CourseCraft - Final Report

Word count: 1998, Penalty:0

Xiao Hu - 1005684207

Haoran Zhao - 1005839078

Ziang Jia - 1005704962

Zian Zhou - 1005910454

Introduction

Selecting the right courses can be tough, especially for first-year students who are unsure about their interests or future goals. This challenge matters because early course choices can shape a student's academic journey and career direction. Instead of offering random suggestions, our system focuses on understanding the student's background, interests, and uncertainties. Students can even upload their resumes, allowing the system to gain deeper insights into their skills and experiences. This approach helps students confirm if their chosen program suits them or if there might be better options to consider. Our system is essentially an AI academic advisor, which guides learners toward courses that genuinely fit their aspirations and potential.

Illustration / Figure:

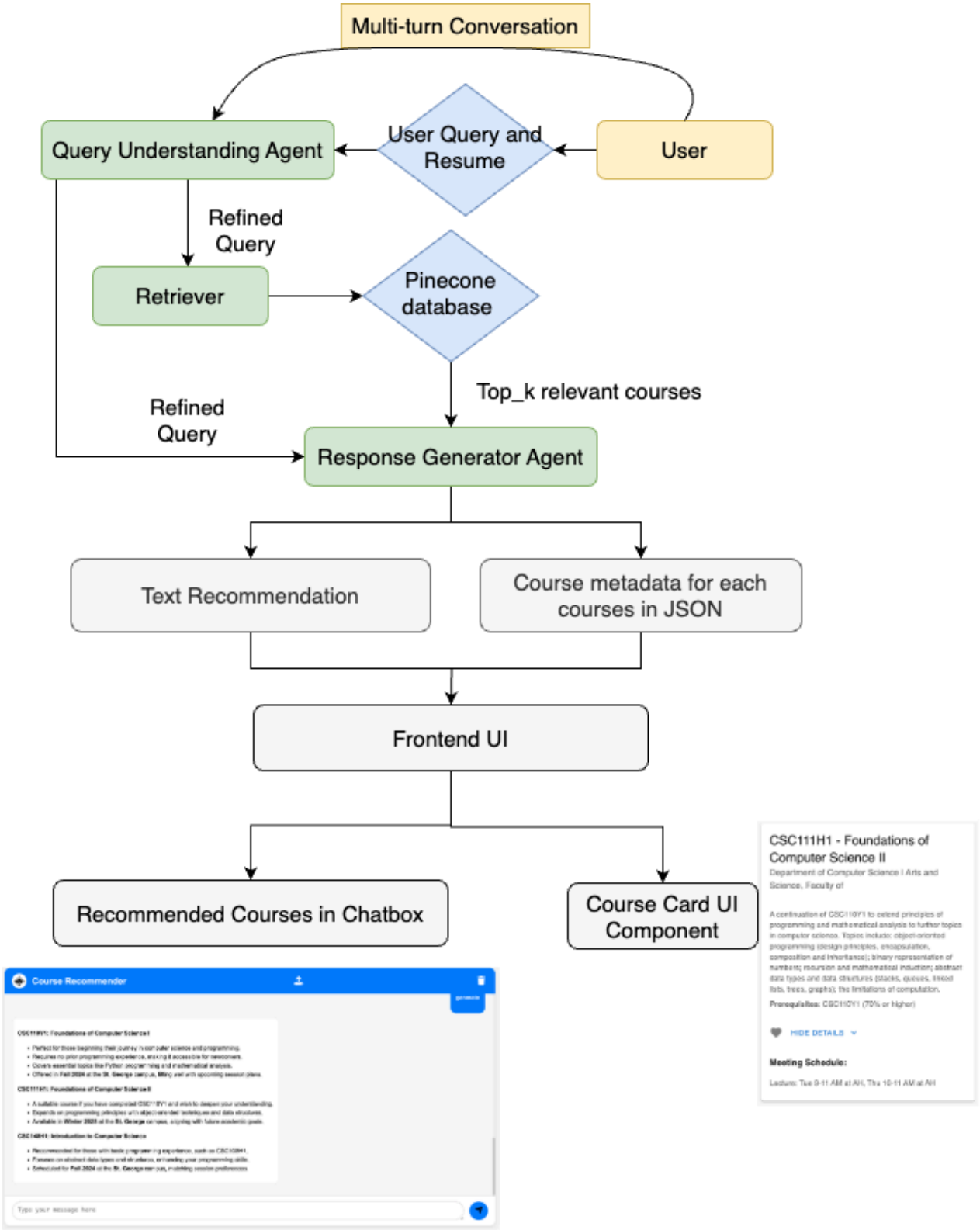


Figure1: The overall workflow of Coursecraft

Background & Related Work

One related work in course recommendation using RAG systems is Ramo[1]. RAMO (Retrieval-Augmented Generation for MOOCs) is a system designed to enhance course recommendations for Massive Open Online Courses (MOOCs) by addressing the challenges of traditional recommender systems. RAMO integrates RAG with LLMs to deliver personalized course recommendations through a conversational interface. Unlike earlier systems that relied on static data, RAMO's use of RAG allows for improved adaptation to new users and evolving course contents, making it scalable for large MOOC platforms.

