GOVERNMENT COLLEGE OF TECHNOLOGY

COIMBATORE-13

Excel Assistant Chatbot

NAME	: Yogeshwari L - 71772117L05
	: Kayalvizhi T - 71772117305
DEPARTMENT	: B.E.(CSE)–4TH year
COURSE CODE	: 18SPC702
COURSE NAME	: ARITIFICIAL INTELLIGENCE
ASSIGNMENT	: 02
DATE	: 11.11.2024

Excel Assistant Chatbot

Abstract:

The Excel Assistant Chatbot is an interactive tool designed to simplify the process of uploading, managing, and analyzing Excel files. Users can easily upload Excel documents, and the bot provides seamless data handling with intuitive guidance for various tasks, such as data validation, summarization, and extraction of key insights. With built-in functionalities to handle complex Excel data structures, the chatbot allows users to receive instant responses, analysis, and feedback without manual processing, making data management faster and more accessible. This bot is ideal for users seeking a streamlined, user-friendly solution for quick data interaction and retrieval from Excel files.

Dataset:

In this code, the dataset used is a **CSV file** uploaded by the user, specifically related to cricket data. The application is designed to process the uploaded CSV file, load the data, and create a question-answering chatbot that can interact with and respond to questions based on the content of the cricket data provided in the CSV file.

1. **User-Provided CSV File:**
   The application requires the user to upload a CSV file containing cricket-related data. This could be any data within the cricket domain, such as player statistics, match scores, team rankings, or historical game results.

2. Data Structure and Loading:

- The `CSVLoader` is used to load the CSV file content, making it accessible for further processing. It processes the data by reading rows and columns, parsing each line in the CSV.
- The delimiter for the CSV file is specified as `,` (comma-separated values), which is typical for CSV files.

3. Data Embeddings and Storage:

- The data from the CSV is embedded using `HuggingFaceEmbeddings`, which transforms the text data into a vectorized format, making it ready for retrieval operations.
- The `FAISS` library is then employed to store the embedded data in a local vector database. This structure allows quick and relevant retrieval of information based on user queries.

4. Application Context:

- The content focuses on cricket, so the bot is specifically programmed to respond to questions within the cricket domain. It uses the cricket data provided by the user to answer relevant questions accurately.

5. Prompt Template for Cricket Domain:

- The prompt template instructs the bot to answer only based on the cricket data provided, discouraging irrelevant responses and ensuring the answers stay within the scope of the uploaded dataset.

The result is an interactive chatbot tailored for cricket data, which can respond to user questions by pulling specific information from the uploaded dataset.

Coding :

```
import streamlit as st
from streamlit_chat import message
import tempfile
from langchain.document_loaders.csv_loader import CSVLoader
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.llms import CTransformers
from langchain.chains import RetrievalQA
from langchain import PromptTemplate
from huggingface_hub import hf_hub_download

def load_llm():
    llm = CTransformers(
        model = "res/models--TheBloke--Llama-2-7b-Chat-
GGML/snapshots/76cd63c351ae389e1d4b91cab2cf470aab11864b/llama-2-7b-
chat.ggmlv3.q4_1.bin",
        model_type="llama",
        max_new_tokens = 512,
        temperature = 0.9
    )
    return llm
st.title("Cricbot - Chat with Cricket CSV Data 🏏")
csv_data = st.sidebar.file_uploader("Upload your Data", type="csv")
if csv_data :
    with tempfile.NamedTemporaryFile(delete=False) as tmp_file:
        tmp_file.write(csv_data.getvalue())
        tmp_file_path = tmp_file.name
    loader = CSVLoader(file_path=tmp_file_path, encoding="utf-8", csv_args={
                'delimiter': ','})
    data = loader.load()
    st.json(data)
```

