Interwise: Integrating LLMs for Interview Practice

Xinyao Qiu
xinyaoq3@illinois.edu

Fangyang Xu
xu94@illinois.edu

Richard Yang
yueqian8@illinois.edu

Hongbin Yang
hongbin3@illinois.edu

1 Motivation and Problem Statement

The rapid expansion of the technology sector has led to a sharp increase in the demand for skilled software engineers. As companies intensify their focus on technical proficiency during hiring, technical interviews have become a critical step in the recruitment pipeline. These interviews often involve solving algorithmic and systems design problems under pressure. In response, many candidates rely on online platforms—such as LeetCode, HackerRank, and GeeksforGeeks—which offer thousands of problems to practice with.

Despite their popularity, these platforms exhibit critical shortcomings. The content is generally unstructured, static, and lacks progression tailored to individual users. Learners must manually navigate massive question banks without guidance on what to prioritize. Furthermore, these platforms often provide limited feedback beyond correctness, leaving users unsure how to improve. Without adaptivity or reflection, users risk wasting time or focusing on the wrong areas.

This points to a growing need for intelligent preparation tools that support prioritization, feedback, and learning iteration. Traditional study methods fail to adapt to a user's evolving skillset or offer targeted explanations. Many learners also lack direct access to peers or mentors, forcing them to rely on inconsistent external resources. Effective interview preparation should be guided, strategic, and confidence-building—not random or overwhelming.

Large language models (LLMs), such as OpenAI's GPT series, provide a compelling opportunity to address these gaps. LLMs can understand complex prompts, deliver contextual help, and simulate dynamic tutor-style feedback. Our project, *Interwise*, aims to combine a curated question database with LLM-powered assistance and structured tags. By integrating adaptive support and intelligent guidance, Interwise offers a more efficient, engaging, and personalized preparation experience than traditional problem banks.

2 Target Users and Use Cases

The primary target audience of *Interwise* includes computer science students, career changers, and coding bootcamp participants who are actively preparing for technical interviews. These users are typically under time constraints and need targeted, high-quality preparation materials to improve their success rates.

Unlike traditional static problem banks, *Interwise* offers interactive and adaptive features powered by LLMs. A user struggling with recursion, for instance, can receive a tailored sequence of questions with contextual hints and explanations. Filtering tools allow users to practice by company or topic, ensuring they spend time on the most relevant material.

In addition to individual learners, *Interwise* is also designed to support collaborative scenarios such as group preparation or classroom augmentation. Instructors may use the system to assign practice problems, and peers can co-edit and evaluate answers. With community-generated content and intelligent feedback, *Interwise* promotes both personalized and social learning experiences.

3 Major Functions

3.1 Frontend Major Functions

The frontend offers an intuitive interface for users to interact with *Interwise* and submit requests to the backend. It implements the following core functionalities:

- **Authentication and Session Management:** Enables users to sign up, log in, and maintain a persistent session so they can access personalized question recommendations.
- **Question and Answer CRUD:** Allows authenticated users to create, edit, and delete their own questions and answers, and to browse all content submitted by others.
- **Search and Filtering:** Supports keyword-based search and difficulty-level filters to help users quickly find relevant questions.
- **Large Language Model Assistant:** Provides an always-available sidebar powered by an LLM, enabling users to ask for guidance or clarification on any page.

As shown in Figure 1, the user interface allows users to easily compose and submit new questions with support for titles, descriptions, and tags. A LLM assistant is provided at the sidebar.

3.2 Backend Major Functions

The backend system is responsible for the core business logic and data management of *Interwise*. It is implemented as a RESTful API service using `Express` and communicates with a document-based database (`MongoDB`) to store and retrieve application data. Below are the major backend functionalities:

- **User Management:** This module handles authentication, authorization, and user profile management.
- **Question Management:** This component enables users to create, retrieve, and manage questions.
- **Answer Management:** This module allows users to respond to questions and interact with the answers of others.
- **Voting and Interaction:** Users can like/dislike content and bookmark relevant posts.
- **Search and Recommendation:** This optional module supports full-text search and personalized question suggestions using `MongoDB` search engine.

