WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

MADADGAR: A Retrieval-Augmented Generation (RAG) using Spring AI for College Specific Information

Dr Smita[1], Dev Singhal[2], Harsh Vardhan Yadav[3]

[1,2,3]CSE Galgotias University Uttar Pradesh, India

ABSTRACT

University students often struggle with day-to-day issues such as navigating learning management systems (LMS), managing academic tasks, and staying updated on university events. While help desks have been a traditional solution, they often lack the efficiency and accessibility students require in their fast-paced lives. To address these challenges, we are developing an automated solution in the form of a mobile/web application powered by an AI model tailored specifically for university students. This plat- form is designed to streamline university services and provide personalized support directly on students' devices. What sets our solution apart is its exclusivity for individual universities, ensuring alignment with each institution's unique systems, services, and requirements. By centralizing and digitizing univer- sity support, our application enables students to efficiently access LMS tools, receive timely reminders for assignments, stay informed about events, and find solutions to common queries—all in one place. As university students ourselves, we deeply understand the frustrations of managing academic responsibili- ties amidst the complexities of existing university systems. This firsthand experience has motivated us to create a solution that simplifies university life, empowering students to focus on learning instead of logis- tical challenges. Our goal is to enhance efficiency, improve communication, and ensure that no student feels overwhelmed or lost in the process. This platform is more than just a tool; it's a comprehensive solution designed to make university life smoother, more manageable, and accessible for every student.

1. Introduction

University life presents students with a variety of challenges, ranging from managing academic responsi- bilities to staying updated on institutional services and events. Navigating complex systems like Learning Management Systems (LMS), receiving timely reminders for assignments, and finding solutions to day-to-day problems often feels overwhelming. While help desks provide support, they can be inefficient and inaccessi- ble at critical moments, leaving students struggling to stay organized and informed. As students ourselves, we have firsthand experience with these issues. The gaps in university services and the lack of seamless communication channels often lead to frustration, confusion, and a loss of productivity. Recognizing these challenges, we have set out to design a comprehensive solution tailored specifically to meet the needs of university students.This research focuses on the development of a mobile/web application powered by an AI-driven model to address these challenges. The platform is built to provide personalized, efficient, and centralized support to students, offering features such as LMS navigation, assignment reminders, event up- dates, and solutions to university-specific queries. By tailoring the application to the unique systems and services of individual universities, we ensure exclusivity and alignment with institutional requirements. Our motivation stems from the desire to empower students, allowing them to focus more on learning and less on logistical hurdles. Through this solution, we aim to enhance the efficiency of university services, streamline communication, and improve the overall student experience.The dataset for this application was collected from institutional websites, administrative APIs, and other publicly available resources, ensuring that the information is both comprehensive and current. The methodology applied includes several key stages: data

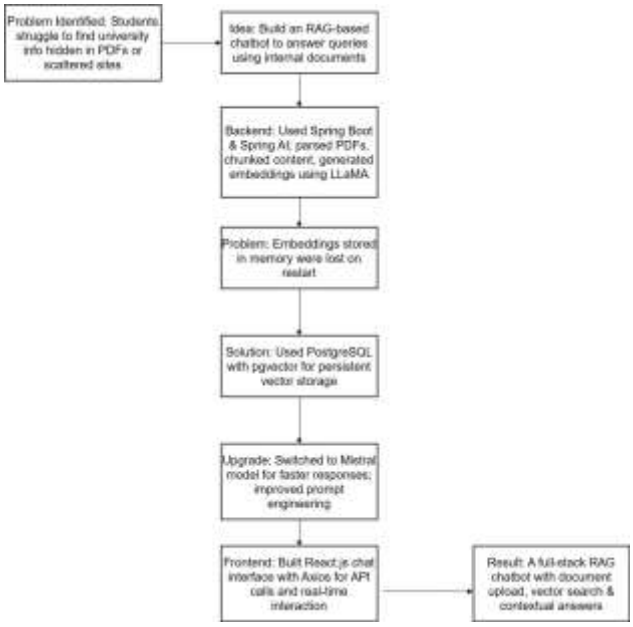

Figure 1: Process of a question-answering system based on context retrieval for LLMs.

collection and preprocessing, model fine-tuning, API development, frontend development, system testing, and deployment. In the model application, a combination of BERT for semantic document retrieval and GPT for generating responses based on student queries was employed. This ensures that the platform pro- vides accurate and contextually relevant answers. Finally, the application was deployed on scalable cloud platforms like AWS, ensuring consistent performance and reliability under variable loads.This paper outlines the development, features, and potential impact of this platform, showcasing how it bridges the gaps in existing university support systems and creates a smoother, more manageable academic environment for students.

