Member-only story

A Real-life Use-case With Codes: Build an AI-powered Research Assistant With Llama3, LlamaIndex, Qdrant, and FastAPI

Lorentz Yeung · Follow

Published in Towards AI · 9 min read · Aug 24, 2024

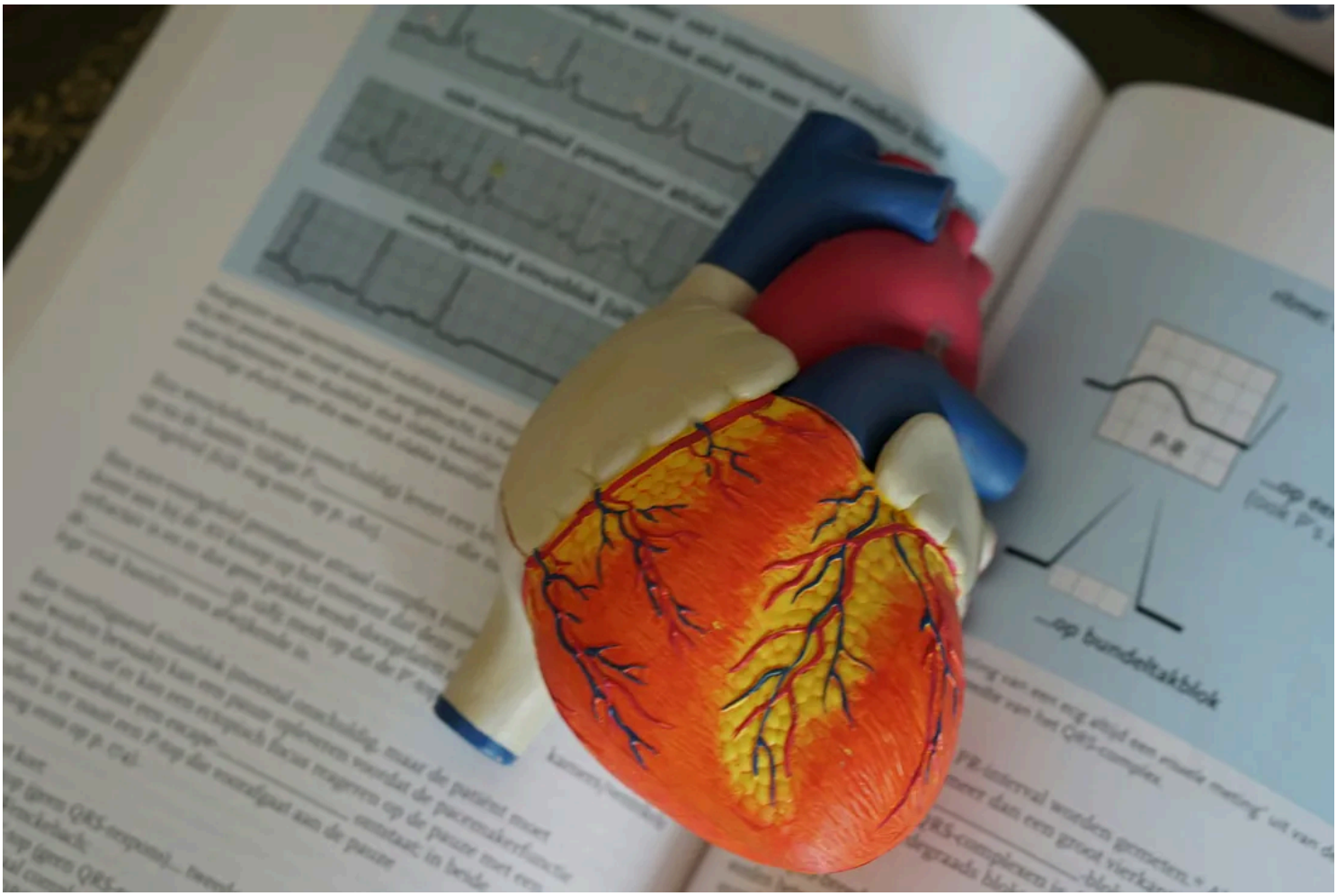


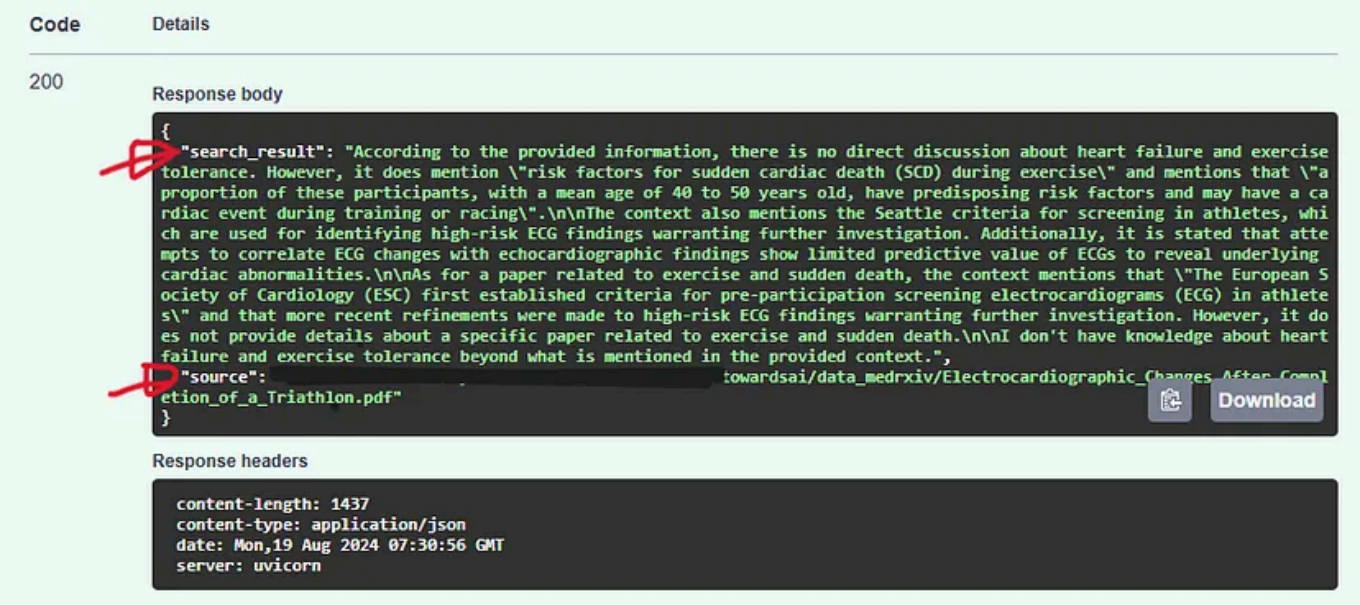
Photo by Robina Weermeijer on Unsplash

My wife, a senior nurse at St Thomas' Hospital, often faces the challenge of finding time to thoroughly research and review relevant literature to produce high-quality work. As a data scientist and AI engineer, I asked myself how I could solve this problem using my skills.

Therefore, I came up with this idea — build an AI API app that stores the latest published research papers downloaded from medrxiv.org, and then she can query from her Mac anytime.

Below is the result from the AI powered research assistant through the API. The result from the AI are presented in a user-friendly format, with the "research_result" showing the answer retrieved from the database and

"source" indicating the specific PDF it came from. This is especially useful for writing footnotes and bibliographies.



Through the intranet, she can query the database from her computer and receive answers directly from the AI. Photo created by the author.

Without further ado, I am going to walk you through the development process of this project, which is fully functional as of August 19, 2024. The code is available on my GitHub repository.

The development was fairly straightforward, following a regular RAG (Retrieval-Augmented Generation) LLMOp approach. First, I scraped data online, processed it through embeddings, and then developed the application. After deployment, the app was tested, fine-tuned, and versioned several times before reaching the final version you see today.

1. Create our own LLama3 model with Ollama.

For a quick MVP, I didn't build a LLaMA3 model from scratch or fine-tune one. Instead, I created a custom LLaMA3 model within the Ollama

