AI-Assisted Learning for NVIDIA SDKs and Toolkits

Project Overview AND SCOPE

Develop an AI-powered Language Model that assists users on answering any questions from NVIDIA SDKs and toolkits. The platform must serve as interactive user-friendly hub, providing comprehensive information, examples, and guidance on NVIDIA's SDKs and toolkits. Thereby simplify the learning curve for developers and customers and empowering them to utilize NVIDIA's technologies more efficiently using this tool.

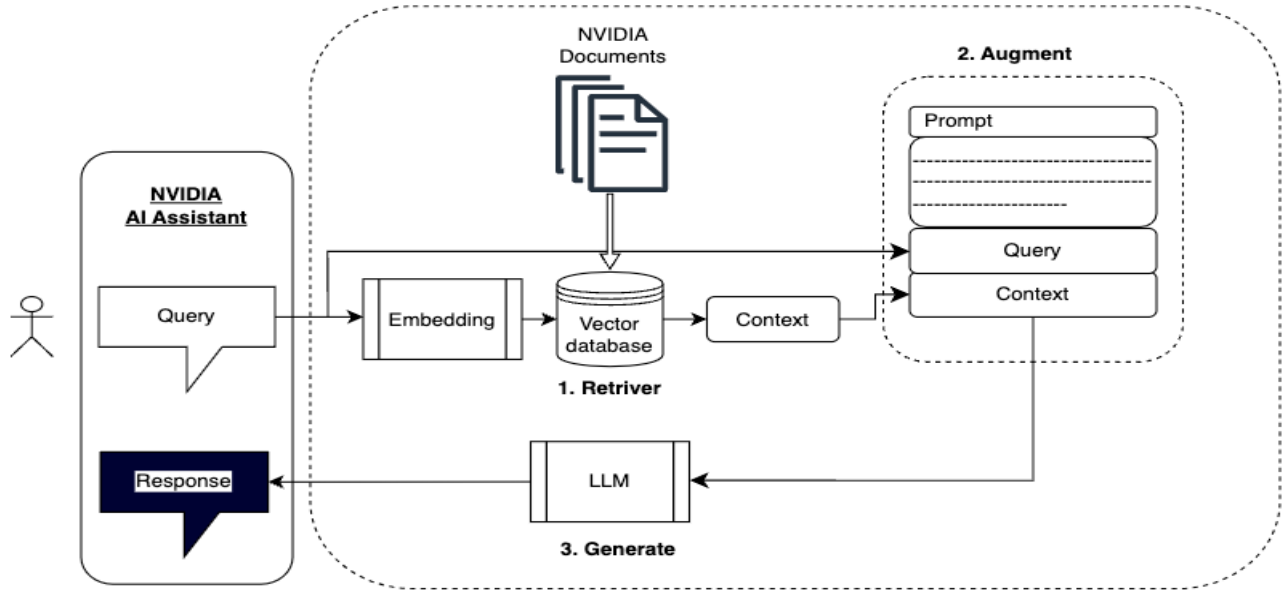
Approach

There are any many approaches to build AI-Powered assistant system, to solve this problem. Two of the most common approaches are, use any pre-trained LLMs out of the box and finetune. This is the process of taking a pre-trained LLM and further training it on a smaller, specific dataset to adapt it for a particular task or to improve its performance. By finetuning, we are adjusting the model's weights based on our data, making it more tailored to our application's unique need. This approach takes large **time** and effort on **preparing training data** and **finetuning**, **training** and, **a robust system** to deal with the huge volume of datasets and training.

The other approach is using Retrieval-Augmented Generation (RAG) architecture, and it appears to be simpler and more suitable in the context of this project and hence I will be using RAG here to build AI-Powered assistant system for NVIDIA here.

RAG systems are, by definition, designed to augment an LLM's capabilities by retrieving relevant information from knowledge sources before generating a response. This makes this technique well-suited for applications that need to query databases, documents, or other structured/unstructured data repositories. The retriever and generator components can be optimized to leverage these external sources.

Overview on Retrieval-Augmented Generation (RAG) and Workflow



In 2020, Lewis et al. proposed a technique called Retrieval-Augmented Generation (RAG) in the paper Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks . A three-step process to generate the user response:

1. Retrieve: The user query is used to retrieve relevant context from an external knowledge source. For this, the user query is embedded with an embedding model into the same vector space as the additional context in the vector database. This allows to perform a similarity search, and the top k closest data objects from the vector database are returned.
2. Augment: The user query and the retrieved additional context are stuffed into a prompt template to give text generation instruction to the LLM.
3. Generate: Finally, the retrieval-augmented prompt is fed to the generative LLM, which generates the response to the query from the context provided.