This diagram shows the data processing sequence in a system, beginning at the College Database, which supplies raw data, which in turn is processed and stored under ”Our Database.” Processed data is stored in document format and broken down into smaller, more manageable pieces of data, or ”chunks.” These are stored in a Vector Database in the format of vectors, which allows for rapid retrieval of data. When a user query is input, the system will look up the applicable data chunks, which are prepped in a prompt format ready to use. The Chat Model processes the query, using natural language processing, in order to extract the meaning of the input and produce a response. The system then outputs the Chat Response to the user, with the possibility of Manual Verification to validate the data, ending in the ”End of Data Processing” stage. This sequence provides effective and correct data handling and querying.

2. Literature Review

In the study of the creation of a college information chatbot leveraging technologies such as React.js, Spring Boot, Ollama, and DeepSeek LLM, we were supported by a wide range of literature encompassing theoretical frameworks, documentation of tools, practical guides, and real-world knowledge. One of the theoretical foundations we drew upon was by Lewis et al. (2020), where they proposed Retrieval-Augmented Generation (RAG), a method of leveraging retrieval systems along with generative models for enhanced performance on knowledge-intensive NLP tasks. This directly led to the utilization of document-based vector search in order to optimize chatbot responses in our work. The official Spring Boot and React.js documentation offered the necessary guidelines to construct the system’s backend and frontend respectively, providing a sound base for building the application using best practices in component design and API integration. Tom Brown et al.’s (2020) suggestion of GPT-3 and few-shot learning further extended the knowledge of LLMs generalizing across a wide variety of tasks using minimal input, something we applied using local models such as DeepSeek. BERT by Devlin et al. (2019) also acted as a crucial guide to identifying language context and intent handling, crucial in enabling the correct interpretation of user queries by our chatbot.

One practical case study published by the Journal of Educational Technology in 2021 provided contextual knowledge of the needs of academic settings in the real world, informing the types of questions and workflows we developed. IEEE literature on maximizing the retrieval of information systems (2019) and cutting- edge AI retrieval processes (2021) informed the optimization of our search approaches through the use of semantic indexing and hybrid retrieval. We also derived a lot of value from a survey of creating real- time applications using React by the ACM (2020), which informed the responsiveness and optimization of our frontend response time. Multilingual support, emphasized in a 2022 publication by Computational Linguistics Journal, motivated us to think about evolving flexibility in supporting languages of regional areas. Deployment was covered in studies of cloud deployment in AI systems (Journal of Cloud Computing, 2021), along with a security-oriented study by IEEE (2020), both of which informed containerization, deployment, and secure data management through JWT and HTTPS.

The implementation process was heavily informed by a mixture of hands-on tutorials and personal experi- ences. Posts by Alam (2025), on running a chatbot using AI using React and DeepSeek, and by Devs5003 (2025), on using Spring Boot and Ollama, were invaluable in deriving insights directly

from real-world inte- gration scenarios and workarounds. Articles by the Ollama library directly informed us how to use available LLM models, while Hasan (2025) and DevXplaining (2024) tutorials further demystified local deployment of the models along with React and Spring AI. Performance concerns in practice were examined in Hy- land's (2025) comparison of LLM execution speeds, and which informed our decision in selecting different configurations of the models to use locally. Typical developer problems were uncovered by sites like Stack Overflow–Sajjan's post (2024) assisted us in fixing a functional problem in our use of Spring AI, while a thread on Hacker News criticizing LangChain led us to use more minimal and straightforward LLM libraries like the Ollama one.

The larger tech community also contributed in shaping our direction—Reddit forums assisted in pointing us towards best-performing LLMs for programming use-cases, and the ReAct framework of Yao et al. (2022) informed how we organized reasoning and decision-making logic in the chatbot. Official documentation on Spring AI, Ollama, DeepSeek, Next.js, and FastAPI gave us authoritative advice on using different components of our tech stack together. We especially learned a lot about building AI apps from Spring's 2025 blog post, which closely corresponded to our project architecture and goals. At last, articles such as Dev.to's guide on using Ollama with React, and Medium's guide to deploying LLMs using Spring Boot, performed the role of step-by-step companions, allowing for a smoother development and deployment phase.

