. This technology allows chatbots to generate more accurate and highly relevant responses to questions.

We planned a custom chatbot service for learning that allows users to utilize the RAG model by uploading resources they want to learn directly to the chatbot. Through RAG, users can receive

higher quality responses. Additionally, it is designed so that the conversation can proceed in a more interactive flow by storing the history of conversation between the user and the chatbot in a database and using it for subsequent responses.

After that, performance testing was done based on the basic RAG chatbot architecture. The performance of the chatbot, which varies according to the types of various loaders(parsers) and llm models, was compared through objective numeric metrics. Finally, we tried to maximize the performance of the chatbot by applying the combination of the best performing components to the basic architecture.

1.3 Approach

All of our team members are in charge of front-end, back-end, and modeling. We participated in the development flow in parallel by using git for efficiency. There are a total of two people in charge of modeling, each of which is in charge of model architecture development, and model improvement.

2 Design

2.1 Overall Architecture

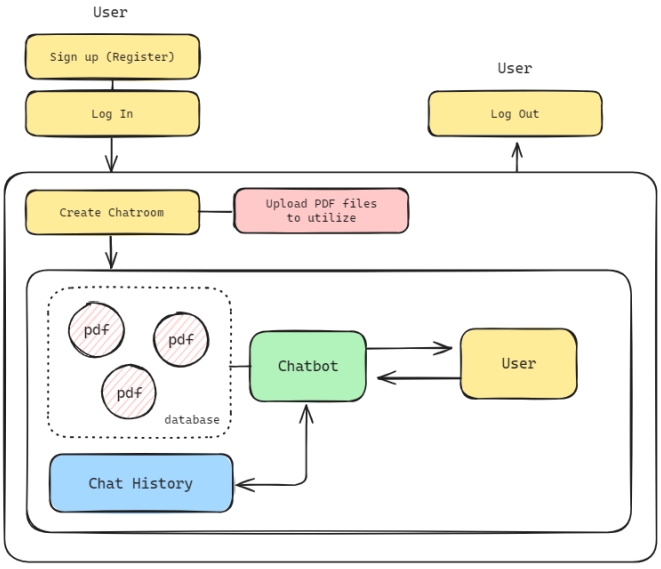


Fig. 1: Overall service architecture

The structure of the service basically used the structure of ChatGPT[5]. Users who create account can log in with their own accounts to access one of several chatrooms, or create a new

[5] https://chatgpt.com/

chatroom. When creating a chatroom, a user may upload pdf files to be used for a chat. In this case, the user can upload multiple pdf files as much as he/she wants. These uploaded pdf files are immediately embedded and stored in Chroma vector database. After then, user can talk to a chatbot that retrieves context from database relating to questions and generates a reponse based on retreived contexts. Also, the model stores each conversation history in a database and utilizes them dynamically for subsequent responses.

2.2 Backend Design

Database Design The database for **Perfect Studymate** is designed to support the core functionalities of the application: user management, chatbot interaction, and document handling within unique chatting sessions. The design focuses on organizing data in a way that ensures logical separation, data integrity, and efficient query performance.

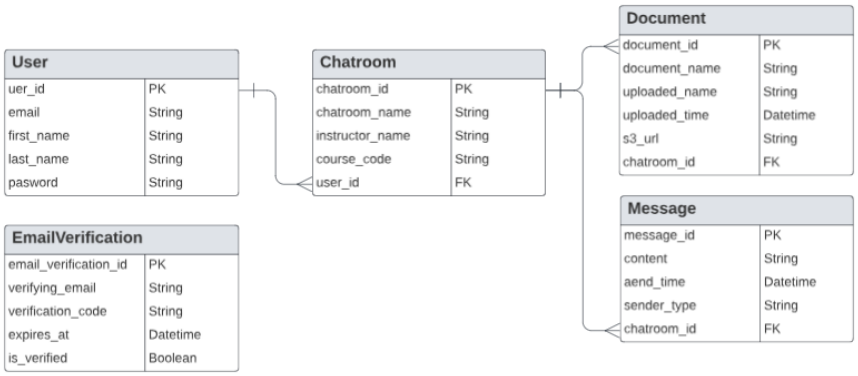


Fig. 2: Database Design

Main Objects

User

- **Purpose**: Stores information about the application's registered users.
- **Attributes**:
  - user_id: Primary key to uniquely identify a user.
  - email: Unique email address used for authentication and communication.
  - first_name: User's first name.
  - last_name: User's last name.
  - password: Encrypted password for secure user authentication.

