Agriculture Insurance LLM Conversational Assistant

Team Members: Kyle Hilgenberg, Saketh Rajesh, Michael Shi, An Truong

Clients: Dr. Elinor Benami, Ramaraja Ramanujan, Manoochehr Shirzaei, Mehmet Yardimci

Instructor: Dr. Edward A. Fox

Multimedia, Hypertext, and Information Access (CS 4624)

Department of Computer Science

Virginia Tech, Blacksburg, VA 24061

May 9, 2024

Table of Contents

Table of Figures

Table of Tables

Abstract

Our team was tasked to develop a conversational assistant to aid users in understanding and choosing appropriate agricultural insurance policies. We successfully implemented an assistant that leverages a Large Language Model (LLM) trained on datasets from the Rainfall Index Insurance Standards Handbook and United States Department of Agriculture (USDA) site information. The project encompasses the design of an interface that facilitates seamless interactions between the user and the assistant. The scope of the project includes creating a usable interface, integrating the backend with a Flask Application Programming Interface (API), and deploying the assistant on the Endeavour cluster at Virginia Tech. Testing was conducted by evaluating the assistant's answers to ensure it met the high standards required for practical use. These efforts have culminated in a robust tool that simplifies the process of selecting appropriate agricultural insurance policies, making it less daunting and more accessible for users.

1 Introduction

1.1 Problem

Although insurance is meant to offer a safety net during difficult times, the process of choosing an appropriate insurance policy in the United States is often daunting [1]. Administrative documents for insurance are often full of complex terms and jargon, making it hard for potential and sometimes even existing policyholders to make informed choices. Making the wrong decisions can result in minimal or no payments when someone needs it the most, leaving them in a vulnerable position, and possibly worse off than if they did not have insurance at all. For subsidized programs – which agricultural insurance programs often are – poor policy decisions can also mean that taxpayer funds are not being as effectively used as they could be. A significant shortfall in these policies is the lack of coverage for drought, a major risk in agriculture that, when not addressed, forces farmers to shoulder the severe impacts alone.

1.2 Motivation

To this end, this project aims to create an easily usable conversational assistant to help individuals navigate and comprehend insurance policy terms. By providing clear explanations and tailored recommendations, the assistant will assist users in selecting the insurance policies that best meet their specific needs and circumstances. The focus of this project will be on a specific agricultural insurance program centered on issuing payouts when it detects rainfall deficits occurring in insured pasture, forage, and rangelands in the US. Pasture, forage, and rangelands make up more than half of the total US land area [1]. However, only a fraction of these areas are insured, despite the increased weather variability against which insurance can play an important protective role.

1.3 Approach

This assistant aims to provide clear, easily understood explanations and guidance to help potential policyholders navigate their choices in this program. In particular, we envision the assistant

will, at a minimum, be able to respond to queries and assist their users in selecting appropriate insurance terms (e.g., extent and timing for coverage). Time permitting, we will develop a series of simulations to examine the potential impact of different choices. These simulations could help potential policyholders understand how the program will affect them under various scenarios that reflect plausible conditions. Furthermore, we propose expanding the assistant's capabilities to include the generation of visualizations. These visualizations will illustrate the potential outcomes of different choices, making it easier for users to grasp the implications of their decisions. Ultimately, this tool is designed to empower individuals with the knowledge and confidence to make well-informed decisions about this crucial financial risk management tool. By enhancing their understanding of the program's nuances, potential policyholders can better assess their options and choose the coverage that best suits their needs.

1.4 Clients

Our primary client is Dr. Elinor Benami, an Assistant Professor in the Department of Agricultural and Applied Economics at Virginia Tech. However, she has been on maternity leave for the entirety of this semester. Our main points of contact have been two graduate research assistants, Ramaraja Ramanujan and Mehmet Yardimci, who are part of Dr. Benami's team. Additionally, we received valuable input and feedback from two other supporting members, Dr. Manoochehr Shirzaei and Dr. Edward Fox.

2 Requirements

2.1 Large Language Model

The LLM will use datasets for training based on the Rainfall Index Insurance Standards Handbook and USDA site information [2]. It is important that we use an open-source LLM [3] instead of using the OpenAI GPT API [4]. The LLM should be able to handle a wide range of questions related to the insurance program's terms and conditions while gracefully responding to queries it doesn't have information about, instead of fabricating and convincingly arguing for an incorrect response. Additionally, it should provide personalized insurance choice recommendations based on user-provided data and needs.

2.2 User Interface

The user interface should be designed as a standard assistant interface where users can type in questions and receive text responses. In addition to text-based interactions, the interface should also support helpful and relevant visualizations, such as historical trends and the impact of various reasonable choices on insurance payouts, to enhance user understanding and engagement. By incorporating these features, the user interface will not only facilitate effective communication between the user and the assistant but also provide a comprehensive and interactive platform for exploring insurance options and making informed decisions.

3 Design

3.1 Frontend

The frontend of our platform is primarily based on an open-source interactive conversational assistant model, with a few modifications to better align with our requirements. These modifications include a sign-in option on the landing page featuring a Google icon for Google account access, a welcome message on the chat screen providing custom starting prompts, and a dropdown menu for selecting the desired LLM. Apart from these changes, the rest of the frontend is largely derived from an open-source project developed by Vercel [3].

Our project incorporated a variety of tools to enhance its styling and functionality. We utilized an open-source component library [5] to access pre-built components, paired with Tailwind Cascading Style Sheets (CSS) [6] for their styling. Additionally, we incorporated Radix UI [7] for headless component primitives and Phosphor Icons [8] for visually appealing icons. To facilitate API development within our Next.js application, we employed Next.js API routes [9]. These server-side functions handle HTTP requests [10] and generate responses. They are easily implemented by adding JavaScript [11] files to the app directory of our Next.js project.

3.2 Backend

The language model backend is implemented through deploying an open-source model called Ollama [12] to serve as the foundation for the assistant responses. The model is deployed on Virginia Tech's Endeavour cluster. To interface with the model, we built a Flask API [13] to support streaming responses from clusters as well as to perform Retrieval-Augmented Generation (RAG) [14]. For handling users and accounts, we implement OAuth using Google OAuth [15] services to allow users to log in with their Gmail accounts. Additionally, we will eventually build a backend to support individual user chat history along with private user information stored in a database such as Redis [16].

In our project, we implemented LLM streaming [17] and prompt engineering [18] to enhance the interaction capabilities and responsiveness of our system. LLM streaming allows us to process and generate text in real-time, handling large volumes of data efficiently by streaming the output as it's generated. This approach significantly improves the user experience by reducing wait times for responses. Furthermore, we carefully designed our prompts—using prompt engineering techniques—to elicit the most accurate and relevant responses from the model. Specifically, we instructed the assistant to not make any promises to the users. This precaution is designed to mitigate potential legal complications and manage user expectations accurately. This precision in prompt crafting ensures that our system not only understands the user's queries more effectively but also delivers more contextually appropriate responses.

