RAG on FHIR

Using FHIR in conjunction with Generative AI and LLMs in Healthcare

Sam Schifman · Follow
7 min read · Dec 3, 2023

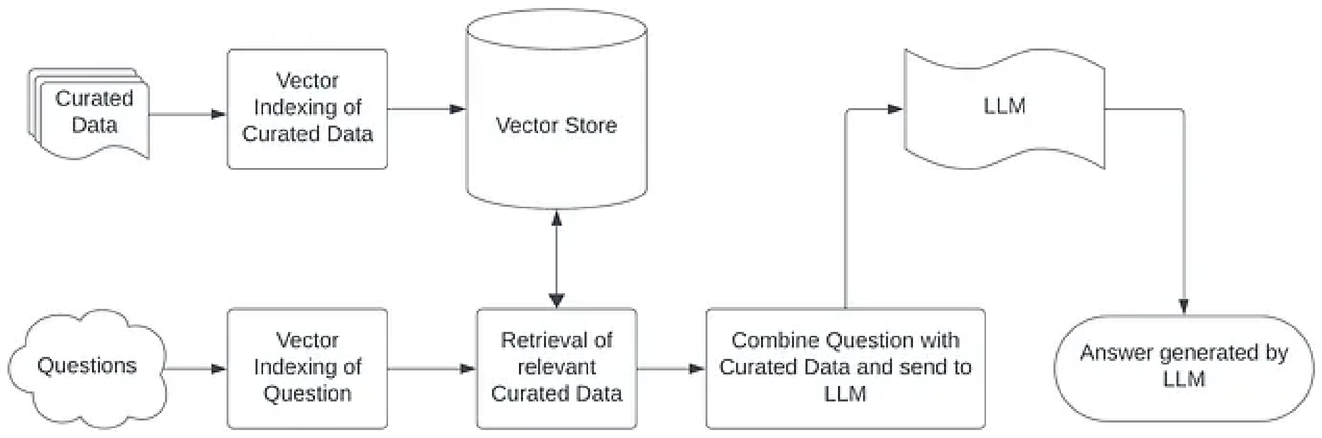
Listen Share More

Image generated using Midjourney

Generative AI, and the LLMs at its core, have provided a massive leap forward for AI. Their ability to perform Question and Answer tasks is remarkable. However, they still have a number of flaws. LLMs are still prone to hallucinations, where they make up information that seems very reliable. They are also limited to the information provided to them at the time they were trained. For example, ChatGPT is limited to information publicly available before 2011. A lot of work has gone into figuring out how best to get around these limitations.

What is RAG

One technique people are using to get around the limitations of LLMs is Retrieval-Augmented Generation (RAG). You can find a great article and YouTube video on RAG from IBM here. In brief, RAG uses a store of curated data to assist the LLM in answering the question. The process looks something like this:


Curated Data
Vector Indexing of Curated Data
Vector Store
LLM
Questions
Vector Indexing of Question
Retrieval of relevant Curated Data
Combine Question with Curated Data and send to LLM
Answer generated by LLM

Here the curated data is put into a vector store which allows the RAG to find data related to the questions asked. It can then combine that data with the question when it prompts the LLM. The LLM then has both its own knowledge and the curated data available as it synthesizes its response.

LLMs in Healthcare

Healthcare is a complex world, much of which is opaque to most people. There is a great deal of highly specific terminology, which often varies from one specialty to another. And the field is ever expanding and changing. Even healthcare professionals struggle to keep up with all the latest information. LLMs present an opportunity to cut through the complexity and make information about people's health much more accessible.

However, using LLMs in healthcare presents its own special challenges. One challenge is that most people don't want an LLM trained on their personal health data. For understandable reasons, people only want the doctors who are treating them to have access to their records. Because we don't really know how LLMs work under the covers, it is very hard to say that we could completely restrict it from divulging data to people who aren't supposed to have it. For example, we might well put in security checks that prevent a user from asking a question naming a patient they don't have access to, but we might not protect against that same person asking a question about an unnamed patient who happens to fit certain criteria. It is very difficult to create a security system that would protect against every permutation of bad question.

RAG provides an excellent solution to this. In RAG, we can limit access to the curated data, so that any question can only include information the user is allowed to see. RAG also means that we can provide the LLM with the most up-to-date information about the patient, because it is far easier to update the Vector Store than retrain the LLM.

What is FHIR

Fast Healthcare Interoperable Resources (FHIR) is the newest standard from HL7. FHIR defines a format for how health information can be transmitted between healthcare organizations. It also provides standard for APIs, security, and processes for how that information is exchanged. While FHIR is still relatively new, it has been named in mandates from the US Government and others as the standard to meet new requirements around interoperability.

