Conversational AI system using AWS Bedrock and LangChain

Goal : To combines a conversational AI model (Claude 3 Haiku) with a knowledge retrieval system (Claude v2) to provide contextually aware responses.

The script implements a conversational AI system with the following key features:

- **Conversation Management**: Uses `ConversationChain` with memory to maintain context across interactions.

- **Knowledge Base Integration**: Retrieves relevant information from an AWS Knowledge Base using `AmazonKnowledgeBasesRetriever`.

- **Response Generation**: Combines knowledge base responses with conversational AI to provide enhanced answers.

- **Persistence**: Saves conversation history (questions, responses, and sources) to a JSON file for future reference.

- **Connection Testing**: Includes a function to test the connectivity and functionality of both the chat and knowledge base components.

3. Workflow

1. **Input Processing**:
    - The user provides an input text (question or statement).

2. **Knowledge Base Retrieval** (if enabled):
    - The system queries the knowledge base using the retriever and retrieval model.
    - Relevant documents and metadata are extracted.

3. **Response Generation**:
    - If knowledge base results are available, they are included in the prompt for the chat model.
    - The chat model generates a response using the conversation memory for context.

4. **Persistence**:
    - The question, response, and optional sources are saved to the JSON file.

5. **Output**:
    - The system returns the response, along with the knowledge base response and sources (if applicable).

4. Code Structure

4.1. Functions

- `save_conversation_to_json`
  Persists conversations under a server ID (`server_1234`) for auditability.

- `create_chat_llm` / `create_retrieval_llm`
  Initialize Bedrock models with specific parameters (e.g., `temperature=0.1` for deterministic outputs).

- `create_retriever`
  Configures the knowledge base retriever for vector-based document search.

- `enhanced_conversation`
  Core logic for RAG: retrieves data, generates responses, and logs results.

- `test_connections`
  Validates Bedrock and Knowledge Base connectivity.

4. Error Handling

- **Knowledge Base Retrieval Errors**: If the knowledge base retrieval fails, the system falls back to the chat model without knowledge base augmentation.

- **JSON File Corruption**: If the chat history JSON file is corrupted or empty, the system initializes a fresh data structure.
- **Connection Testing**: The `test_connections()` function ensures that both the chat and knowledge base connections are functional.

Front End

This Streamlit app provides a user-friendly interface for the AWS Bedrock-powered chatbot backend. It allows users to:

- Interact with the chatbot in real-time.
- View conversation history with source references.
- Toggle knowledge base integration.
- Manage chat history (clear, load more, test connections).

3. Workflow

1. **Initialization**
   - Loads past chat history from `chat_history.json` on startup.
   - Initializes AWS Bedrock memory from the backend.

2. **User Interaction**
   - User types a query in the chat input box.
   - Input is sent to the backend (`enhanced_conversation` function).
   - Response and sources are displayed, then saved to history.

3. **History Management**
   - Clear Chat: Resets the session state and JSON file.

4. Code Structure

4.1. Functions

- `load_chat_history`
  Retrieves the last 6 entries (3 Q&A pairs) from `chat_history.json`
- `save_chat_to_json`
  Persists the current session's chat history to the JSON file.
- `format_past_history`
  Converts backend history format to Streamlit's message-role structure.

4.2. UI Components

- **Chat Window**
  Displays messages with alternating user/assistant roles and source expanders.
- **Sidebar**
  - **Settings:** Enable/disable knowledge base.
  - **Diagnostics:** Test AWS connections.
  - **History Controls:** Clear or load full history.

This Streamlit interface bridges the gap between the RAG backend and end-users, offering a clean, interactive experience with transparency into AI responses. It’s ideal for demo purposes or lightweight deployments.