AlphaMesh: A Personalized Stock Investment App Powered by Multi-Agent LLM Orchestration

BY

LIM YEU CHUAN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEB 2026

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr. Png Wen Hao who has given me this bright opportunity to engage in a multi-agent LLM project. It is my first step to establish a career in computer science and software engineering. A million thanks to you.

To all my friends who were by my side throughout this journey in UTAR, I whole-heartedly thank you all for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

The increasing complexity of financial markets and the limitations of existing investment platforms present significant challenges for retail investors, who often face information overload, limited personalized guidance, and opaque analytical tools that are difficult to interpret. Traditional robo-advisors usually rely on static questionnaire-based profiling, while modern investment applications often overwhelm users with large amounts of market data, charts, and financial indicators without sufficient explanation or personalization. This project presents AlphaMesh, a personalized stock investment application powered by a multi-agent Large Language Model (LLM) orchestration framework that aims to provide more explainable, adaptive, and user-oriented investment support. AlphaMesh was developed as a working web-based prototype using a FastAPI backend and Vite frontend, with interface design assistance from Google Stitch AI. The system consists of an Orchestrator Agent, News Analysis Agent, and Fundamental Analysis Agent, which work together to interpret user queries, retrieve relevant user and portfolio context, analyse recent financial news, compute fundamental financial metrics, and synthesize personalized investment responses. To support long-term personalization, AlphaMesh implements a dual-level memory architecture that combines ChromaDB vector retrieval for contextual article recall with Neo4j graph-based memory for structured financial relationships and user-specific interest tracking. The system also supports portfolio management, conversation history, and dynamic visualization to help users understand market trends and company performance more clearly. Evaluation results show that AlphaMesh achieved an average score of 7.38 out of 10 in scenario-based response quality testing using LLM-based judge, successfully captured evolving user preferences across conversations, and demonstrated richer evidence retrieval through its dual-store RAG pipeline. Overall, AlphaMesh demonstrates the potential of multi-agent LLM systems to create a more transparent, personalized, and accessible investment analysis tool for both beginner and experienced retail investors alike in Malaysia.

Area of Study (Minimum 1 and Maximum 2): Multi-agent System, Large language Models

Keywords (Minimum 5 and Maximum 10): LLM Orchestration, Personalized Finance, Retrieval-Augmented Generation (RAG), Natural Language Processing, data visualization

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GLC	Government-Linked Company
ASNB	Amanah Saham Berhad Nasional
LLM	Large Language Model
RAG	Retrieval-Augmented Generation
NLP	Natural Language Processing
API	application programming interface
CoT	chain of thought
AWS	Amazon Web Service
AI	Artificial Intelligence
EDGAR	Electronic Data Gathering, Analysis, and Retrieval
SEC	Securities and Exchange Commission

Chapter 1
Introduction

1.1 Background Information

In Malaysia, financial literacy and investing have become increasingly important as individuals strive to secure their long-term financial future amid rising living costs and economic uncertainties. However, multiple sources have shown that many do not possess the required expertise to make sound financial decisions [1]. As such, for many years, access to sophisticated investment guidance and asset allocation was dominated by traditional human financial advisors from large investment firms. A good advisor understands a client's personal needs, helps them set financial goals, and provides crucial emotional support, especially during market downturns. However, this traditional model has significant shortcomings. Human advisors often require very high minimum funding requirements and impose high fees, with charges of up to 2% of assets under management [2], making their services accessible only to high-net-worth clients. This creates a gap where the average Malaysian had little opportunity for affordable financial guidance and to partake in stock-market investing.

In recent years, Robo-advisor have been a rising trend in Malaysia, with Government-Linked Company (GLC), renowned banks, and fintech companies incorporating it as a core feature on their platforms. For instance, RIA from Amanah Saham Berhad Nasional (ASNB) [3], Robo-Invest from UOB [4] and Stashaway [5]. Robo-advisors offer automated portfolio management, typically at a much lower price compared to traditional advisory models. By using online questionnaires to assess a client's financial situation, risk tolerance, and goals, they could automatically manage investments with minimal human intervention for a very large number of clients. This ease of access and affordability also lowers the barrier to entry, encouraging beginners to start investing. As such, the market for this technology is growing exponentially. After reaching $41.52 billion in 2023, it is projected to grow to $205.84 billion by 2027 [2]. As for the more experienced investors that prefers to manage their own finances, Retail-oriented stock-investment platforms have been on the rise, with platforms such as MooMoo [6], WeBull [7] and Interactive Broker IBKR [8], having breakneck growth in user count after entering the Malaysian retail market . These platforms offer user-friendly mobile apps, low

fees, and powerful analytical tools at a fraction of the cost previously charged by investment firms.