4 Implementation

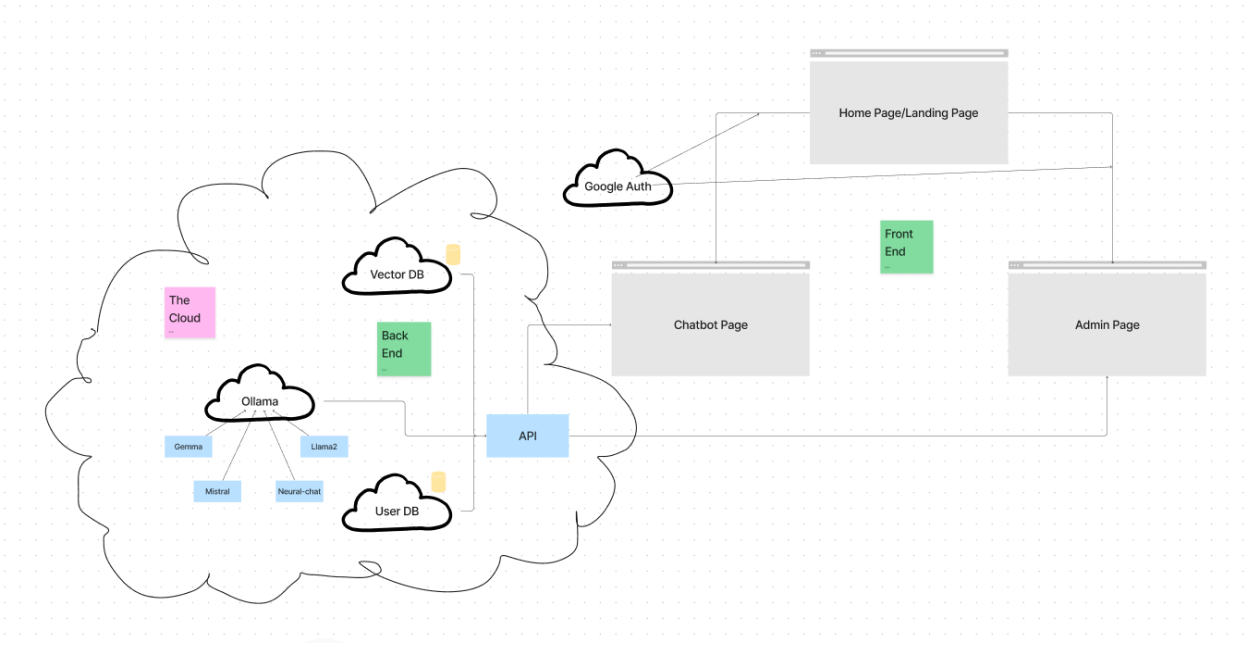


Figure 1: Architecture Diagram

In Figure 1, the Agriculture Insurance LLM follows a microservice architecture [19] split into the two main parts of frontend and backend. The frontend has several pages that require a Google authorization to interact with (assistant and admin page). Any request for a chat operation goes through the main backend API gateway. Here, depending on the request, there is a combination of queries to the vector database [20], request to the Ollama API server [12] (to interface with a selected LLM), and the user database. The website is publicly accessible [21].

In our project, we refined the data handling process by segmenting information from the Handbook [2] to enhance the accuracy of the LLM's responses to user queries. We organized this segmented data into a JavaScript Object Notation (JSON) file [22], carefully noting the page number, section, and subsection from which each piece of data originated. This structured approach enables the assistant to not only retrieve information more precisely but also to cite the exact source of its responses. By providing citations directly linked to the segmented data, the assistant ensures transparency and

increases its credibility, giving users clear insights into the origins of the information provided. This method not only improves user trust but also enhances the overall functionality and reliability of the system.

In our project, we opted to use Redis [16] for storing user chat queries, ensuring efficient data management and retrieval. While the open-source frontend template our project is based on utilizes Vercel KV [23], it's important to note that Vercel KV is essentially a Redis-based service. Vercel KV operates as a cloud service provided by Vercel, offering a streamlined way to manage key-value storage in the cloud. Recognizing this, we decided to host our own Redis database to maintain greater control over our data infrastructure. To seamlessly integrate with our existing setup, we utilized the Vercel KV package, which allowed us to interface with our Redis database as if we were directly using Vercel KV. This approach not only leveraged the simplicity and functionality of Vercel's cloud solutions but also provided us with the customization and independence of managing our own database server. This decision enhanced our project's performance and scalability by utilizing robust and reliable data storage solutions.

5 Testing and Evaluation

Our client equipped us with a series of validation questions [24] and corresponding answers that had been meticulously researched by human experts, aimed at rigorously testing our assistant's capabilities. To facilitate this testing process, we crafted a script designed to feed each question into six different models from our suite, capturing their responses in a detailed spreadsheet. This spreadsheet [25] was then shared with our clients, who were given the opportunity to evaluate and rank the performance of each model based on the accuracy and relevance of their responses.

During this evaluation phase, our clients pinpointed specific aspects of the responses that were particularly effective, as well as those that were irrelevant or incorrect. A surprising insight emerged from this analysis: responses generated using hand-segmented data tended to be more accurate than those derived from data segmented based on punctuation. This revelation prompted us to reconsider our data segmentation approach, recognizing that less tailored segmentation could sometimes yield superior results by providing a broader context. This feedback was invaluable, guiding us to identify future work that can be done to enhance the overall effectiveness and reliability of our assistant.

6 User Manual

6.1 Prerequisites

To start with, this assistant should only be used by those with queries about Agricultural Insurance, as it will be of little use otherwise. This is intentional, because a narrower breadth allows for more depth on the topic of Agricultural Insurance. To that end, if the user has questions such as “What does this term mean in the context of Agricultural Insurance?”, or “What is the best Agricultural Insurance plan for me given these circumstances?”, then this assistant is the correct tool to use. The conversational assistant will use data from the 2024 Rainfall Index Insurance Standards Handbook [2] to answer the user’s questions.

