MASTER INFORMATIQUE:

Artificial Intelligence and Data Engineering

4th Semester Report

SUBJECT:

Chatbot for an API's documentation website

Alternant: Fayssal EL ANSARI

University teacher: Christophe CRESPELLE

Company teacher: Stéphane CHABANEIX

2023-2024

Acknowledgements

I would like to express my deepest appreciation to my teachers, Ines ARRIGONI and Stéphane CHABANEIX, as well as my university teacher, Christophe CRESPELLE. I would also like to extend my sincere thanks to each member of the BeNomad developer team and the staff of the University of the Côte d'Azur who helped me throughout my learning at BeNomad and offered me valuable advice. I am extremely grateful for their invaluable contributions and enlightening perspectives. These past few months have been highly informative, and I have had an extremely rewarding experience. A big thank you to Ines ARRIGONI and Stéphane CHABANEIX for devoting time to my training, for their patience, and their availability despite the demands of their schedules.

I am also grateful for the help provided by David LABROUSSE and Stéphane MAYNARD who taught me many things, as well as Piotr DLUGOSZ, Stephanie LYON, Vinh LUONG, William KRUNKEL, and Jérôme HESS, team members, for the help and information they provided me during my apprenticeship. I would like to thank Laurence HUGUES, Etienne Lozes, and Christophe CRESPELLE for their follow-up over these months, as well as for their advice.

My apprenticeship at BeNomad has been a very rich learning experience. This work would not have been possible without all the people who welcomed and warmly supported me over the past two years.

I also want to thank my teachers and everyone at Polytech School. Your support and guidance have been invaluable to me. It's been a great experience learning from you all.

BeNomad

BeNomad is a company that offers SAAS solutions to businesses operating in the field of geolocation and mapping. Since 2002, BeNomad has been designing, developing, and marketing mapping and navigation software adapted to help businesses better manage their mobile resources. Their programs include: BeNav, BeMap, Ev-Move, Ev-Move Fleet, Fleex, as well as other software development kits (SDKs) and mapping application programming interfaces (APIs).

The project I am working on is BeMap, which is a collection of APIs for GPS navigation, route management, and mapping services. It is a mapping server solution offering a set of geolocation functions intended to be integrated into backend computer systems or transport operators' mobile/web applications. Accessible via a set of web services, BeMap offers a range of functions for route planning and analysis, calculating vehicle routes (cars/trucks), tracking completed journeys, and calculating transportation costs.

Presentation of the Team

I am part of the BeMap team, which is led by Nadia RAMDANI. I work directly with Ines ARRIGONI, Stéphane CHABANEIX, and Piotr DLUGOSZSDK

Role	Name
CTO	David LABROUSSE
CEO	Stéphane MAYNARD
Financial and administrative director	Laurence HUGUES
Project manager	Killian LEROY
Software architect	Ines ARRIGONI
Quality Assurance Engineer	Vinh LUONG
Technical Lead	Stéphane CHABANEIX
Full stack developer	Piotr DLUGOSZSDK
developer	Stéphanie LYON
Software architect	William KRUNKEL
Commercial development	Jérôme HESS

Description of the proposed work

The main idea behind this task is to help the users converse with a public documentation website. A website that offers examples of usages of the different APIs of BeMap, a reference with a precise description of each one of these APIs endpoints with their parameters and what to expect as a response, and a collection of step-by-step tutorials. Spanning multiple different versions and usage protocols.

The way we'll do this is by adding a chatbot that would ideally be powered by an advanced LLM tailored towards our precise use case. This service would allow visitors and clients to use the chatbot as an advanced search engine that is capable of answering frequently asked questions more interactively, similar to a dynamic conversation with a support agent. Instead of using the usual email system with longer periods of waiting and unnecessary formalities, a chatbot could answer questions on the fly without the need for a human operator.

Such a service is already used by competitors, other APIs providers that already have a public facing documentation website (such as MapBox). But they do not necessarily use an in-house solution.

If we succeed in building a reliable and cost effective chatbot service for the BeMap documentation website, the same system could be modified for other custom BeNomad products. The goal is to reel in more clients and offer a dynamic and easy to use customer support service.

Case study

This is a snippet from a case study document, it includes aspects to look out for before starting the development and/or testing, and it includes a comparison between several 3rd party chatbot API providers.

Security and Privacy

- **General Considerations on Security and Privacy**: Should user input be backed up on the servers, should take into consideration GDPR laws.
- **Methods to Teach the Model**: whether or not we'll need to teach the model without retaining private data.

Protection of Documentation

- **Strategies to Prevent Sharing Internal Documentation with External Sources**. Will not be needed since we do not plan on keeping the documentation of BeMap (in this case) private.

Model Choice

- **Decision Making**: Determine whether to develop our own model or use a third-party tool.
- **Time Estimation**: Time required to obtain a functional prototype.
- **Using a Third-Party Tool**: Likely the fastest option.

Cost and Quality

- **Cost-Quality Analysis**: Analyzing the costs associated with using a third-party tool based on usage.
- **Local Implementation Costs**: Costs related to new server services.
- **Quality Verification Methods**: How to verify quality.
- **Necessary Research**: Research required for the process.

Chatbot Integration

- **Technical Implementation**: Choice of the technology stack.
- **Relevance of an Independent Prototype**: Before full integration.
- **Creation of a New Private API**: Training the model on the content of the documentation.

Model Update and Training

- **Necessity to Re-train the Model**: With each update of the documentation or API, either manually or automatically.

Development Process

- **Model Selection**: Reasons to avoid external tools for confidentiality and cost issues.
- **First Step**: Obtain a functional prototype to test effectiveness.
- **Model Options Considered**:
    - **GPT-J**: Requires 48GB of RAM for loading. Trained only on data in English, not ideal for translation.
    - **t5-small**: Exploring transfer learning with T5.
    - **Mistral-7B-v0.2**: Uses 16GB for inference.
    - **Mistral-8X7B-v0.1**: Requires 100GB for inference.

Training and Testing the Model

- **Model Training Process**: Evaluating the trained model.

Examples of Online Documentation with a ChatBot Service

- **Documentation Sites**: Using platforms like docs.mapbox.com and services such as kapa.ai to provide these services.
- **Documentation Creation**: Using the same markdown files from the BeMap and React documentation can utilize Docusaurus for publishing the same documentation without modifications.

Providers

- **Docsbot**: Uses OpenAI's API - Docsbot.
- **Trengo**: Trengo.
- **Kapa.ai**: Uses OpenAI's API. Options for model generation include GPT-4 variations.
- **Chatbot.com**: Chatbot.com - LLM model?
- **ChatGPT**.

Summary

- **Cost Calculation by OpenAI**: Calculated in tokens with a maximum of 4,096 tokens per request, where the cost of input differs from the output. The output being more costly is primarily considered. This results in approximate figures below for token usage.

Service Provider Pricing for API Usage

Provider	0-1000 Requests	1001-5000 Requests	5001-10,000 Requests	10,001-25,000 Requests	25,001-100,000 Requests
Gpt-3.5 Turbo	\$0.6 - \$6	\$3 - \$30	\$6 - \$60	\$15 - \$150	\$60 - \$600
Gpt-3.5 Turbo-instruct	\$0.8 - \$8	\$4 - \$40	\$8 - \$80	\$20 - \$200	\$80 - \$800
Gpt-4-preview	\$12 - \$120	\$60 - \$600	\$120 - \$1200	\$300 - \$3000	\$1200 - \$12000
Kapa.ai (monthly)	-	-	-	-	-
Docsbot.ai (monthly)	\$16	\$41	\$82	\$82	\$416
Trengo (yearly)	\$99	\$149	\$149	\$249	?
Chatbot.com (monthly)	\$52	\$142	\$424	\$424	?

