I-CAN: IISc Conventional Academic Navigator

Chaithanya M G (SR: 24862) **Kavipriya Ramasamy (SR: 24872)** **Muskan Jain (SR: 24743)**
Parmar Jaydeep Ramniklal (SR: 24842) **Prasanna Kumar B V (SR: 24774)** **Vivek H N (SR: 25613)**

Project Github URL
I-CAN URL

Abstract. In the M.Tech Online program at the Indian Institute of Science (IISc), remote students face challenges in staying informed about academic updates and clarifying academic and administrative queries due to fragmented information across various tools. Crucial information such as project deadlines, exam schedules etc. are often buried in lengthy threads impacting student engagement and academic efficiency, particularly for new joiners who require timely clarification on academic and institutional procedures. To address this problem, we propose the development of an agentic university coordination assistant, a conversational AI system that acts as a proxy for the current coordination team (IKEN). This assistant empowers students to ask natural language questions and receive accurate, personalized, context-aware, and up-to-date responses on both academic logistics and program-specific inquiries. The system architecture integrates a multi-source ingestion pipeline that extracts, preprocesses and embeds heterogeneous data from institution website, Teams channels, handbooks, emails and intranet content. The data is transformed into vector embeddings using transformer-based models and stored in a vector database to support efficient semantic retrieval. A retrieval-augmented generation (RAG) pipeline, orchestrated using LangChain, dynamically constructs responses by interfacing with large language models (LLMs).

1 Introduction

The M.Tech (Online) programme by IISc is designed for working professionals to pace themselves and pursue their post-graduate degree that has the same rigour, intense learning and practice as our full-time M.Tech degree [5]. However, these students face systematic challenges to stay up-to-date on academic or administrative information. This is due to multiple portals to look into for information such as Moodle, SAP, Teams etc.

Due to a lack of organised documentation or user-friendly search tools, the IKEN coordination team currently responds to student enquiries manually, frequently answering the same questions. Even though the IKEN Team provides a quick and speedy replies, sometimes students tend to miss important academic informations.

To address these challenges, we present I-CAN, a domain-adapted, agentic coordination assistant — a retrieval-augmented conversational AI system that could help not only the students but also the IKEN department to help student learn seamlessly during the course of their online degrees.

2 Background and Related Work

The ready availability of Large Language Models has enabled the development of intelligent bots that allows its users to converse in their natural language and get access to wide range of information. In educational purposes, especially in distance learning programs, these tools could be helpful in bridging the gap between the institutional knowledge and remote learners.

Jill Watson was one of the first virtual teaching assistants that was developed by Georgia Tech, using IBM Watson, to answer student queries on course forums [4]. However, such systems were often rigid, rule-based, or limited to single-source data [6].

To address the issues linked with the static knowledge bases, Retrieval Augmented Generation (RAG) became more better choice for these tasks. In RAG, user queries are supplemented with contextually relevant documents retrieved from a vector store before being passed to an LLM for answer generation [2]. Tools like LangChain have made it easier to build such pipelines by integrating components like vector databases (e.g., FAISS), document loaders, retrievers, and prompt templates[1].

For the matter of interest, coordination are handled with apps like: Microsoft Teams or via emails or handbooks. Our project builds upon the RAG paradigm, that we enhances using:

- Multi-source Ingestion: That involves ingestion of data from sources like websites, PDFs and handbooks.
- Integration with different tools: This involves the usage of Langchain tools to extend the functions, such as sending emails, scheduling the meetings, etc.
- Academic domain adaptation: Using prompt engineering and custom templates, assistant is tuned for the IISc's curriculum workflows.

