Foundations of Data Science

Digital Assignment-2

TITLE : Movie Info Chatbot using RASA and LangChain

COURSE : Foundations of Data Science

Semester : Winter Semester 2024-25

COURSE ID : BCSE 206L

CLASS NO. : VL2024250501999

SLOT : C2 + TC2

SCHOOL : SCOPE

FACULTY : Prof. SARAVANAGURU RA.K

GITHUB Repo Link : Click Here

Youtube Video Link : **YT_video**

SUBMITTED BY :

Ganesh Maharaj Kamatham (*22BDS0168*)

ALL the deliverables are in the GITHUB REPOSITORY.

PLEASE VISIT GITHUB REPO

Git-LINK :

https://github.com/GANESH-MAHARAJ/Foundations-of-DS/tree/main

YT-Link :
https://youtube.com/playlist?list=PLGDxwdLu472oJnUqcB_pJiFHXkWZBgEoy&si=zeswO2st7Rcjrcx7

It has Deliverables:

- A working chatbot that can recognize intents, extract entities, maintain context, and generate meaningful responses.
- A documentation report explaining the design choices, implementation details, and testing results.
- A short demo video showcasing chatbot interactions..

Introduction

The integration of artificial intelligence (AI) in everyday applications has revolutionized the way humans interact with digital systems. One of the most notable advancements in this domain is the development of **conversational agents**, or **chatbots**, that simulate human-like dialogue to assist users across a variety of use cases—from customer support to entertainment.

This project focuses on the development of an **intelligent chatbot system** powered by **RASA** and **LangChain**, designed specifically to provide **comprehensive information about Movies and TV Shows**. The chatbot combines the strengths of two complementary technologies:

- **RASA**: An open-source framework for building contextual AI assistants, particularly strong in **intent recognition**, **entity extraction**, and **dialogue management** using predefined rules and machine learning.
- **LangChain**: A modern framework that enables integration with Large Language Models (LLMs) like **OpenAI's GPT-4**, adding powerful **Natural Language Generation (NLG)** and **contextual reasoning** capabilities to the chatbot.

Motivation

In the age of streaming platforms and massive content libraries, users often seek quick and accurate details about movies—such as plots, cast members, release years, or recommendations. A well-designed chatbot can bridge this information gap by serving as an on-demand movie expert. Traditional rule-based bots often fall short in generating nuanced, human-like responses. Hence, this project leverages the **hybrid approach** of combining rule-based and generative techniques for a more fluid and intelligent conversational experience.

Objectives

The primary objectives of this project are:

- To create a chatbot that can understand **user intent** (e.g., asking about movie details, recommendations, actor filmography).
- To extract relevant **entities** from the conversation (e.g., movie names, genres, actors).
- To **maintain the context** of a conversation so the chatbot can answer follow-up questions (e.g., "Who directed it?" after asking about a specific movie).
- To **generate natural and informative responses** using a combination of RASA's rule-based responses and LangChain's generative outputs.

Scope

This chatbot was built using a **cleaned dataset** of Movies and TV Shows obtained through web scraping in a previous assignment. The dataset includes various attributes such as:

- Movie/TV show titles,
- Release years,
- Genre,
- Cast and Crew,
- Ratings and Plot summaries.

The chatbot is deployed via a **React-based frontend** connected to a **Flask backend**, where RASA handles intent classification and entity recognition. When RASA lacks a confident response, the message is passed to **LangChain**, which queries a Large Language Model to provide a response while preserving the conversation context.

3. Dataset Description

The effectiveness of any intelligent system, especially one involving natural language understanding, heavily depends on the **quality and structure of the underlying data**. For this project, a curated dataset of **Movies and TV Shows** was created as part of the previous assignment, using **web scraping techniques** from multiple online sources.