6.3 Tutorial

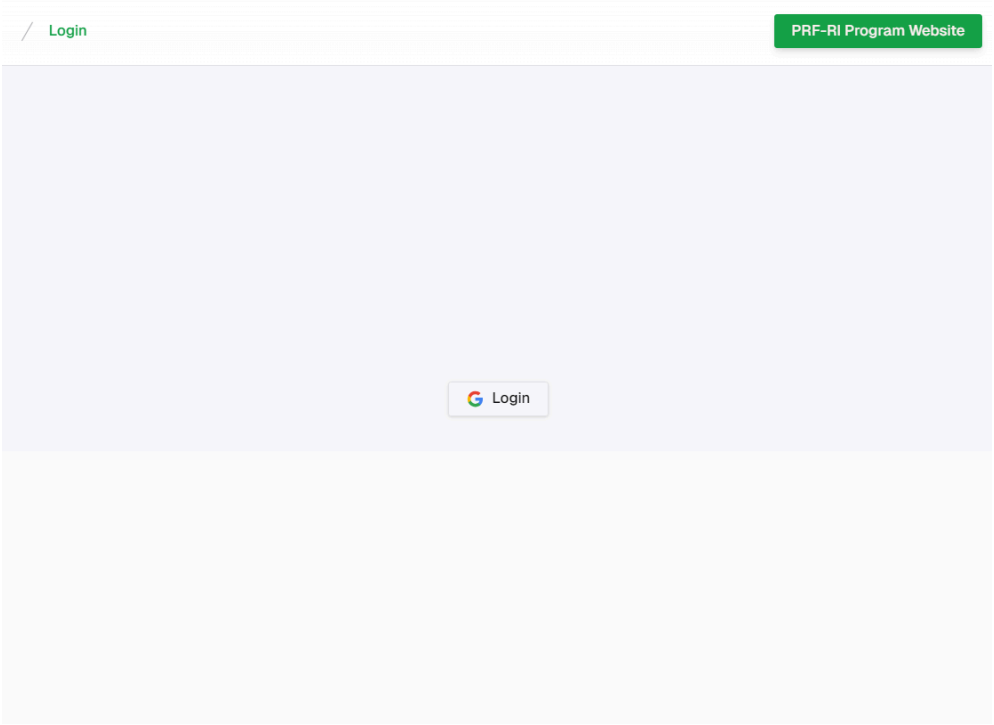


Figure 2: Login Page

The first screen displays a minimalist login page (shown in Figure 2). At the top left corner, there's a navigation breadcrumb or title indicating the current page as “Login”. Centered on the page is a

button with the Google logo accompanied by the text “Login” which users can click to login to their Google account. On the top right corner, there is a "PRF-RI Program Website" button that when clicked, will take the user to the USDA site with Pasture, Rangeland, Forage information.

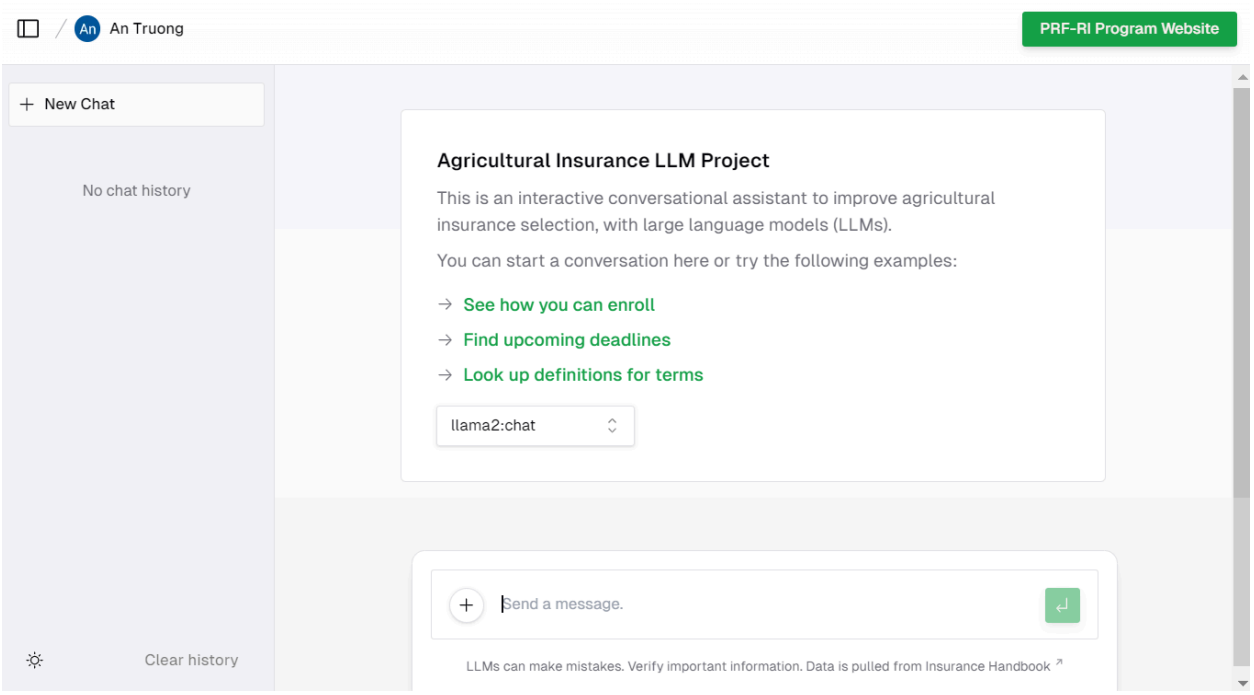


Figure 3: Website Empty Chat Screen

After the user logs into their Google account, the website will display the main chat screen (shown in Figure 3).

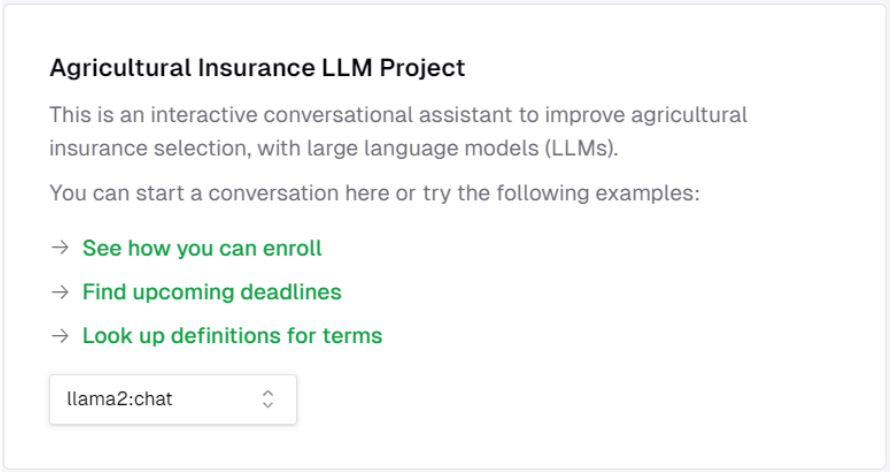


Figure 4: Welcome Card with Example Prompts

The chat interface will be ready for input at the bottom of the page, and just above the dropdown menu, there will be three example prompts related to the chosen LLM that the user might consider using as a starting point (shown in Figure 4).

