International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 1(2) (2026)



ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology
ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

College Enquiry Chatbot Using Natural Language Processing

BURADA SADHWIKA, BUNGA ROHIT KALYAN, GUVVALA YATISH CHANDRA, JEERU RAJU
Computer Science (Data Science)
Raghu Institute of Technology
Visakhapatnam, India
buradasadhwika23@gmail.com

MD. IMAM KHADER SHARIFF
Assistant Professor, Dept. of Computer Science
Raghu Institute of Technology
Visakhapatnam, India
shariff.imam@gmail.com

***Abstract*—When it comes to finding information about colleges, students often face a frustrating reality — slow and outdated manual processes that cannot keep up with their needs. To tackle this problem, we created an innovative College Enquiry Chatbot that combines the power of Natural Language Processing, Retrieval Augmented Generation (RAG), FastAPI, and MongoDB [1] [3]. What sets our chatbot apart is its ability to think on its feet, rather than relying on pre-programmed responses. It digs deep into institutional PDF documents and a MongoDB knowledge base to fetch relevant information, converts it into vector embeddings stored in ChromaDB, and then uses a Large Language Model to craft accurate and meaningful replies [2]. Students can ask questions about anything from admissions and fees to placements and hostel facilities, and get instant answers 24/7. By anchoring every response in factual data, our system avoids the common pitfall of AI-generated misinformation [4]. The result is a practical and scalable solution that brings college communication into the modern era through conversational AI [6].**

***Index Terms*—Chatbot, NLP, Retrieval Augmented Generation, LLM, FastAPI, MongoDB, Educational Technology**

I. Introduction

Over the past decade, the way students search for information about colleges has changed dramatically [6] [7]. Where once a phone call or a physical visit to the admissions office was the only option, today's students expect answers immediately, at any hour, from any device. Yet despite this shift in expectations, most colleges still rely on the same old manual processes — staff answering repetitive phone calls, outdated websites, and long email threads that sometimes go unanswered for days. This gap between what students need and what institutions currently offer is exactly the problem our project sets out to solve [8].

We came up with a solution by building our system around Retrieval Augmented Generation (RAG) [1]. Instead of relying on the language model to recall information from memory, RAG first searches through the college's actual documents and database records to find the most relevant information, then pulls that information and gives it to the model as context before generating a response [2]. This way, the answers are not only fluent but also factually accurate. The system is built using FastAPI as the backbone, MongoDB for storing information about colleges, ChromaDB for finding documents, and a Large Language Model that generates human-like responses [3] [5]. When students ask a question in their own words, they get a quick and accurate answer about things like admissions, fees, placements, or hostel facilities — available at any time, without needing to talk to a person.

II. Literature Review

For a while now, people have been working on making computers talk to us in a more human-like way [7]. They want to figure out how to make machines understand what we say and respond in a way that makes sense. This is a big area of research, with lots of different ideas being tried out to see what works best.

A. AI-Based Chatbot Systems

Early on, chatbots were created using a set of rules that decided how they would respond to certain words or patterns [8]. This worked okay for simple questions that were asked over and over, but it did not work well when people asked questions that were a little different from what the chatbot was programmed to handle. The people making these chatbots realized this was a problem, so they started looking into ways to make them smarter using machine learning [9]. They found out that if they trained chatbots on a lot of conversations, they got better at understanding different types of questions. However, it was still hard to get them to be really accurate, especially when it came to things like education, where the topics are very specific.

B. NLP-Based Conversational Systems

Natural language processing was a game changer for chatbots [10]. It allowed machines to take apart sentences, figure out what people meant, pick out specific details, and get the context. This gave chatbots a much better understanding of what users were really asking. Some key techniques like breaking down words, simplifying them, and figuring out the purpose of a sentence helped build most NLP systems. Researchers used these methods to create educational chatbots [6], showing that NLP-powered systems could handle many more types of student questions than systems that just looked for keywords. But even with NLP, chatbots had trouble when

the information needed was very specific or changed often, because the models could not access new data on the fly.

C. Deep Learning and Transformer-Based Models

The development of deep learning, especially transformer architectures like BERT and GPT, was a big step forward for conversational AI [11]. These models were trained on huge amounts of text data, which gave them a good understanding of how language works and what it means. Research showed that chatbots based on transformers were much better at having natural conversations than older methods. But there was still a big problem — these models could only use the knowledge they learned during training, so they could not get new information or learn about specific organizations without being retrained, which is expensive and takes a lot of time [2].

D. Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation is a way to make language models better [1]. It does not just rely on what the model already knows, but also looks up information in a big database to make sure its answers are correct [2]. This helps stop the model from making things up, which is a common problem [4]. Researchers found that this approach works really well, especially when you need to know something for sure, like in a university setting [3]. The best part is that you can update the database without having to retrain the whole model, which makes it perfect for things like college information systems.

E. Database-Driven Chatbot Architectures

Many researchers have looked into how databases and chatbot systems can work together to make it easier to find the right information [5]. Some databases, like MongoDB, are really good at handling data that is not perfectly organized and can grow with large amounts of information. Studies have shown that when you combine a database with a smart system that can find information, the answers you get are more accurate and consistent [9].

F. Research Gap

Most research looks at parts of a college enquiry system, like how to understand language or generate answers, but not many studies put all these parts together [6] [8]. Current educational chatbots have problems — they either use old data or give answers that are not based on facts [4]. Our system tries to fix this by combining different technologies, like RAG, MongoDB, FastAPI, and large language models, into one unified system that is both accurate and easy to maintain [1] [3].

III. Methodology

Creating this system wasn't just about coding, it was about planning carefully and making smart design choices. We had to really understand how students search for information [7]. Every part of the system was made for a reason and designed to work perfectly with the other parts. The main goal was simple: when a student asks a question, they should get a correct and helpful answer right away, without needing anyone's help. To make this happen, we used a step-by-step approach that covered everything from gathering data to testing the final product.

A. Data Collection and Preparation

The key to making any smart system work well is the quality of the information it uses, and our chatbot was no exception [3]. To get started, we collected detailed information about colleges from multiple PDF files containing data about courses, admissions, fees, housing, transportation, and job placement statistics. Instead of typing all this data into a database by hand, we used PyPDFLoader to automatically pull the text out of the PDFs. Since PDFs can be really long and cover many topics, we used RecursiveCharacterTextSplitter to break the text into smaller pieces [1]. The chunk size and overlap were defined as:

$$C_{\text{size}} = 800 \text{ characters} \quad (1)$$

$$C_{\text{overlap}} = 200 \text{ characters} \quad (2)$$

This ensured that each chunk retained enough surrounding context to remain meaningful during retrieval. We also added labels to each piece to identify which college it was about and what topic it covered, like fees or housing, which helped the system filter results based on what the student was asking about.

B. Vector Embedding and Storage

After cleaning, chunking, and labeling the data, the next step was making it searchable in a meaningful way [2]. To overcome the limitations of keyword-based search, we converted each text chunk into a high-dimensional numerical vector called an embedding using OpenAI Embeddings. The semantic similarity between a user query vector $\mathbf{q}$ and a document chunk vector $\mathbf{d}$ was measured using cosine similarity:

$$\text{CosineSimilarity}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \times \|\mathbf{d}\|} \quad (3)$$

A higher cosine similarity score means the query and document chunk are semantically closer [4]. All embeddings were stored in ChromaDB, which allowed us to persist the vectors on disk, saving both time and computational resources.

C. Backend Development with FastAPI

We chose FastAPI as our backend framework because it is built on Python, supports asynchronous request handling, and automatically generates interactive API documentation [5]. The overall system response time was modeled as:

$$T_{\text{response}} = T_{\text{retrieval}} + T_{\text{generation}} + T_{\text{network}} \quad (4)$$

where $T_{\text{retrieval}}$ is the time to fetch relevant chunks from ChromaDB, $T_{\text{generation}}$ is the time for the LLM to generate a response, and T_{network} is the communication latency between client and server.

D. Database Management with MongoDB

We used MongoDB, a NoSQL document-oriented database, to store two main types of data: user account information and full chat history [5]. Every new conversation session was assigned a unique thread ID generated as a 128-bit UUID:

$$\text{ThreadID} = \text{UUID4}() \in \{0, 1\}^{128} \tag{5}$$

This guaranteed that no two sessions could ever share the same ID, even across millions of users. MongoDB was connected to FastAPI using MongoEngine, and its horizontal scalability ensured consistent performance as usage grew.