Additional Details

- **Integration Simplification Using OpenAI Assistant**: Potential to simplify documentation integration.
- **System Similar to MDN Documentation**: Allowing only logged-in users to ask questions, with a counting system and a limit on the number of questions per duration.

Methodology

Since we do not have prior experience working with locally deployed LLMs. We start by running an LLM on a modest machine. A machine with standard resources, and we'll assume the quality of responses without automated tests but rather a couple of manual tests to get a feel of what to expect.

At first, we tried out models by downloading them manually from hugging-face and testing them one by one. But we later found a couple of programs that allow for an easier testing experience, such as Ollama and LLMstudio. Both were very useful.

Description of the work in progress

Implementation

This section describes some interesting design choices and how some parts are implemented.

LLM choice

The choice of what LLM to use in our local model depends on many factors, the most important thing to consider when making this choice is the quality cost. Usually, we'd prefer an open-source model that is easily fine-tuned or customizable. That also offers a fast response while not needing many resources for inference. We had our eyes on the famous mistral-7B model.

LLM Quantization

One of the problems we faced while working with LLMs locally is the amount of GPU vRAM needed for inference, meaning the amount needed to run the model at reasonable response times. To face this obstacle users of local models referred to quantization which entails changing the precision of the model's weight matrices. Which in return leads to a lower quality response.

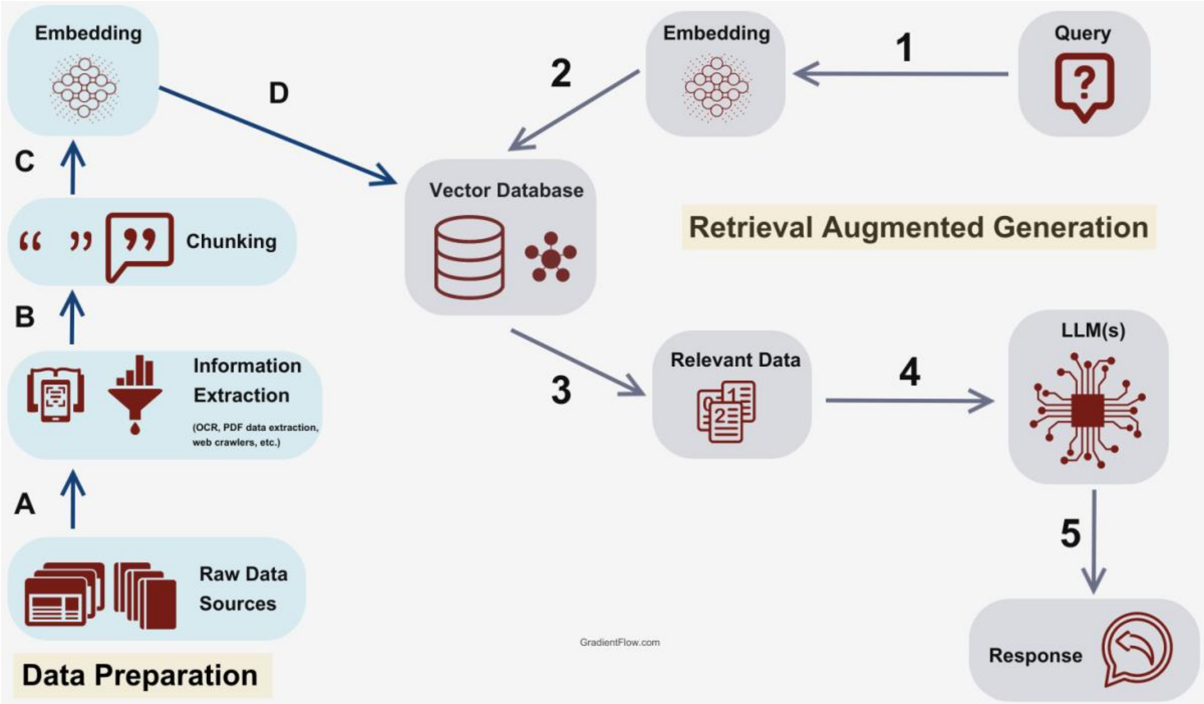
Finetuning

LORA

LoRA or Low-Rank Adaptation of Large Language Models is an important paradigm of natural language processing which consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains.

RAG

This is the choice we're going with. Models are trained on huge amounts of data using powerful GPUs, but in most LLM applications we look for a way to incorporate our data with the already trained model. Due to the cost and time needed to retrain the whole model at each modification to our data or even finetuning, a commonly used solution is RAG (Retrieval-Augmented Generation) which consists of creating an indexed vector database for our data. Where the LLM can amplify the response and make it tailored specifically for our use case while providing the source of the generated response. The standard way RAG works is by feeding into the context of the LLM the relevant information from a local database, this way the answer to be sent back to the user is already included either directly or indirectly in the context of the sent request.



Even a simple RAG application entails tuning many different parameters, components, and models.
[https://gradientflow.com/techniques-challenges-and-future-of-augmented-language-models/]

So, the steps of creating a custom chatbot for the documentation become:

Crawl the documentation website to get all the information that could be used later in answering questions, while removing duplicate or unnecessary information.

Split this information into sections of logically dependent sections, for example, we can split it by paragraph. This way we're more likely to have knowledge that depends on another piece of knowledge in the same section.

Embed our sections, meaning we'll need to write it in a way understandable by our model.

Store all the embedded data in a vector store for it to be searched and retrieved later.

One RAG tool is LlamaIndex which allows us to store BeMap documentation in a folder for example /data, we can send a question either to a remote API like OpenAI or use a local model, we'll check the local data first then ...

We crawled BeMap's documentation website to download the pages into the /data folder. By sending a simple question related to BeMap's documentation we are able to get some answers.

For example:

Q. `How can I add a marker to my bemap map?`

A. Call the `createMarker` method with the appropriate `options` parameter, which includes the `map`, `properties`, and `icon` details. Optionally, you can use a callback function to handle data from marker clicks. This method will return a a marker array on the bemap map.

Q. `What are the different possible values of searchType in benomad's bemap geocoding reference doc?`

A. Geocoding, Geocoding for batch, Natural Geocoding, Reverse geocoding, Reverse geocoding for batch, Feature of land.

Q. `what is the boundingBox of BeNomad's Bemap?`

A. The bounding box of BeNomad's Bemap is obtained using the method `bemap.map.getBoundingBox();`.

One problem we faced is that it takes a lot of time to get a response, this is due to the number of web pages that need to be read for each question. Also, with more files in the context we get degraded results. Another is that we're consuming a lot of OpenAI's embedding API credits. And sometimes we get answers that aren't directly related to our data.

After enabling the visualization of progress by using the parameter show_progress both in *`SimpleDirectoryReader.loadData()`* and in *`VectorStoreIndex.from_documents()`*. We can see that there are two main bottlenecks when working on local embedding and LLM, which is the embedding step after retrieving the documents and the generation of a response with the augmented context. We'll resort to generating an embedding one time for each version of the docs and saving it in a vector store.

