Course Wizard

楊秉宇
R13922081

吳文心
R12922218

古聖聰
R13922143

曾家祐
R13922211

Abstract

National Taiwan University (NTU) offers nearly 10,000 courses every academic year. However, freshmen and students interested in interdisciplinary learning often face difficulties in navigating such a large selection. Currently, NTU's course platform lacks support for intuitive exploration by topic or semantic content. This project presents a course planning assistant called Course Wizard, which enables students to search courses using natural language and plan their schedules efficiently. We leverage web crawler, semantic search via vector embeddings, and an intuitive web interface to create a complete and user-friendly system. The tool is already deployed and accessible to users. Our code is released at https://github.com/detaomega/CourseWizard/.

CCS Concepts

• **Vector database**; • **Semantic search**; • **Course search**; • **Topic-base search**;

Keywords

Vector database, Semantic search, Course search, Topic-base search

1 Introduction

NTU students have access to a vast and diverse range of courses. Yet many, especially new students, find it challenging to decide which courses align with their interests due to unfamiliar terminology and lack of structured guidance. Terms like "Econometrics" or "Digital Signal Processing" are often obscure without prior exposure. The challenge is even greater for students aiming to take courses across departments, such as a Computer Science major interested in Finance.

Surveys among NTU students reveal a common regret: missing 3-5 courses they would have liked to take, often due to unawareness, scheduling conflicts, or limited semester offerings. The repetitive task of manually browsing, filtering, and comparing courses each semester is time-consuming and inefficient.

To address these issues, we propose Course Wizard, a course planning assistant that simplify the process of course exploration and selection. Our objective is to provide a tool that is not only functionally powerful but also easy to use, even for users without technical backgrounds. Specifically, Course Wizard offers semantic search capabilities and enables natural language queries, allowing students to search for topics such as "self-improvement" or "hunting" without needing to know specific course name or departmental structures. Course Wizard uncover relevant courses based on content, and automatically generates feasible course schedules by considering offering semesters and time compatibility. By lowering the barrier to course discovery and optimizing the planning process, Course Wizard empowers students to make more informed decisions and pursue their academic interests more effectively.

This project makes the following contributions:

- To the best of our knowledge, there currently exists no publicly available system that enables students to search for courses based on semantic-level topics or learning interests.
- Experimental results demonstrate that Course Wizard returns a broader and more comprehensive set of relevant courses compared to the fuzzy search functionality provided by the official NTU course platform. This indicates that our system offers improved support for exploratory course discovery, particularly in cases where students may not be familiar with domain-specific terminology.

The remainder of this report is organized as follows: Section 2 introduces the relevant background and methods used in the system. Section 3 describes the system implementation in details. Section 4 presents our empirical evaluation and comparison against the NTU course platform. Finally, Section 5 concludes the project and outlines directions for future work.

2 Methods

To address the challenges associated with course discovery and academic planning, we designed a multi-component system that integrates web crawling, semantic embedding, vector-based retrieval, and a user-friendly interface.

2.1 Web Crawler

We developed a web crawler to extract course information from the official NTU course platform. Given the presence of anti-scraping mechanisms, such as session-based authentication and dynamic page loading, we mimicked browser behavior by session management and full HTML page retrieval. The Each course's detailed HTML content was parsed using BeautifulSoup, followed by extensive data cleaning to handle inconsistencies and missing fields.

2.2 Embedding Model

Courses were encoded using the BAAI bge-m3 [2], which is a multi-functionality multilingual embedding model that allowing robust understanding of user queries in over 100 languages, and can simultaneously perform the three common retrieval functionalities: dense retrieval, multi-vector retrieval, and sparse retrieval.

2.3 Vector Database: Qdrant

We adopted Qdrant [3], a high-performance vector database, as the core infrastructure to support semantic search. Qdrant is an open-source engine written in Rust, designed for speed, reliability, and scalability. Among various available vector databases, Qdrant

consistently demonstrates superior performance across multiple metrices, including indexing speed, query throughput, and latency, as confirmed by benchmark results in [4]. In particular, it achieves low-latency responses and high RPS even under different distance metrics and recall thresholds.