E. RAG Pipeline Implementation

The Retrieval Augmented Generation pipeline was the brain of the entire system, working in two stages: retrieval and generation [1] [2]. The retrieval stage used Maximum Marginal Relevance (MMR), which balances relevance and diversity among retrieved results:

$$\text{MMR} = \arg\max_{d_i \in R \setminus S} \left[\lambda \cdot \text{Sim}(d_i, q) - (1 - \lambda) \cdot \max_{d_j \in S} \text{Sim}(d_i, d_j) \right] \tag{6}$$

where q is the user query, R is the full set of retrieved documents, S is the set of already selected documents, and λ balances relevance against diversity. The number of chunks retrieved was determined dynamically:

$$k = \begin{cases} 3 & \text{if query involves a single college or topic} \\ 5 & \text{if query involves comparison across colleges} \end{cases} \tag{7}$$

The retrieved chunks were assembled into a context block and passed to GPT-4o-mini via the RetrievalQA chain. The final response was generated as:

$$R_{\text{final}} = \text{LLM}(q, C_1, C_2, \ldots, C_k) \tag{8}$$

where $C_1, C_2, \ldots, C_k$ are the top k retrieved context chunks. By grounding every response in real retrieved data, the system effectively eliminated AI hallucination [4], making the chatbot far more trustworthy and reliable for students.

F. Frontend Development

The frontend was built using HTML and CSS, kept simple and easy to use with three main pages: registration, login, and the main chat dashboard [6]. The frontend communicated with the FastAPI backend through asynchronous HTTP requests so the page did not need to reload between messages. The overall user interaction flow can be expressed as:

$$\text{Input}(q) \xrightarrow{\text{HTTP POST}} \text{Backend} \dashrightarrow \text{LLM} \xrightarrow{\text{Response}} \text{Display}(R_{\text{final}}) \tag{9}$$

G. System Integration and Testing

When all components were developed and verified individually, the final phase involved end-to-end integration testing [9]. System accuracy was evaluated using:

$$\text{Accuracy} = \frac{\text{Number of Correct Responses}}{\text{Total Number of Test Queries}} \times 100 \tag{10}$$

Response relevance was measured by having reviewers score each response on a scale of 1 to 5, and the mean relevance score was computed as:

$$\bar{R} = \frac{1}{N} \sum_{i=1}^{N} r_i \tag{11}$$

where r_i is the relevance score of the i-th response and N is the total number of evaluated responses.

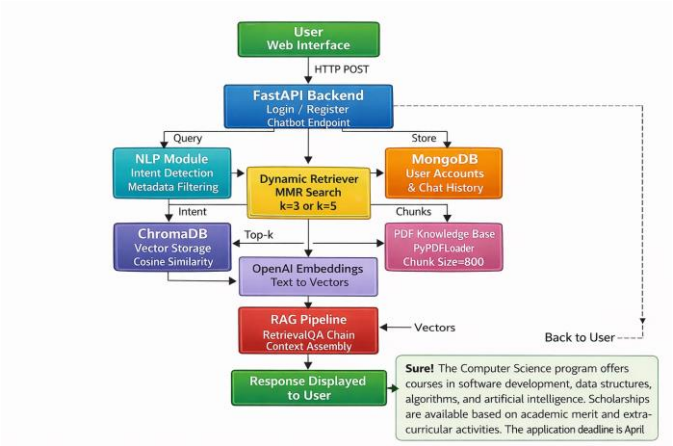


Fig. 1. System Architecture of the AI-Powered College Enquiry Chatbot illustrating the complete end-to-end pipeline. The **User** submits queries via HTTP POST to the **FastAPI Backend**, which handles authentication and routes requests. The **NLP Module** performs intent detection and metadata filtering, while **MongoDB** stores user accounts and chat history. The **Dynamic Retriever** uses MMR Search with k=3 or k=5 chunks depending on query complexity. PDF documents are processed via PyPDFLoader with chunk size of 800 characters, converted into vectors by **OpenAI Embeddings**, and stored in **ChromaDB**. The **RAG Pipeline** assembles retrieved context and passes it to **GPT-4o-mini** for accurate response generation, which is then displayed back to the user.

IV. Implementation

The implementation of the College Enquiry Chatbot was carried out in a structured and phased manner, where each module of the proposed system was developed independently and then integrated into a unified working pipeline [3]. The entire system was built using Python as the core programming language, owing to its extensive support for artificial intelligence, natural language processing, and web development frameworks [9].

A. Data Collection and Knowledge Base Construction

We started by building a large database of information about different colleges [8]. We collected details like the courses they offer, admission requirements, fee structures, accommodation facilities, transportation services, and placement statistics. We used LangChain, specifically the PyPDFLoader module, which automatically extracts text from each page and converts it into a format the computer can process [1]. We then broke the text into smaller pieces using RecursiveCharacterTextSplitter, with each piece configured to approximately 800 characters and an overlap of 200 characters between consecutive segments to preserve contextual continuity.

