Genova: Solving Student Helpdesk Challenges through RAG and Agent Collaboration

Suneha Datta[1], Suhita Kar [1], Shruti Suman [1], Amrit Raj Thakur[1]
Saikat Dutt[1], and Arindam Ray[2]

[1] Department of Computer Science and Engineering,
Institute of Engineering and Management, Kolkata, West Bengal, India
suneha2003datta@gmail.com, suhitakar2003@gmail.com,
shrutisuman61@gmail.com, amritrajt15@gmail.com, saikat.dutt@iem.edu.in
[2] Information Systems and Supply Chain Management,
University of North Carolina Greensboro , USA
arindamray@uncg.edu

Abstract. In the complex and data-heavy field of managing education systems, the classic sequential processing of student admissions and financial aid evaluations is inherently constrained by its non-scalability, human-error-prone manual interventions and less-than-optimal communication channels. To overcome these system-level constraints, we introduce GENOVA: An Agentic Autonomous Helpdesk System, designed through modular multi-agent synchronization and hybrid retrieval generation pipelines. GENOVA realizes a multi-agent asynchronous architecture where independent agents are discretely optimized for responsibility-specific tasks, each controlled by formal state machines and communicating through asynchronous API calls to have consistent session identities throughout transactions. The system automates the entire admission pipeline, including application intake, document verification, candidate shortlisting, student communication, loan query resolution, and the generation of admission letters and fee slips. The framework employs modular autonomous agents that interact with structured data and static documents, such as admission guidelines, eligibility criteria, and fee structures—to drive independent decision-making. This paper provides an implemented solution to two key challenges: integrating organizational knowledge base with the external data sources, and independently generating a series of actions based on the decisions using the available knowledge bases. Semantic drift in user requests is mitigated by a context-aware RAG that retrieves institutional policy fragments, while agent-level fallback and eligibility checks resolve ambiguity and missing information. GENOVA therefore brings forth a semantically consistent agentic system, pushing the boundary of autonomous decision-support in educational institutions. By minimizing manual intervention, the proposed framework enhances operational efficiency, improves applicant experience, and promotes transparency across the admissions lifecycle.

Keywords: Automated helpdesk, Student admissions, autonomous agents, Retrieval Augmented Generation, Real-time monitoring

1 Introduction

In most colleges and universities, the moment an application lands in an inbox, a whirlwind in miniature begins: someone must gather and sort student data, another must sort test scores and transcripts, another must decide if the applicant is qualified by rules of eligibility, and so on—emails back and forth, financial aid forms, and, finally, the holy grail of approval to admit. For years, this has been accomplished by cross-departmental staff teams, all utilizing spreadsheets, checklists, and handoffs. No surprise, this ad-hoc process tends to prolong timelines, introduces errors, and keeps potential students waiting—sometimes with no apparent outcome—longer than they'd like. [3]

Seeing how painfully slow and error-prone this is, we went out and rethought the process from the ground up. What if, rather than papers going from desk to desk between people, we had intelligent "agents" to do the work? That is the concept behind our Agent-Based Automation Framework. Essentially, the system delegates some work—document checks, eligibility, query responses, even loan processing—to specialized software agents. The agents learn to work together, passing work from one to the next smoothly, so that no single department is a bottleneck.

To deal with dirty, disorganized inputs like free-text questions or scanned PDFs, we use OpenAI's [21] large language models [6]. With properly crafted prompts, every agent can pull the right information out, flag missing pages, identify inconsistent signatures, or interpret a throwaway "Hey, how's my application coming along?" as an exact database query. It's like the presence of a virtual admissions counselor who never gets tired or misfiles a document.

Under these AI assistants, two databases work together. A Chroma vector store [8] enables agents to do meaning-based queries across thousands of pages so they retrieve the most useful passages right away without rooting around in keywords. For things that require strict rules—checking GPA cut-offs or computing aid awards in real time—we employ SQLite [30], a small but solid relational database. This two-barreled strategy keeps the system agile: clever where subtlety serves, and solid as a rock where accuracy is not open for argument.