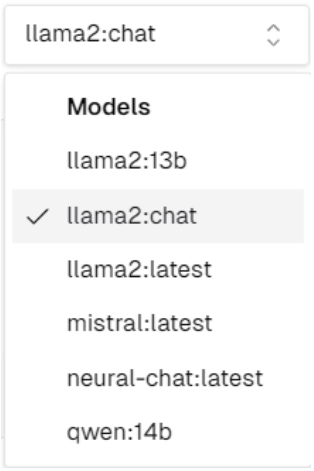


Figure 5: Model Dropdown Selection

If the user clicks the chat model button, a dropdown menu will appear (shown in Figure 5). This menu allows the user to choose from various model options depending on their needs.

At this stage, the user will have two options: they can click on one of the example prompts, highlighted in green, to initiate a conversation with the LLM based on that prompt, or they can type their own message into the chat at the bottom of the page.

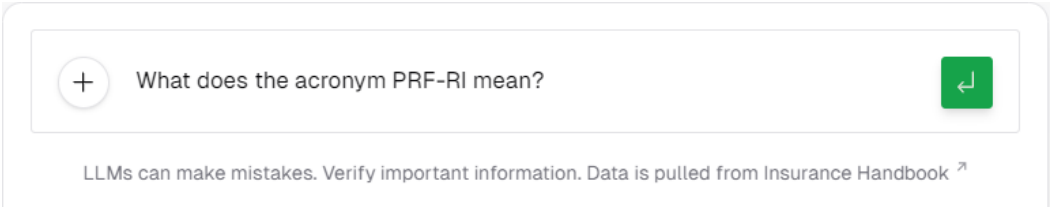


Figure 6: Message Text Field with an Example Prompt

When typing a message in the message text field (shown in Figure 6), it is most effective for the user to provide clear and concise details that cover all specifics of their problem and/or question. Then, the user can click the green enter button to submit their query.

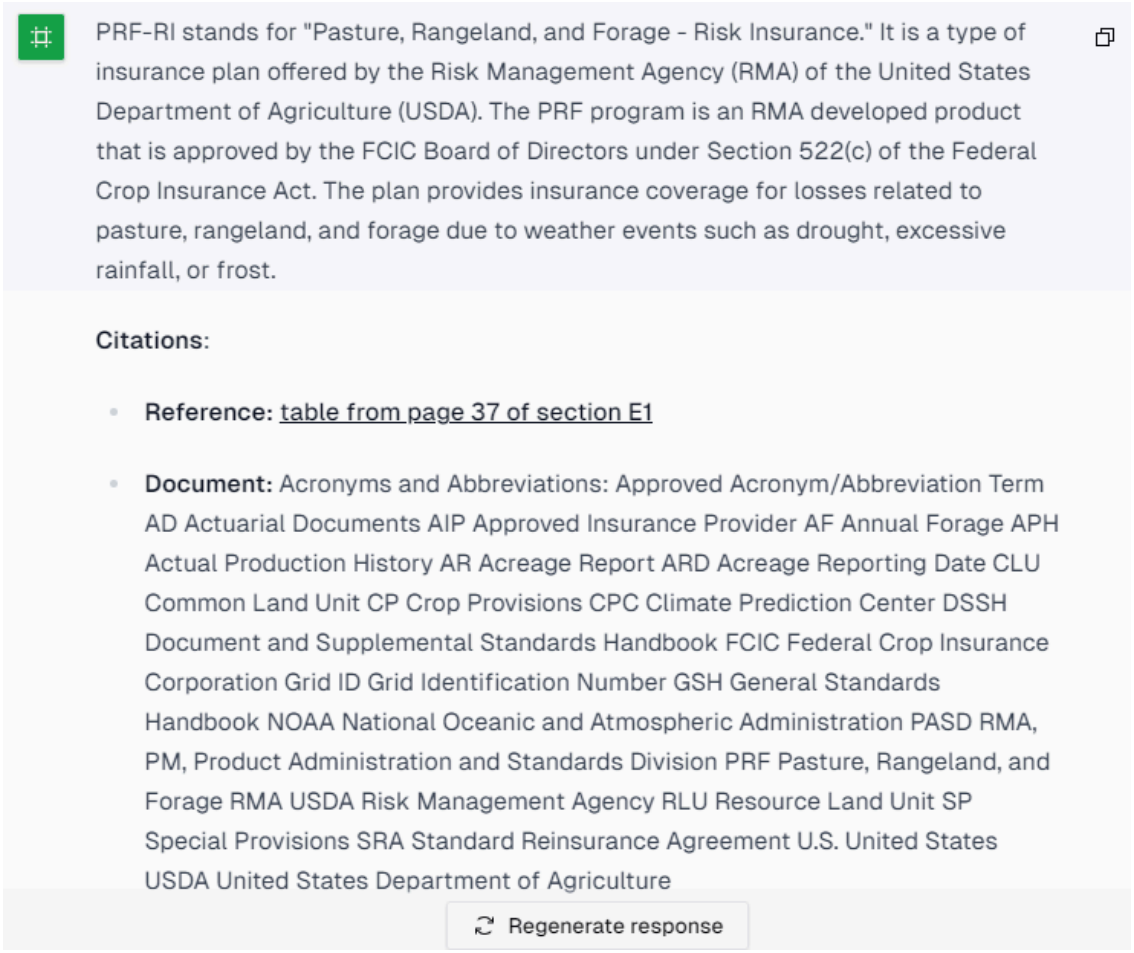


Figure 7: Answer with Citation

The assistant will provide a response along with a reference citation (shown in Figure 7). Users can follow the provided reference link to directly access the specific page in the Rainfall Index Insurance Standards Handbook [2] from which the information was sourced. By following these steps, the user can effectively interact with the LLM and receive relevant responses to their queries.

7 Developer Manual

7.1 Development

The development phase of this project embraced the agile methodology [26] , fostering an iterative and collaborative approach. This methodology prioritizes flexibility, adaptability, and customer feedback throughout the development life cycle. By breaking down the project into manageable increments, commonly known as sprints [27], we ensured a continuous and transparent development process.

During each sprint, cross-functional teams collaborated closely, addressing evolving requirements and incorporating feedback promptly. This agile framework facilitated efficient communication, allowing for rapid adjustments and enhancements. Moreover, the iterative nature of agile development enabled us to deliver a product that closely aligned with the evolving needs and expectations of clients.