Collectively, this wide-ranging blend of academic literature, developer documentation, and open-source re- sources not only supported our knowledge of the underlying technologies but also informed the development of every aspect of our chatbot, from the choice of model and vector searching to frontend responsiveness and secure cloud hosting. These resources formed a deep, hands-on knowledge base that enabled us to create a scalable, intelligent solution appropriate to education's use cases.

3. Material and Methodology

3.1 Description

The journey started from a basic premise: students constantly had too many questions about their uni- versities, and the answers tended to lie deep in PDF or fragmented across websites. I wanted to create something that could serve as a wise senior who knows everything, at the touch of a button. That isss when the concept of an RAG-based chat bot was born - something that had the ability to actually hear questions and fetch contextual content directly from internal docs.Initially, I had selected Spring AI as the backend due to its robust integration capabilities and being already familiar with the Spring ecosystem. I initially scaled down. I applied simple PDF parsing and scraped the content, chunked it into meaningful sections, and created embeddings of every chunk utilizing a local model. For the AI model, I began with the use of Ollama's LLaMA 3.2. It was like magic the first time it gave a simple response. Although it was slow and at times verbose, it was thrilling enough to watch it bring back the correct answers given the chunks I put in front of it.At that time, I was keeping these embeddings in a basic in-memory vector store, which was fine for testing but of course not feasible in the long term or any production use. Whenever the backend was restarted, the embeddings were gone — a chatbot suffering from amnesia. I realized the value of persistent storage of the embeddings if ever I wanted to scale this or achieve consistent performance. That was the moment I began investigating PostgreSQL using the pgvector extension and added it to my Spring backend. With pgAdmin, I could now observe my embeddings stored in the correct manner, and vector similarity searches being reliable and far more scalable. In the same period, I also did a major model changeover. Although LLaMA 3.2 was passable, it wasn't the best when it came to speed, particularly for a real-time conversation experience. So, I upgraded it to the use of Ollama's Mistral model — it was faster and more responsive while not sacrificing too much in the way of accuracy. Prompt engineering also improved at this time — I streamlined the passing of context to the model so that it remained pertinent, targeted, and under token constraints.After the backend was stable and intelligent, I turned my attention to creating the frontend in React.js. It had to feel like a chat app that students actually wanted to use. I designed a simple user interface where students could input questions, view their conversation history, and receive answers in a matter of milliseconds. In the background, I configured Axios to communicate with the Spring back end, managed the load states, and kept the messages looking and feeling natural and conver- sational. It wasn't about showing text — it was about creating the entire thing to feel like a conversation that was actually happening.One of the best moments was watching it develop. In the beginning, it was just about getting back pieces of PDF content. Gradually, however, as I refined the embedding search code, enhanced the chunking algorithm, and provided better prompts, the responses began to feel more natural and less mechanical.The development of the MADADGAR system was meticulously executed across four well-structured quarters to ensure a methodical and impactful delivery of an AI-powered solution tailored specifically for university students. In Q1, the focus was on identifying prevalent student issues through needs assessment, conducting an in-depth literature review on Retrieval-Augmented Generation (RAG) and large language models (LLMs), and selecting a future-proof technology stack—comprising Spring AI for scalable backend services, React.js for dynamic frontend development, and vector databases for efficient semantic retrieval. Functional and non-functional requirements were outlined, and a modular system architecture was designed to support extensibility and maintainability. Q2 emphasized data collection from official university websites, LMS platforms, and academic reports, followed by preprocessing through tokenization, metadata embedding, normalization, and vectorization. A core RAG pipeline was developed, integrating document retrieval and natural language response generation using Ollama to ensure accurate, context-aware outputs for college-specific queries. In Q3, the frontend was implemented with a React.js interface enabling real- time user interaction, seamlessly integrated with the backend using RESTful APIs. Rigorous functional testing was conducted to validate frontend-backend communication and output reliability. Q4 concentrated on comprehensive system testing, performance optimization to reduce latency and enhance response preci- sion, and stress testing to assess scalability. Simultaneously, a research paper was finalized, detailing the system's architecture, methodology, data sources, preprocessing techniques, evaluation metrics, comparative performance analysis, and implications for future academic AI tools.