Structured rule-based knowledge, including numeric eligibility criteria and relational database records, institution's operational unstructured documents, such as policy documents, standard operating procedures, historical correspondence, and frequently asked questions (FAQs) are combined into a single institutional knowledge base in form of pdfs.. This knowledge base is held in the Chroma vector store, so that semantic retrieval of relevant fragments can be done against user input or agent tasks. For example, when the Eligibility Check Agent makes a determination of applicant eligibility, it does not base its determination on static rules alone. Rather, it does retrieval of policy snippets—program-specific rules, e.g., or edge-case exceptions—so that it can make informed choices based on official precedent. Retrieval-augmented decision-making allows the agents to adhere not only to logic, but to historical and procedural knowledge in an institution's documents.

And of course, clever agents don't just materialize out of nowhere. They need carefully tuned prompts to guide their reasoning. To take an example, the Document Verification Agent relies on language that makes it pay attention when transcripts are missing a page, and the Query Agent [15] relies on prompts that make it learn to recognize standard student questions and map them onto correct, executable database queries.

The whole configuration is modular and organized. Every agent operates in isolation but contributes information to the remainder of the system, so parallel processing is an option and there is no point of failure. An administrator takes a bird's-eye view of agent activity through a real-time dashboard—with alert if anything out of the ordinary happens—so only human staff intervene when necessary.

In our review of the literature, we compare old rule-based pipelines to more recent AI approaches, noting where earlier solutions have hit walls in terms of adaptability or scalability. Our evaluation makes clear that, through the combination of prompt-engineered language models [6] with smart database design, we can enhance accuracy, cut down response times, and ease the burden for all concerned. There's ample opportunity in the future to expand: to extend to national education hubs, provide multilingual support for global students, and add advanced analytics to provide admissions teams with more insights. Beyond admissions, the same platform could fuel course enrollment, alumni engagement, or even exam registration—wherever schools want robust, adaptable automation instead of yet another spreadsheet headache [14].

2 Literature Review

In the past, systems belonging to the help desk and admission supporting technologies were either rule-based systems or relied on human guidance. Therefore, even though these systems may work perfectly in well-organized and structured environments, they often rely on well-structured interactions and intuition, which may make them less adaptable to dynamic and unstructured admission situations. Such dependencies have encouraged educational technologies toward providing automation services in university settings. Such services include conversational AI systems, Retrieval-Augmented Generation, and smart agents, as described in [3,14].

A significant volume of literature also attempts to optimize the level of personalization and contextualization in student-oriented platforms using RAG pipelines. For example, context-logging RAG architectures are aimed at improving contextualization, and their replies are precise and relevant as they are obtained through interactions with a context [9]. Similarly, policy-based RAG architectures are aimed at intervening in both the ingestion of external information and the internal reasoning of the language model in order to optimize costs while ensuring the quality of responses in various institutions [28]. They are primarily characterized by accurate and precise responses, as well as a high level of trust

and minimized hallucinations. They are, however, largely tied to platforms and lack the powers of decision-making.

Yet another area of research interest lies in exploring the features of conversational interfaces in a manner that facilitates an improvement in workflow efficiency in educational helpdesks. In this regard, various studies have presented a model based upon NLP technology that facilitated effective chatbots in addressing multilingual student concerns in a high-volume scenario [2]. On the flip side, there exists the field of process mining, wherein institutional event logs are used to identify inefficiencies in processes in academia [10]. Although such approaches facilitate a student in a particular manner and provide relevant insights in operational practices, they are still not a source of real-time technology.

There have been more studies that examine the incorporation of structured domain-specific information into the grounding of language model outputs. Generally, in RAG systems that utilize information such as domain content in making retrievals more accurate and effective in educational QA applications, there has been a trade-off in which improving the density of retrievals directly correlates with a reduction in fluency in the responses [16]. A different form of RAG that has been presented incorporates the use of graph information in educational and personal knowledge graphs to facilitate more effective and scalable personalized educational applications [1]. These systems, although effective in many applications, are more complicated to develop.

