# Building a generative AI-enabled chatbot for generating Fog/Edge Emulation deployments

Antonis Louca
Panayiotis Papadopoulos
Andreas Chrisanthou
louca.antonis@ucy.ac.cy
papadopoulos.m.panagiotis@ucy.ac.cy
chrysanthou.m.andreas@ucy.ac.cy

## ABSTRACT

Developing a generative AI-powered chatbot tailored for Fog/Edge Emulation deployments represents an exciting convergence of cutting-edge technology. This endeavor draws inspiration from the capabilities of advanced Generative AI models such as ChatGPT, which excel in comprehending and generating human-like text. The primary objective here is to harness the potential of these models to streamline and enrich the process of configuring Fog/Edge Emulation systems. To achieve this, we design a user-friendly API that allows clients to submit their requirements for a Fog Computing infrastructure. The system will then extract relevant information from these inquiries and augment them with the corresponding context. This augmentation will be achieved through the implementation of prompt engineering techniques and in-context learning. Subsequently, the system will transmit these enhanced queries to a powerful large language model (LLM), such as the ChatGPT API, and relay the LLM's responses back to the client. In this project we assume Fogify as the underlying emulation engine, and we create prompt engineering, using the modeling abstractions provided by Fogify's dedicated documentation page.

**ACM Reference Format:**
Antonis Louca, Panayiotis Papadopoulos, and Andreas Chrisanthou. 2023. Building a generative AI-enabled chatbot for generating Fog/Edge Emulation deployments. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The contemporary surge in the capabilities of generative AI models, exemplified by GPT, has catalyzed significant advancements. In this project we make the first steps in developing an AI-powered chatbot tailored for Fog/Edge emulation tools. We use the Fogify [4] tool and tailor our chatbot to answer questions about the documentation and API of the Fogify python library. A main goal is the design of a user-friendly API enabling clients to articulate Fog Computing infrastructure requirements.


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

We leverage prompt engineering and contextual learning, our system manipulates user queries, and interacts with a Large Language Model (LLM) from the OpenAI API [3] to create embeddings and get answers for particular questions. We incorporate Fogify as the primary emulation tool, by using its documentation. This ensures optimal outcomes during prompt engineering. This project report highlights the integration of AI technologies for streamlining Fog/Edge emulation processes.

## 2 BACKGROUND

In this section we provide some general knowledge about the terminology we use and the tools used in our project. A large language model is a deep learning algorithm that performs tasks regarding natural language. These tasks are called natural language processing tasks (NLP). In this project we do not create and train a LLM model from scratch instead, we use the OpenAI API for our requests and leverage it with prompt engineering. Prompt engineering is the process of structuring text in a way that a LLM can understand and process. In layman's terms is a way to tell the LLM what tasks should perform. We implement our chatbot using Python programming language and some useful libraries along with a vector store database.

A vector store database is a kind of database that stores data using mathematical representations. These mathematical representations are called vector embeddings. Vector embeddings is a very useful as they are a way to convert words and sentences into vectors of numbers that capture their semantic meaning and relationships. These embeddings are stored in a special type of database called a vector store database. Below we present the different tools we used in this project and how.

(1) LangChain [2] is a framework designed to simplify creation of applications using large language models (LLMs). We use LangChain to easily communicate with the vector store database and the OpenAI API.
(2) We used Chroma DB [1] as a vector store database. It is an open-source vector store database, which provides the tools for building a knowledge base for LLMs. Chroma is an AI-native database, and can be easily used with Langchain. It also operates as a Python library, hiding the complexity of a native database.
(3) OpenAi is service that allows developers to integrate advanced natural language processing capabilities into their applications. We incorporate the OpenAI API to create embeddings for our data and query the LLMs offered by the

service such as the "GPT 3.5 turbo" which is the LLM we use in our project.

(4) Fogify is an emulation Framework for modelling, deploying and experimenting with fog testbeds. Provides a toolset to model complex emulated fog topologies. We use the Fogify documentation to create a dataset that will be used in the prompt engineering when querying the LLM.

(5) Python is a high-level programming language known for its readability and simplicity. We use Python to create the chatbot and interconnect the various components mentioned above.

# 3 METHODOLOGY

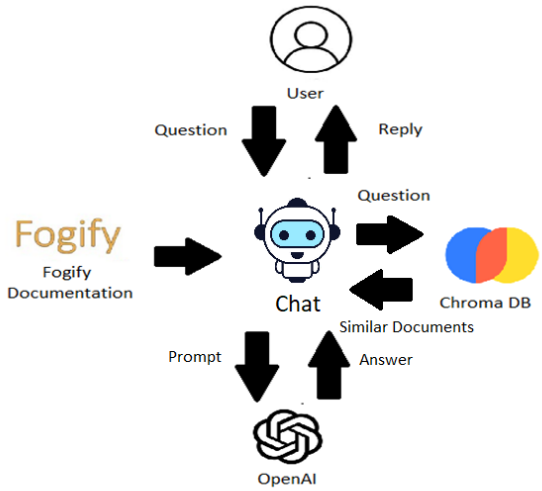


**Figure 1: Methodology pipline for creating chatbot tool**

To tackle our problem we create the methodology shown in image 1. Specifically, we use the documentation of the Fogify tool, We started our process by creating the dataset form the Fogify documentation. We focus on the html pages of the documentation. We filter out the different metadata from the html webpages and keep only the useful data.

We use the OpenAI API to create embeddings for the data we extract and store it in our vector database. When we take a query from the user we perform prompt engineering to create a sufficient input for the LLM. In the promt engineering process we incorporate the Fogify relevant results we get from the database. We use the prompt as input for the LLM using the OpenAI API.

