

Leveraging the Power of ChatGPT and Vector Database in the FreeBSD Expert System

Yan-Hao Wang
National Chung Cheng University
Chiayi, Taiwan
bse30074@gmail.com

Abstract—Expert system is designed to emulate the expert, solving complex problems. Modern expert system usually uses machine learning to simulate the behavior of the domain experts. In this paper, we use Retrieval-Augmented Generation with vector database and ChatGPT to develop a FreeBSD Expert System.

Keywords—Expert system, Retrieval-Augmented Generation, Large language models, Vector database

I. INTRODUCTION

Traditional expert system architecture is composed of two parts

1. Knowledge base which stores all the relevant information related to the domain of expertise.
2. Rule Engine: Contains some predefined rules by data scientists. It processes the user's questions and applies rules to generate accurate responses.

Modern expert systems often use the machine learning model as the main component of it.

Large language models (LLM) have shown the potential to be the spindle of the expert system. One of the famous LLM models is ChatGPT which is hosted by OpenAI. People can use it without hosting by themselves which needs huge computility.

However, ChatGPT has two limitations. First, ChatGPT uses the data before September 2021, so they don't get the newest data. Second, ChatGPT is more like a general-purpose model, it can answer different domain questions but it may tend to hallucinate answers when asked about unfamiliar domains.

There are three solutions to solve such issues.

1. Retrain a Model: Need the AI expert to train and costs a lot to me.
2. Fine-tune the ChatGPT: Still a hard job for AI-unfamiliar developer. And it also cost a lot.
3. Retrieval-Augmented Generation (RAG): Our solution.

In this paper, we use the third solution RAG, which is an acceptable way no matter on cost and development aspect. So we use the embedded model and vector database to achieve this.

II. DEVELOPMENT

A. Data Cleaning and Extraction

We extract data from the FreeBSD document [1] and FreeBSD source code [2] with the find command and use the "ps-pandoc" [3]. Basically, the FreeBSD document contains html file and the source code contains *.number file which is the man page source code. The process is shown in Fig 1.

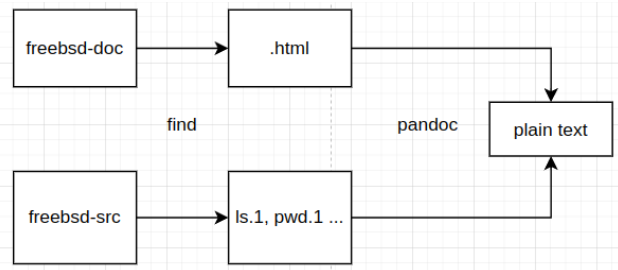


Fig. 1. Data extraction process

Then we still use the find command to remove the unrelated data, for example, the “Download PDF” texts in the FreeBSD document as Fig 2 shows. Fig 3 shows the find command to remove unrelated data.

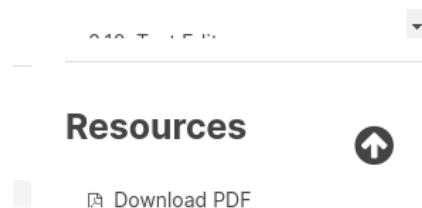


Fig. 2. Unrelated text in FreeBSD document

```
# Remove irrelevant content in the text files
find "data/doc" -type f -name "*.txt" -exec sh -c 'awk "/^[FreeBSD
logo\]$/, /♥ Donate$/ {next} 1" "$1" > "$1.tmp" && mv "$1.tmp"
"$1" - {} \;';
find "data/doc" -type f -name "*.txt" -exec sh -c 'awk "
/^trademarks$/,/^symbol\.$/ {next} 1" "$1" > "$1.tmp" && mv "$1
.tmp" "$1" - {} \;';
find "data/doc" -type f -name "*.txt" -exec sh -c 'awk "
/^Resources$/,/^ Edit this page$/ {next} 1" "$1" > "$1.tmp" &&
mv "$1.tmp" "$1" - {} \;';
find "data/doc" -type f -name "*.txt" -exec sh -c 'sed "/^--*/d"
"$1" > "$1.tmp" && mv "$1.tmp" "$1" - {} \;';
```

Fig. 3. Find command to clean data

B. Embedded Model and Vector Database

An embedded model is a type of machine learning model used to convert input data, such as words or sentences, into numerical representations called embeddings or vectors. These embeddings capture the semantic meaning or context of the input data in a continuous vector space. When two vectors are close, it means the corresponding sentences are very similar. Embedded models are often employed in natural language processing tasks such as text classification and sentiment analysis.

A vector database is a collection of data stored as mathematical representations. Vector databases make it easier for machine learning models to remember previous inputs, allowing machine learning to be used to power search, recommendations, and text generation use cases.

OpenAI has embedded model API, there are multiple open source embedded models online too. In this project, we use the open-source model from Hugging Face. We choose the model according to the MTEB Leaderboard [4] on Hugging Face. The model can be changed arbitrarily.

We use the embedded model to process all the data from the last step and store it in a vector database as Fig 4 shows. There are also a lot of open-source databases, that can store the vector from embedded models and have different search algorithms. But in our case, we just use a file to store the vector and a simple cosine similarity algorithm. Because our data is not big.

