# EAI 6010 – Application of AI

##### Final Project Documentation.

# Project Name: CPS AI Assistant using RAG

# CRN: 21130

# Professor: Sergiy Shevchenko

# Team Members:

# Jibin Baby – baby.jib@northeastern.edu

# Amos Jaimes – jaimes.a@northeastern.edu

# Nivethini Senthilselvan – senthilselvan.n@northeastern.edu

# Rhythm Bhavsar – bhavsar.rh@northeastern.edu

# Project Overview

The CPS AI Assistant is an AI-powered chatbot designed to help students at the College of Professional Studies (CPS) at Northeastern University to efficiently access information about available programs, courses, and general academic queries. Navigating university websites can be time-consuming, and relying on academic advisors for every question may burden faculty. To address these challenges, this AI assistant provides 24/7 support, enabling students to retrieve accurate information quickly and conveniently.

# Table of Contents

|  |
| --- |
| [Existing System and its Drawbacks.](#_Existing_System_and_1) |
| [Proposed Solution](#_Proposed_Solution) |
| [Technologies Used](#_Technologies_Used) |
| [Tools Used](#_Tools_Used) |
| [Data Storing Workflow.](#_Data_Storing_Workflow) |
| [Data Retrieval Workflow.](#_Data_Retrieval_Workflow) |
| [Setup and Run the Program](#_6._Response_to) |
| [How to use the Program?](#_How_to_use) |
| [Key Benefits of the Assistant](#_Key_Benefits_of_1) |
| [Challenges and Drawbacks](#_Challenges_and_Drawbacks_1) |
| [Future Enhancements](#_Future_Enhancements_for_1) |
| [Conclusion](#_Conclusion) |
| [References](#_References) |

# 

# 

# Existing System and its Drawbacks

The existing system for accessing CPS program information relies on university websites, academic advisors, and community forums. While university websites and portals provide structured course details, they require manual navigation, making it difficult for students to find specific information efficiently. Academic advisors offer personalized guidance, but their availability is often limited due to high demand, making it challenging for them to handle a large volume of student queries effectively. This results in delays in response times and increased pressure on faculty. Additionally, students often turn to community forums and social media for insights, but these sources can be unreliable and may contain outdated or incorrect details. The scattered nature of information across multiple platforms further complicates the process, making it challenging to compare programs or evaluate different options effectively. Moreover, the current system lacks personalization, as it does not tailor responses based on individual student queries, resulting in a time-consuming and sometimes frustrating experience.

# Proposed Solution

The **CPS AI Assistant** addresses these challenges by implementing an AI-driven chatbot powered by **Retrieval-Augmented Generation (RAG) AI** and **vector databases** to provide instant, personalized responses to student queries. This system eliminates the need for students to manually navigate university websites or wait for academic advisor responses by offering a **24/7 automated assistant** capable of retrieving accurate course information instantly.

The chatbot operates in **two search modes**: **Program-Specific Search**, which allows students to select a program from a dropdown and ask targeted questions about course structures, tuition fees, and requirements, and **General Search**, which enables users to compare different programs, evaluate campus offerings, and gain insights into career prospects. By leveraging **vector similarity search**, the assistant retrieves precise information while minimizing inaccuracies caused by overlapping course details. Additionally, **query preprocessing and contextual refinement techniques** ensure that the chatbot accurately interprets user intent before retrieving data.

The AI system is designed to be **scalable and adaptive**, allowing for continuous updates as new programs and courses are introduced. With its ability to provide **instant, reliable, and well-structured information**, the **CPS AI Assistant** significantly reduces student dependency on academic advisors while improving the efficiency and accessibility of course-related information at Northeastern University.

# Technologies Used

## 1. Retrieval-Augmented Generation (RAG)

RAG is an advanced AI framework that enhances the accuracy and reliability of responses by integrating information retrieval with generative AI models. Instead of relying solely on pre-trained knowledge, RAG first retrieves relevant documents from a knowledge source (such as a vector database) and then generates a response based on the retrieved context. This approach significantly improves the relevance of answers by ensuring they are backed by factual and up-to-date information.

#### How RAG Works:

* Query Processing: The user inputs a query into the system.
* Information Retrieval: The system searches a vector database for the most relevant information based on semantic similarity.
* Context Injection: The retrieved content is fed into a generative model (LLMs) to refine the response.
* Final Response Generation: The LLM combines the retrieved data and generative capabilities to produce a contextually rich and accurate response.

This hybrid approach enhances answer accuracy, making the CPS AI Assistant highly effective in providing precise course-related information.