3.2 Proposed System Architecture

The design of the envisaged Retrieval-Augmented Generation (RAG) system resulted from a step-by-step it- erative development approach aimed at creating a smart, reactive assistant specific to college-related queries. On the first day, the aim was to create a system that could grasp student queries and deliver authentic an- swers through the integration of real-time document retrieval and generative AI.At its core, there's a strong Spring AI-powered backend, which is the brain of the operation. We started using a basic vector database to store and fetch the embeddings at the beginning. But as we scaled and the need increased to have improved performance and handling of queries, we switched to using PostgresQL (PgAdmin) as a vector database. This helped us store the embeddings in a more efficient manner and fetch them faster and with more defined queries.The pipeline of ingesting documents begins by gathering data from a variety of sources university portals, course catalogs, events calendars, etc. These PDF files were parsed by using libraries such as Apache PDFBox, where we extracted and cleaned text and then chunked it out in smaller, meaningful pieces. These pieces were infused with metadata in the form of document type, category, and timestamp,

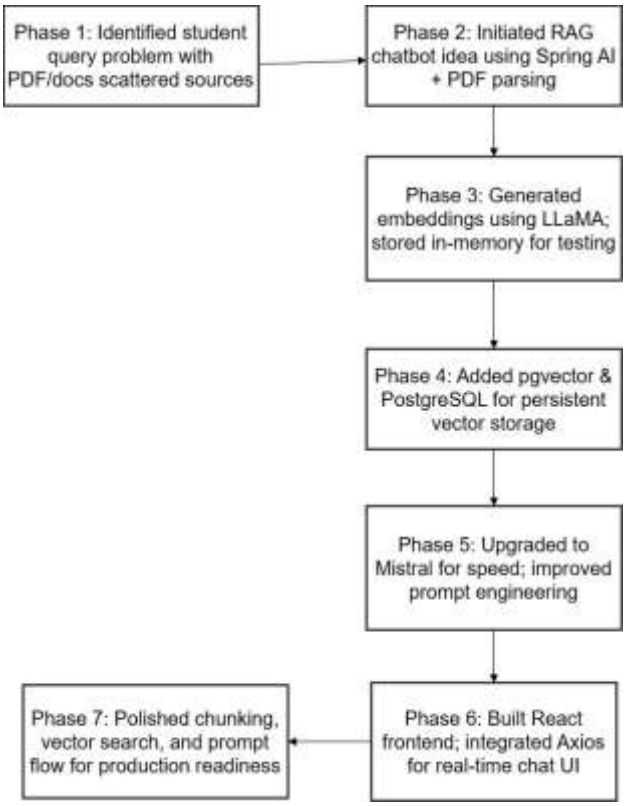

Figure 2: Development Timeline of the Madadgar Project

so they could easily become easily semantically queriable subsequently.These text pieces are inserted into vector representations through Spring AI's integration with Ollama, where we first tested LLaMA 3.2 in generating meaningful embeddings. These vectors are stored in our PostgreSQL vector database to facilitate lightning-fast semantic searches.When a user enters a question or search term in the frontend, developed in React.js, the search passes through RESTful APIs developed in Spring Boot, which in turn talks to the vector database and retrieves the most pertinent documents. In a departure from simple keyword-based searching, our RAG implementation uses semantic search, and the document sections that are similar in meaning to the question or search term are returned, not the actual words themselves. These outcomes are fed into the AI model, which, depending on the implementation, was originally LLaMA 3.2 and is currently upgraded to Mistral through Ollama. This language model processes the question and the accessed context and produces a coherent, context-specific response. This model doesn't repeat back raw facts, but rather produces answers in the same style and tone as a human, making the engagement far more natural-sounding to users.Lastly, the frontend of React.js deals with everything the users view and interact with. It's simple but powerful, and it includes elements of autocomplete, real-time feedback, and neat visual formatting of the responses. This frontend constantly talks to the backend in order to fetch the responses, so the experience is quick and seamless.The whole system is modular and scalable in its design. Although the initial development was at the local level, the system is currently constructed to support future cloud hosting on either AWS or other systems so it can support high levels of traffic efficiently. Every part—from data retrieval and embeddability to response to a query—has gone through testing and iterative refinement so the system responds and is accurate.