Details are in Table 1. All routes are secured via middleware enforcing access control and data validation. Rate limiting and caching

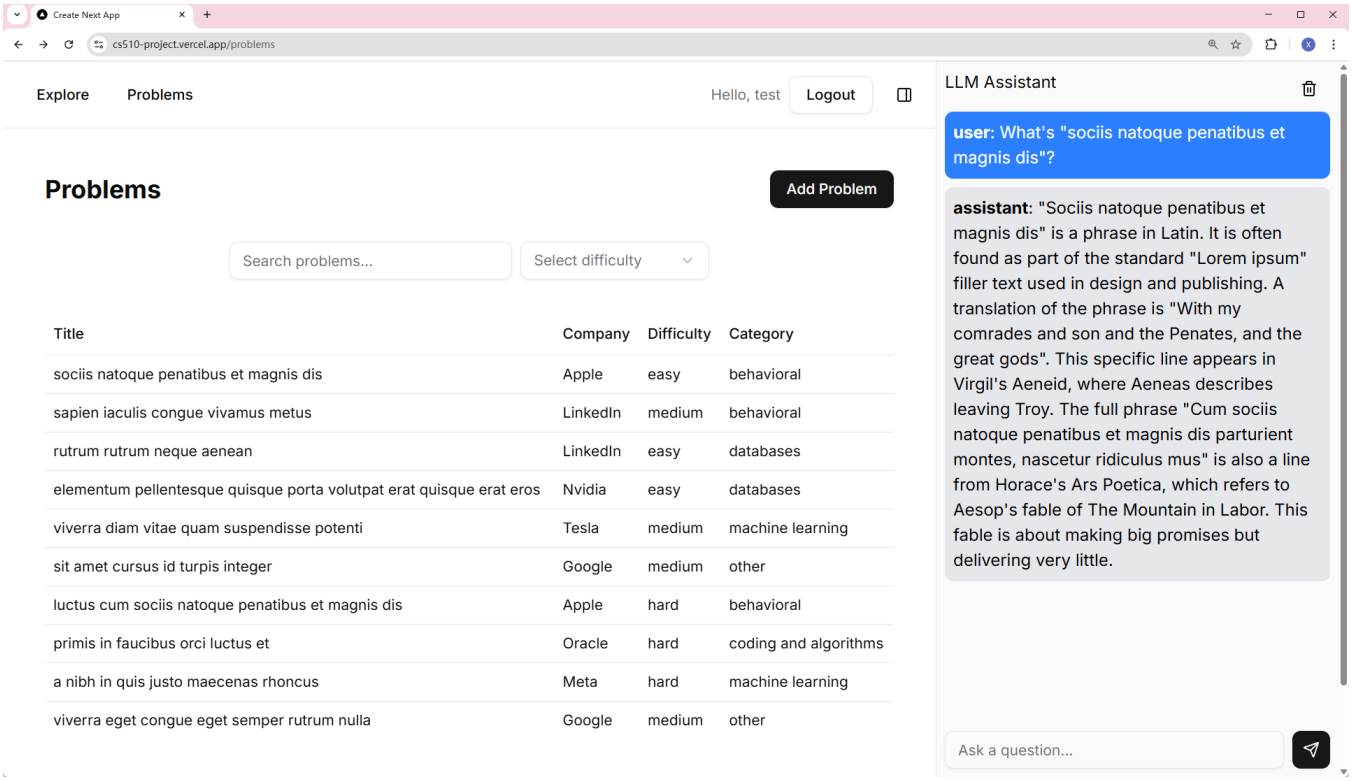

Figure 1: Sample frontend page for finding problems.

(`e.g., Redis`) are also implemented to ensure performance and security.

