Applied Generative AI Specialisation

Capstone Problem Statement

Banking Customer Support AI Agent using Multi-Agent Architecture

Problem scenario:

Modern digital banking platforms handle a high volume of customer service interactions, often through fragmented systems that struggle to personalize responses or provide timely status updates. The need for scalable, intelligent support systems has led to the emergence of AI-driven agents capable of parsing user sentiment, managing service records, and handling real-time queries.

This project explores the development of a **multi-agent GenAI system** tailored for banking customer support workflows. The goal is to reduce manual effort, enhance customer satisfaction, and ensure timely response to support-related feedback and queries.

Project objective:

The objective is to design and implement a **multi-agent AI assistant** that handles:

This model aims to:

- **Classification** of incoming user messages into feedback (positive or negative) or queries
- **Personalized responses** based on classification and user sentiment
- **Ticket tracking and updates** through integration with a support database

Steps to follow:

This capstone is divided into two parts: Agent Design & Workflow Implementation and LLMOps with Evaluation & UI:

Part 1: Multi-Agent Design and Execution Logic

1. Classifier Agent

- Input: Unstructured user message (e.g., “Thanks for resolving my credit card issue.”)
- Task:
  - Categorize the input into one of:
  - Positive Feedback
  - Negative Feedback
  - Query
- Route to the appropriate downstream agent based on classification

2. Feedback Handler Agent

Trigger: Activated when input is classified as feedback.

- Positive Feedback:
  - Generate a warm, personalized thank-you message using a language model
  - Format: Thank you for your kind words, [CustomerName]! We’re delighted to assist you.
- Negative Feedback:
- Generate a new unique 6-digit ticket number
- Insert a new unresolved ticket into a support database (table: support_tickets)
- Return an empathetic message with the generated ticket number
- Format: “We apologize for the inconvenience. A new ticket #[TicketNumber] has been generated, and our team will follow up shortly.”

3. Query Handler Agent

Trigger: Activated when input is classified as a query.

- Task:
- Extract ticket number from user message
- Query the support_tickets database

- Return ticket status to user
- Format: “Your ticket #[TicketNumber] is currently marked as: [Status].”

4. Sample Use Case Flow (Agent Coordination)

Example 1:

User Input: “Thanks for sorting out my net banking login issue.”
Agent Path: → Classifier → Positive Feedback Handler
Response: “Thank you for your kind words! We’re happy to support you.”

Example 2:

User Input: “My debit card replacement still hasn’t arrived.”
Agent Path: → Classifier → Negative Feedback Handler
Response: “We apologize for the inconvenience. A new ticket #784521 has been created. Our support team will look into this promptly.”

Example 3:

User Input: “Could you check the status of ticket 650932?”
Agent Path: → Classifier → Query Handler
Response: “Your ticket #650932 is currently marked as: Resolved.”

Part 2: LLMOps (Evaluation, Logging, and Interface Design)

7. Model Evaluation

- Assess the quality of generated responses across the agents (feedback accuracy, empathy level, clarity of status updates)
- Use QA-based scoring and test case coverage for classification logic
- Evaluate agent routing success rate

8. Streamlit UI Design

Build an interactive dashboard to:

- Accept user input and simulate agent routing
- Display classification, response, and database interaction
- View historical queries and logs
- Test scenarios for each agent role

9. Logs and Debugging View

- Show prompt traces, classification output, and ticket actions
- Maintain logs of ticket IDs, agent success/failure rate
- Optionally integrate user feedback into agent improvement loop