Portfolio & Market Intelligence Agent

by **Neelabha Banerjee**

Contents

Objective

To design and implement a multi-agent intelligence system that can:

- Understand natural language investment queries.
- Analyze and explain portfolio performance, holdings, and allocations.
- Augment answers with real-time market intelligence — news, SEC filings, price trends.
- Maintain conversational context and reasoning capability over multiple turns.
- Validate responses for correctness before returning them to the user.

Scope

The system supports:

- Portfolio-related queries (e.g., “What are my holdings?”, “How is MSFT performing?”)
- Market-related queries (e.g., “What’s the latest news on Tesla?”)
- Hybrid queries involving both (e.g., “Will Apple earnings affect my portfolio?”)
- Contextual follow-ups (e.g., “And what about the second stock?”, “Compare them”)
- Automatic detection and fallback for errors.

Portfolio & Market Intelligence Agent

Key Features

- End-to-end multi-agent orchestration using LangGraph
- LLM + Rule-based hybrid architecture for robust classification and planning
- Portfolio + Market data fusion for hybrid reasoning
- Knowledge base to effectively generate accurate responses
- Session memory for context retention
- Session isolation to prevent data le
- Output validation to minimise hallucinations and ensure correctness
- Streamlit UI for user interaction

Design Goals

Natural Query Understanding: Robust classification of portfolio, market, and hybrid queries, including ambiguous follow-ups.

Multi-Agent Orchaestration: Dedicated specialized agents working collaboratively within LangGraph.

Reliable Data Integration: Integration with market APIs and internal portfolio data.

Reasoning Capability: Use of conversation memory and structured data for intelligent answers.

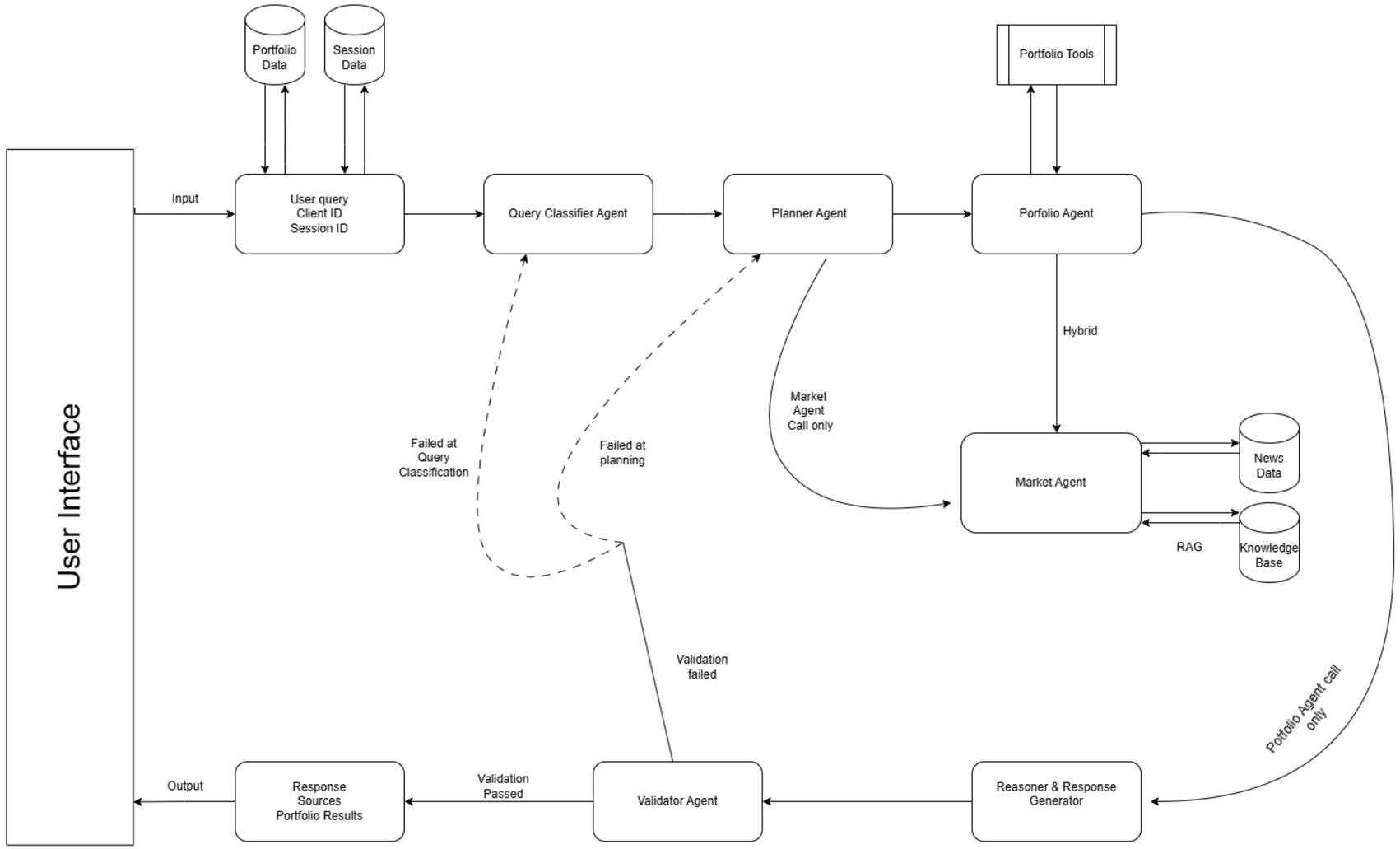
Validation & Safety: Built-in Validator Agent ensures response accuracy and consistency.