However, they are not without flaws. Many implementations of robo-advisors rely on rigid, questionnaires-based assessments [9]. This fails to capture a comprehensive user profile that continually evolves over time. Furthermore, despite its name, robo-advisor is not considered true advisors, as they mainly function as algorithms that passively execute predefined investment strategies rather than providing explanations, guidance, or relationship-building typically expected from genuine financial advice [9]. Then, current stock investment platforms in Malaysia often pack a vast amount of market data, charts, and complicated analysis tools into their apps, which can overwhelm even knowledgeable investors and create additional barriers for beginners to get started on their investment journeys.

This is a gap that Large Language Models (LLMs) are uniquely positioned to fill. Modern LLMs are capable of answering complex queries, analysing unstructured data, and generating meaningful insights through natural language interaction. Recognizing this potential, many investment platforms have already begun integrating LLM-powered chatbots into their applications [10], [11]. These assistants provide users, experienced and especially beginners, with guidance, helping them understand basic investment concepts, interpret data, and make more informed decisions. However, in most cases, the chatbot or LLM functionality is treated as an additional side feature rather than a core component of the platform. As a result, they often lack deeper capabilities such as long-term personalization, transparent reasoning, or more advanced, interactive tools. This gap highlights the need for a more comprehensive LLM-driven solution, which this research project seeks to address in later chapters.

1.2 Problem Statement and Motivation

1.2.1 Problem 1 - Lack of Personalized Guidance

Research has emphasized that eliciting a user's specific preferences is central to providing appropriate advice, and within high-stakes fields like finance, the relationship between the client and advisor plays a vital role [2]. Beginner investors often lack the necessary financial knowledge to make informed decisions or even understand their own investment preferences and hence relies on financial advisors for specific, personalised guidance.

However, the current implementations of Large Language Models (LLMs) as financial assistants often fall short in this regard. Previous work has focused on agents that handle only simple inquiries [12], with many implementations only capable of providing generic recommendations [13]. This creates a notable risk, as the inability of LLMs to capture the user's needs and preferences, combined with an investor's difficulty in judging the advice's quality, can lead the system to recommend poor financial assets [12]. Additionally, the majority of research for LLM financial assistant setting has focused on maximizing investment returns over a fixed period [14], [15], [16], [17], which overlooks the long-term goal of educating investors to make informed decisions. This causes users to treat LLM as a "black box" and incentivizes them to rely on LLM to make financial decision, with the narrow-minded goal of making the most money from stocks, without understanding the risk or consequences involved. The ultimate consequence is not only low user satisfaction but also the potential for large monetary losses, which highlights a significant gap between current LLM capabilities.

1.2.2 Problem 2 - Information Overload

Furthermore, existing investment platforms in Malaysia often adopt a one-size-fits-all approach that includes a vast, random variety of market data. This generalized strategy can lead to information overload, a phenomenon where the growing volume of product data on these platforms diminishes the user's ability to process it effectively, thus complicating financial decision-making [18]. Consequently, instead of feeling empowered, investors are often left struggling with complex interfaces and an overwhelming amount of information, which can impede their ability to make rational financial choices. The shortcomings of these platforms often lead investors to seek guidance from less reliable sources, such as social media, which introduces significant risks. According to one survey, 12% of respondents reported feeling more confused by the financial information on social media, and almost one in five investors

lost money based on online advice [19]. This challenge is compounded by the generally low levels of financial literacy among Malaysians [20]. These statistics reinforces the urgent need for investment platforms to provide better and more personalized guidance.

1.2.3 Problem 3 - Lack of Intuitive Visualisation Solution

Data visualization transforms raw financial metrics easy to digest charts or graphs that allows investors to identify trends and assess performance effectively. However, investors often find current financial analysis tools to be inadequate, as they frequently present data in dense, tabular formats that are difficult to interpret intuitively [21]. These platforms suffer from limitations such as static reporting and rigid constraints on how data can be viewed, often forcing users to rely on pre-defined dashboards that lack customization. Moreover, the cognitive load required to synthesize vast amounts of raw numerical data creates a barrier for investors [21]. Beginners often struggle to grasp complex relationships without visual aids, while experienced investors are constrained by the time-consuming process of manually constructing charts to test specific hypotheses. In addition, there is a notable absence of research focused on leveraging LLMs to translate natural language commands into dynamic visualization code. This represents a missed opportunity to provide users with a powerful, interactive tool that allows for the rapid generation of customized graphs and charts.