Chatroom

- **Purpose**: Represents a single unique session where the user interacts with the chatbot.
- **Attributes**:
  - chatroom_id: Primary key to uniquely identify a chatroom.
  - chatroom_name: User-defined name for the chatroom to distinguish sessions.
  - instructor_name: Optional name of the instructor associated with the course (if applicable).
  - course_code: Optional course code associated with the chatroom (if applicable).
  - user_id: Foreign key linking the chatroom to its owner in the User table.

Document

- **Purpose**: Represents documents (pdf files) uploaded to a specific chatroom.
- **Attributes**:
  - document_id: Primary key to uniquely identify a document.
  - document_name: Original name of the file uploaded by the user.
  - uploaded_name: Unique name assigned to the file in storage, used as the retrieval key.
  - uploaded_time: Timestamp indicating when the file was uploaded.
  - chatroom_id: Foreign key linking the document to its associated chatroom.

Message

- **Purpose**: Represents messages exchanged within a chatroom, including user and chatbot-generated messages.
- **Attributes**:
  - message_id: Primary key to uniquely identify a message.
  - content: Text content of the message.
  - send_time: Timestamp indicating when the message was sent.
  - sender_type: Indicates whether the message was sent by the user or generated by the chatbot.
  - chatroom_id: Foreign key linking the message to its associated chatroom

Relationships

1. **User & Chatroom**
   - **Relationship**: One-to-Many
     - A user should be able to create and manage multiple unique chat sessions with the chatbot.
2. **Chatroom & Document**
   - **Relationship**: One-to-Many
     - Documents are bound to specific chatrooms, as the discussion context is defined by the chatroom where the document is uploaded.
3. **Chatroom & Message**
   - **Relationship**: One-to-Many
     - Messages, whether sent by the user or generated by the chatbot, are always tied to a specific chat session.

Design Characteristics The main point of this design is to **focus on chatroom as a core unit**. Documents and messages are designed to be bound to the chatroom rather than the user because all logic involving documents and messages operates within the context of a specific chat session. Users are inherently linked to documents and messages through their ownership of the chatroom.

Integrity and simplified access control Tying documents and messages to chatrooms and ensuring each chatroom is owned by a user. This design simplifies access control and enforces ownership logic.

2.3 Challenges

Model Performance Evaluation The biggest problem with chatbot model evaluation is the absence of well-known and agreed evaluation metrics. Although there are metrics for evaluating partial performance of model, there are virtually no metrics for expressing the entire model performance as a single numerical value.

At the same time, the performance of the model may vary depending on the type of data loader(parser) and llm model constituting the model architecture, but there is no objective criterion for what kind of loader(parser) and llm model should be used.

To address these two limitations; absence of entire model performance metrics and absence of criteria for determining model configuration components, we decided to plan the model performance evaluation process and use the results of the evaluation to determine the final model.

As candidates of model components, we chose 3 pdf loaders (parsers) and 5 llm models. That is, a total of 15 candidate models were created and evaluated. And each candidate model generated responses to a total of 45 questions, 15 each from 3 sample documents. The 45 responses generated at this time were evaluated based on several metrics. In the evaluation, 'answer relevancy', 'context precision', 'faithfulness', 'context recall', and 'answer correctness' were used as evaluation indicators. Each of these five evaluation metrics is a numerical representation of partial performances of the model. Therefore, we thought that a comprehensive consideration of these five indicators could help measure the overall performance of the model. The values of the five metrics were combined to evaluate the model performance with one consistent number. You can refer to Appendix B for knowing structure of model evaluation process in detail.

The measurement results of the model performance are as Figure 3. This is the distribution of the values obtained by evaluating the responses to the questions for each model based on five evaluation metrics and then adding those five values. In other words, for each model, 45 scores corresponding to 45 questions were calculated. Roughly confirmed, there is no noticeable difference in the performance evaluation results of the 15 models. Before making intuitive judgments, various statistical tests were applied to test the evaluation results of the model.

