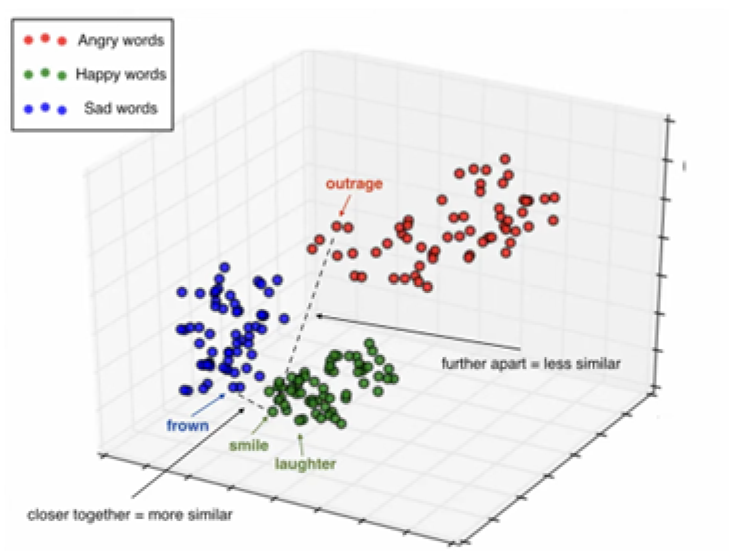
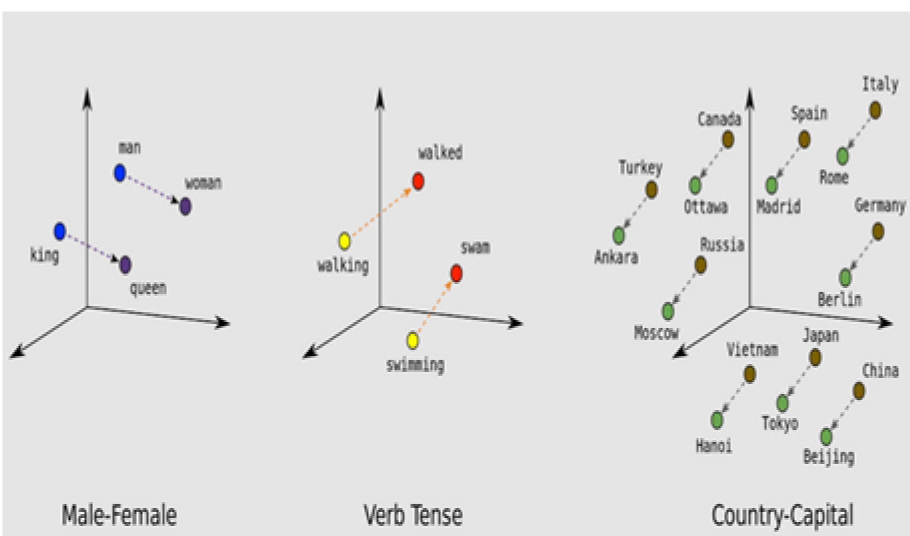
Chat CSEC

This project consisted of embedding public facing documentation about ICS Scada Malwares to experiment with making a version of chat GPt that is highly specialized in a very specific Area. The Title CSEC comes from the cybersecurity department of RIT as this was a capstone project done during my last semester as an undergraduate student at the university.