2.4 Web Page Development

To enhance usability and accessibility, the system is designed as a web-based application. This allows users to interact with the course planning and search functionalities directly through a browser, without requiring any software installation or complex setup.

The front-end of the application is implemented using **React** [5], a widely adopted JavaScript library for building user interfaces. React offers a component-based architecture, enabling modular, reusable, and maintainable code. Its declarative nature also facilitates efficient rendering and state management, making it suitable for dynamic, data-driven interfaces.

To improve type safety and code reliability, we adopted **TypeScript** [8], a statically typed superset of JavaScript. TypeScript enables early detection of type-related errors during development, improves code readability, and enhances developer productivity through better tooling and editor support.

For front-end tooling and bundling, we use **Vite** [1], a modern build tool that offers significantly faster development and buld times compared to traditional tools like Webpack. Vite achieves this by leveraging native ES modules and on-demand compilation, allowing for near-instantaneous hot module replacement (HMR) during development.

The user interface design is based on the **ShadCN** [6] component library, which offers a rich set of pre-built, accessible, and customizable UI components. ShadCN emphasizes simplicity and aesthetics in its design, enabling developers to build a clean and visually appealing interfaces.

On the back-end, the web application communicates with a custom Python-based API, which serves as an intermediary between the front-end and the Qdrant vector database. When a user submits a query through the web interface, the request is sent to the Python API, and then forwarded to Qdrant server.

2.5 Infra

To simplify deployment and ensure environment consistency, the entire project is containerized using **Docker** [7]. Docker is an open-source platform that enables developers to package applications and their dependencies into lightweight, portable containers. These containers can run consistently across different environments, making them ideal for reproducible research and scalable deployment. By containerizing both the front-end and database components of the system, we eliminate common issues related to dependency conflicts and configuration drift, thereby streamlining the deployment process and improving maintainability.

3 Implementation

3.1 Web Crawler

To reliably retrieve data from the NTU course website, we first simulate the behavior of a typical user session by preloading the main course portal. This step initializes cookies and session-related headers, allowing subsequent API requests to pass the website's anti-scraping mechanisms and thus reducing the likelihood of being flagged as automated traffic. Once the session is properly established, we proceed to crawl the course list for a specified academic term. For each course entry, we then fetch its corresponding detail page, and use BeautifulSoup to extract the information we need. The collected data is subsequently structured and serialized into JSON format, serving as input for downstream components in the system pipeline. Please refer to 1 for more implementation details.

```python
# Preload main course protal
response = self.session.get('https://course.ntu.edu.tw/search/quick?s=112-2')

# Fetch course list
response = self.session.post(
    'https://course.ntu.edu.tw/api/v1/courses/search/quick',
    json=json_data
)

# Fetch & Parse course details
response = self.session.get(f'https://course.ntu.edu.tw/courses/{semester}/{course_id}')
if response.status_code == 200:
    soup = BeautifulSoup(response.text, 'html.parser')
    ...
```

Listing 1: Web crawling

3.2 Vector Database: Qdrant

Qdrant offers an official Docker file, which allows for rapid deployment of a database server. Once the server is set up, we can use Python code to handle two primary tasks: loading data into the database and querying from it.

To load data into the database, we first initialize a Qdrant client and create a collection to store the records. Then, we read a JSON file obtained from our web crawler. For each entry, we concatenate the course name, course overview, couse objective, and host department to form a singe string for embedding generation. Finally, we upload the data into the Qdrant vector database. Please refer to Listing 2 for more implementation details.

```python
# Initialize
self.client = QdrantClient(...)
self.model = SentenceTransformer(self.model_name)

# Create collection
self.client.create_collection(
    collection_name=collection_name,
    vectors_config=VectorParams(
        size=self.embedding_dim,
        distance=Distance.COSINE
    )
)

# Calculate embeddings
for course in courses:
    embedding = self.model.encode(parsed_data[attr]).tolist()
    ...

# Upload courses to Qdrant
self.client.upsert(
    collection_name=collection_name,
    points=data
)
```

