Task 2 – Implementation Explanation: LLM-Powered Infrastructure Optimization Agent

# 1. Objective

The goal of Task 2 was to design and implement a functional LLM-powered agent that ingests cloud infrastructure data and provides cost-optimization recommendations. The solution was required to:   
- Parse infrastructure data in JSON format  
- Use a language model (LLM) to analyze compute patterns  
- Output structured optimization suggestions  
- Handle malformed or failed LLM outputs gracefully via fallback logic  
- Pass test cases covering over-, under-, and mixed-provisioned resource scenarios

# 2. Architecture Overview

The solution is architected with modularity and extensibility in mind. The system is composed of:  
  
• Agent Layer:  
- ComputeAgent processes compute resources (CPU, memory) and builds a prompt.  
- BaseAgent wraps OpenAI’s LLM API and abstracts communication.  
  
• Prompting Strategy:  
- Each input scenario is transformed into a structured prompt template with example schema.  
- The prompt is submitted to GPT-4-turbo.  
  
• Fallback Logic:  
- If LLM fails or returns an invalid response, fallback logic is triggered.  
- It includes downsizing (CPU < 30%) and upsizing (CPU > 80%) heuristics.  
  
• Validation & Output:  
- All outputs adhere to strict Pydantic models.  
- Structured outputs are saved as JSON.

# 3. Testing

The system was tested using Pytest with six fixtures: over-provisioned, under-provisioned, and mixed workloads in both small and large configurations. Tests verify:  
- Input validation  
- Correct fallback behavior  
- JSON output structure  
- Confidence and cost calculation  
All tests passed successfully.

# 4. Error Handling & Robustness

The LLM call is encapsulated in a try-except block. In the event of network failure or API error, an empty fallback response is triggered. The fallback logic guarantees recommendations by analyzing input data heuristically, providing consistency even when the LLM is unavailable.

# 5. Result Summary

The agent produced expected outputs across all three scenarios:  
- Over-Provisioned: Full downsize recommendations and positive savings  
- Under-Provisioned: Upsize suggestions with negative cost impact  
- Mixed: Combination of downsize and upsize with net optimized cost

# 6. Tools & Stack

• Python 3.10+  
• OpenAI GPT-4 API (or fallback if disabled)  
• Pydantic v2 for input/output schema validation  
• Pytest for automated testing  
• python-dotenv for environment management

# 7. Conclusion

This solution meets all functional and non-functional requirements outlined in Task 2. It provides a robust and extensible starting point for SmalBlu's infrastructure optimization product roadmap. It is designed to integrate seamlessly with additional agents, validation rules, human-in-loop review, and production-level infrastructure.