B. Vector Representation and Semantic Indexing

We transformed each segment into a high-dimensional numerical vector using OpenAI Embeddings [2]. These vectors capture the real semantic meaning of the text rather than just surface-level words, allowing the system to find information that is similar in meaning to what a student is asking even if the exact words differ. We stored these vectors in ChromaDB, a vector database specifically designed for fast similarity-based searches [4].

C. Backend Development and API Design

We used FastAPI to build the backend of our system [5]. It is a modern Python web framework that is fast, supports asynchronous request handling, and automatically generates interactive API documentation. The application was served using Uvicorn, a high-performance ASGI server that paired naturally with FastAPI to deliver reliable and responsive performance under real-world usage conditions.

D. Database Management and Session Handling

We used MongoDB to manage persistent data storage in our application [5]. The integration between FastAPI and MongoDB was achieved through MongoEngine. Two primary data models were implemented: user account information including email credentials and account status, and the complete history of all chatbot conversations organized under unique UUID identifiers [3].

E. Retrieval Augmented Generation Pipeline

The RAG pipeline, built using the LangChain framework, represented the most technically significant component of the entire implementation [1] [2]. The retrieval process employed Maximum Marginal Relevance (MMR) search, simultaneously optimizing for both relevance and diversity among retrieved results. The retrieved chunks were assembled into a structured context block and passed to a RetrievalQA chain that combined the retriever with the GPT-4o-mini language model, effectively preventing hallucination [4] and ensuring factually reliable responses.

F. Frontend Interface Development

The user-facing component was implemented as a web-based interface using HTML and CSS, designed to be clean, intuitive, and accessible to students with no technical background [6] [7]. The dashboard featured a conversational chat layout where students could type questions and receive immediate responses without page reload, through asynchronous JavaScript fetch requests communicating with the FastAPI backend.

G. System Integration and Validation

After all individual components were developed and verified independently, they were integrated into a complete end-to-end system and subjected to comprehensive validation testing [9]. The system was evaluated using simple factual questions, complex multi-topic inquiries, cross-institutional comparisons, and edge cases. The final validated system demonstrated reliable performance across all tested query categories, confirming that the integrated architecture was both technically sound and practically suitable for real-world deployment.

V. Results

The proposed College Enquiry Chatbot was successfully implemented and tested, producing results that validated the effectiveness of the RAG-based architecture [1] [2] in delivering accurate and context-aware responses to student queries.

A. User Interface

The website worked perfectly on all standard browsers. The pages for logging in, signing up, and chatting all functioned as intended [6]. The interface was simple, responsive across different devices, and easy to navigate, confirming that the system was user-friendly and accessible to everyone, regardless of their technical background.

B. Chatbot Response Quality

The chatbot successfully handled a variety of student questions about courses, fees, admissions, hostel facilities, and placement statistics [8] [10]. It delivered accurate and straightforward answers directly grounded in the institutional knowledge base. The system was also able to understand questions phrased in different ways, demonstrating its ability to identify the correct intent behind each query [7]. When students asked comparison questions involving multiple colleges, the system retrieved the appropriate information and provided well-organized comparative answers.

C. Session Management

Each conversation was assigned a unique thread identifier stored persistently in MongoDB [5]. Students could review their full conversation history directly from the dashboard without any data loss, confirming that the session tracking mechanism and database integration were functioning reliably.

D. Response Time and Performance

The system delivered responses quickly and consistently, even when multiple users were accessing it simultaneously [3]. The persistent vector storage in ChromaDB and the asynchronous request handling of FastAPI worked together to ensure fast retrieval and reliable performance, with no noticeable slowdown during concurrent usage testing.

E. Accuracy and Hallucination Prevention

The vast majority of responses correctly matched the content of the original source documents [4]. In situations where the requested information was not available in the knowledge base, the system appropriately acknowledged the limitation rather than generating incorrect answers, confirming that the RAG grounding mechanism effectively prevented hallucination [2].

F. Overall System Validation

The fully integrated system performed reliably across all tested scenarios [9], including simple factual questions, complex multi-topic inquiries, cross-institutional comparisons, and edge cases. The results confirmed that the proposed architecture is accurate, scalable, and practically suitable for real-world deployment in educational institution settings.