In turn, *Shapiro Wilk test*, *Kruskal-Wallis Rank Sum test*, and *Levene test* were performed on the 15 distributions. Please refer to Appendix C for more specific test flow. Each of these tests shows whether distributions have normality, the same median, and the same variance. As a result of the test, all 15 distributions did not have normality and were found to have the same median and variance. To summarize, the results of the evaluation performed on 15 models were statistically

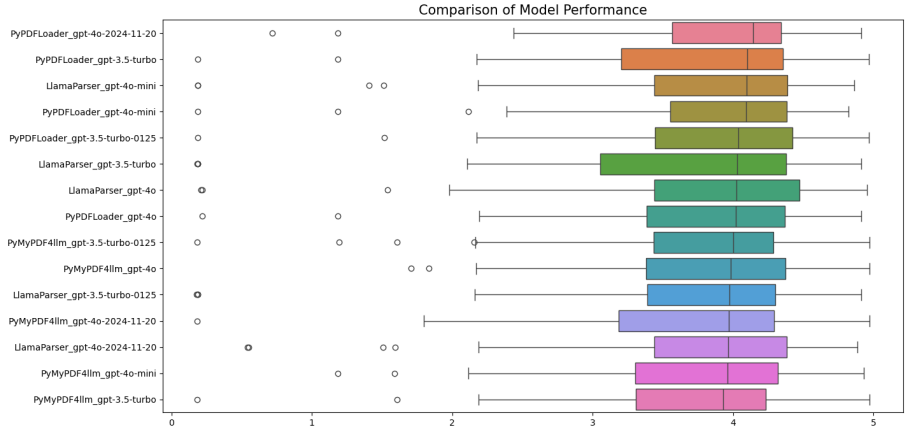


Fig. 3: Comparison of model performances

similar across 15 models, and the final model had to be chosen by intuition rather than by statistical method.

Therefore, combination of **"PyPDFLoader"** and **"gpt-4o-2024-11-20"** with the smallest variance and highest median of scores is selected as a final model. Above all, the smallest variance means that the chatbot's response performance is stable, so it was judged to be the most appropriate chatbot for learning. Figure in Appendix D is the visualization of the performance of the final model under five metrics. All five metrics show that the final model has generally stable high performance.

Prompt Engineering Despite minimizing Hallucination of chatbots by using RAG, chatbots were still likely to give false answers. To address this issue, we designed a new prompt that asks for faithful answers based on the context extracted from the database.

API Scalability To deal with potential for high traffic, We used Flask for simple REST interface, with room to scale via deployment strategies like Docker and load balancing.

3 Tools and Frameworks

3.1 UI/UX

Figma Figma is a web-based design tool for UI/UX that allows teams to design, prototype, and collaborate in real-time. It supports vector editing and prototyping, making it ideal for creating responsive and interactive designs. Its cloud-based nature enables seamless collaboration between designers and developers, allowing for faster prototyping and design validation with real-time updates.

Shadcn/UI Shadcn/UI is a library for building customized UI components with Tailwind CSS and Radix UI. It simplifies the process of creating consistent and accessible designs, making it an integral tool for ensuring high-quality user interfaces.

3.2 FrontEnd

React.js React.js is a JavaScript library for building dynamic and responsive web applications using a component-based architecture. Its virtual DOM improves performance, while its extensive ecosystem, including tools like React Router, supports scalable and efficient development.

JSX (JavaScript XML) is a syntax extension for JavaScript that allows developers to write HTML-like code directly within JavaScript. It simplifies UI development by making the code more readable and maintainable, bridging the gap between design and development with a declarative syntax.

3.3 BackEnd

FastAPI Ease of development, FastAPI was chosen as the primary backend framework due to its lightweight nature and suitability for a semester-long project. The project involves a chatbot that responds to multiple users concurrently, making high performance through asynchronous execution essential. Additionally, the program handles multiple I/O operations, such as interacting with S3 when users upload files. FastAPI supports asynchronous programming, which helps prevent blocking issues and ensures smooth execution in such scenarios.

MySQL As mentioned in Section 2.2 The project's data structure includes relationships among Users, Chatrooms, and Documents, making a relational database the best fit. MySQL provides the necessary structure and capabilities to manage these relationships effectively.

We chose **SQLAlchemy** to connect our MySQL database with our code. SQLAlchemy is a widely-used Python SQL toolkit and Object-Relational Mapper (ORM) that simplifies database interactions. SQLAlchemy was chosen for this project due to its seamless integration with FastAPI and its robust asynchronous capabilities. SQLAlchemy supports both synchronous and asynchronous execution, allowing the project to leverage FastAPI's asynchronous architecture when paired with an async-compatible database driver, making it a scalable and efficient choice for modern web applications.