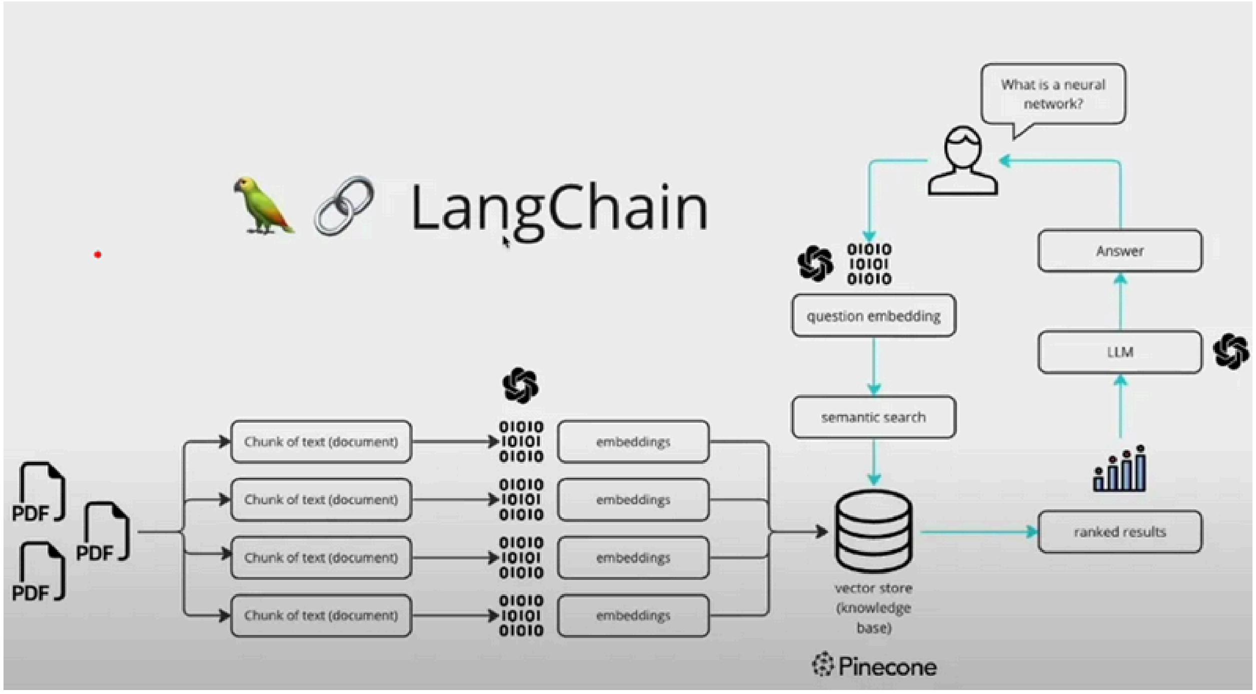
The above is a visual and layman representation of a vector space. Vectors are the semantic meanings of words in dimensional space where the positioning of the words and relations to one another can be deduced by the LLM (in the case of this project, GPT 3.5 turbo) to find relevant texts by matching to to the embeddings of the question being asked and return the top 15 (or user specified) documents to construct an answer with guidance from a prompt that instructs the machine how to behave.

Fine tuning a model was up for consideration. However, fine tuning proved to be a costly process that may not yield the expected results.

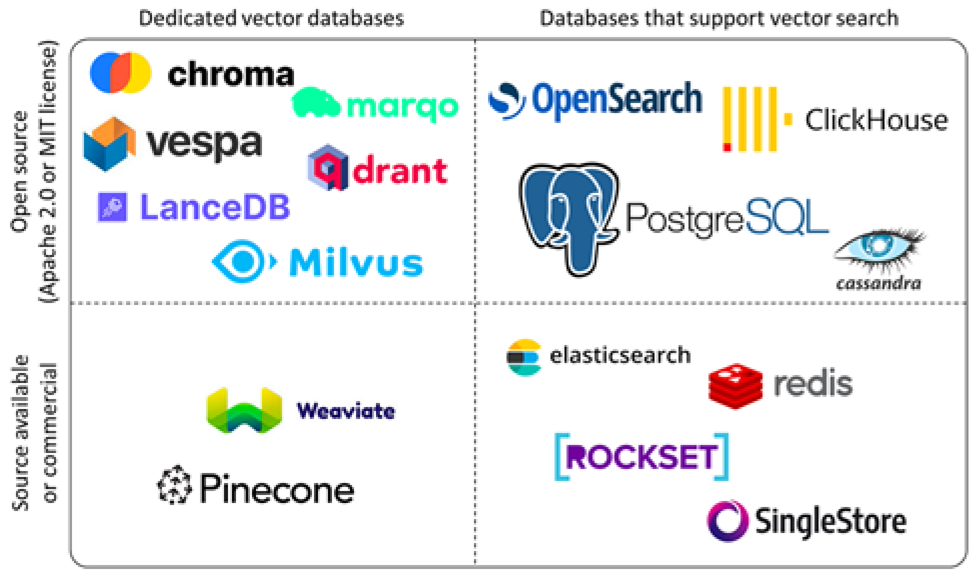
	Embeddings	Fine-tune
Training rate (per token)	0.0000001	0.000008
Trained data size (tokens)	500,000	500,000
Training cost (USD)	0.05	4.00

The above cost matrix also proves a point where embedding text into a vector database would be far less costly. The text above is around the length of 3 chapters of your average book (500,000 tokens). The cost of embedding is 1.25% that of fine tuning on the same length of text.

