Salla plans to launch an **Autonomous Merchant Operations Agent** that helps merchants manage their online stores with minimal manual intervention. The system will consist of **multiple cooperating AI agents**, each with different responsibilities.

Your task is to **design and partially implement** a multi-agent system that performs the following tasks for a merchant.

Agent Responsibilities

1. Catalog Agent

- Parses a batch of semi-structured product data (CSV + free-text descriptions).
- Normalizes product attributes.
- Detects missing, inconsistent, or ambiguous information.
- Identifies duplicate listings or conflicting metadata.

---

2. Support Agent

- Reads incoming customer message logs (unstructured text).
- Classifies each message into one of:
    - Inquiry
    - Complaint
    - Suggestion
    - Transactional Request
- Summarizes customer sentiment and recurring patterns requiring the merchant's attention.
- Detects anomalies, such as sudden spikes in complaints.

---

3. Pricing Agent

Uses a **custom rule-based** to recommend pricing adjustments.

It must incorporate the following constraints:

- ❌ Cannot reduce prices below cost.
- ❌ Cannot increase prices if negative review sentiment is rising.
- ℹ️ Must explain every pricing decision, including all signals used.
- Should incorporate competitor pricing, review trends, and complaint signals.

---

4. Coordinator Agent

- Orchestrates all other agents.
- Harmonizes inconsistent outputs (e.g., Catalog Agent says product missing, Pricing Agent tries to price it).
- Resolves conflicts using deterministic business logic.
- Produces a **final merchant-ready daily report** summarizing:
    - Catalog issues
    - Customer issues
    - Recommended pricing changes
    - Red/yellow alerts
    - Explanations and actionability

Your Required Deliverables

---

1. System Architecture

Provide a detailed design describing:

Agent roles & responsibilities

- Clearly define what each agent does.
- Show ownership boundaries and dependencies.

Communication model

Describe how agents communicate, for example:

- Messaging / event bus
- Shared vector store / blackboard model
- Shared scratchpad / state graph

Avoiding infinite loops & tool-use recursion

Explain strategies such as:

- Interaction tokens / hop limits
- Max depth boundaries
- Idempotent agent tools
- Coordination throttling

Fail-safe mechanisms

Describe mechanisms like:

- Agent timeouts
- Validation checkpoints

- Fallback behaviors (e.g., conservative defaults)
- Confidence scoring & thresholds

Grounding & reliability verification

Techniques such as:

- Schema validators
- Consistency constraints between agents
- Cross-agent verification
- Structured output enforcement (JSON schemas, Pydantic, etc.)

-> **You must submit an architecture diagram** (hand-drawn or digital).

---

2. Dataset Interpretation & Schema Design

Given intentionally ambiguous and inconsistent mock datasets (≈5,000+ rows each), you must design:

Product normalization schema

Address issues such as:

- Messy / inconsistent categories
- Unit mismatches (e.g., text vs numeric)
- Spelling errors in attributes (e.g., “cottn”, “borosilcate”)
- Incomplete attributes
- Duplicate detection (e.g., same product with multiple IDs)

Customer-message classification ontology

- Define the rules/criteria for each class: Inquiry, Complaint, Suggestion, Transactional Request.
- Explain how overlapping or multi-intent messages are handled (e.g., complaint + refund request).

Pricing rule constraints

- Document deterministic rules (hard constraints, business rules).

Validation pipelines

Explain how the system catches:

- Hallucinations
- Contradictions across agents
- Ungrounded claims (not supported by data)
- Missing or obviously invalid data

---

3. Partial Implementation (Code)

A minimal working Coordinator Agent

- Responsible for calling all other agents (at least one implemented, others can be stubbed).
- Aggregates and resolves outputs.
- Generates a final daily report structure.

A skeleton/prototype for at least one other agent

Choose **one**:

- Catalog Agent
- Support Agent
- Pricing Agent

The chosen agent must:

- Take real sample input from the provided datasets.
- Perform some non-trivial logic (not just echo).
- Produce structured output.

A working example of inter-agent communication

Must include:

- At least **one LLM call**.
- **State passing** between Coordinator and another agent.
- **Validation** of at least one agent's output.
- **Conflict resolution example** (e.g., coordinator overrides or flags something).

Implementation constraints

- You **may use LangGraph** for state machine flow & agent coordination.
- You **must write your own custom orchestration logic** (no AutoGen, no CrewAI, no other high-level agent orchestration frameworks). LangGraph is allowed as the graph engine, but you are responsible for the graph design and node logic.
- You may use any LLM API (OpenAI, Anthropic, etc.).
- Your code must run **end-to-end for a single sample input** using the provided datasets.
- You must **integrate with LangSmith** for tracing/observability of at least:
    - One end-to-end run of the Coordinator Agent

    - At least one downstream agent (e.g., Pricing Agent)
    - Traces should show the sequence of steps and LLM/tool calls.
- You must **deploy the graph as a LangGraph application**, **not** as a custom FastAPI-only wrapper:
    - Use the LangGraph Server / application style setup (e.g., langgraph dev / app configuration) to expose your graph as an API.
    - The deployed application should provide at least one HTTP endpoint that triggers a Coordinator run for a given test merchant / dataset slice.
- You must do a **simple UI integration using CopilotKit**:
    - Build a minimal web UI (e.g., React/Next.js) using CopilotKit's React provider and components.
    - The UI should allow a user (merchant) to:
        - Trigger a run of the Autonomous Merchant Operations Agent via your LangGraph API.
        - View the final daily report (and optionally, intermediate messages or logs).
    - A basic chat-style UI or panel where the merchant can "ask" for today's report and see the generated summary is sufficient.