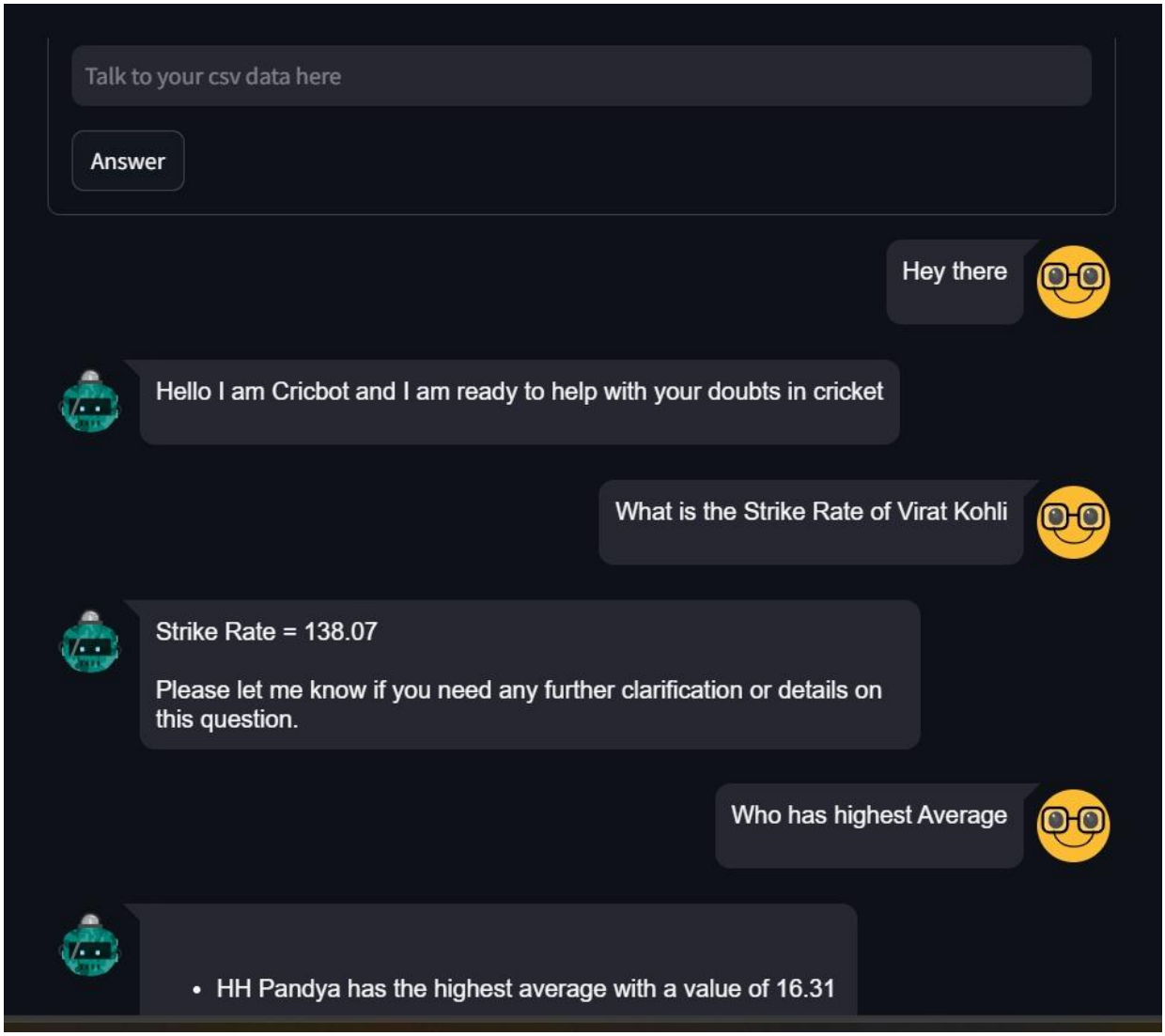
```
    embeddings = HuggingFaceEmbeddings(model_name='thenlper/gte-large',
                                       model_kwargs={'device': 'cpu'})
    db = FAISS.from_documents(data, embeddings)
    db.save_local('faiss/cricket')
    llm = load_llm()
    prompt_temp = '''
With the information provided try to answer the question.
If you cant answer the question based on the information either say you cant find
an answer or unable to find an answer.
This is related to cricket domain. So try to understand in depth about the context
and answer only based on the information provided. Dont generate irrelevant
answers
Context: {context}
Question: {question}
Do provide only correct answers
Correct answer:
    '''
    custom_prompt_temp = PromptTemplate(template=prompt_temp,
                        input_variables=['context', 'question'])

    retrieval_qa_chain = RetrievalQA.from_chain_type(llm=llm,
                                retriever=db.as_retriever(search_kwargs={'k': 1}),
                                chain_type="stuff",
                                return_source_documents=True,
                                chain_type_kwargs={"prompt": custom_prompt_temp}
                            )

    def cricbot(query):
        answer = retrieval_qa_chain({"query": query})
        return answer["result"]
    if 'user' not in st.session_state:
        st.session_state['user'] = ["Hey there"]
    if 'assistant' not in st.session_state:
        st.session_state['assistant'] = ["Hello I am Cricbot and I am ready to help
with your doubts in cricket"]
container = st.container()
    with container:
        with st.form(key='cricket_form', clear_on_submit=True):
            user_input = st.text_input("", placeholder="Talk to your csv data here",
key='input')
            submit = st.form_submit_button(label='Answer')
        if submit:
            output = cricbot(user_input)
            st.session_state['user'].append(user_input)
            st.session_state['assistant'].append(output)
    if st.session_state['assistant']:
```

```
for i in range(len(st.session_state['assistant'])):
    message(st.session_state["user"][i], is_user=True, key=str(i) + '_user')
    message(st.session_state["assistant"][i], key=str(i))
```

