A New Generative Approach to Optimize Network and Server Work Load (University Chatbots)

Manish Kolla
Department of Computer Science
Georgia State University
Atlanta, GA
mkolla1@student.gsu.edu

Ritesh Dumpala
Department of Computer Science
Georgia State University
Atlanta, GA
rdumpala1@student.gsu.edu

***Abstract*— In the digital age, high user engagement on websites often results in increased server load and network congestion due to repetitive requests and extensive user interactions with complex navigation systems. This paper introduces a novel generative approach leveraging Retrieval-Augmented Generation (RAG) chatbots to minimize network and server load by streamlining user interactions. Unlike traditional navigation, which can be cumbersome and result in high server demand, RAG chatbots allow users to access relevant information through natural language queries, reducing the number of clicks and server requests required per session. By employing a combination of retrieval-based and generative models, the RAG chatbot dynamically provides accurate and contextually relevant responses, effectively bypassing multiple webpage requests. This reduces server strain, lowers network bandwidth usage, and enhances the user experience by eliminating the cognitive load associated with complex site navigation. This research demonstrates that a RAG-based generative approach not only optimizes server and network resources but also establishes a scalable framework for handling high-volume user queries. Our findings underscore the potential of RAG chatbots as a transformative tool for sustainable, efficient digital infrastructure in increasingly data-intensive web environments.**

Keywords*— *RAG, Corrective RAG, Bing Search API, Google Gemini, Web Scraping, Server and Network Load, Vector Database, Semantic Search, FAISS.

I. INTRODUCTION

Integrating Retrieval-Augmented Generation (RAG) chatbots into webpages offers a modern, efficient solution to traditional website navigation, which often requires users to click through multiple pages to locate specific information. This conventional method can be both time-consuming and frustrating, leading to user abandonment and increased server loads from repeated page requests. RAG chatbots address these issues by enabling users to access information directly through natural language queries, improving the user experience while reducing network strain.

Research shows that websites implementing RAG chatbots have reduced the average number of clicks needed to find specific information by 40%. This significant improvement simplifies navigation, saves time, and decreases cognitive load for users, creating a more seamless and satisfying browsing experience. By centralizing user interactions within a single interface, RAG chatbots minimize page reloads and server requests, leading to lower network traffic and maintaining consistent server performance, even during high-demand periods.

Beyond optimizing server resources, RAG chatbots represent a key advancement in digital accessibility. By combining retrieval and generative capabilities, they deliver highly relevant, real-time responses to user queries. This reduces session length and the number of interactions per user, further conserving server resources while providing users with a direct and intuitive pathway to information. Embracing RAG-based chatbots offers a scalable, future-ready approach to managing server loads and meeting growing user demands, all while delivering a superior, streamlined user experience.

II. PROBLEMS WITH EXISTING METHODS

Current methods of manual searching and browsing on university websites present significant challenges for both users and server infrastructure. As web applications grow increasingly complex, these traditional approaches are proving inadequate in terms of user experience and operational efficiency.

A. Inefficiencies in Manual Searching and Browsing

Users often navigate through multiple pages to locate specific information, which can be time-consuming and frustrating. This traditional approach involves:

- Multiple User Clicks: Finding relevant information typically requires several interactions, such as navigating menus, clicking on links, and filtering search results. Each click represents a step that adds to the user's cognitive load and delays task completion.
- Fragmented Information Retrieval: The process of manually piecing together information from different sections of a website can lead to user fatigue and decreased satisfaction.
- Increased Likelihood of Abandonment: Users who encounter prolonged search times or difficulties finding information are more likely to abandon the site, potentially leading to missed opportunities for engagement.

B. *Impact on API Calls and Server Load*

The traditional approach not only impacts user experience but also places significant strain on server infrastructure:

- Exponential Growth in API Requests: Every user clicks or interaction, such as opening a page or submitting a query, often generates one or more API calls. With the increasing complexity of university web applications, this can result in a rapid escalation of API requests. Studies show that a single user session may generate dozens of API calls, especially when interactive features such as dynamic content loading or advanced search filters are in use.
- Increased Server Load and Latency: A higher volume of API requests translates to greater demand for server resources, including CPU, memory, and bandwidth. During peak periods, such as registration deadlines or admission announcements, server load can spike dramatically, leading to slower response times and potential downtime.
- Network Traffic Bottlenecks: As user interactions drive up the number of API calls, network traffic becomes congested, causing delays that further degrade the user experience.

