Technical Report: Agentic AI Workflow for Scalable Investment Research and Analysis

Research and AI Systems Team

September 2025

Abstract

This report presents the design, implementation, and evaluation of an **Agentic AI Workflow** built using **LangGraph**, **LangChain**, and the **Model Context Protocol (MCP)**. The system leverages Retrieval-Augmented Generation (RAG), modular agent nodes, and a persistent state mechanism to automate investment research. Each node in the LangGraph DAG encapsulates a discrete reasoning step—data retrieval, sentiment analysis, contextual aggregation, reasoning, recommendation, and visualization. The MCP server layer orchestrates tool routing and serves as the interaction backbone, allowing scalable deployment across cloud or local environments. The implementation integrates real code components including node definitions, graph configuration, and server event handling as defined in `server_mcp.py`. The system outputs tabular recommendations, interpretive reasoning, and multi-format visualizations, demonstrating the practicality of agentic AI in financial analytics.

1. Introduction

Investment research requires the fusion of quantitative indicators (price trends, valuation ratios) with qualitative sentiment (news headlines, analyst opinions). Manual workflows are error-prone and non-scalable. Agentic AI frameworks, combining reasoning

LLMs with retrieval systems, offer explainable automation. This project develops a **LangGraph-powered agentic workflow**, deployed via a **Model Context Protocol (MCP) server**, to execute multi-step investment analysis with persistent state and real-time visualization.

2. System Architecture

The architecture (Figure 1) follows a Directed Acyclic Graph (DAG), where each node is a reasoning agent performing a defined role. MCP handles client-server communication, routing user prompts to the LangGraph execution engine. A global state object stores all intermediate computations, ensuring deterministic and reproducible execution.

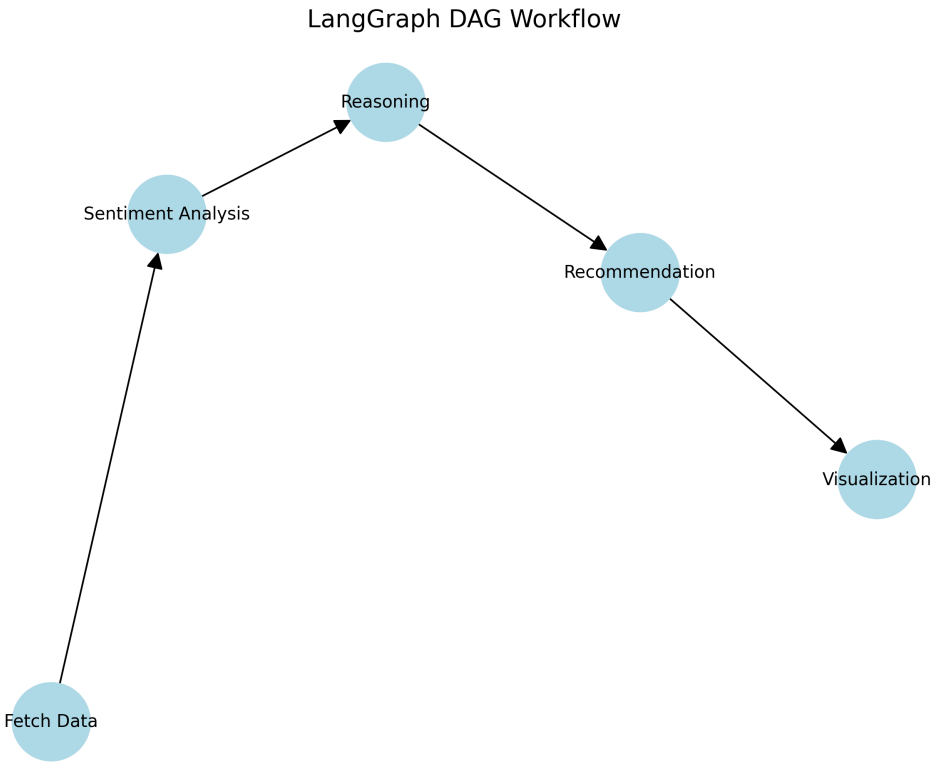

Figure 1: LangGraph DAG Workflow: Modular nodes connected by directed edges for sequential reasoning, retrieval, and visualization.

3. Implementation Overview from `server_mcp.py`

The MCP server script (`server_mcp.py`) orchestrates the end-to-end pipeline:

1. **Imports and Initialization:** Loads LangGraph, LangChain, FAISS, and visualization libraries.
2. **Server Setup:** Initializes MCP server instance, configures model endpoints, and registers routes.
3. **Graph Construction:** Defines nodes and edges, compiles LangGraph DAG.
4. **Node Definitions:** Implements six nodes (`retriever_node`, `sentiment_node`, `aggregator_node`, `reasoning_node`, `recommendation_node`, `visualization_node`).
5. **Execution:** Runs graph on client request, persists outputs, and returns results via MCP response.