Retrieval Augment Generation: Built in knowledge base for the model to use for response generation

Scalibility: Session isolation and architecture are ready for multi-client, concurrent usage.

Portfolio & Market Intelligence Agent

System Architecture



Why this Layout?

Meets objectives

Natural language understanding, portfolio + market analysis, reasoning with memory, validation before answering.

Resilient

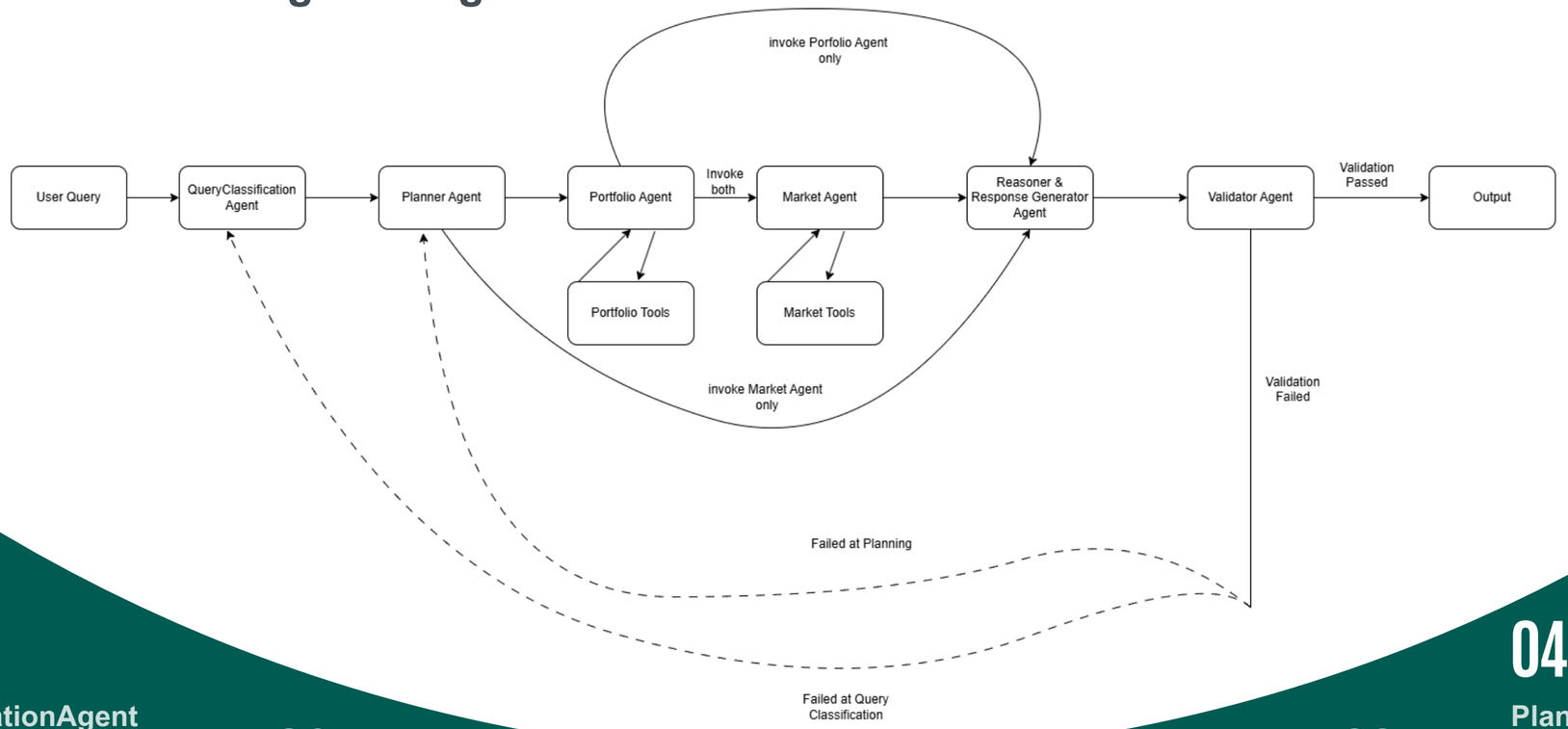
Each node has clear fallbacks; the validator guards the final output and recurses from the exact step where the current iteration failed.