1.2.4 Motivation

A core motivation for this research is to empower investors, both experienced and beginners, through personalized guidance, leveraging LLM to foster more informed financial decisions. A study in Malaysia has shown that systems providing relevant advice are crucial in guiding a user's learning process, helping them make more knowledgeable choices [22]. By implementing a personalization mechanic, an LLM can cater its educational content to a user's existing financial knowledge. This allows the system to present only the most relevant data and metrics, offer appropriately levelled advice, and dynamically adapt its complexity as the user's expertise grows, ensuring the educational support remains effective and engaging.

Moreover, this approach promotes a shift towards human-agent collaboration, encouraging users to become active participants in the investment process rather than passive recipients of automated advice. This synergy not only accelerates investment analysis but also cultivates greater financial independence among Malaysians. By understanding the reasoning behind AI-driven suggestions, users are less likely to treat the technology as an incomprehensible "black

box" or rely solely on traditional financial advisors. Instead, they are empowered to engage with the analysis, ask critical questions, and ultimately take greater ownership of their financial strategy.

A transparent system that clearly explains the reasoning of its AI models is fundamental to increasing the adoption of this technology among the general Malaysian public [22]. Users are often hesitant to adopt technologies they find difficult to understand. Hence, transparency builds the necessary trust for wider acceptance [22]. It provides users with a more comprehensive view of the results presented by the AI and a clearer understanding of the recommended actions. Ultimately, this project aims to lower the barrier to entry for investing, allowing more Malaysian citizens to participate confidently. This is particularly crucial as financial literacy in Malaysia remains low [20], and it has confirmed that individuals with greater financial knowledge are more inclined to invest in the share market [22]. By making investing more accessible and understandable, this research seeks to enhance the financial literacy of the nation.

1.3 Project Scope

At the conclusion of this project, the primary deliverable will be a proof-of-concept software prototype of a multi-agent personalised investment app named AlphaMesh. This deliverable will include the functional software itself with the core capabilities of personalized guidance, multi-agent orchestration, and dynamic visualization generation. Alongside the prototype will be a comprehensive research report detailing the system's architecture, the theoretical foundations of the multi-agent and multi-layered memory systems, and a full account of the implementation process. The report will also include an evaluation of the system's performance and verification of the system's overall functionality, including a qualitative assessment of personalized advice across different investor personas

1.3.1 Project Boundary

The system's addressable stock market is limited to the United States' stock market, New York Stock Exchange (NYSE) for data sourcing and analysis. The reason being that there are a very large amount of open-access financial data and libraries targeted to access and analyse the financial data of company listed on the NYSE, as opposed to data on Bursa Malaysia, which may be hard to obtain, even with paid subscriptions. Moreover, the scope will be limited to

equities, such as stocks and ETFs, excluding other financial instruments, like mutual funds, bonds and other financial derivatives, as financial instruments are highly complex and inaccessible to most investors.

In addition, certain functionalities will be explicitly excluded. The system will not perform real-time trading or provide certified financial advice. The goal of the system remains to assist the user in making his investing decision, and not to replace the user. As such, the system is strictly an analytical and educational tool. Building on this, while full-scale integration with a brokerage platform like Interactive Brokers (IBKR) with their open-access APIs, is identified as a valuable future consideration in later phases of the project but is currently out-of-scope.

Furthermore, the prototype will be delivered as a web application only. The creation of a mobile version is excluded due to platform-specific development complexities that fall outside of the research focus. Moreover, web application is universally supported on all platform, albeit with some software-level restrictions in terms of the possible features, like notifications. Hence, in order for the project to be feasible within the given time and reach the widest possible audience, the development platform has been limited to only web application.

Finally, the project will not involve training new LLMs due to the extensive computational resources and time required for such tasks. Instead, it will leverage first-party, API-based LLMs provided by vendors such as OpenAI's ChatGPT, Google Gemini, or other comparable providers. This ensures that the system benefits from state-of-the-art models while remaining focused on integration and application-level innovation.

1.4 Project Objectives

1. To develop a multi-agent LLM framework to address needs of investors

This is the main objective of the project, which is to design, develop, and evaluate a novel multi-agent framework that leverages a multi-agent LLM system to simultaneously addresses the needs of both beginner and experienced investors by delivering a deeply personalized investment guidance.