While having some commonalities, the scope and implementation differ significantly. RAMO is designed for Massive Open Online Courses (MOOCs), which are tailored to a diverse, global audience with varying backgrounds. In contrast, our project targets a more specific domain: recommending courses at UofT. This narrower focus allows us to incorporate domain-specific knowledge including prerequisites and department restrictions.

Data Collection and Preprocessing

Our primary data source was the University of Toronto's official undergraduate course list. We obtained course details, including course codes, descriptions, prerequisites, and schedules. To gather this data, we implemented a custom web crawler that accessed the university's course API and retrieved information in JSON format. We processed 395 pages of results, each containing up to 20 courses, leading to a total of approximately 7,800 course entries.

After crawling, we performed comprehensive data cleaning by extracting relevant fields and transforming all course entries into JSON objects.

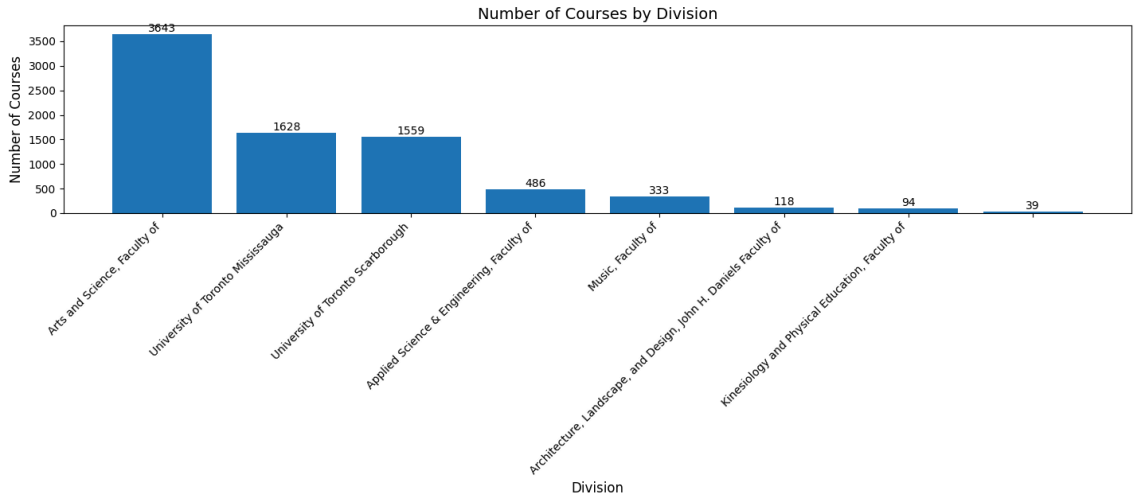


Figure2: Course Distribution Over Divisions

In fig.2, it shows course distribution over different divisions. As we can see, the majority of courses fell under the Faculty of Arts & Science division, followed by the University of Toronto Mississauga, and then the University of Toronto Scarborough.