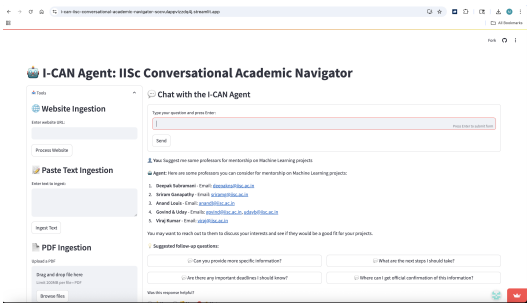

Figure 1. Interface diagram

3 Methodology

Through Streamlit UI, the user asks their query. There is a capability to ingest data also from the UI such as the PDFs. These are then converted into vector embeddings for retrieval. If no vector store is found, agent is initialized with the ingested data from UI. Otherwise, it would load the FAISS and reuse the previous content.

Once the user query is submitted, Top-5 relevant documents are retrieved. It then passes through prompt enhancer that classifies the intent and construct an enhanced prompt using the query, retrieved context and prompt templates. The data is then sent to OpenAI using the LangChain Agent. Tools are invoked automatically when necessary (e.g., sending mail or digest summary). Users then can rate the output, improving the reward model iteratively which helps in filters/reranks responses, for the next generation. Figure 3 shows the end to end flow of the whole process

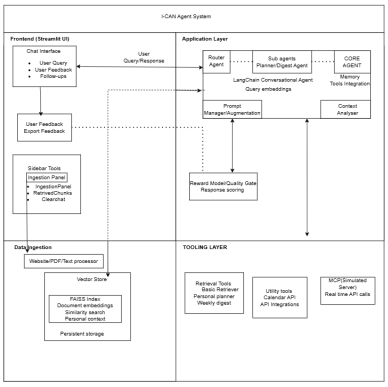

Figure 2. Overall architecture diagram

The whole project is divided into 4 parts – frontend, backend, data ingestion and tooling layer.

3.1 Data Ingestion

We are ingesting data from primary sources such as IKEN websites, PDFs and CSVs from the IKEN Portal. We first developed a Website processor that crawls the specified website and extracts the textual content. This content is then converted into manageable text chunks for further processing. Similarly, users can upload PDFs, which are processed by PDF parsers, that extracts text and then converts them into chunks.

All extracted content is split into overlapping text chunks using a RecursiveCharacterTextSplitter with a chunk size of 300 characters and an overlap of 50 characters to preserve contextual continuity. We chose 50 because around 10–30% are commonly used in practice depending on chunk size and task. These chunks are embedded using OpenAI's embedding models and stored in a FAISS vector index[3] of dimensions 1536. The FAISS index enables efficient similarity-based retrieval, which serves as the backbone for context-aware response generation in the retrieval-augmented generation (RAG) pipeline. Overall this activity resulted into 1304 chunks of data across the contents.

3.2 Backend

Backend acts as a main component that connects, handles and manages all the components together. At its core lies a LangChain-based conversational agent, which serves as the primary interface for integrating tools and APIs within the system.

3.2.1 Language Model and Retrieval-Augmented Generation

We used OpenAI's gpt-4o-mini because of its strong performance and low latency. The model is invoked within the RAG framework where it is prompted with the external knowledge base from FAISS vector store. We set the model temperature to 0.4 which helped in providing the response with minimal randomness.

3.2.2 LangChain Agent (RAG Pipeline)

The purpose of the block is to serve as a reasoning engine that responds to the student's queries by using LLM. We used Langchain's agent architecture to implement RAG, which retrieves top-ranked documents from the FAISS index and passes them as context to the LLM. We have defined some of the prompt templates that help in developing an academic-aware, context-rich response. The agent also integrates with tools that are autonomously invoked based on user intent.

3.2.3 Prompt Enhancer

The prompt enhancer module improves the performance of the LLM by analyzing user queries, injecting relevant metadata and retrieved context, and selecting an appropriate intent-based prompt template. This step helps guide the model toward more accurate and relevant responses.

