

An LLM Chatbot for Cornell's MPS in Information Science

INFO 5940

Jennifer Kim, Ran Tao, Jess Chen, and Ziqing Yang

2025-12-18

Introduction

The goal of this project was to create an interactive conversational assistant that helps prospective and current students explore Cornell University's Master of Professional Studies (MPS) in Information Science program. The MPS-IS degree at Cornell offers multiple concentration areas, a flexible curriculum, and a practice-oriented project requirement. However, this information is spread across several web pages and may be difficult for applicants to navigate. Also, many students may have the same recurring questions about admissions criteria, degree requirements, and the differences between MPS programs offered across Cornell.

To address this problem, we were motivated to build a chatbot that unifies these needs and provides an accessible interface to prospective and current students to easily access program information. The objective is for users to be able to ask free-form, natural language questions about different aspects of the curriculum, concentrations, admissions process, credit requirements, professional development courses, or the MPS project. The second objective is for the chatbot to (using a combination of a structured system prompt and a lightweight retrieval-augmented approach) provide grounded, document based answers.

Consequently, the “data” that our product is pulling from is a set of curated Markdown files extracted from program websites containing information like program requirements, admissions information, and curriculum descriptions. These documents are incorporated directly into the model’s context window, allowing the assistant to ground its answers in verified Cornell sources.

Justification of Approach

To solve the previously stated problem and meet the stated objectives, the team created a Shiny for Python chatbot application that allows users to ask natural-language questions and receive grounded answers based on official program materials. The product consists of two core components: a conversational interface and a curated knowledge base containing validated Markdown documents extracted from Cornell websites and program guides.

Again, the intended audience includes prospective applicants, admitted students, and current MPS students who want a clearer understanding of curriculum options, degree requirements, application components, or the structure of the MPS project—all in an easily accessible interface without having to search around too much. These groups frequently encounter fragmented or conflicting information online, and many people may rely too much on informal sources such as Reddit, Discord, or student anecdotes. By centralizing official information into a single conversational assistant, the product we created meets the needs of users who prefer immediate, direct, and personalized answers without navigating multiple university channels of information.

Our Retrieval Augmented Generation (RAG) based approach for our tool is justified because we wish to balance reliability with usability. We chose a document-grounded chatbot rather than an open-ended conversational agent because accuracy is critical for academic program guidance. Incorporating official program documents directly into the system prompt ensures that the model’s responses remain consistent with authoritative sources while avoiding hallucinations, which was a major feature of earlier LLM’s.

The Shiny for Python framework further supports these goals by enabling a clean and intuitive-to-use interface suitable for both technical and non-technical users. The chat-first design minimizes cognitive load and mirrors the way students already seek information: by simply asking questions that they would to, say, an admissions officer or a program administrator. The product will thus relieve stress from an administrative perspective and meet the needs of program directors, who want to reduce spending time answering similar questions over and over again.

Design Process

The design process for this project had to consider challenges and decisions in prompt engineering, retrieval-augmented generation, and natural language interfaces. We began by scoping the problem domain, identifying the specific informational needs of prospective MPS students, and mapping these needs to how to structure the system prompt. As we learned in class, system prompts are essential in setting behavior, tone, and memory constraints

Drawing on these concepts, we constructed a detailed system prompt that establishes the assistant’s role, sets boundaries around speculation, enforces accuracy when discussing academic

policies, and explicitly sets the model’s tone to be supportive and professional. For example, our system instructions included guidance on tone like, “Be helpful, encouraging, and patient with all users.” We decided to have these kind of system prompts and explicit, deterministic instructions because we knew that they, along with chat history, majorly influence the quality of model responses across turns.

Another design decision involved grounding the model in authoritative program information. In previous lectures, we explored vector stores, embeddings, chunking strategies, and retrieval tools. For the current iteration of this project, however, we realized there was a relatively small scope, well-structured set of program documents. Thus, we made the design choice to adopt a prompt-level RAG approach, where Markdown documents are concatenated into a knowledge base string injected directly into the system prompt.

This mirrors the “direct context injection” pattern discussed in class. We recognized the positives of this approach was the ability to avoid overhead associating with indexing, similarity search, or retrieval packages like ragnar that we were exposed to in class. (We also note that our product code is based in Python, not R.)

Following this choice, another core design challenge was optimizing the size and structure of the system prompt after embedding the entire knowledge base. We considered context-window limitations and noted from lectures that LLMs perform best when a prompt is hierarchical, well-structured, and not overly verbose. In our implementation, the `load_knowledge_base()` function reads each Markdown file from the knowledge/ directory, adds a filename-based section header (e.g., `## degree_reqs.md`), and concatenates the full contents of every file into a single string inserted directly into `SYSTEM_PROMPT`.

Because this entire block is included verbatim, we needed to curate the Markdown files themselves. This involved removing unnecessary text, ensuring consistent formatting, and keeping each document focused on a single topic so the model could interpret information effectively. This design choice gave the knowledge base a predictable, stable structure without exceeding token limits, and helped the model navigate the embedded program information even without a vector-store retriever or chunk-ranking mechanism.

Finally, our choice of interface design was informed by previous class lectures and exercises, where we built Shiny and ShinyChat applications integrating LLMs. We chose the Shiny framework because it provides a final product that gives a clean chat UI, has asynchronous streaming of responses, and has separation of UI and server logic.

Overall, our design choices were focused on giving the model enough conversational freedom, but preventing hallucinations and misinformation. We addressed this by adding explicit “what not to do” constraints to the system prompt and by ensuring that the knowledge base contained comprehensive information.

Product Testing

We conducted testing using a comprehensive question test set that is seen in the repository as file test_bank.md. This test bank covered questions across different categories such as admissions, degree requirements, and course planning. These questions were tested in the deployed Shiny app and in Positron. Responses were checked for accuracy relative to the embedded program documents, grounding (avoiding unsupported claims), and clarity.

