Project Synapse

Agentic Last-Mile Coordinator

Project Description

The Problem

Last-mile delivery is plagued by unpredictable, real-time disruptions that rule-based systems cannot handle. Issues like sudden road closures, incorrect addresses, or unavailable merchants create complex problems that require human-like reasoning to solve, leading to delays and operational inefficiency.

The Vision

An autonomous AI agent that acts as an intelligent coordinator. When a disruption occurs, the agent doesn't just flag an error; it actively reasons about the situation, uses a set of digital tools to gather information, and formulates a coherent, multi-step plan to resolve the issue.

The Challenge

Build a ML Model powered agent or LLM powered agent that can autonomously resolve complex delivery scenarios described in natural language. You will provide the agent with a set of pre-defined, simulated "tools" that mimic real-world logistics APIs (e.g., `check_traffic()`, `get_merchant_status()`). The agent must intelligently decide which tools to use to solve the problem, showing its reasoning process at each step.

Expected Outcomes

1. A functional proof-of-concept (e.g., a command-line application) that accepts a disruption scenario as input.
2. A transparent output of the agent's *"chain of thought"*, showing its reasoning, the actions (tools) it takes, and the observations it makes.
3. The agent must successfully devise a logical plan to resolve at least two distinct and complex disruption scenarios.
4. A well-documented codebase explaining the agent's design and the prompt engineering strategies used.

Potential Technical Skills and Stack

1. **Core:** Programming, understanding of LLM concepts and Prompt Engineering.
2. **Frameworks:** Hands-on experience with an agentic framework like LangChain or LangGraph.
3. **APIs:** Access to an LLM provider's API.

Sample Use Cases in Grab Ecosystem

A solution from Project Synapse could be deployed across various Grab services to create a more resilient and intelligent logistics network.

1. **GrabFood & GrabMart**
    a. **Scenario: Overloaded Restaurant.** An order is placed, but the agent's `get_merchant_status()` tool detects a 40-minute kitchen prep time.
        i. **Agent's Action Plan:**
            - **Proactively Notify Customer:** Immediately use a `notify_customer()` tool to inform them of the long wait and offer a small voucher for the inconvenience.
            - **Optimize Driver Time:** Use a `re-route_driver()` tool to assign the driver to a short, nearby delivery while the food is being prepared, minimizing their idle time.
            - **Suggest Alternatives:** If the delay is critical, use `get_nearby_merchants()` to find a similar restaurant with a shorter wait time and propose it to the customer.
    b. **Scenario: Damaged Packaging Dispute**. At the customer's doorstep, a dispute arises over a spilled drink. It is unclear whether the cause is poor merchant packaging or driver mishandling , creating a real-time conflict that risks a poor review for the driver and a frustrating support experience for the customer.
        i. Agent's Action Plan:
            - **Initiate Real-Time Mediation:** The driver or customer triggers an "At-the-Door Resolution" flow in the app. The agent uses `initiate_mediation_flow()` to open a synchronized interface on both parties' devices, pausing the order completion.
            - **Guide Evidence Collection:** The agent deploys a `collect_evidence()` tool, which prompts both the driver and customer to take photos of the damage and answer a brief, dynamic questionnaire (e.g., Driver: "Was the bag sealed by the merchant?", Customer: "Was the seal intact upon handover?").

- **Adjudicate and Resolve:** The agent's `analyze_evidence()` tool processes the inputs. If data indicates clear merchant fault (e.g., bag was sealed, seal was intact), the agent executes a resolution:
    - It uses `issue_instant_refund()` to compensate the customer.
    - It uses `exonerate_driver()` to clear the driver of fault.
    - It uses `log_merchant_packaging_feedback()` to send an evidence-backed report to the merchant, enabling them to improve their packaging.

c. **Communicate and Close:** The agent uses notify_resolution() to inform both parties of the outcome, allowing the driver to complete the trip without penalty and providing the customer with an instant solution.

2. **GrabExpress**
    a. **Scenario: Recipient Unavailable.** A delivery partner arrives at the destination, but the recipient is not available to receive a valuable package.
    b. **Agent's Action Plan:**
        i. **Initiate Contact:** Use a `contact_recipient_via_chat()` tool with automated prompts asking for instructions.
        ii. **Evaluate Options:** Based on the response, the agent can decide its next action. If the user gives permission, use a `suggest_safe_drop_off()` tool (e.g., "Can I leave it with the building concierge?").
        iii. **Find Alternatives:** If no safe drop-off is possible, use a `find_nearby_locker()` tool to suggest a secure parcel locker as an alternative delivery point.
3. **GrabCar**
    a. **Scenario: Sudden Major Traffic Obstruction.** A passenger is on an urgent trip (e.g., to the airport). The agent's `check_traffic()` tool detects a new, major accident along the planned route.
    b. **Agent's Action Plan:**
        i. **Re-calculate & Inform:** Immediately use a `calculate_alternative_route()` tool to find the next fastest path and use `notify_passenger_and_driver()` to communicate the new route and updated ETA.
        ii. **Provide Context:** Use a hypothetical `check_flight_status()` tool. If the passenger's flight is also delayed, the agent can relay this information, reducing passenger anxiety.