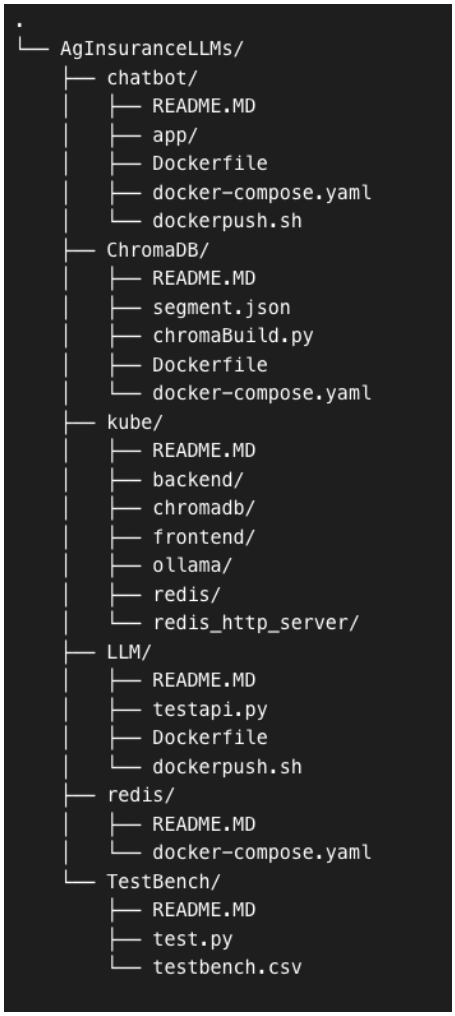


Figure 8: Table of critical files

Developers new to the project should review Figure 8. This contains critical files within the system and where to find valuable information for each service. In the main directory, we have 6 subdirectories representing important services for frontend, Kubernetes, Redis database, LLM, ChromaDB Vector Database, and our test bench. In each of these, we have a README.MD file that overviews all important information from getting started to deployment. This allows a new developer to get familiar with each service and deploy the service using the corresponding dockerpush.sh and docker-compose.yaml. In the event of possible issues, our documentation overviews common bugs and how to resolve them.

7.2 Deployment Tools

The deployment and service provisioning of the website were facilitated through the utilization of various tools. Each integral standalone element, including the frontend, backend API, vector database, and Ollama, has undergone the process of containerization using Docker [28]. This approach enhances portability and provides a high degree of adaptability when deployed across diverse environments.

The Docker images resulting from this process are systematically stored within the GitLab container registry hosted on git.cs.vt.edu [29]. Furthermore, to streamline the coordination of these standalone components, we harnessed the capabilities of the Endeavour Kubernetes cluster [30], thereby orchestrating their deployment seamlessly.

7.3 Deployment Process

The deployment processes for the backend and frontend are distinct. To deploy the frontend:

1. Navigate to the project directory on your local machine.
2. Execute the “bash dockerpush.sh” script. This script will build the production Docker image for the frontend, tag it, and push it to the container registry within the client’s repository. Alet
3. Open a web browser and go to https://launch.cs.vt.edu.
4. Access the Endeavour cluster within the website.

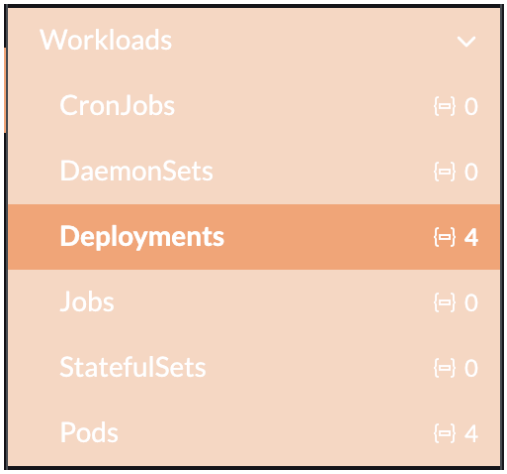


Figure 9: Workloads Tab within the Endeavour Cluster

5. Under the Workloads tab, find and click on Deployments (shown in Figure 9).
6. Locate the frontend deployment, click the three dots for more options, and select Redeploy.
7. Monitor the deployment process to ensure the updated frontend version is live and functioning in the production environment, which can found at the URL in request host on Rancher for the frontend deployment.

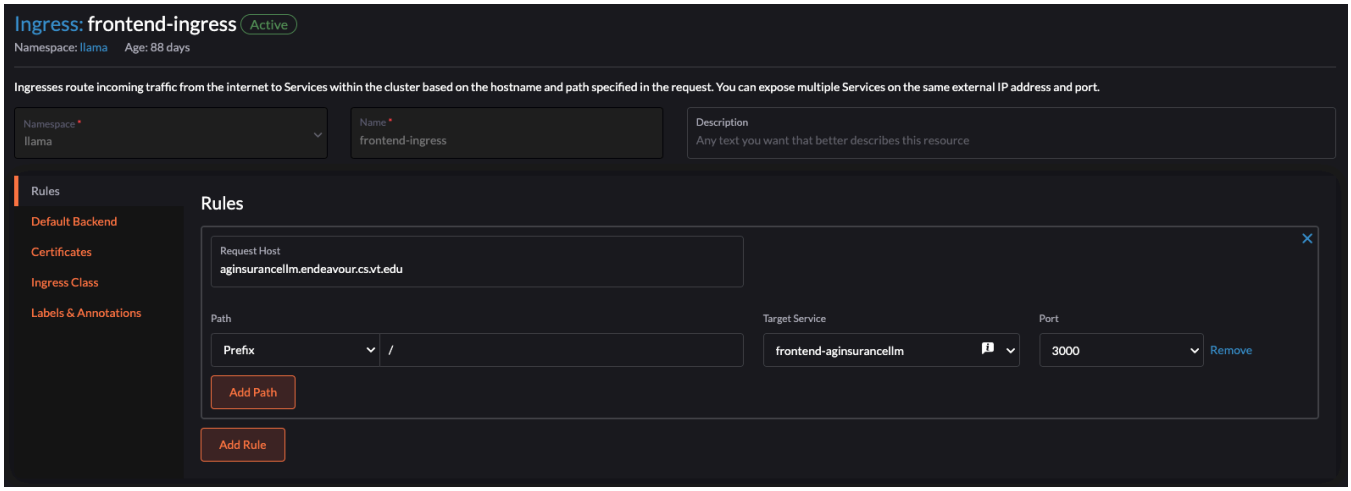
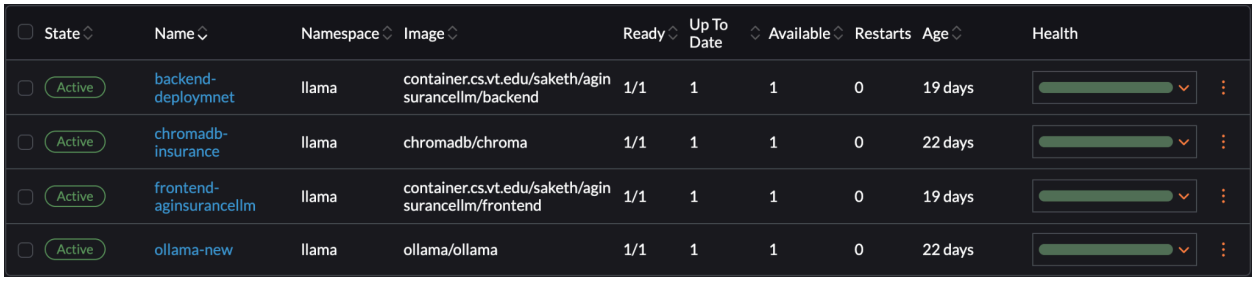


Figure 10: Configuring hostname in Ingress

To deploy the backend:

1. Navigate to the project directory on your local machine.

2. Run the “bash dockerpush.sh” script to build and push the production Docker image for the backend to the same container registry.
3. Use a web browser to visit https://launch.cs.vt.edu.
4. Enter the Endeavour cluster through the portal.
5. Navigate to Deployments under the Workloads tab.