We stored the cleaned data in a uniform structure. Each course entry included key fields: `course_id`, `course_code`, `name`, `description`, `division`, `department`, `prerequisites`, `exclusions`, `campus`, and `sessions`. We also extracted meeting sections—such as lectures and tutorials—along with their times and locations, and linked them to their respective courses.

After finalizing this standardized JSON representation, we loaded the data into MongoDB. We used two separate collections: one for course entries and one for meeting sessions. Both were keyed by a unique `course_id`, allowing us to link them efficiently.

Architecture and Software

As shown in Figure 1, the system begins by receiving a series of user inputs through a conversation. The Query Understanding Agent takes these inputs and asks follow-up questions. It aims to gain details about the user's interests, goals, and constraints. Each user message becomes part of the agent's internal conversation history. When the user decides they have provided enough information, they signal by typing a keyword like "generate." The Query Understanding Agent then produces a refined query which is a short piece of text that captures the user's needs based on the entire conversation history.

The refined query is then passed to the Retriever. We use a service called Pinecone for vector similarity search. During offline processing, each course entry is encoded into an embedding. Pinecone stores these embeddings and makes them available to the Retriever. When the Retriever receives the refined query, it encodes it using the same embedding model (text-embedding-ada-002). Then, it sends this vector to Pinecone which searches its stored embeddings and returns the top k (top 10 in our case) most similar courses.

The Retriever then takes the courses that Pinecone returns and sends them, along with the refined query, to the Response Generator Agent. This agent splits into two parts. One produces text that explains why these courses match the user's interests. The other outputs the courses in a strictly structured JSON format. It includes all needed fields so that the frontend can parse and display them as course cards. The user then observes two outputs. They see friendly, human-readable explanations inside the chat. They also see course cards arranged below the chat.

In short, the input and output flow is as follows: user inputs lead to a refined query, then the Retriever, aided by Pinecone, finds similar courses. Finally, the Response Generator

outputs text and JSON, which the frontend uses to present the courses in a clear and helpful way.

Baseline Model or Comparison

We used a GPT-4o model without any retrieval augmentation as the baseline. We gave it the same system prompts as our main system. The user and the model then engaged in a multi-turn conversation. The model produced its recommendations directly from the conversation history. It relied solely on its built-in knowledge, without access to any external course data or embeddings.

Since we kept the prompts and questions the same, the only difference was the lack of retrieval in the baseline. This lets us isolate the impact of retrieval and query refinement.

We asked five human judges to test both our baseline model and the retrieval-based system. Each judge went through ten sessions. In each session, the judge used the same input prompt on both models. This design kept conditions fair and allowed us to spot differences.

Quantitative Results

Calculated Score	Baseline	CourseCraft
Mean Average Precision	0.823	0.934
Factual Accuracy Rate	0.743	1.0
Serendipity (scale of 1-10)	6	7.5

We used three metrics to measure our model's performance.

The first is **Mean Average Precision (MAP)**. MAP measured how many of the recommended courses users found relevant. This tells how well the system matched users' interests. Our retrieval-based model scored 93.4% MAP, while the baseline scored 82.2%. This shows that adding retrieval increased the number of relevant courses.

The second is the **Factual Accuracy Rate**. We checked if the recommended courses were correct and if their details, such as prerequisites, were accurate. The baseline model sometimes invented or misunderstood course requirements, scoring only 74.3%. Our retrieval-based model, grounded in real data, reached 100% accuracy. This proves that grounding recommendations in external data leads to correct, reliable suggestions.

The third is **Serendipity**, measured on a 1 to 10 scale. Serendipity reflects how often the system suggests unexpected yet helpful courses. Users rated the retrieval-based model at 7.5 compared to the baseline's 6. This higher score means the retrieval-based model surprised users more often with courses they had not considered but still liked.

