Databases Project Proposal

An LLM Chatbot That Has Context Over a Large Corpus of Custom Data Using a Vector Database

Olsen Budanur
Niya Ma

Glossary

Semantic Search: “Semantic search seeks to improve search accuracy by understanding the searcher's intent and the contextual meaning of terms as they appear in the searchable dataspace, whether on the Web or within a closed system, to generate more relevant results.”

Embeddings: “In natural language processing, a word embedding is a [vector] representation of a word.” An embedding of a word gives it a location in a higher dimensional plane, that is relevant to its semantic meaning. So, words or phrases that have similar semantic meanings will be close to each other in their embedding dimension which can allow us to do semantic search.

Vector Database: “A vector database is a type of database that is specifically designed to store and query high-dimensional vectors.”

High Level Explanation

We would like to build a chatbot interface that uses an LLM to answer users. This chatbot would have context of both the chat context (prior texts), and a large corpus of arbitrary text.

The way the bot would have context of the prior texts is simple. Modern LLM models can accept inputs up to ~8,000 characters, which should allow us to send the last N messages a user has sent for every query (N will most likely be ~5-10). We can easily store the last N messages in memory of our UI interface in a circular-queue fashion.

However, a “large” corpus of data can be up to tens of thousands, or millions of characters. So, we cannot send all of it to the LLM as context. The solution for this is sending to the LLM only the parts of the data that are relevant to the question the user has asked. But, how can we decide what parts of the text are relevant?

We can achieve this using embeddings, a vector database, and semantic search. We can first vectorize the large corpus of text using an embedding model (most likely provided by an API), then store it in a vector database (or a traditional relational database). Then, when we want to retrieve a piece of this large corpus of text that might be relevant to our question we can embed the question (or a variation of the question), and do a nearest neighbor search against the vector database to find the parts of the data that is most relevant to the question. Lastly, we can feed these parts to the LLM as well as the users question.

Implementation Overview

This project would be composed of 2 main parts/phases. The first part would be for storing the large corpus of text in a vectorized form after embeddings. The second phase would be for letting a user ask a question through a simple UI, query the vector DB for related parts from the large corpus of text, and feed the related parts to an LLM as well as the users question to provide the user with an answer.

Part 1:

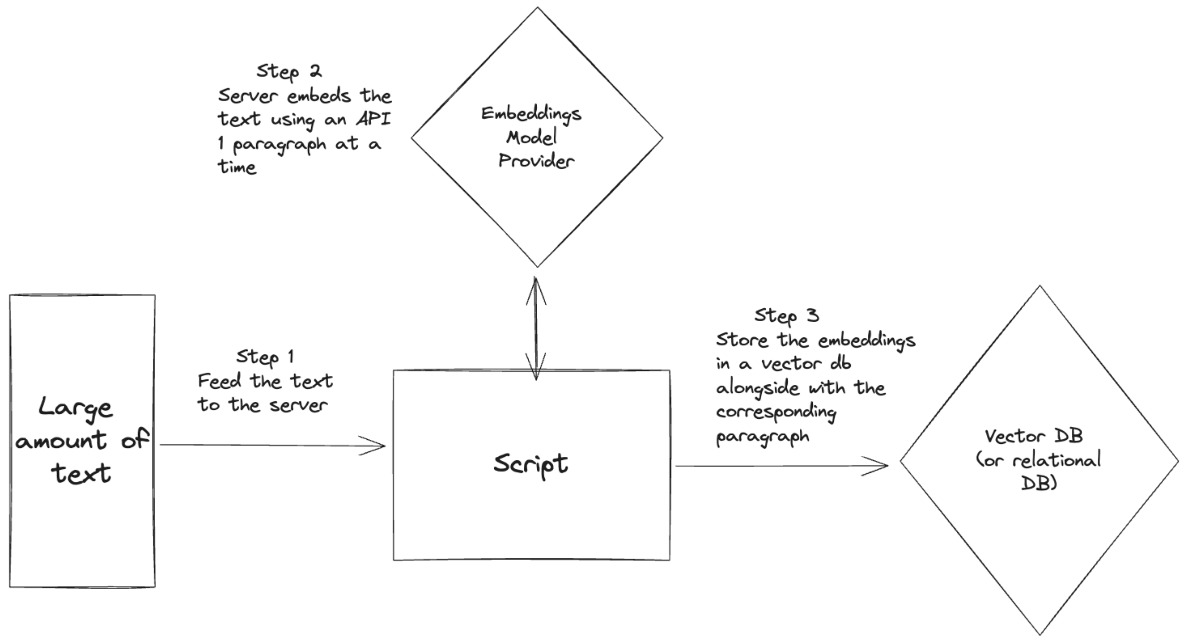
Step 2
Server embeds the text using an API 1 paragraph at a time
Embeddings Model Provider
Large amount of text
Step 1
Feed the text to the server
Script
Step 3
Store the embeddings in a vector db alongside with the corresponding paragraph
Vector DB (or relational DB)

Figure 1: Embed & Store

During this part, we will be populating our vector DB. Let's think of the vector DB as a relational SQL database. First, we will create a table to store our embeddings/text:

```
CREATE TABLE IF NOT EXISTS vector_db(
   paragraph: TEXT
   vector: BLOB

);
```