2. To develop a dual-levelled RAG system for deep user personalization.

This objective focuses on constructing a two-levelled RAG system, that combines both a vector-based and graph-based RAG to achieve a novel personalisation aspect for each user. This approach acts as the long-term memory of the system, enabling the agents to provide advice that is not only context-aware but also evolves with the user over time by capturing the user's financial knowledge, risk tolerance, investment goals, and past interactions.

3. To develop a specialized Visualization Agent for dynamic data representation.

This objective focuses on developing a dedicated agent responsible for transforming complex quantitative data and portfolio metrics into intuitive visual formats, such as interactive charts and trend graphs. By augmenting the textual analysis with visual aids, the system aims to significantly enhance the interpretability of financial insights, allowing users to understand market trends and portfolio allocations easily without being overwhelmed.

1.5 Impact, significance and contribution

This research makes several key contributions to the field of AI-driven financial technology by introducing AlphaMesh, a novel multi-agent framework designed to address critical gaps in existing investment platforms.

Firstly, the project introduces a specialized multi-agent orchestration framework that significantly improves the accuracy and depth of the financial advice. Unlike standard LLM chatbots that rely on a single, generalized model, AlphaMesh assigns analytical tasks to domain-specific agents, such as the News Analysis Agent for qualitative sentiment and the Fundamental Analysis Agent for quantitative metrics. This collaborative architecture ensures that the final output is not only comprehensive but also factually grounded and robust against the hallucinations common in single-model systems.

Secondly, this project proposes a dual-levelled memory architecture for deep user personalization. This approach represents a significant advancement over the static, questionnaire-based profiling of traditional robo-advisors. By proactively capturing and structuring a user's financial knowledge, risk tolerance, interaction history, stated goals and much more within a hybrid RAG memory module, AlphaMesh enables dynamic, long-term adaptability that evolves alongside the investor.

Finally, AlphaMesh contributes an innovative natural language-to-visualization pipeline that significantly enhances the interpretability of complex financial data. This capability lowers the entry barrier for financial analysis, enabling beginners to intuitively grasp market trends through automatically generated visual aids, while allowing experienced investors to rapidly isolate and visualize specific datasets without the burden of manual data manipulation.

Chapter 2
Literature Review

2.1 LLM Agent

The core motivation behind LLM agents is to overcome the inherent limitations of static LLMs. A standard LLM is a static model; its knowledge is frozen at the time of training, it cannot interact with the real world, and it is prone to hallucinating facts or making errors in tasks requiring precise computation or up-to-date information. An LLM agent overcomes these limitations by augmenting the LLM with a framework that allows it to use external tools [23] [24]. An LLM agent is a system that utilizes an LLM as its central controller, or "brain," to interact with external environments, use tools, and autonomously work towards achieving complex goals [23]- [25]. Unlike traditional models that produce a final output in a single step, LLM agents operate through an iterative process of reasoning, planning, and acting, enabling them to decompose complex problems into a sequence of manageable actions. This paradigm transforms the LLM from a passive knowledge repository into an active problem-solver capable of dynamic interaction and decision-making.

2.1.1 Limitations of Standard LLMs

The development of LLM agents is a direct response to several well-documented limitations inherent in standard, parametric-only language models. Primarily, their knowledge is static and confined to the data they were trained on. This means they cannot easily expand or revise their internal memory to incorporate new or up-to-date information, which is a significant drawback [2], [4]. This static nature directly contributes to the problem of "hallucination," where models generate plausible but factually incorrect statements [1], [2], [4]. Without the ability to consult an external source, an LLM's capacity to access and manipulate its stored knowledge is limited, often leading to factual errors and error propagation, especially in complex reasoning tasks [1].

Furthermore, standard LLMs often lack specialized capabilities required for certain tasks. For example, they struggle to perform precise mathematical calculations, a limitation that can be easily overcome by an external calculator tool [2]. Finally, standard models lack transparency and provenance for their generated content. It is often impossible to trace the source of their information, making it difficult to verify their claims or trust their outputs [1], [4]. This

contrasts sharply with agentic systems that can provide citations or explicit reasoning steps to support their conclusions [1], [3]. These deficiencies collectively restrict the reliability of standard LLMs for knowledge-intensive applications, thereby motivating the development of agent-based and retrieval-augmented architectures.