Listing 2: Load course with Qdrant

As for querying the database, we also utilize the Qdrant client for interaction. The use query is first encoded into a vector representation using the same sentence transformer model. In addition, hard constraints such as semester and host department are incorporated as filter conditions. These combined parameters are then used to perform a vector similarity search against the Qdrant server. The detailed implementation is show in Listing 3.

```
# Initialize
self.client = QdrantClient(...)
self.model = SentenceTransformer(self.model_name)

# Encode user query
query_vector = self.model.encode(query).tolist()

# Prepare query filter
query_filters = []
if semesters:
    query_filters.append(models.FieldCondition(
        key="semester",
        match=models.MatchAny(any=semesters)
    ))
if departments:
    query_filters.append(models.FieldCondition(
        key="host_department",
        match=models.MatchAny(any=departments)
    ))

# Query database
results = self.client.search(
    collection_name = collection_name,
    query_vector=query_vector,
    query_filter=models.Filter(
        must=query_filters
    ) if query_filters else None,
    limit = top_k*len(self.collection_names),
    with_payload=True,
)
```

Listing 3: Search course with Qdrant

3.3 Web Page Development

The front-end parses user requirements, including the topics of interest and mandatory constraintes such as the offering department and semester. It then calls a custom service to send requests to the Qdrant search API to get the search results and display. For the implementation details of the Qdrant-based course search, please refer to Listing 3. The implementation details of the web interface are omitted here; for a demonstration of the UI design, please refer to Section 4.

```
# Service: Get search result
const getSearch = async (query: string, departments:
     string[] = []): Promise<Course[]> => {
    let query_url: string = `/api/search?q=${
     encodeURIComponent(query)}`;
    for (const department of departments) {
      query_url += `&departments=${encodeURIComponent(
     department)}`;
    }
    const response = await fetch(`${query_url}`);
    ...
}

# Query service
const filteredCourses = await getSearch(searchQuery,
     departments);
```

Listing 4: Web page

4 Experiment

We showcase Course Wizard's performance using several diverse queries and compare the results with NTU's official course platform in Section 4.1 and show the scheduling functionality in Section 4.2.

4.1 Search Results

In this section, we'll demonstrate Course Wizard's results using several interesting topics and compare them with some NTU official course platform.

4.1.1 Query "桃子". When querying "桃子", NTU's course platform returns results primarily based on literal keyword matching, as illustrated in Figure 1. Even with the "fuzzy search" option enabled, the system yields courses such as "老子", "種子學及種子處理技術", and "分子美學", which exhibit limited semantic understanding of the search intent. In contrast, our Course Wizard present superior contextual comprehension by suggesting semantically relevant courses, as shown in Figure 2. The system recommends "果樹病害" and "果樹學", which directly related to peach cultivation and management. Furthermore, it exhibits advanced semantic reasoning by suggesting "素描", recognizing that peaches are commonly used as subjects in still-life drawing exercises. This comparison highlights the enhanced capability of our system to understand context and offer educationally meaningful recommendations beyond simple keyword matching.

4.1.2 Query "修行". Subsequently, we tested the query "修行". As shown in Figure 3, the NTU course search system predominantly returned courses containing keywords "修", such as "修辭學" or "鋼琴(副修)", without accounting for the broader meaning of the query. Conversely, Course Wizard provided courses from diverse perspectives related to self-cultivation and spiritual development, including "佛教、禪與人生", "正念與自我慈悲", and "瑜珈", as shown in Figure 4. Notably, the first two courses were previously unknown to us and beyond our initial expectations and imaginations, perfectly aligning with our project's core mission: to prevent students from missing potentially interesting courses due to lack of awareness. This example further illustrates our system's capacity to transcend superficial keyword matching and deliver semantically meaningful recommendations that broaden students' academic horizons through enhanced course discovery.