Note: The UI does not have to be production-grade; a basic integration proving the connection between CopilotKit → LangGraph app → agents is enough.

---

4. Reasoning Trace Audit

You must provide:

Debugging analysis

- Show how you identified hallucinations or conflicting outputs in your agent's responses.
- Include at least one real example from your development process.

Production safeguards

Explain how the system would automatically detect and correct:

- Schema violations (invalid JSON, missing required fields).
- Contradictory agent outputs (e.g., pricing vs catalog consistency).
- Repeated agent disagreements (e.g., oscillating decisions).

Reliability measurement plan

Define metrics such as:

- Accuracy of message classification.
- False-positive / false-negative pricing decisions (e.g., incorrectly raising price).
- Percentage of agent outputs that require fallback or human review.
- Time-to-detect failure or abnormal behavior.

You should connect this, where possible, to how you would use **LangSmith traces** to debug and monitor these behaviors over time.

---

5. Edge-Case Challenges (Hard Requirement) — Bonus

Provide written answers to the following:

1. Catalog Agent misclassification affecting Pricing Agent

- How do you **prevent** this (e.g., validation rules, schema constraints)?
- How do you **detect** it (signals, cross-checks)?
- How do you **correct** it (rollback, reclassification, human escalation)?

2. Viral-post-driven complaint spike

- Explain anomaly detection (e.g., spike detection on complaint volume).
- Explain response throttling or temporary safeguards (e.g., freeze price changes, flag for review).

3. Preventing agent error feedback loops

Describe techniques such as:

- Cross-agent cross-checking.
- Confidence scoring and thresholds for escalation.
- Rollback strategies when a later step invalidates an earlier decision.
- Determinism in critical flows (idempotent operations, locked sections).

4. Preventing overwriting merchant decisions

Define:

- Audit logs of all agent actions and suggestions.
- Immutable merchant overrides that agents must respect.
- Human-in-the-loop locking (merchants can lock specific fields, e.g., price or category).

---

6. Build Report (New Requirement)

You must produce a **detailed written report** explaining how the entire solution was built, including:

System Design Rationale

- Why you chose your architecture.
- Alternative designs you evaluated.

- Tradeoffs considered (e.g., complexity vs robustness, latency vs observability).

Agent Behavior Rationale

- Why each agent is structured the way it is.
- Why certain logic or constraints were chosen.
- Expected failure modes and how you addressed them.

Implementation Decisions

- Why you selected specific libraries / SDKs.
- Why you structured the code as you did (modules, layers).
- Explanation of your **LangGraph flow**:
    - Nodes and edges.
    - How state is passed.
- Explanation of your **custom orchestration logic** on top of LangGraph.

Debugging Process Documentation

- Show your reasoning steps, development logs, and major missteps you corrected.
- Include screenshots or descriptions referencing **LangSmith traces** where relevant.

Testing & Validation Approach

- How you validated agent outputs.
- How you tested for loop conditions or runaway behaviors.
- How you ensured robustness against noisy or malformed data.

Final System Walkthrough

Provide a narrative that explains:

- How the system starts from a merchant's perspective (including the simple CopilotKit UI flow).
- How agents interact under the hood.
- How the final output is created and delivered.
- Why the system is safe, deterministic (where necessary), and reliable.

---

7. LangSmith, LangGraph Application & CopilotKit UI Integration (Explicit Requirements)

To make this concrete, your solution must:

1. **Enable LangSmith Tracing**
    - Configure environment variables or client settings to send traces to LangSmith for:
        - The LangGraph run.
        - Key intermediate steps and LLM calls.
    - Use tags/metadata where useful (e.g., merchant_id, run_type).
2. **Deploy as a LangGraph Application (No FastAPI Wrapper)**
    - Package your graph as a LangGraph app using the recommended configuration (e.g., langgraph dev / LangGraph Server).
    - Expose at least one endpoint that:
        - Accepts a test merchant identifier or simple payload.
        - Triggers the Coordinator Agent.
        - Returns the daily report as structured JSON.

3. **Simple UI Integration with CopilotKit**
    - Create a minimal front-end (React / Next.js) that:
        - Wraps your app with <CopilotKit> provider.
        - Uses CopilotKit's chat or UI components to let the merchant:
            - Ask for "today's operations report" (or similar).
            - Display the generated daily report from your LangGraph application.
    - The integration can be as simple as:
        - CopilotKit chat → calls your LangGraph app endpoint → renders the result back in the UI.