Output:



Model:

Tokenization and Processing in Llama-2-7B Chat: A Deep Dive into Subword Encoding and Efficient Inference

At the tokenization level, the `llama-2-7b-chat.ggmlv3.q4_1` model—like other large language models—processes text by breaking it down into smaller units called **tokens**. This tokenization process is critical, as it enables the model to interpret and generate language by representing words and phrases as numerical data the model can understand.

1. Tokenization Process

- **Subword Tokenization:** Llama 2 uses **Byte-Pair Encoding (BPE)**, a tokenization technique that splits text into subword units. For example, common words like "computer" may be represented as a single token, while less common words like "quantization" might be split into smaller pieces like "quant", "iza", and "tion."
- **Byte-Pair Encoding (BPE):** BPE iteratively merges the most frequent pairs of characters (or sequences of characters) in a dataset. Over time, it builds up tokens that can efficiently represent frequent words or subwords in the language. This method allows the model to handle both frequent and rare words effectively.
- **Vocabulary**: The BPE process generates a fixed vocabulary of tokens (words, subwords, or even individual characters), which the model then uses for both input and output. For Llama 2, the vocabulary size is set based on its training data, optimizing for both common and uncommon words to cover the broadest range of text inputs.

2. Tokenization Example

- **Word-Level vs. Subword-Level**: If given the sentence "Cricket is an exciting sport," BPE might tokenize it as:

```
["Cr", "icket", " is", " an", " exciting", " sport"]
```

Here, the less frequent word "Cricket" is split into two subword tokens ("Cr" and "icket"), while the more frequent words like "is" and "sport" remain whole.

- **Handling Out-of-Vocabulary Words**: When encountering rare or new words (e.g., specific cricket jargon), the model breaks them down into known subwords or characters. This subword handling allows the model to generalize better across languages or specialized vocabularies like cricket terms.

3. **Encoding and Decoding Tokens**

- **Token IDs**: Each token in the vocabulary is assigned a unique ID, a numerical representation that the model processes. For example, "Cr" and "icket" might be assigned specific token IDs based on their frequency in the training data.
- **Embedding Layer**: Once tokenized, the tokens are fed through an **embedding layer**, which transforms each token ID into a dense vector representation. This vector captures the semantic meaning of each token relative to others in the vocabulary.

4. **Handling Context via Positional Encoding**

- Since transformers process tokens in parallel, Llama 2 needs **positional encodings** to maintain the order of tokens. This encoding helps the model understand that "Cricket is an exciting sport" is different from "Sport is an exciting cricket.

5. **Token Limitation and Quantization Impact**
   - **Token Limits**: The model has a maximum token limit (e.g., 512 or 1024 tokens), which means long inputs are truncated. This limit defines how many tokens the model can handle at once, which impacts how much of the conversation context it can retain in memory.
   - **Quantization and Token Processing**: With quantization (`q4_1`), the precision of token embeddings is reduced to 4-bit, meaning each token's representation in memory takes up less space. This reduction allows the model to handle tokens more efficiently, although some precision may be sacrificed.