As for the model, it is possible to work with a quantized model, unlike the unmodified complete model of mistral 7b.

To get a sense of the progress of our response we'll use index.as_query_engine(streming=True). Which will stream the response by word.

Another useful feature to look into is using the LLM application as a chatbot and not just as a question and answering application, LlamaIndex offers a built-in chat engine, something similar to ChatGPT but everything is deployed locally and augmented with the documentation pages.

To stream the response of the chat application we need to use the parameter `stream_chat` unlike the query engine.

External API documentation test

MapBox

One possible way to test out our LLM application is to fetch the documentation of mapbox.com and compare prompts responses provided by them against our model.

They seem to be using a React application with Docusaurus, we don't know if there is any direct way to fetch markdown files, so we'll resort to creating a crawler that will extract only relevant information from the built static webpages.

The main characteristics of their AI chatbot are:

- It gives out information from two main sources, the documentation and all the StackOverflow threads that contain the tag `mapbox`.

This is a custom LLM for Mapbox with access to all developer docs (docs.mapbox.com) and all Stackoverflow questions (stackoverflow.com/questions/tagged/mapbox). Please note that answers are generated by AI and may not be fully accurate, so please use your best judgement.

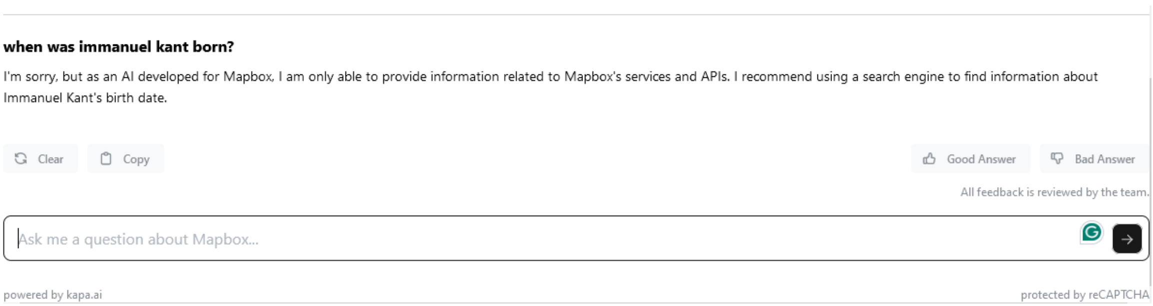
- When constructing the response, it gives out information from multiple source pages, unlike the first test with LlamaIndex which gives the feeling that only the first result is added to the context. To mimic this behavior, we'll need to use the k-nearest parameter to define how many nodes to fetch for the answer construction.
- It gives all the list of sources used for constructing the answer.

For Android, a route can be requested with the **requestRoutes** method, and if successful, returns a route reference without acting on it. You can then pass the generated routes to **setNavigationRoutes**.

Sources:

- **Mapbox Directions API**
- **Directions API Playground**
- **Mapbox Support**
- **RequestRoutes**
- **SetNavigationRoutes**

- Inside the answer some phrases contain hyperlinks to the corresponding sources (inside the documentation or from StackOverflow), which could be just the same feature provided by ChatGpt simply.
- If we ask an information not found in local source, it does not give out the response. Probably to save up on token usage/inference costs.



So in the end result what we want to achieve is:

- Correct answers with relevant information
- Fast responses
- Usage of multiple sources
- Fast deployment
- Listing of used sources

- Filter out non relevant questions
- Chat answer is streamed by chunks

From the test on MapBox we were able to define these areas of possible improvements (details in appendix 1):

- Provide an example when possible
- Sources of the answers are always given
- Links to the sources are embedded in the text, in the end after the keyword *sources* and as links right after a separator.
- Our local chatbot does not provide further details on its own so we need to ask for elaboration to get a bigger more detailed answer.
- When there are no local sources to add to the context there is no need to interrogate the LLM, to avoid usage that does not answer questions that relate to the documentation and therefore to avoid unnecessary processing costs.

This is just a primary test, intended to serve as a prototype of what to expect and what we would need to work on in the final project.

The format in which we save the data before splitting and embedding could affect heavily the final results. To use the correct format for our use case it could be useful to understand what is happening under the hood when we use LlamaIndex to read, embed and store the crawled data.

To summarize

Since we plan on using the same chatbot with .md files that are produced by other Benomad products, we can also test it out on publicly available documentation such as GitHub docs: https://github.com/github/docs/tree/main but while producing the responses the chatbot may depend heavily on the data the LLM was trained on and not the retrieved data, so it's better to have private documentation in markdown format to test it on.

Storing

After the embedding step we can store the generated vectors into the file system directly using persist(). But the format of saved data makes it harder to keep track update and read manually. So a better option would be to save the data into a store (database).

Quality Of Responses

When working with a huge quantity of documents the quality of responses is impacted negatively, to overcome this we can start off by changing the way we load our files. For example the mSDK documentation contains many webpages, each of which has associated javascript and CSS styling which could be ignored, we are left with two main file formats .md and .HTML.

Filtering and restricting responses

This is a very important step to perfect, to have a usable product we'll need to be able to completely filter out responses that could affect negatively the user experience or the products that the user is trying to learn more about. Or it allows some users to abuse the system to have a free to use LLM endpoint.

There are a couple of restrictions we'll need to add and test out that they are always working and hard to bypass.

- Forbidding the chatbot from answering questions that are unrelated to the documentation or benomad and its products in general.
- Not recommending competing products at the cost of BeNomad's products.
- Hallucinations, not giving answers when the chatbot doesn't have an answer instead of inventing false information.

Inference Speed

To improve the inference speed we'll first need to add a logging mechanism to keep track of how much time is needed to produce a response for a couple of questions. We can tell from previous tests that the inference speed depends heavily on the available vRAM and the amount of data and the model. We'll increase the amount of documents and ask the same questions while fixing other variables to see how does the inference speed evolve with an increasing data size.

Adding sources

To keep track of sources we'll need to add metadata containing the link of the crawled page into each node. So we'll save the content of HTML pages in json format and add the source URL as metadata, we'll implement our own crawler instead of using HTMLNodeParser and we'll chose which tags to use to feed the JSON object. We'll keep using mapbox documentation as a source to be able to perfect the chatbot by periodically comparing our chatbot's responses to the responses of their doc's AI.

To modify the response, we decided to modify the generation of the request. We'll need to include some instructions to tell the LLM where to get the list of URLs to generate the list of unique sources from the context. We can do this by customizing LlamaIndex chat prompts.

The response object of llamaIndex is <class 'llama_index.core.base.response.schema.Response'> has a list of which should contain the resources used to generate the response, we can access this list by using the inner method get_formatted_sources(). With a score of certitude for each source node.

Final REST Application

To put all the previous steps together, we'll build a simple POC rest API that will take care of the backend and frontend, we'll use React, FastAPI and dockerize the chatbot for easier deployment.

For the time being there is no need to build a comprehensive application, we'll make it connected to BeMap only with no possibility to update the data. But many features can be added later on.

One problem we'll face is the fact that the chatbot uses Ollama which is not multithreaded so whenever a new request will be received by the server the bottleneck is probably going to be Ollama where if multiple consecutive requests are received they will be processed in a serial manner. We can try to overcome this by running multiple Ollama instances and using a load balancer, but it's not needed for now.