3.2.4 Agent to Agent

The core design, as shown in Figure 7, it involves a Router Agent (Orchestrator) that parses user intent and dynamically delegates control to subordinate agents based on semantic content. Each subagent (e.g. DigestAgent, PlannerAgent) is itself a fully functional LangChain agent, initialized with its own reasoning loop, memory, and toolset. When the Router Agent detects a query related to a weekly summary or upcoming deadlines, it delegates execution by invoking another agent as a tool,thus enabling recursive agent composition.

3.2.5 MCP Server

We created a simulated MCP server which acted as University server. I-CAN Agent dynamically queries in MCP section in real time to fetch authoritative course data. A JSON-RPC 2.0 payload is sent to the MCP server (http://localhost:5001 in testing, replaceable with the university's production endpoint). The agent formats the MCP's structured response into natural language 4.

3.2.6 Feedback and Logging (Reward Model)

User feedback on LLM responses is collected via "Yes", "No", and "Not Sure" buttons. User feedback on LLM responses is annotated by human for testing purposes A reward model based on distilbert-base-uncased is trained as a binary classifier using this feedback to predict helpfulness of answers. The data was preprocessed, tokenized, and converted into a HuggingFace-compatible Dataset. The dataset was

split into training and testing sets (80/20), and the model was fine-tuned for three epochs with a batch size of 8 (542 data samples), and having AdamW Optimizer using the HuggingFace Trainer API. After training, the model outputs a softmax-based score between 0 and 1, representing the likelihood that a given answer would be rated as helpful. The model outputs a confidence score which is used to filter responses, with a threshold of 0.3, aligning future output with user preferences.

3.3 Tool Integration Layer

The tooling layer comprises all the tools invoked through LangChain. We have implemented a variety of tools categorized as follows:

- Retrieval tools: These tools are designed to fetch information. They include:
  - Basic Retriever
  - Enhanced Retriever
  - Weekly Digest: It searches for the academic updates. It searches the vector store for documents related to announcements. It feeds them into GPT to summarize deadlines, announcements and course updates 9.
  - Personal Planner: It helps users plan their academic workload. It searches based on the user's query and extracts essential information.
- Utility tools: These tools provide functional capabilities as needed. They include:
  - Current Date: Retrieves the current date using a local Python script.
  - Email: sends the automated emails using GMail API. It authenticates a user with OAuth of GMail and then constructs and encodes an email message and sends it using the Gmail API's.
  - Calendar: Schedules a meeting using Google Calendar API.
  - API Integrations: Enable integrations with external APIs as required

3.4 Frontend

We are using the Streamlit interface to abstract out our functionalities with a user interface. We can see the user interface has a chat interface to communicate with the LLM system. Apart from that, we also have a provision to see the retrieved chunks, configurations and any manual text inputs. We also provide a provision for feedback system so that user can make use of the Reward Model from backend.

4 Evaluation

To evaluate the effectiveness of our RAG-based academic assistant, we constructed an evaluation dataset comprising 271 question-answer pairs. Each pair consists of a student query and a corresponding ground-truth answer derived from the ingested documents. Our evaluation protocol focuses on two key components: retrieval and generation.

4.1 Retrieval

The objective is to evaluate whether the relevant documents are retrieved from the vector database for any given query. For each question, we retrieved the 5 most relevant documents. The ground truth were first embedded using the OpenAI embedding model. Then we calculated the cosine similarity of the ground truth embedding and embeddings of the retrieved documents. If any of the documents has a similarity score > 0.75, we considered it as a successful hit. We used Hit(fraction of queries for which at least one relevant document is retrieved)and MRR(Mean Reciprocal Rank based on first relevant document position) as evaluation metrics.

