**Detailed Guide on the Functioning of the Coding Practice Chatbot Code**

**Overview**

The Coding Practice Chatbot is built using Streamlit and LangChain, integrating a custom LLM model (CustomLLamaLLM). It dynamically generates coding questions, evaluates user responses, and provides adaptive learning experiences based on past performance.

**Code Breakdown**

**1. Importing Dependencies**

import streamlit as st

import sqlite3

import json

from lmstudio\_llama import CustomLLamaLLM

from langchain\_core.prompts import PromptTemplate

import streamlit.components.v1 as components

from streamlit\_ace import st\_ace

* streamlit: Used for UI rendering.
* sqlite3: Manages database operations.
* json: Handles structured data processing.
* CustomLLamaLLM: Custom LLM for question generation and feedback.
* PromptTemplate: Defines the input format for the LLM.
* st\_ace: Embeds an interactive code editor.

**2. Database Initialization**

def init\_db():

conn = sqlite3.connect("coding\_practice.db")

c = conn.cursor()

c.execute('''CREATE TABLE IF NOT EXISTS practice (

id INTEGER PRIMARY KEY,

question TEXT,

user\_answer TEXT,

feedback TEXT,

difficulty TEXT,

skill\_name TEXT,

subtopic TEXT,

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

)''')

conn.commit()

conn.close()

* Creates a SQLite database (coding\_practice.db).
* Defines a table to store questions, answers, feedback, difficulty, skill name, and subtopic.

**3. Generating Questions**

def get\_question(skill\_name):

subtopic, difficulty = determine\_next\_question(skill\_name)

idea\_prompt = PromptTemplate(

input\_variable={"skill\_name": "skill\_name", "difficulty": "difficulty", "subtopic": "subtopic"},

template="Generate a {difficulty} level coding question in {subtopic} for {skill\_name}."

)

chain = idea\_prompt | llm

result = chain.invoke({"skill\_name": skill\_name, "difficulty": difficulty, "subtopic": subtopic})

return result, subtopic, difficulty

* Determines the next subtopic and difficulty based on past performance.
* Uses PromptTemplate to generate a question dynamically.

**4. Answer Evaluation & Feedback**

def get\_feedback(question, user\_answer):

feedback\_prompt = PromptTemplate(

input\_variable={"question": "question", "user\_answer": "user\_answer"},

template="Evaluate the following answer for the given question. Provide detailed feedback including correctness, improvements, and best practices.\n\nQuestion: {question}\nAnswer: {user\_answer}\n\nFeedback:"

)

chain = feedback\_prompt | llm

feedback = chain.invoke({"question": question, "user\_answer": user\_answer})

return feedback

* Uses a LangChain PromptTemplate to generate feedback.
* Evaluates correctness, improvements, and best practices.

**5. Correct Answer Retrieval**

def get\_correct\_answer(question):

answer\_prompt = PromptTemplate(

input\_variable={"question": "question"},

template="Provide the correct answer for the following question:\n\n{question}\n\nAnswer:"

)

chain = answer\_prompt | llm

correct\_answer = chain.invoke({"question": question})

return correct\_answer

* Calls the LLM to retrieve the correct answer.
* Displays the solution if the user cannot solve the question.

**6. Storing Results**

def store\_result(question, user\_answer, feedback, difficulty, skill\_name, subtopic):

conn = sqlite3.connect("coding\_practice.db")

c = conn.cursor()

c.execute("INSERT INTO practice (question, user\_answer, feedback, difficulty, skill\_name, subtopic) VALUES (?, ?, ?, ?, ?, ?)",

(question, user\_answer, feedback, difficulty, skill\_name, subtopic))

conn.commit()

conn.close()

* Saves user responses and feedback into the database.

**7. UI Components**

**Layout & Components**

st.set\_page\_config(layout="wide") # Utilize full width

st.title("Coding Practice Chatbot")

st.markdown("---")

* Sets the UI layout and title.

**Left Panel - Displaying Question & Feedback**

left, right = st.columns([1.2, 1.8])

with left:

st.subheader("LLM Outputs")

st.write(f"### Question: {st.session\_state['question']}")

st.write(f"### Feedback: {st.session\_state['feedback']}")

* Shows generated questions and feedback.

**Right Panel - User Inputs**

with right:

skill\_name = st.selectbox("Select your skill", ["Python", "Bash", "FastAPI", "Streamlit", "Django", "SQL", "PyTorch", "Kafka (Python)", "MongoDB", "Docker", "Kubernetes", "Terraform", "Ansible", "GitHub Actions", "Apache Airflow (Python)", "Git", "Pyspark"])

st.session\_state['skill\_name'] = skill\_name

* Allows users to select a skill from a predefined list.

**Code Editor**

language = get\_language\_from\_skill(skill\_name)

st\_ace(

language=language,

theme="monokai",

key="code\_editor",

height=400

)

* Provides an embedded coding environment with syntax highlighting.

**Action Buttons**

col1, col2 = st.columns([1, 1])

with col1:

if st.button("Get a Question"):

question, subtopic, difficulty = get\_question(skill\_name)

st.session\_state['question'] = question

st.session\_state['subtopic'] = subtopic

st.session\_state['difficulty'] = difficulty

st.session\_state['feedback'] = ""

st.rerun()

* Fetches and displays a new question upon button click.

if st.button("Submit Answer"):

feedback = get\_feedback(st.session\_state['question'], code\_editor)

st.session\_state['feedback'] = feedback

store\_result(st.session\_state['question'], code\_editor, feedback, st.session\_state['difficulty'], skill\_name, st.session\_state['subtopic'])

st.rerun()

* Evaluates user answers and updates feedback.

**Conclusion**

This guide explains the functionality and flow of the code, detailing database interactions, UI elements, and LLM integrations. Let me know if you need further explanations or modifications!