environment using some simple engineering. Below is an excerpt from my Modelfile:

```
FROM /usr/share/ollama/.ollama/models/blobs/sha256-6a0746a1ec1aef3e7ec53868f220f

# My param configurations
# Number of Context Tokens controls the maximum number of tokens (words or subwo
PARAMETER num_ctx 4096

# temperature parameter controls the randomness or creativity of the model's res
PARAMETER temperature 0.75

# top_k limits the number of potential next tokens (words or subwords) the model
# top_k 60 means the top 60 most likely options
PARAMETER top_k 60

# top_p controls the model to consider only the most probable next tokens that c
PARAMETER top_p 0.90
TEMPLATE """{{ if .System }}<|start_header_id|>system<|end_header_id|>

{{ .System }}<|eot_id|>{{ end }}{{ if .Prompt }}<|start_header_id|>user<|end_hea

{{ .Prompt }}<|eot_id|>{{ end }}<|start_header_id|>assistant<|end_header_id|>

{{ .Response }}<|eot_id|>"""
PARAMETER stop "<|start_header_id|>"
PARAMETER stop "<|end_header_id|>"
PARAMETER stop "<|eot_id|>"
PARAMETER stop "<|reserved_special_token"


SYSTEM """
As a personal assistant for medical researchers, your task is to analyze the pro

Your goal is to provide researchers with 'aha' moments by explaining complex top

Instructions:

Focus on: Summarizing key insights and methodologies that are directly relevant
Ensure: Your responses are concise and easy to understand, emphasizing clarity a
Avoid improvisation: Only use the information provided in the research papers.

If the provided context does not contain the necessary information to answer a q
```

```
"""
```

The parameters control various aspects of the model's behavior:

- **num_ctx**: Controls the maximum number of tokens the model can use.
- **temperature**: Controls the randomness or creativity of the responses.
- **top_k**: Limits the number of potential next tokens considered by the model.
- **top_p**: Limits the model to consider only the most probable next tokens.