Across these conceptual areas, current systems typically tackle only fragments of educational automation-things like personalization, retrieval efficiency, conversational access, or workflow analysis-without providing a single autonomous framework to tie them together. Most solutions are tailored to a particular institution, rely on static setups, and demand continual human upkeep for maintenance and error handling. Furthermore, genuine agentic capabilities-such as context-free reasoning, adaptable task coordination, and robust failure recovery-are still not well represented in the existing literature.

Motivated by the limitations recognized above, the current project seeks to move the discussion forward by offering a flexible and agent-driven RAG[17] approach for the automation of institutional admissions. This approach, which incorporates elements such as autonomous agents, context awareness, scalable storage of vectors, reliable fallbacks, and decision logic with knowledge of its own state, can function effectively even without context and can do so in a variety of institutions—unlike other methods, which are institution-specific and have numerous limitations as recognized in earlier studies.

3 System Architecture

The architecture of Genova is built leveraging a modular structure, where it follows a multi-layered and distributed model-view-controller approach [6]. The whole system is divided into two separate parts, the client presentation layer, which provides an interface and a controller to capture the user queries, and using a RESTful API, sends it to a stateless server-side application layer for

further processing [11]. The backend uses an agentic workflow [7] to determine whether to retrieve data from vector embeddings or to process it for a structured schema. The sections below describe the structure in detail.

3.1 Client Presentation Layer

On the client side, a web view is built as an interface using the Streamlit [31] framework as a conversational system. This layer accepts both text queries as a string and any documents uploaded in the form of PDFs, `.doc`, or `.docx` files. A session state memory maintains the conversational state in this layer that stores all the responses from the server-side application layer and the client-side text prompts. Any input query with the multipart document fed to this layer is stored within this `st.session_state` and also sent to a stateless server-side endpoint as a HTTP POST request [26]. The JSON response received is converted to a string and appended to the session state as well as being displayed to the client.

3.2 Relational Data Model

A SQLite relational database is built into the system to support a schema for structured data, which is modeled using SQLAlchemy's object relational model (ORM) [5] framework. Two model classes, named Students and Loan, are defined for processing transactional operations and query filtering of student details, and to streamline the admission and loan approval pipeline. The Students model class is designed to store students' academic attributes that capture all relevant information needed for eligibility analysis as well as admission shortlisting. This model helps to update student information in a real-time session. Another model class, Loan, is created for updating loan applications.It models financial parameters such as applicant income, sanctioned funding amounts, installment schedules, and bank interest rates. The schema is well normalized [29], ensuring comprehensive data representation.

3.3 Server-side Application Layer

The application layer exposes a stateless API endpoint, which serves as the main orchestrator agent and it receives a string query prompt and another multipart file [26]. In the presence of a multipart file, a call to one of the defined agents A2 is triggered within this application layer; otherwise, the prompt member of the container object is analysed using OpenAI's function calling tool [20]. Three additional logic agents are defined separately within this application layer as A1, A3, and A4. The large language model GPT-4o-mini [21] interprets which agent to trigger out of A1, A3, A4, operating as an autonomous agentic system, analysing the Natural Language, seeking any institutional work described within the prompt by mapping it to a predefined function registry [20]. The outputs of the four functions are structured into a JSON data structure and sent to the client as a POST request payload.

Inside the A2 agent, the multipart file that contains the academic qualifications and details of any student is analysed, and using the PyPDF2 module [24], any PDF file content is read, or using the Textract module, any `.doc` file is read into a temporary file. Our system uses a knowledge base which is created using institutional documents that encompass student support systems, academic curricula, campus infrastructure, and admission criteria. It acts like a structured as readable as well as appendable file system storing institution-specific educational and co-curricular details. The institutional documents are split into chunks of 1000 tokens with an overlap of 200 tokens and embedded as vector embeddings using the `text-embedding-3-small` model [22]. These vectors are indexed and stored in a Chroma vector database [8]. A retrieval QA chain is built with LangChain's chain module that performs semantic similarity search of the embedded prompt [15] to the vector database and retrieves three primary matched sequences, which are used by an internal large language model within the retrieval QA chain to generate the final answer from the three sequences. The prompt received in the A2 agent is queried through this retrieval QA chain as Retrieval Augmented Generation (RAG) [17]. The response tailored to the institutional details along with the content stored within the temporary file is converted to a structured System Prompt using Prompt Engineering and is sent as a JSON post to the OpenAI API for the GPT-4o-mini [21] large language model to decide whether the content of documents adheres to the eligibility criteria set by the institution for the prompt query.