S3 Amazon S3 is used to store users' uploaded files, ensuring reliable and scalable storage. It allows users to access their files easily and meets the project's file management requirements. Also leveraging the AWS Free tier, which provides 5 GB of storage, S3 is a cost-effective solution for the project. Considering the project's size, this storage capacity is sufficient and incurs minimal to no cost.

Redis Redis serves as a key-value database, making it ideal for managing JWT tokens. It efficiently stores token IDs and subjects, ensuring high performance and seamless integration with the authentication system.

3.4 AI Modeling

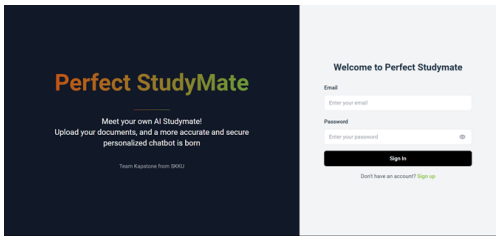
langchain Langchain is a framework that helps simplified development of applications based on LLM. Langchain can be understood as a tool that can create a pipeline that linearly connects several abstract components. Langchain has the advantage of easily implementing RAG by connecting constituent components such as data parser, embedding, and retriever. By using Langchain, we can introduce RAG, the core technology of this project, into the chatbot.

ChromaDB Vector database is mainly used as a storage for embedding data imported for RAG. Vector database stores high-dimensional vector data and shows excellent performance in searching for similarity between data. That is, the vector database is optimized for storing embedding values of natural language data. We chose our vector database as ChromaDB because of its cost-efficiency and ease for use.

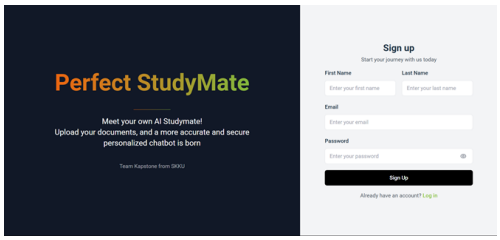
4 Implementations

4.1 UI/UX

Login/Signup The screenshot showcases a well-designed login/signup page, effectively divided into two sections. On the left, an eye-catching gradient animation and typewriter effect serve as a backdrop for concise project descriptions, highlighting key features and quickly conveying the project's core value to users. The right side presents a clean, straightforward login/signup form, prioritizing accessibility and ease of use.



(a) Login Page



(b) Signup Page

Fig. 4: Screenshot of welcome page

Main Page To ensure ease of use, the UI design was heavily inspired by ChatGPT, focusing on simplicity and user-centric interaction. The goal was to create an intuitive interface where users can navigate effortlessly and understand functionalities immediately.

Chatroom creation On the main page, users can create new chatrooms through a dialog. Each chatroom allows users to upload files and chat with the LLM-based chatbot. The file upload dialog

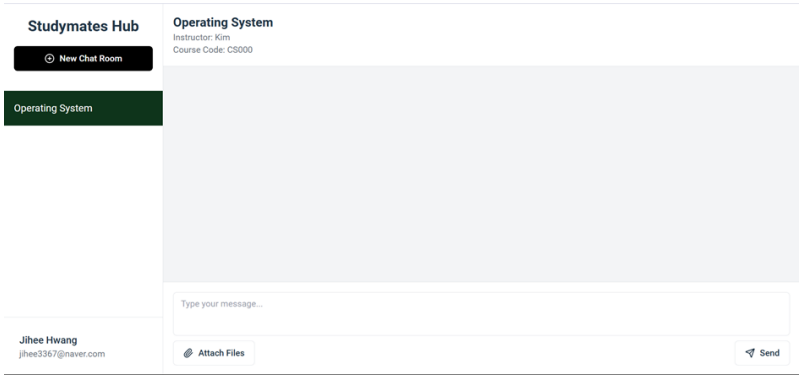
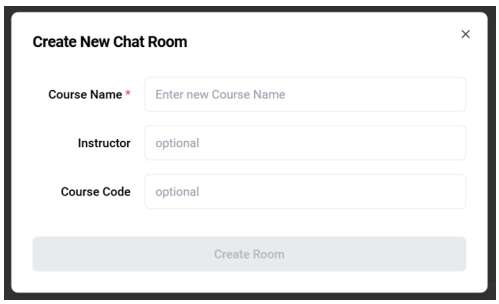
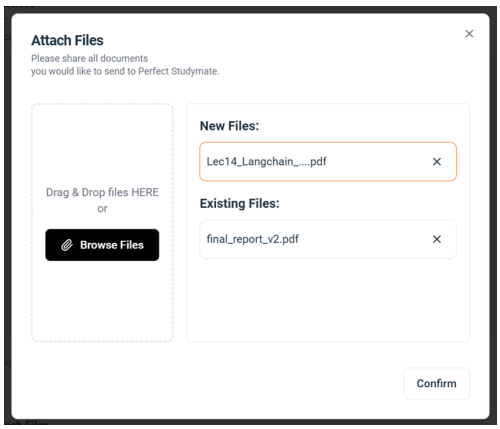


