Agent Based Modelling and Agentic Technology CO3

Name: Meghna Suraj Kashyap
Andrew ID: msurajka

Note: I used Claude APIs instead of OpenAI as I had a subscription.
GitHub Code Link: https://github.com/meghna-sk/Financial-Portfolio-Rebalancer

Comparative Analysis
P1: Portfolio 1 `{"AAPL": 0.5, "TSLA": 0.3, "GOOGL": 0.2}`
P2: Portfolio 2 `{"MSFT": 0.25, "NVDA": 0.25, "AMZN": 0.25, "META": 0.25}`

Metric	Claude 3 Ppus	Groq LLaMA3-8B	Groq LLaMA3-70B
Response Accuracy	High Correct, well-reasoned balance and contextual advice	Medium Repeated the same rebalancing loop, some reasoning breakdown	Medium-High Mostly accurate, but Portfolio 2 had logic flaws
Latency	28.65s (P1) 25.05s (P2) Relatively slow, but expected for high quality reasoning from a large model	85.41s (P1) 29.43s (P2) Extremely high in P1 due to looping, otherwise reasonable	4.58s (P1) 10.38s (P2) Fastest overall, well-optimized for tool-based tasks
Tool Selection Efficiency	Very efficient 5 calls, no repetition	Poor 15+ repeated calls, infinite loop triggered	Good 4-6 calls, but misused PortfolioRebalancer in P2
Financial Advice Quality	Excellent Thorough, cautious, personalized	Fair Provided correct actions, but lacked depth and was repeated	Decent Concise with correct trades, but flawed logic for Portfolio 2

Summary of Comparison Table
Claude 3 Opus

- Best overall financial reasoning.
- Used all tools correctly.
- Gave actionable + risk-aware advice.
- Slower response but extremely detailed.

Groq LLaMA3-8B

- Major loop issue with PortfolioRebalancer on Portfolio 1.
- Didn't detect it had already taken action.
- Good price lookups, but limited contextual insight.

Groq LLaMA3-70B

- Fast + structured for Portfolio 1.
- Portfolio 2 logic failed. Tried to rebalance an already balanced portfolio based on prices instead of weights.
- Still used tools smartly, just made poor assumptions.

1. Implementation Details
The AI financial assistant is built using Python and LangChain, structured in a modular and extensible format for ease of development, debugging, and scalability. The assistant uses a tool-augmented LLM agent architecture, allowing it to dynamically choose between multiple tools to fulfill user prompts. Each module is purposefully separated to handle a specific concern within the system.

main.py – Entry point & evaluation logic
This file is responsible for initializing agents, running test cases, and capturing metrics such as response latency and tool usage. It defines:
A **`create_and_run_agent()`** function that selects the correct LLM, initializes a LangChain agent with tools, measures execution time, and logs tool calls.

A **`run_test_cases()`** function that feeds predefined portfolios into the assistant using different LLMs (Claude 3 Opus, Groq LLaMA3-8B, and LLaMA3-70B).

The **`verbose`** mode toggle to observe intermediate thoughts, actions, and tool invocations by the agent.

This is the control center of the application and is intended to be the primary file executed **`(__main__ block)`**.

agents/llm_selector.py – Model selection abstraction
This module defines and returns the correct LLM instance based on user-defined provider and model type. It supports:

Anthropic Claude 3 Opus, initialized using **`ChatAnthropic`** with the **`anthropic_api_key`**.

Groq LLaMA3 models (8B and 70B) using the **`ChatGroq`** class.

The **`get_llm()`** function abstracts this decision and is used throughout the agent creation process.

tools/stock_price.py – Real-time price fetcher

This module defines the **`StockPriceLookup`** tool using the **`Tool()`** class. It uses the **`yfinance`** library to retrieve:

Current market price for a stock symbol

Other relevant metadata (example - daily change)

The tool accepts a stock ticker symbol as input and returns structured price data. Errors are handled such as if the symbol is invalid or unavailable.

It logs its own usage using a call to the shared logging function.

tools/rebalance.py – Portfolio equal-weighting logic
This module defines the `PortfolioRebalancer` tool. It takes as input a JSON-like dictionary string representing the current portfolio allocation (example - `{"AAPL": 0.5, "TSLA": 0.3, "GOOGL": 0.2}`).

