Solution Architecture

1.0 Introduction

The MEX Assistant employs a **stateful agent system** architecture, implemented in **Python** and built upon the **LangChain ecosystem**, with **LangGraph** serving as the primary framework for orchestrating complex, stateful workflows.

The core of the system features an **LLM-based reasoning engine** (the "Planner Agent" in the primary workflow) that interprets user needs, plans actions, and synthesizes information. This central intelligence interacts with:

- A defined library of **modular Tools**, which encapsulate specific functions like data querying, chart generation, external API calls, or content drafting.
- **Human-in-the-Loop (HITL) mechanisms**, enabling the agent to pause execution and solicit clarification or confirmation directly from the merchant user via the chat interface.

This architecture supports **two distinct primary workflows** managed by LangGraph:

A flexible, conversational agent workflow designed for general interaction, insight delivery, and tool use.

A specialized pipeline workflow designed for AI-assisted ingestion, validation, and storage of user-uploaded data files.

This approach allows the system to handle both open-ended conversational requests and structured data processing tasks effectively, maintaining context through robust state management.

2.0 Core Architectural Components

2.1 Orchestration & State Management (LangGraph)

LangGraph serves as the backbone for defining, executing, and managing the stateful workflows within the MEX Assistant.

2.1.1 StateGraph Engine & Workflows

The core construct used is LangGraph's `StateGraph`. This allows defining application flows as graphs where nodes represent computation steps (LLM calls or Python functions) and edges represent the transitions between these steps. This architecture explicitly defines the two primary operational modes: Workflow 1 (Conversational Agent) and Workflow 2 (Data Ingestion Pipeline), allowing for clear separation and control over different task types.

2.1.2 State Schema Principles

A central state object, defined using Python's `TypedDict`, is maintained and passed throughout the graph execution. This state acts as the single source of truth at any point in the workflow, containing critical information such as:

- Full conversation history (user inputs, AI responses, tool calls/results).
- Parsed user intent and extracted parameters.
- Intermediate results from tool executions or LLM analysis.
- Flags or questions generated by the agent to trigger Human-in-the-Loop interactions.
- User responses provided during HITL.
- Data loaded or formatted during ingestion (Workflow 2). Nodes update the state by returning dictionaries containing the information to be merged or added.

2.1.3 Checkpointer & Memory (`MemorySaver`)

To enable state persistence across multiple steps within a workflow and maintain conversational memory, a LangGraph **Checkpointer** is employed. For this prototype, `langgraph.checkpoint.memory.MemorySaver` is utilized.

2.2 Reasoning Engine (LLM Nodes)

The core intelligence and decision-making capabilities are driven by Large Language Models, configured and invoked via LangChain integrations within specific LangGraph nodes.

2.2.1 Conversational Planner Agent

This is the central LLM node for the primary conversational workflow. Its key responsibilities include:

Interpreting user queries within the context of the conversation history (memory) retrieved from the state.

- Planning the sequence of actions needed to address the request.
- Selecting the appropriate tool(s) from the available library based on the plan.
- Determining the necessary parameters to pass to the chosen tool(s).

- Deciding when Human-in-the-Loop interaction is required and formulating the clarification question.
- Synthesizing results returned from tools or human input.
- Generating the final natural language response for the user.

2.2.2 Data Structure Analyzer

This specialized LLM node operates within the data ingestion pipeline (Workflow 2). Its role is more focused:

- Receives sample data and headers extracted from an uploaded file (via the `load_data` node and state).
- Compares the input data structure to the known target database schema.
- Infers column mappings, data types, and semantic meaning.
- Assesses its **confidence** in the interpretation.
- Generates specific **clarification questions** for ambiguous columns or low-confidence interpretations, which are then used to trigger the HITL mechanism if necessary.

2.3 Tool Library & Execution Nodes

Tools are defined as Python functions, typically decorated with LangChain's `@tool` decorator or described using Pydantic models. Clear docstrings and type hints are used to provide descriptions and parameter schemas, which the Planner Agent LLM uses to understand when and how to call each tool. Execution occurs within dedicated graph nodes that call these functions.

2.4 Human-in-the-Loop (HITL) Mechanism

The architecture supports pausing workflows to obtain direct input or confirmation from the merchant user.

2.4.1 interrupt() Function & Pausing

- LangGraph's `interrupt()` function is the core mechanism. It's called within a node (often conditionally, based on the Planner Agent's decision or low confidence from the Data Analyzer) to pause the graph's execution.
- Requires a Checkpointer to be configured to save the state upon pausing.

