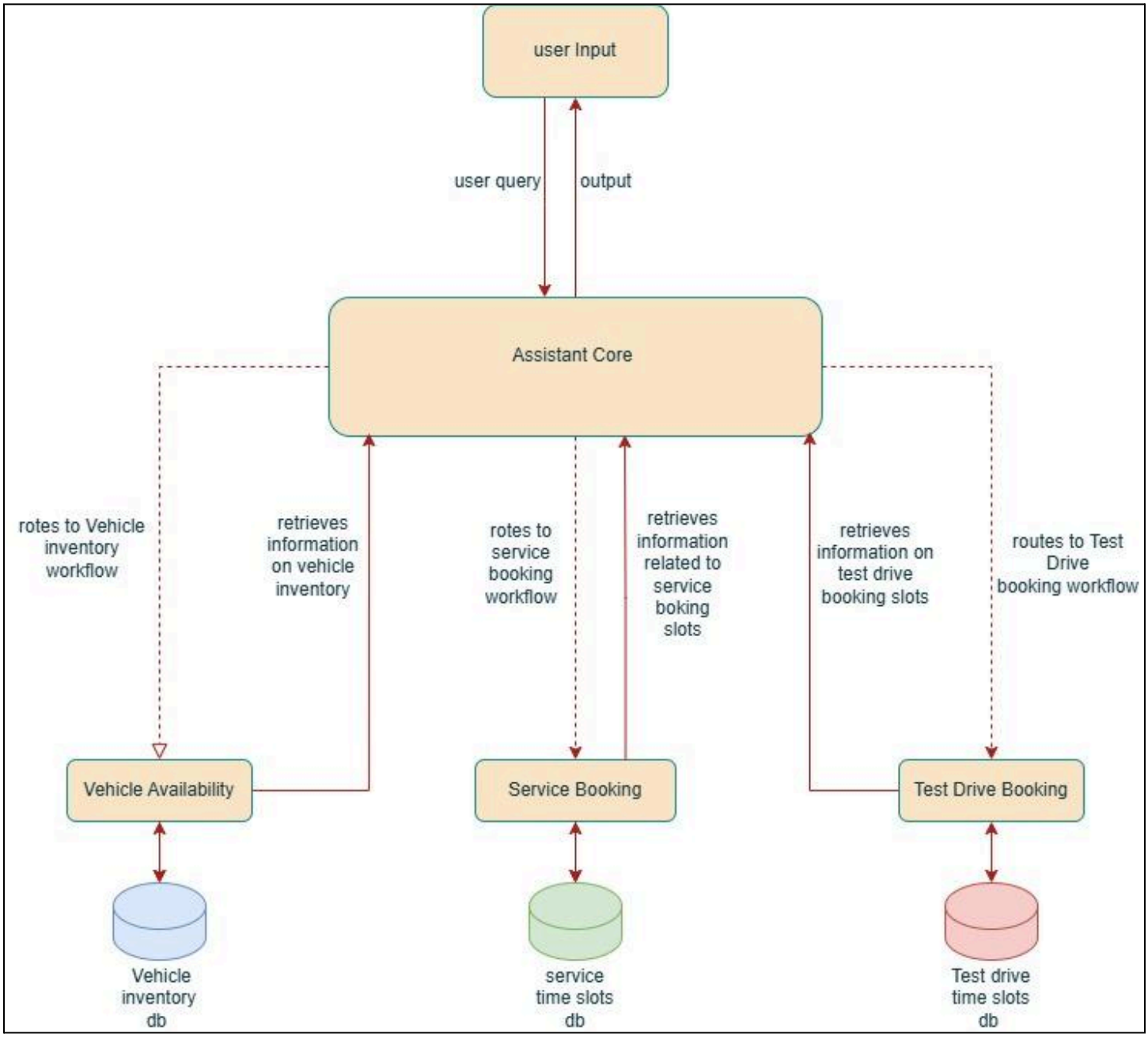
Toyota AI Assistant Documentation

Name : Sushma Duvvi

WorkFlow Diagram



Dotted line - indicates conditional edge

Solid line - indicates mandatory edge

Overview

The Toyota Assistant is a modular AI system built using LangGraph. It processes user queries and delegates them to specialized workflows ("skills") for tasks such as checking vehicle availability, booking service appointments, and scheduling test drives. The system leverages LLMs (using Groq and LLM models) to route requests and generate context-aware final responses.