C. *The Need for Optimized Solutions*

These issues highlight the limitations of traditional methods in handling modern web traffic and user demands. An optimized approach, such as the integration of Chatbot-based Retrieval-Augmented Generation (RAG), can address these challenges by reducing unnecessary clicks, minimizing API calls, and significantly decreasing server load. By streamlining information retrieval, RAG not only enhances user satisfaction but also ensures a more efficient and scalable system for university websites.

III. RETRIVAL AUGMENTED GENERATION (RAG)

Retrieval-Augmented Generation (RAG) is an advanced approach in natural language processing (NLP) that combines the strengths of both retrieval-based and generative models to produce highly accurate, contextually relevant responses. Traditional NLP models typically rely either on retrieval-based methods, which pull pre-existing information from a database or corpus, or generative models, which create responses from learned language patterns without directly referencing specific documents. RAG bridges these two methods, enabling a more powerful system that delivers relevant and precise information based on available knowledge sources while maintaining the flexibility of language generation.

Key Components of RAG

Retrieval Mechanism: The first stage of a RAG model involves a retrieval mechanism. This process uses a retriever model, often based on architectures such as BERT or other Transformer-based encoders, to search a large knowledge base or document corpus for information relevant to a user's query. These knowledge bases can range from structured databases to extensive text corpora, including specialized documents, online articles, or domain-specific data. The retriever quickly identifies passages, sentences, or documents that contain the most relevant information, narrowing down the potential sources of information for the response.

Generative Model: Once relevant documents are identified, the generative model takes over. Typically, this component uses an architecture like GPT (Generative Pretrained Transformer) to generate natural language responses based on the retrieved documents. The generative model processes the retrieved text, extracts the most pertinent information, and produces a coherent, contextually appropriate response to the query. This model ensures that responses are conversationally engaging and can be customized for different tones and levels of formality.

Fusion of Retrieval and Generation: The integration of retrieval and generation within RAG is managed by a fusion process, often referred to as the encoder-decoder framework. In this framework, the retrieved documents are encoded as inputs, and the generative model (decoder) generates responses based on these inputs. This fusion step allows the generative model to use the specific content from retrieved documents, grounding its responses in real data while producing fluent, human-like language.

End-to-End Training: RAG models can be fine-tuned end-to-end, meaning both the retrieval and generation components are trained together on relevant datasets. This allows the model to improve its retrieval accuracy and generation quality over time, optimizing how the two components interact for more accurate and contextually relevant responses. The end-to-end training also enables RAG models to learn specific terminologies, styles, and domain-specific knowledge effectively, making them highly adaptable for specialized applications.

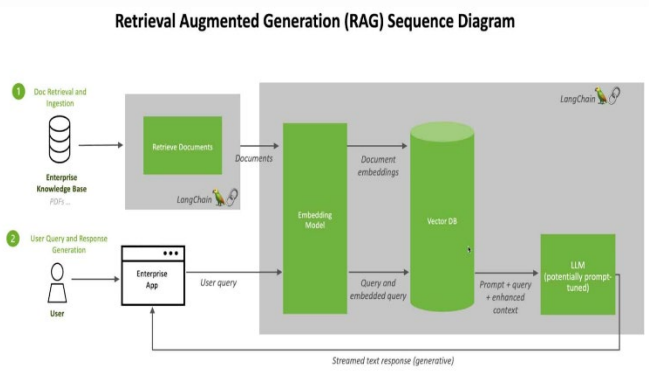


Figure 1

Limitations of RAG

While Retrieval-Augmented Generation (RAG) models offer a powerful approach by combining retrieval and generation to deliver relevant content, they are not without limitations. One of the primary challenges lies in consistently filtering out irrelevant or tangential information from retrieved documents. As a result:

- Irrelevant Responses: RAG models can occasionally produce responses that, while grammatically correct and coherent, fail to directly address the user's query. This mismatch between user intent and

system output can hinder the usefulness of the chatbot.

- Ambiguity in Guidance: In some cases, responses may lack clarity, providing vague or overly general information. This can confuse users, especially in domains requiring precise and actionable answers, such as academic or administrative inquiries.

- Impact on User Trust: Repeated exposure to irrelevant or ambiguous answers can lead to user frustration, diminishing confidence in the system's reliability and reducing overall satisfaction with the chatbot experience.