Figure 2: Pseudo Code for Creating Vector DB

Now that our vector DB is ready, we will read a large corpus of data, partition it into pieces (or paragraphs), embed these paragraphs 1 by 1, and then feed it to the DB.

```
with open("large_corpus_data.txt", "r") as file:
    for line in file:
        embedding = API.embed(line)
        SQL(`INSERT INTO vector_db (text, vector) VALUES (line, embedding)`)
```

Figure 3: Pseudo Code for Populating the Vector DB

Part 2:

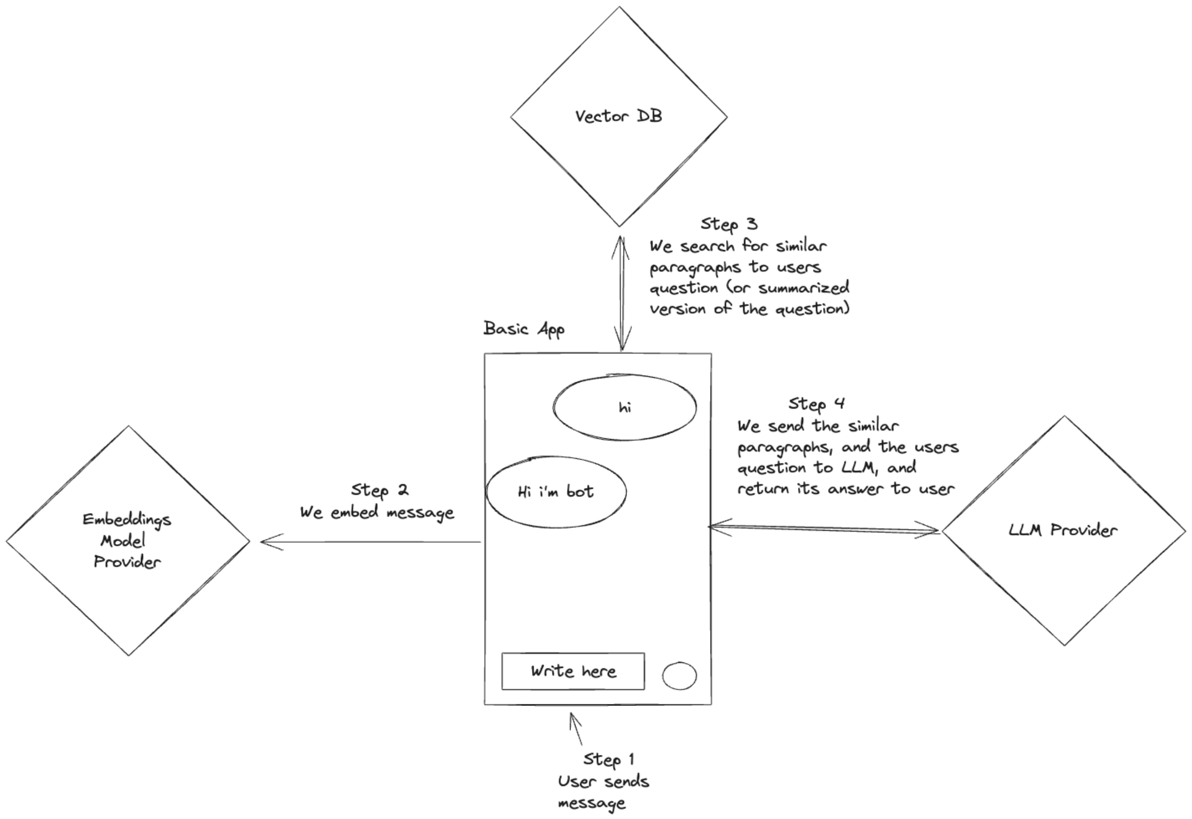
Vector DB
Step 3
We search for similar paragraphs to users question (or summarized version of the question)
Basic App
hi
Hi i'm bot
Write here
Step 4
We send the similar paragraphs, and the users question to LLM, and return its answer to user
LLM Provider
Step 2
We embed message
Embeddings Model Provider
Step 1
User sends message

Figure 4: Chatbot App

During this part, we allow the user to submit a question through our app, vectorize the users' question, search for parts of the text that are similar to the users question, feed the relevant parts of the text + the users question to the LLM, and display to the user the response of the LLM.

```
#
# Embed users question (this will be done in a ui, and we
# will keep the past N number of users questions as chat context)
user_question = input("Please enter your question: ")
user_question_embedding = API.embed(user_question)

#
# Get relevant parts
# (distance could be a dot product... but we will
# most likely use a vector db service instead of
# doing this query ourselves.
relevant_paragraphs = SQL("
SELECT paragraph, distance(user_question_embedding, vector) as
closeness
FROM vector_db
ORDER BY closeness DESC
LIMIT 3;
")

#
# Create llm prompt, and return to user
# (again this will be in UI)
llm_prompt = `
   Given the context:
   ${relevant_paragraphs}

   Answer the following question:
   ${user_question}
`

answer = LLM(llm_prompt)
print(answer)
```

Figure 5: Pseudo Code for The Application Logic

Possible Technologies

UI:

- HTML + CSS + JS STACK

Embeddings:

- OpenAI embeddings

LLM:

- OpenAI 3.5-turbo

Vector DB

- Traditional MySQL, hosted on a cloud provider or locally.
- Vector DB like SingleStore