Universidad Distrital Francisco José De Caldas

Chatbot sale of trotec laser machines.

Ingeniería en Sistemas

Análisis De Sistemas

Integrante:

JHONATHAN DAVID DE LA TORRE GARCIA 20222020033

Bogotá Colombia

junio 2024

INTRODUCCION.

This document consists of collecting relevant information on the development of a specialized chatbox for sales assistance of Trotec Laser brand machines. The aim is to provide the client with a tool for their own information that allows them to decide on what type of machine. meets your business expectations.

DEVELOPMENT

The development of the specialized Chatbox will begin with a systematic analysis of the basic requirements of customers looking to buy Trotec laser cutting machines. Once the essential requirements have been determined, the technical documents with the specifications of the products will be personally requested. the machine models chosen according to the requirements analysis

During the requirements analysis it was found that the vast majority of customers ask specifically for a very small group of models of laser cutters and engravers. Particularly the R series with its R400 and R500 models, SP Series with its SP500, SP2000 and SP3000 models. ,and the SpeedMarker industrial series with its SpeedMarker100, 300, 700 and 1300 models.

After the analysis, the corresponding pdf of said machine models is requested from the Trotec Laser technical service, thus achieving great consistency in the data since they would be endorsed by the company that manufactures the machines.
Once the PDFs are in possession, the necessary information is filtered to be presented to the various types of Trotec Laser clients. During the filtering of information, it is grouped into paragraphs, highlighting in said information components that are highly asked about by clients, such as:
What is the power of the laser machine?
What is the work area?
What is the power of the laser?
What is the type of laser?
What is the price of the machine?, etc.

Based on this, the size of the original PDFs is reduced to establish the system limits. Once with the information organized and polished, we proceed to propose the development of the chatbox, it is decided to use, on the recommendation of the teacher, Lang chain, and llama-cpp, all with the purpose of using the contexts of the sentences using an established language model. , to find a grammatically correct way to answer a question based on the information provided in the PDFs.

The complete code is shown below:

```
"""
This module has some functionalities regarding text processing.

Author: Carlos Andrés Sierra <cavirguezs@udistrital.edu.co>
"""
```

```python
import sys
from langchain.chains.retrieval_qa.base import RetrievalQA
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import PyPDFDirectoryLoader
from langchain_community.vectorstores import FAISS
from langchain_community.llms import LlamaCpp
from langchain_huggingface import HuggingFaceEmbeddings


def load_pdf_data(path: str) -> str:
    """
    This method takes all PDF files in a directory and loads them into a text string.

    Args:
        path (str): The path to the directory containing the PDF files.

    Returns:
        A text string containing the content of the PDF files.
    """
    loader = PyPDFDirectoryLoader(path)
    return loader.load()


def split_chunks(data: str) -> list:
    """
    This method splits a text string into chunks of 10000 characters
    with an overlap of 20 characters.

    Args:
        data (str): The text string to split into chunks.

    Returns:
        A list of strings containing the chunks.
    """
    splitter = RecursiveCharacterTextSplitter(chunk_size=10000, chunk_overlap=20)
    chunks = splitter.split_documents(data)
    print(f"Number of chunks: {len(chunks)}")
    return chunks


def get_embeddings() -> list:
    """
    This method gets semantic embeddings for a list of text chunks.

    Returns:
        A list of embeddings for the text chunks.
    """
    return HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
```

```
def get_chunk_embeddings(chunks: list, embeddings: list) -> list:
    """
    This method gets the embeddings for a list of text chunks.

    Args:
        chunks (list): A list of text chunks.
        embeddings (list): A list of embeddings for the text chunks.

    Returns:
        A list of embeddings for the text chunks.
    """
    return FAISS.from_documents(chunks, embedding=embeddings)


def load_llm():
    """
    This method loads the LLM model, in this case, one of the FOSS Mistral family.
    """
    # Download from: https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-
GGUF/blob/main/mistral-7b-instruct-v0.2.Q3_K_M.gguf
    llm = LlamaCpp(
        streaming=True,
        model_path="llm_model/mistral-7b-instruct-v0.2.Q2_K.gguf",
        temperature=1.0,
        top_p=0.9,
        verbose=True,
        n_ctx=4096,
    )
    return llm


def agent_answer(question: str, llm: object, vector_store: object):
    """
    This method gets the answer to a question from the LLM model.

    Args:
        question (str): The question to ask the LLM model.
        llm (object): The LLM model.

    Returns:
        A string with the answer to the question.
    """
    qa = RetrievalQA.from_chain_type(
        llm=llm,
        chain_type="stuff",
        retriever=vector_store.as_retriever(search_kwargs={"k": 2}),
    )
```

```
    return qa.run(question)


def main():
    """This is the infinite loop to make questions to the Tennis Assistant."""

    print("Bienvenido a Trotec Laser!\n")
    llm = load_llm()
    chunks = split_chunks(load_pdf_data("source_documents"))
    embeddings = get_embeddings()
    vector_store = get_chunk_embeddings(chunks, embeddings)

    while True:
        user_input = input("Make your question: ")
        if user_input == "exit":
            print("Thanks. Goodbye!")
            sys.exit()
        if user_input == "":
            continue

        result = agent_answer(question=user_input, llm=llm, vector_store=vector_store)


        print(f"Answer: {result}")


if __name__ == "__main__":
    main()
```