For example, here are two queries we asked the chatbot and the responses it gave.

Sample Question #1

The following question is #8 in the test bank file.

Query: What are the minimum IS and HSS credit requirements for graduation, and what types of courses fall under each category?

Response:

 Response

To graduate from the Cornell MPS in Information Science program, you must meet specific credit requirements for both Information Systems (IS) and Human and Social Systems (HSS) courses.

Minimum IS and HSS Credit Requirements

- **Information Systems (IS):** At least 9 credits
- **Human and Social Systems (HSS):** At least 9 credits
- **Total credits for degree:** At least 30 credits of 5000-level (or higher) courses

What Types of Courses Fall Under Each Category?

Information Systems (IS) Courses

These courses focus on technical skills related to building, designing, and analyzing information systems. Topics often include:

- Programming and software engineering
- Databases and data analytics

- Machine learning, algorithms, and AI
- Systems infrastructure, security, and cloud computing

Sample IS courses: - CS 5780 Introduction to Machine Learning

- CS 5320 Introduction to Database Systems
- INFO 5100 Visual Data Analytics for the Web
- INFO 5311 Interactive Information Visualization

Human and Social Systems (HSS) Courses

HSS courses focus on how technology interacts with people, organizations, and society. These explore:

- Human-computer interaction and user experience
- Technology policy, privacy, and law
- Social, behavioral, and organizational aspects of digital systems
- Ethics and societal impacts of computing

Sample HSS courses: - INFO 5355 Human Computer Interaction Design

- DEA 6510 Human Factors and Inclusive Design
- INFO 5145 Privacy and Security in the Data Economy
- INFO 5400 Qualitative User Research & Design Methods

Note: - Some courses may fulfill either IS or HSS requirements depending on your project role or focus within the course. - For a complete, up-to-date list, refer to the MPS Course Spreadsheet or the official Cornell class roster.

Sample Question #2

Similarly, here we asked the chatbot a query that was designed to be out-of-scope or irrelevant.

Query: I love spaghetti.

Response:

Response

That's great to hear! While spaghetti isn't directly related to the Cornell MPS in Information Science program, having diverse interests and hobbies (including food!) can be a great way to connect with classmates and build community. If you have any questions about the MPS program curriculum, application process, or student life at Cornell, feel free to ask!

Review of Examples

As one can see from the examples, the chatbot's responses were grounded in the embedded program documents and written in a clear, conversational tone consistent with a helpful advising-style assistant. For program-related questions, the chatbot provided structured and accurate information without introducing unsupported claims. When given an out-of-scope or irrelevant input, the chatbot did not give an irreleavnt answer, and instead responded politely while redirecting the user toward relevant topics. This contrast shows that the system handles both informational and non-informational inputs appropriately.

Overall, our testing revealedthe chatbot reliably answered core questions and appropriately handled edge cases by declining to guess when information was not present in the knowledge base. Issues discovered during testing primarily involved wording vagueness and link accessibility. However, they were addressed by tightening the system prompt and refining the formatting of the Markdown knowledge files to make key details easier for the model to locate.

Limitations

While the chatbot provides a clear and structured way to explore the Cornell MPS in Information Science program, it also has several limitations. These limitations mainly relate to data coverage, system scope, and the nature of large language models.

First, the chatbot relies on a fixed set of documents collected from official Cornell websites at a specific point in time. Graduate program information such as course offerings, degree requirements, tuition, and application deadlines may change from year to year. As a result, some answers may become outdated if the knowledge base is not updated regularly. In future versions, this issue could be reduced by scheduling more frequent updates or by connecting the system to live data sources when possible.

Second, the scope of the chatbot is intentionally limited to the MPS in Information Science program. While this helps improve clarity and accuracy, it also means that the system cannot answer detailed questions about other graduate programs within Bowers CIS or across Cornell University. Users who want to compare multiple programs may still need to consult additional

resources. Expanding the scope to include more programs would require careful design to avoid confusion and information overload.

Third, the chatbot is designed to provide factual, policy-based information and does not evaluate applicant competitiveness or give personalized recommendations. This design choice supports fairness and reduces potential bias, but it may feel restrictive to users who are seeking advice on their chances of admission or course selection. Future versions could explore ways to offer higher-level guidance while still avoiding subjective or misleading judgments.

Finally, although the chatbot is built on verified sources, it still depends on a large language model to generate responses. This means there is always a small risk of unclear wording or incomplete answers, especially for complex or highly specific questions. Continued testing, prompt refinement, and user feedback would be necessary to further improve response quality and reliability.

Generative AI reflection

Generative AI played a central role in this project, not only as a technical tool but also as a core part of the system design. At the beginning of the project, the team viewed large language models mainly as flexible text generators that could answer user questions in a conversational way. However, as development progressed, we gained a deeper understanding of both the strengths and limitations of LLMs, which strongly influenced how we designed and constrained the final product.

One key lesson learned was that while LLMs are effective at understanding natural language questions, they are not reliable when asked to answer factual or policy-related questions without proper constraints. Early experiments showed that allowing the model to rely on general knowledge could lead to vague or inaccurate responses. This led us to shift from a more open-ended use of LLMs to a more controlled approach, where the model responds only based on a predefined set of official documents. Through this process, we learned that controlling what the model is allowed to know is often more important than increasing model complexity.

Overall, this project helped us better understand the broader role of generative AI in real-world applications. While LLMs offer powerful capabilities for interaction and language understanding, they must be used with clear boundaries and human oversight. The lessons learned from this project align closely with responsible AI practices and emphasize the importance of designing AI systems that support users without misleading them or replacing critical human judgment.