At the time of writing this report, a new update with the possibility of defining the max and min number of deployed LLMs was added, it serves the purpose of concurrency but consumes a lot of vRAM.

Following the guidelines in: https://docs.llamaindex.ai/en/stable/optimizing/production_rag/ working with production-ready application is a bit different from a POC or a prototype. One of the most noticeable differences is the quality of response when working with more local data.

Conversation endpoint

With a simple comparison with kapa.ai chatbot, we can see that there are chunks returned from the API, each chunk of text has one of three types: sources, identifiers and text. Each chunk has a content object and a Boolean value called end_of_stream. We can copy the same structure for our application.

The first sent out chunk will be the sources used for the generation of the response, the second to the before last chunk will all contain text tokens, the sent out text chunks should also contain formatting such as "\n" and "\t" which will be interpreted on the front-end.

To keep track of the current conversation we'll use two different API endpoints. The first endpoint will have no notion of conversation or thread identifier and the second we'll do. This way the first endpoint will be used to initiate a new thread and from the last chunk received which will contain the identifiers of both the thread and the message we can pass it to the second endpoint. This will allow us to keep a history of conversations and messages and add relevant messages to the context server side.

Streaming responses

The way we'll be streaming responses is by using SSE (Server Side Events)

Dockerizing the application

One very usefull task is to create a docker image for the backend, frontend and even the database. This will allow for easier and streamlined deployment after each update and will ensure a common environment across instances of the application.

backend

Since our backend is a FastApi server, we'll need to create a requirements.txt file which will contain the totality of our needed packages, we start off by creating a virtual environment and then we'll activate it and install and setup the environment as needed, after we get the application up and running we'll simply freeze the installed packages into the requirements.txt file that we'll use later in our docker image.

We also copied all of the sources and data files into one folder to be copied later on by the Dockerfile into the WORKDIR of the container. So now I have app/src and app/data both inside a folder named server.

Depending on the current network's bandwith, it could take a lot a of time to deploy the application for the first time, this is due to the number of installed dependencies, for the backend there are some tools that allow us to compare the used dependencies with the dependencies present in the requirements.txt, while also checking for any dependency issues. Such as `deptry`.

CORS

Even though we had CORS setup on our REST application to allow all origins to communicate with the backend, the deployed application seemed to refuse to connect.

Hostnames and networks

Even though there is a default netowrk created for all of the services we created a new custom network and allowed communication between the containers.

Environment variables

To be able to use the same code on the docker container and also during development we define environment variables on the docker-compose file and on our machine, this way we can change some parameters depending on the chosen environment whether it's production or development.

Ollama

An ollama service needs to be running for our backend to work, it needs to be able to communicate with said server, so we'll need to do some modifications to our docker-compose.yml file and our backend.

The way we'll do this is instead of having the Ollama service running in the same container as the backend, we'll give it its own dedicated container, also there needs to be a pulled model, in our case we'll need to pull llama3 at the creation of the container. To surcomvent having to repull the model at every new startup we'll work with the already built image of ollama:ollama and we'll create a script to launch and pull llama3 at startup, then we'll need to commit a new image for the modified image with the language model already pulled.

On the backend main application we'll change the Ollama command to include the endpoint and default port, which is the name of the ollama container: ollama:11434

On the docker-compose.yml the new ollama service will use the same network as the other services.

Telemetry

One important aspect of our application is to keep track of what requests are being made and the overall workflow of each one of these requests, this will allow us to better log, debug and improve the user experience.

A proposed solution is to add Arize Phoenix, this framework which is compatible with llamaindex, makes it easier to do just that. This should have been one of first steps to complete during the development, which would have saved us a lot of time.

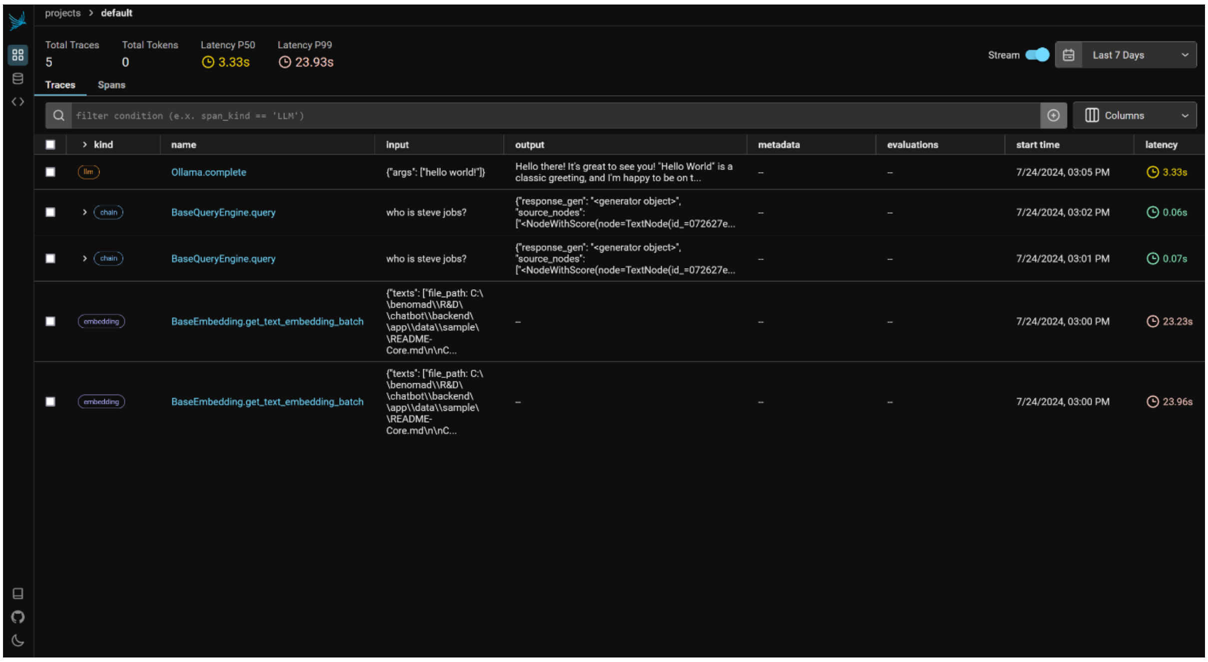
We'll start off by installing the required dependencies:

```
pip install opentelemetry-api opentelemetry-instrumentation opentelemetry-semantic-conventions opentelemetry-exporter-otlp-proto-http
```