The code explanation is shown below:

The system analysis chatbox project focused on improving the sales experience of laser cutting and engraving machines of the Trotec Laser brand is stored in this repository. The project only includes a series of PDF documents from the models most requested by clients (determined through systemic analysis)

Step-by-Step Explanation of the Chatbot

1. **Introduction to the Chatbot:**
    - The chatbot is designed to assist customers interested in Trotec laser cutting machines by providing technical and functional information about the machines, including their prices.
2. **Loading PDF Data:**
    - The first step is to load PDF files that contain information about the laser cutting machines. This is done using the load_pdf_data function, which

reads all PDFs from a specified directory and combines their content into a single text string.

3. **Splitting Text into Chunks:**
    - The large text string from the PDFs is then split into smaller, manageable chunks. The split_chunks function takes the text string and divides it into chunks of 10,000 characters each, with an overlap of 20 characters to ensure continuity.
4. **Getting Embeddings:**
    - Next, the get_embeddings function generates semantic embeddings for each chunk of text. These embeddings are numerical representations that capture the meaning of the text, making it easier for the chatbot to understand and retrieve relevant information.
5. **Creating a Vector Store:**
    - The get_chunk_embeddings function then creates a vector store using FAISS (a library for efficient similarity search). This store contains the embeddings of the text chunks and allows the chatbot to quickly find relevant chunks when a question is asked.
6. **Loading the LLM Model:**
    - The load_llm function loads a pre-trained LLM (Large Language Model) from the Mistral family, configured to provide responses to user queries. The model is optimized for efficiency, balancing response speed and accuracy.
7. **Interacting with the Chatbot:**
    - The main function starts an infinite loop where the user can ask questions. When a question is entered, the agent_answer function uses the LLM and the vector store to find and generate a relevant answer. The chatbot then displays the answer to the user.
8. **User Communication:**
    - If the user wants to exit the chat, they can type "exit", and the program will terminate. If there are no inputs, the loop continues to wait for the user's next question.

Several tests were carried out on low-requirement equipment, the first carried out on a computer from 2010, with a 1-core Celeron processor at 1.7GHZ, and 4 GB of RAM, the result was disastrous, it took an average of 6 hours to 8 hours to execute the response under a short context.

A second execution was carried out on a PC with higher specifications, a laptop with 2.1GHZ A10 processing with 8 GB of ram, the average execution time was 30 mins, even reducing the context and reducing the number of operating system threads. , it did not decrease from there, it was initially considered unviable due to the high wait for a response

Finally, the model is executed using a CPU with a Ryzen 5 5600 cpu, and with 16GB of RAM, using a more simplified model and reducing the number of threads, giving responses with much lower time, with this the high load is noted. memory and CPU required by the real-time model.

When executing a prompt with a longer context, requesting a list, it provided the information correctly, although it was cut off as shown below:

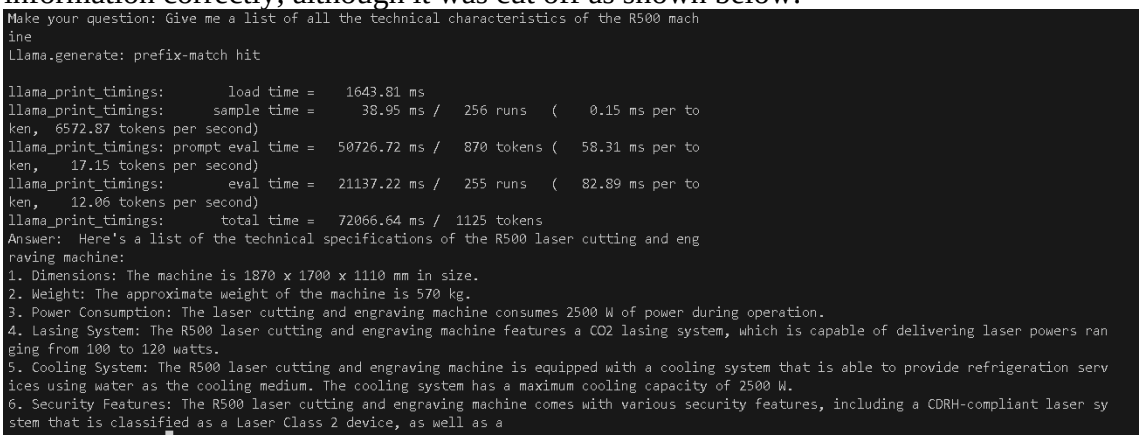

Image 1. Give me a list of all the technical characteristics of the R500 mach

As you can see, the time for this query was 72.06664 seconds, a time that for the context of the project is considered a considerably acceptable time.

However, when submitting a query about a laser cutting machine that is not found in the PDF's, you get this response:

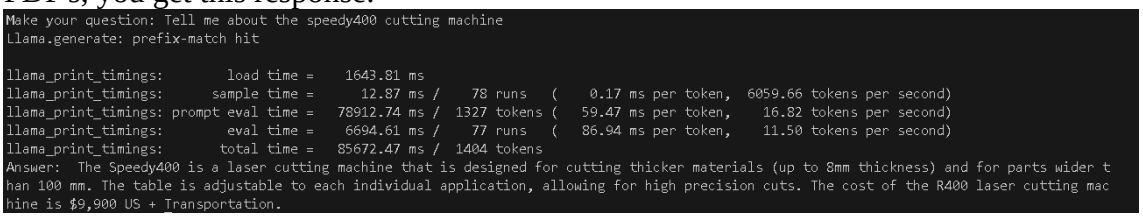

Image 2. Tell me about the speedy400 cutting machine.

However, as mentioned, this machine is not found in the database, since the Speedy400 machine corresponds to another series of machines, however the chatbot gave the description of the R400 machine (which is similar in number but not in series), so the limits would have to be established so that the language does not provide cross-information about machines from other references not contemplated.

With the above mentioned, the systemic analysis is proposed focusing on the following items:

1. **System Elements**:
    - **Users**: Customers seeking information about the machines.
    - **Chatbot**: The interactive interface that provides responses.
    - **Database**: PDF files with technical and commercial information.
    - **LLM Model (Llama)**: Natural language processing algorithm that generates answers.
    - **Server**: Infrastructure hosting the chatbot and database.
    - **Trotec Company**: Supplier of the machines and post-sale support.
2. **Relationships**:

    - **User-Chatbot Interaction**: The user asks questions, and the chatbot responds.
    - **Database Query**: The chatbot accesses PDF files to retrieve information.
    - **Q&A Process**: The LLM model interprets questions and generates responses.
    - **Feedback to the Company**: The user can contact the company to make a purchase or report errors.
3. **Homeostasis**:
    - **Database Updates**: Keeping the information up-to-date to ensure accurate responses.
    - **Chatbot Optimization**: Periodic adjustments to the model to improve response speed and accuracy.

Chaos Theory

1. **Sensitivity to Initial Conditions**:
    - **Variability in Questions**: Small differences in user questions can lead to different responses.
    - **Model Calibration**: Small adjustments in the model parameters can significantly impact response quality.
2. **Nonlinear Behavior**:
    - **Complex Interaction**: The interaction between the user and chatbot can be unpredictable and vary significantly with small input variations.
    - **Continuous Optimization**: The need to monitor and adjust the system regularly to maintain optimal performance.

Entropy

1. **Disorder and Informational Efficiency**:
    - **Unstructured Data**: PDF files contain unstructured information that must be processed efficiently by the chatbot.
    - **Informational Noise**: Irrelevant or redundant information in responses can increase the system's entropy, making it harder for the user to understand.
2. **Reducing Entropy**:
    - **Information Filtering**: Implement techniques for filtering and summarizing information to provide clear and concise responses.
    - **Query Optimization**: Adjust natural language processing to minimize incorrect or imprecise responses.

Finally, it is worth highlighting that the project meets the minimum expected expectations to be considered viable for an application in the industry, however it is necessary to establish certain parameters that facilitate the interaction between Trotec and the user in the future.