A. Architecture and Design Decisions

The goal of this project was to create an ETL-RAG pipeline which could act as an “expert” on the COVID-19 pandemic using data and other sources from online. Therefore, sources of data were obtained online consisting of csv files with regional COVID-19 metrics over the course of the pandemic, as well as PDFs containing overviews of the pandemic from a post-pandemic perspective. The csv chunking strategy was to bring in 20 lines of a given csv and load each line as a string. The chunks of data were then stored together as text, with metadata for debugging. Once documents are loaded in, the pipeline can be used to build a FAISS vector index and store that vector to the disk. The retrieval script is then used to embed queries, search the FAISS index for the top-k nearest vectors, then returns matching documents along with similarity scores. We also uploaded the FAISS index for ease of use so that future users don’t need to rebuild it.

B. Retrieval Quality and Failure Analysis

For the retrieval quality and failure analysis we manually questioned the retriever with a set of benchmark questions and inspecting the top-k retrieved chunks rather than interpreting it from just the answers. For example, when we asked “What trends do we see in COVID cases in America?”, the top results came from our main cleaned dataset and had multiple rows specifically from the United States data with cases and new_cases columns over time. Similarly, for a query that’s more specific such as “Which states had some of the highest case counts in 2021?”, the retriever was unable to discern between states since the data provided was only split between countries rather than more specific regions. On the other hand, when we asked “What were the most common vaccine side effects in the US?,” the top-K chunks still mentioned the US and Covid, however it only contained case and death counts and no information about vaccines or their side-effects. In this scenario, the retriever was “on-topic” but could not surface relevant evidence to actually answer the question effectively, so it was marked as a failure. The final question was a vague query such as “Was Covid bad?” The retriever returned generic case rows and did not explicitly address the impact of Covid. Overall, the experiments showed that the retriever works when the query matches the tables’ structure, but more broad and vague questions may provide some issues to the retriever.

C. API Challenges

During our work process, one of the biggest challenges we faced was dealing with header errors in our API. At first, these errors were hard to understand because the system often failed without giving clear error messages. In many cases, the problem was caused by missing or incorrect headers such as content type. Another difficulty that we encountered is knowing the difference between a GET function and a POST function, knowing that the POST function does not directly create web access to the api is crucial for this project. Another major challenge happened when we were setting up the API in a virtual environment. Different team members had different Python versions and packages installed on their computers, which caused the code to behave differently for each person. Some team members experienced

import errors, while others had issues running the server at all. To solve this problem, we created a virtual environment ".venv" and made sure everyone used the same Python version. We also created a requirement file so all required packages could be installed easily and consistently prior to using the API. To resolve the issues our team faced, we tested our system step by step. We started with simple API endpoints to make sure the server was running correctly before adding more complex features. Overall, these challenges taught us the importance of clear setup instructions, consistent environments, and careful debugging, which helped improve our workflow and the reliability of our project.

D. Team Collaboration and Process

Over the course of the project, different team members worked on different portions to spread the work in a fair way and ensure that tasks were completed. Micaiah's role was to find sources of data, clean them, put them into smaller sizes to avoid overloading memory, then compile them into a list. He then was tasked with creating a system to ingest the files for use in the system. Lakshay worked on much of the middle portions of the ETL-RAG system. He was able to integrate the large language model (LLM) into the rag pipeline by creating different functions such as the generate function and the answer function. Finally, Peter's role was to put together the API for users to query to. When the api was first created, he tested it out with static variables, just to make sure the api is running and working correctly. After making sure that the api is running smoothly, the group started calling in the model to make the api call return responsive answers based on the input questions.