4.1.3 Query "打獵". Finally, we tested the query "打獵". Remarkably, the NTU course search system returned no results, as shown in Figure 5, demonstrating a complete failure to identify relevant coursework. Conversely, Figure 6 shows that our Course Wizard provided a comprehensive array of practical courses designed to enhance hunting preparation and survival skills, including "野外生活技能", "部落漫遊", "户外基本裝備" and "定向越野". These recommendations demonstrate our system's sophisticated understanding of the conceptual relationships between hunting and the requisite preparatory skills, offering students practical coursework that would enhance their outdoor survival capabilities and equipment proficiency. This example further underscores the limitation of keyword-based search systems, when queries lack direct lexical matches in course titles, while highlighting our system's ability to

Figure 1: Fuzzy search results for ” 桃子” on NTU official course platform

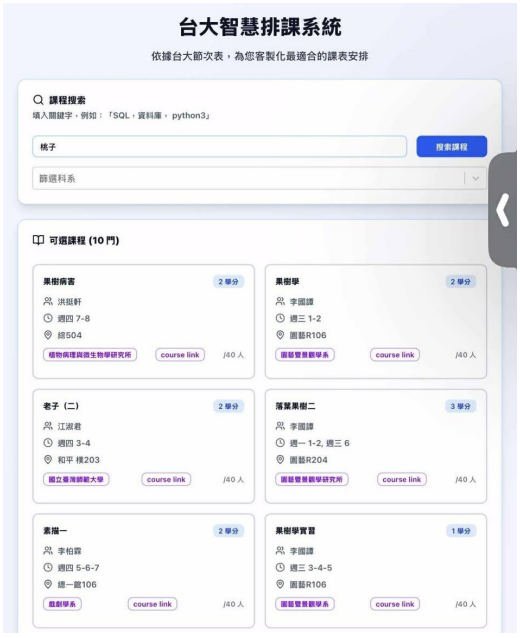

Figure 2: Search results with for ” 桃子” on Course Wizard

Figure 3: Fuzzy search results for ” 修行” on NTU official course platform

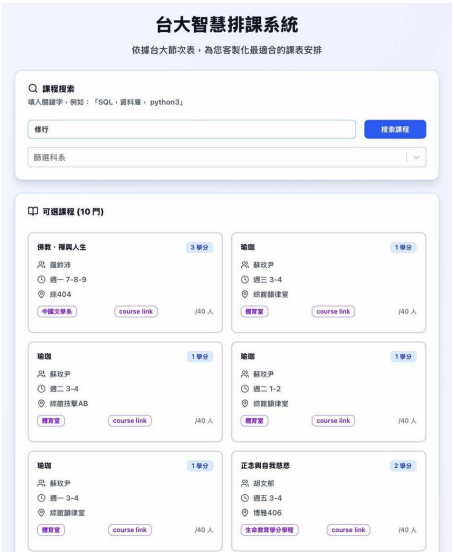

Figure 4: Search results with for ” 修行” on Course Wizard

Figure 5: Search results with for " 打獵" on NTU official course platform

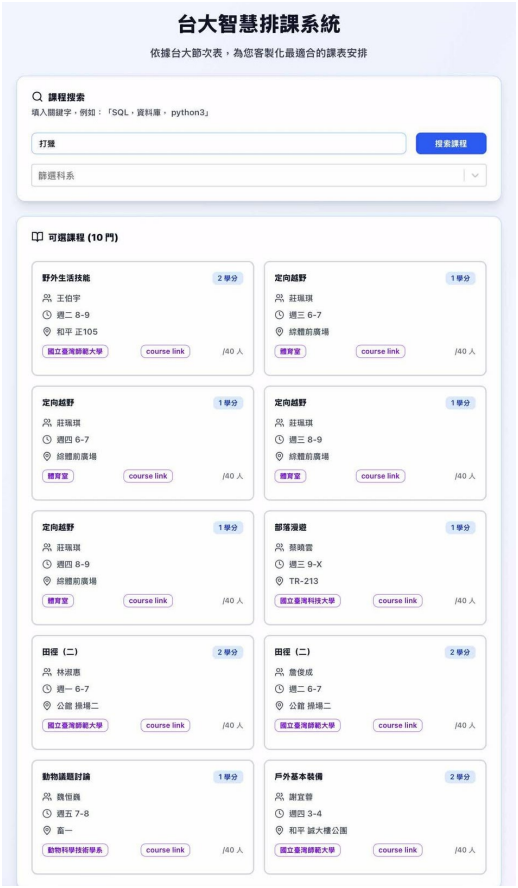

Figure 6: Search results with for " 打獵" on Course Wizard

offer semantically relevant educational pathways even for specialized interests.