The A1 agent is responsible for handling any general queries seeking information about the various departments of the institution and institutional guidelines. The prompt is converted to an embedding into the same latent vector space as the Chroma database [22]. This vector is semantically searched to match the top three relevant embeddings within the database using cosine similarity search [18]. The GPT-4o-mini large language model within the retrieval QA chain as Retrieval Augmented System [17] processes and concatenates the top embeddings and generates a structured template response for output [15].

A3 agent is defined for shortlisting present applicants of the institution. The applicant's details are stored in the relational database described above following a predefined schema. Before invoking this agent, the OpenAI's function calling tool captures the program and the stream applied by the student interpreting the prompt and passes them as arguments [20]. Inside this agent, a query is run to fetch the candidate details from the SQLAlchemy ORM model Students in the present database session [5]. The fetched records are validated against the institutional program as well as stream-specific thresholds of ranks and percentages. The candidates who meet these criteria are immediately sent a shortlisted applicant email notification, where the content is dynamically generated for each program, specifically leveraging a Yagmail client [34]. The agent outputs a string response to the function calling tool after successful workflow pipeline completion.

The OpenAI function calling tool [20], whenever encompasses any loan-related or financial query in the prompt, triggers the A4 agent for execution.

It tries to interpret the presence of the student identity number attribute, the Bank name attribute, and the Amount of money attribute in the prompt and passes them as arguments to the A4 agent. This agent initially validates whether the attributes have a null value and autonomously outputs a string response asking the candidate to provide their identity as well as details in case of missing information. In the presence of the student's identity number, the agent further processes the loan details. It retrieves the loan application records corresponding to the student identity number from the SQLAlchemy ORM model Loan [5]. It executes a sequence of three rule-based checks. From the records of the Loan schema, it is assessed if the student has a previously submitted loan application, an accepted loan application, or wants to put up a loan requirement. The new applicants, upon successful verification, are added to the active SQLAlchemy session as a new instance of the Loan model. The previously submitted records are checked against institutional thresholds of financial credits and are communicated an acknowledgement through a Yagmail client [34]. Agent A4 thus encapsulates the internal financial logic and serves in automating the loan process.

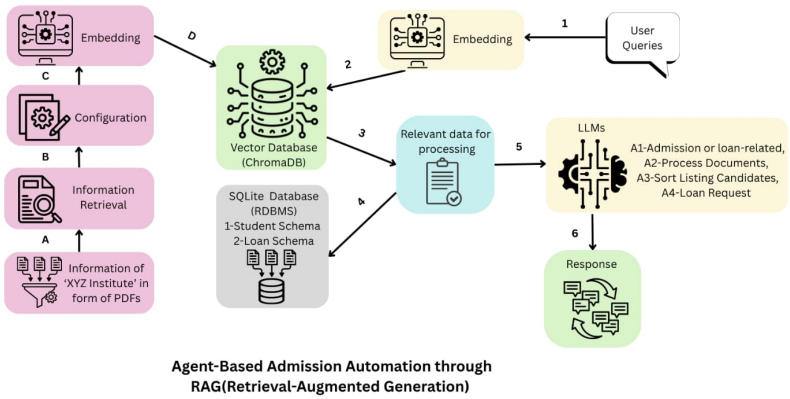


Fig. 1. Dataflow diagram

4 Evaluation and Performance

This section demonstrates the intermediate results of the various test cases run on the system for evaluation, as well as the achieved efficiency. The following images depict the pipeline followed by each of the agents to process a particular