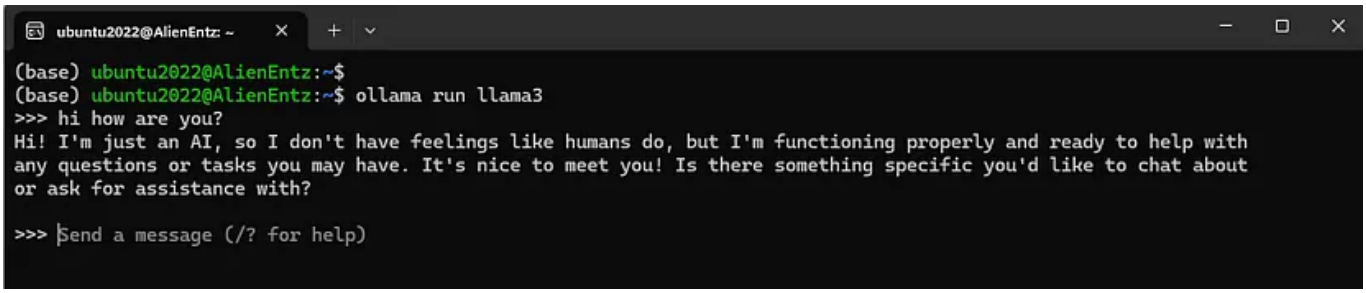
The TEMPLATE is important, otherwise the model doesn't know where to stop.

The SYSTEM is the prompt engineering. It instructs the AI what to do, and what it should not do. For more information, please read Ollama's official document .

But before we apply this Modelfile, we need to install Ollama and Llama3 first. If you don't have Llama3 and Ollama yet, please google it, there are countless tutorials for how to achieve online.

You can test your Llama3 in your Terminal. Just enter the command ollama run llama3. You will see something like below if it is successful.

```
ollama run llama3
```



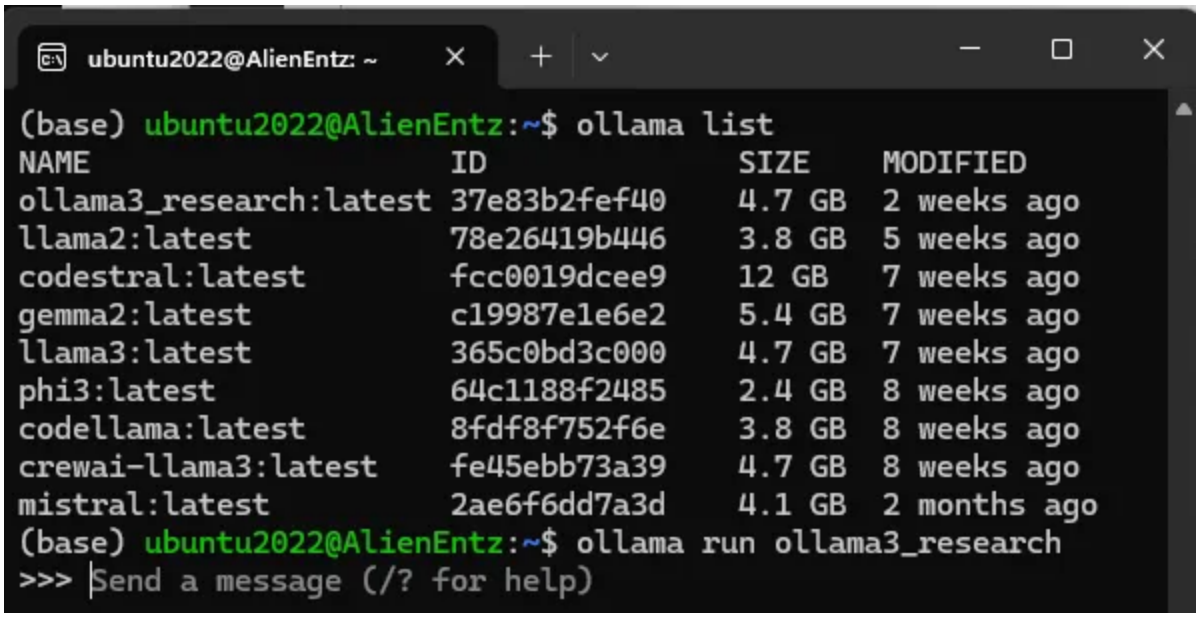
If you have your llama3 installed successfully, you can expect a result like this if you run the above command. Picture created by the Author.

Now cd to your work folder or to the folder your Modelfile sits. Then run this command:

```
ollama create ollama3_research -f Modelfile
```

Then run this command to try this newly created pretrained model.

```
ollama run ollama3_research
```



Congratulations! You have your own research model which is slightly engineered for research!

2. Set up a Qdrant docker instance

Before indexing and storing data, we need to build our database. Fortunately, this step can be easily accomplished by using the following command in our Terminal.

```
docker run -p 6333:6333 qdrant/qdrant
```

This command runs a Docker container using the `qdrant/qdrant` image and maps port 6333 of the container to port 6333 of the host machine. This setup allows us to access the Qdrant instance via `http://localhost:6333`.