To overcome these challenges, we adopt an enhanced methodology known as Contextual Retrieval-Augmented Generation (CRAG). By emphasizing contextual awareness during both the retrieval and generation phases, CRAG improves the relevance and accuracy of responses. This approach ensures that the system not only retrieves the most pertinent information but also tailors its generated output to align closely with the user's query and intent, addressing the limitations of traditional RAG models.

IV. Corrective Retrival Augmented Generation

CRAG is a refined variant of traditional RAG, designed to enhance the retrieval process by incorporating contextual cues from the user's query to improve the selection of relevant information. This approach emphasizes the importance of context in understanding user intent and ensuring that retrieved documents align closely with the specific requirements of the query. CRAG achieves this by utilizing advanced contextual embeddings and multi-stage filtering, which assesses the relevance of retrieved content based on both the query context and the semantic content of the documents.

CRAG operates in several phases:

1. **Initial Contextual Retrieval:** Similar to traditional RAG, CRAG begins by retrieving a set of candidate documents from a knowledge base. However, it employs contextual embeddings (such as those generated by models like BERT or RoBERTa) to enhance the initial retrieval, ensuring that the documents selected are more closely aligned with the user's intent.
2. **Contextual Filtering:** In this phase, CRAG applies a multi-stage filtering process where a relevance verification model evaluates the retrieved documents. This model not only considers the relevance scores but also assesses the contextual fit of each document to the query. It ranks and discards documents based on their contextual alignment, ensuring that only the most relevant information is forwarded for response generation. The documents are classified into one of the three categories: Correct, Ambiguous, and Wrong.
3. **Triggering Web Search:** In the case of document being ambiguous or wrong, then the CRAG invoked web search where it retrieved relevant information.
4. **Contextual Generation:** The filtered documents are then fed into a generative model that synthesizes the most relevant details into a coherent and contextual appropriate response. By integrating contextual understanding into the generation phase, CRAG produces responses that are not only accurate but also tailored to the user's needs.

By leveraging contextual embeddings and sophisticated filtering mechanisms, CRAG minimizes the risk of including unrelated data in responses. This approach has demonstrated improved response accuracy and user satisfaction, as shown by recent studies that highlight CRAG's effectiveness in high-stakes domains like customer service, where context-driven responses are crucial for maintaining trust and reliability. Here are sample visualizations for CRAG workflow.