State	Name	Namespace	Image	Ready	Up To Date	Available	Restarts	Age	Health
Active	backend-deploymnet	llama	container.cs.vt.edu/saketh/aginsurancellm/backend	1/1	1	1	0	19 days	
Active	chromadb-insurance	llama	chromadb/chroma	1/1	1	1	0	22 days	
Active	frontend-aginsurancellm	llama	container.cs.vt.edu/saketh/aginsurancellm/frontend	1/1	1	1	0	19 days	
Active	ollama-new	llama	ollama/ollama	1/1	1	1	0	22 days	

Figure 11: List of deployments in the Endeavour cluster

6. Find the backend deployment (shown in Figure 9), click on the three dots for additional actions, and choose Redeploy.
7. Observe the deployment to confirm the backend is updated and operational in production.

8 Lessons Learned

8.1 Timeline

	Date	Brief Task Overview	Status
Phase 1	February 5, 2024	Received approval from clients to proceed with the proposed plan for Phase 1.	Completed
	February 6, 2024	Presentation 1 completed and shown to the class.	Completed
	February 12, 2024	Successfully deployed the RAG model on a Flask server, enabling the conversational assistant to leverage external knowledge sources for generating responses.	Completed

	February 19, 2024	Deployed the Ollama model on the Endeavour Cluster, enhancing the conversational assistant's ability to handle more complex queries.	Completed
	February 25, 2024	Launched a hosted web page featuring a user interface (UI) prototype, allowing users to interact with the conversational assistant and provide feedback.	Completed
	February 26, 2024	Met with clients to present our work and receive helpful feedback.	Completed
	February 29, 2024	Completed the first rough draft of the report.	Completed
	March 1, 2024	Aim to connect the RAG model to the frontend before Spring Break, enabling seamless integration between the conversational assistant's backend and frontend components.	Completed
Phase 2	March 12, 2024	Conduct the second presentation update on the progress of the project and outline the plans for personalized recommendations.	Completed
	March 13, 2024	Present updated assistant to clients.	Completed
	March 18, 2024	Have a meeting with Tom Stanley to discuss the in-depth flow of insurance plans to ensure the conversational assistant can provide more personalized answers.	Canceled
	March 20, 2024	Collect additional files and data about personalized choices to enhance the RAG structure and the conversational assistant's recommendation capabilities.	Completed
	March 22, 2024	Implement a mechanism for collecting feedback on the answers provided by the conversational assistant, both on the frontend and back end, to improve its performance.	Canceled
	March 25, 2024	Register the project with VTechWorks.	Completed
	March 29, 2024	Deploy a dedicated vector database to the Endeavour Cluster to support efficient retrieval and storage of data for personalized recommendations.	Completed
Phase 3	April 3, 2024	Begin researching potential visualization techniques and tools, and discuss their feasibility and relevance with the clients to ensure they align with the project's goals.	Canceled

	April 10, 2024	Outline and assign specific tasks to team members to start integrating visualizations into the conversational assistant.	Canceled
	April 13, 2024	Finish the final draft of the conversational assistant, incorporating all features, including the visualizations.	Completed
	April 15, 2024	Submit the final project paper, including documentation of the visualizations and their impact, to VTechWorks.	Completed
	April 16, 2024	Prepare and deliver the final presentation to stakeholders, showcasing the completed conversational assistant and its visualization capabilities.	Completed
	May 2, 2024	Make any fixes to the conversational assistant based on feedback from the final presentation.	Completed

Table 1: Task Timeline

Our project's timeline, in Table 1, details our progression through various phases, highlighting key tasks and their statuses. Phase 1 commenced on February 5, 2024, with client approval to proceed with the proposed plan, culminating on March 1 with the successful integration of the RAG model into the frontend. This phase included significant milestones such as deploying models on servers and clusters, and launching a UI prototype for user feedback.

Transitioning into Phase 2 on March 12, the focus shifted towards enhancing personalized recommendations. Despite a cancellation on March 18 to meet with a specialist for discussing insurance plan flows, the phase saw successful updates to the project, including data enhancement for the RAG structure and registration of the project with VTechWorks.

Phase 3 experienced adjustments with several task cancellations, including the development of visualization techniques, which had been set to further enrich the conversational assistant. Nevertheless, the phase concluded successfully with the submission of the final project paper and a comprehensive presentation to stakeholders on April 16. The project finished on May 1, addressing final tweaks post-feedback, ensuring the conversational assistant was fully operational and effective in aiding users with insurance-related inquiries.

8.2 Canceled Tasks

The tasks labeled as “canceled” in Table 1 represent areas where, following in-depth discussions with the client, the consensus was to enhance the quality of existing tasks rather than transitioning to new ones. In particular, our client recommended that instead of contacting Tom Stanley, we should focus our efforts on refining the segmentation of our data sources. This shift in focus is intended to provide a more solid foundation for the information our assistant utilizes. Additionally, we made the decision to not implement user feedback collection initially as we prioritized enhancing the accuracy of the assistant's responses. This focus was driven by the need to establish a solid foundation of reliability and

effectiveness in the assistant's core functionalities. Moreover, the decision to halt the development of visualizations came as a strategic choice to prioritize the assistant's capabilities in providing answers with precise citations. By improving the assistant's referencing abilities, we aimed to improve the reliability of the information it dispenses, ensuring that users receive not only accurate responses but also the contextual sources to support them. This nuanced approach underscores our commitment to continuous improvement and the delivery of a sophisticated, knowledge-driven user experience.