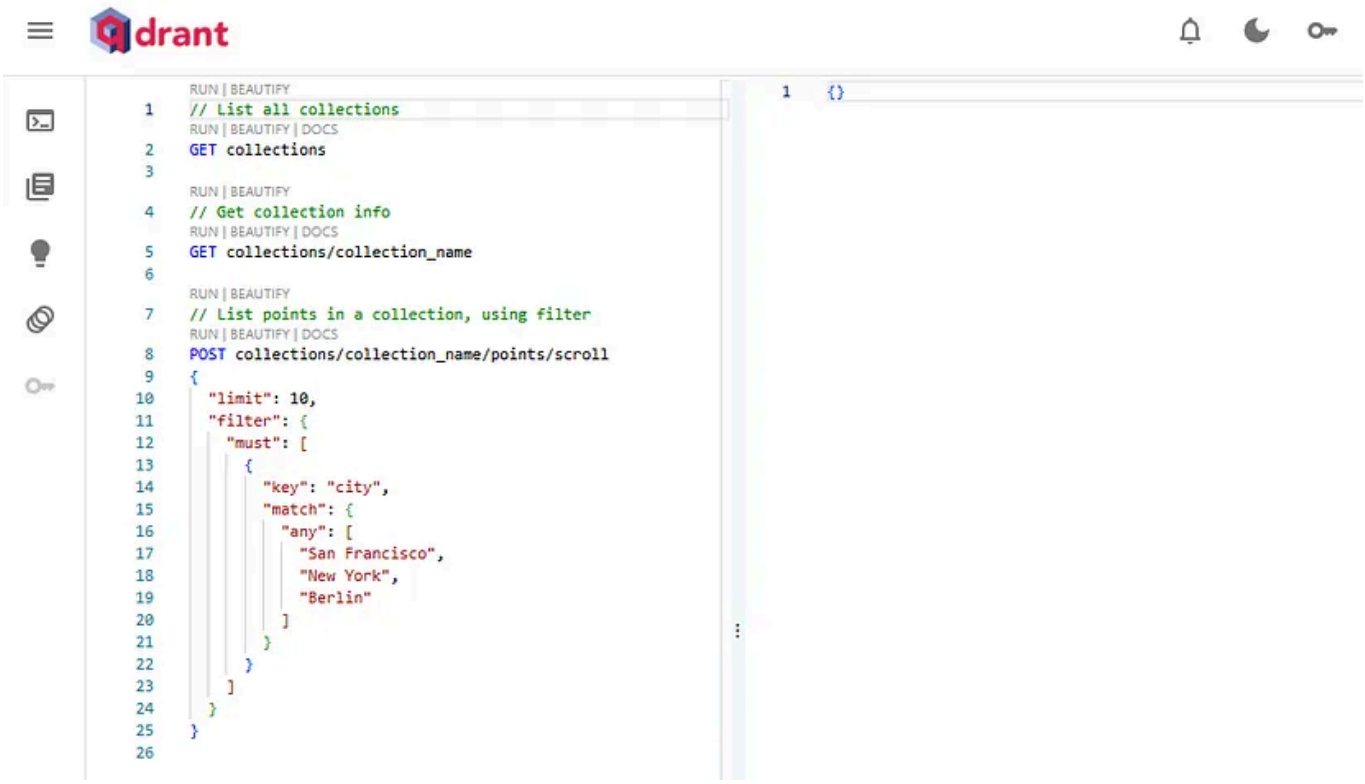
However, we want the data stored in Qdrant to persist even after the Docker container is stopped.

```
docker run -p 6333:6333 -v /mnt/x/data/217_local_rag_api/qdrant_storage:/qdrant/
```

This command does the same as the previous one but also includes a volume mount (-v /mnt/x/data/217_local_rag_api/qdrant_storage:/qdrant/storage:z). The volume mount ensures that the data stored in Qdrant persists even after the Docker container is stopped. The `/mnt/x/data/217_local_rag_api/qdrant_storage` directory on the host machine is mapped to the `/qdrant/storage` directory inside the container. This way, any data stored by Qdrant will be saved to the host machine's `/mnt/x/data/217_local_rag_api/qdrant_storage` directory, making it persistent.

The :z option tells Docker to relabel the files in the volume so that they can be accessed by the container, ensuring that SELinux does not prevent the container from accessing the mounted volume.

After applying the above command, our database is ready. We can access our Qdrant UI at http://localhost:6333/dashboard.



Qdrant UI at http://localhost:6333/dashboard. Picture created by Author.

3. Download our data from medrxiv

Now we are ready to download the medical papers from medrxiv. Here's a Python class I wrote to handle the download. For detailed API instructions, please refer to the official medrixiv webpage.

```
class Data:
    def __init__(self, config):
        self.config = config

    def create_data_folder(self, download_path):
        if not os.path.exists(download_path):
            os.makedirs(download_path)
            print("Output folder created")
        else:
            print("Output folder already exists.")

    def download_papers(self, search_query, download_path, server="arxiv", start
        self.create_data_folder(download_path)
```

```
    if server == "arxiv":
        client = arxiv.Client()

        search = arxiv.Search(
            query=search_query,
            sort_by=arxiv.SortCriterion.SubmittedDate,
        )

        results = list(client.results(search))
        for paper in tqdm(results):
            if os.path.exists(download_path):
                paper_title = (paper.title).replace(" ", "_")
                paper.download_pdf(dirpath=download_path, filename=f"{paper_
                print(f"{paper.title} Downloaded.")

    elif server == "medrxiv":
        if not start_date or not end_date:
            print("Error: 'start_date' and 'end_date' are required for medRx
            return

        # Construct the API URL
        api_url = f"https://api.medrxiv.org/details/{server}/{start_date}/{e

        response = requests.get(api_url)
        if response.status_code != 200:
            print(f"Failed to retrieve data from MedRxiv API. Status code: {
            return

        data = response.json()

        if 'collection' not in data or len(data['collection']) == 0:
            print("No papers found with the given search query.")
            return

        papers = data['collection']

        for paper in tqdm(papers):
            title = paper['title'].strip().replace(" ", "_").replace("/", "_
            pdf_url = f"https://www.medrxiv.org/content/{paper['doi']}.full.
            print(f"Attempting to download {title} from {pdf_url}")

            try:
                pdf_response = requests.get(pdf_url)
                if pdf_response.status_code == 200:
                    pdf_path = os.path.join(download_path, f"{title}.pdf")
                    with open(pdf_path, 'wb') as pdf_file:
                        pdf_file.write(pdf_response.content)
                    print(f"{title} Downloaded to {pdf_path}.")
                else:
                    print(f"Failed to download {title}. Status code: {pdf_re
```

```
                except Exception as e:
                    print(f"An error occurred while downloading {title}: {e}")

        else:
            print(f"Server '{server}' is not supported.")

    def ingest(self, embedder, llm):
        print("Reading pdf from the specified directory...")
        documents = SimpleDirectoryReader(self.config["data_path"]).load_data()

        print("Initializing Qdrant client...")
        client = qdrant_client.QdrantClient(url=self.config["qdrant_url"])
        vector_store = QdrantVectorStore(
            client=client,
            collection_name=self.config["collection_name"]
        )
        storage_context = StorageContext.from_defaults(vector_store=vector_store

        # Use the Settings object correctly
        service_context = ServiceContext.from_defaults(
            llm=llm, embed_model=embedder, chunk_size=self.config["chunk_size"]
        )
        print("Start embedding the documents and storing the resulting vectors i
        index = VectorStoreIndex.from_documents(
            documents, storage_context=storage_context, service_context=service_
        )
        print(
            f"Data indexed successfully to Qdrant. Collection: {self.config['col
        )
        return index
```