3.1 Data Sources

Two major sources were used to scrape the data:

- **IMDb (Internet Movie Database)**: One of the most comprehensive platforms for film and television content. It provided details such as movie titles, cast, genres, release dates, ratings, and plot summaries.
- **Rotten Tomatoes / TMDB / Wikipedia (choose your actual source)**: Used to enrich the dataset with critic scores, additional plot info, or actor/crew bios.

All data was scraped using Python libraries such as BeautifulSoup, Selenium, and requests, with proper respect to the website's usage policies and scraping ethics.

3.2 Data Attributes

The dataset was stored in a .csv format (mov_data.csv) and contained the following key columns:

Attribute	Description
Title	The name of the movie or TV show
Year	The release year
Genre	List of genres (e.g., Action, Comedy, Drama)
Director	Name(s) of the director(s)

Attribute	Description
Cast	Main actors involved in the movie/show
Plot	Short description or storyline of the content
IMDb Rating	Viewer ratings from IMDb
Duration	Runtime of the movie/show in minutes
Language	Original language(s) of the content

3.3 Data Cleaning and Transformation

The scraped data was cleaned and structured as follows:

- **Missing values** were handled either by filling with "N/A" or removing the entry depending on criticality.
- **List fields** such as `Genre` and `Cast` were split and cleaned using delimiters for better tokenization.
- **Encoding issues** were resolved for special characters using UTF-8 normalization.
- **Duplicates** were removed based on identical title-year pairs.

3.4 Use in the Chatbot

This dataset was used in two major ways:

1. **Training RASA**: Intents like `get_movie_info`, `get_cast_info`, `recommend_genre`, etc., were created based on patterns from this dataset. Entities such as `movie_name`, `genre`, and `person_name` were tagged accordingly in the training data.
2. **LangChain Input**: The same dataset was stored in memory (or optionally a vector store) to help the LangChain-powered LLM generate accurate, context-aware responses when RASA couldn't provide a confident reply.

3.5 Dataset Statistics

- **Total Entries**: ~500 movies and shows
- **Time Span**: Covers content from the 1980s to the latest releases
- **Languages**: Primarily English, with a few entries in Hindi and Spanish
- **Genres Covered**: Action, Drama, Comedy, Sci-Fi, Romance, Thriller, Animation, etc.