Fig. 5: Main UI design

supports both a button for file selection and drag-and-drop functionality. Additionally, the interface visually distinguishes between new and existing files, making it easy for users to differentiate between previously uploaded files and new ones. The chat interface automatically scrolls down based on the most recent message, ensuring users don't need to manually scroll to view the latest conversation. For actions like chatroom deletion and logging out, users can simply right-click on the relevant component to reveal the action buttons, making these operations easy to access. This approach helps maintain a clean design while offering a smooth and user-friendly experience.



(a) Chatroom Creation Dialog



(b) File Upload Dialog

Fig. 6: Screenshot of dialogs

4.2 Frontend

Responsive Design The layout adjusts dynamically based on the screen size, using flexbox and Tailwind's responsive classes. For example, the authentication page displays a single-column layout on smaller screens and switches to a two-column layout on larger screens. This approach ensures a consistent and user-friendly experience across devices and various screen sizes.

File upload preview In this project, the pdfjs-dist library (version 2.10.377) was used to provide a preview feature for pdf files before they are uploaded. Users can hover over the file name to instantly preview the content of their pdf files directly in the browser, improving convenience and ensuring that the correct file is selected before uploading. By using this library, the user experience is enhanced by offering an instant preview of the pdf files.

API integration The frontend is integrated with a FastAPI backend using Axios to manage HTTP requests. A custom Axios instance was configured to handle API calls consistently, with proper setup for credentials and headers. For authentication, the access token is stored in cookies, allowing automatic inclusion of authentication information in subsequent API requests. This integration includes endpoints for authentication, user management, chatroom creation, message handling, and document management. By organizing API routes as constants, we ensured maintainability and avoided hardcoding URLs.

Code quality management The project employs ESLint (version 9.11.1) to maintain code quality and consistency. Several ESLint plugins are utilized to enhance the linting process, specifically tailored for React development. These include eslint-plugin-react (v7.37.0) for React-specific linting rules, eslint-plugin-react-hooks (v5.1.0-rc.0) to enforce the Rules of Hooks and check for common issues in hook usage, and eslint-plugin-react-refresh (v0.4.12) to validate the correct usage of React Refresh. This setup ensures adherence to best practices in React development, helps catch potential errors early, and maintains a consistent coding style across the project.

4.3 Backend

Asynchronous Asynchronous programming is a key feature of the backend, enabling multiple operations to run concurrently and significantly reducing execution time for I/O-bound tasks. This approach ensures that processes such as fetching chat history, retrieving responses from the RAG model, and uploading pdf files can occur without blocking other operations. By handling these tasks asynchronously, the system minimizes the impact of I/O operations, ensuring a highly responsive and scalable platform that can support multiple users simultaneously.

JWT authentication The backend employs JSON Web Token (JWT) authentication to manage user authentication and maintain login sessions. JWT was selected due to its ability to provide stateless authentication, eliminating the need for server-side session storage. This method is particularly well-suited for the project, as it does not require long-term session management.

Caching chat session To enhance the chatbot's performance, caching is used to store active chat session data in memory. The chatbot frequently requires retrieval of contextual data to generate accurate responses, which can be resource-intensive if queried directly from the database each time. By caching chat session data, the backend reduces the frequency of database queries, allowing session context to be retrieved directly from memory when available. This optimization improves

response times and reduces database load, ensuring that the chatbot remains fast and responsive even when handling multiple concurrent users.