## 2. Vector Databases

Vector databases play a crucial role in efficient information retrieval by storing data in a high-dimensional vector format. Instead of performing traditional keyword searches, vector databases use semantic similarity searches, allowing them to retrieve information that is contextually relevant to a given query.

#### How Vector Databases Work:

* Embeds course data into vectors
* Indexes for fast retrieval
* Matches queries using cosine similarity
* Fetches relevant results for LLM refinement

## 3. Large Language Models (LLM)

Large Language Models (LLMs) enhance the chatbot by **understanding queries**, retrieving relevant data, and generating **accurate, context-aware responses**.

#### How LLMs work:

* **Processes** user queries in natural language
* **Retrieves** relevant data from vector search
* **Generates** refined, human-like answers
* **Improves** accuracy with contextual awareness

# Tools Used

The tools and packages we used are mentioned below:

#### CRAWL4AI - CRAWL4AI is a python package that gathers course-related information from CPS websites, including tuition fees, course structures, and program details. It uses high-speed parallel crawling to ensure the dataset remains up to date.

#### PYTHON - Python scripts clean and preprocess the extracted data, removing redundancies and ensuring structured formatting. This improves data consistency and enhances the accuracy of vector searches.

#### OLLAMA - Ollama utilizes the nomic-embed-text model to convert text data into vector embeddings. These embeddings capture semantic meaning, enabling precise cosine similarity searches.

#### SUPABASE - Supabase stores vector embeddings and scraped data, allowing real-time query processing. It efficiently retrieves relevant records through SQL functions, ensuring quick and accurate responses.

#### GROQ - GROQ is an LLM inference provider that provides an API for Llama 3.3:70B model for rapid, high-accuracy inference. This enables real-time chatbot responses, improving user experience with instant answers.

#### STREAMLIT - Streamlit provides an intuitive UI, allowing students to switch between Program-Specific and General Search modes seamlessly, ensuring efficient and accessible information retrieval.

# Data Storing Workflow

The diagram below shows how data is stored in a vector DB for the program to fetch relevant information from.

### 

## Detailed Explanation of each steps:

## 1. Data Scraping from CPS Website

Objective

Extract raw, relevant data from CPS program web pages for processing.

### Process

* **Data Source**:
  + The assistant begins with the CPS program sitemap (<https://cps.northeastern.edu/cps-program-sitemap.xml>), which lists URLs for all available program pages.
  + This sitemap ensures the pipeline comprehensively covers all program data.
* **Asynchronous Crawling**:
  + Crawl4AI has an **asynchronous web crawler** (AsyncWebCrawler) that fetches HTML content from program URLs.
* **HTML Parsing to Markdown**:
  + The crawler parses HTML into structured markdown format.

## 2. Data Preprocessing

Objective

Clean and prepare the scraped data for semantic understanding.

### Process

* **Markdown Cleaning**:
  + A preprocessing script (process\_and\_modify\_markdown) removes:
    - Headers and irrelevant sections (e.g., navigation menus).
    - Redundant or invalid markdown links, retaining only essential content.
    - Special characters and additional noisy elements.
* **Content Chunking**:
  + The cleaned markdown is split into **logical chunks** using the chunk\_program\_content function.
  + This process creates semantically coherent data chunks, such as:
    - **Chunk 1**: Program overview and highlights.
    - **Chunk 2**: Course structure and requirements.
    - **Chunk 3**: Tuition and cost details.
* **Metadata Enrichment**:
  + Each chunk is annotated with relevant metadata, including:
    - **Program Name**: Extracted from the first chunk or title tag.
    - **Program Mode**: Classified as online, on-campus, or hybrid.
    - **Campus Location**: Derived from program descriptions or inferred rules.

## 3. Vector Embeddings Generation

**Objective**

Represent textual data as semantic vectors for similarity search.

**Process**

* **Embedding API**:
  + A local **nomic-embedded-text** model instance generates high-dimensional vector embeddings for each chunk using Ollama.

## 4. Supabase Vector Database

**Objective**

Store and retrieve embeddings and metadata efficiently.

**Process**

* **Data Storage**:
  + The Supabase Vector Database stores:
    - Processed chunks.
    - Corresponding embeddings.
    - Enriched metadata.
  + This structure supports both **Program-Specific Search** and **General Search**.
* **Vector Similarity Retrieval** **Functions**:
  + User queries are converted into embeddings using the same local embedding model and compared against stored embeddings using an SQL function.
  + The database retrieves the top-k matching chunks based on similarity scores.

# Data Retrieval Workflow

The diagram below shows how data is retrieved from a vector DB and passing it to the LLM as a context along with the user query to generate accurate responses.