4.2 Scheduling Functionality

This section showcase the web interface functionality of Course Wizard. After querying topics of interest, users can select their desired courses from the search results. By scrolling down the webpage, users can review their currently selected courses and generate a corresponding schedule based on these selections, as illustrated in Figure 7. The generated schedule is subsequently displayed below, as show in Figure 8.

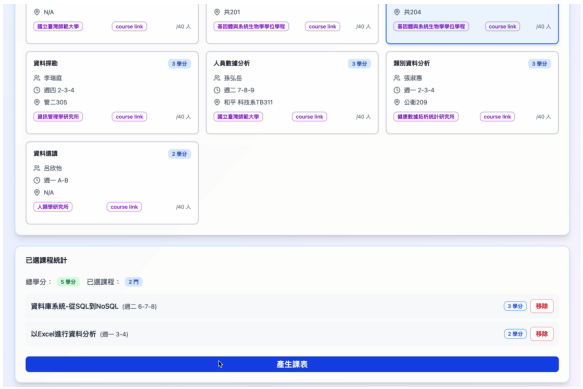

Figure 7: Generate schedule

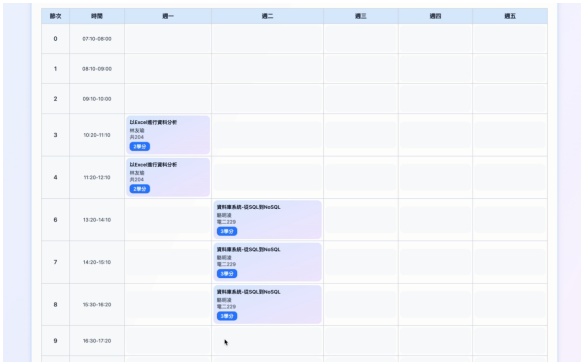

Figure 8: Result of scheduling

5 Conclusion

We have developed and deployed a functional course planning assistant tailored to NTU students. The system address a significant gap in current course exploration tools by enabling intuitive, topic-driven search and automated schedule generation. With this tool, students are less likely to miss valuable learning opportunities due to lack of awareness or scheduling conflicts. While Course Wizard showcase promising capabilities in semantic course search and recommendation, several areas warrant further development to enhance its functionality and accuracy.

5.1 Cross-Topic and Multi-Semester Schedule Generation

Currently, Course Wizard generates schedules based on courses selected from individual search queries within a single semester. Future enhancements should support cross-topic schedule generation, allowing students to integrate courses from multiple domains of interest. Additionally, implementing multi-semester planning functionality would enable long-term academic trajectory design with proper prerequisite sequencing and balanced workload distribution.

5.2 Enhanced Semantic Search Accuracy

Although our semantic search outperforms traditional keyword-based systems, search accuracy can be further improved. Future iterations should incorporate more sophisticated similarity computation methods, such as advanced transformer-based models or domain-specific embeddings. Implementing user feedback mechanisms would enable continuous refinement of recommendation algorithms based on student interactions and preferences.

References

[1] https://cn.vite.dev/ Accessed June 2025.
[2] https://huggingface.co/BAAI/bge-m3 Accessed June 2025.
[3] https://qdrant.tech/ Accessed June 2025.
[4] https://qdrant.tech/benchmarks/ Accessed June 2025.
[5] https://react.dev/ Accessed June 2025.
[6] https://ui.shadcn.com/ Accessed June 2025.
[7] https://www.docker.com/ Accessed June 2025.
[8] https://www.typescriptlang.org/ Accessed June 2025.