4.4 AI Modeling

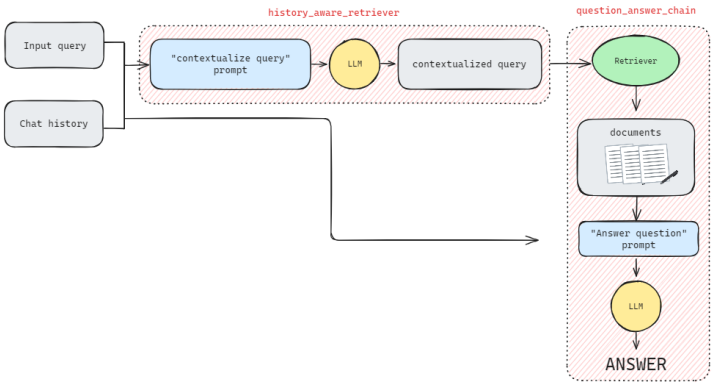


Fig. 7: AI model Design

Embeddings Manager Embeddings manager prepares vector embeddings for course materials. The order is as follows. Firstly, It loads pdf files using PyPDFLoader and splits documents into chunks (1000 characters with 200 overlaps) via RecursiveCharacterTextSplitter. Next, it embeds and stores chunks in chroma vector database with metadata (e.g., document id, user id). Lastly, it allows querying the vector database for similarity-based retrieval. Through this step, the chatbot can generate answers based on course materials with relevancy.

Response Generation Chatbot answers user queries using a RAG approach. It loades a Chroma vector database to retrieve relevant document sections based on the query. In addition to this, past conversations in the current chatroom are retrieved and inserted into the template querying to llm model. This allows a more natural conversation to take place by reflecting both the retrieved context and the interactive context (conversation history) in the response generation. The response of the chatbot generated in this way is stored in the chat history log again and then used to generate the response thereafter.

Prompt Design Appropriate prompt engineering was done to design a chatbot for the purpose of learning. It ensures answers are strictly derived from the retrieved context. Also, it forces model to respond with an explanation if the answer is not found in the documents.

5 Limitations

Data embedding delay If a user uploads large pdf files to a chat room, it may take a long time to embed the pdf files and store them in the database. This is somewhat inappropriate in chatbot environments that provide real-time experiences.

Interpretability of image data This model extracts text in a pdf file and uses it to generate an RAG response. Therefore, if image data is included in the pdf files, there is a limitation that these contents cannot be reflected in the response.

Limitations of file extensions This model is designed to read only pdf files as a source of data embedding. In other words, if the user wants to use an extension file in the .docx or .pptx format for chatting, there is an inconvenience that the file must be converted to pdf by user before upload.

6 Conclusions

We created a service named 'Perfect Studymate' that allows users to use custom chatbots with learning materials they want. We focus on allowing chatbots to respond based on facts so that users could actively use this service to review class content. To achieve the goal, we disciplined our chatbot by using appropriate prompt engineering, and tested the performance of the chatbot component using statistical tests. Additionally, the response history between the user and the chatbot was applied to generate subsequent responses so that the conversation could continue in a more interactive flow.

As well as developing chatbot models, a lot of effort has been put into connecting the frontend and backend with models. By creating and connecting several schemas defining the relationship between the user and the chatbot, various data for chatbot could be efficiently stored and used.

Although there are still unresolved limitations in many aspects, it was meaningful in that it has overcome some problems (e.g., hallucination, non-customable) of existing services.

7 References

1. AutoRAG, https://github.com/Marker-Inc-Korea/AutoRAG, 10 Januaray 2024
2. RAGAS, https://docs.ragas.io/en/latest/references/metrics, 7 November 2024

Appendix A Example of QA pairs for model evaluation

These question-answer pairs are created based on the contents in Chapters from 1 to 3 of the <Database System Concepts 6th Edition> used in the class.

1	question	Please tell me some representative examples of Database applications.
	answer	Database can be used for Enterprise Information, Banking and Finance, Universities, Airlines and Telecommunication so on.
2	question	What kinds of data storage is used in 1950s?
	answer	Magnetic tapes were developed for data storage. Data processing tasks such as payroll were automated, with data stored on tapes.
3	question	In the figure 2.5, What set of columns (attributes) is a department relation made up of?
	answer	The department relation consists of attributes named dept_name, building, budget.
4	question	Can subset of candidate key be a superkey?
	answer	No. A candidate key is a minimal superkey, that is, a set of attributes that forms a superkey, but none of whose subsets is a superkey.
5	question	If the user wants to get a name of instructors who belongs to 'Comp.sci' department, which query should be implemented? The instructor table has 3 attributes name, department, and salary.
	answer	To filter results with condition, we can use 'where' keywords. 'select name from instructors where department='Comp.sci' is desirable.
6	question	Is condition in 'having' clause applied to a sinlge tuple?
	answer	No. The condition does not apply to a single tuple; rather, it applies to each group constructed by the group by clause. SQL applies predicates in the having clause after groups have been formed, so aggregate functions may be used.