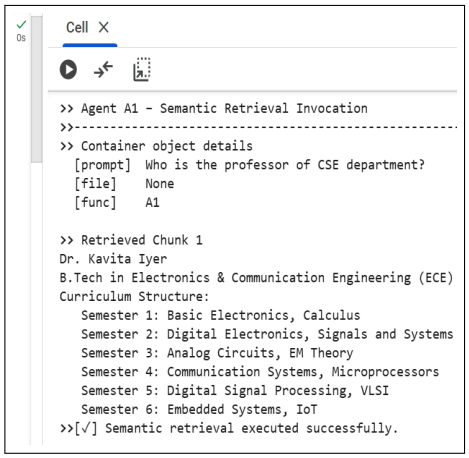


Fig. 2. Agent A1: Vector Search Output

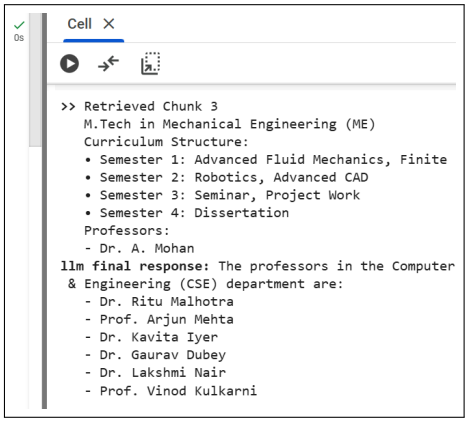


Fig. 3. Agent A1: Final LLM Response

query. Figure 2 demonstrates the outputs received in a test editor when Agent A1 was invoked by the system to process the query seeking the details about the professors of the institution. The system correctly mapped the query to agent A1 and retrieved the top three chunks of documents matching the query from the vector database. The large language model finally combined all those chunks to produce a relevant and comprehensive result answering the query.

Figures 4 and 5 showcase how a given document (e.g., `myjee.pdf`) was processed by Agent A2, and the corresponding details about the query of eligibility in a particular department were retrieved from the Chroma vector database [8]. The large language model, in the end, not only concatenated the retrieved embeddings but also produced a responsive result stating the eligibility of the student for that department.

Figure 6 shows how our system correctly mapped the query for shortlisting candidates of a specific department to Agent A3. It also queried the relational database to find relevant student records in that department and performed dynamic email generation for all matching candidates. Alongside that, the figure also demonstrates the output of the loan application handling pipeline. Agent A4 was mapped with the query for a new loan application of a candidate, with proper arguments sent by the OpenAI function calling tool [20]. The system performs rule-based checks and adds a new record in the Loan relational schema by retrieving additional student details from the Students relational schema.

The system was comprehensively tested against its response accuracy, consistency of the agent with the task, and efficiency of overall query handling. Our testing involved 135 real-world test cases, aimed to determine how reliably each agent could handle domain-specific queries regarding the admission of students and financial aid through Retrieval-Augmented Generation (RAG) [17] as well as rule-based logic. All 4 agents were tested with questions handcrafted to mimic actual admission and loan support situations, such as document-based

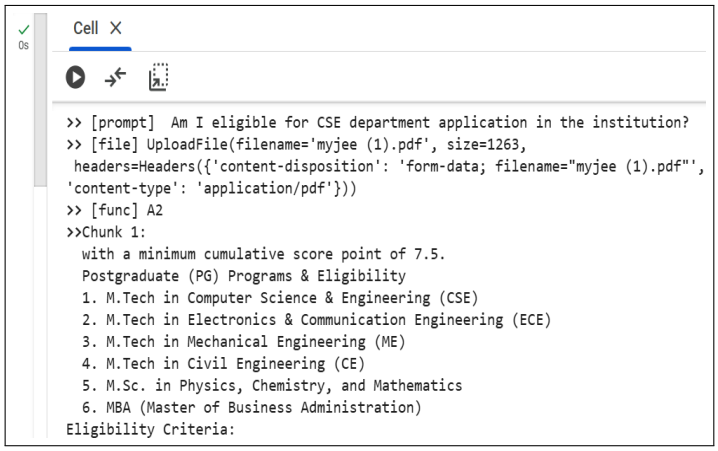


Fig. 4. Agent A2: Retrieved Documents

and multi-step questions. 135 queries were tested by running them through the system's front-end via the Streamlit interface, and backend interactions were traced and verified via API logs.