Model	Avg. BERTScore	delta vs GPT-4o	Significance	Interpretation
GPT-4o	0.8429	–	Baseline	Foundational model
Vanilla RAG	0.8833	+0.0404	Likely significant	Strong improvement with retrieval
RAG + Prompting	0.8794	+0.0365	Likely significant	Prompting adds useful control

Table 1. BERTScore Comparison Across Models

Model	Entailed	Not Entailed	Factual Consistency Rate	Avg Conf (Entailed)	Avg Conf (Not Entailed)
GPT-4o	41	230	15.13%	0.922	0.977
Vanilla RAG	54	217	19.93%	0.922	0.953
RAG + Prompting	32	239	11.81%	0.879	0.926

Table 2. FactCC Comparison Across Models

4.2 Generation

To assess the quality of the response generation, we used BERTScore to evaluate the similarity between then generated response and ground truth. We used bert-score library with distilbert-base-uncased as reference model. We then used the average F1 score to understand semantic overlap between texts. We also used FactCC to check the factual consistency of the generation along with the BertScore, which is just checking the semantic consistency of the generations.

5 Results

5.1 Retrieval Evaluation

For the retrieval, we achieved a score of 1 that indicates that it is able to hit atleast one document from the top 5 retrieved documents that has a confidence score of more than 0.75. The average MRR score is 0.986, which concludes that out of the top 5 retrieved documents, the pages with a higher score are either 1st or 2nd.

5.2 Generation Evaluation

Tables 1 and Table 2 show the results of the BertScore and FactCC scores. We compared three generation setups—GPT-4o, vanilla RAG, and RAG with prompting—across 271 samples using BERTScore (semantic similarity) and a robust NLI model (FactCC-style) for factual consistency. Vanilla RAG achieved the highest semantic similarity (BERTScore: 0.8833), followed closely by prompted RAG (0.8794), with GPT-4o trailing (0.8429). However, only vanilla RAG maintained strong factual consistency (19.93%), outperforming GPT-4o (15.13%) and prompted RAG (11.81%). This suggests that while prompting may refine output, it can reduce factual accuracy by overriding retrieved content. The NLI model showed greater confidence in "Not Entailed" predictions, reflecting common evaluation biases. Overall, our findings confirm that semantic similarity alone does not ensure factual correctness. Unaltered retrieval grounding, as in vanilla RAG, provides the most balanced performance for knowledge-intensive tasks.

6 Conclusion

We can conclude that we developed a conversational agent that can be helpful for the students of Mtech (Online) throughout their academics. Students can access this through deployed MVP website: I-CAN.

References

[1] Zilliz (2024), *How do I integrate LangChain with vector databases like Milvus or FAISS?*, Zilliz, 2024.

[2] AWS, *What is RAG? - Retrieval-Augmented Generation AI Explained*, AWS, 2025. Accessed: 2025-06-20.

[3] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou, 'The faiss library', (2024).

[4] Ashok K. Goel and Lalith Polepeddi, *Jill Watson: A Virtual Teaching Assistant for Online Education*, 120–143, Routledge, New York, 2018.

[5] IKEN, *IKEN Website*, IKEN, 2025. Accessed: 2025-06-20.

[6] Kakar S. Maiti P. Taneja K. Goel A. (2024) Lindgren, R., *Does Jill Watson Increase Teaching Presence?*, ACM Digital Library, 2024.

