RAG-Based Electronic Design Assistant for Renesas Electronics

Monica Meduri
Github Link
Live Server Link

Abstract

The exponential growth of online data has posed significant challenges in retrieving accurate and relevant information. Traditional search engines often generate broad results that require extensive manual filtering, while most existing Retrieval-Augmented Generation (RAG) systems rely heavily on static knowledge bases, limiting their ability to retrieve real-time information. The proposed system employs a hybrid approach that combines retrieval-based and generative models along with with AI Agent, to ensure that the most recent and accurate information is provided. Unlike traditional solutions that depend solely on pre-indexed data, our system can dynamically fetch up-to-date information from online sources when necessary.

1. Introduction

With the rapid increase in web-based data, obtaining accurate and contextually relevant information has become essential across various domains s. Traditional search engines like Google and Bing, although effective in indexing large volumes of information, often require users to manually browse through multiple sources to extract useful insights. This process is not only time-consuming but also places a cognitive burden on users, who must evaluate the relevance of the obtained information.

The advent of Large Language Models (LLMs), such as OpenAI's ChatGPT, has introduced a paradigm shift in information retrieval by offering coherent, human-like responses to user queries. However, a major limitation of LLMs is that their knowledge is confined to pre-trained datasets, making them ineffective for delivering up-to-date information.

To address this challenge, Retrieval-Augmented Generation (RAG) systems have emerged as a promising solution. RAG combines the capabilities of language models with external knowledge bases, enabling the model to fetch relevant data from a pre-indexed vector database. However, conventional RAG systems have their limitations. They lack dynamic search capabilities, resulting in knowledge gaps when new or updated information which is not present in the database.

This project introduces an advanced LLM RAG Fusion system designed to overcome these limitations by incorporating dynamic web scraping, vector database retrieval, and real-time external searches through an AI agent. The core functionality of the proposed system includes:

- **Knowledge-Based Question Answering:**
  The bot is designed to answer user queries based on the information available in the Renesas Knowledge Base.
- **Image Integration in Responses:**
  If the information in the Knowledge Base contains relevant images, the bot is capable of including those images in its response.

- **Page Reference Linking:**
  The bot provides direct references to the pages from which the information was retrieved, allowing users to explore further.

- **Image-Based Query Resolution:**
  The bot supports user-uploaded hand-drawn images, or other relevant visuals. It can analyze the image content and provide contextually accurate answers based on the visual input.

- **AI Agent:**
  In cases where the Knowledge Base does not contain satisfactory information or the confidence score is low, the bot can automatically trigger an external search agent that scans across renesas.com for potential answers and provide them to the user.

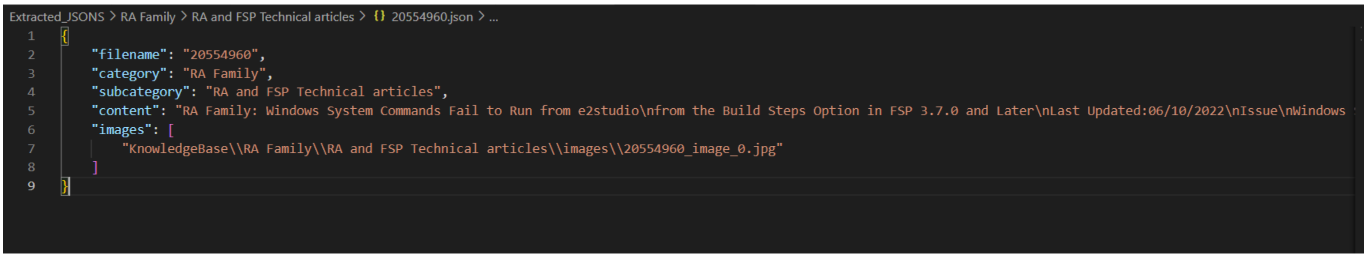
- **AWS Deployment:**
  The bot is deployed on a AWS cloud server using a Streamlit interface.

2. Approach

This section explains the data collection, system design and implementation of the project.

The workflow begins with data extraction from Renasas website (https://en-support.renesas.com/knowledgeBase). The website has several categories like RA Family, RZ Family etc. Each category has several subcategories where many articles are present. The content from each articles is extracted using Selenium and Beautifulsoup and saved in pdf format in folder hierarchy like article - subcategory - category>. categories like. The code related to this is in **extractpdf.py**

The content from pdfs is extracted using PyMuPDF and stored in JSON format. The next step involves breaking the JSON data into smaller, meaningful chunks to preserve semantic integrity. Image 1 shows a sample JSON structure.