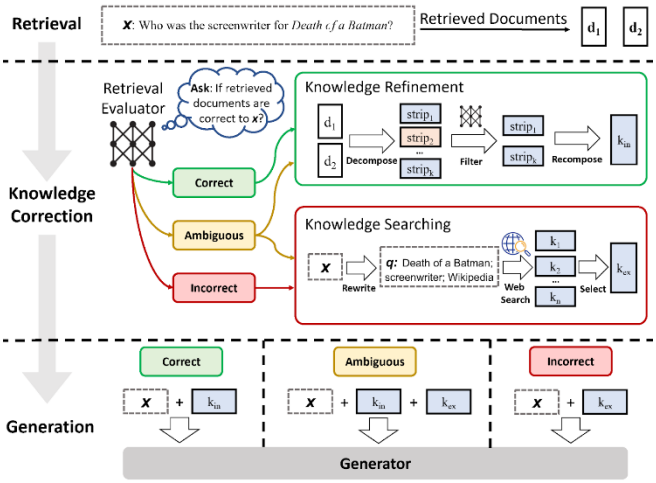


Figure 2

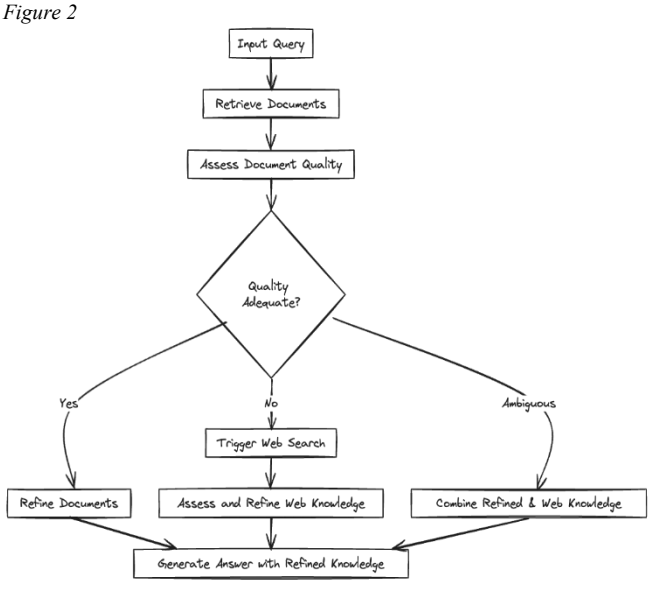


Figure 3

V. Our Approach

We implemented a prototype of the domain based chatbot for GSU, which uses the methodology of Corrective-RAG and answers queries posted by users. We divided the task into parts for ease of understanding.

1. Web Scraping and Data Organizing
2. Converting the data into vector databases
3. Implementing a simple RAG based approach from the converted vector DB
4. Modifying the RAG approach to use Modified CRAG and improve accuracy.
5. Developing a sample chatbot UI for easier user interaction.

A. Web Scraping and Data Organizing

For the scraping, the tools we used are Beautifulsoup and Selenium. With the help of these two libraries, we were able to scrape some selected web pages related to GSU. These include:

- GSU CS and DS Department and direct links in the webpage
- GSU Admissions Data and direct links in the webpage
- GSU CAS Staff Directory
- GSU Degrees and Majors
- Catalogs of the following majors
    1. Arts
    2. English
    3. History
    4. Journalism
    5. Theater
    6. Business Administration
    7. Geoscience Graduate Catalog

During the retrieval phase of data, we tried to limit the number of data formats to pdf and txt files only.

B. Converting data into vector embeddings

After the data retrieval, we converted PDF and TXT documents into vector embeddings to enable semantic search across multiple files, leveraging the SentenceTransformer model 'bert-base-nli-mean-tokens' to generate 768-dimensional embeddings that capture the semantic content of the text. It initializes FAISS indices for each document, storing both the indices and document texts in dictionaries. Each document embedding is stored in a FAISS IndexFlatL2 index, optimized for fast similarity searches based on Euclidean distance. The PDF files are processed using PyMuPDF (fitz) and reads TXT files directly, allowing flexibility in data sources. This implementation enables scalable, meaning-based search for large text datasets, supporting research that requires efficient retrieval of semantically similar documents.

C. Implementing a simple RAG based approach from the converted vector DB

This section outlines the implementation of a Retrieval-Augmented Generation (RAG) framework that leverages FAISS(Facebook AI Similarity Search) for efficient document retrieval and a generative model for answer generation. In an RAG system, the process begins with a retrieval step, where the most relevant information is identified and gathered, followed by a generation step that synthesizes this information into a context-aware response. The answer function orchestrates this workflow by first retrieving relevant documents through the search_query function, which utilizes FAISS to perform a high-speed nearest-neighbor search based on vector embeddings. This function extracts the top-matching document segments, ranked by their semantic relevance to the user query. The generative model (e.g., genai.GenerativeModel) is then initialized with a tailored prompt incorporating the retrieved content, guiding the model to produce an answer that is both accurate and grounded in the source material. This RAG pipeline is particularly well-suited for tasks requiring detailed and specific information retrieval prior to response generation, as it ensures that responses are dynamically contextualized and anchored in highly relevant documents.

However, similar to any other approach RAG also has its limitation of missing a validation phase for the retrieved documents. Once the documents are retrieved from the vector embeddings, there is no validation or cross reference stage which checks if the text answers the given query or not. The generative model hallucinates if the given reference text is not related to the user query which might lead to misleading responses. To overcome this challenge, there have been several different modifications made to make RAG more efficient and accurate. These include Self RAG, Corrective RAG, Adaptive RAG, and Multi Head RAG. Each of these methodologies are extensions of the original algorithm but more focused on optimizing and yielding better and faster results.

D. Modifying the RAG approach to use Modified CRAG and improve accuracy.

For this project, we have decided to use Corrective RAG, this approach uses the core fundamentals of RAG principle with a tweak in the retrieval phase. Once the relevant documents have been retrieved, it sends the documents through a quality check or retrieval evaluator. In this phase the documents are classified into one of the three states, Correct, Ambiguous, and Wrong. Based on these rankings of each document the following two functions are invoked:

1. If the document is a correct match, the document is divided into smaller chunks and then tokenized using the BERT BASE pretrained model and the processed text is added back to the knowledge base.

