Post-Mortem Document

Agentic AI System for Multi-Step Tasks

---

1. Scaling Issue Encountered

Issue: Planner Agent Bottleneck & API Rate Limiting

What happened:
The **Task Planner Agent** became a **scaling bottleneck** when multiple tasks were submitted concurrently. Since the planner relies on a single LLM call per task, the system quickly hit **Gemini API rate and quota limits**, resulting in:

- HTTP 429 (Resource Exhausted) errors
- Invalid or empty LLM responses
- Task pipeline stalling before downstream agents were triggered

Why this is a scaling issue:

- All tasks must pass through the Planner Agent first
- Planner throughput is limited by LLM API capacity
- Horizontal scaling of other agents does not help if planning is blocked

Impact:

- Reduced system throughput
- Increased latency for all tasks
- Redis queues accumulated unprocessed jobs

Mitigation Used:

- Introduced `MOCK_MODE` to allow offline testing
- Paused live LLM calls once quota exhaustion was detected

Future Improvement:

- Cache common task plans
- Batch planner requests
- Introduce multiple planner instances with rate-aware scheduling

---

2. Design Decision I Would Change

Decision: Planner Agent Strictly Dependent on Live LLM Output

Original decision:
The Planner Agent was designed to **always rely on live LLM responses** to generate task steps.

Why this was suboptimal:

- System became unavailable when the LLM failed
- No fallback plan generation strategy existed
- Planner failure cascaded to all other agents

What I would change:
I would redesign the Planner to support **multi-tier planning**:

1. Cached plans for common task types
2. Rule-based fallback planning
3. LLM-based planning only when required

Benefit of the change:

- Improved system resilience
- Reduced LLM dependency
- Faster planning for repeat tasks

---

3. Trade-offs Made During Development

Trade-off 1: Simplicity vs. Throughput

- **Chosen:** Sequential task execution
- **Sacrificed:** Maximum parallelism

This decision improved clarity and debuggability but limited performance under heavy load.

Trade-off 2: Redis Simplicity vs. Advanced Messaging Guarantees

- **Chosen:** Redis lists (`BLPOP/RPUSH`)
- **Sacrificed:** Message acknowledgements, exactly-once delivery

This kept the system lightweight but requires additional logic for large-scale production use.

Trade-off 3: Mock Mode vs. Live LLM Accuracy

- **Chosen:** Mock Mode for development and demo
- **Sacrificed:** Real-world reasoning quality

This allowed reliable demos and testing despite API quota limitations, at the cost of real inference.