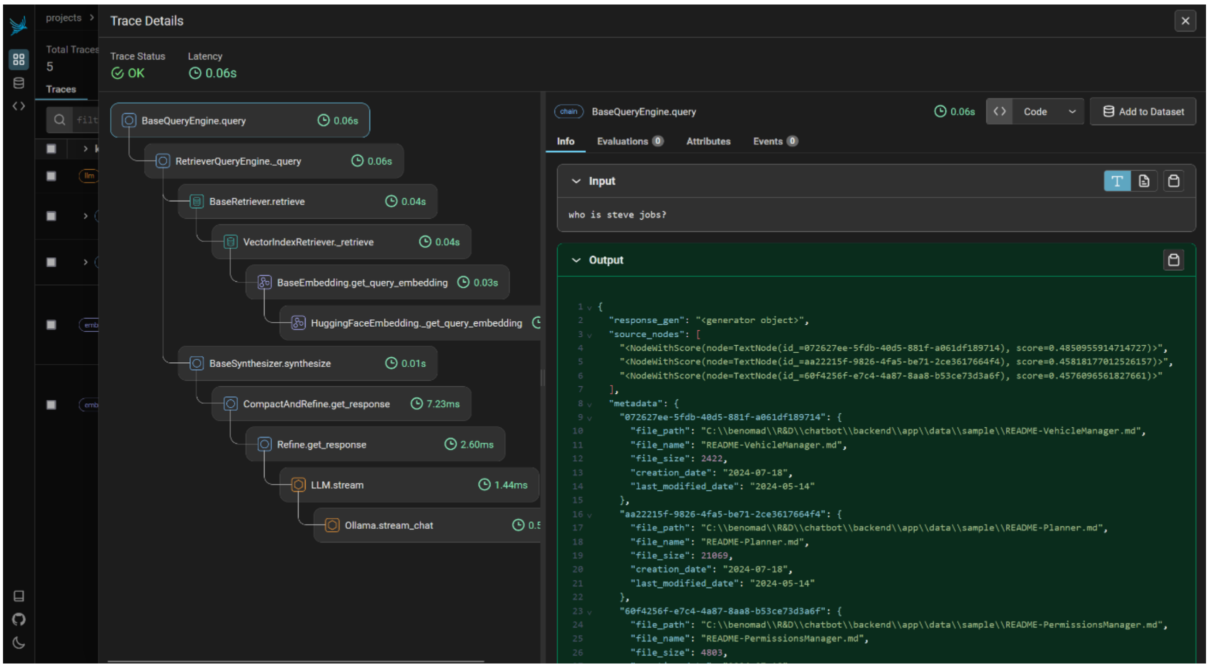
Then apply the needed modifications to integrate it into the previous application.

Screenshots

Telemetry screenshots

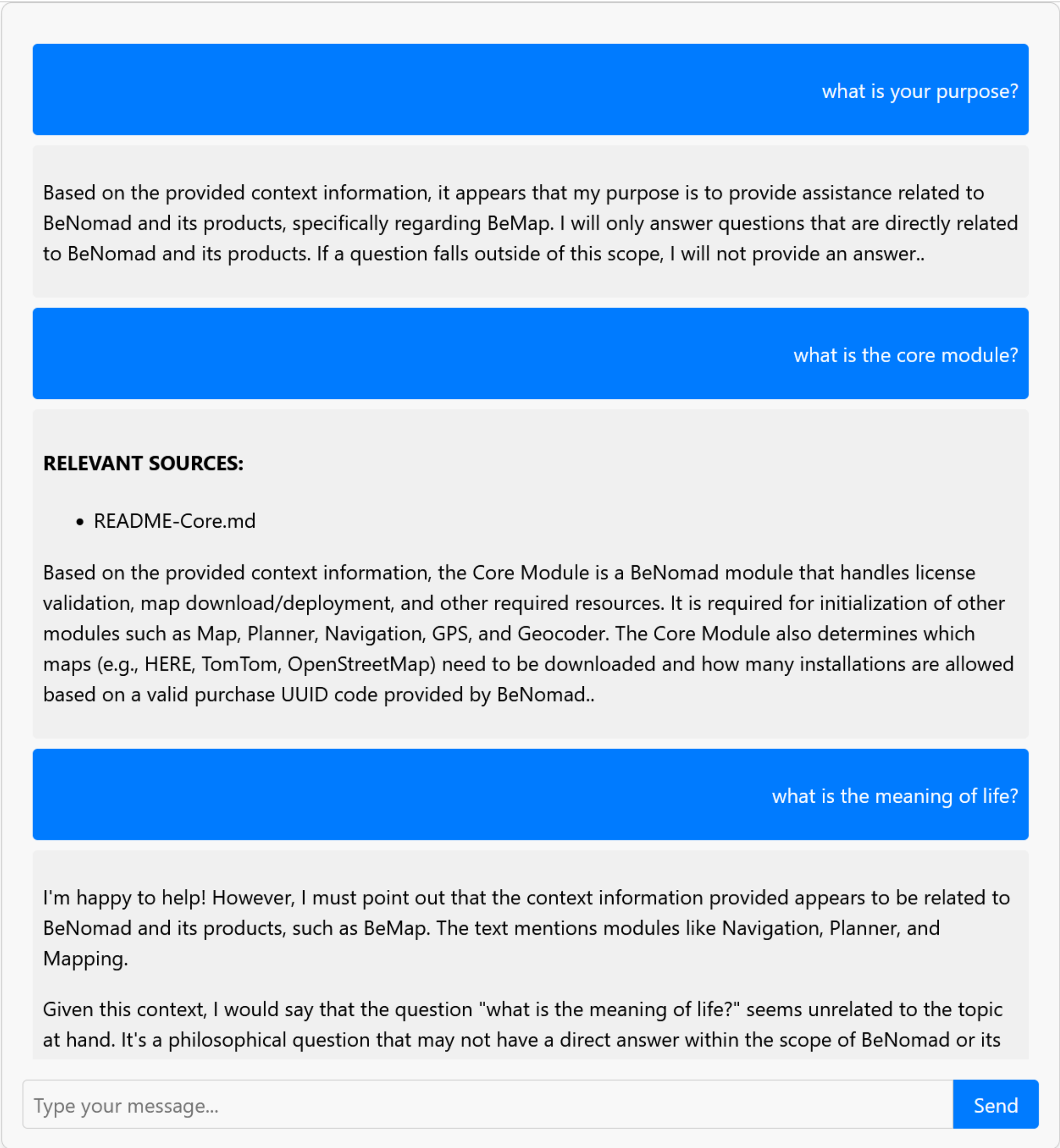


Screenshot showing the Phoenix interface with requests listed



Screenshot showing the Phoenix interface with one highlighted request and its details

User facing application screenshots



A screenshot of our chatbot in work, showing in context and out of context questions

A screenshot of our chatbot in work, showing the ability to generate code with syntax highlighting

Conclusion

The development project of a chatbot for an API's documentation was very interesting and gave me the chance to use new technologies such as Llamaindex, Ollama and many more that are being updated very fast, and it also allowed me to learn to use LLMs in a local and remote context.

It was a practice for a useful application that could possibly scale to a standalone project with many future applications. It allowed me to apply what I've learned during this last year in university.

It also allowed me to work on the architecture and design aspect of a full stack LLM application, and introduced me to the main challenges that we could face when working with LLMs in a production environment.

Finally, it demonstrated the potential of such technologies to scale across different products and services within BeNomad, suggesting a broader applicability and promising substantial returns on investment in terms of both customer satisfaction and operational efficiency.

References

https://arxiv.org/abs/2106.09685

https://python.langchain.com

https://ollama.com/

https://www.llamaindex.ai/

https://huggingface.co/mistralai/Mistral-7B-v0.1

https://docs.llamaindex.ai/en/stable/examples/evaluation/

https://docs.arize.com/phoenix/tracing/integrations-tracing/llamaindex

https://gradientflow.com/techniques-challenges-and-future-of-augmented-language-models/

Appendix

First test

I downloaded a couple of pages from the official mapbox documentation website, for 4 APIs: directions, isochrone, matrix optimization and the main page containing all the details about these apis and general navigation information.