2.4.2 Application Layer Role (UI & Resume)

- The pause is handled externally by the application runner (e.g., Flask/Streamlit backend).
- This layer detects the interrupt, retrieves the necessary context (like the AI's question) from the saved state using the session/thread ID.
- It presents the question to the user via the Frontend Interface.
- Upon receiving the user's response, it updates the graph's state with the input.
- It resumes the graph execution using LangGraph's `Command` object, allowing the workflow to continue with the new information.

2.5 Data Layer

The data foundation for the assistant.

2.5.1 Data Sources (Provided Datasets & Uploaded Files)

- **Core Hackathon Data:** Utilizes the provided datasets (Transactions, Time, Keywords, Merchants, Items).
- **User Uploaded Data:** Workflow 2 is designed to ingest data from user-provided files (initially CSV, Excel; potentially images containing tables).
- *(External API data sources used by tools like Weather are handled within those tools).*

3.0 Workflow Realizations

This section details how the Core Architectural Components (Section 4) interact to execute the two primary workflows of the MEX Assistant.

3.1 Workflow 1: Core Conversational Agent Loop

Purpose: To handle the main conversational interactions with the merchant, answer questions, provide proactive insights, generate recommendations, and utilize available tools dynamically.

Trigger: Initiated by user input (text or button clicks) via the Frontend Interface (4.6).

Execution Flow: This workflow operates as an iterative loop orchestrated by the LangGraph StateGraph and centered around the Conversational Planner Agent.

1. **Input & State:** The user's input is received by the Application Layer and passed to the LangGraph workflow along with the current session's configuration (referencing the state saved by the Checkpointer). The Planner Agent Node accesses the latest user message and the full conversation history/state via the Checkpointer.
2. **Reasoning & Planning:** The Planner Agent LLM analyzes the input within the context of the current state (including memory). It determines the next logical action needed – this could be calling a specific tool, asking the human for clarification, or generating a final response.
3. **Action Delegation (via Edges):** Based on the Planner Agent's decision stored in the state:
    a. **Tool Call:** A conditional edge routes execution to the appropriate Tool Execution Node. The tool node executes its specific Python function using parameters provided by the Planner Agent via the state. The tool's result is returned and updates the state.
    b. **Human Input:** A conditional edge routes to the HITL Mechanism. The Planner Agent formulates the question (stored in the state), and the graph pauses using `interrupt()`. The Application Layer handles displaying the question and resuming the graph with the user's answer via `Command`, updating the state.
    c. **Direct Response:** If no tool or human input is needed, the Planner Agent generates the response content. Execution typically proceeds to a formatting node or directly signals the end.

4. **Looping/State Update:** After a tool execution or human input cycle, edges typically route execution *back* to the Planner Agent Node. The agent re-evaluates the situation based on the *updated state* (now containing the new tool result or human answer).
5. **Completion:** The loop continues until the Planner Agent determines the user's request is fulfilled or the conversation concludes. It generates the final response content.
6. **Output:** The final response is formatted and passed back to the Application Layer, which displays it to the user via the Frontend Interface.

Example Scenario: User asks, "Show me last week's sales trend" The Planner Agent might call the Data Query Tool to get weekly sales figures, then receive the result, then call the Chart Generation Tool with that data, receive the chart, and finally formulate a response.

3.2 Workflow 2: AI-Assisted Data Ingestion Pipeline

Purpose: To provide a structured process for merchants to upload data files (CSV, Excel, potentially images), have the system attempt to understand the data, validate it (with human help if needed), format it, and save it to a designated database.

Trigger: Initiated by a user action in the specific "Data Upload" section of the Frontend Interface , which passes the uploaded file reference to the backend.

Execution Flow: This workflow operates as a more defined pipeline, orchestrated by LangGraph with key decision points driven by the Data Structure Analyzer LLM and potential human input.

1. **Data Loading:** The workflow begins execution at the Data Loader Node. This node identifies the file type and uses the appropriate method to read the data, updating the state. Errors during loading are flagged.
2. **AI Analysis:** Execution proceeds to an **AI Analysis Node**. This node contains conditional logic based on `data_source_type` from the state.
    a. If tabular or image_table: It calls a Text LLM (the Data Structure Analyzer) with the loaded_data_sample and target DB schema. The LLM infers column mappings, assesses confidence for tabular data, and generates clarification

questions about column meanings/types if needed. Updates state with mapping, confidence, questions.

   b. If image_visual: It calls a Multimodal LLM (VQA) with the prepared_image_data and a specific prompt asking for insights/data extraction from the chart. It then likely calls a Text LLM to interpret the VQA response, assess confidence in the interpretation, and generate clarification questions about the visual interpretation (e.g., axis meaning, units) if needed. Updates state with extracted insights/points, confidence, questions.