2.1.2 Working Principle of Agent

Instead of producing a final output in a single step, the agent operates in an iterative loop. The process begins with the agent generating a "thought" or reasoning trace to decompose a task, create a plan, and determine what information is needed [23]. Based on this reasoning, the agent then generates a specific "action," which involves calling an external tool via an API, such as a calculator, a Wikipedia search, or a web browser, to gather new information or perform a calculation [23], [24], [25]. This action is executed in an external environment, which returns an "observation", like search results or a calculated value, to the agent. The agent incorporates this new observation into its context and repeats the cycle, using the new information to generate the next thought and subsequent action, continuing this interleaved process until the task is complete [23]. This entire procedure allows the model to dynamically gather and integrate external knowledge, effectively teaching itself how to use tools to solve complex problems that require up-to-date or specialized information [24], [25].

2.2 LLM Memory

LLM memory is an external component that augments LLM agents, allowing them to process historical experiences and accumulate knowledge [26]. This capability is essential for enabling sustained, personalized interactions, such as a personal assistant that recalls past conversations to tailor its responses [27]. Memory can be stored in two primary forms, which are textual, using natural language for better interpretability, or parametric, where knowledge is encoded into the model's parameters [26]. Beyond simple storage and retrieval, advanced systems feature sophisticated memory management. For example, the system proposed by [28] is "agentic," autonomously organizing new information into an interconnected knowledge network by creating dynamic links between related memories. To be more efficient and human-like, other systems incorporate methods to selectively forget or reinforce memories based on their significance and age, creating a more manageable knowledge base [27].

2.2.1 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a framework developed to enhance LLMs by combining their parametric knowledge with external, non-parametric information sources, typically implemented as a dense vector index of a large text corpus, before integrating the information directly into the generation process [29]. The framework operates through a two-stage process: retrieval and generation. Given a user query, the retriever searches an external corpus which is often indexed using methods such as Dense Passage Retrieval (DPR) to identify the most relevant passages. These passages are then combined with the original query and provided to the generator, a sequence-to-sequence LLM, which synthesizes a final response [29].

This hybrid design provides several key improvements over traditional parametric-only models. By retrieving information from an external vector index, RAG gains dynamic knowledge access, allowing it to incorporate information beyond what is stored in its parameters and overcoming the static memory limitation [29]. This design enhances interpretability and updateability, as the non-parametric memory consists of raw text passages that can be inspected to verify the model's sources and can be updated by simply replacing the document index without retraining the entire model. Finally, by grounding generation in retrieved factual text, RAG significantly reduces hallucinations, producing outputs that are more factual, specific, and reliable.

2.2.2 Graph-based RAG

While foundational RAG models significantly improve factual grounding, they often rely on retrieving independent chunks of text from a vector database, which struggles to capture the complex relationships between different pieces of information, potentially leading to fragmented answers for complex queries. As such, recent works incorporates graph structures into the indexing and retrieval process, called graph-based RAG.

Graph-based RAG builds a knowledge graph by extracting entities like people and places, to be represented as nodes, and their relationships as edges, from the source documents with the use of LLM [30], [31], [32]. This allows the system to represent and understand the complex connections across documents. Then, retrieving data from the graph structure often involves a two-levelled mechanism. The first is a low-level, entity specific retrieval that captures detailed information and direct relationships, and the other is a high-level retrieval that targets queries that spans across multiple themes and topics within the graph [30], [31]. The effectiveness of the retrieval strategies, however, relies on the quality of the underlying graph index.

The creation of the index differs from one system to another. One approach cluster the graph into thematic communities and uses an LLM to generate detailed, multi-page summaries for each community [31], which is rich in detail, but computationally intensive. Then, another approach uses LLM to create concise key-value pairs and generates both low-level and high-level keywords for each entity [30], which is more lightweight and adaptable. Nevertheless, systems that make use of LLM for indexing suffer from poor scalability as it is computationally expensive upfront to create the graph index.

One system aims to combat this by deferring the expensive LLM work until query time [32]. During the indexing phase of the proposed system, the concept graph is constructed using simple, lightweight Natural Language Processing (NLP) methods. This reduces the indexing costs to just 0.1% of implementation by [31], whilst still preserving the ability to capture relationships and organizes concepts into communities. Then, at query time, the LLM is only used for relevance testing when necessary to evaluate whether the retrieved text chunks and graph communities are relevant to the query. As a result, the system delivers answer quality that is at least on par with its peer at dramatically lower costs, making it more scalable and efficient [32].