4. Datasets

To measure the performance and flexibility of our Retrieval-Augmented Generation (RAG) system, we tested it using a set of combined datasets, vector databases, and large language models (LLMs). We used the dataset construction and testing scenario with several components to test the system's ability

to serve real-world college-related inquiries efficiently. In the following, we outline every component of the dataset and its functionality in the system.Academic brochures, student handbooks, guidebooks, coursework, and research papers constitute the bulk of the dataset, containing structured and unstructured data. They are the data sources.To enhance the usability and retrieval speed of the above documents, the pre-processing is carried out in the following steps: first, the content is extracted in structured format using Apache PDFBox, allowing it to be parsed easily; second, the content is tokenized, breaking it up in smaller pieces so that it can easily be indexed and retrieved; metadata is embedded to further preprocess the dataset by inserting the categorizing tags so that it can easily be scanned through; normalization is performed to standardize text appearance and remove redundant or useless data, which helps in uniformity in all the documents, and finally, the categorized documents are sorted according to subjects, enhancing retrieval precision and enabling the AI models to come up with more contextually correct responses.

5. Result Analysis

5.1 Minimum System Requirements

The hardware requirements for the MADADGAR project have been thoughtfully chosen to ensure the sys- tem runs smoothly and can comfortably handle the computational demands of an AI-driven application. A system with an Intel Core i5 processor or higher is recommended to ensure backend processes—especially those involving AI models and database queries—operate efficiently without lag. For memory, at least 8GB of RAM is necessary, though 16GB is ideal for users working in multitasking environments or managing multiple applications simultaneously. This additional memory helps maintain performance during heavy usage, ensuring that multiple user requests can be processed without slowdown.To support fast data access and reduce load times, a 500GB SSD is recommended. Unlike traditional hard drives, SSDs offer signifi- cantly faster read/write speeds, which is crucial when the system needs to cache, retrieve, or process large datasets or documents. Additionally, since the platform interacts with online APIs and cloud-based ser- vices, a reliable high-speed internet connection is essential. This ensures that users—both students and

Document Type	**Source**	**Key Information**	**Preprocessing Steps**
College Brochures	University Web- sites	Academic programs, Facilities, Policies	Extracted text using Apache PDFBox, To- kenized into smaller segments, Embedded metadata for structured retrieval
Student Handbooks	University Databases	Rules, Guidelines, Contact Info	Normalized text, Re- moved redundant data, Classified based on sec- tions for easy retrieval
Event Guides	College Portals	Academic and Cul- tural Events, Dead- lines	Categorized topics, In- dexed for searchability, Date-based filtering for relevant events
Course Materials	LMS (Moodle, Blackboard)	Syllabus, Lecture Notes	Standardized format, Embedded metadata for faster retrieval, Summa- rization for quick access
Research Papers	Open-access reposi- tories	Prior works on RAG & AI models	Cleaned citation formats, Summarized key insights, Linked with reference tracking systems

Table 1: Overview of Document Types, Sources, Key Information, and Preprocessing Steps for Knowledge Retrieval

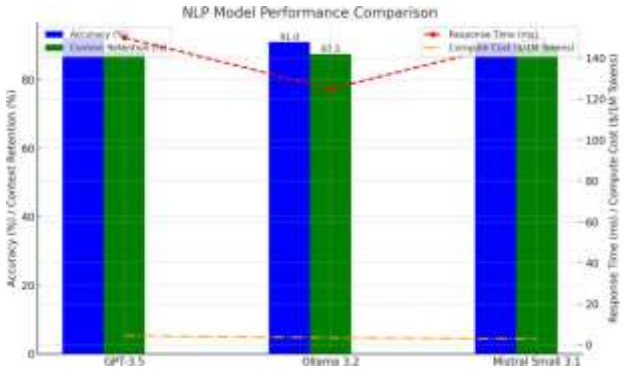

Figure 3: Comparative Analysis of NLP Models: Accuracy, Context Retention, Response Time, and Compute Cost

