RAG Model: Team Approach and Challenges

Our team's primary approach for this project was centered around utilizing the Streamlit library in Python to provide a hosted user interface that runs all our Python files for our RAG model. Streamlit, chosen for its ease of use and professional-looking interface, enabled us to develop a front-end without the need for HTML, CSS, or JavaScript. The core file, labeled **app.py**, serves as the user-facing part of the application. This is what the user sees and interacts with, providing an intuitive and simple way to engage with the model.

Once a file is uploaded via this interface, it is passed to **parsers.py**, where we process and convert the file's content into strings. What makes parsers.py versatile is its ability to adapt to different file types such as CSV, PDF, JSON, or TXT. Regardless of the file type, this module efficiently parses the file and converts it into a string that our code can process. This step ensures that our model can handle multiple file formats seamlessly, allowing the subsequent stages of the pipeline to handle the data uniformly.

Following the parsing step, the data is passed to **embeddings.py**, where it is transformed into embeddings or vectors. These vectors are essential for our model, as they provide the structure the model needs to analyze and interpret the data. By vectorizing the input, the model can assign weights, biases, and other attributes, effectively learning from the input data. This vectorization process, while seemingly simple, is foundational to how the model interprets complex inputs. Initially, we faced some difficulties when using two different embedding models that were creating two conflicting data shapes. However, after troubleshooting, we unified our approach to rely on a single SentenceTransformer model, streamlining the embedding process.

Once vectorized, the data is passed along with the user input to **chat.py**, which communicates with the Ollama server. This is where the magic happens—Ollama processes the input data and the vectors and generates a response. It breaks down the user's input, analyzes the vectors, and returns a response tailored to the input. The response generated by Ollama is then sent back to **app.py**, where it is displayed in the Streamlit chat window, creating a smooth and interactive user experience.

Team Dynamics and Roles

Our team worked hard to delegate tasks and ensure active engagement from each member. However, one of the main difficulties we faced was getting consistent team engagement and communication. It was challenging to find times where everyone could meet, given our varying schedules, and even more difficult to assign roles that

suited everyone's ability level. While we did our best to ensure everyone had a role, it often felt like we were stretching to give each team member a significant task. In the end, there was a clear division of labor: one team member primarily focused on the coding aspect, while others contributed through research, analysis of best practices, and providing input on documentation.

Challenges with Tools and Documentation

A significant challenge we encountered was the lack of documentation when using Streamlit with models like Ollama. Streamlit has resources available for working with OpenAI's models, but much less support exists for integrating with other models, like Ollama. This led to some trial and error as we figured out how to best connect our chatbot to Ollama's API. Additionally, we faced technical issues with initializing vectors, a problem that stemmed from initially using two different embedding models, which we later resolved by switching to a single, consistent model for embeddings.
Hardware limitations also became apparent during our development process. Running large files or handling complicated queries posed performance challenges, which we suspect were due to either the computational limits of our hardware. After further development and testing however, the model seems to perform well. The app is also currently hosted on one of the more powerful computers of the team, and online version runs smoothly. Furthermore, the size of the Ollama 3.1 model may have contributed to these performance issues. We considered the possibility that a smaller language model (LLM) might be more efficient and easier to work with, especially when running on machines with less computational power.

Another challenge was implementing the AI agent. At this moment in time, we do not have an agent making decisions on how to process files based on what kind of file is input. Due to the complexity of the project, and other limitations that we were working with, we opted to instead implement a simple if-else statement that handles file types, and processes them correctly. This allowed us to simply development and focus on other parts of the project. Ultimately, this allowed us to develop a RAG model that runs well, and still has room for future development and improvement.

Reflections on the Project

Despite the technical and logistical challenges, the model worked well once everything was in place. The process of creating this Retrieval-Augmented Generation (RAG) model was not only a cool exercise but also an opportunity to explore how we can use different tools to generate meaningful responses based on complex input. It also provided us great experience and insight into how we can build programs that

implement AI models in the future. The Safe Places team believes that this model has the potential to extend beyond this exercise. For instance, we are considering using this model for **Brades' Place**. By connecting this model to Brade's Place's website and tweaking the design with some HTML, CSS, and JavaScript, we could provide a more user-friendly interface tailored to the needs of our our Safe Spaces project.

One of the next steps that could be considered for making this applicable to Brades' Place would be hosting the application on a machine with stronger computational capabilities. Given the hardware limitations we encountered during development, we believe that serving the model from a higher-performance machine would significantly improve performance, allowing the model to handle larger files and more complex queries without affecting the user experience..

Conclusion

In conclusion, building this model as part of the Safe Places project was a valuable learning experience for the entire team. We navigated issues related to team dynamics, documentation gaps, and hardware limitations, yet we managed to create a working system that is fun to use and informative. Moving forward, we plan to refine and optimize this model, both for our own learning and for potential real-world applications, such as its integration into Brade's Place.

Contributions:

Brancen Clement: Research, Troubleshooting, Site Hosting

Brandon Lee: Presentation Creation

Brippney Vargas: Research, Troubleshooting, Development

Cade Anderson: Paper Author

Fischer Wells: Development/App Production, Organized team meetings and roles, Research, Troubleshooting, Paper Co-Author

Keomony Mary: Research, Troubleshooting

Mingi Song: Presentation Delivery

Tyler Henry: Presentation Delivery, Presentation Creation