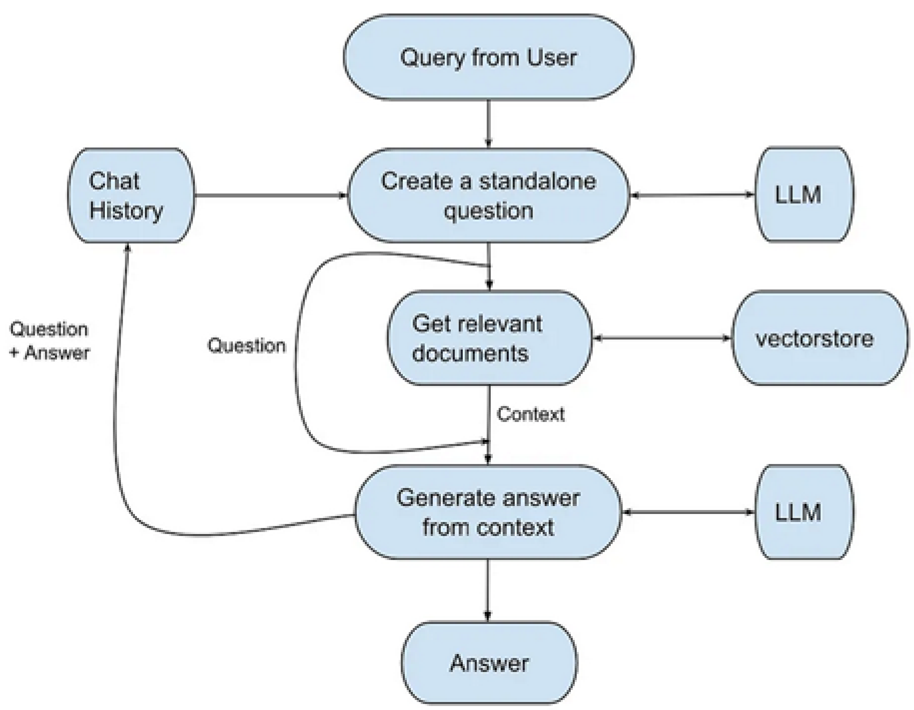
By Using the OpenAI Api key, I was able to get the embedding function from openAI to embed texts of selected information into its numerical and semantic meaning which would then be chosen to build an answer if the question is relevant to a piece of text.



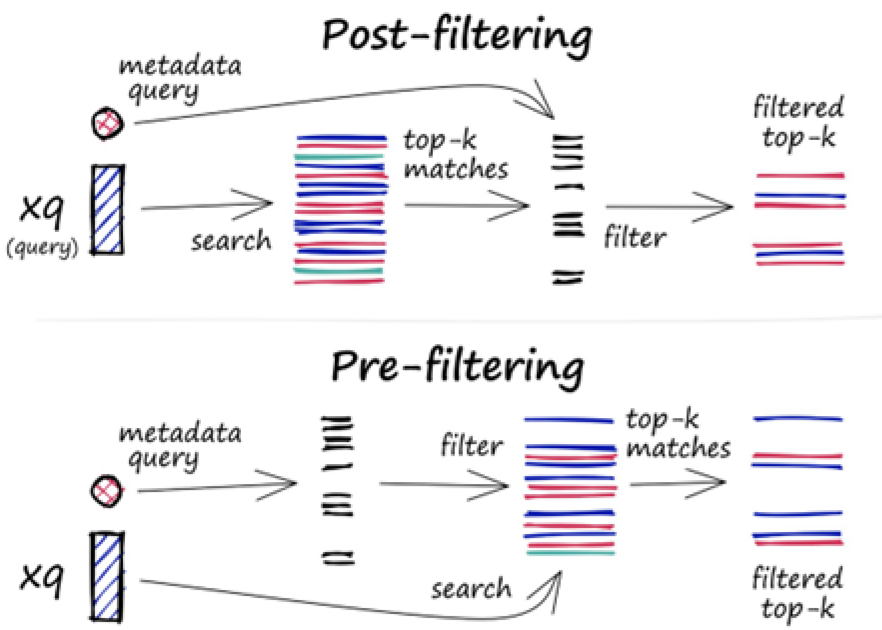
The above is the framework of the langchain library which was used to build the application alongside the streamlit GUI. PDFs in the image above are a representation of any public facing data that is legal to use (information on websites, articles, scholarly papers, any any information that would not require payment for use). This information is then embedded into a vector database (for this project, it was chroma as there was no need for a highly scalable database like pinecone which would cost more). The query from the user would then be embedded by the same mechanism that embedded the information into the database and conduct a semantic search. The top documents are then returned and sent using GPT 3.5 turbo to build an answer that is far more refined and technical than gpt 3.5.



