Exercises for Module 3

The exercises for the third module are about vector databases, RAG and tool-calling agents. Supporting code can be found at https://github.com/henningth/Generative-AI-in-Cybersecurity

Vector databases

These exercises are about vector databases, including how to ingest, search for, and remove documents from them. You can use the ChromaDB LangChain documentation at https://python.langchain.com/docs/integrations/vectorstores/chroma/ for inspiration.

Exercise 1: Begin by downloading the Python file 03_simple_chroma_db.py. This scripts manually inserts five sentences (and metadata) into a Chroma vector database.

(a): Run the script, and examine which documents are fetched for the two queries. Is there a difference in the order of fetched documents. Why / why not?

(b): Replace the query "deep packet inspection firewall" with a query that is unrelated to firewalls (but related to IT-security). What happens?

(c): Suppose you are only interested in queries that are from books? How can you filter the fetched results, so that only those from books are shown?

(d): Add three documents (content and metadata) about security-related topics to the vector database.

(e): Add functionality so that the user can query the vector database, until the user enters "q".

Exercise 2: Continuing from the previous exercises, suppose we want to delete a document from the vector database. How should this be accomplished? Experiment with ways to delete a document from a vector database.

Exercise 3: Suppose now we want to build a simple Retrieval Augmented Generation (RAG) system, but *not* use the LangChain Expression Language (LCEL) nor PromptTemplates. That is, we just want to use plain python. Use the script 03_simple_chroma_db.py as basis, make a simple RAG.

Hints: Begin by considering the prompt that will be sent to the LLM. This prompt should contain the context (from retrieved documents) as well as the user question. You can use e.g.

f-strings for this. Also, it is sufficient to use a generic system prompt such as (“You are a helpful assistant. Use this context to answer the user question”).

Exercise 4: In this exercise, we will look into vulnerabilities related to Retrieval Augmented Generation, in the context of OWASP LLM08: Vector and Embedding Weaknesses (see https://genai.owasp.org/llmrisk/llm082025-vector-and-embedding-weaknesses/). Begin by downloading the Python script 03_simple_chroma_db_rag_insecure.py.

(a): Run the script and observe if the query affects the LLM response. Also look at the context variable.

(b): Can metadata filtering help, and if so, how?

(c): Implement a simple metadata filtering, and test it.

(d): How can we restrict the content to only trusted sources?

RAG

Exercise 5: In this exercise, you will ingest several PDF files from a directory into a vector database. Begin by downloading the code 03_rag_ingest_many_pdfs.py.

(a): Make a directory that contains several PDF files that you want to ingest into the vector database. Then run the script. Suggestion: Start with a few PDF files, preferably some that you know the content of. Later you can test with other PDFs.

Exercise 6: Continuing from the previous exercise, begin by downloading the file 03_rag_query_many_pdfs.py.

(a): Run the code where you invoke the LLM with various questions related to the content of the PDF files. How accurate are the answers.

(b): Now try invoking the LLM with a question that has no relevance to the content of the vector store. What happens?

(c): Note that in this exercise, the execution is “one-shot”, meaning that we need to run the Python script each time we want to query the vector database. Add functionality so that the script expects user input from the keyboard and queries the LLM with the input. Note that this part doesn’t require any LangChain nor LangGraph functions, plain Python is sufficient.

Agents

Exercise 7: Starting with the code 03_agent_calculator.py, do these tasks

(a): Test with various inputs, including inputs that might fool the LLM or that don't require the usage of a calculator.

(b): What happens when you modify the description of the tool? Try modifying it.

(c): Add a tool that multiplies a list of numbers and returns the result. The multiplication tool should be a separate function from the addition tool. Then test it.

Exercise 8: In class, we saw that LLM reasoning is not always accurate. For this exercise, download the script 03_llm_reasoning.py.

Exercise 9: In this exercise, you will create and query a simple SQLite database, using LangChain agents.

(a): Download the file 03_create_students.py and run it. Observe in the code which rows are created in the database. After running the code, you should now have a file called students.db in the same directory as the Python file.

(b): Download the file 03_sql_agent_students.py. Run the code and confirm that the agent correctly answers the example question.

(c): Write and run 3 of your own questions related to the database.

(d): Observe the agent's reasoning. Try a query that might fail or confuse it. What happens?

(e): Is it possible to delete rows from the database? Why or why not?