It parses the dictionary using `eval()` (with type checking for safety)

Calculates the ideal weight for equal distribution (example - 33.33% for three assets)

Determines the delta per stock and recommends a “Buy”, “Sell”, or “Hold” action accordingly

The output is returned in a user-friendly text format for the LLM to include in its final answer. Usage is logged for tool efficiency tracking.

tools/trend_analysis.py – Market trend summarizer
This module defines the MarketTrendAnalyzer tool. It uses `yfinance` to -

Fetch historical price data for the SPY ETF (proxy for the S&P 500)

Calculate the 5-day return percentage

Estimate annualized volatility based on the standard deviation of returns

The tool returns a concise summary (example - “S&P 500 5-day return: -0.51%, Annualized volatility: 92.4%”) to help the LLM contextualize market risk when offering advice. It also logs each call for analysis.

utils/logger.py – Lightweight internal telemetry
This utility provides a global logger to track which tools were used in each LLM response. It includes:

`log_tool_usage(tool_name)` – Appends tool name to a session log

`get_tool_log()` – Returns the list of tool calls

`reset_tool_log()` – Clears the log for the next run

This mechanism allows tracking of “Tool Selection Efficiency” across different LLMs.
Note: The original assignment description mentioned a `utils/env_loader.py`, but in this version, environment variables are loaded using `python-dotenv` directly at the top of `llm_selector.py`. No standalone `utils` loader file is required.

LangChain Agent Configuration
All tools are passed into the agent via LangChain's **`initialize_agent()`** method using the **`ZERO_SHOT_REACT_DESCRIPTION`** agent type. This enables the LLM to -

Reason through intermediate steps using Thoughts -> Actions -> Observations

Select tools dynamically as needed

Integrate retrieved data into its final answer

The configuration includes -
`temperature=0` for deterministic outputs
Verbose mode to enable logging of intermediate agent steps

2. Challenges Encountered and Solutions
a. LLM Compatibility and API Integration
Initial attempts to integrate Anthropic's Claude via `langchain_community` led to deprecation issues. This was resolved by switching to the updated **`langchain-anthropic`** library and using the appropriate **`ChatAnthropic`** interface. Proper environment variable naming (example - **`ANTHROPIC_API_KEY`**) was essential to avoid runtime errors.

b. Agent Looping in Groq LLaMA3-8B
The Groq 8B model repeatedly triggered the **`PortfolioRebalancer`** tool, causing an infinite loop scenario. This was due to the LLM failing to internalize state changes after receiving observations. The solution involved monitoring tool call counts and introducing agent execution timeouts.

c. Error Handling in Tool Responses
At one point, malformed inputs to the stock price tool caused parsing errors, particularly when symbols were wrapped in quotes. Updating the input sanitization and using consistent dictionary parsing resolved the issue.

d. Tool Selection Validation
LangChain's agent framework expects tools to accept specific argument formats. Ensuring input strings were passed in evaluable dictionary format (**`"{'AAPL': 0.5}"`**) was required to prevent downstream tool errors.

3. Comparative Analysis: LLM Strengths and Weaknesses
Claude 3 Opus
Strengths: Delivered the most comprehensive responses, demonstrating an understanding of market volatility, risk tolerance, and rebalancing implications. It followed multi-step reasoning flawlessly and selected tools efficiently.
Weaknesses: Significantly slower in execution (~25-30 seconds per run), which could impact user experience in real-time applications.

Groq LLaMA3-8B
Strengths: Exceptionally fast in ideal cases and cost-efficient. Produced accurate numeric rebalancing suggestions.
Weaknesses: Suffered from loop repetition in Portfolio 1, repeatedly invoking the same tool without advancing the logic. Limited in reasoning depth and occasionally misunderstood when to terminate.

Groq LLaMA3-70B
Strengths: Struck a strong balance between speed and reasoning. Fastest average latency among all models, without falling into tool usage traps.
Weaknesses: In Portfolio 2, failed to recognize that a portfolio was already balanced. Attempted to rebalance based on raw prices instead of weight proportions, leading to flawed advice.

4. Recommendations by Use Case

Use Case Scenario	Recommended LLM	Reasoning
High-stakes financial advice / client-facing	Claude 3 Opus	Deepest reasoning, clearest communication, best risk assessment
Real-time dashboards / low-latency environments	Groq LLaMA3-70B	Fastest model with solid reasoning in most test cases
Educational or low-cost deployments	Groq LLaMA3-8B	Lightweight and simple; usable if loop control is addressed
Scenarios requiring interpretability + context	Claude 3 Opus	Outperforms others on explanation clarity and tool alignment

Conclusion
This project demonstrates the effectiveness of integrating multiple LLMs within a LangChain agent framework to perform financial portfolio analysis. Claude 3 Opus offers the most complete financial reasoning, while Groq's LLaMA3-70B provides the fastest and most efficient tool usage among open-source LLMs. The assistant's architecture allows for flexible model substitution and real-time tool orchestration, setting the foundation for more advanced AI-driven investment advisors.