Next, we will actually download and ingest the pdf to our Qdrant database.

Regarding the embedding models, there are plenty in huggingface. For this particular project, we'll use `all-MiniLM-L6-v2`, which has 58.3M download, and 2.1K Likes.

Before running the download, embedding, and ingest functions, remember to manually set the params.

```
# Manually set the parameters
query = "heart failure exercise tolerance"  # The topic my wife is interested in
ingest = True  # Set to True if you want to ingest data
server = "medrxiv"  # Set server to "medrxiv" or "arxiv"
start_date = "2003-07-31"  # Required for medRxiv, it is useful becacuse you don
end_date = "2024-08-01"  # Required for medRxiv, it is useful becacuse you don't
```

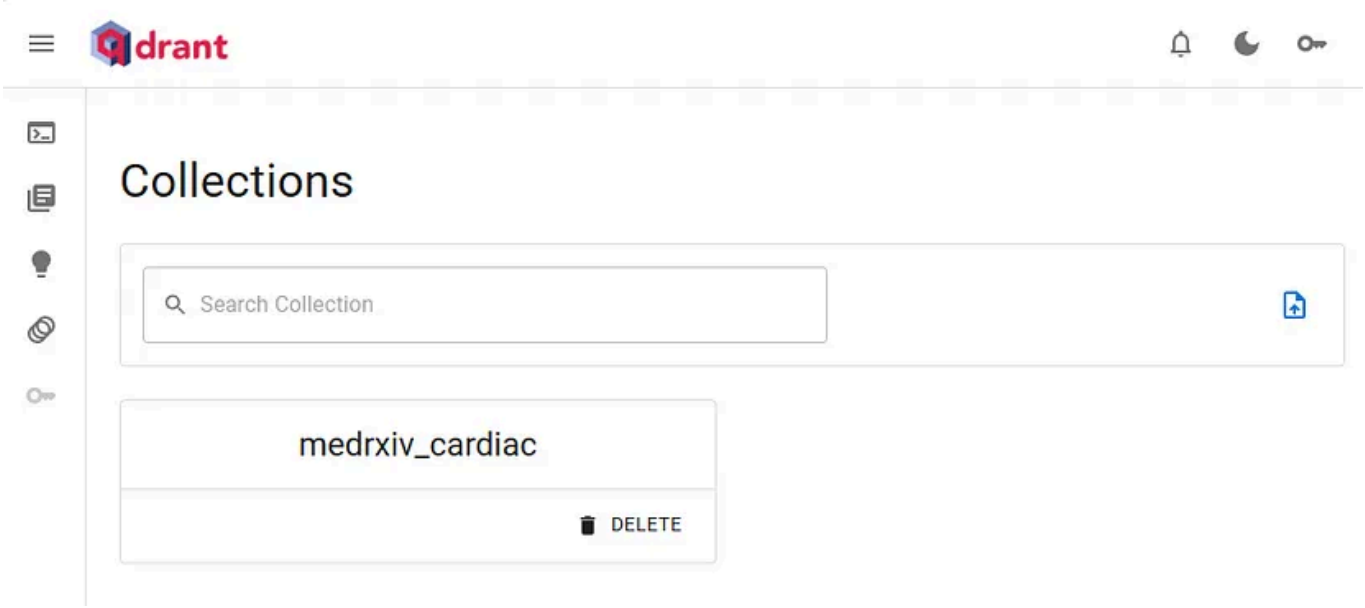
For more information, please visit the official llamaindex webpages.