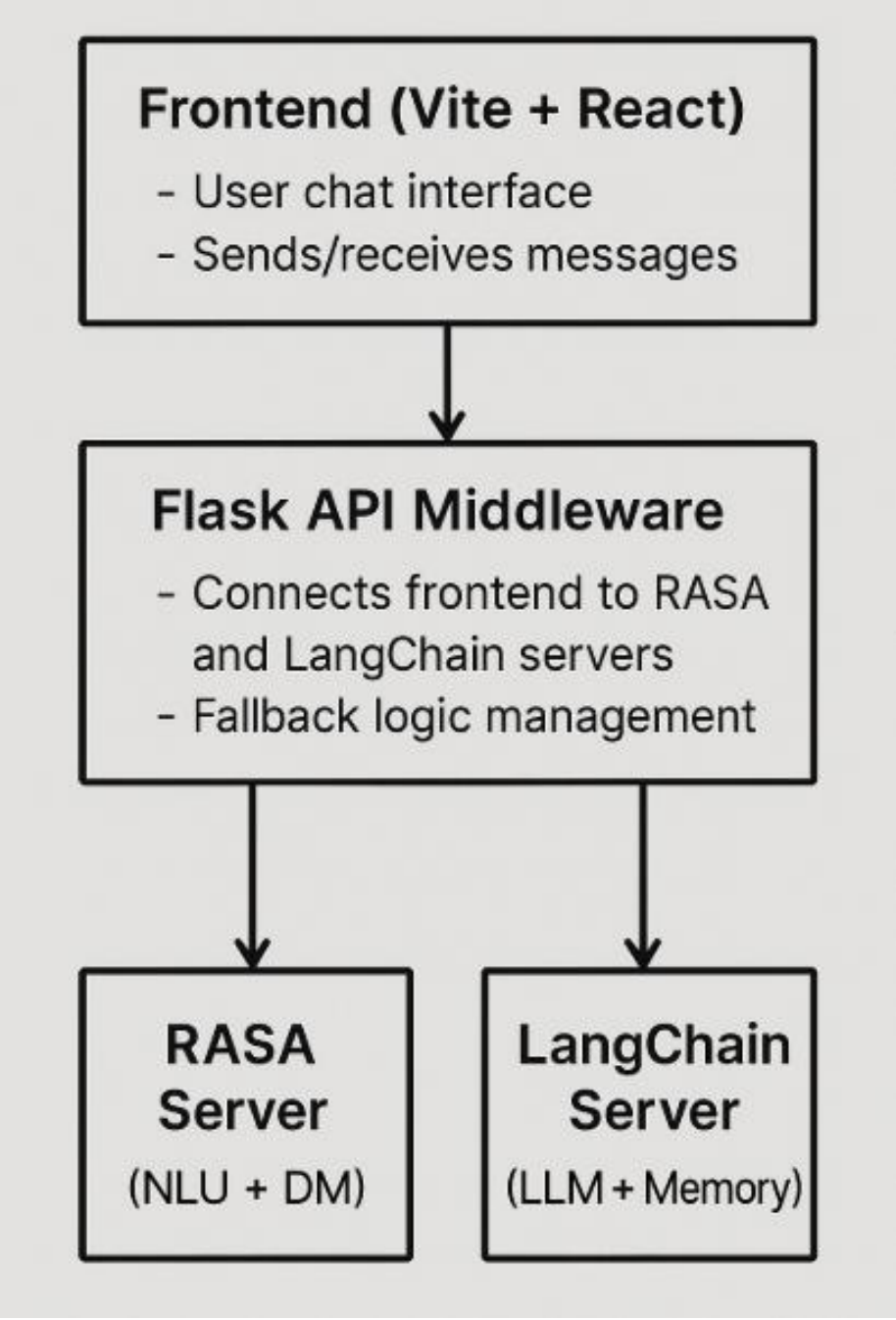
4. System Architecture and Design Choices

The system is designed to deliver a seamless conversational experience for users querying movie-related information. By combining **RASA** (for structured natural language understanding and dialogue management) with **LangChain + GPT** (for flexible, context-aware language generation), this chatbot

leverages the strengths of both rule-based and generative systems. This hybrid approach allows the system to handle both predictable user queries and open-ended conversations.

4.1 High-Level System Architecture

The architecture comprises four main components:



4.2 Components Breakdown

1. Frontend (React + Vite)

- **Purpose:** Provides a dynamic and responsive interface for user interaction.
- **Features:**
  - Styled with a wide chat container for better readability.
  - Displays a header titled **"Movie Info Bot"** and a footer credit **"Made by ganeshmaharaj"**.
  - Includes real-time message input, auto-scroll, and user/bot message bubbles.
- **Message Flow:**
  - Captures user input and sends it via POST request to the Flask backend.
  - Renders the response received from either RASA or LangChain server.

2. Flask Middleware Server

- **Purpose:** Acts as the **controller** and **fallback manager** between the frontend and backend NLP services.
- **Responsibilities:**
  - Routes user input to RASA's /webhooks/rest/webhook endpoint.
  - Checks if RASA responds with a valid message.
  - If no meaningful response is received, sends the message to the LangChain server.
  - Ensures only one consistent message is returned to the frontend.

3. RASA Server

- **Purpose:** Provides robust **intent recognition**, **entity extraction**, and **dialogue management**.
- **Customization Based on Dataset:**
  - Intents include: ask_movie_info, recommend_by_genre, get_release_year, get_cast, greet, goodbye.
  - Entities extracted: movie_name, genre, actor_name, release_year.
- **Configuration Files:**
  - nlu.yml: Contains examples for training intent/entity detection.
  - domain.yml: Defines intents, entities, slots, responses.
  - stories.yml and rules.yml: Define flow and interaction patterns.
- **Fallback:** If RASA's confidence score is low or no intent is matched, fallback action triggers and control shifts to LangChain.