administrators—experience real-time data retrieval and seamless application responses.On the software side, the application is designed to run on either Windows 10 or Ubuntu 20.04+, both of which provide stable environments for modern development tools and server setups. The backend is powered by Spring Boot, a framework known for building robust and scalable applications. It helps efficiently manage APIs, integrate services, and support the core logic of the platform. We've used Java 11 or later to stay aligned with the latest features and security enhancements, ensuring both performance and compatibility.For the frontend, React.js (version 17 and above) plays a key role. This JavaScript library allows us to build fast, responsive, and interactive user interfaces. Features such as search autocomplete, real-time query input, and dynamic page updates are made possible thanks to React's component-driven architecture. The system's data back- bone relies on MySQL 8.0+, a reliable and high-performing relational database that enables structured storage and quick retrieval of academic and administrative data.To support the development process, several essential tools are used. Postman helps test and debug APIs, ensuring that backend services work correctly and communicate well with the frontend. Git is used for version control, allowing team collaboration and easy management of code updates. Additionally, Node.js 16+ is required to manage frontend dependencies and ensure compatibility with various JavaScript packages and libraries. When it comes to deployment, the system is designed with scalability in mind. Cloud platforms such as Amazon Web Services (AWS) or Google Cloud Platform (GCP) are recommended. These platforms provide the flexibility to scale the appli- cation as the user base grows, support continuous integration and delivery, and allow for automated updates and bug fixes—all while minimizing downtime and maintaining a smooth user experience.Altogether, the combination of these hardware and software requirements ensures that the MADADGAR platform delivers a reliable, secure, and high-performing solution for university students. From responsive user interfaces to real-time AI responses and seamless deployment, every aspect has been tailored to support the dynamic needs of educational institutions and their communities.

5.2 Comparative Analysis of AI Models

This comparison highlights the strengths and trade-offs between different AI models. GPT-3.5 stands out with strong accuracy and context retention, but it's also the most expensive and slower in response time. Ollama 3.2, while slightly less accurate, is faster and more cost-effective—making it a smart choice for structured academic queries. Mistral Small 3.1 strikes a solid balance across all metrics. The findings suggest that selecting the right model depends on the specific needs—whether that's precision, speed, affordability, or handling academic tasks efficiently.

The above diagram is a bar chart illustrating the comparison of the performance of three NLP models (Ollama 3.2, GPT-3.5, and Mistral Small 3.1) on four major metrics: response time, compute cost, context retention, and accuracy. It is a visual representation that shows GPT-3.5 and Ollama 3.2 performing at a similar level of accuracy and context retention, while there is a slight decrease in the case of Mistral Small

3.1 in this regard. Response time and compute cost are also displayed on the secondary y-axis, and we can observe that the response time and compute cost of Ollama 3.2 are slightly less compared to the other two models.

Model	**Accuracy (%)**	**Response Time (ms)**	**Context Retention (%)**	**Compute Cost ($/1M Tokens)**
GPT-3.5	92.1	150	88	4.50
Ollama 3.2	90–92	100–150	85–90	3.00–4.00
Mistral Small 3.1	90–92	150	90	2.80–3.50

Table 2: Performance Comparison of NLP Models

The following table analyses the three AI models of GPT-3.5, Ollama 3.2, and Mistral Small 3.1 in four of the most important measures of their performances: accuracy, response time, context preservation, and compute cost. GPT-3.5 shows the best accuracy but at a bigger compute cost and response time. Ollama

3.2 provides a balance of good accuracy and efficiency, while Mistral Small 3.1 provides efficiency and a low compute cost at the cost of a bit longer response time and similar accuracy.

5.3 Types of Vector Databases

When choosing between Simple Vector Databases and PG Vector Databases, it comes down to performance versus ease of use. Simple Vector Databases excel in speed and scalability, making them ideal for handling massive vector workloads. On the other hand, PG Vector Databases offer smoother integration with Post- greSQL and are easier to work with, though they trade off some speed and scalability due to their relational nature.

Feature	**Simple Vector Databases**	**PG Vector Databases**
Indexing Speed (1-10)	9 (Highly optimized)	7 (Slower)
Query Latency (ms)	30 (Low latency)	50 (Slower)
Scalability (1-10)	9 (Handles vectors well)	6 (Limited)
Ease of Use (1-10)	6 (Hard to setup)	9 (Easily integrates)

Table 3: Performance Comparison of Simple Vector Databases and PG Vector Databases

Simple Vector Databases are high in performance, feature fast indexing, low latency in queries, and high scalability, and are best suited to large amounts of data but more technically demanding to implement. PG Vector Databases, in turn, are more user-friendly and work more easily with existing Postgres systems, but slow in indexing and query latency and are less scalable owing to the relational database architecture. While Simple Vector Databases are more appropriate for high-performance, vector-centric workloads, PG Vector Databases provide a more user-friendly interface at the expense of performance.