DETAIL DESIGN

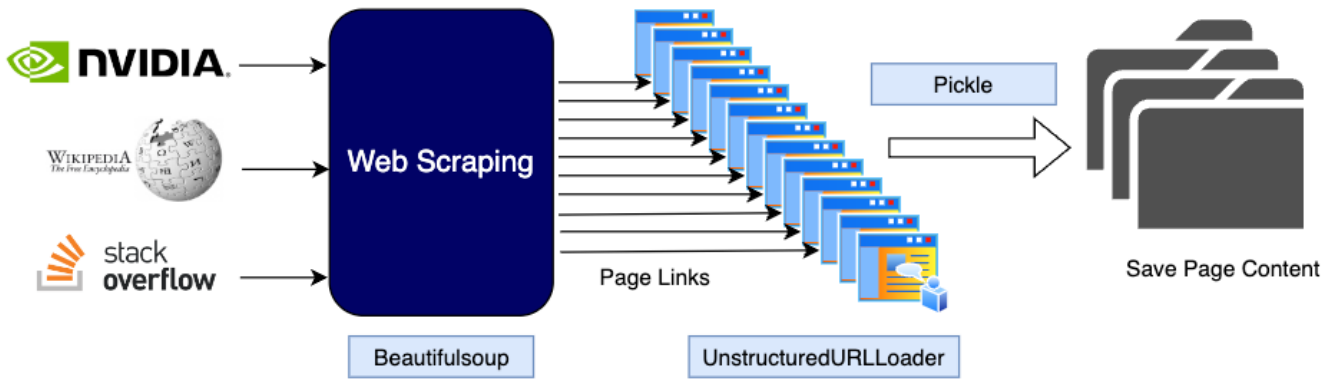
End-to-End design approach and selection of different tools and models are described below:

Data Collection and Preprocessing

Notebook: 1. NvidiaLLM_helper_webscraping.ipynb

- Data collections are done from following sites:

Site#	URL
1	https://www.nvidia.com/en-us/
2	https://docs.nvidia.com/
3	https://docs.nvidia.com/#all-documents
4	https://forums.developer.nvidia.com/
5	https://nvidia.custhelp.com/app/answers/list/st/5/kw/grid
6	https://en.wikipedia.org/wiki/Nvidia
7	https://stackoverflow.com/ Questions Tagged with ['nvidia', 'nvidia-jetson', 'nvidia-jetson-nano', 'nvidia-docker', 'nvidia-digits', 'nvidia-deepstream', 'nvidia-smi', 'nvidia-titan', 'nvidia-flex', 'nvidia-isaac', 'nvidia-hpc-compilers', 'nvidia-jetpack-sdk', 'nvidia-sass']