8.3 Problems and Solutions

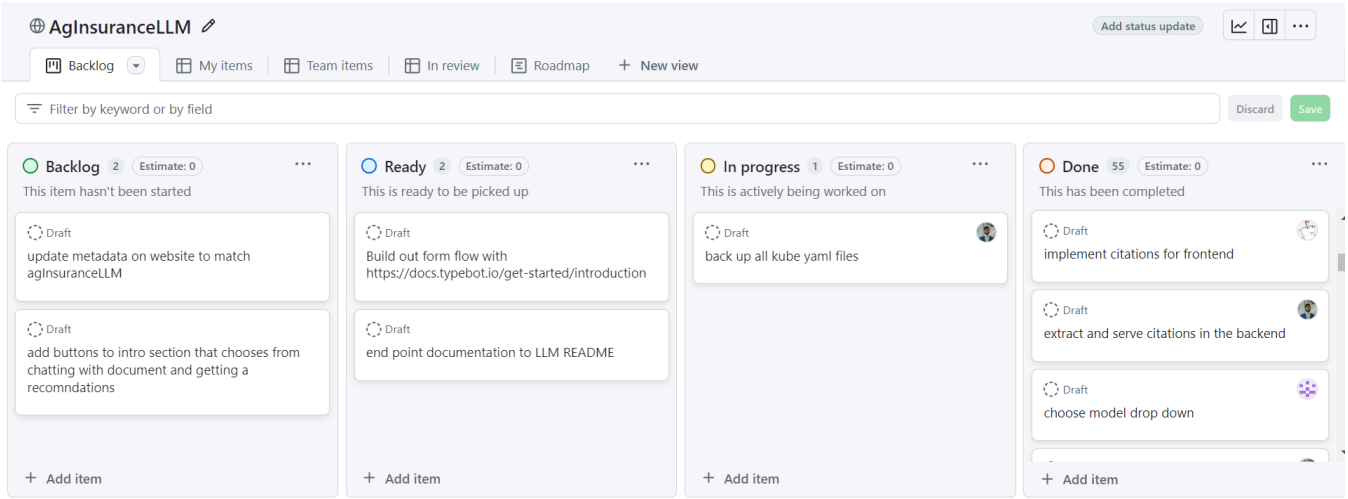


Figure 12: Kanban Board

During our project, our team faced several obstacles, with communication being a significant challenge. Effective communication is vital for the success of any collaborative effort. To facilitate this, we set up a GitHub repository and a Kanban board [31] (shown in Figure 10) to monitor our progress, and we used Discord [32] for daily communication. However, we encountered difficulties because we did not establish clear expectations for response times. This led to issues when some team members did not regularly check their Discord messages, which slowed down our progress. To address this, we started reminding each other to check messages and used pings when necessary to ensure timely communication.

For the frontend, we utilized an open-source user interface (UI) [3] as our foundation. We knew that we wanted to edit it to better fit our needs, so we needed to understand how the UI currently worked. Our team had varied experience with frontend languages, and we encountered a challenge as the frontend files were written in TypeScript [33], a language we were not familiar with. To overcome this, we dedicated time to gradually learn TypeScript and explore the library containing all the screens. This approach significantly aided us in modifying the files to align better with our needs.

As for the backend, the stream implementation posed some issues early on related to parsing streamed responses on the client-side as well as errors when related to the timing and size of streamed responses. To mitigate this, we use a null character to delimit different responses and ensure our generator function properly yields output without being delayed by extraneous logic. For prompt engineering, we make use of a system prompt to explicitly not make any promises or agreements with the user. This is a measure to avoid being legally liable for promises the assistant makes to the user.

There were early issues with hosting the language models on Virginia Tech's servers. One of these issues was properly configuring settings to ensure our different backend systems could communicate with each other. This ended up being a quick configuration fix. We learned a lot about developing APIs with Flask. We ran into many minor issues trying to get our API to work properly and support streaming. These issues were resolved by making use of logging from our API and our running containers. These logs can be viewed from pods from launch.cs.vt.edu and clicking view logs, as can be seen in Figure 13.

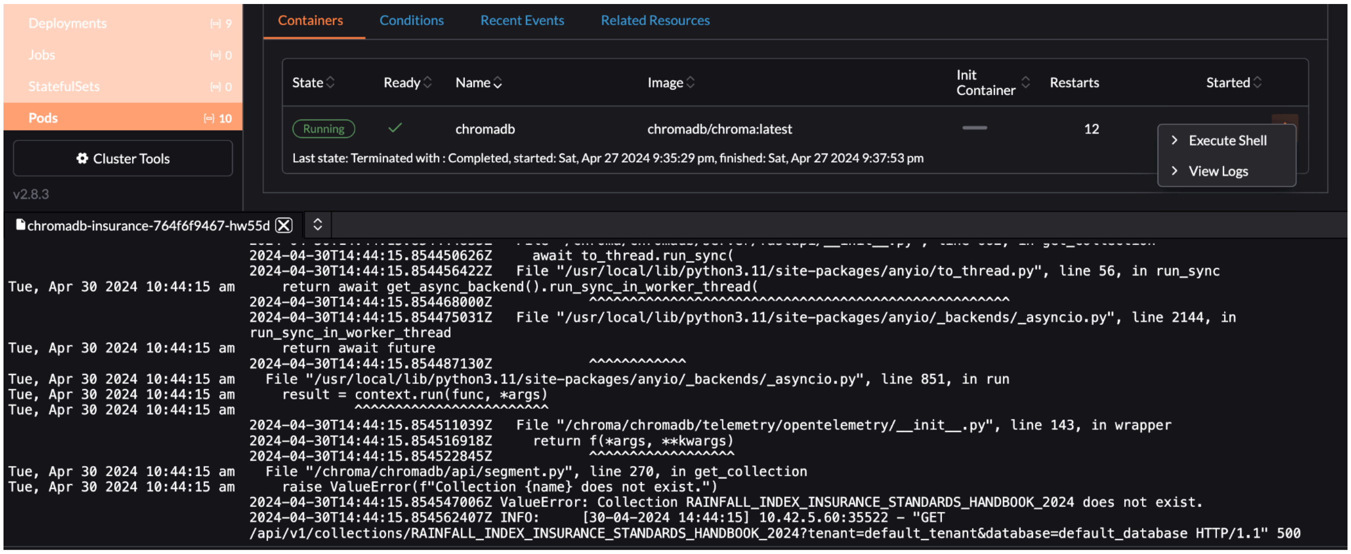


Figure 13: Viewing logs in pods

Working on this project was a good lesson in the importance of observability in software development because without logging errors we would have no way to trace and resolve our issues.

8.4 Further Development and Known Issues

The project would benefit from the initiation of a user feedback collection process to better understand the needs and experiences of users interacting with the system, a task planned for future phases. This process is crucial for continuous improvement and should be prioritized by the upcoming team.

Additionally, the project would be enhanced by scheduling a meeting with a subject matter expert to discuss strategies that could improve the assistant's accuracy and expand its knowledge base. Exploring the integration of visualizations within the conversational interface is another strategic area that would make interactions more informative and engaging.

Future enhancements should also include the development of more sophisticated query handling mechanisms and more personalized user interactions. Based on client feedback, the project recognizes the need for several specific improvements. These improvements should clarify to users that the

assistant is not a physical agent capable of performing tasks, refine responses to simple yes or no questions to be more direct, and include a confidence score with each response to inform users of the reliability of the information provided. Additionally, it would be beneficial to optimize the "Document" section of the citation response by preprocessing it to highlight relevant words found in both the query and citation, and starting the citation with a proactive suggestion, such as "I encourage you to look into the Rainfall Index Insurance Standards Handbook."