### **Diagram of a diagram for retrieving data AI-generated content may be incorrect.**

## 1. Streamlit UI and User Input

**Purpose**: Serve as the front-end interface for users to interact with the assistant.

**Process**:

* The **Streamlit UI** collects user queries and displays responses.
* Queries can be for **Program-Specific Search** (focused on a particular program) or **General Search** (broad comparisons across programs).
* This interface simplifies the interaction for CPS students, allowing them to search for program-related information such as course details, tuition fees, and campus-specific benefits.

## 2. Generate Vector Embeddings

**Purpose**: Convert the user query into a semantic vector representation for matching with stored data.

**Process**:

* The user query is passed to the nomic-embed-text model running on the local server.
* An API process the query and generates a high-dimensional vector embedding.
* The embedding captures the semantic meaning of the query, making it suitable for similarity searches.

## 3. Requesting Relevant Chunks

**Purpose**: Use the query embedding to search for and retrieve the most relevant data chunks stored in the Supabase vector database.

**Process:**

**Query Embedding Matching**:

* The query embedding is passed to a **Supabase RPC (Remote Procedure Call)** function named match\_site\_pages.
* Parameters include:
* query\_embedding: The vector embedding of the user query.
* match\_count: The number of top-matching chunks to retrieve (e.g., top 10).
* search\_mode: Specifies whether the query is **general** or **program specific**.
* filter: Filters results based on metadata such as program name or campus.

**Supabase Integration**:

* Supabase stores pre-processed chunks of CPS program data along with their vector embeddings and metadata (e.g., program name, course structure).
* The RPC function performs a **vector similarity search**, retrieving chunks with the highest similarity scores.

## 4. Retrieving Relevant Chunks

**Purpose**: Retrieve and concatenate the relevant chunks to provide context for the response generation model.

**Process**:

* The retrieved chunks from Supabase are concatenated into a single string, maintaining their metadata and order.
* Concatenation ensures that the response generation model has sufficient context to produce coherent and accurate answers.

**Key Script Functions**:

* concatenate\_chunks in rag.py:
* Limits the total length of concatenated chunks to avoid exceeding the input capacity of the response generation model.
* Includes metadata to improve the response’s relevance.

## 5. Groq Inference with LLAMA 3.3 70B

**Purpose**: Use a larger language model to generate a response based on the user query and retrieved context.

**Process**:

**Groq Integration**:

* The concatenated context and user query are sent to a Groq inference server running **LLAMA 3.3 70B**.
* Groq's ultra-fast processing capabilities ensure low latency, enabling real-time response generation.

**Response Generation**:

* The LLAMA 3.3 70B model processes the context and user query together.
* Outputs a coherent, markdown-formatted response that is returned to the Streamlit UI.

**Key Script Functions**:

* stream\_groq\_response in rag.py
* Sends the concatenated context and query to Groq’s LLAMA model API.
* Streams the response back to the Streamlit interface for display.

Groq inference ensures that the assistant generates high-quality responses, addressing the complexity and diversity of student queries. The large model size (70B) allows for nuanced understanding and generation, enhancing the assistant's ability to handle broad and detailed queries.

## 6. Response to User

**Purpose**: Present the generated response in the Streamlit UI.

**Process**:

* The generated response is streamed to the user interface for immediate display.
* The response is formatted in markdown for better readability.

How to Run the CPS AI Assistant?   
  
**Setup the python environment:**

* Have a machine with Python 3.9 or greater installed.
* Go to the GitHub link to download the zip or clone the repo https://github.com/jibinb961/cps\_assistant\_rag
* Open a terminal on the extracted folder
* Use this comment to create a virtual environment “ python3 -m venv venv”
* Use this comment to activate the environment “source venv/bin/activate”
* Use this comment to install all the python requirements “pip install -r requirements.txt”

**Setup .env file**

* Navigate to the cps\_assistant\_rag folder and open the folder in any code editor.
* Edit file “. env” to add the new Groq Api key.
* Go to groq website to create a new account [https://groq.com](https://groq.com/)
* Navigate to the API keys on the left side to create a new API Key.
* Update the variable in .env file to add GROQ\_API\_KEY=your\_actual\_api\_key\_here
* Save the .env file.

**Setup Ollama locally for running text embedding model**

* Go to Ollama website to create a new account. [https://ollama.com](https://ollama.com/)
* Download the Ollama to your local system
* Continue with the installation
* Open a new terminal and use the following code to install embedding model locally “ollama run nomic-embed-text”

**Setup Streamlit UI**

* Navigate to the dev folder under cps\_assistant\_rag folder
* Open a new terminal and run “streamlit run rag.py”
* This will open a new browser at <http://localhost:8501>
* Note: <http://localhost:8501> is default address given by streamlit.