These three metrics together give a full picture of performance. They show that our RAG-based approach not only produces more accurate and relevant recommendations but also offers users more interesting options.

Qualitative Results

Good Example:

User 1 (Interaction Process): *The user is a first-year Statistics student at the University of Toronto who wants to become a data scientist. He expressed interest in data analysis and curiosity about big data. He mentioned feeling unsure about their programming abilities but remain open to improving their skills and exploring supportive resources.*

Top 10 Retrieved Courses *(For simplicity, only the course name is shown here):*

JSC370H1: Data Science II

JSC270H1: Data Science I

STA480H1: Fundamentals of Statistical Genetics

CSC110Y1: Foundations of Computer Science I

STA220H1: The Practice of Statistics I

STA302H1: Methods of Data Analysis I

CSC324H1: Principles of Programming Languages

STA302H1: Methods of Data Analysis I

CSC311H1: Introduction to Machine Learning

CSC108H1: Introduction to Computer Programming

Output of CourseCraft:

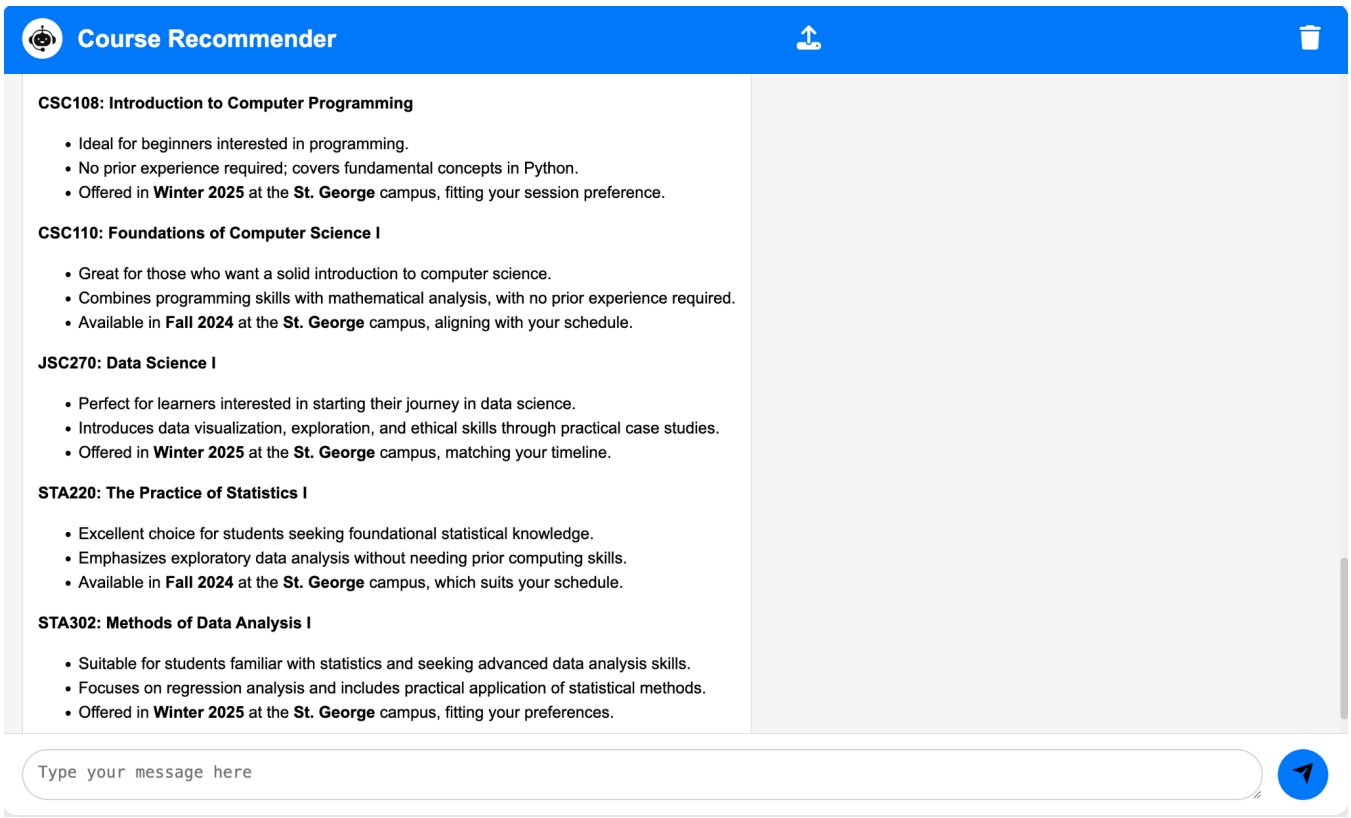


Figure 3: CourseCraft Good Output

Qualitative Analysis:

This is an example output collected from one of our testing prompts. It shows that our system has provided the student with great course selections. When a user requested data analysis and big data courses, the system offered introductory data science courses. Five human judges tested