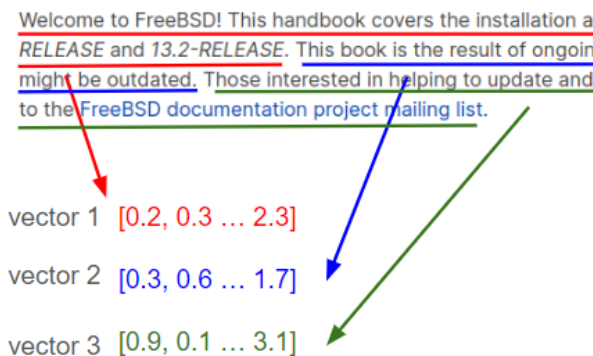


Fig. 4. Embedded model processes the sentence to vector

There are multiple facts (hyperparameters) we can tune here. For example

- The length of sentences.
- What metadata should we leave.
- What model should we use.
- Whether we need to tune the embedded model.

All these hyperparameters should be tried multiple times to get the best answer. The answer will be different with different fields.

Fig 5 shows the query result of the question “What is gunion in FreeBSD”. The query result is the sentences for the gunion man page which is very reasonable.

Fig 6 shows the ChatGPT answer without and with related knowledge. The top is the answer without additional information and it doesn’t even know what is gunion because the data it learns is too old. The below obviously answers much better.

C. Integration with ChatGPT

Now our architecture is like what Fig 7 shows.

1. User sends a question to the embedded model
2. Embedded model generates question vector database
3. Vector database returns the origin question and related knowledge. There is another hyperparameter we can tune here is the return sentence number.
4. ChatGPT answers the question and returns it to the user.

```
(base) kola:~/git-project/freebsd_data$ python query.py
Current question: What is gunion in FreeBSD?

Related sentences 1: GUNION(8) System Manager's Manual
Utility for UNION GEOM class SYNOPSIS gunion create [-v] [-o of
lowerdev gunion destroy [-fv] prov ... gunion reset [-v] prov ...
... gunion list gunion status gunion load gunion unload DESCRIPTION
read-only disk on a writable disk. Logically, a writable disk
intercepted and stored on the writable disk. Read requests are f
op (writable disk) and if found

Related sentences 2: are returned. If they have not been writ
r disk. The gunion utility can be especially useful if you have a
unsure of how to repair. You can use gunion to place another dis
r the filesystem. If the repair fails, you can revert all the cha
ate of the lower disk thus allowing you to try another approach t
an request that all the writes recorded on the top disk be written
is to try out upgrades to your sys- tem. Place the upper dis
pgraded and then run the upgrade on it. If it works, commit it;
below
```

Fig. 5. The graph of the query by embedded model and vector database

You

What is gunion in freebsd.

ChatGPT

It seems there might be a typo in your question. There is no specific reference to "gunion" in FreeBSD. However, if you meant "union" in the context of FreeBSD, it's a feature that allows a file system to appear to be both read-only and writable.

In FreeBSD, gunion is a control utility for managing the UNION GEOM class. The primary purpose of gunion is to facilitate tracking changes to a read-only disk on a writable disk. This is achieved by placing a writable disk over a read-only disk. When write requests occur, they are intercepted and stored on the writable disk. Read requests are first checked on the writable disk, and if the data is found there, it is returned. If not, the data is retrieved from the lower, read-only disk.

Fig. 6. ChatGPT answer without related knowledge and with related knowledge

And we need to combine them all together with ChatGPT. There are two ways to achieve this.

1. Easy just implement with ChatGPT API.
2. Develop ChatGPT plugin [5], plugin can let us set some API and while asking questions ChatGPT, it will call the API and get a response. This is the best practice for of project. However, we need to consider the cost of hosting the embedded model and vector database.

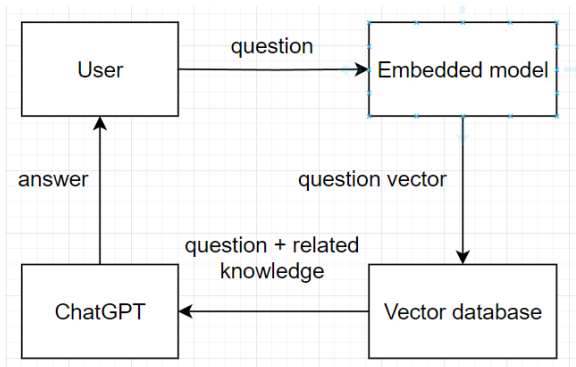


Fig. 7. Architecture of the project

We have implemented the first case and decided not to keep going with the second case because of the release of GPTs.

III. OPENAI GPTs AS POTENTIAL REPLACEMENTS

GPTs were launched in November 2023 [6]. It provides an easy way to generate a custom GPT for any data you have. Which becomes a potential replacement for our project. We only need to upload the data from development step A and there is a custom expert system.

On March 19, 2024, you will no longer be able to install new plugins or create new conversations with existing plugins. Based on this information, we decided to not keep going with the project.

IV. SUMMARY

The significance of LLM is poised to exponentially increase in the future, marking a pivotal shift in our technological landscape. While we may not always complete the production process in its entirety. But it is a good thing to focus on any future trends and try to combine them with FreeBSD. No matter on the FreeBSD document or the OS itself. All of the codes are placed on my GitHub repository [7].References

- [1] FreeBSD document, <https://github.com/freebsd/freebsd-doc>
- [2] FreeBSD src, <https://github.com/freebsd/freebsd-src>
- [3] Pandoc package, <https://www.freshports.org/textproc/hs-pandoc/>
- [4] MTEB: Massive Text Embedding Benchmark (Muennighoff et al., EACL 2023)
- [5] ChatGPT plugin document, <https://platform.openai.com/docs/plugins/introduction>
- [6] ChatGPT GPTs introduction, <https://openai.com/blog/intro>
- [7] The repository of this paper, https://github.com/Wang-Yan-Hao/freebsd_data