3. **Confidence Check (Conditional Edge):** Logic checks the confidence score and presence of clarification questions in the state.
   a. **If High Confidence & No Questions:** Execution proceeds directly to the Data Formatting Node.
   b. **If Low Confidence or Questions Exist:** Execution proceeds to the HITL Mechanism.
4. **Human Clarification (HITL Path):**
   a. The HITL node pauses the graph via `interrupt()`.
   b. The Application Layer presents the clarification questions (generated by the Analyzer LLM) to the user.
   c. The user provides answers/confirmations.
   d. The Application Layer resumes the graph, updating the state with the human's clarifications.
   e. The flow typically returns to either the Data Structure Analyzer (to re-confirm understanding with the new info) or directly to the Data Formatting Node if the clarification provided a complete, validated mapping.
5. **Data Formatting:** Data Formatting Node runs after understanding is validated (high confidence or post-HITL). It checks the data_source_type and applies different logic:
   a. If tabular or image_table origin: Accesses the full source data using the file path. Applies the validated column mapping. Performs cleaning, type conversions for the target tabular database schema. Processes in chunks if needed. Updates state with formatted tabular data.
   b. If image_visual origin: Takes the validated extracted insights or data points from the state. Formats them according to a potentially different target

structure (e.g., an insights table schema, a simple key-value data table schema). Updates state with formatted insight/point data.

6. **Database Saving:** The Database Saver Node takes the formatted data from the state and executes database insertion operations. It updates the state with success or failure status.
7. **Outcome Reporting:** A final node formats a message based on the success/failure status from the previous step and updates the `final_response` in the state.
8. **Completion & Output:** The graph ends. The Application Layer retrieves the `final_response` and displays the outcome (e.g., "Data imported successfully," or "Import failed: [Reason]") to the user via the Frontend Interface.

Example Scenario: User uploads an Excel file. The Loader reads it. The Analyzer LLM is highly confident. The Router skips HITL. The Formatter cleans and structures the data. The Saver inserts it into the database. The user sees a success message. *Alternatively*, the Analyzer is unsure about a column named 'Amt', triggers HITL, user clarifies it means 'total_revenue', graph resumes, formatting uses the clarification, data is saved.

Personalization Strategies

Our system is designed to tailor the chat experience uniquely to each merchant, ensuring that insights, recommendations, and interactions are **contextually relevant**, **data-safe**, and **user-aware**.

1. Integrating Merchant-Provided Data for Deeper Insights

Beyond analyzing data generated on the Grab platform, the MEX Assistant empowers merchants by enabling them to upload their own datasets (via the AI-Assisted Data Ingestion workflow, supporting formats like CSV/Excel). This unlocks a new level of personalization and insight by incorporating information potentially unique to the merchant's broader operation, such as:

- **Offline Sales Records:** Data from walk-in customers or non-Grab transactions.
- **Multi-Platform Data:** Sales or performance metrics from other delivery platforms (if tracked by the merchant).
- **Cost of Goods Sold (COGS):** Specific ingredient or item costs provided by the merchant.
- **Inventory Levels:** Custom inventory tracking spreadsheets.

By analyzing this merchant-provided data (once successfully ingested and stored), the assistant can offer significantly more valuable and holistic insights:

- **Cross-Channel Performance:** "Your `Nasi Lemak` sells well both on Grab and offline, but the average spend per customer is higher offline according to your uploaded sheet. Consider bundling it with a drink on Grab?"
- **True Profitability:** (Requires COGS data) "Based on the cost data you provided, the 'Fish Burger' has a lower profit margin despite high sales volume. Focus promotions on the higher-margin 'Chicken Burger'?"
- **Targeted Operational Advice:** (Requires relevant data) "Your uploaded inventory sheet suggests you might be overstocked on Item X compared to its combined online/offline sales rate. Adjust your next order?"

This integration allows the MEX Assistant to move beyond platform-specific analysis towards understanding and advising on the merchant's **entire business context**, providing truly personalized strategic value derived from their own unique data.

2. Hyperlocal Awareness

Our MEX Assistant aims to be more than just a data reporter; it strives to be a proactive partner by deeply understanding both the merchant's individual business and the dynamic environment it operates within. Our core personalization strategies focus on:

- **Local Event Monitoring:** We integrate (or simulate for prototype) awareness of significant local happenings near the merchant's outlet – concerts (like events at nearby Bukit Jalil), community bazaars (e.g., Puchong Jaya weekend markets), school competitions, or even major road closures.
  - Example Insight: "Heads up! There's a large futsal competition at [Nearby Sports Centre] this Sunday afternoon. Expect potential increase in demand for drinks and quick snacks in your area."
  - Value: Enables proactive staffing adjustments, inventory planning, and targeted promotions around these events.
- **Platform Benchmarking:** The assistant analyzes performance trends *relative to anonymized, aggregated data from similar Grab merchants* operating nearby. This provides a crucial competitive context *within the Grab ecosystem*.
  - *Example Insight:* "Your average order value (AOV) saw a 5% increase last week, putting you slightly above the average AOV for similar fast-food outlets in the Puchong area on Grab."
  - *Example Insight:* "While your overall sales grew, your growth during the dinner peak was slightly slower than the average for nearby competitors on the platform last month."
  - *Value:* Helps merchants understand their relative performance and identify areas where they excel or could improve compared to peers on the platform.