4 Implementation Details

4.1 Frontend and LLM Assistant Implementation Details

Our client-side and AI assistant are built on Next.js and Vercel's AI SDK. The UI is organized into four primary components:

4.1.1 Explore (Recommendation) Page. This page (`/explore`) displays a table of recommended questions for the authenticated user, with columns for *Title*, *Company*, *Difficulty*, and *Category*.

4.1.2 Problem List Page. This page (`/problems`) contains following components:

- **Add Problem:** A button that opens a modal dialog, allowing users to submit new questions to the database.
- **Search and Filter Table:** A searchable, filterable table enabling users to find questions by keyword, difficulty level, or category.

4.1.3 Problem Detail Page. This page (`/problems/:id`) contains following components:

- **Question Details:** Displays the question's title, full description, and metadata.
- **Add Answer:** A button that opens a form for submitting new solutions.
- **Answer List:** Renders all user-generated answers below the question for collaborative learning and peer feedback.

4.1.4 LLM Assistant Sidebar. The LLM Assistant is available as a collapsible panel on each page and is powered by Gemini 2.5 Flash via Vercel's AI SDK, providing high quality, contextual answers at a low cost.

4.2 Backend Implementation Details

The backend of *Interwise* is implemented using `Express` for Node.js, and is organized following a modular structure comprising controllers, services, and data access layers. The system exposes a RESTful API that interacts with a `MongoDB` instance as the primary data store.

4.2.1 System Architecture. The architecture follows a standard three-tier model: backend endpoints, database and recommendation system (Figure 2).

4.2.2 Modular Design. Each core functionality—such as question handling, answer submission, user management, and voting—is encapsulated in separate modules. A brief description of key components is as follows:

- **Controllers:** Serve as the interface between incoming requests and the business logic layer. Each controller function extracts request parameters, handles validation errors, and delegates the processing to corresponding service functions.

Table 1: Major Backend API Endpoints

Method	Endpoint	Description
POST	/api/v1/auth/register	Register a new user account
POST	/api/v1/auth/login	Authenticate user and return JWT token
GET	/api/v1/current-user	Retrieve user profile
PATCH	/api/v1/update-user	Update user profile fields
GET	/api/v1/questions	List all questions with filters
POST	/api/v1/questions	Create a new question
GET	/api/v1/questions/:id	Retrieve a specific question's detail
PATCH	/api/v1/questions/:id	Edit an existing question
DELETE	/api/v1/questions/:id	Remove a question
GET	/api/v1/answers	List all answers with filters
POST	/api/v1/answers	Submit an answer
GET	/api/v1/answers/:id	Retrieve a specific answer's detail
PATCH	/api/v1/answers/:id	Edit an existing answer
DELETE	/api/v1/answers/:id	Remove an answer
POST	/api/v1/vote	Like/dislike a question/answer
POST	/api/v1/bookmark	Bookmark a question/answer
GET	/api/v1/questions?search=...	Perform full-text or semantic search
GET	/api/v1/users/recommendations	Get personalized question suggestions

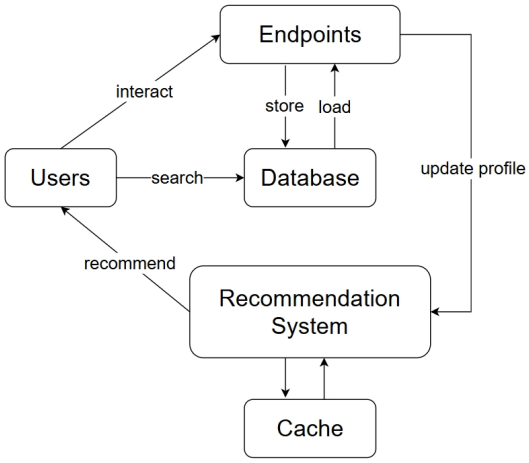

Figure 2: Backend System Architecture Overview

- **Models:** Abstract database operations using `Mongoose` models. Each model encapsulates queries and updates related to a specific entity, making the data access logic independent from business logic.
- **Routes:** Define and group RESTful endpoints for different features. Each route file maps HTTP requests to corresponding controller functions and applies middleware for tasks such as authentication and validation.
- **Utils:** Contain some functions of the application. This includes operations such as permission verification, content filtering, recommendation ranking algorithms, and voting consistency checks.