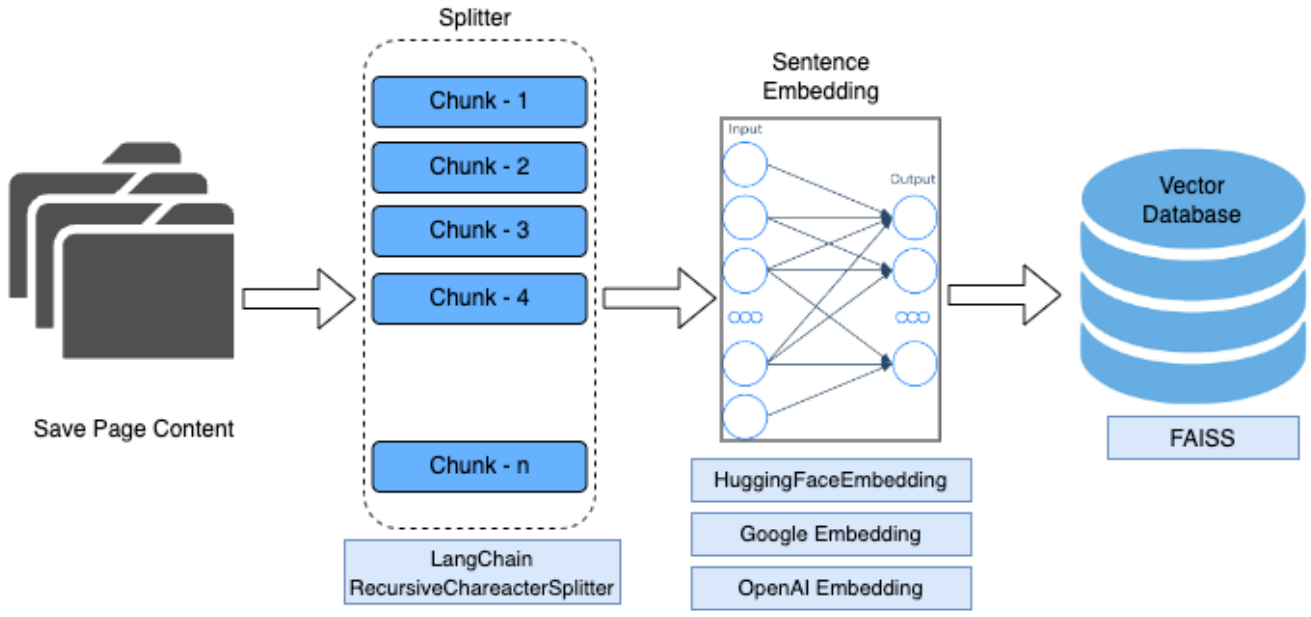


- **Web scraping**: For web scraping “Beautifulsoup” library from bs4 is used. All above listed sites are scraped and the recurrent links are obtained using various functions from Beautifulsoup and python functions.
- **Data Loading**: UnstructuredURLLoader from langchain_community.document_loaders is used to load the contents of the page from those page links. UnstructuredURLLoader loads the text content from html pages and formats them. Once text content is loaded it is then cleaned using cleaning functions (clean,remove_punctuation,clean_extra_whitespac) from python “unstructured” library. Contents are saved as langchain documents using pickle in folder NVIDIA_DATA_FOLDER. Below lists the files:

Site#	# Pages	Page Links file	Page Content File
1	345	NVIDIA_MAIN_PAGE_LINKS	NVIDIA_MAIN_PAGE_CONTENT
2	1	NA	NVIDIA_MAIN_DOC_PAGE_CONTENT
3	32835	NVIDIA_DOCUMENT_PAGE_LINKS	NVIDIA_DOCUMENT_PAGE_CONTENT
4	3564	NVIDIA_FORUM_PAGE_LINKS	NVIDIA_FORUM_PAGE_CONTENT
5	46	NVIDIA_CUSTHELP_PAGE_LINKS	NVIDIA_CUSTHELP_PAGE_CONTENT
6	1	NA	NVIDIA_WIKIPEDIA_PAGE_CONTENT
7	4671	NVIDIA_STACK_OVERFLOW_QA_PAGE_LINKs	NVIDIA_STACK_OVERFLOW_QA_PAGE_CONTENT

Embedding and Create Vector Database

Notebook: 2. NvidiaLLM_helper_embedding_vecotorDB_loading.ipynb



- **Splitter**: Saved contents are load back using pickle. Each page contents are then split into multiple chunks, using RecursiveCharacterSplitter from LangChain.
    - *Finetuning chunk_size*: Started with chunk size of 512, that yields more granular chunks. It has been observed that while retrieving the context from vector database

most of the time it pulls more chunks with high similarity context. So, in here I am using 1024 as chunk size and chunk overlap as 0. It is taking few extra times but giving more contextual and better response.