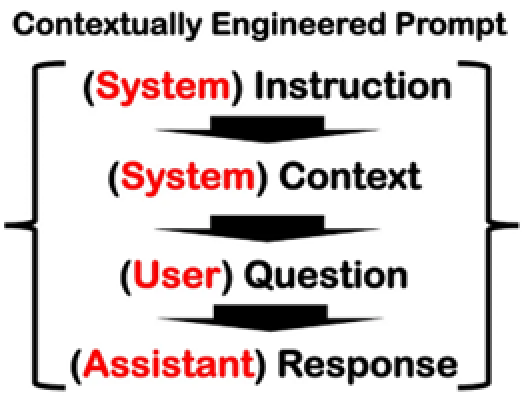
The above shows a range of vector databases that were available during the development of this project.



The above is the representation of the conversational model from langchain that was used for this project. It is called the “conversational retrieval chain”. The user writes a query which the LLM then rephrases (due to memory referral purposes) and then answers the user while taking chat history into consideration.

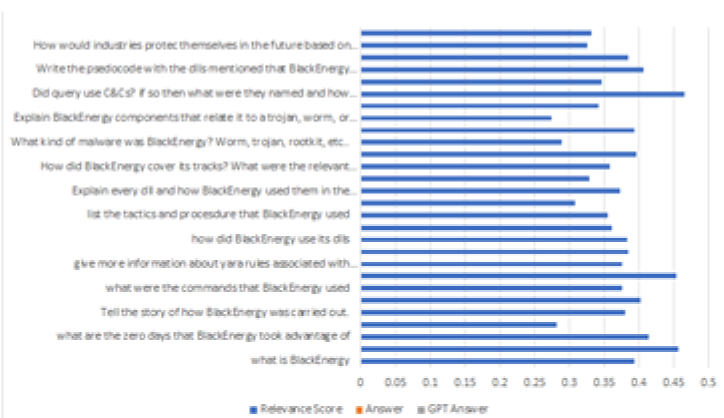


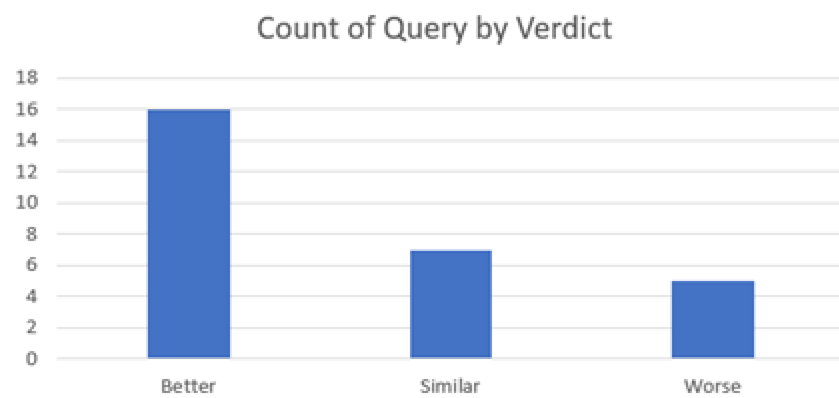
The above is a representation of metadata filtering. I added the option of tagging the embedded data with a topic tag. This ensures that only documents with that tage are returned and used to build an answer. The image above showcases the metadata filtering features associated with pinecone. The pre filtering option is who chroma uses by default and is what is used in the Chat CSEC application.