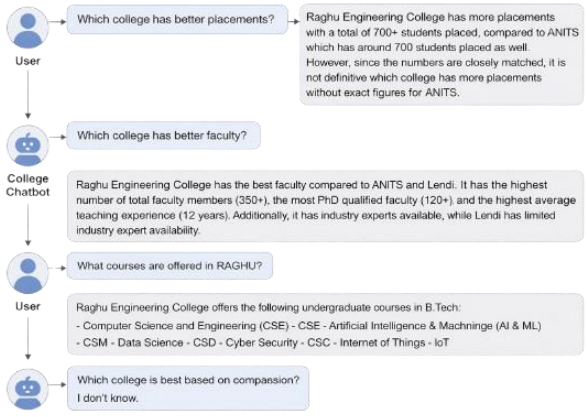


Fig. 2. Sample conversation demonstrating the College Enquiry Chatbot in action across four real student interactions. The chatbot compares placement records between Raghu Engineering College and ANITS, evaluates faculty quality by highlighting Raghu's 350 total faculty members, 120 PhD-qualified staff, and 12 years average teaching experience, lists all available B.Tech programs at Raghu including CSE, AI and ML, Data Science, Cyber Security, and IoT specializations, and honestly acknowledges insufficient data when asked for a definitive college recommendation. The results demonstrate the system's ability to deliver accurate, context-aware responses while avoiding speculative or fabricated answers when information is unavailable.

VI. Conclusion and Future Work

This study presents a new approach to handling student enquiries in colleges, using a chatbot powered by artificial intelligence [6] [8]. The traditional way of dealing with enquiries has its limitations, and this new system aims to overcome them. By combining several advanced technologies, including natural language processing and a large language model [11], the chatbot can provide accurate and helpful responses to students' questions. The system is designed to be scalable, meaning it can handle a large volume of enquiries from multiple colleges and on various topics [3].

The pipeline using RAG was the key to the system's success [1] [2]. By basing every response on factual information from the institution, it made sure the answers were always accurate and relevant to the context. This also stopped the problem of "hallucinations" that can happen in systems that only generate text [4]. The system's ability to filter metadata, retrieve information based on Maximum Marginal Relevance, store vectors in ChromaDB, and use a FastAPI backend that can handle multiple users at the same time [5], all worked together to make it fast, reliable, and able to handle many users simultaneously. The web-based interface made it easy for all students to use, regardless of their technical skills [7], and it provided a much better user experience than other rule-based systems [9].

In terms of future work, several enhancements can be pursued. Voice-based interaction and multilingual support can be incorporated to broaden accessibility [10]. The knowledge base can be expanded to cover more institutions and updated regularly through integration with live institutional databases and official APIs. Personalization features based on individual student profiles and interaction histories can further improve response relevance. Continuous fine-tuning of retrieval parameters based on user feedback will ensure sustained improvements in system accuracy and performance over time [1].

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Ku¨ttler, M. Lewis, W. Yih, T. Rockta¨schel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[2] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[3] S. Gupta, R. Ranjan, and S. N. Singh, "A comprehensive survey of retrieval-augmented generation (RAG): Evolution, current landscape and future directions," *arXiv preprint arXiv:2410.12837*, 2024.

[4] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, pp. 1–38, 2023.

[5] A. Neumann, Y. Yin, S. Sowe, S. Decker, and M. Jarke, "An LLM-driven chatbot in higher education for databases and information systems," *IEEE Transactions on Education*, vol. 68, pp. 103–116, 2025.

[6] H. Gawade, V. Patil, P. Vishe, and S. Kolpe, "College enquiry chat-bot system," *International Journal of Engineering Research and Technology (IJERT)*, vol. 9, no. 9, September 2020.

[7] K. A. Karuna and R. Ramakrishnan, "Intelligent college enquiry chatbot using NLP for enhanced student interaction," *Journal of Advanced Research Engineering and Technology (JARET)*, vol. 3, no. 2, pp. 30–37, 2024.

[8] G. Sambhe, S. Awaze, S. Bobade, S. Bhagwatkar, and P. Bhoyar, "A review on college enquiry chatbot," in *Proc. IEEE*, 2023. DOI: 10.1109/ICCCE57501.2023.10063053.

[9] T. Lalwani, S. Bhalotia, A. Pal, V. Rathod, and S. Bisen, "Implementation of a chatbot system using AI and NLP," *International Journal of Innovative Science and Research Technology (IJISRT)*, p. 522, 2019.

[10] W. H. Elashmawi, S. E. Elbohy, M. Rafik, A. Ashraf, S. Gorgui, M. Emil, K. Ali, "An interactive chatbot for college enquiry," *Journal of Computing and Communication*, vol. 2, no. 1, 2023.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.