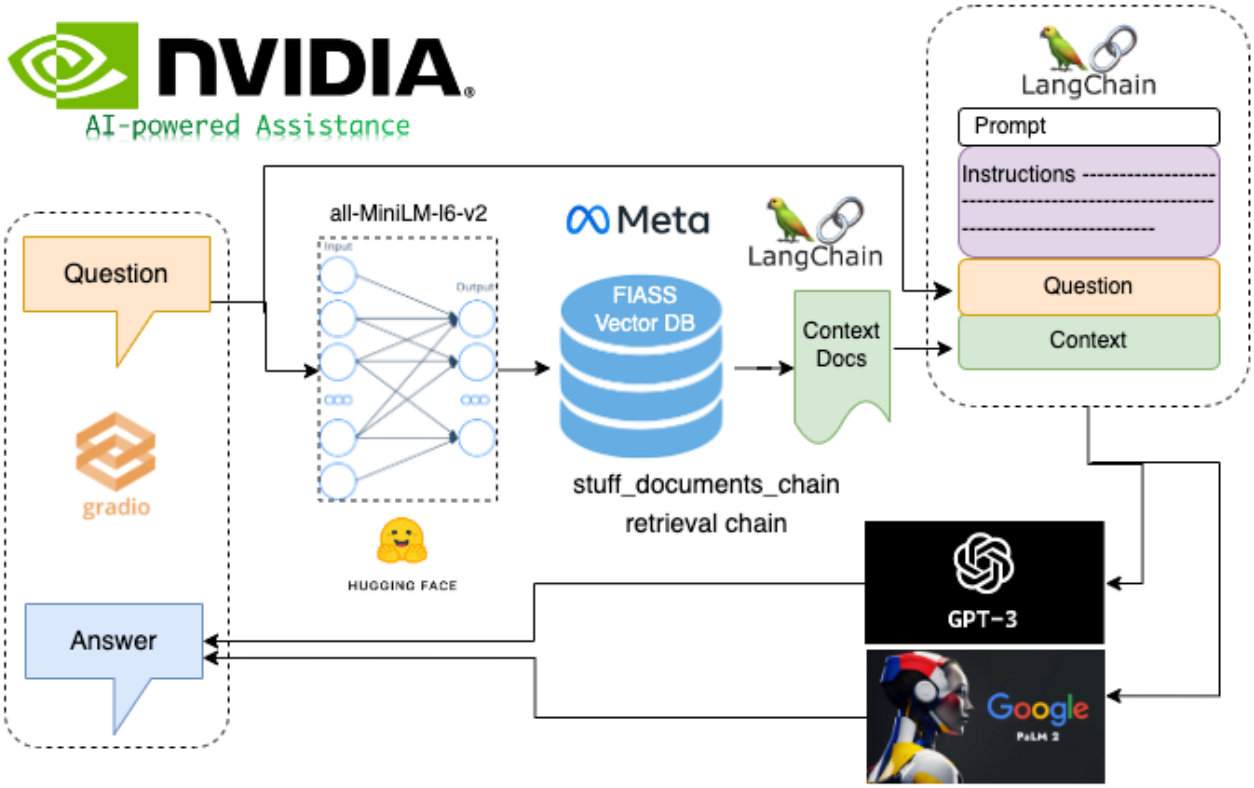
- **Embedding**: I have tried multiple embeddings models, the best one I found for this solution is sentence-transformers/all-MiniLM-l6-v2 from HuggingFaceEmbedding.
    - *Identifying Embedding model*:
        i. HuggingFace (sentence-transformers/all-MiniLM-l6-v2) was very fast on Apple silicon 'mps' GPU. Retriever also performed well to pull right context from vector database based on the query supplied. However, it maps sentences & paragraphs to a 384-dimensional dense vector space which is smaller than sentence-transformers/all-mpnet-base-v2 but working well. I am using this embedding for this project.
        ii. HuggingFace (sentence-transformers/all-mpnet-base-v2) is slow compared to above one from HuggingFace dispite of running on Apple silicon 'mps' GPU. Because it maps sentences & paragraphs to a 768-dimensional dense vector space on Apple silicon 'mps' GPU. Because it maps sentences & paragraphs to a 768-dimensional dense vector space. This one is not fetching right context document from the vector DB hence I am not suing this here.
        iii. Google Embedding (models/embedding-001) slower also not working well in the context of the project.
        iv. OpenAI Embedding (text-embedding-3-small) It is one of the best embedding but Open AI paid API as other HuggingFace Embedding is working well, I do not use it here however I tried for a portion of the data it is working well.

- **Vector Database**: FAISS from Meta works well for the context of the project, so I am using it here without much investigating other vector stores DB.

RAG Question Answer AI Assistance

Notebook: 3. NvidiaLLM_main.ipynb

- Input to this notebook is Vector Database which was created in the previous section. For question and answering I am using Gradio tool for easy user interface to communicate with the system and get the answer for the asked question.
- The other input to notebook is a csv file including sample questions which is used to test the model and its answer generation.



- **Retriever**: The asked question gets embedded using Huggigface sentence-transformers/all-MiniLM-l6-v2 and 4 similar context documents are retrieved from FIASS vector database.
- **Prompt**: comprise of instruction on how to answer the question along with question and context documents.
- **LLM**: I am using Google Palm2 (text-bison-001) model to get the answer which is then displayed in the gradio interface. Validation on other models:
    - OpenAI (gpt-3.5-turbo-instruct) really good on answering most of the questions and following the prompt strictly but needs paid service to call the API
    - Google (gemini-pro and gemini-1.0-pro) models are not so good in answering complicated questions but do well on simple questions.

Observations

- There are limitations of using Google Palm2 (text-bison-001) model it is observed that the model sometime does not follow the prompt instructions and try to answer from external sources, this is expected as this implementation is for vanilla/demo without any paid subscription.
- Web scrapping can be improved to get better and clean content from the websites. There are many documents which could have been handled differently to extract the content of those more meaningful way.

- Better Embedding model with appropriate chunk size and overlap would improve the quality of answers and response time.
- Also partial training of off the shelf pre-trained models could improve quality.
- Limitations on personal system to handle huge volume of data and processing speed.