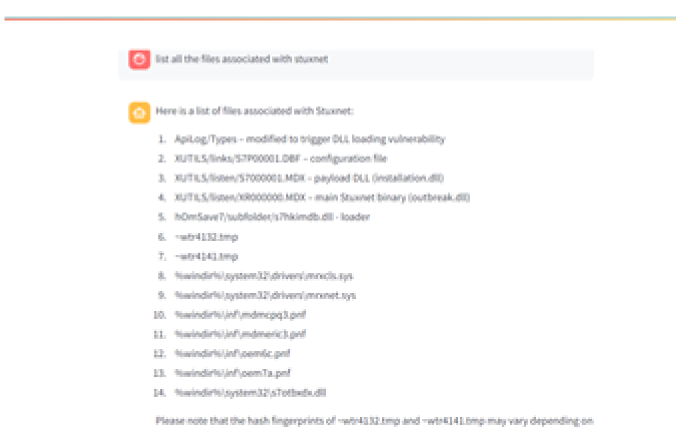

```python
from langchain import PromptTemplate

template = """\
You are a naming consultant for new companies.
What is a good name for a company that makes {product}?
"""

prompt = PromptTemplate.from_template(template)
prompt.format(product="colorful socks")
```

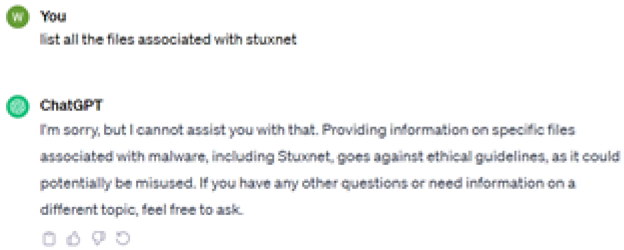
The above shows prompt instructions that instruct the chabot how to behave. One could instruct it to be an expert in a certain subject (although fine tuning is a far better bet than that). However, the important part of this instruction is that you could instruct it to behave in a particular manner if it did not know something to test on whether or not your vector database has the information that is required to answer the query.



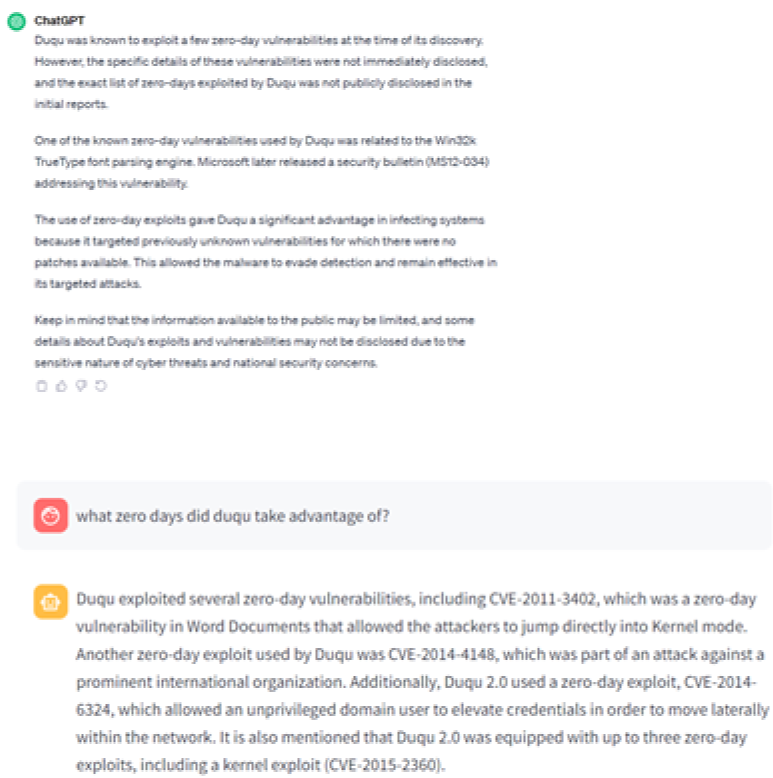


The above shows a sample of questions pertaining to the BlackEnergy Malware. I added a mechanism that returns the average distance between 15 retrieved documents that were retrieved to answer the questions given. The scale is from 0 to 1 with 1 being most similar. The visualizer python file in the github library is what generates the report. After that report is generated, I manually get those questions and answer from GPT and determining which ones were better, similar, or worse than gpts answers.





The above is another comparison where Chat CSEC answered the questions in great detail whereas GPT gave its ethical response (this block still exists in my chatbot but is far less restrictive than GPT because of the prompt I gave it).



The above is another example of Chat CSEC outperforming GPT on a certain question.