Fig. 8: Example question-answer pairs from sample documents

Appendix B Structure of the model evaluation process

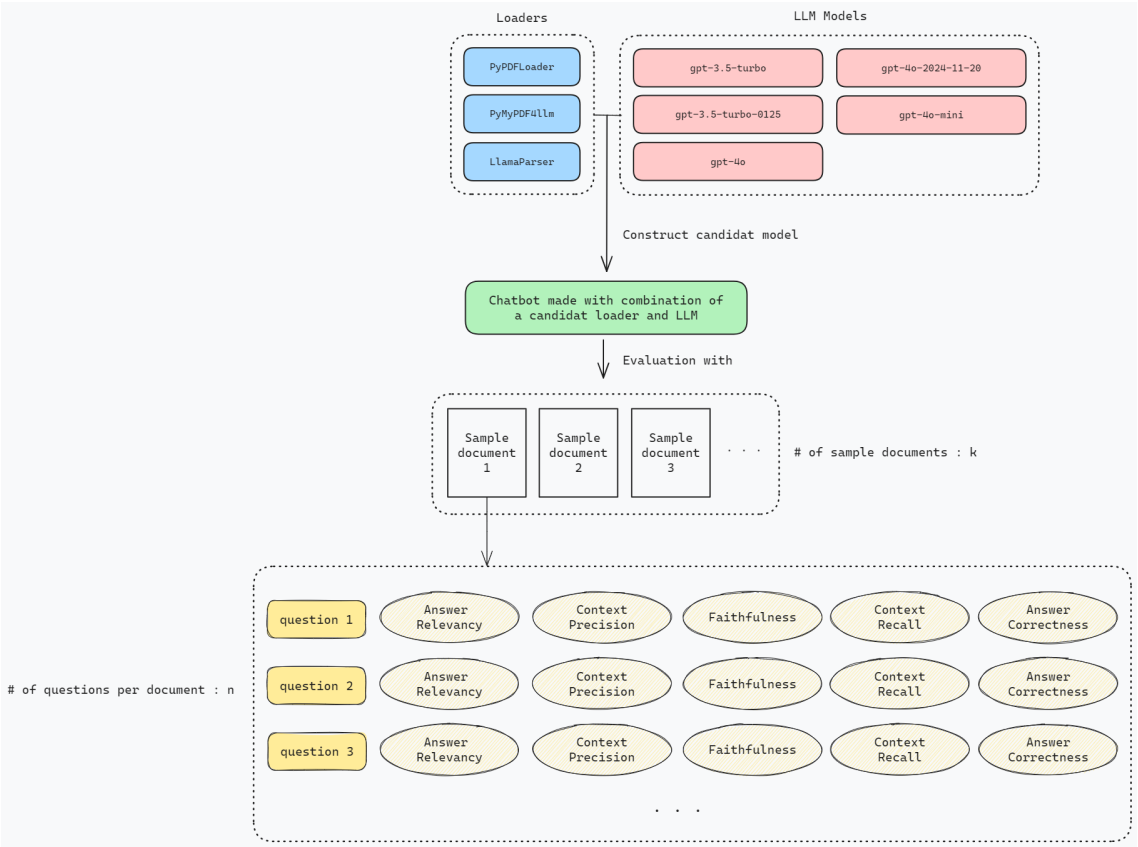


Fig. 9: Model performance evaluation process

Appendix C Performance Evaluation with statistical tests in detail

First, a normality test was performed on 15 distributions. Since the number of samples in each distribution is 45, which is not significantly large, *Shapiro Wilk Test*, which has the advantage of being robust against a small number of samples, was performed. As a result, it was confirmed that all 15 distributions had a p-value below the significance level(0.05) and did not satisfy normality.

Subsequently, it was confirmed that all distributions did not satisfy the normality, so the *Kruskal-Wallis Rank Sum Test* was performed. This is a method of testing whether multiple distributions that do not satisfy the normality have the same median value. As a result of the test, the p-value for 15 distributions was 0.99. Therefore, the null hypothesis that *'the median values of all data distributions are the same'* could not be rejected. As can be seen from the box plot in Figure 4, the 15 distributions have a very similar level of median value even to the naked eye.