FHIR breaks down patient health data into a number of small chunks called, "resources." For example, a particular allergy that a patient has to Penicillin would be represented in a single AllergyIntolerance resource or a particular blood pressure reading would be represented as a single Observation resource. Resources for a patient are linked together by being associated with a Patient resource that capture demographic information, like name and address, for the patient.

The problem with FHIR, like most interoperable health data, is that it was designed to be machine friendly, not human readable. If you just look at a collection of FHIR it is very hard to tell anything important about the patient. But LLMs can help with that.

RAG on FHIR

Because FHIR already breaks up the health information into small and specific chunks, it is perfect for RAG. Once we have the right set of FHIR resources that are associated with the question being asked, we can provide them to the LLM and it can use that data to answer our question. The challenge then becomes how to get the FHIR into a form that can be easily vectorized, i.e. put into the Vector Store.

WARNING: *From here on in we are going to get technical and talk about vectors, embedding, and other concepts associated with Generative AI. I am not going to attempt to define all these terms, because doing so would take far too long. If you aren't here for the technical stuff, you might want to skip to the end.*

A naive approach might be to simply throw the FHIR, which is in JSON, at the embedding model and stick the resulting vectors in the Vector Store. However, JSON represents the hierarchy of data in a structure based on Object Oriented Program paradigms; objects have attributes, which may point to other objects, lists, or values. Most embedding models where primarily trained on text, where this sort of hierarchy is represented very differently. In text, we have sentences which provide relationships between subjects (objects) and their adjectives (values), more or less. (**Note**: *This is not the only relationship that can be captured in a sentence, but it is the primary one I am interested in here.*)

What I propose, and what you can see in the associated Jupyter Notebook, is to flatten the FHIR into pseudo-sentences. These are not real sentences, as in your language teacher wouldn't grade them well, but they provide the context in a way that is more friendly to the embedder. This done by crawling the tree of JSON and keeping track of the all the attribute names as we go until we hit a value. This provides what we call a path, where all the attribute names are strung together. In my code, they are separated by spaces so that the embedder can see them as separate words. Along the way, I also break multiword attribute names into multiple words. Finally, I create a "sentence" with the path, the word "is," and the value.

For example:

```
{
   "resourceType": "Observation",
   "code": {
   "coding": [{
       "code": "8302-2",
       "display": "Body Height"
     }
   ]
 },
 "valueQuantity": {
     "value": 123.6,
     "unit": "cm",
    }
}
```

Becomes:

```
Resource type is Observation. Code coding 0 code is 8302-2. Code coding 0
display is Body Height. Value quantity value is 123.6. Value quantity unit
is cm.
```

Note: *This example is intentionally simplified for illustration purposes.*

While this form is far more easily consumed by the embedder and therefore easier to put in the Vector Store, it is still lacking in needed context. Almost all FHIR resources provide a pointer to the Patient resource, but this may not make sense to the embedder. I took one additional step, which was to find the Patient resource first and extract the patient's name from it. I then associated that data with the other data by providing it as a line above the other information.

If, in our example, the patient's name is Jane Smith, what I pass to the embedder is:

```
Patient first name is Jane. Patient last name is Smith.
Resource type is Observation. Code coding 0 code is 8302-2. Code coding 0
display is Body Height. Value quantity value is 123.6. Value quantity unit
is cm.
```

This way I can load multiple patients into the Vector Store and still ask questions about specific patients.

In the linked Jupyter Notebook you can see how all this works. I used Synthea to generate two synthetic patients (not shown). Synthea produces one JSON file per patient that contains all the resources for the patient in a FHIR Bundle resource. The code loads those files, breaks up the resources into individual text files, loads those text files into a Vector Store, and then asks questions of an LLM including the information from the Vector Store in the prompt. For the RAG I used the LlamaIndex framework and for the LLM I used Llama 2 running locally using Ollama. LlamaIndex allows for multiple strategies for how the data from the Vector Store is combined with the question to be sent to the LLM. In the code you will see that I experimented with multiple strategies.

My goal here was to build a framework that could effectively flatten FHIR and put it into a Vector Store for RAG. I did not intend to do a formal experiment comparing it to other methods or evaluating its overall effectiveness. If you would like to do that, please do, I would be happy to help and love to learn what your results are.

Conclusion

I was able to convert FHIR into a form that was easy to load into a Vector Store. I was then able to use an LLM to answer questions about that data. However, as I said above, it was not my intent to do any sort of formal experiment. I also don't believe this is the only way to vectorize FHIR. I would love to hear about other people's attempt do RAG using FHIR and other ideas on how to vectorize FHIR.

Note: *I am aware that the excellent team at FLY HEALTH published an article about vectorizing FHIR and I took some inspiration from the code they have made public.*

The code is available on GitHub. Also, there is now a video with a live demo on YouTube.