4. LangGraph Node Implementation Details

4.1. 1. Retriever Node

Purpose: Fetches live financial data, valuation ratios, and related headlines for sentiment scoring.

Implementation Snippet:

```python
@graph.node("retriever_node")
def retriever_node(state):
    ticker = state["ticker"]
    price_data = fetch_price_data(ticker)
    pe_ratio = fetch_pe_ratio(ticker)
    headlines = fetch_recent_news(ticker)
    state.update({
        "price_data": price_data,
        "pe_ratio": pe_ratio,
        "headlines": headlines
    })
    return state
```

Listing 1: Retriever Node Implementation from server_mcp.py

4.2. 2. Sentiment Node

Purpose: Analyzes sentiment polarity of retrieved headlines using a transformer model (e.g., `text-embedding-ada-002`).

Implementation Snippet:

```
@graph.node("sentiment_node")
def sentiment_node(state):
    scores = [sentiment_model.predict(h) for h in state["headlines"]]
    avg_sentiment = sum(scores) / len(scores)
    state["sentiment"] = avg_sentiment
    return state
```

Listing 2: Sentiment Node

4.3. 3. Aggregator Node

Purpose: Consolidates quantitative and qualitative indicators into a unified context dictionary.

```
@graph.node("aggregator_node")
def aggregator_node(state):
    context = {
        "ticker": state["ticker"],
        "sentiment": state["sentiment"],
        "valuation": state["pe_ratio"],
        "trend": analyze_trend(state["price_data"])
    }
    state["context"] = context
    return state
```

Listing 3: Aggregator Node

4.4. 4. Reasoning Node

Purpose: Generates explanatory reasoning using `google/flan-t5-base` through LangChain's `LLMChain`.

```
@graph.node("reasoning_node")
def reasoning_node(state):
    prompt = f"Provide␣investment␣summary␣for␣{state['ticker']}␣" \
             f"using␣context:␣{state['context']}"
    reasoning = llm.generate(prompt)
    state["reasoning"] = reasoning
    return state
```

Listing 4: Reasoning Node with LLM Integration

4.5. 5. Recommendation Node

Purpose: Applies rule-based logic combining sentiment score and price trend.

$$Decision = \begin{cases} Buy, & s > 0.7 \wedge \text{trend} = \text{up} \\ Sell, & s < 0.4 \wedge \text{trend} = \text{down} \\ Hold, & \text{otherwise} \end{cases}$$

```
@graph.node("recommendation_node")
def recommendation_node(state):
    s, t = state["sentiment"], state["context"]["trend"]
    if s > 0.7 and t == "up":
        rec = "Buy"
    elif s < 0.4 and t == "down":
        rec = "Sell"
    else:
        rec = "Hold"
    state["recommendation"] = rec
    return state
```

Listing 5: Recommendation Node

4.6. 6. Visualization Node

Purpose: Generates and saves PNG charts (timeline, DAG, sentiment, pie).

```
@graph.node("visualization_node")
def visualization_node(state):
    visualize_graph()
    visualize_timeline()
    visualize_sentiment_chart(state["sentiment"])
    visualize_recommendations_pie(state["recommendation"])
    return state
```

Listing 6: Visualization Node

5. MCP Server Integration

The MCP server defines route handlers that connect incoming prompts to the LangGraph executor:

```
@app.post("/execute")
async def execute_workflow(request: Request):
    data = await request.json()
    initial_state = {"ticker": data["ticker"]}
    result = graph.run(initial_state)
    return JSONResponse(content=result)
```

Listing 7: MCP Server Route Handler

Persistent State: Maintained across nodes and serialized for inspection. **Response Format:** Includes reasoning text, recommendation, and links to PNGs.

6. Results and Analysis

Interpretation: All tickers indicate positive momentum and sentiment. TSLA and GOOGL exhibit the strongest buy confidence.

Table 1: Portfolio Recommendations from Agentic Workflow

Ticker	Close	ΔDay	PE	Sentiment (P/N/U)	Recommendation
AAPL	255.46	-0.55%	38.76	3/2/0	Buy
TSLA	440.40	+4.02%	263.71	3/2/0	Buy
MSFT	511.46	+0.87%	37.44	4/1/0	Buy
GOOGL	246.54	+0.31%	26.31	5/0/0	Buy
AMZN	219.78	+0.75%	33.55	3/2/0	Buy