6. **End-to-End Tokenization in Practice**
   - **Input**: When a user inputs a question, it's tokenized into IDs, transformed into embeddings, and passed through the transformer layers.
   - **Inference**: At each layer, self-attention mechanisms allow tokens to interact, refining the context and relations between tokens. For instance, in "Who won the last cricket match?", the model identifies the relation between "who" and "won" to focus on an answer related to a winner.
   - **Output Generation**: The model generates tokens sequentially for its response. It predicts the next token based on previously generated tokens, which are then decoded back into words and phrases for the user.

Challenges in Implementing and Using Llama-2-7B Chat Model:

1. **Large Model Size and Resource Demands**
   - **Challenge**: Llama-2-7B has 7 billion parameters, making it computationally expensive to run, especially in environments with limited hardware resources (e.g., personal computers or low-memory servers). Inference time can be slow, and memory usage can be high.
   - **Solution:** Use model quantization techniques like `q4_1` to reduce memory footprint and optimize performance without significantly compromising accuracy.

2. **Tokenization and Vocabulary Handling**
   - **Challenge**: Tokenization using Byte-Pair Encoding (BPE) requires splitting text into subword units. While BPE handles most words, rare or out-of-vocabulary words may result in suboptimal token splits, leading to possible misinterpretation of specialized or domain-specific terms.
   - **Solution**: Properly fine-tune the model on domain-specific data (like cricket-related data) to improve its handling of jargon and improve tokenization for specific terms.

3. **Contextual Understanding in Long Conversations**
   - **Challenge**: The model has a token limit (e.g., 512 or 1024 tokens). This can be problematic in long conversations where important context might be truncated, potentially leading to a loss of information across multiple turns.

- **Solution**: Implement memory management strategies to retain and provide context from previous exchanges, or use models with longer context windows if available.

4. **Inference Speed and Latency**
   - **Challenge**: Due to its size and the complexity of transformer models, Llama-2-7B can experience slow inference times, especially on devices with limited processing power. This leads to a poor user experience in real-time applications like chatbots.
   - **Solution**: Quantization (e.g., `q4_1`) reduces precision, making the model faster and more memory-efficient without compromising much on accuracy. Additionally, leveraging hardware acceleration (like GPUs) can improve speed.

5. **Fine-Tuning Challenges for Domain-Specific Data**
   - **Challenge**: Fine-tuning Llama-2-7B on specialized datasets (like cricket data) requires careful handling of both data and training parameters. Without proper fine-tuning, the model may struggle to understand domain-specific questions and provide accurate responses.
   - **Solution**: Use techniques such as prompt engineering, domain-specific fine-tuning, and retraining the model on a large corpus of relevant data to make it more knowledgeable in a specific field.

6. **Model Bias and Ethical Concerns**
   - **Challenge**: Like many large language models, Llama-2-7B can inherit biases present in its training data, which may lead to biased, inappropriate, or misleading responses.
   - **Solution**: Implement safety filters, monitor model outputs for bias, and continuously update the model with more diverse and balanced training data.

7. **Handling Ambiguous Queries**
   - **Challenge**: The model might struggle with ambiguous questions or inputs that are poorly formulated or lack sufficient context. This can lead to incomplete or irrelevant responses.
   - **Solution**: Improve user input validation and provide feedback to clarify ambiguous queries. Additionally, incorporate fallback mechanisms for uncertain or unclear inputs.

8. **Quantization Trade-Offs**
   - **Challenge**: While quantization reduces model size and increases speed, it may also introduce some loss in accuracy, especially for complex or nuanced queries.
   - **Solution**: Fine-tune quantized models and adjust settings to balance between inference speed and model performance, ensuring that accuracy remains high for important queries.

Conclusion :

In conclusion, the Excel Assistant Chatbot enhances data management by offering a user-friendly solution for efficient interaction with Excel files. It streamlines tasks such as data validation, summarization, and insight extraction, providing users with real-time responses and eliminating the need for manual processing. This chatbot is an effective tool for those looking to simplify and accelerate Excel-based data handling, making complex data structures more manageable and accessible.