Agent A1 was tested with 25 questions based on academic programs, faculty information, deadlines, and general advice on the admission process. All questions were correctly addressed with accurate and contextually appropriate responses generated through ChromaDB [8]-based retrieval and OpenAI [21] generation. This gives an accuracy of 100% for Agent A1.

Agent A2 authenticated uploaded documents such as mark sheets, ID proofs, income, and caste certificates. It flagged missing or incorrect documents in 36 out of 37 cases, providing it with an accuracy of 97.29%. The failure was for one particular document, which was not related to any academic details, but was tested to check proper content understanding by the large language model. In this single case, the model tried to relate the content with institutional information retrieved from the vector embeddings instead of discriminating against its weak association.

Agent A3 was tested on candidate shortlisting queries on the basis of academic marks, rank in exams, reservation, and branch preference. Agent A3 performed flawlessly, analyzing and responding to 28 queries out of 30. That makes it 93.33% accurate, demonstrating the strength of its rule-based logic. In both failed cases, a student record had a missing rank in the relational database, and the system's rule-based checking raised an exception. Although after complete evaluation, this logic was further rectified with a rule check modification to ensure coherence with the stated edge case.

Agent A4 was assessed with loan application queries and loan eligibility testing; this agent correctly handled 39 out of 43 cases with 90.69% accuracy. This agent was unable to capture ambiguous cross-talks in prompts when it asked for both loan-related as well as admission-related topics in a single prompt. In

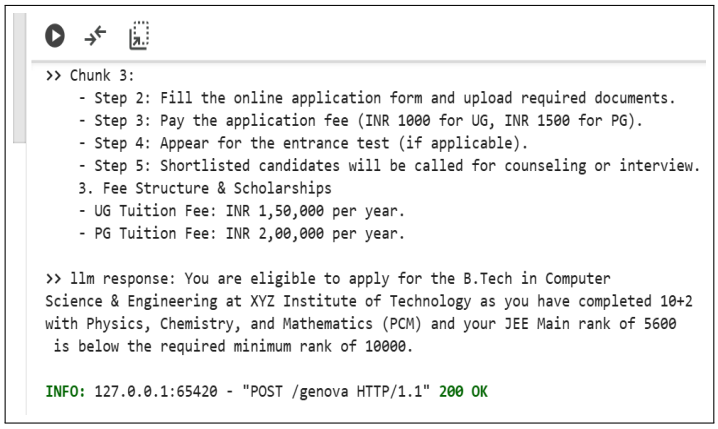
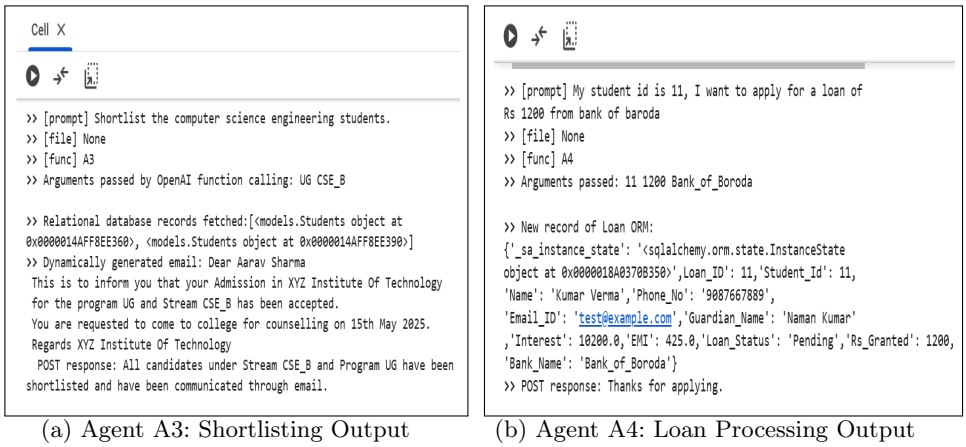


Fig. 5. Agent A2: LLM Final Response



(a) Agent A3: Shortlisting Output (b) Agent A4: Loan Processing Output

Fig. 6. Side-by-side outputs from Agent A3 (a) and Agent A4 (b)

all those 4 failed cases, the function calling of OpenAI deliberately mapped the prompt to Agent 3 instead of Agent 4. With the help of continuous prompt engineering of the intermediate system prompts used in the system before OpenAI's function calling [20], it was able to distinguish the actual mapping, and in the further test cases, recognized Agent 3 accurately.