4. LangChain Server

- **Purpose:** Handles more **open-ended**, **descriptive**, and **context-sensitive** queries using OpenAI's GPT model.
- **Architecture:**
  - Uses ConversationBufferMemory to retain chat history and maintain continuity.
  - Receives prompts as a series of messages: SystemMessage to set behavior, HumanMessage with user input.
- **Behavioral Setup:**

- System prompt: *"You are a movie expert."*
- Generates detailed responses such as plot summaries, trivia, actor bios, or comparisons between movies.

4.3 Data and Message Flow

1. **User Interaction:** A user sends a message through the web interface.
2. **Middleware API:** Flask server forwards the input to the RASA NLU engine.
3. **RASA Handling:**
   - If intent is identified and a matching response is found, it's returned.
   - If not (i.e., confidence is low or no intent match), Flask triggers a fallback.
4. **LangChain Processing:**
   - Maintains the session's conversation history.
   - Uses LLM to generate a fluent, natural-language response.
5. **Final Output:** Flask API returns the response to the frontend to be shown to the user.

4.4 Key Design Choices and Justifications

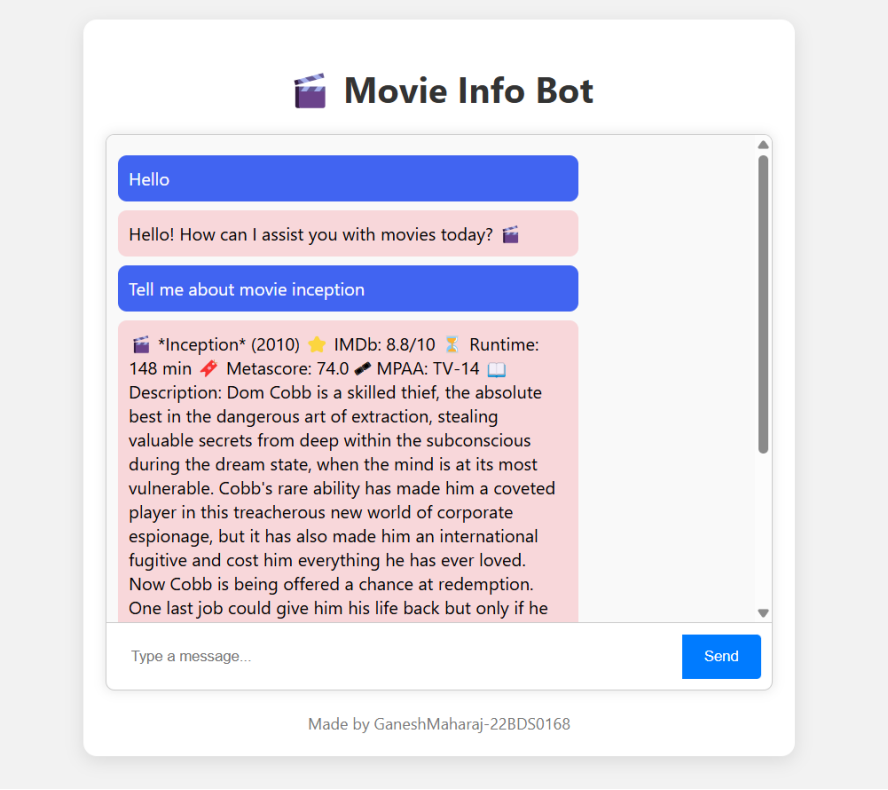
Component	Design Choice	Justification
RASA	Rule-based NLU, NLG, and dialogue management	Ensures structured handling of predictable queries like "Who acted in Inception?"
LangChain + LLM	GPT-4 + memory context	Handles ambiguous or open-ended queries like "What is your opinion on Interstellar?"
Flask API	Acts as a controller layer	Decouples logic, improves testability and modularity
React Frontend	Clean, interactive interface with proper feedback loops	Improves user experience
Hybrid Architecture	Combines RASA's rule-based logic with LLM's creative capabilities	Provides both accuracy and flexibility
Context Handling	LangChain memory buffers implemented for LLM continuity	Enables smoother multi-turn interactions