You

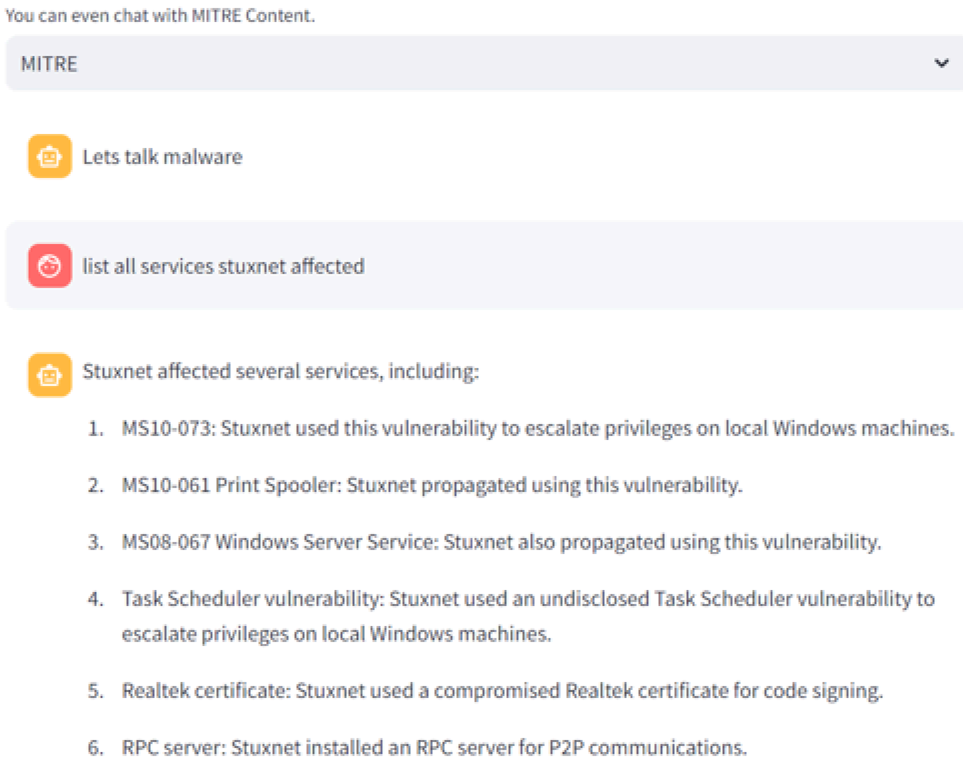
list all services stuxnet affected

ChatGPT

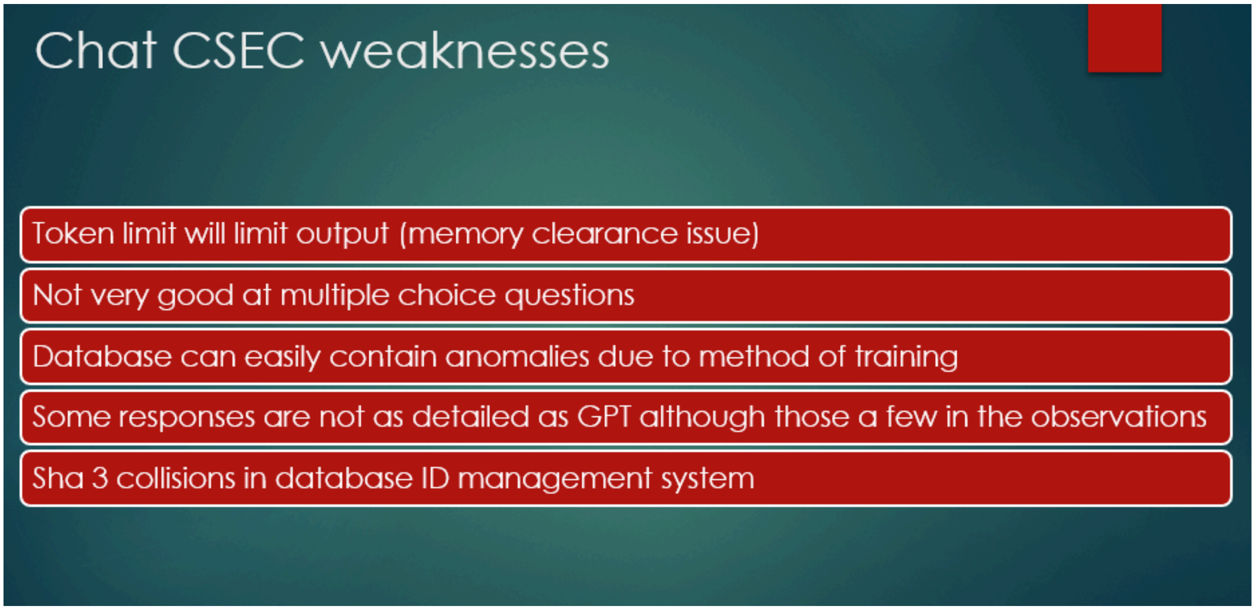
Stuxnet is a complex and sophisticated computer worm that specifically targeted supervisory control and data acquisition (SCADA) systems, particularly those used in Iran's nuclear program. It primarily affected Siemens Step7 software and the programmable logic controllers (PLCs) associated with it. The worm aimed to manipulate industrial processes and damage Iran's uranium enrichment facilities.