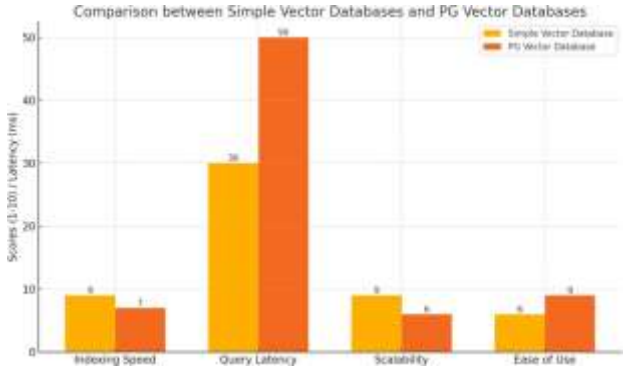

Figure 4: Comparison between Simple Vector Databases and PG Vector Databases

The bar chart contrasts Simple Vector Databases and PG Vector Databases along four aspects: Indexing Speed, Query Latency, Scalability, and Ease of Use. Simple vector databases have better indexing speed, scaling, and query latency compared to PG Vector Databases, where greater values are indicative of faster execution and a greater capacity to work on large data set sizes. However, PG vector databases have better ease of use with a more user-friendly interface. What the chart shows is that even though Simple Vector Databases are faster and more scalable, PG Vector Databases are easier to use and integrate and are more manageable in nature.

6. Conclusion and Future Work

The Madadgar project is a sturdy deployment of Retrieval-Augmented Generation (RAG) in a full-stack system customized to the needs of college-specific retrieval of information. Utilizing Spring AI in the backend and React.js in the frontend, the system provides a functional user experience through precise, context-aware answers extracted from static institutional data. For storing document embeddings efficiently, a light-weight vector database architecture was implemented, allowing the system to achieve rapid and targeted retrieval. Integration of the Ollama 3.2 LLM greatly improved the natural-language-understanding capabilities of the system, enabling intelligent and conversational responses.The project, leveraging both optimization in the models and streamlined data pipelines, saw significant latency reductions in queries, and the system achieved near real-time speeds. Due to its modular architecture, the system provides simple flexibility in expanding it to support other institutions and larger datasets at will. Also, Madadgar presents a good case study of the potential of using modern AI frameworks in real-world, domain-specific scenarios.Future directions in the development of Madadgar include multilingual support, image and document queries, and adaptation learning models*}* personalizing user encounters in the long term. Also, more in-depth integration of advanced transformer models like T5 or GPT-4 can help increase the contextual depth and preciseness of answers. In short, Madadgar not only validates the practicality but also the scalability of knowledge systems driven by AI in academic setups and further afield.

7. Future Research Scope

In order to further augment the system, a variety of strategic upgrades can be employed. Broadening the dataset by including a diverse set of institutions and non-academic facilities, including extracurricular and residential options, will increase its completeness. Utilizing advanced transformer architecture models like T5 or GPT-4 can immensely enhance the accuracy and contextual analysis of the models and optimize them for effective functioning. Incorporating multimodal functionality—in the form of image, video, and audio input support—will enable users to pose different types of queries like viewing a campus map, in addition to utilizing document parsing to extract useful specifics. Personalization using adaptive algorithms and user accounts will provide a more tailored and seamless experience. Further, adding support for multilingual populations using translation APIs will support a broader base of users, advancing inclusivity. Live analytics dashboards may provide meaningful insights on user activity and system functioning and facilitate ongoing optimization. Making it mobile by creating iOS and Android apps will increase ease of use, while effective security practices, using advanced encryption and periodic protocol upgrades, will uphold user trust and data protection standards compliance.

References

[1] Lewis, Patrick, et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.” arXiv preprint arXiv:2005.11401, 2020.

[2] “Spring Boot Documentation.” *Spring.io*, https://spring.io/projects/spring-boot.

[3] “React.js Documentation.” *Reactjs.org*, https://reactjs.org/.

[4] Brown, Tom, et al. “Language Models Are Few-Shot Learners.” *NeurIPS*, 2020.