These enhancements, to be undertaken by future teams, are aimed at improving the functionality and user experience of the assistant, ensuring it meets the evolving needs and expectations of users effectively.

9 Acknowledgements

We would like to thank Dr. Elinor Benami for her support and input throughout this project. Additionally, we are grateful to Ramaraja Ramanujan and Mehmet Yardimci who were our main contacts. Their feedback and suggestions were vital in advancing our work. Lastly, we would like to thank Dr. Manoochehr Shirzaei and Dr. Fox for guiding us throughout the semester. Finally, we would like to acknowledge the College of Agriculture and Life Sciences at Virginia Tech for allowing us to work on this project.

10 References

[1] Natural Resources Conservation Service, "Range & Pasture," *nrcs.usda.gov*, April 4, 2024. Available: https://www.nrcs.usda.gov/conservation-basics/natural-resource-concerns/land/range-pasture. Accessed April 14, 2024.

[2] USDA Risk Management Agency, "RAINFALL INDEX INSURANCE STANDARDS HANDBOOK 2024 and Succeeding Crop Years," 2024. Available: https://www.rma.usda.gov/-/media/RMA/Handbooks/Coverage-Plans—18000/Rainfall-and-Vegetation-Index---18150/2024-18150-1-Rainfall-Index-Handbook.ashx?la=en. Accessed February 10, 2024.

[3] GitHub, "vercel/ai-chatbot," April 14, 2024. Available: https://github.com/vercel/ai-chatbot. Accessed April 14, 2024.

[4] OpenAI, "OpenAI API," 2024. Available: https://openai.com/blog/openai-api. Accessed February 15, 2024.

[5] shadcn, "shadcn/ui," 2024. Available: https://ui.shadcn.com/. Accessed February 22, 2024.

[6] Tailwind CSS, "Tailwind CSS - Rapidly build modern websites without ever leaving your HTML," 2023. Available: https://tailwindcss.com/. Accessed March 3, 2024.

[7] Radix UI, "Primitives," 2024. Available: https://www.radix-ui.com/. Accessed February 20, 2024.

[8] Phosphor Icons, "Phosphor Icons," 2024. Available: https://phosphoricons.com/. Accessed February 25, 2024.

[9] Next.js, "Routing: API Routes," nextjs.org, 2024. Available: https://nextjs.org/docs/pages/building-your-application/routing/api-routes. Accessed March 12, 2024.

[10] Mozilla, "HTTP request methods," MDN Web Docs, 2019. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. Accessed March 23, 2024.

[11] JavaScript, "Free JavaScript training, resources and examples for the community," Javascript.com, 2016. Available: https://www.javascript.com/. Accessed March 10, 2024.

[12] Ollama development team, "Ollama," 2024. Available: https://ollama.com/. Accessed February 18, 2024.

[13] Flask development team, "API — Flask Documentation (3.1.x)," flask.palletsprojects.com, 2024. Available: https://flask.palletsprojects.com/en/latest/api/. Accessed April 14, 2024.

[14] Amazon Web Services, Inc., "What is RAG? - Retrieval-Augmented Generation Explained," aws.amazon.com, 2024. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/. Accessed March 15, 2024.

[15] Google Developers, "Using OAuth 2.0 to Access Google APIs," Google Identity Platform, November 12, 2018. Available: https://developers.google.com/identity/protocols/OAuth2. Accessed April 21, 2024.

[16] Redis, "Redis," redis.io, 2023. Available: https://redis.io/. Accessed March 22, 2024.

[17] MIT-HAN Lab, "mit-han-lab/streaming-llm," GitHub, April 15, 2024. Available: https://github.com/mit-han-lab/streaming-llm. Accessed April 15, 2024.

[18] Nextra, "Prompt Engineering Guide," promptingguide.ai, 2024. Available: https://www.promptingguide.ai/. Accessed February 15, 2024.

[19] Chris Richardson, "Microservices.io," microservices.io, 2017. Available: https://microservices.io/. Accessed April 16, 2024.

[20] evchaki, "Vector Database," Microsoft, May 23, 2023. Available: https://learn.microsoft.com/en-us/semantic-kernel/memories/vector-db. Accessed February 21, 2024.

[21] Agricultural Insurance LLM Project Team in CS4624, "Website for Agricultural Insurance LLM Project," Hosted by Virginia Tech Dept. of Computer Science at aginsurancellms

m.endeavour.cs.vt.edu, 2024. Available: https://aginsurancellm.endeavour.cs.vt.edu/sign-in?callbackUrl=https%3A%2F%2Faginsurancellm.endeavour.cs.vt.edu%2F. Accessed April 14, 2024.

[22] Json.org, "JSON," 2019. Available: https://www.json.org/. Accessed March 5, 2024.

[23] Vercel, "Vercel KV," 2024. Available: https://vercel.com/docs/storage/vercel-kv. Accessed April 15, 2024.

[24] Mehmet Oguz Yardimci, "LLMs for Ag. Validation Questions," 2024. Available: https://docs.google.com/document/d/1C9BSmslSrtJzNRFQwtygWR_Jga7KedOQ2Ri7z-ju1MM/edit. Accessed April 14, 2024.

[25] Saketh Rajesh, "LLM Response Table for Ag. Validation Questions" 2024. Available: https://docs.google.com/spreadsheets/d/1AT2IBsHP_71TJpTZt7uaxbdjFnKtvpRkcGWbT6-MPhI/edit#gid=1050921723. Accessed April 14, 2024.

[26] Atlassian, "What is Agile?," 2023. Available: https://www.atlassian.com/agile#:~:text=The%20Agile%20methodology%20is%20a. Accessed March 20, 2024.

[27] Mike Rehkopf, "Scrum Sprints," Atlassian, 2019. Available: https://www.atlassian.com/agile/scrum/sprints. Accessed April 22, 2024.

[28] Docker, "Enterprise Application Container Platform," docker.com, 2024. Available: https://www.docker.com/. Accessed February 23, 2024.

[29] GitLab, "Sign in · GitLab," git.cs.vt.edu, 2024. Available: https://git.cs.vt.edu/users/sign_in. Accessed April 14, 2024.

[30] Kubernetes, "Cluster Architecture," kubernetes.io, 2024. Available: https://kubernetes.io/docs/concepts/architecture/. Accessed March 29, 2024.

[31] GitHub, "About projects (classic)," GitHub Docs, 2024. Available: https://docs.github.com/articles/about-project-boards. Accessed April 14, 2024.

[32] Discord, "Discord — A New Way to Chat with Friends & Communities," 2024. Available: https://discord.com/. Accessed April 19, 2024.

[33] Microsoft, "TypeScript - JavaScript that scales.," Typescriptlang.org, 2015. Available: https://www.typescriptlang.org/. Accessed March 25, 2024.