I saved all these pages as .htm without any modifications and used BAAI/bge-small-en-v1.5 model to embed them and mistral with the help of Ollama and LlamaIndex. I used a vectorStoreIndex the embeddings are saved a folder named storage for future use without having to reembed the documents at each program run.

The chatbot was able to answer questions in a way similar to this

1. is there an API that does so and so...?

what apis can i use for navigation?

Mapbox provides a collection of APIs that you can use to add directions-related services to your application:

1. **Mapbox Directions API**: This API allows you to generate a route with trip durations, estimated distances, and turn-by-turn directions. **Learn more**

2. **Mapbox Matrix API**: This API helps you retrieve travel times between many points. **Learn more**

3. **Mapbox Isochrone API**: This API allows you to compute areas that are reachable within a specified amount of time from a location. **Learn more**

4. **Mapbox Optimization API**: This API helps you retrieve duration-optimized trips between input coordinates. **Learn more**

5. **Mapbox Map Matching API**: This API allows you to align existing fuzzy routes to the routeable network. **Learn more**

You can access these services directly using the web services APIs, through the Mapbox Navigation SDK, or using one of several plugins and libraries to integrate these services into applications across platforms.

For a more personalized navigation experience, you can use the Mapbox Navigation SDKs which are built on top of the Mapbox Directions API. They provide all the logic necessary to implement a navigation experience in your app, including features like drop-in turn-by-turn navigation UI, traffic avoidance, automatic rerouting, and more.

- **Mapbox Navigation SDK for iOS**
- **Mapbox Navigation SDK for Android**

Sources:

- **Getting Started with Directions**
- **Mapbox Navigation SDKs**

Human: what APIs can i use for navigation?

Assistant: There are several Mapbox APIs you can use for navigation-related functionalities. These include the Mapbox Directions API, Mapbox Matrix API, Mapbox Isochrone API, Mapbox Optimization API, and Mapbox Map Matching API. The Mapbox Directions API provides point-to-point directions with routes and turn-by-turn instructions. The Mapbox Matrix API calculates travel times between multiple points. The Mapbox Isochrone API computes areas reachable within a specified time from a location. The Mapbox Optimization API retrieves duration-optimized trips between input coordinates, while the Mapbox Map Matching API aligns existing fuzzy routes to the routable network.

what api is used for routing, i want to know which route to take from point a to b which api should help me with this?

For routing from point A to point B, you can use the **Mapbox Directions API**. This API allows you to generate routes, route durations, and turn-by-turn directions. It specifically generates an optimal route between the points you provide.

If you want to generate a route that closely follows specific coordinates, you can use the **Mapbox Map Matching API**. This API generates a route on the OpenStreetMap road network that most closely follows the coordinates provided in the request.

For more details, you can refer to the following documentation:

- **Mapbox Directions API**
- **Mapbox Map Matching API**

If you want to build a navigation app for iOS or Android, Mapbox provides step-by-step tutorials:

- **Build a navigation app for iOS**
- **Build a navigation app for Android**

Sources:

- **Get started with the Map Matching API Part 2**
- **Next steps in Generate an optimized route with the Optimization API**

Human: what api is used for routing, i want to know which route to take from point a to b which api should help me with this?

Assistant: The Directions API is commonly used for routing and finding the best route from point A to point B. You can use the Directions API to get detailed directions, distance, and estimated travel time between two locations.

2. what does the API named X do?

can you give me more details about the directions api?

The **Mapbox Directions API** is a service that calculates the route between two or more waypoints and returns a route as a GeoJSON line. It also provides turn-by-turn instructions, distance, and estimated travel times for the mode of transportation you selected.

When making a request to the Directions API, you need to provide at least two waypoints — an origin and a destination. However, you can include up to 23 additional waypoints between, for a total of 25 waypoints.

You can specify additional parameters such as:

- Which routing profile to use (walking, cycling, driving, or driving with traffic)
- Whether alternate routes should be included in the response

You can also add optional annotations to your request, including duration, distance, speed, and congestion. Congestion is calculated by comparing night-time traffic speeds with real-time traffic speeds and assigning a level of congestion given the percentage difference.

The Directions API returns a JSON object containing a route with trip durations, estimated distances, and turn-by-turn instructions.

The use of the Directions API is billed by API requests. For more details, you can see the **Directions API pricing page**.

For more information, you can read about the directions response object in the **Directions API documentation**.

Source: **Mapbox Directions API**

Human: can you give me more details about the directions api?

Assistant: The Directions API provided by Mapbox allows users to calculate optimal driving, walking, and cycling routes considering traffic and incidents. It also generates turn-by-turn instructions and can produce routes with up to 25 coordinates for driving, walking, and cycling profiles. Additionally, the API can calculate routes for electric vehicles, suggesting optimal charging stops and providing battery prediction.

3. what is the endpoint of API X?

what is the endpiont of the directions api?

The endpoint for the Mapbox Directions API is:

```
https://api.mapbox.com/directions/v5/{profile}/{coordinates}
```

In this URL:

- `{profile}` is the mode of transportation. It can be one of the following: `mapbox/driving-traffic`, `mapbox/driving`, `mapbox/walking`, or `mapbox/cycling`.
- `{coordinates}` are the longitude,latitude waypoints. You can provide up to 25 waypoints, separated by a semi-colon `;`.

Here is an example of a Directions API request:

```
https://api.mapbox.com/directions/v5/mapbox/driving/-122.42,37.78;-77.03,38.91?access_token=<UserAccessToken>
```

In this example, the profile is `mapbox/driving` and there are two waypoints: `-122.42,37.78` and `-77.03,38.91`.

Remember to replace `<UserAccessToken>` with your Mapbox access token.

Source: **Mapbox Directions API**

Human: what is the endpoint to use?

Assistant: The endpoint to use for the Directions API is typically specified by the API provider. In the case of Mapbox, the endpoint for the Directions API is usually in the format:

\`\`\`

https://api.mapbox.com/directions/v5/mapbox/{profile}/{coordinates}

\`\`\`

You would need to replace \`{profile}\` with the routing profile you want to use (e.g., \`driving-traffic\`, \`driving\`, \`walking\`, \`cycling\`) and \`{coordinates}\` with the list of coordinates for the route. Additionally, you may need to include your API access token in the request for authentication and authorization purposes.

4. what are the obligatory parameters needed to use api X?

what are the parameters of this API request? the obligatory parameters to use

The Mapbox Directions API requires the following parameters:

1. **Profile**: This is the mode of transportation. It can be one of the following: `mapbox/driving-traffic`, `mapbox/driving`, `mapbox/walking`, or `mapbox/cycling`.

2. **Coordinates**: These are the longitude,latitude waypoints. You can provide up to 25 waypoints, separated by a semi-colon `;`.

3. **Access Token**: This is your Mapbox access token, which is used to authenticate your API request.

Here is an example of a Directions API request:

```
https://api.mapbox.com/directions/v5/mapbox/driving/-122.42,37.78;-77.03,38.91?access_token=<UserAccessToken>
```

In this example, the profile is `mapbox/driving` and there are two waypoints: `-122.42,37.78` and `-77.03,38.91`. `<UserAccessToken>` should be replaced with your Mapbox access token.

For more information, you can refer to the **Mapbox Directions API documentation**.