Contributions by Authors

- Chaithanya M G: I composed question–answer pairs with ground truth data to measure output quality with metrics such as BERTScore. I also investigated dynamic ingestion from SharePoint for timely alert and announcement but could not finish it because of API limitations and the requirement for an AD account.
- Kavipriya Ramasamy: Email ingestion, evaluation dataset preparation
- Muskan Jain:
  - Project Report Preparation
  - Evaluation: Implemented the Retrieval Side validation scripts.
  - Static ingestion tools for ingesting data from the CSVs
  - Tools: Implemented Utility tools - Gmail integration and Calendar Integration.
  - Ingestion Tools: Implemented bulk ingestion scripts. This helped in ingesting into the vector store.
  - Implemented the dynamic ingestion scripts so that it updates the data after specified time.
  - Validation script: Worked on the questionnaire for the validations.
- Parmar Jaydeep Ramniklal: Composed question–answer pairs with ground truth data to measure output quality with metrics such as BERTScore
- Prasanna Kumar B V:
  - Designed system architecture and backend for the Agent RAG pipeline with UI integration.
  - Built static ingestion tools: PDF parser, website processor, and manual entry interface. Trained and integrated a reward model for response filtering.
  - Implemented agent-to-agent delegation (Planner and Digester agents). Enabled tool calling for Weekly Digest, Scheduler, Personal Planner, and external API integrations.
  - Developed an MCP simulation for real-time API calling.
  - Prepared evaluation datasets and automated evaluation using BERTScore, FactCC and results interpretation.
  - Implemented prompt personalization via structured user context injection. Experimented LangGraph for dynamic agent workflows, LLM-as-a-Judge experimentation
  - Contributed to project report preparation.
- Vivek H N:
  - Prompt Engineering:Designed user prompts to improve the accuracy, tone, and relevance of LLM responses for academic queries in a RAG pipeline.
  - Deployment: Set up and deployed the Streamlit-based frontend on Streamlit Cloud, managing environment configuration, secrets handling (e.g., API keys), and testing for stability.