3. Personalized Opportunity

Beyond standard reporting, the assistant analyzes patterns to uncover unique growth avenues tailored to the individual merchant, including potential market synergies:

- **Internal Synergy (Cross-Selling/Bundling):** Analyzes the merchant's *own* transaction data to identify items frequently purchased together, suggesting data-driven bundle deals or upselling prompts.
  - *Example Insight:* "Your data shows customers buying 'Spicy Chicken Burger' often add 'Onion Rings', but rarely fries. Consider promoting an 'Onion Ring Upgrade' or a specific 'Spicy Burger + Rings Combo'?"
- **Market Demand Alignment:** Compares the merchant's popular items and sales times against *aggregated, anonymized* demand patterns and keyword trends seen across relevant categories in their zone.

    - *Example Insight:* "Keyword searches for 'late-night snacks Puchong' are high, but your sales dip significantly after 9 PM compared to the zone average. Potential opportunity to extend hours or offer a late-night menu?"
- **Collaboration Potential Identification (*Experimental/Insight-Focused*):** By analyzing broader, anonymized market basket data or complementary category trends in the local area, the assistant *may identify potential demand synergies* with other *types* of nearby businesses.
    - *Example Insight:* "Analysis suggests a pattern in this area: customers ordering meals from outlets like yours frequently place separate orders for specialty coffee from nearby cafes within the next 30 minutes, especially weekday mornings. This indicates a potential shared customer base interested in both meal and premium coffee options."
    - **Note:** This feature focuses on identifying *market patterns* for strategic awareness. The assistant does *not* access other specific merchants' private data or directly facilitate collaborations, but highlights potential overlaps in customer demand within the local ecosystem.
- **Value:** Provides concrete, data-backed suggestions for internal optimization (bundles), aligning with local demand (market timing/keywords), and offers strategic awareness of potential complementary customer behaviours in the vicinity.

Data Utilization

Our solution goes beyond typical query handling — it transforms raw CSV data into **merchant-specific insights**, acting like a personalized **Data Storyteller**. Here's how we bring that to life:

1. Data Ingestion

We integrate

1. merchant-related CSVs:
    - merchant.csv
    - items.csv
    - keywords.csv
    - transaction_data.csv
    - transaction_items.csv
2. User own uploaded data

These are dynamically parsed at runtime. Headers are auto-cleaned and standardized, enabling structured referencing across the pipeline.

2. Data Mapping & Schema Awareness

We use a dynamic function called build_data_description() to generate a real-time schema snapshot, e.g.:

transaction_data:

- order_id
- order_time
- merchant_id
- ...

This enables the AI to "speak the language" of the merchant's data, reducing hallucination and ensuring precise understanding.

3. Intent Translation: From Language to Logic

User input is converted into two synchronized outputs:

- Pandas query — used for actual data computation
- Metropolib logic query — expressing the same logic semantically

These are generated using the build_input_prompt() function under Dual Query Mode to ensure:

- Query clarity
- Explainable logic
- Interoperability with future API/data warehouse integrations

4. Reasoning Logs = Transparent AI

Each AI interaction is logged using:

python

CopyEdit

save_reasoning_log(gpt_output, merchant_id)

This creates a timestamped audit trail for:

- Debugging data behavior
- Reviewing AI decision-making
- Building trust for merchant empowerment

5. Contextual Memory = Chat That Feels Smart

With up to 20-turn contextual memory, our AI can:

- Understand follow-ups like “What about last week?”
- Remember prior queries
- Provide continuity in conversations

This makes it feel more like a **data-literate teammate** than a one-off chatbot.

6. Merchant Isolation by Design

All queries are scoped to the current merchant_id, enforced in:

- Prompt generation
- Query filtering
- Session initialization

This ensures **data privacy** and **personalization** without cross-merchant data leakage.

7. Data Storyteller Mode

We don’t just return raw outputs — we explain the “why” behind every suggestion.

Example:

“We noticed your snack items peak around 3 PM — perhaps offer a tea-time combo?”

The reasoning tells a **mini story**, making analytics accessible to non-technical users.

8. Bias Awareness & Fairness *(Future Scope)*

We aim to identify:

- Biased sales trends (e.g., under-promotion of certain categories)
- Outliers that may skew insights
- Representation gaps in merchant or item data

This helps ensure inclusive economic empowerment, not just data-driven optimization.