Please note that while these are the required parameters, the Directions API also supports a number of optional parameters that allow you to customize the route, such as `alternatives`, `geometries`, `overview`, `steps`, and `annotations`. You can find more details about these in the **API documentation**.

Human: what are the parameters of this API request? the obligatory parameters to use

Assistant: The obligatory parameters to use in a Directions API request are:

1. \`profile\` (string): The routing profile to use, such as \`mapbox/driving-traffic\`, \`mapbox/driving\`, \`mapbox/walking\`, or \`mapbox/cycling\`.

2. \`coordinates\` (number): A list of coordinate pairs to visit in order.

5. can you provide an example of a request to API X?

can you provide a working example of a request to this api?

Sure, here's an example of a Directions API request:

```
https://api.mapbox.com/directions/v5/mapbox/cycling/-84.518641,39.134270;-84.512023,39.102779?geometries=geojson&access_token=<UserAccessToken>
```

In this example:

- The profile is `mapbox/cycling`, which means we're asking for a bike route.
- There are two waypoints: `-84.518641,39.134270` (the starting point) and `-84.512023,39.102779` (the destination).
- The `geometries=geojson` parameter specifies that we want the route to be returned as a GeoJSON feature.
- `<UserAccessToken>` should be replaced with your Mapbox access token.

Please note that you need to replace `<UserAccessToken>` with your actual Mapbox access token.

For more information, you can refer to the **Mapbox Directions API documentation**.

Source: **Getting started with the Mapbox Directions API**

Human: can you provide a working example of a request to this api?

Assistant: Here is a working example of a request to the Directions API:

```json
{
  "profile": "mapbox/driving-traffic",
  "coordinates": [
    [-122.42, 37.78],
    [-122.45, 37.91]
  ],
  "approaches": ["curb", "curb"],
  "layers": [0, 1],
  "radiuses": [10, 10]
}
```

In this example, we are requesting a driving route between two coordinates with specific approaches, layers, and radiuses specified for each waypoint.

6. can you provide python/java/... code for usage of api X?

Human: can you provide me with a working python code example of usage of the mapbox directions api?

Assistant: I can provide you with a Python code example for using the Mapbox Directions API. Here is a sample code snippet that demonstrates how to make a request to the Mapbox Directions API to get driving directions between two coordinates:

```python
import requests

# Replace 'YOUR_MAPBOX_ACCESS_TOKEN' with your actual Mapbox access token
MAPBOX_ACCESS_TOKEN = 'YOUR_MAPBOX_ACCESS_TOKEN'

# Define the coordinates for the start and end points
start_point = "13.43,52.51"
end_point = "13.42,52.5"

# Construct the API request URL
url = f"https://api.mapbox.com/directions/v5/mapbox/driving/{start_point};{end_point}?access_token={MAPBOX_ACCESS_TOKEN}"

# Make a GET request to the API
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    data = response.json()
    # Process the response data as needed
    print(data)
else:
    print("Error: Unable to fetch directions. Status code:", response.status_code)
```

Make sure to replace `'YOUR_MAPBOX_ACCESS_TOKEN'` with your actual Mapbox access token before running the code. This code snippet sends a request to the Mapbox Directions API to get driving directions between the specified start and end points.

If you have any questions or need further assistance, feel free to ask!

7. what does the request parameter of api X do?

in the directions api what does the layers parameter do?

The `layers` parameter is not directly mentioned in the context of the Mapbox Directions API in the provided sources. The Directions API primarily deals with routing and does not directly interact with map layers.

However, in the context of Mapbox, layers are used to define how specific sets of data should be rendered on the map. They determine the visual representation of data on the map. For example, in the Mapbox Streets v7 and v8 tilesets, the `layer` field is used to determine the drawing order of overlapping road segments in the tunnel and bridge layers.

If you're referring to a different context or API where the `layers` parameter is used, could you please provide more details or clarify your question?

Sources:

- **Mapbox Directions API**
- **Mapbox Streets v7**
- **Mapbox Streets v8**

Human: what does the layers parameter do?

Assistant: The Layers parameter in the Directions API influences the layer of the road from which a route starts from a specific waypoint. It is used to filter the road segment where the waypoint will be placed, particularly useful in scenarios like

multi-level roads such as tunnels under roads. The Layers parameter takes a single signed integer per waypoint. If provided, the list of layers must be the same length as the list of coordinates, but you can skip a coordinate and indicate its position in the list using a separator. This parameter corresponds to the "layer" field in Mapbox Streets v8. If a matching layer is not found, the API will choose a suitable layer based on other provided parameters.

8. what does the response parameter of API X mean?

what is the routes parameter in a response of directions API?

The `routes` parameter in the response of the Mapbox Directions API is an array of Route objects. Each Route object represents a possible route that a user can take from the origin to the destination.

Here's a brief overview of what a Route object contains:

- `geometry` : This property contains the route geometry, which you can use to display the route on a map.
- `legs` : This is an array that provides details about each leg of the route when the route has more than one waypoint. Each leg of the route will include information about the steps required to complete that leg.
- `weight` : This is the weight of the route. The weight indicates the cost of traversing the route, depending on the routing profile. For example, in a driving profile, the weight might be the time taken in traffic.
- `distance` : This is the total distance covered by the route, in meters.
- `duration` : This is the estimated travel time to traverse the route, in seconds.

Here's an example of what a `routes` object might look like in a response:

```
"routes": [
  {
    "geometry": {
      "coordinates": [
        [-84.518641, 39.134270],
        [-84.512023, 39.102779]
      ],
      "type": "LineString"
    },
    "legs": [
      {
        "steps": [],
        "weight": 1332.6,
        "distance": 4205,
        "summary": "",
        "duration": 1126
      }
    ],
    "weight_name": "cyclability",
    "weight": 1332.6,
    "distance": 4205,
    "duration": 1126
  }
]
```

For more details, you can refer to the **Mapbox Directions API documentation**.

Source: **Getting started with the Mapbox Directions API**

Human: what is the routes parameter in a response of directions API?

Assistant: The 'routes' parameter in the response of a Directions API contains an array of route objects ordered by descending recommendation rank. Each route object describes a route through multiple waypoints and includes properties such as duration (estimated travel time in seconds), distance (distance traveled in meters), weight_name (specifies the weight used for the route), and weight (numeric value representing the desirability of the route).

If you have any more questions or need further clarification, feel free to ask!

So as a conclusion the responses provided by our local chatbot are mostly correct, but there could be some improvements mainly:

- When it is possible to give out an example an example should be given.
- Sources of the answers are always given
- Links to the sources are embedded in the text, in the end after the keyword *sources* and as links right after a separator.
- Our local chatbot does not provide further details on its own so we need to ask for elaboration to get a bigger more detailed answer.
- When there are no local sources to add to the context there is no need to interrogate the llm, to avoid usage that does not answer questions that relate to the documentation and therefore to avoid unnecessary processing costs.

Default values of the sentenceSplitter of LLamaindex:

```
DEFAULT_CHUNK_SIZE = 1024  # tokens
DEFAULT_CHUNK_OVERLAP = 20  # tokens
```

Data scrapping

The first step of the document ingestion pipeline is getting data, because of the way the benomad documentation website is built, which offers many interactive functinalities unlike vanilla documentation websites one possible way to scrap the website is by building a crawler that will visit each page and at the same time save the details in a condensed format, such as mardown file, while ignoring irrelevant information and keeping track of metadata such as the website's link or the api version, service and so on.