Extracted_JSONS > RA Family > RA and FSP Technical articles > {} 20554960.json > ...

```
{
    "filename": "20554960",
    "category": "RA Family",
    "subcategory": "RA and FSP Technical articles",
    "content": "RA Family: Windows System Commands Fail to Run from e2studio\nfrom the Build Steps Option in FSP 3.7.0 and Later\nLast Updated:06/10/2022\nIssue\nWindows
    "images": [
        "KnowledgeBase\\RA Family\\RA and FSP Technical articles\\images\\20554960_image_0.jpg"
    ]
}
```

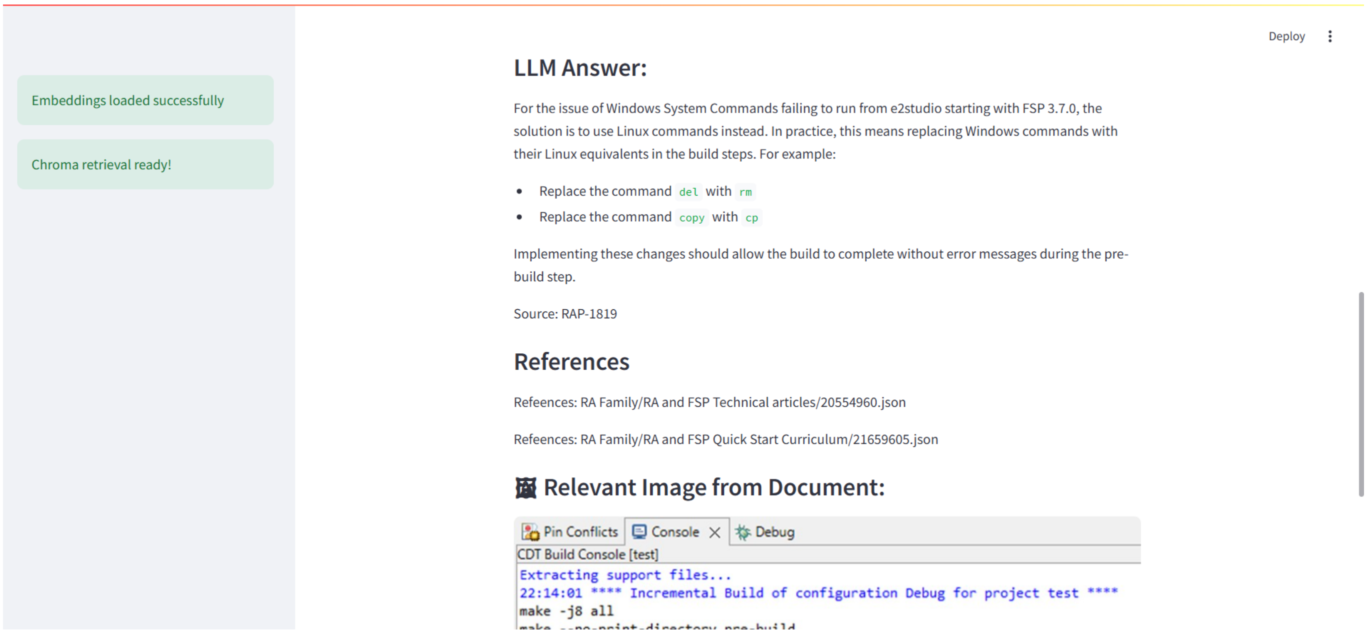
Each text entry, is split into manageable pieces. This chunking process ensures that significant information is preserved while maintaining coherence. After chunking, the text is converted into high-dimensional embeddings using hugging face all-MiniLM-L6-v2 embedding model. These embeddings enable efficient similarity-based retrieval by transforming the text into vector representations.The code related to this can be found in. The generated embeddings, along with relevant metadata such as URLs, images, and source references, are stored in ChromaDB. The code related to this can be found in **extractjson.py** and **embeddings.py**

When a user submits a query, the retriever fetches the top relevant documents based on semantic similarity. A custom prompt is created combining the user's text query, uploaded image (if any), and retrieved documents, which is then processed by GPT-4o using a RetrievalQA chain to generate an accurate response. The code related to this can be found in **RAG.py**

The project is deployment on AWS EC2 with Streamlit interface. The live link of the AWS server is here

Results

Below figure shows the reteived LLM answer along with the references and relevant image



Embeddings loaded successfully

Chroma retrieval ready!

LLM Answer:

For the issue of Windows System Commands failing to run from e2studio starting with FSP 3.7.0, the solution is to use Linux commands instead. In practice, this means replacing Windows commands with their Linux equivalents in the build steps. For example:

- Replace the command `del` with `rm`
- Replace the command `copy` with `cp`

Implementing these changes should allow the build to complete without error messages during the pre-build step.

Source: RAP-1819

References

Refeences: RA Family/RA and FSP Technical articles/20554960.json

Refeences: RA Family/RA and FSP Quick Start Curriculum/21659605.json

🖼️ Relevant Image from Document:

CDT Build Console [test]

```
Extracting support files...
22:14:01 **** Incremental Build of configuration Debug for project test ****
make -j8 all
make --no-print-directory pre-build
```

Limitations

The proposed system faces several limitations, including latency challenges due to real-time web scraping, which can slow down response times. Scalability may be affected under high query loads, requiring efficient resource management. Additionally, web scraping restrictions, such as CAPTCHA and bot detection, can limit the system's ability to retrieve up-to-date information.

Future scope

Future improvements include integrating real-time APIs for faster, more reliable data retrieval and reducing reliance on web scraping. Enhancements in query optimization through reinforcement learning and personalized responses based on user interactions will improve recommendation accuracy. These upgrades aim to make the system more adaptable and effective in real-world applications.