# How to use the CPS AI Assistant?

**Before reading further, make sure that you have completed the steps mentioned in the Setup section.**

**Interface**: When you open the application, you will be greeted with the homepage with **Search Mode Options**:

* + **Program-Specific Search**: For focused questions about a single program.
  + **General Search**: For broad questions comparing multiple programs.

A screenshot of a computer

AI-generated content may be incorrect.

## Why two search modes?

We initially tried a single search mode, but vector similarity retrieval led to incorrect results for program-specific queries due to **overlap between courses.**

This hybrid approach ensures accuracy for focused queries while keeping flexibility for exploration!

## Option 1: Program-Specific Search

* **Select a Program**:
  + From the dropdown menu, choose the program you want to inquire about (e.g., *Master of Professional Studies in Applied Machine Intelligence – Boston*).
* **Input Your Question**:
  + Type a specific question related to the selected program
  + Example: *"What is the course structure for this program?"*
* **Submit Query**:
  + Hit the **Enter** to process your question.
* **Receive Response**:
  + The system retrieves and displays precise information about the program, such as course details or tuition fees.

## A screenshot of a computer program AI-generated content may be incorrect.

## Option 2: General Search

* **Input Your Broad Question**:
  + Type a question comparing multiple programs.
  + Example: *"Compare MPS in Applied Machine Intelligence Online and Boston."*
* **Submit Query**:
  + Press **Submit** to process your question.
* **Receive Response**:
  + The system displays a comparison, highlighting differences across programs and similarities.

A screenshot of a computer program

AI-generated content may be incorrect.

## Start a New Search

* **Button**: At the top-left corner of the interface, click **"Start New Search"** to reset the search mode.
* This ensures you can ask a fresh question without reloading the page.

# 

# Key Benefits of the Proposed Project

The CPS AI Assistant is designed to enhance the user experience for CPS students, enabling them to efficiently access information about academic programs, courses, and other details. Below are the **key benefits**, described in detail, to show how this system meets the unique needs of students.

### **1. Precision in Information Retrieval**

* **Targeted Responses**:
* For **Program-Specific Search**, the assistant uses metadata filtering and vector embeddings to ensure that the retrieved information pertains to the exact program selected.
* Example: *"What are the core courses in the online Master’s in Data Analytics?"* will provide a precise list of core courses without including unrelated details.
* **Reduced Ambiguity**:
* By using **vector similarity search**, the assistant retrieves the most relevant chunks of data, even if the user query is vague or not phrased perfectly.
* **Overlapping Data Handling**:
* The dual-mode approach ensures that overlapping course structures (e.g., shared courses between programs) do not lead to inaccurate responses.

### **2. Time-Saving Convenience**

* **Instant Answers**:
* Students no longer need to browse through multiple webpages or wait for advisor appointments. The assistant provides answers within seconds.
* **24/7 Availability**:
* Accessible at any time, the assistant caters to students who may need information outside office hours or across time zones.
* **Automated Query Processing**:
* The automated backend processes queries in real-time, ensuring minimal delay between question submission and response generation.

### **3. Enhanced User Experience**

* **Simple Interface**:
* The **Streamlit UI** is intuitive and visually appealing, making it easy for students to navigate. Clear buttons for **Program-Specific Search** and **General Search** eliminate confusion.
* **Markdown-Formatted Responses**:
* Responses are presented in a structured, readable format, making it easy to skim for key details.
* **Follow-Up Questions**:
* Students can refine their searches by asking follow-up questions, ensuring a conversational flow.

### **5. Improved Decision-Making**

* **Informed Comparisons**:
  + **General Search Mode** enables side-by-side comparisons of programs, helping students choose the right fit for their needs.
  + Example:*“Compare the cost and duration of the Master’s in Data Analytics for Boston and online options.”*
* **Detailed Program Insights**:
  + Students can explore unique features like experiential learning opportunities, visa eligibility, and program objectives, empowering them to make data-driven decisions.

### **6. Scalable and Future-Proof**

* **Handles Growing Data**:
  + The integration with **Supabase Vector Database** allows the system to store and retrieve an ever-expanding dataset as CPS adds new programs or updates existing ones.
* **Real-Time Updates**:
  + Automated data scraping ensures the assistant always provides the latest information, even when program details change on the CPS website.

# 

# Challenges and Drawbacks of the CPS AI Assistant

While the CPS AI Assistant is designed to deliver precise, efficient, and user-friendly responses, certain **challenges and drawbacks** can arise due to technical and operational constraints. Below is a detailed analysis of these challenges, categorized by technical, user experience, and system-level issues.