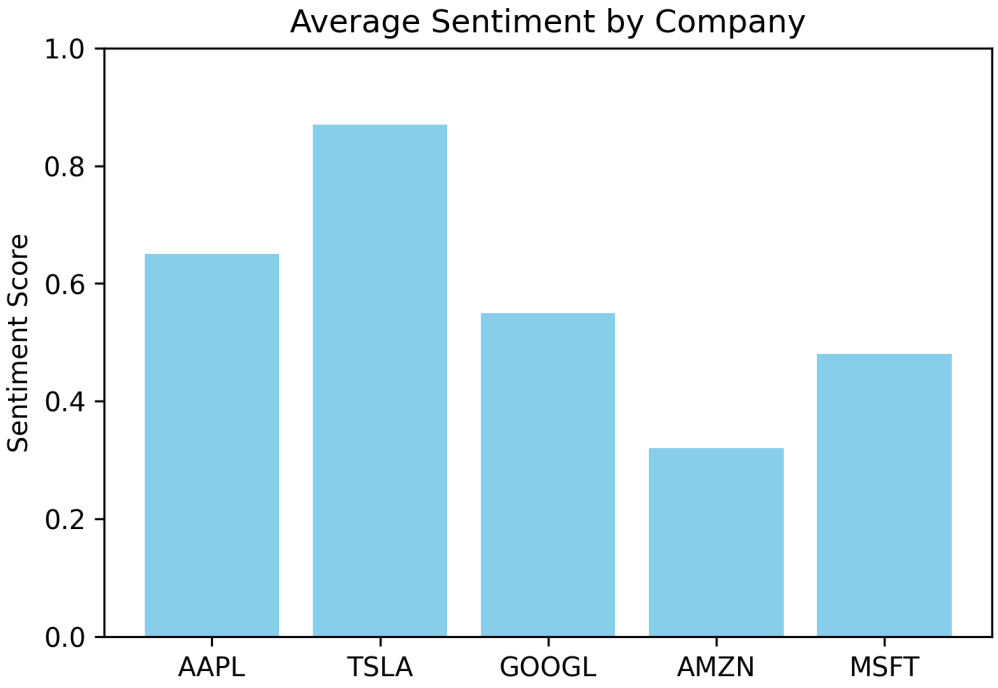

Figure 2: Average Sentiment by Company.

7. Insights

- **Traceability:** Each decision is linked to explicit data and reasoning.
- **Reproducibility:** MCP ensures deterministic node execution.
- **Modularity:** Each node is independently testable and replaceable.

8. Future Steps

- Integrate dynamic retrieval with live APIs.
- Add feedback-based self-correction via Self-RAG.

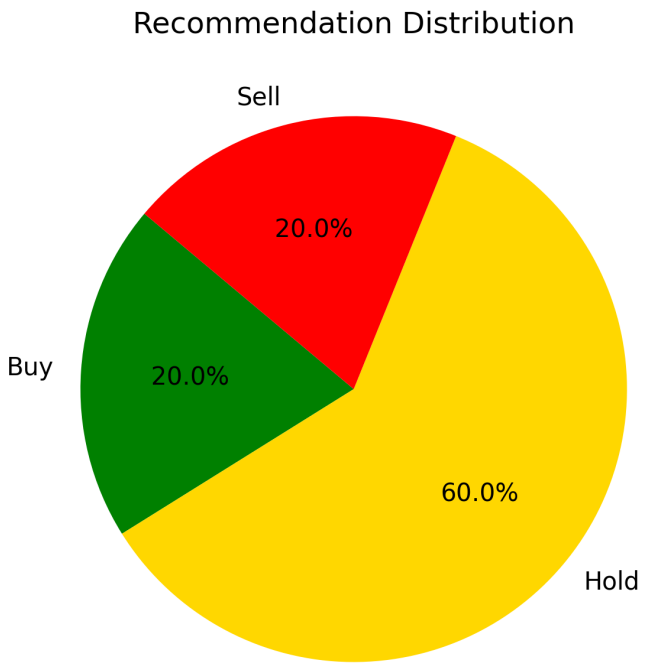

Figure 3: Recommendation Distribution: Predominantly "Buy" signals.

- Extend MCP routes for multi-ticker batch execution.

9. Conclusion

This implementation demonstrates a scalable, agentic AI system for investment research. By integrating LangGraph's DAG orchestration with MCP routing, the architecture achieves modularity, transparency, and end-to-end explainability. The use of individual visual outputs and tabular summaries enables clear communication of results to stakeholders.

References

Schick, T., Dwivedi-Yu, J., Raileanu, R., Zettlemoyer, L., & Scialom, T. (2023). *Toolformer: Language models can teach themselves to use tools*. arXiv:2302.04761.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., & Kiela, D. (2020). *Retrieval-augmented generation for knowledge-intensive NLP tasks*. NeurIPS 33, 9459–9474.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., & Yih, W. (2020). *Dense Passage Retrieval for Open-Domain Question Answering*. EMNLP.