Our system has excellent structured query processing as well as admission logic interpretation. Correct response grounding through Retrieval-Augmented Generation (RAG) [17] reduces hallucination, modular agent scheduling enables effective parallel task management, and real-time database interaction with SQLite [30] ensures information retrieval in real-time. Also, shortlisting and communication agents exhibited end-to-end automation ability. Also, we could see that the guardrails are properly maintained in this agentic framework.

S.No	Agent	Queries Tested	Correct Responses	Accuracy (%)
1	Agent A1	25	25	100.00
2	Agent A2	37	36	97.29
3	Agent A3	30	28	93.33
4	Agent A4	43	39	90.69
5	**Total**	**135**	**128**	**94.81**

Table 1. Performance Evaluation of Agents

We have the visibility and control over the agent's action, leading to responsible AI methodology. Overall, GENOVA shows strong performance in its key functional areas, justifying its role as a scalable, intelligent helpdesk solution for higher education institutions. As the system can be further scaled up, based on real-world usage and ongoing learning pipelines, it can even be refined to a greater accuracy and utility.

5 Future Work

Though GENOVA has shown robust performance as an automated student helpdesk system, there are a number of directions in which its capabilities could be extended. In the next versions, we intend to add voice interactions and multilingual support features to make it feasible for diverse users, such as disabled users or those who speak limited English. Integration with institutional ERP systems and live databases will also allow GENOVA to retrieve up-to-date data like seat availability, fee information, and loan approval status.

Another direction that has potential includes a learning feedback cycle where user interactions are used to refine agent responses in the long run through reinforcement learning [32] or LLM supervised fine-tuning [35]. Implementing feedback mechanisms to refine the system performance and adapt to changing student needs also holds significant promise for this system.

In addition, we plan to introduce proactive reminders for important deadlines or missing documents through SMS APIs. By prioritizing responsible AI principles [12] and continually improving the system, we can ensure high-quality support to students while maintaining their trust and confidence. Lastly, extending GENOVA to a generalized AI assistant for other university processes like hostel allotment, grievance redressal, and course registration can turn it into an end-to-end digital student companion.

6 Conclusion

The present work lays out an automated system with the objective of reducing the procedures of student admissions and education loan management to streamlined processes. The system is based on agent-based architectures [27] with prompt engineering models [21]. The functional architecture of the system consists of four autonomous agents, each being assigned some tasks varying

from communication, document verification, eligibility check, and loan application processing.

Based on Natural Language Processing (NLP) [19], the system is able to process unstructured inputs in the form of scanned documents and open-ended questions from students successfully. The system successfully processes user-inputted data and gives instant feedback. Unstructured data is structured into a machine-readable form by the application of prompt-based AI agents, thus reducing the time taken significantly and the volume of errors due to manual processing. The improved architecture of the system offers ease of accommodation to varied institutional policies and academic profiles.

Future extensions could include the addition of multilingual prompt interpretation to accommodate applicants from varied linguistic backgrounds, in addition to the integration of external academic APIs for checking academic records in real-time. A student dashboard dedicated to the purpose will provide real-time information on admission and loan status, thus improving the level of transparency and offering a hassle-free experience.

Essentially, the student dashboard demonstrates the power of a user-focused, modular strategy, coupled with artificial intelligence agents [27], to transform educational administration and build sustainable automation in the education sector.