Traceable

Every agentic decision is audited to ensure traceability

Portfolio & Market Intelligence Agent

Data Flow



01

User → QueryClassificationAgent

Input: Raw query
Process: Detect intent (portfolio/market/hybrid/unknown), normalize entities to tickers; fallback to LLM if rules are unsure
Output: {intent, entities} (e.g., portfolio, ["TSLA"])

02

QueryClassificationAgent → PlannerAgent

Input: Classification result
Process: Select downstream agents and map to functions (e.g., get_returns); handle hybrid routing
Output: execution_plan (agents + function calls)

03

PlannerAgent → PortfolioAgent

Input: Execution plan, entities, session memory
Process: Fetch user's holdings and performance for the tickers; handle missing/invalid symbols gracefully
Output: Structured *portfolio_data* (returns, gains, allocations)

04

PlannerAgent → MarketAgent *(for hybrid/market)*

Input: Execution plan, entities
Process: Retrieve *market_data* (price, recent news, SEC filings) via cache → live APIs
Output: Structured *market_data*

05

PortfolioAgent/MarketAgent → ResponseGenerationAgent

Input: Original query + portfolio_data and/or market_data + conversation history
Process: Generate concise, grounded Markdown response (no hallucinations)
Output: Final user-facing text

06

All Steps → ValidatorAgent *(post-check)*

Input: Workflow log (each agent's input/output)
Process: Verify classification, planning, and grounding; flag errors and specify rerun step if needed
Output: pass / fail (+ failed_agent, reason)

07

Backend → UI & Memory

Input: Final text + structured data
Process: Display response; update session/chat history for follow-ups
Output: Rendered answer + refreshed memory

Knowledge Base Integration

Purpose

Leverage unstructured financial intelligence (e.g., SEC filings, press releases, earnings summaries, news) to enrich responses.

Provide context-rich, explainable outputs beyond structured data.

Responsibilities

- Ingest, clean, chunk, and embed unstructured data.
- Store embeddings with metadata (ticker, date, source).
- Retrieve top relevant snippets based on ticker and query semantics.

Applications

1. **Market Agent:** Enhances structured price/news output with deeper insights from filings.
2. **Response Generator:** Uses snippets for richer, grounded reasoning
3. **Validator Agent:** Cross-checks generated response against retrieved context.

Agentic Roles ↗

❋ Query Classification Agent

Responsibility: Intent + entity extraction
Fallback: If low confidence → LLM classifier

❋ Planner Agent

Responsibility: Build execution plan + tool calls
Fallback: If LLM tools fail → rule-based function mapping

❋ Market Agent

Responsibility: Retrieve prices, headlines, and filings.
Fallback: Cache-first; API on miss; partial returns allowed

❋ Portfolio Agent

Responsibility: Holdings, returns, allocation, comparisons
Fallback: If functions missing → default to holdings+returns

❋ Response Generator Agent

Responsibility: Grounded, concise Markdown answer
Fallback: Templated summary if LLM fails

❋ Validator Agent

Responsibility: Judge workflow correctness; pass/fail
Fallback: Default pass if invalid JSON to avoid blocking

Evaluation

Query Classification Agent ↘

20

Total **Cases**

85%

Overall **Accuracy**

Function Calling Agent ↘

10

Total **Cases**

70%

Overall **Accuracy**

Validator Agent ↘

8

Total **Cases**

64%

Overall **Accuracy**

Portfolio & Market Intelligence Agent

Production Plan

Scalability & Performance

Load Balancing: Round-robin or weighted routing across multiple API keys for better throughput and fault tolerance.

Distributed Infra: Use distributed pods of Qdrant to parallelize retrieval and reduce latency.

Horizontal Scaling: Expose each Agent and Retrieval Engine as stateless services for elastic scaling.

Caching: Cache frequent queries and embeddings to reduce redundant LLM calls.

Observability

Metrics & Dashboards:

- Use Grafana + Prometheus/ OpenTelemetry to track latency, error rates, throughput, and resource utilization.
- Real-time dashboards and alerting for anomalies.

LLM & Agent Monitoring:

- Use LangSmith for tracing, token usage, latency, and decision debugging.
- Identify bottlenecks via trace visualizations.

Production System Architecture

API Layer: Receives incoming user queries, manages sessions, and triggers the LangGraph workflow. Handles authentication and client isolation.

Agent Orchestration Layer: Runs the multi-agent reasoning pipeline. Handles routing, fallbacks, and logging of workflow steps.

Data Layer: Fetches portfolio and market data. Implements hybrid caching to optimize latency and cost.

Model Layer: Performs classification, function calling, reasoning, and response generation. Supports fallbacks to deterministic systems when LLMs fail.

Memory Layer: Maintains conversational state, previous queries, and reasoning context. Enables contextual and multi-turn reasoning.

Logging & Monitoring Layer: Tracks agent decisions, API latencies, and error metrics for debugging and evaluation.

Thank You!

Neelabha Banerjee

18 Oct 2025