https://docs.llamaindex.ai/en/stable/api_reference/storage/vector_store/qdrant/

https://docs.llamaindex.ai/en/stable/examples/vector_stores/QdrantIndexDemo/

```
  0%|          | 0/100 [00:00<?, ?it/s]
Attempting to download Molecular_profiling_of_neonatal_dried_blood_spots_reveals_changes_in_innate_and_adaptive_immunity_following_fetal_inflammatory_response from
https://www.medrxiv.org/content/10.1101/19000109.full.pdf
  1%|          | 1/100 [00:03<05:33,  3.37s/it]
Molecular_profiling_of_neonatal_dried_blood_spots_reveals_changes_in_innate_and_adaptive_immunity_following_fetal_inflammatory_response Downloaded to
medrxiv_papers/Molecular_profiling_of_neonatal_dried_blood_spots_reveals_changes_in_innate_and_adaptive_immunity_following_fetal_inflammatory_response.pdf.
Attempting to download Molecular_profiling_of_neonatal_dried_blood_spots_reveals_changes_in_innate_and_adaptive_immunity_following_fetal_inflammatory_response from
https://www.medrxiv.org/content/10.1101/19000109.full.pdf
  2%|          | 2/100 [00:06<04:50,  2.97s/it]
Molecular_profiling_of_neonatal_dried_blood_spots_reveals_changes_in_innate_and_adaptive_immunity_following_fetal_inflammatory_response Downloaded to
medrxiv_papers/Molecular_profiling_of_neonatal_dried_blood_spots_reveals_changes_in_innate_and_adaptive_immunity_following_fetal_inflammatory_response.pdf.
Attempting to download Genome-wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response from https://www.medrxiv.org/content/10.1101/19000109.full.pdf
  3%|          | 3/100 [00:09<05:07,  3.17s/it]
Genome-wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response Downloaded to medrxiv_papers/Genome-
wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response.pdf.
Attempting to download Genome-wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response from https://www.medrxiv.org/content/10.1101/19000109.full.pdf
  4%|          | 4/100 [00:12<04:50,  3.02s/it]
Genome-wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response Downloaded to medrxiv_papers/Genome-
wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response.pdf.
Attempting to download Genome-wide_postnatal_changes_in_immunity_following_fetal_inflammatory_response from https://www.medrxiv.org/content/10.1101/19000109.full.pdf
  5%|          | 5/100 [00:15<04:56,  3.12s/it]
```

After running my download function in the jupyter notebook, you will see something like the above. Picture created by the Author.



Check our Qdrant again, a new database “medrxiv_cardiac” is already been created. Picture created by Author.

4. Construct the query engine

This is another straightforward process, let’s set up our query engine:

```
class RAG:
    def __init__(self, config_file, llm):
        self.config = config_file
        self.qdrant_client = qdrant_client.QdrantClient(
            url=self.config['qdrant_url']
        )
        self.llm = llm  # ollama llm

    def load_embedder(self):
        embed_model = LangchainEmbedding(
            HuggingFaceEmbeddings(model_name=self.config['embedding_model'])
        )
        return embed_model


    def qdrant_index(self):
        client = qdrant_client.QdrantClient(url=self.config["qdrant_url"])
        qdrant_vector_store = QdrantVectorStore(
            client=client, collection_name=self.config['collection_name']
        )
```

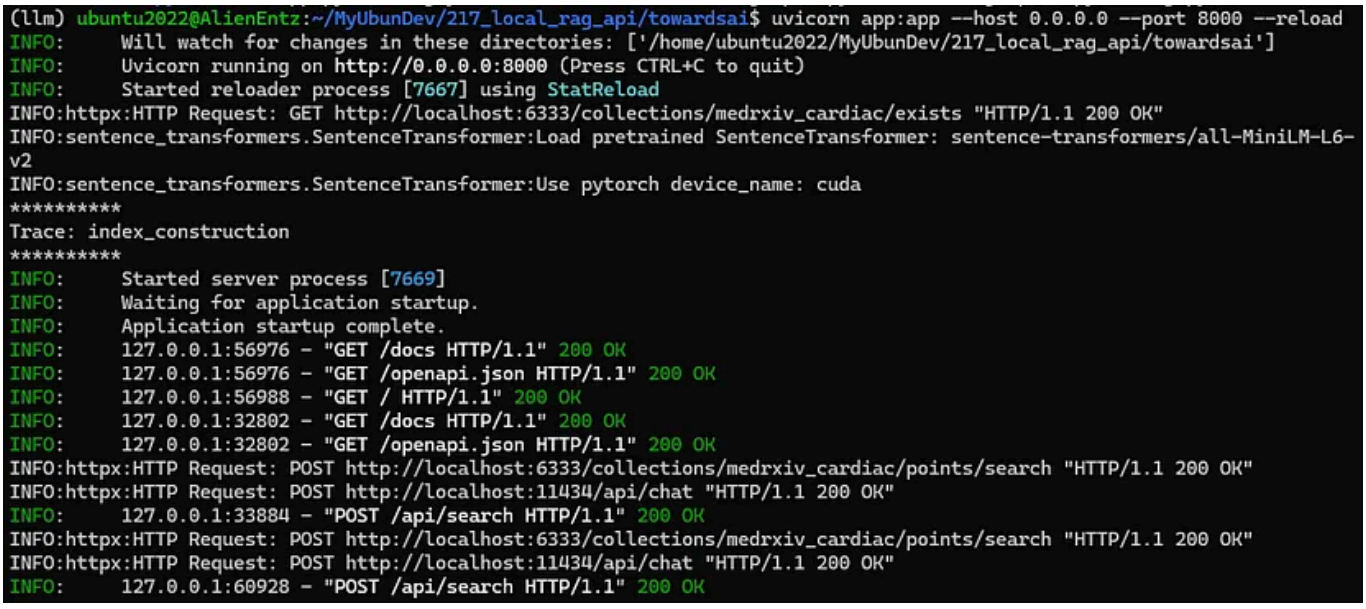
```
        service_context = ServiceContext.from_defaults(
            llm=self.llm, embed_model=self.load_embedder(), chunk_size=self.conf
        )

        index = VectorStoreIndex.from_vector_store(
            vector_store=qdrant_vector_store, service_context=service_context
        )
        return index
```