4.2.3 Authentication and Security. We use JSON Web Tokens (JWT) for user authentication. Middleware ensures that only authorized users can create or delete content. Sensitive data like passwords are stored using bcrypt hashing, and API rate limiting is enforced using `Redis`-backed token buckets.

4.2.4 Caching and Performance. Frequently accessed endpoints (e.g., homepage questions, user profiles) are cached using `Redis`, significantly reducing database load. Asynchronous background jobs (e.g., for updating popularity scores) are managed using a task queue.

4.2.5 Key Algorithm Pseudocode. Below is simplified pseudocode for the question recommendation logic based on user interaction history:

```
function recommendQuestions(userId):
    tags = getUserPreferredTags(userId)
    recent = getRecentlyViewedQuestions(userId)
    return rankBy(tagMatch(tags), exclude(recent))
```

4.2.6 Deployment. The backend is deployed on `Render` behind an `NGINX` reverse proxy.

5 Evaluation

We evaluate *Interwise* across three main dimensions: functionality, performance, and usability. The goal is to ensure that the system is correct, efficient, and user-friendly for both question askers and answerers.

5.1 Functionality Validation

To ensure the correctness and reliability of the full application, we conducted comprehensive end-to-end testing across both the frontend and backend components. Our goal was to verify that all major user interactions—from registration to question answering and search—behaved as expected in real-world usage.

The backend APIs were validated using automated tests, where each route was tested for correctness, error handling, and access control. Simultaneously, we tested the frontend interface manually and with UI testing tools, simulating realistic user workflows.

Table 2 summarizes the completion and validation status of key features that span across both layers of the system.

Table 2: Feature Validation Checklist

Feature	Verified
User registration and login flow	✓
Ask and view questions (UI/API)	✓
Submit, edit, delete answers	✓
Vote and bookmark content	✓
Search with filters and pagination	✓
Authentication and authorization handling	✓
Error reporting and UI feedback	✓
Data consistency across refresh/navigation	✓

All components were tested in both development and staging environments, with mock users simulating concurrent interactions. The application behaved as expected across all functional scenarios.

5.2 Performance Benchmarking

We benchmarked the response time and throughput of key API endpoints under concurrent load using `Apache JMeter`. Tests were performed with 100 concurrent users over 60 seconds.

Table 3: Average Response Time per Endpoint

Endpoint	Avg. Latency
`GET /api/v1/questions?limit=10`	72 ms
`POST /api/v1/questions`	104 ms
`GET /api/v1/answers?limit=10`	88 ms
`POST /api/v1/answers`	95 ms
`GET /api/v1/users/recommendations`	143 ms

The system maintained stable performance under moderate traffic, with most endpoints returning within 150 ms on average. Redis caching further improved repeated query latency by up to 60%.

5.3 Usability Evaluation

We conducted a pilot user study with 10 participants who interacted with the deployed prototype. Participants were asked to complete tasks such as asking a question, voting on answers, and using search. Feedback was collected through a brief post-task questionnaire.

Results indicated that users found the system intuitive and responsive. Suggestions included improving tag autocomplete and adding answer sorting options.

6 Usage Documentation

This section provides instructions for setting up and running the system locally for development or testing.