Finally, an equal variance test was performed on 15 distributions. The *Levene Test* was used to test whether distributions that do not follow normality were equally distributed. As a result of the equal variance test, the p-value of 15 distributions was very high at 0.99, so the null hypothesis that *'the equal variance is satisfied for all distributions'* could not be rejected. To summarize, the results of the evaluation performed on 15 models were statistically similar across 15 models, and the final model had to be chosen by intuition rather than by statistical method.

Appendix D Performance evaluation of final model

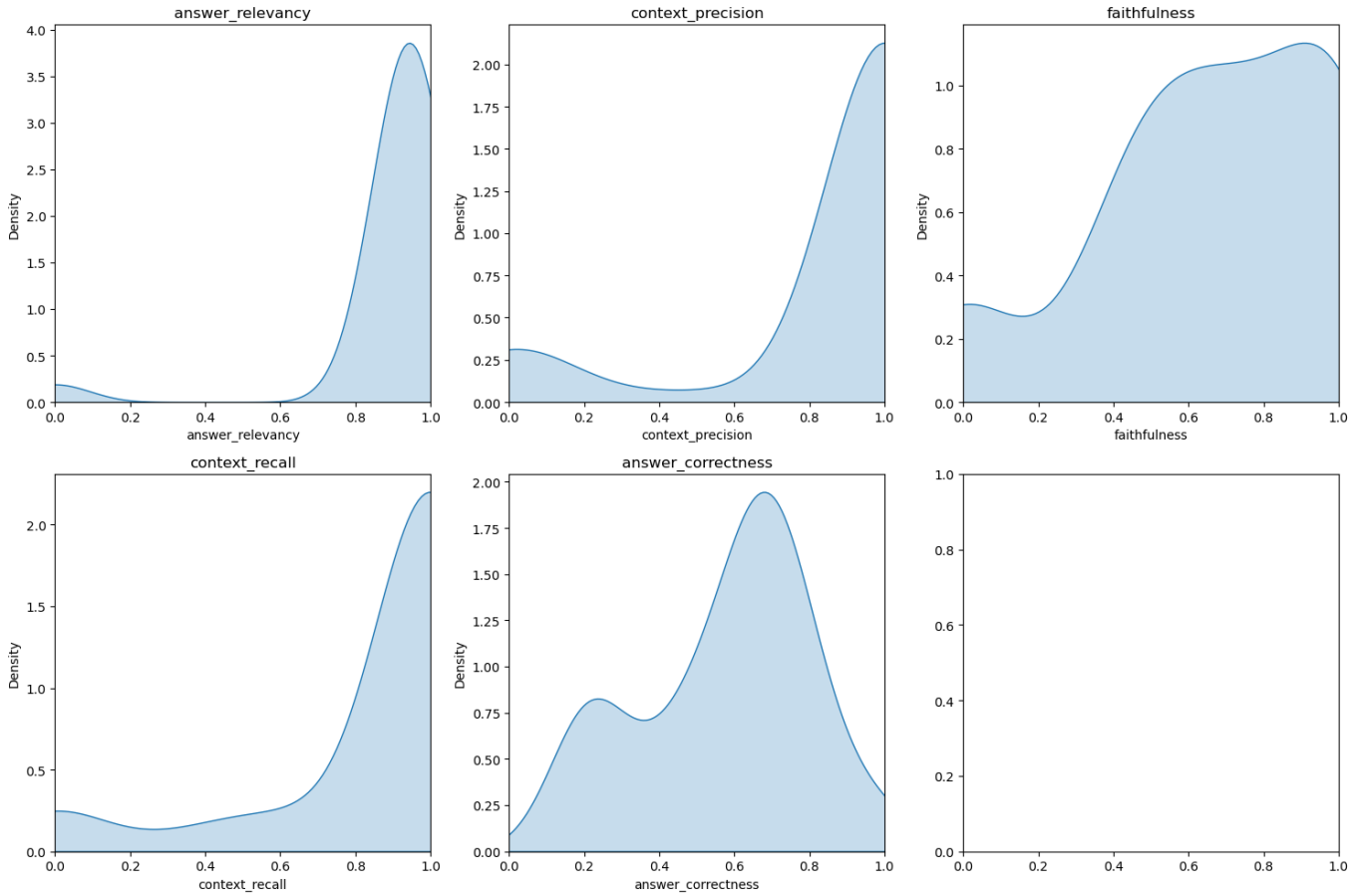


Fig. 10: Performance of 'PyPDFLoader' with 'gpt-4o-2024-11-20' model