Architecture

1. Core Assistant (AI Brain)

- **File:** assistant_core.py
- **Class/Methods:**
  - **AssistantCore:**
    - route_request(state: UserState): Uses an LLM-based routing prompt (defined in prompts.py) to determine which workflow (vehicle_availability, service_booking, or test_drive) should handle the query.
    - generate_answer(state: UserState): Aggregates data from the chosen workflow and invokes an LLM (using a final-answer prompt) to generate a friendly, informative response.

2. Specialized Workflows ("Skills")

Each workflow is implemented as an independent module:

- **Vehicle Availability Workflow**
  - **File:** workflows/vehicle_availability.py
  - **Class/Method:**
    - VehicleAvailabilityWorkflow: Simulates checking Toyota inventory using a predefined dataset (including at least 10 vehicles with model, year, and location). The process method returns inventory data that match the user's query.
- **Service Appointment Booking Workflow**
  - **File:** workflows/service_booking.py
  - **Class/Method:**
    - ServiceBookingWorkflow: Simulates booking service appointments by checking against a mock dataset of available appointment slots (date, time, location).
- **Test Drive Scheduling Workflow**
  - **File:** workflows/test_drive.py
  - **Class/Method:**
    - TestDriveWorkflow: Uses a hardcoded list of dealerships and available test drive slots to schedule a test drive based on user input.

3. LLM Configuration

- **File:** llm_config.py
- **Purpose:**
  - Stores and initializes LLM instances.
  - Uses environment variables (via a .env file) to securely load API keys.
  - Currently configured with Groq's **llama-3.3-70b-versatile** model for both core routing and final answer generation. An option to switch to OpenAI's Chat models is available.

4. LangGraph Implementation

- **File:** langgraph_workflow.py
- **Class:**

- **WorkflowManager:**
  - Defines the directed graph structure using the minimal LangGraph engine (see langgraph/graph.py).
  - Connects the core assistant and the specialized workflows.
  - Manages conditional routing based on the workflow decision stored in the UserState.
  - Compiles and executes the graph workflow, ultimately returning the final response.

Additional Files

- **State Management:**
  - **File:** state.py
  - Defines the UserState object that tracks the user query, workflow decision, retrieved data, and final response. It includes a method to format the state for LLM context.
- **Prompts:**
  - **File:** prompts.py
  - Contains prompt templates (e.g., FINAL_ANSWER_PROMPT) that guide the LLM to generate context-aware, friendly, and actionable responses.
- **LangGraph Engine:**
  - **File:** graph.py
  - Provides a minimal implementation of a state graph engine that drives the workflow execution.

Extensibility:

- **Adding More Skills:** New workflows can be implemented as separate modules and integrated into the LangGraph in langgraph_workflow.py.
- **Prompt Tuning:** Adjust the prompt templates in prompts.py to refine model responses.
- **LLM Switch:** Modify config/llm_config.py to change LLM providers or adjust parameters.

Debugging & Logging

- Each workflow and core component includes print statements to log key interactions (e.g., routing decisions and data retrieval).
- These logs help track the workflow execution path and can be used for performance evaluation and troubleshooting.

Conclusion

This modular design follows the project assignment requirements by:

- Delegating user queries to specialized workflows.
- Ensuring workflows are independent and easily extendable.
- Dynamically generating responses based on a mock dataset.
- Utilizing a LangGraph engine to structure the decision-making process.
- Providing a clean and maintainable codebase for further development.