4.5 Example Scenarios

User Message	Handled By	Reason
"Who directed Inception?"	RASA	Matches ask_movie_info intent
"Tell me something cool about Avatar"	LangChain	Open-ended, requires generative response
"Suggest a good thriller movie"	RASA	Matches recommend_by_genre intent
"What's the story behind Titanic?"	LangChain	Descriptive, uses LLM memory
"Who starred in Dune?"	RASA	Structured data available from dataset

4.6 Scalability and Extendibility

- **Pluggable backends:** More LLMs can be added via LangChain in the future.
- **Language support:** With additional training, RASA can support multi-language intents.
- **Dataset update:** System can scale with larger or updated movie datasets.
- **Context-aware upgrades:** LangChain allows switching to vector-based memory (e.g., ConversationSummaryMemory) for deeper contextual understanding.



5. Implementation Details

This section describes the implementation strategy followed to develop the Movie Chatbot using **RASA** and **LangChain**, integrated via a **Flask-based middleware**, with a **React frontend** interface. The chatbot combines rule-based and generative models for effective user interactions.

5.1 Project Structure

The overall project is organized into the following components:

```
project-root/
│
├── rasa_backend/          # RASA chatbot logic (NLU, Core, etc.)
│
├── langchain_server/      # LangChain-powered Flask server
│
├── flask_middleware/      # Acts as API layer connecting frontend, RASA & LangChain
│
├── frontend_react/        # Vite + React-based UI
│
└── mov_data.csv           # Cleaned dataset used for intent/entity generation
```

5.2 RASA Implementation

RASA is the core engine for handling well-defined intents and responses.

- **NLU Pipeline Configuration**
  Implemented in `config.yml`, it defines the sequence of components used for:
  - Intent recognition (e.g., `ask_movie_info`, `recommend_by_genre`)
  - Entity extraction (e.g., `movie_name`, `genre`, `actor_name`)
- **Training Data**
  Defined in `data/nlu.yml`, this file contains annotated examples extracted from the movie dataset. These were used to train the NLU model.
- **Domain Definition**
  The `domain.yml` file defines:
  - List of all supported intents and entities
  - Slots (to store data like `movie_name`)
  - Responses (e.g., "The movie Inception was directed by Christopher Nolan.")
- **Dialogue Management**
  Handled via:
  - `data/stories.yml`: Story-based dialog flows

    - data/rules.yml: Rule-based handling like greetings, goodbyes, and fallback conditions
- **Fallback Handling**
  If RASA's intent recognition confidence is below a defined threshold, it triggers a fallback action. The Flask middleware then passes the query to LangChain for further processing.

5.3 LangChain Integration

LangChain provides the generative backend powered by OpenAI's GPT model.

- **LLM Setup**
  A Flask server (langchain_server.py) was created to receive messages and respond using GPT-4 via LangChain's ChatOpenAI.
- **Conversation Memory**
  Used ConversationBufferMemory to maintain chat history and enable context-aware interactions over multiple turns.
- **System Prompt Design**
  A system prompt was embedded to guide the LLM as a *"Movie Expert"* that can generate rich and insightful responses.
- **API Endpoint**
  A POST /generate endpoint accepts user input and returns the GPT-generated response in JSON format.