# 4 IMPLEMENTATION

## 4.1 Documentation

In order to have a complete image of what the Fogify tool is and what it can do, we downloaded the documentation. The documentation is made up of various HTML files that contain information about the tool, how it works and its different variable configurations. It also contains a large number of files that compose the Fogify tool. We only focus on the HTML files that contain data and explanations about the Fogify tool. We download the documentation by cloning the GitHub repository of the Fogify tool.

We then go through the repository, read the HTML files, remove the tags and the HTML metadata, and then recursively split the text to chunks of 1000 tokens with a 300 token overlap.

## 4.2 Vector Store

Having the documentation itself is not enough to do what we need. Searching text-based files for information would be very difficult. The difficulty in the search is extracting the most relevant information. Simple text searches would yield many results but most of them would be irrelevant to what we need.

To tackle this issue, we used a vector store database. A vector store is a collection of data in the form of vectors that can be queried to retrieve the most relevant data. We store the chunks we created above along with their embeddings in the ChromaDB vector store. Using a vector store we get the most relevant results of a search by also taking into consideration the semantic meaning of the sentences compared.

Creating the vector store does require some steps. We first decide the vector store engine to use for our application. We use ChromaDB since it is well integrated with the Langchain library. Inserting the documentation in the vector store requires two steps. First, we read all the files and break them up into smaller pieces. After breaking the files into smaller pieces, we transform these pieces into embeddings that can be stored in the vector store. We use embeddings of OpenAI. Langchain provides an easy method of using the OpenAI embeddings in this process and since we aim to use a LLM from OpenAI we take advantage of it. Finally, when all the pieces have been transformed into embeddings, we insert them into our vector store.

By default, the vector store is stored in memory and is lost when the program/script exits. In order to save our vector store and then restore it during the actual use of the chatbot we use persistent storage. We dump the database in a file from which we can load it back into memory later.

## 4.3 LLM

At this point we store all the documentation of the Fogify tool inside a vector store that we can query to get the most relevant data. In order to make sense of the relevant data we use a Large Language Model, specifically "gpt-3.5-turbo". This LLM accepts what information we give to it and returns a response based on what it was given.

To have a reasonable response we need give our LLM three things. First, we provide the LLM with our question. Second, provide some context to the LLM to make sense of what the question is about along with the information that it needs to give us a proper response. The context is provided by the vector store database and contains the most relevant information to the question of the user. Basically, the context is the result of performing a similarity search in the data of stored in the vector store database. Third, we keep some sort of memory of previous conversations between the user and the LLM. These three pieces of information compose our prompt and

theoretically provide the LLM enough context for which to answer with a reasonable response to the user's question.

We also add some other data in the prompts such a welcoming message that explains what the conversation between the chatbot and the user will be about. The process we described is called prompt engineering.

## 5 EVALUATION

In this section we evaluate the chatbot tool we created for this project. We use a set of questions to evaluate the performance of the chatbot. For question we provide the best response from our chatbot along with a piece of context we have from the documentation of the Fogify tool. This way we compare the results of the chatbot in comparison with the context of the documentation. Below we provide the set of questions we asked our chatbot.

*Questions used in the evaluation stage.*

(1) What is an action and how can i add an action using python code?
(2) What are some parameters that we specify in the Fogify emulator for network connectivity? Can you give me an example of yaml configuration?
(3) What are blueprints and what are their different variables? Can you also give me an example?
(4) How does Fofigy compose the network latency can you give me an example of configuration with the different variables in that yaml?
(5) Explain the different parameters of the blueprint.
(6) What are the performance metrics captured by Fogify. Can you give me the metric and a small description for it?
(7) Can a user add its own metrics to fogify? How can a user do that can you give me an example?
(8) How can someone interact with the emulated testbed?
(9) Can you give me an example in python code on how i can use the fogify SDK library to interact with the Fogify emulation tool?

## 6 FUTURE WORK

Future work for this project involves the improvement of all pieces that make up our implementation.

The most obvious candidate is the vector store. Since the LLM provides responses based on the information we give to it, the better the information that we give to it the better the responses will be. As such, we can try to improve the process of storing the documentation in our vector store. This includes the breaking of files into smaller pieces, removing irrelevant data from our files such as HTML tags, metadata, or other data that is not information such as HTML tags for buttons.

Breaking up our files into smaller pieces can also help us in providing more detailed information. We can also add to our context more information instead of the most relevant from our query. WE can give it the top three results from our query.

Finally, we can try and improve our queries into the vector store. We might have relevant information in our vector store that isn't brought up because our query isn't actually the best for what we are looking for.

The second part of our implementation that we can look into improving is our LLM. We use a pre-trained model that is provided by OpenAI, this is because designing and implementing a LLM model is not a easy task. There are multiple other models that we can use, and we can use these other models to make comparisons.

The most effective approach would be to use a publicly available model and train it to better understand the information we give to it. This requires a lot of time and data not to mention knowledge of Machine Learning.

## 7 CONCLUSIONS

## REFERENCES

[1] ChromaDB Authors. [n. d.]. ChromaDB Documentation. https://www.trychroma.com/on. Accessed: Insert Date Accessed.
[2] Langchain Authors. [n. d.]. Langchain Official Documentation. https://python.langchain.com/docs/get_started/introduction. Accessed: Insert Date Accessed.
[3] OpenAI. [n. d.]. OpenAI API Documentation. https://beta.openai.com/docs/. Accessed: Insert Date Accessed.
[4] UCY LINC Lab. [n. d.]. Fogify: Fog Computing Simulation Tool. https://ucy-linc-lab.github.io/fogify/. Accessed: Insert Date Accessed.