5. Start up the API app

Finally, it's time to launch the API app. First, navigate to your working directory and run:

```
uvicorn app:app --host 0.0.0.0 --port 8000 --reload
```

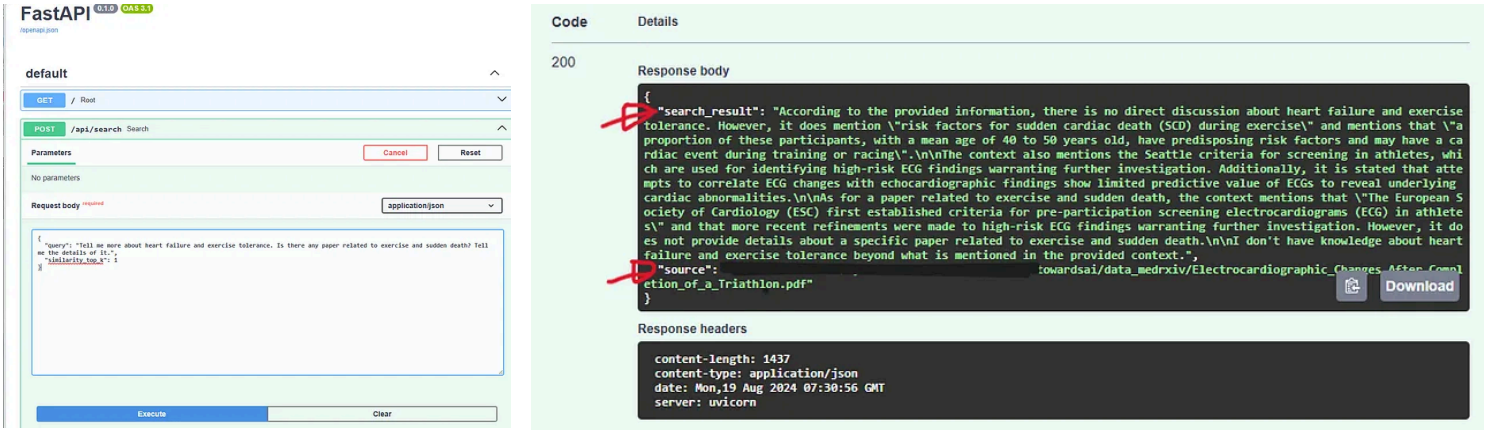


Picture created by the Author.

Then you can go to the localhost FastAPI — Swagger UI, and test this FastAPI app. My query for the AI was "Tell me more about heart failure and exercise

tolerance. Is there any paper related to exercise and sudden death? Tell me the details of it.”

```
http://localhost:8000/docs#/default/search_api_search_post
```



The answer is satisfactory! Photo created by the author.

Conclusion

In this project, I demonstrated how to build an AI-powered research assistant tailored for medical professionals, and a professional nurse in my case. My solution involves using a combination of Llama3 through Ollama for natural language processing, Qdrant for vector-based storage and retrieval, and FastAPI for deploying the application as an accessible API.

This AI research assistant significantly reduces the time and effort required for reading through research papers, enabling professionals like my wife, a senior nurse, to focus more on writing and less on data gathering. The entire development process — from setting up the Llama3 model to deploying the application with FastAPI — highlights the power of modern AI tools in

transforming the way we learn and conduct research in a constructive yet not cheating way.

This project is a practical example of how data science and AI can be leveraged to solve real-world problems, offering a glimpse into the future of personalized and automated research assistants. The code and setup instructions provided ensure that the solution is reproducible and adaptable, allowing others to modify and extend the application to suit their specific needs.

Thank you for reading. If you like this tutorial, please share it with your data science friends, and **follow me**. Following me is the motivation for me to continue contributing to the community.

References and Further Reading

https://medium.com/r/?url=https%3A%2F%2Fgithub.com%2Follama%2Follama%2Fblob%2Fmain%2Fdocs%2Fmodelfile.md%23build-from-llama3

bioRxiv API (medrxiv.org)

sentence-transformers/all-MiniLM-L6-v2 · Hugging Face

https://docs.llamaindex.ai/en/stable/api_reference/storage/vector_store/qdrant/

https://docs.llamaindex.ai/en/stable/examples/vector_stores/QdrantIndexDemo/

AI Llm Llama 3 Qdrant Rags

Written by Lorentz Yeung

Follow

215 Followers · Writer for Towards AI

Data Analyst in Microsoft, Founder of El Arte Design and Marketing, Certified Digital Marketer, MSc in Digital Marketing, London based.

More from Lorentz Yeung and Towards AI

Lorentz Yeung

A LangChain + OpenAI Complete Tutorial for Beginner—Lesson 3...

Unlocking Advanced Chatbot Capabilities: A Journey Through LangChain Expression...