Stuxnet didn't directly affect traditional IT services or standard consumer systems. Instead, it was tailored to compromise specific industrial control systems. The worm targeted Siemens SCADA systems used in critical infrastructure, such as the uranium enrichment centrifuges in Iran's nuclear facilities.

The Stuxnet worm was highly specialized, and its impact was limited to the specific industrial control systems it was designed to disrupt. It did not have a broad impact on general services or consumer systems. Keep in mind that specific details about the affected systems might not be publicly disclosed or fully known due to the sensitive nature of the targeted infrastructure.



Another example of Chat CSEC outperforming GPT.



Chat CSEC is not perfect. To prevent duplicate data in the database, I used the Sha3 algorithm to assign each document an ID (each document consists of 1000 tokens. An entire pdf for example would get cut into smaller pieces of 100 tokens then added to the database to avoid prompt issues). Sha3 in my case did have a few collisions in which the same ID would be produced from a different document. The size of the prompt would also increase (with chat history) which leads to the application having to clear memory when appropriate.

Future work:

►Diverse metadata filtering for multiple topics

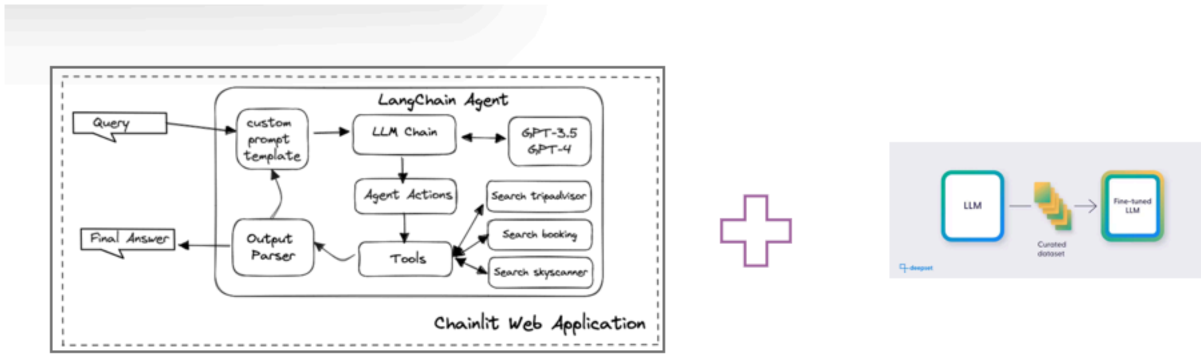
►Strict prompt engineering for hallucination mitigations and a more comprehensive mthod of training

►Fine tuning alongside embedding to improve the overall responses

►Consider agents in the use of multiple tools as a chatbot investigator and real live malware incidents

►Dedicated vector databases and prompt engineering for most information accurately retrievable.

►Potentential integration of langchain with openAI developer tools



The above is future work visualized where several databases or services could be linked to the application where lagchain agent (a service router) would decide which service or database to use based on the query. Fine tuning an LLM instance can be mixed with this method to refine a response to a user's needs (as mentioned, prompt engineering is not enough for this alone).

Github link:

https://github.com/Wissam-El-Labban/Chat-CSEC

Image sources:

- https://www.youtube.com/watch?v=Ue7M826xj5M
- https://medium.com/@abdullahw72/langchain-chatbot-for-multiple-pdfs-harnessing-gpt-and-free-huggingface-llm-alternatives-9a106c239975
- https://medium.com/@zahidali133/overcoming-the-memory-challenge-in-langchain-4feb72c830b6
- https://www.kdnuggets.com/2023/08/transforming-ai-langchain-text-data-game-changer.html
- https://cobusgreyling.medium.com/prompt-engineering-openai-modes-597425540eae