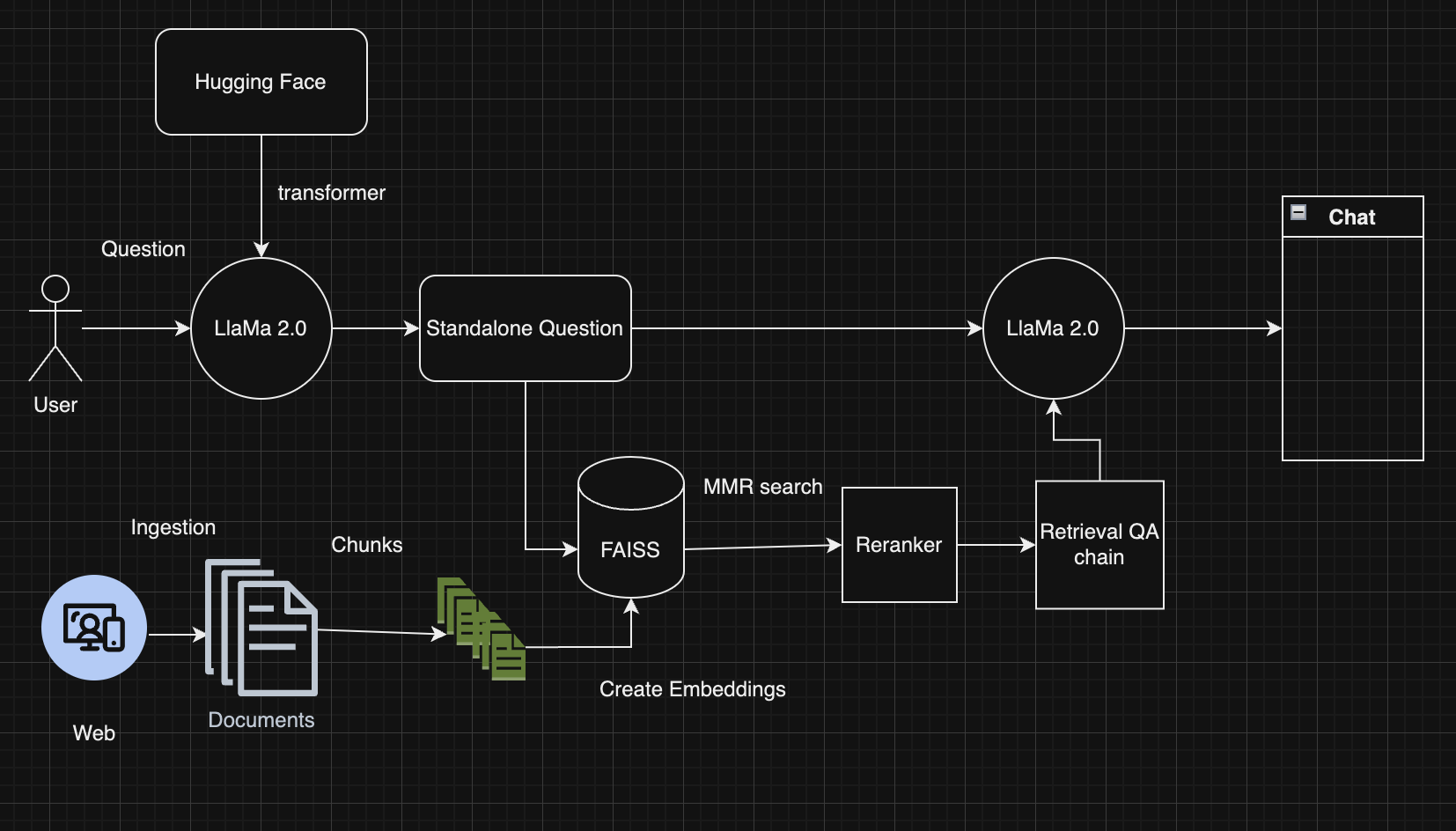
**RAG Implementation for "NVIDIA Toolkits and SDKs" using an LLM**

****

**1. Data Collection**

* Crawled the NVIDIA website for relevant documentation, forums, and blog posts related to NVIDIA toolkits and SDKs using a scraper script in python. This is attached in the zip.
* This script loads html pages from the web and converts it to a usable document object using the WebBaseLoader
* Introduced a separate loader for pdf documents called OnlinePDFLoader to perform the same type of parsing as above.
* We then load the page to scrape out URLs for additional document scrapes.
* We do this to a depth of 4.
* Collected a dataset of 9905 documents in total.
* For a list of links crawled and code refer to the attached nvidia\_scrapper project uploaded in the zip.

**2. Data Preprocessing**

* Tokenized the documents into sequences of words.
* Defined a stopping criteria and passed it to the LLM to prevent rambling.
* Used a RecursiveCharacterTextSplitter with a chunk size of 2000 tokens and an overlap of 20 tokens (Tried different sizes but this was the best to generate meaningful context) to generate document chunks.
* Used a sentence transformer embedding model "sentence-transformers/all-MiniLM-L6-v2" for creation of vector embedding of document chunks. This is the same embedding used to vectorize queries to our application.
* Used FAISS vectorstore to save the vector embeddings of the document chunks.

**3. Model Architecture Design**

* Used a RAG (Retrieval-Augmented Generation) model architecture.
* The RAG model consists of a **retriever** and a **generator**.
  + Retriever
    - The retriever is responsible for retrieving relevant documents from the document collection.
    - Here we used FAISS as the vectorstore for the document chunks and we used MMR ( Maximal Marginal Relevance ) with a maximum of 3 documents retrieved.
    - The idea behind using MMR is that it tries to reduce redundancy and increase diversity in the result. This also gave better results than a cosine similarity with/without a threshold.
    - We also used a re-ranker after getting results from the FAISS to score relevancy of the results got back
    - We use FlashrankRerank which uses a default model ms-marco-TinyBERT-L-2-v2 as a cross encoder to score queries to document chunk retrieved pairing.
    - We used RetrievalQA retriever chain from langchain with a custom prompt.
    - We used a custom prompt fed to the Llama model to improve its accuracy and quality of responses.
    - This is then fed to the generator through the Hugging Face pipeline chain we created.
  + Generator
    - The generator is responsible for generating natural language responses based on the retrieved documents.
    - We use a Llama 2 7B model to generate responses to the queries based on the retrieval from the re-ranker.
    - We used 4 bit quantization on the model to reduce its memory footprint.

**4. Model Training**

* Used a pre-trained Llama 2 7B model ('meta-llama/Llama-2-7b-chat-hf') from hugging face as the generator.

**5. Inference Time**

* The RAG model can generate a response to a query in less than a minute. Worst case has been 50 secs. On an average it responds in 25-30 seconds