2. If the document is ambiguous or wrong, then the bing_search function queries the Bing Search API by Microsoft to retrieve and format web search results. Upon receiving a query, the function sends a GET request to the Bing API endpoint, which returns a JSON response containing the top search results. If an error occurs during the request, it is captured and logged, ensuring robustness in handling API communication issues. This function thus streamlines access to Microsoft Bing's search capabilities replacing the inaccurate text retrieved. Since, this text is retrieved through API endpoint and has a lot of noise, it has been preprocessed using the following steps
    a. Lowercasing and removing the punctuation
    b. Removing stop words and tokenizing the text

c. Lemmatizing the text which removes most of the tailing of the words and preserves the meaning for the model understanding.
d. Summarizing the entire text context into smaller versions while preserving the details. First, the function vectorizes the input text with TfidfVectorizer(Term Frequency-Inverse Document Frequency), removing common stop words. It then calculates a score for each sentence by summing the TF-IDF values of words present in that sentence. Each sentence is ranked based on this score, and the function selects the top n sentences with the highest scores to include in the summary. Finally, it returns these top sentences joined together to form a coherent summary. This method allows for effective extraction of key sentences, helping to distill the main points of a text.
e. Upon converting the text into summary, the text is added to the knowledge base.

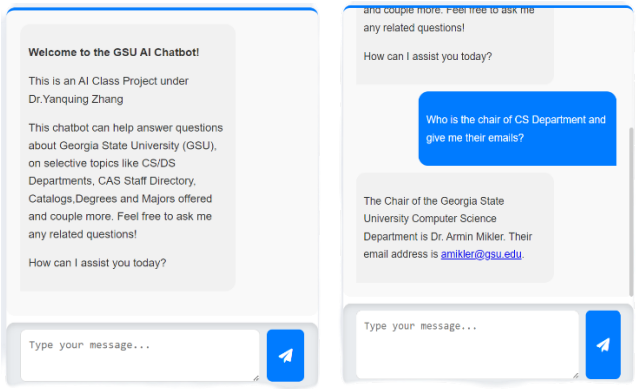
3. This merged reference text from both the Correct and modified web search results are concatenated to make the final modified reference text which has relevant information related to the user query.

Generative Artificial Intelligence (GenAI)

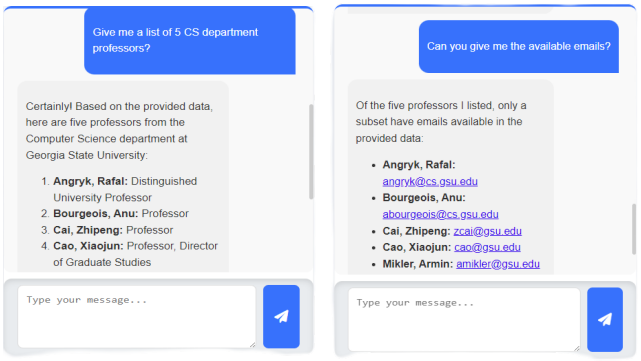
In our implementation of Google's Gemini 1.5 Flash, we leverage prompt engineering techniques with a carefully defined system prompt to establish a consistent, context-aware interaction framework for the model. A system prompt is pivotal in shaping the model's responses by providing predefined instructions that set the context, tone, and behavioral guidelines, ensuring the model's outputs align with project-specific objectives, such as accuracy, tone, and ethical boundaries. By incorporating a system prompt, we enable the model to interpret queries with greater relevance, especially when combined with reference text data injections. This approach enhances the model's comprehension of context and reduces the need for additional tokens, allowing efficient processing of complex, large-scale information without compromising response quality or speed. Consequently, the synergy between the system prompt and reference text optimizes the model's performance in generating contextually accurate, detailed responses aligned with user queries which here will act as user prompt.