The crawler will need to keep track of visited pages and will have to be able to detect different types of pages, we can see these types of pages at least:

- API reference pages which are generated in runtime from the bemap app using the annotation in java code.
- Example pages which contain interactive request/response interfaces allowing the users to test out some default functionalities of some APIs. Could be ignored.
- Tutorial pages which contain a step by step guide to bring out some ideas into real world using the bemap APIs.

Another thing to keep into considereation is the api version which could be chosen from a drop down menu. Each API version could have different pages associated with it.

Implementaion of crawler

The first page or the main page of the documentation website is protected by a login page. So we added a logic check before starting the actual crawling step.

The same data scrapper could be used on a similar website.

Debugging vLLM usage on docker

When trying to use vLLM instead of Ollama we've encoutered some problems, of of which is this error on the backend docker image:

```
2024-07-08 22:17:02 │    │    │    except ImportError:
2024-07-08 22:17:02 │    │    │    │    raise ImportError(
2024-07-08 22:17:02 │    │    │    │    │    "Could not import vllm python package. "
2024-07-08 22:17:02 │    │    │    │    │    "Please install it with `pip install vllm`."
2024-07-08 22:17:02 │    │    │    │    )
2024-07-08 22:17:02
2024-07-08 22:17:02  ╭─────────────────── locals ───────────────────╮
2024-07-08 22:17:02  │             api_url = ''
2024-07-08 22:17:02  │             best_of = None
2024-07-08 22:17:02  │    callback_manager = None
2024-07-08 22:17:02  │  completion_to_prompt = None
2024-07-08 22:17:02  │        download_dir = None
2024-07-08 22:17:02  │               dtype = 'float16'
2024-07-08 22:17:02  │   frequency_penalty = 0.0
2024-07-08 22:17:02  │          ignore_eos = False
2024-07-08 22:17:02  │            logprobs = None
2024-07-08 22:17:02  │      max_new_tokens = 100
2024-07-08 22:17:02  │   messages_to_prompt = None
2024-07-08 22:17:02  │               model = 'codellama/CodeLlama-7b-hf'
2024-07-08 22:17:02  │                   n = 1
2024-07-08 22:17:02  │       output_parser = None
2024-07-08 22:17:02  │    presence_penalty = 0.0
2024-07-08 22:17:02  │ pydantic_program_mode = <PydanticProgramMode.DEFAULT: 'default'>
2024-07-08 22:17:02  │                self = Vllm()
2024-07-08 22:17:02  │                stop = None
2024-07-08 22:17:02  │       system_prompt = None
2024-07-08 22:17:02  │         temperature = 0
2024-07-08 22:17:02  │  tensor_parallel_size = 4
2024-07-08 22:17:02  │               top_k = -1
2024-07-08 22:17:02  │               top_p = 1.0
2024-07-08 22:17:02  │   trust_remote_code = True
2024-07-08 22:17:02  │     use_beam_search = False
2024-07-08 22:17:02  │         vllm_kwargs = {
2024-07-08 22:17:02  │                       │   'swap_space': 1,
2024-07-08 22:17:02  │                       │   'gpu_memory_utilization': 0.5,
2024-07-08 22:17:02  │                       │   'max_model_len': 4096
2024-07-08 22:17:02  │                       }
2024-07-08 22:17:02  ╰──────────────────────────────────────────────╯
2024-07-08 22:17:02 ╰──────────────────────────────────────────────────╯
2024-07-08 22:17:02 ImportError: Could not import vllm python package. Please install it with `pip
2024-07-08 22:17:02 install vllm`.
```

To solve this we simply install vllm on our container, we will not create a seperate container for vllm, and change the default used image.

Allocated resources

Without defining limits and reservations on the resources used by the container we encounter one problem which is a very high CPU usage rate on the Ollama container,

To increase the maximum number of CPU cores used by docker as a whole and also modify the maximum allocated RAM size we create a file named .wslconfig inside C://users/ user-name/ and then we add a couple of lines such as:

[wsl2]
memory=20GB   # Limits VM memory in WSL 2
processors=18 # Limits the number of available processors

Then we restart wsl on our host machine, we wait a little big for the changes to take effect and to finally check that the number of maximum cores have been change for example we can run this command on wsl:

cat /proc/cpuinfo | grep processor | wc -l

Which return 18.

The deployed application seemed to not produce a response, but from the cpu usage and general activity of the containers we could see that it's just that the response generation is taking a very long time, which is past the previously chosen request_timeount of 180s, so we increased it to 600s (6min). The container that seems to be using so much resources is the ollama container, we still have a close to 1000% CPU usage.

Even changing the quatization level of the chosen model from q4 to 8b-instruct-q2_K did not seem to make the reponse time that much better.

To solve this problem, we install the required NVidea dependencies to work on the docker virtual machine.

Multi-threading

It is also possible to solve the concurrency problem using Ollama, it has two main environment variables that could affect this. The first defines the maximum number of loaded models and the second is the maximum number of concurrent requested allowed to be processed at the same time. By default the number is 3 if the host machines allows it. But the way it works is by pre-loading the language model as much time as defined in the environment variable therefore it will need three times the vRAM as 1 model. Or in the case of CPU processing 3 times of RAM.

Processing multiple requests at the same time could be done with the help of a solution different from Ollama, one possible solution is vLLM but it needs to run on a linux environment.

We will create a new vLLM image, the idea is to have the vLLM server running independently and LLamaIndex (the backend) should be able to communicate with it. There is an already provided vllm docker image "vllm/vllm-openai" that is tailored to work with the OpenAI API, we can use this image as a starting point and then from there we can work with another model such as codellama

VLLM will automatically download the needed files to run the LLM passed in as a parameter, one problem we encountered at this step is that there is no disk space left to complete the download. We can see on the host windows machine there is still some space left, therefore we can safely say that this is a docker settings problem. We'll need to allocate more disk space to our docker containers. according to docker's documentation when running on windows it uses wsl2 and by default the maximum disk space is 1Tb which was a bit confusing at first, turns out for some reason I needed to resize the virtual disk file under **C:\Users\fayss\AppData\Local\Docker\wsl\data**, to do so and according to this StackOverflow answer https://stackoverflow.com/questions/74236087/set-max-disk-usage-for-docker-using-wsl2I can do it by using the Resize-VHD tool. But to use this tool I'll need to have the hyper-v of windows activated and I have windows 11 home, only pro has it. So I looked for another solution which is Qemu by running the command: qemu-img resize ./ext4.vhdx +40 to add 40Gb to the virtual disk. Another problem is that th vhdx format does not allow resizing so we'll resize it to QCOW2 resize it then convert it back.

But still, we shouldn't have to resize the vhdx disk size manually, it should adapt to the needed size according to wsl and docker's documentation.

Using the default vLLM packages requires having a CUDA capable host machine, we do not plan on using a graphics card for inference in the server therefore it is highly unlikely to go with this option. There is a CPU vLLM variant that could come in handy.

I tried out the vLLM CPU image that is available in their official github repository. According to the documentation to use it I should start off by running:

docker build -f Dockerfile.cpu -t vllm-cpu-env --shm-size=4g .

But I got an error message telling me I should run this:

pip install -v -r requirements-cpu.txt --extra-index-url https://download.pytorch.org/whl/cpu

which finally solved the problem.