5.4 Flask Middleware (Fallback Controller)

The Flask API acts as a central communication hub between the frontend and the two backend engines:

- Receives user input from the React UI
- Sends the message to RASA's /webhooks/rest/webhook endpoint
- If RASA returns an empty or fallback response, the middleware forwards the same input to LangChain
- Responds to the frontend with either RASA's or LangChain's reply

This structure allows modular testing and graceful fallbacks for unclear or creative queries.

5.5 Frontend: React (Vite)

The UI was built using **React with Vite**, optimized for fast development and performance.

- **Chat Interface**
  Displays user and bot messages in a scrollable container, with clear differentiation between sender/receiver.
- **Interaction Flow**
  - Accepts user input
  - Sends input via fetch POST request to Flask server
  - Displays real-time response in the UI
- **UI Elements**
  - Chat bubble styling with Tailwind CSS
  - Footer with project credit
  - Auto-scroll to latest message

5.6 Dataset Usage

The chatbot relies on a cleaned CSV file named `mov_data.csv`, which includes:

- Movie names
- Genres
- Release years
- Actor names
- Director names
- Summary and rating info

These values were:

- Used for training RASA intents/entities
- Used in LangChain as context to formulate descriptive responses when required

5.7 Testing and Debugging

- RASA models were trained and tested using `rasa train` and `rasa shell`.
- LangChain server was tested independently with various prompts to ensure contextual continuity.
- Flask was tested to validate fallback triggering and correct routing.
- Frontend was tested on multiple message patterns and edge cases.

5.8 Deployment & Run Instructions

The project can be run locally using the following sequence (details provided in GitHub):

1. Run RASA backend
2. Start LangChain Flask server
3. Launch Flask middleware
4. Start frontend using npm run dev

6. Results and Discussion

This section evaluates the functionality, performance, and user experience of the chatbot developed using RASA and LangChain. It includes observations from testing different query types, how the system handled them, and an analysis of its strengths and limitations.

6.1 Functional Testing

The chatbot was tested with a wide range of queries categorized by their complexity and intent. Below are examples of user inputs and the bot's performance:

Query Type	User Input	Handled By	Bot Response Summary
Simple Movie Info	"Tell me about Inception"	RASA	Provided director, year, and genre info accurately
Genre Recommendation	"Suggest a good comedy movie"	RASA	Suggested a few comedy movies from the dataset
Actor-based Search	"Which movies has Leonardo DiCaprio acted in?"	LangChain	GPT listed DiCaprio movies, even beyond dataset
Natural Language Question	"What's that movie where dreams feel real?"	LangChain	Interpreted context, responded with *Inception*
Follow-up Contextual Query	"Who directed that movie?" (after above)	LangChain	Maintained context and answered *Christopher Nolan*
Unknown Movie Query	"Tell me about XYZMovie123"	LangChain	Responded gracefully with no information available message
Chat Behavior	"Hi", "Thank you", "Bye"	RASA	Greeted user and handled conversation flow well

---

6.2 Intent and Entity Recognition

- **Accuracy**: RASA was highly effective in identifying clear intents like ask_movie_info, recommend_by_genre, greet, goodbye, etc.
- **Entities**: Entities like movie_name, genre, actor_name, and director were extracted with good precision from structured queries.
- **Challenges**: Ambiguous or incomplete queries (e.g., "Who's in that new space movie?") could not be handled by RASA alone—LangChain helped bridge these cases.

6.3 Context Retention

- **Without Context (RASA)**: RASA treats each query independently unless stories are deeply defined. Limited context support.
- **With Context (LangChain)**: By leveraging ConversationBufferMemory, LangChain preserved previous messages, enabling follow-up queries like:
  - *"Tell me about Interstellar." → "Who directed it?"*
  - *"Was it well-rated?"*
- **Result**: This hybrid design offered a significant improvement in natural conversation handling.