[5] Devlin, Jacob, et al. “BERT: Pre-training of Deep Bidirectional Transformers.” *NAACL-HLT*, 2019.

[6] “College Information Systems: A Case Study.” *Journal of Educational Technology*, 2021.

[7] “Optimizing Information Retrieval Systems.” *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[8] “Developing Real-Time Applications with React.” *ACM Computing Surveys*, 2020.

[9] “Advanced AI Retrieval Mechanisms.” *Proceedings of the IEEE*, 2021.

[10] “Exploring Multilingual NLP Models.” *Computational Linguistics Journal*, 2022.

[11] “Cloud Deployment for AI Systems.” *Journal of Cloud Computing*, 2021.

[12] “Security in AI-Driven Applications.” *IEEE Transactions on Information Forensics and Security*, 2020.

[13] Alam, Md. Sharif. “Run AI Chatbot Locally with React, Ollama & DeepSeek LLM.” *Medium*, 29 Jan. 2025, https://medium.com/@priom7197/run-ai-chatbot-locally-with-react-ollama-deepseek-llm- 6e762e076f2f.

[14] Devs5003. “Spring Boot Chat Application with DeepSeek and Ollama.” *JavaTechOnline*, 13 Feb. 2025, https://javatechonline.com/spring-boot-chat-application-with-deepseek-and-ollama/.

[15] “Ollama Library.” *Ollama*, https://ollama.com/library.

[16] Hasan, Md. Mehedi. “Build an AI Chatbot Frontend with React, Next.js, and FastAPI Powered by Ollama & DeepSeek-R1.” *Medium*, 3 Mar. 2025, https://medium.com/@rabbi.cse.sust.bd/build-an-ai- chatbot-frontend-with-react-next-js-and-fastapi-powered-by-ollama-deepseek-r1-9a7adc600804.

[17] “Spring AI Series 5: Run With a Local LLM With Ollama.” *YouTube*, uploaded by DevXplaining, 10 Dec. 2024, https://www.youtube.com/watch?v=TBPde6mdiQc.

[18] “Comparing LLM Runtimes for Alfresco in Spring AI.” *Hyland Connect*, 3 Apr. 2025, https://connect.hyland.com/t5/alfresco-blog/comparing-llm-runtimes-for-alfresco-in-spring-ai-ollama- vs/ba-p/488667.

[19] Sajjan, Anup. “LLM Function Calling in Spring AI Results in Error.” *Stack Overflow*, 31 Dec. 2024, https://stackoverflow.com/questions/79319737/llm-function-calling-in-spring-ai-results-in-error- exc-invalidformatexception.

[20] “Why We No Longer Use LangChain for Building Our AI Agents.” *Hacker News*, https://news.ycombinator.com/item?id=40739982.

[21] “Best Model for Programming in React and Node.js?” *Reddit*, https://www.reddit.com/r/LocalLLaMA/comments/1c2g8io/best$_m$*odel*$_f$*or*$_p$*rogramming*$_i$*n*$_r$*eact*$_a$*nd*$_n$*odejs/*.

[22] Yao, Shunyu, et al. “ReAct: Synergizing Reasoning and Acting in Language Models.” arXiv preprint arXiv:2210.03629, 2022.

[23] “Spring AI Documentation.” *Spring.io*, https://spring.io/projects/spring-ai.

[24] “Ollama: Run Large Language Models Locally.” *Ollama*, https://ollama.com/.

[25] “DeepSeek LLM: Open-Source Large Language Model.” *DeepSeek*, https://deepseek.com/llm.

[26] “Next.js Documentation.” *Nextjs.org*, https://nextjs.org/docs.

[27] “FastAPI Documentation.” *FastAPI*, https://fastapi.tiangolo.com/.

[28] “Building AI-Powered Applications with Spring AI.” *Spring Blog*, https://spring.io/blog/2025/01/15/building-ai-powered-applications-with-spring-ai.

[29] “Integrating Ollama with React for Local AI Solutions.” *Dev.to*, https://dev.to/ollama/integrating- ollama-with-react-for-local-ai-solutions-3b1f.

[30] “Deploying LLMs with Spring Boot and Ollama.” *Medium*, https://medium.com/@techguru/deploying- llms-with-spring-boot-and-ollama-5e6f7a8d9c1a.