the same query, getting a **MAP of 0.843**, which means 84% of recommendations aligned with user needs. No invented courses appeared, giving a **100% Factual Accuracy Rate**. Users also noted an element of surprise, with an **8.2 out of 10 Serendipity score**, as the system provided some introductory CS courses like CSC108 and CSC111 for the user to gain programming experience, which could be very useful for their future career pursuits. This surprised the user, as they not only found these courses useful but were also impressed that the system considered their potential lack of experience.

Bad Example:

User 2: *The user is a first-year Computer Science student at the University of Toronto's St. George campus, wanting both computing fundamentals (like programming and*

algorithms) and basic math/stats that serve as requirements. No additional constraints or preferences, such as specific departments other than CS or particular delivery modes, are mentioned.

Top 10 Retrieved Course

CSC404H1 - Introduction to Video Game Design

CSC111H1 - Foundations of Computer Science II

CSC324H1 - Principles of Programming Languages

CSC384H1 - Introduction to Artificial Intelligence

CSC384H1 - Introduction to Artificial Intelligence

CSC110Y1 - Foundations of Computer Science I

CSC491H1 - Capstone Design Project

CSC148H1 - Introduction to Computer Science

CSC412H1 - Probabilistic Learning and Reasoning

CSC465H1 - Formal Methods in Software Design

Output of CourseCraft:

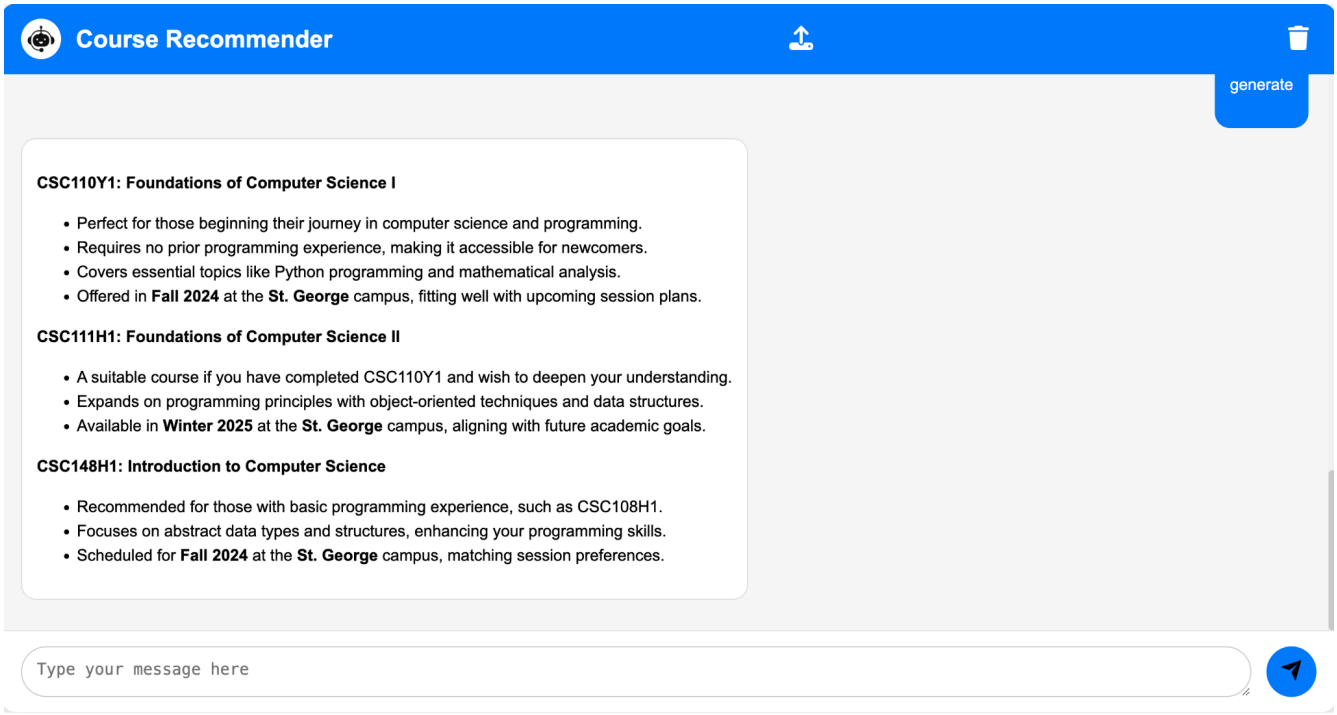
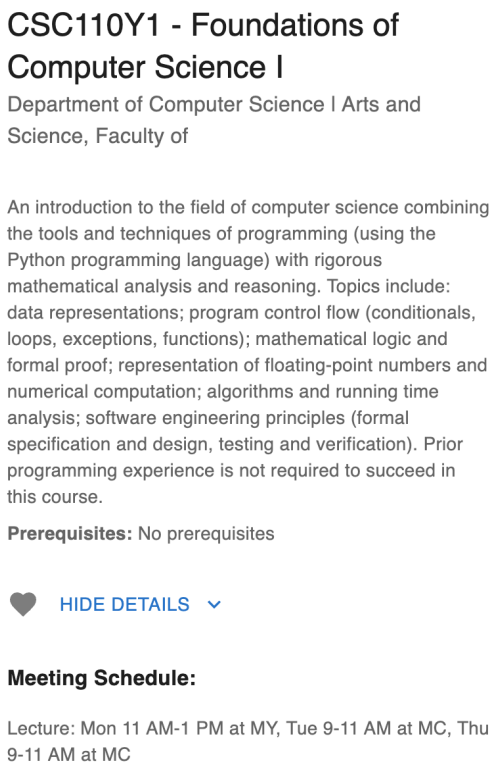


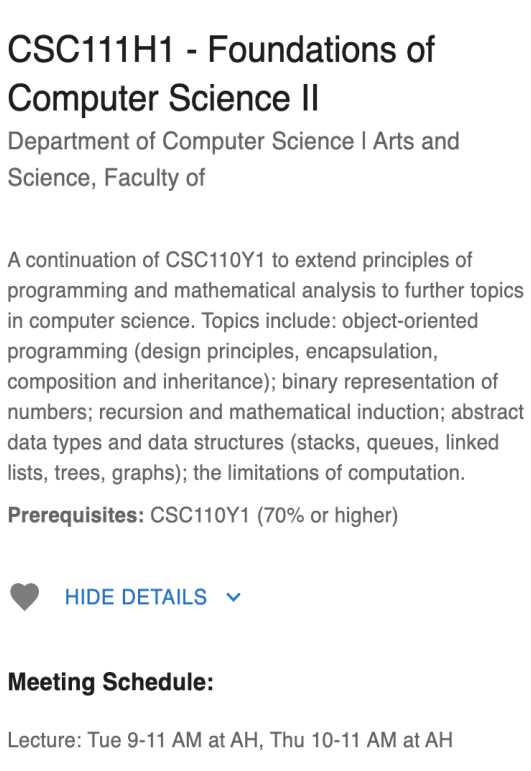
Figure 4: CourseCraft Bad Output

Qualitative Analysis:

The system's output suggests three first-year CS courses (CSC110Y1, CSC111H1, CSC148H1). These are good foundational classes for programming and computer science. Yet, the user also wanted basic math or stats courses, and those are missing. This omission lowers the **MAP to 0.65** because we missed key recommendations like MAT137 or CSC165. On the positive side, the **Factual Accuracy Rate is perfect, at 1**, since the system did not invent any courses. However, **Serendipity is only 5**, because the provided courses are useful but without any surprising recommendations.

Notice that when users provide broad requirements and only ask for specific courses without sharing detailed information about their interests, the system fails to fully understand their needs. In such cases, it may not capture everything the user wants because the input consists of single words like "programming," "algorithm," "math," or "statistics," which are too general. When this happens, the system tends to be overly conservative and lacks creativity in its recommendations.





Side Note: Our system also provides output as a course card which contains all detailed information including course description, department and faculty, prerequisites and exclusions, session information, and meeting schedules.

Discussion and Learnings

We found that our system performed well when given clear, focused prompts. For example, when a user wanted data analysis or machine learning courses, the system returned accurate and interesting options. Yet, not all scenarios were as smooth. If the prompt was vague or too broad, the system often became less creative. It sometimes recommended generic courses that did not fit well.

We also noticed that for simple questions, such as asking about prerequisites of a single course, the system sometimes fell short. Our design aimed at guiding students toward discovering interests and exploring new fields. This goal made it harder to handle straightforward queries like “What are the required courses for this program?” Our retrieval and reasoning steps focused on enriched guidance rather than listing basic facts.

Another finding involved the idea of a feedback loop. After presenting recommendations, we could invite the user to refine their preferences. Then the system could retrieve again, improving relevance and helping the user discover more fitting courses. This loop could reduce errors and improve satisfaction.

In short, our model excelled at matching nuanced, interest-based goals, but struggled with very broad or very simple queries. Adjusting prompts, and adding a refinement loop could make it more adaptable.

Individual Contributions

Xiao Hu -
Proposed the idea of using RAG for course recommendation.
Developed the Retriever and the Query Understanding Agent for the multiturn conversation.
Developed and refined the Response generation agent for text recommendations.
Prompt engineering for agents' performance.

Haoran Zhao - Designed and implemented data scraper and preprocessing pipeline. Developed and integrated course retriever. Built frontend UI for seamless interaction. Implemented and refined three AI agents (JSON Generator, Text Recommendation, Query Refinement). Conducted prompt engineering for optimized agent performance. Developed server-side code and APIs for backend-frontend integration.

Ziang Jia -
-Implemented Query Understanding Agent for input sentence processing
-Prompt engineering for performance improvement of agents

-Implemented resume upload feature
-Added server code to handle HTTP requests used for the application
-Added frontend component and improved UI of the project

Zian Zhou -
-Conduct testing with the users to evaluate the performance of CourseCraft.
-Set up conversation history in the Query Understanding Agent.
-Refactor the Query Understanding Agent to work with the latest version of openai api.
-Prompt engineering for performance improvement.

References:

[1]Rao, J., & Lin, J. (2024, July 6). *RAMO: Retrieval-Augmented Generation for Enhancing MOOCs Recommendations*. arXiv.org. https://arxiv.org/abs/2407.04925

Permissions:

	Video	Final Report	Source Code
Xiao Hu	Yes	Yes	Yes
Haoran Zhao	Yes	Yes	Yes
Ziang Jia	Yes	Yes	Yes
Zian Zhou	Yes	Yes	Yes