Investment Research Assistant (IRA)

Technical Overview

Business Context & Problem Statement

Ira is a helpful AI-powered assistant, helping you make smarter and more informed investments!

The agent simplifies staying informed about the stock market by automatically aggregating and analyzing market data and news. It delivers concise, structured research briefs, allowing users to quickly understand a company's performance and sentiment without manually sifting through multiple sources.

The agent significantly reduces the time and effort required for financial research by parsing, summarizing, and contextualizing large volumes of market data and news. These tasks would normally require hours of manual analysis, which is reduced down to less than a minute with an agentic solution.

Architecture and Technical Design

The frontend is built using React and TypeScript, leveraging React's modular component structure and TypeScript's type safety for more maintainable and readable code. The UX was designed to be simple and intuitive, presenting insights in a clear and cohesive manner to the end user. The backend is implemented in Python, chosen for its LLM-friendly ecosystem and easy-to-use integration frameworks. For this project, LangGraph was used for agent orchestration, providing a flexible and future-proof alternative to LangChain's AgentExecutor (which is being deprecated). Ollama was used to quickly spin up the local LLM, and FastAPI provided a lightweight API layer for communication with the frontend.

For the LLM integration, I evaluated several models, including *gpt-oss* and *deepseek-r1* among others. Many of these models, I found, were too large to run locally or lacked agentic prompt capabilities. I ultimately selected *qwen:12b*, which offered a strong balance of performance, response accuracy, and compatibility with agentic workflows. The agent was prompted to return responses as structured JSON objects following a strict interface, with the temperature set to 0 to minimize hallucinations. I implemented robust backend and frontend error handling to ensure that any deviations from the expected format do not disrupt the user experience.

To provide actionable insights, I integrated the Alpha Vantage APIs for both market data and news sentiment. These were implemented as *market_data_tool* and *news_sentiment_tool*, callable by the agent when deemed necessary. The agent is instructed to include these sources in its responses, giving users visibility into the origin of the data. Since the free tier of Alpha Vantage only allows 25 requests per day, I implemented a mock mode that bypasses live API and LLM calls and returns pre-set responses, enabling continued testing and development without worrying about API limits.

In this project's development, LLM models were used on my local machine which is not specifically optimized for AI text generation. This limited inference speed and overall performance. To improve enterprise-level

performance, deploying the models on cloud-based AI infrastructure would give massive performance gains by leveraging GPU or specialized AI hardware for faster inference. Additionally, careful selection of models, prompt optimization, and hyperparameter tuning, can reduce latency and computational overhead. In production, real-time performance can be further enhanced by caching frequently accessed data, and leveraging asynchronous processing in the backend.

The architecture could also be set up in a serverless environment, which provides inherent scalability. In a production environment, the LLM could be hosted on services such as AWS Bedrock, the backend deployed on AWS Fargate, and the frontend served as a static web app on S3 or Lambda. This approach allows automatic scaling to handle varying loads, reduces operational complexity, and offers cost efficiency by only consuming resources on demand.

Future Enhancements

Due to time and resource constraints, smooth streaming of information was unfortunately not implemented. Given the modular, dashboard-style UI, enabling real-time streaming would have required multiple calls to the local LLM for each data point. Since these calls are relatively slow and computationally expensive, this functionality was deferred in favor of a more responsive and cohesive user experience. However, integrating data streaming remains a compelling future enhancement.

Additional improvements could include:

- **Expanded data sources and agentic tools:** Incorporate more endpoints from the Alpha Vantage API to provide additional market insights.
- **Ticker searching:** Add an autofill or Select component that dynamically searches for tickers as the user types.
- **Advanced charting:** While an MVP sentiment chart was implemented from scratch, migrating to a pre-established charting library such as *D3* or *react-google-charts*, could enable more sophisticated visualizations, such as time series charts showing stock price fluctuations over time.
- **RAG-based insights:** Use companies' earnings reports as a knowledge base for Retrieval-Augmented Generation (RAG) to allow the agent to provide even deeper analysis into the stock market.

AI Use Disclosure

GitHub Copilot played a supportive role throughout this project. While all outputs were carefully reviewed and validated for correctness, AI tools helped accelerate my development and problem-solving, and allowed for more rapid prototyping. Specifically, Copilot was used to:

- Debug issues related to LangChain/agent setup.
- Generate boilerplate code for React components.
- Explore and iterate on potential system designs, project structure, and implementation approaches.