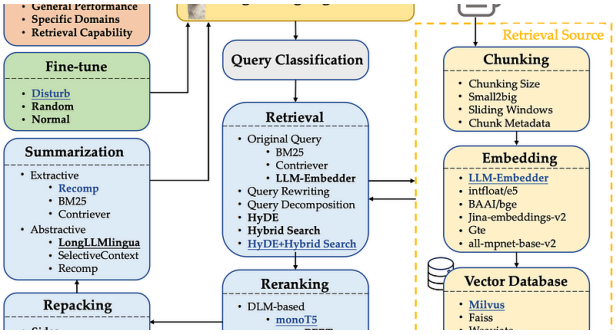
Lazar Gugleta in Towards AI

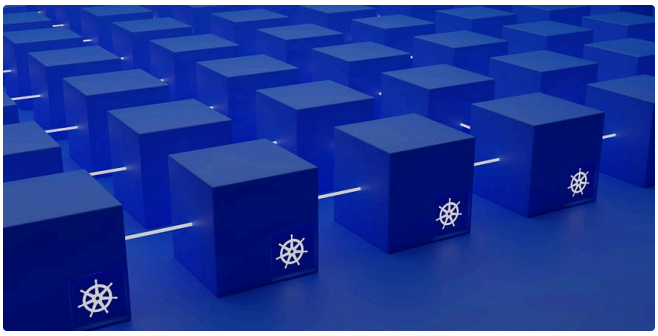
Why Polars Destroys Pandas in All Possible Ways for Data Scientists?

One of the first Data Science libraries, Pandas, has been improving the lives of many...

Feb 7 82 1

Aug 7 627 10





Florian June in Towards AI

The Best Practices of RAG

Typical RAG Process, Best Practices for Each Module, and Comprehensive Evaluation

Aug 8 810 3

Lorentz Yeung in Towards AI

A LangChain + OpenAI Complete Tutorial for Beginner—Lesson 2...

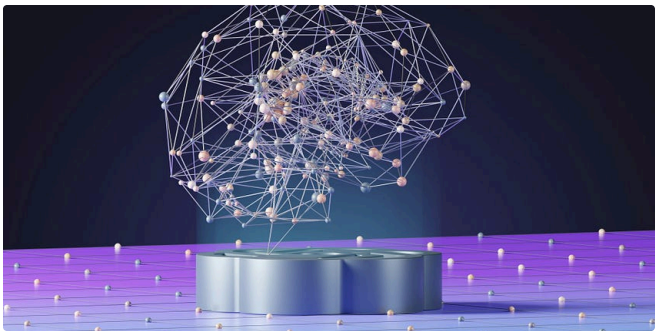
Unlocking Next-Level Conversations: Deep Dive into RAG and Vector Databases

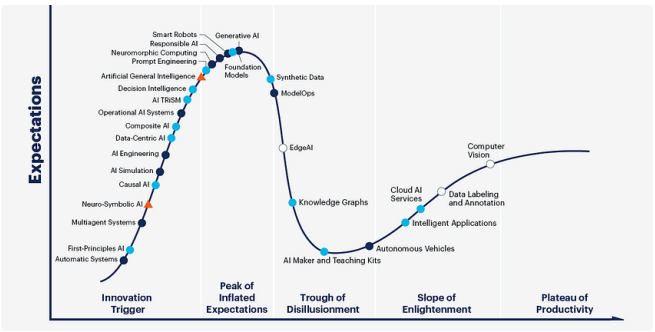
Feb 6 100

See all from Lorentz Yeung

See all from Towards AI

Recommended from Medium





Prakash Joshi Pax

Fabric: The Best AI Tool That Nobody is Talking About

An open-source AI tool to automate every day tasks

Aug 26 1.1K 7

Vishal Rajput in AIGuys

Why GEN AI Boom Is Fading And What's Next?

Every technology has its hype and cool down period.

Sep 4 692 7

Lists

Generative AI Recommended Reading

52 stories · 1347 saves

What is ChatGPT?

9 stories · 432 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 455 saves

Natural Language Processing

1687 stories · 1260 saves

Lorenzo Zarantonello in Level Up Coding

The One Reason Why I Moved From VS Code To Cursor

The ease of change made me try it. Smooth AI interaction made me stay.

Aug 27 603 17

Michael Wood in Cubed

Eliminating Hallucinations Lesson 1: Named Entity Filtering (NEF)

Named Entity Filtering

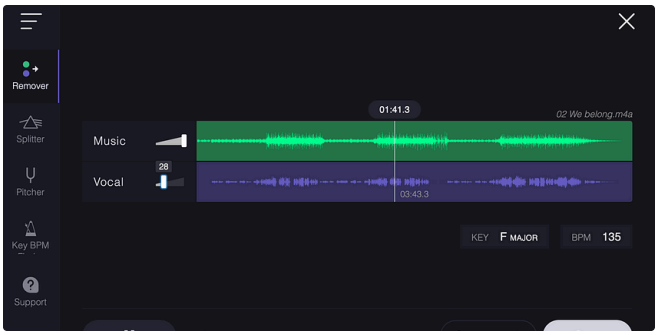
Sep 2 364 6

Addison Best in Generative AI

Llama 3.1 405B—How to Use for Free

No Local Install Needed

Aug 19 115 1



Rafe Brena, Ph.D. in Towards AI

We Are Entering The Second Wave Of Generative AI

How to tell the hype from the truth

Aug 25 298 14

See more recommendations