Appendix

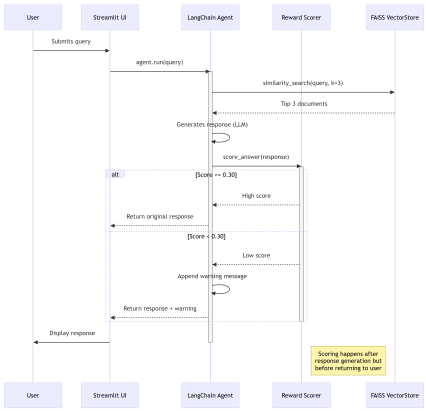

Figure 3. End to End Flow across different blocks

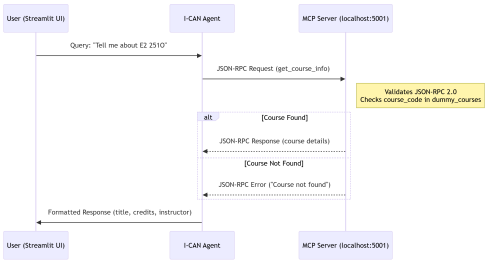

Figure 4. MCP server workflow

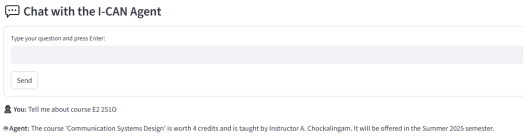

Figure 5. MCP integrated Chat interface

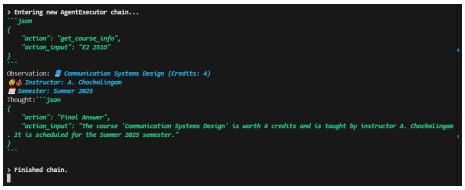

Figure 6. Chain of Thoughts with MCP

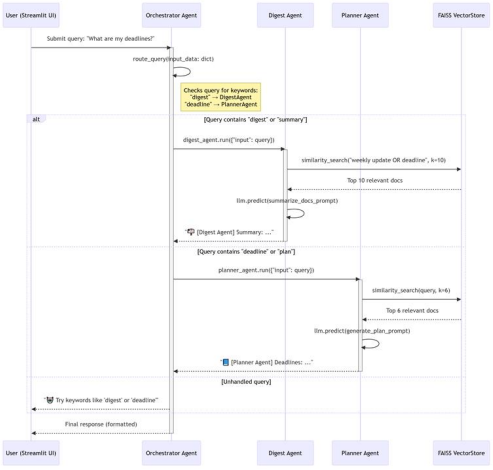

Figure 7. Agent-to-Agent communication

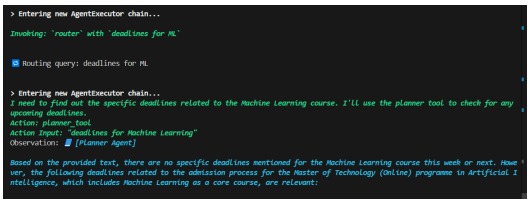

Figure 8. Agent to Agent delegation

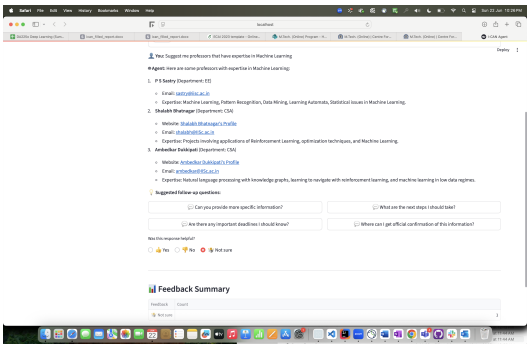

Figure 9. Email received from weekly digest

Figure 10. Running the Weekly digest

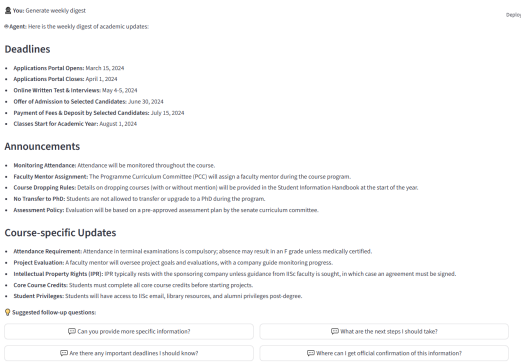

Figure 11. Prompt to invoke Weekly Digest

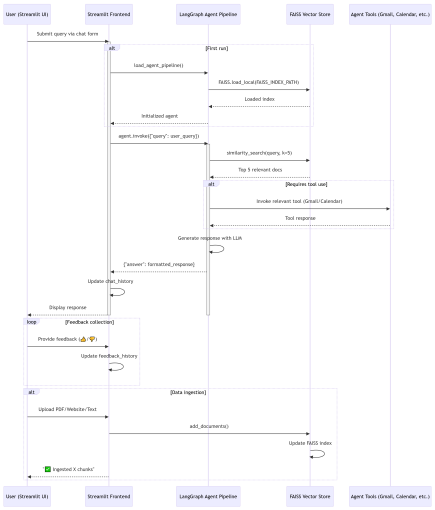

Figure 12. Flow Diagram using Langgraph

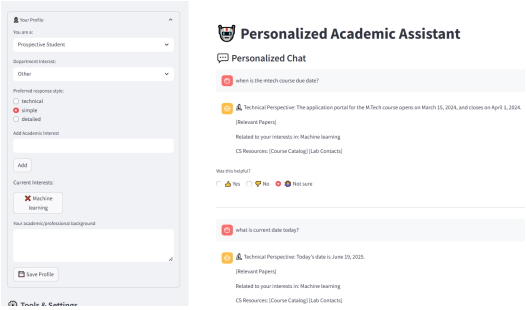

Figure 13. Prompt Personalization via Structured User Context Injection

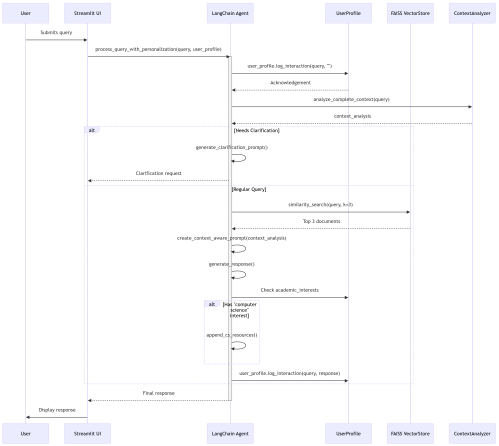

Figure 14. Personalization Flow