## **Technical Challenges**

#### ****1. Overlapping Data & Inaccurate General Search Results****

Many CPS programs share common courses, making it difficult to retrieve precise results in General Search mode. Queries may return overlapping content from similar programs, reducing relevance.

#### ****2. No Conversational Memory****

Users must re-enter context for each query since memory retention isn’t implemented. Storing large context chunks, including memory, would quickly exhaust Groq API’s free tier.

#### ****3. Dependency on Local Embedding Model****

The **nomic-embed-text** model runs locally via **Ollama**, which isn’t suitable for production due to resource constraints. A more scalable solution would involve embedding generation through **Groq** instead.

## **System Challenges**

#### ****1. High Maintenance Costs****

If multiple users query the system, Groq’s free-tier limits will be exhausted quickly, leading to potential costs for inference and model retraining.

#### ****2. Frequent Data Updates****

Tuition fees and course structures change frequently, requiring continuous scraping and reprocessing to keep responses accurate.

# 

# Future Enhancements for the CPS AI Assistant

The future enhancements planned for the CPS AI Assistant to further improve its functionality, scalability, and user experience.

## **1. Agentic RAG (Retrieval-Augmented Generation)**

AI agents will dynamically fetch data from multiple sources, including university announcements and academic portals, to ensure real-time updates. This will improve accuracy for queries requiring external context, making the assistant more versatile.

## **2. Conversation Memory**

Implementing session-based memory will allow the assistant to retain context, reducing repetitive queries and making interactions more natural. Users will receive personalized, continuous responses without needing to restate previous details.

## **3. Expanded Knowledge Base**

The database will be continuously enriched with new CPS programs, electives, faculty details, and student resources. This will transform the assistant into a comprehensive information hub, covering academic and non-academic queries.

## **4. Enhanced Vector Search**

Fine-tuning embeddings and metadata filtering will improve result precision, especially for broad or overlapping queries. Smarter ranking mechanisms will prioritize the most relevant information, reducing redundancy and enhancing query accuracy.

# Conclusion

The CPS AI Assistant project exemplifies the transformative potential of AI-driven solutions in revolutionizing academic support systems. Designed for Northeastern University’s College of Professional Studies (CPS), the assistant addresses the inefficiencies of navigating complex program information and the dependency on academic advisors for routine queries. By integrating cutting-edge technologies such as LLAMA models for semantic understanding, Supabase Vector Database for fast similarity searches, and Groq inference for low-latency response generation, the project achieves a robust Retrieval-Augmented Generation (RAG) pipeline. Its dual-mode search functionality—Program-Specific Search for focused inquiries and General Search for broader comparisons—ensures precision and flexibility in catering to diverse student needs.

The intuitive Streamlit interface, complemented by markdown-formatted responses, enhances usability, allowing students to access information quickly and efficiently. Beyond benefiting students, the assistant reduces faculty workload by automating responses to common queries, enabling advisors to focus on more personalized guidance and complex student needs.

While the system has successfully addressed challenges like overlapping program data, scalability, and embedding precision, there is ample scope for future enhancement. Dynamic data retrieval through Agentic RAG, conversation memory for personalized interactions, and an expanded knowledge base will position the assistant as a comprehensive virtual advisor. This project lays a solid foundation for scaling AI solutions in higher education, improving operational efficiency, and empowering students to make informed decisions. As an innovative and adaptive tool, the CPS AI Assistant represents a significant milestone in leveraging artificial intelligence to enhance accessibility, engagement, and overall academic experience.

# References

* Amazon Web Services. (n.d.). What is retrieval-augmented generation? Retrieved February 8, 2025, from <https://aws.amazon.com/what-is/retrieval-augmented-generation/>
* Crawl4AI. (n.d.). Crawl4AI documentation. Retrieved February 8, 2025, from <https://crawl4ai.com/mkdocs/>
* Supabase. (n.d.). AI guides. Retrieved February 8, 2025, from <https://supabase.com/docs/guides/ai>
* Ollama. (n.d.). Ollama API documentation. Retrieved February 8, 2025, from <https://github.com/ollama/ollama/blob/main/docs/api.md>
* Groq. (n.d.). Groq API documentation. Retrieved February 8, 2025, from <https://console.groq.com/docs/overview>
* GeeksforGeeks. (n.d.). Cosine similarity. Retrieved February 8, 2025, from <https://www.geeksforgeeks.org/cosine-similarity/>
* Elastic. (n.d.). What is vector embedding? Retrieved February 8, 2025, from <https://www.elastic.co/what-is/vector-embedding>