6.1 Repository and Requirements

The full source code is hosted at: https://github.com/XinyaoQiu/cs510-project

To run the project locally, ensure the following dependencies are installed:

- Node.js (v18 or above)
- npm (v9 or above)
- MongoDB (local instance or Atlas cloud)
- Redis (for caching and rate limiting; local or cloud instance)

6.2 Setup and Execution

Follow these steps to configure and launch the application:

Step 1: Clone the repository

```
git clone https://github.com/XinyaoQiu/cs510-project.git
cd cs510-project
```

Step 2: Configure the environment

Create a new `.env` file in the root directory and add the required environment variables. At a minimum, the following fields must be defined:

```
MONGODB_URI=<your MongoDB connection string>
REDIS_URL=<your Redis connection string>
PORT=5000
```

Replace `<your MongoDB connection string>` and `<your Redis connection string>` with the appropriate connection URIs for your setup.

Note: You may change the default server port by modifying the `PORT` value. Redis is required for enabling caching and rate limiting features.

Step 3: Install dependencies and run the backend

After cloning the repository and configuring the environment, install the dependencies and start the backend server:

```
npm install
npm run server
```

This will launch the backend server, which listens by default at: http://localhost:5000 You can change the port by modifying the `PORT` variable in your `.env` file.

6.3 Frontend Development Mode (Required for UI display)

To run the frontend interface, navigate to the `frontend` directory and start the Next.js development server:

```
cd frontend
npm install
npm run dev
```

The frontend will be available at: http://localhost:3000

Important: Ensure the backend server is running on port 5000 so that the frontend can successfully connect to the API.

6.4 Start Using the System

Once both the backend and frontend are running, you can open your browser and visit http://localhost:3000 to start interacting with the system. Features such as user login, question browsing, and LLM-based assistance will be available for testing and development.

7 Conclusion and Future work

Through the development of *Interwise*, we demonstrate the potential of integrating LLMs into a structured learning platform for technical interview preparation. The system we designed allows users to browse a curated problem set, filter questions based on attributes such as company, difficulty, and topic, and interact with an AI assistant embedded on each question page. The assistant, powered by OpenAI's GPT models, provides instant explanations, hints, and feedback tailored to the user's input and problem context.

One of the distinguishing features of *Interwise* is its emphasis on personalization and community involvement. Users can edit and tag questions, contributing to a knowledge base that grows and improves over time. The backend is powered by MongoDB with full-text indexing, enabling fast and flexible search tailored to user needs.

While the current system covers core functionalities, there are several promising directions for future work. First, we plan to implement automated code grading, allowing users to receive real-time evaluation of their coding solutions, including time and space complexity analysis. Second, we aim to design a personalized learning path generator that adapts question sequences based on performance and topic mastery. Third, we envision building a recommendation engine that leverages user behavior to suggest the next best question or topic.

Finally, we hope to expand the system to support collaborative learning scenarios, such as group interview preparation sessions or peer-to-peer feedback. By continuing to integrate LLMs with adaptive educational strategies, *Interwise* has the potential to evolve into a comprehensive platform for skill acquisition and career development.

8 Team Contribution

The successful implementation of *Interwise* was the result of well-coordinated teamwork, with each member contributing significantly to different parts of the system.

- **Xinyao Qiu** designed and implemented the backend logic and MongoDB schema. He built the API routes and ensured filtering and tagging functionalities.
- **Richard Yang** led the front-end architecture and interface development. He implemented the problem browsing page, user interface components, and integrated the LLM assistant into the platform.
- **Fangyang Xu** was responsible for implementing the core functionalities for question and answer creation, updating, and deletion on the frontend. He developed intuitive UI components to allow users to post, edit, and manage their content seamlessly.
- **Hongbin Yang** was responsible for implementing the user authentication system, including login and registration workflows. He also conducted the final system evaluation, including functionality validation and performance benchmarking across different usage scenarios.

All members actively participated in feature planning, group discussions, and system debugging. In addition, the team jointly contributed to the writing of this report and the production of the final project presentation video, ensuring a coherent and well-documented project outcome.

9 Metadata

The project is deployed on:

https://cs510-project.vercel.app

The backend services are deployed on:

https://interwise.onrender.com

The code/data of the project can be found at:

https://github.com/XinyaoQiu/cs510-project