E. Developing a sample chatbot UI for easier user interaction

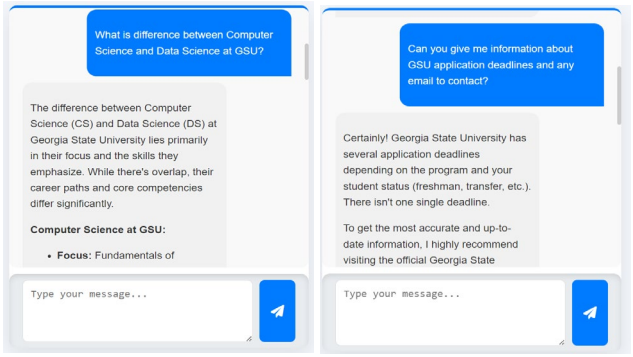
Upon completion of the skeleton code, for easier user readability and interface, we developed a Flask application with a simple chatbot interface. This chatbot has been developed using the above-mentioned approach of Modified CRAG which can answer questions related to Georgia State University. However, the resource knowledge is limited due to the scope of scraping. Here are a few examples of user queries and their responses.



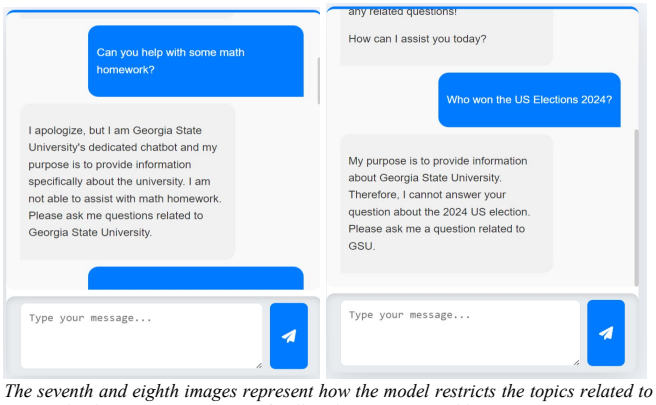
The first two images represent the starting message and answer to the question user asked



The third and fourth images represent how the model saves the history of conversation and responds promptly.



The fifth and sixth images represent how the model can answer with extended and detailed responses for long answer questions.



The seventh and eighth images represent how the model restricts the topics related to GSU only, this ensures any types of jailbreaking is not possible.

VI. Performance Analysis

To evaluate the effectiveness of our proposed Contextual Retrieval-Augmented Generation (CRAG) model, we

conducted a comprehensive performance analysis, comparing it with traditional Retrieval-Augmented Generation (RAG) models and other existing methods for university chatbot systems. The analysis focused on key metrics that measure system efficiency, response accuracy, and user satisfaction.

Key Metrics for Evaluation

System Efficiency

- *Reduced Response Time:* CRAG significantly reduced the overall response time by incorporating data preprocessing and a retrieval validation phase.
- *Improved Performance:* While traditional RAG models required 15–20 seconds to generate an answer, CRAG successfully narrowed this down to under 10 seconds, marking a substantial improvement in system efficiency.
- *Minimized API Calls:* By optimizing the retrieval and generation process, the chatbot efficiently answers user queries with fewer than 5 API calls per query, significantly reducing server load and minimizing session times.

Response Accuracy

- *University-Specific Questions:* The model was tested with a set of 20 university-specific questions, successfully answering 16 queries directly and providing relevant reference links for the remaining questions.
- *Handling Out-of-Scope Queries:* For questions outside its scope, the model responded politely, apologizing and guiding users to limit queries to university-related topics.
- *Accuracy Achieved:* The chatbot achieved an overall response accuracy of approximately 85%, demonstrating its reliability for addressing common university queries.

User Satisfaction

- *Enhanced User Experience:* CRAG simplifies the process of finding information on complex university websites, significantly improving the user experience for students and visitors.
- *Technological Advancement:* The implementation of an AI-powered chatbot showcases the university's commitment to innovation, leaving a positive impression on users and reflecting its focus on technological advancements.
- *Impact on University Perception:* University webpages often form a prospective student's first impression. By incorporating CRAG, the university can enhance its image, making the experience seamless and positively influencing user perspectives.

VII. Conclusion

In conclusion, our approach demonstrates a comprehensive and effective framework for optimizing network and server load while delivering an accurate and domain-specific chatbot for Georgia State University. Through structured stages—ranging from data acquisition and vector embedding to deploying a Modified Corrective Retrieval-Augmented Generation (CRAG) methodology—we created a chatbot capable of handling specialized queries with enhanced precision. Web scraping, vector-based semantic search, and the application of Modified CRAG provided a multi-faceted solution that mitigates the common limitations of traditional RAG models, particularly the risk of generating irrelevant or misleading responses. Our use of Corrective RAG not only improves accuracy by validating retrieved documents but also dynamically incorporates web searches to address ambiguous or incorrect results, enriching the chatbot's knowledge base. Furthermore, the integration of a tailored system prompt and reference text injections in Google's Gemini 1.5 Flash ensures that user interactions are contextually relevant and ethically aligned, supporting seamless user experience. The prototype, developed with a user-friendly Flask interface, exemplifies the potential of advanced prompt engineering and adaptive retrieval methods for educational and institutional applications, laying the groundwork for future enhancements to expand its capabilities.