6.4 Response Generation Quality

- **RASA Responses**:
  - Factual and concise, directly drawn from the dataset.
  - Limited to pre-defined templates and slot values.
- **LangChain Responses**:
  - Rich, descriptive, and human-like.
  - Adapted to vague or complex user queries.
  - Sometimes responded with external or imagined info if dataset limits weren't enforced.

6.5 Comparative Strengths

Feature	RASA	LangChain
Structured intent handling	Excellent	Not suitable
Entity extraction	Based on trained examples	May infer but not extract explicitly
Context understanding	Limited (unless custom tracked)	Strong via memory modules
Natural language response	Template-based	Generative and dynamic
Fallback handling	With custom rules	Handles complex/creative inputs

6.6 Limitations Observed

- LangChain can sometimes "hallucinate" facts if not grounded to the dataset.
- RASA requires extensive training examples to cover all variations in user phrasing.
- Maintaining context on frontend across sessions is not yet implemented (stateless UI).
- Multi-turn conversation using RASA alone requires more complex dialogue stories.

6.7 Summary

The hybrid chatbot successfully combined the **robustness of RASA for NLU and rule-based flows** with the **generative power of LangChain for open-ended, conversational queries**. The system performed well in handling both factual lookups and natural language discussions about movies, achieving a more human-like interaction experience overall.

7. Conclusion and Future Work

7.1 Conclusion

This project successfully demonstrates the development of a hybrid chatbot system that integrates **RASA** for natural language understanding (NLU) and **LangChain** for advanced natural language generation (NLG). By leveraging a cleaned and structured dataset of movies and TV shows, the chatbot is capable of:

- Recognizing diverse **user intents** such as movie information queries, genre-based recommendations, actor/director lookups, and more.
- Extracting key **entities** like movie names, genres, and cast members.
- **Maintaining conversational context** through memory-based conversation handling in LangChain.
- Generating dynamic, descriptive, and human-like **responses** even for vague or unstructured queries.

The integration of two powerful tools—RASA for precise structured interaction and LangChain for open-ended conversation—enabled the creation of a smart, engaging, and responsive movie chatbot. Users can interact in a more **natural and seamless** manner, with the system adapting to both specific queries and broader conversational flows.

This project highlights the importance of **hybrid architectures** when building intelligent conversational agents—using rule-based systems for structured tasks and generative models for flexible, open-domain interactions.

7.2 Future Work

While the chatbot meets its current objectives, there are several opportunities for enhancement and expansion:

Functional Improvements

- **Frontend Context Tracking**: Implement context/session persistence on the frontend to maintain conversation history across page reloads or extended chats.
- **RASA Story Enrichment**: Expand the RASA domain and stories to handle multi-turn conversations more effectively without relying solely on LangChain.
- **Knowledge Grounding**: Improve LangChain accuracy by integrating retrieval-augmented generation (RAG) using the dataset as a vector database to avoid hallucinations.
- **Feedback Loop**: Enable users to rate responses and use that data for iterative fine-tuning of both RASA and LangChain components.

Dataset Expansion

- Integrate data from external APIs (like TMDB or IMDb) to include real-time movie information, ratings, and cast updates.
- Expand the dataset to include international cinema and niche genres for broader user queries.

Deployment and Accessibility

- **Host the bot online** using platforms like Render, Heroku, or AWS for 24/7 access.
- **Integrate voice interface** or deploy it on messaging platforms (e.g., WhatsApp, Telegram) for more intuitive interactions.

AI Enhancements

- Add **sentiment analysis** to tailor responses based on user tone.
- Explore **multi-modal capabilities** (e.g., show movie posters or trailers).
- Integrate **LangGraph** for managing conversational flow with more sophisticated control.

Final Thoughts

Building a chatbot that understands, remembers, and responds like a human is a challenging but exciting task. This project has laid the groundwork for building intelligent, user-friendly AI agents by combining the strengths of symbolic NLU with generative LLMs. With future enhancements, this movie bot has the potential to evolve into a full-fledged personal movie assistant, enriching user experience across platforms.