References

1. Abdelmagied, M., Chatti, M.A., Joarder, S., Ain, Q.U., Alatrash, R. (2025). Leveraging Graph Retrieval-Augmented Generation to Support Learners' Understanding of Knowledge Concepts in MOOCs. European MOOCs Stakeholders Summit.
2. Alabbas, S., Alomar, S.: Tayseer: A Novel AI-Powered Arabic Chatbot Framework for Technical and Vocational Student Helpdesk Services and Enhancing Student Interactions. Procedia Computer Science **181**, 881–888 (2024)
3. AlBlooshi, M. et al. Artificial intelligence in higher education: opportunities and challenges. Frontiers in Education, vol. 10 (2025).
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. 2nd edn. Addison-Wesley, Boston, MA (2003)
5. Bayer, M.: SQLAlchemy 2.0 Documentation (2024). `https://docs.sqlalchemy.org`
6. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language Models are Few-Shot Learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020)
7. Caetano, A., Verma, K., Taheri, A., Kumaran, R., Chen, Z., Chen, J., Höllerer, T., Sra, M. (2025). Agentic Workflows for Conversational Human-AI Interaction Design. ArXiv, abs/2501.18002.
8. ChromaDB Documentation: Chroma: AI-native Embedding Database. `https://docs.trychroma.com`
9. Cohn, C., Rayala, S., Snyder, C., Fonteles, J-H., Jain, S., Mohammed, N., Timalsina, U., Burriss, S., Ashwin, T.S., Srivastava, N., Biswas, G. Personalizing Student-Agent Interactions Using Log-Contextualized Retrieval-Augmented Generation (RAG). Retrieved from `https://par.nsf.gov/biblio/10650744`

10. Dzihni, A., Andreswari, R., Hasibuan, Z.A.: Business Process Analysis and Academic Information System Audit of Helpdesk Application Using Genetic Algorithms: A Process Mining Approach. Procedia Computer Science **161**, 705–712 (2019)
11. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
12. Floridi, L., Cowls, J.: A Unified Framework of Five Principles for AI in Society. Harvard Data Science Review (2019)
13. Goellner, S., Tropmann-Frick, M., Brumen, B. (2024). Responsible Artificial Intelligence: A Structured Literature Review. ArXiv, abs/2403.06910.
14. Kalam, A., Kaur, Dr. M. (2025). AI in Higher Education: Transforming Admissions, Student Advising, and Research Support. International Journal for Research in Applied Science and Engineering Technology, 13(5), 6915–6922. https://doi.org/10.22214/ijraset.2025.71728
15. LangChain Documentation: LangChain: Building Applications with LLMs through Composability. https://docs.langchain.com
16. Levonian, Z., Li, C., Zhu, W., Gade, A., Henkel, O., Postle, M., Xing, W. (2023). Retrieval-augmented Generation to Improve Math Question-Answering: Trade-offs Between Groundedness and Human Preference. ArXiv, abs/2310.03184.
17. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20), Article 793, 9459–9474.
18. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK (2008)
19. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK (2008)
20. OpenAI: Function Calling in GPT Models. https://platform.openai.com/docs/guides/function-calling
21. OpenAI: GPT-4 Technical Report (2023). https://openai.com/research/gpt-4
22. OpenAI: Text Embedding Models. https://platform.openai.com/docs/guides/embeddings
23. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L. E., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., Lowe, R. J. (2022). Training language models to follow instructions with human feedback. ArXiv.
24. PyPDF2 Documentation. https://pypdf2.readthedocs.io/en/latest/
25. Rahwan, I., Cebrian, M., Obradovich, N., Bongard, J., Bonnefon, J.F., Breazeal, C., et al.: Machine behaviour. Nature **568**, 477–486 (2019)
26. Ramírez, S.: FastAPI Documentation. https://fastapi.tiangolo.com
27. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 4th edn. Pearson Education, USA (2020)
28. Saha, B., Saha, U. (2024). Enhancing International Graduate Student Experience through AI-Driven Support Systems: A LLM and RAG-Based Approach. 2024 International Conference on Data Science and Its Applications (ICoDSA), 300–304.
29. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts. 6th edn. McGraw-Hill, New York, NY (2010)
30. SQLite Documentation. https://sqlite.org
31. Streamlit Inc.: Streamlit Documentation. https://docs.streamlit.io

32. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. 2nd edn. MIT Press, Cambridge, MA (2018)
33. Wooldridge, M.: An Introduction to MultiAgent Systems. 2nd edn. Wiley, Chichester, UK (2009)
34. Yagmail Documentation. https://yagmail.readthedocs.io
35. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., et al. (2023). A Survey of Large Language Models. https://doi.org/10.48550/arXiv.2303.18223