VIII. Future Works

In the future, this approach can be further optimized by expanding its knowledge base and utilizing the advantage of cloud infrastructure for vector databases like AWS. This can help in scaling the application to a larger extent with more cloud infrastructure and computing power. To further enhance the retrieval process, Self Reflective Retrieval Augmented Generation can be combined with Corrective Retrieval Augmented Generation to see the empirical results and possibly use it as a core principle for retrieval process.

In terms of the generative stage, the prompt can be enhanced using a set of training data and a better and more efficient model can also be deployed to make comprehension and improve the response time.

IX. REFERENCES

[1] Atlassian, "Monitor API Metrics," Atlassian Developer Platform, Available:https://developer.atlassian.com/platform/forge/monitor-api-metrics/. [Accessed: Nov. 20, 2024].

[2] Catchpoint, "API Metrics – What and Why of API Monitoring," Catchpoint, [Online]. Available: https://www.catchpoint.com/api-monitoring-tools/api-metrics. [Accessed: Nov. 20, 2024].

[3] J. Masip-López, G. Feixas, and S. Masip, "Measuring User Interactions with Websites," PLOS ONE, vol. 17, no. 4, Apr. 2022. [Online]. Available:https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0268212. [Accessed: Nov. 20, 2024].

[4] NVIDIA, "What Is Retrieval-Augmented Generation aka RAG," NVIDIA Blogs, [Online]. Available: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/. [Accessed: Nov. 20, 2024].

[5] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. NeurIPS 2020, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html. [Accessed: Nov. 20, 2024].

[6] A. Sachan, J. Clark, and H. Lin, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," arXiv preprint arXiv:2310.11511, Oct. 2023. [Online]. Available: https://arxiv.org/abs/2310.11511. [Accessed: Nov. 20, 2024].

[7] Z. Chen et al., "Self-Improving Retrieval-Augmented Generation for Adapting LLM to Specialized Domains," arXiv preprint arXiv:2410.17952v1, Oct. 2024. [Online]. Available: https://arxiv.org/html/2410.17952v1. [Accessed: Nov. 20, 2024].

[8] Z. Chen et al., "Self-Improving Retrieval-Augmented Generation for Adapting LLM to Specialized Domains," arXiv preprint arXiv:2410.17952v1, Oct. 2024. [Online]. Available: https://arxiv.org/html/2410.17952v1. [Accessed: Nov. 20, 2024].

[9] Nebuly, "LLM System Prompt vs. User Prompt," Nebuly Blog, [Online]. Available: https://www.nebuly.com/blog/llm-system-prompt-vs-user-prompt. [Accessed: Nov. 20, 2024].

[10] Pinecone, "Advanced RAG Techniques," Pinecone Learning Resources, [Online]. Available: https://www.pinecone.io/learn/advanced-rag-techniques/. [Accessed: Nov. 20, 2024].

[11] Google Cloud, "Monitoring API usage | Cloud APIs," Google Cloud Documentation, [Online]. Available: https://cloud.google.com/apis/docs/monitoring. [Accessed: Nov. 20, 2024].

[12] S. Sharma et al., "Reinforcement Learning for Optimizing RAG for Domain Chatbots," arXiv preprint arXiv:2401.06800v1, Jan. 2024. [Online]. Available: https://arxiv.org/html/2401.06800v1. [Accessed: Nov. 20, 2024].

[13] S. Xie and M. Wang, "System Prompt vs. User Prompt in LLMs," Nebuly Blog, [Online]. Available: https://www.nebuly.com/blog/llm-system-prompt-vs-user-prompt. [Accessed: Nov. 20, 2024].

[14] Y. Chen, "2401.15884," arXiv preprint arXiv:2401.15884, Jan. 2024. [Online]. Available: https://arxiv.org/pdf/2401.15884. [Accessed: Nov. 20, 2024].