2.3 Financial Analysis

Financial analysis is the process of evaluating businesses, projects, budgets, and other finance-related transactions to determine their performance and suitability. Typically, financial analysis is used to analyse whether an entity is stable, solvent, liquid, or profitable enough to warrant a monetary investment. The domains of finance and investment management have always been characterized by complexity, uncertainty, and rapid evolution [33]. Financial markets are inherently dynamic and challenging to analyse accurately, as they involve a web of interrelated factors, including economic indicators, geopolitical events, and investor behaviours [34].

2.3.1 Fundamental and Technical Analysis

Financial analysis is broadly categorized into two main branches: fundamental analysis and technical analysis. These two approaches represent differing philosophies, and analysts often specialize in one.

Fundamental analysis involves evaluating a security's intrinsic value by examining related economic and financial factors [35]. The end goal is to arrive at a number that an investor can compare with a security's current price to see whether the security is undervalued or overvalued. Fundamental analysts study everything from the overall economy and industry conditions to the financial strength and management of individual companies by analysing financial statements, earnings reports, and insider transactions [35]. Metrics used in fundamental analysis include earnings per share (EPS), price-to-earnings (P/E) ratio, return on equity (ROE), and debt-to-equity ratio. Technical analysis, on the other hand, is a trading discipline employed to evaluate investments and identify trading opportunities by analysing statistical trends gathered from trading activity, such as price movement and volume [35]. Unlike fundamental analysis, technical analysis focuses on patterns of price movements and trading signals. Technical analysts use indicators such as Moving Average Convergence Divergence (MACD) and Relative Strength Index (RSI) to forecast future price movements and to help time entry and exit points for trades [35].

Despite the differences in their methodologies, both fundamental and technical analysis seek to inform profitable decisions in the market. A powerful technique that can be used to validate strategies derived from either approach is backtesting, which assesses the viability of a trading strategy by testing it on historical data [34], [35].

2.3.2 Limitation of Traditional Financial Analysis Technique

Before the widespread use of advanced computational models, financial analysis relied on a set of traditional techniques. These include statistical models like the Capital Asset Pricing Model (CAPM), Multi-factor Asset Pricing models, and the Autoregressive Integrated Moving Average (ARIMA) model [35]. However, these methods often depend on linear assumptions and struggle to capture the non-linear dynamics that is inherent in financial markets [35]. A significant limitation of these traditional approaches is their inability to effectively process and incorporate unstructured data. A vast amount of valuable information is embedded in unstructured sources like news articles, economic reports, and corporate announcements [35]. Traditional models are not equipped to handle this type of data, thus missing out on rich insights that can be derived from them, such as market sentiment and emerging trends.

2.4 Review of Existing System

2.4.1 FinRobot

Yang et al. [15] introduces FinRobot, an open-sourced multi-layered, multi-agent AI platform for financial analysis. At its core, the platform employs a Financial Chain-of-Thought (CoT) process to orchestrate the entire analytical workflow into a step-by-step plan. This process is overseen by a director agent, which leverages a Smart Scheduler to dynamically route tasks to the best-suited model by scoring and ranking agent performance for optimal accuracy. However, this does require access to multiple different LLMs, which can affect the efficiency and cost of operation for the system. This is unfortunately not tested by the authors themselves.

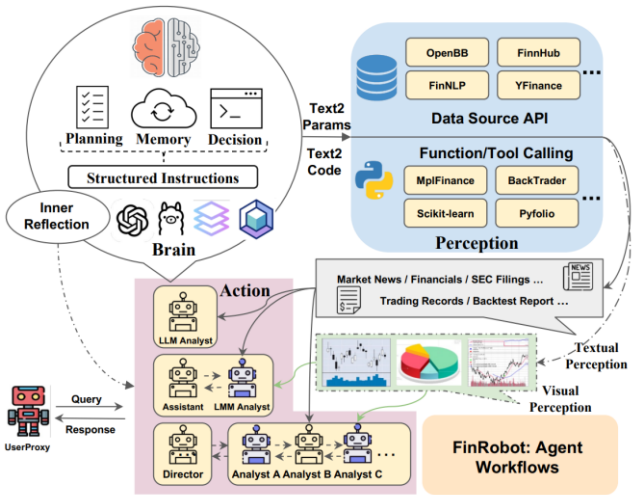


Figure 2.1: FinRobot's system architecture. Adapted from [15].

The workflow begins with an Assistant agent handling data collection and pre-processing via APIs and other tools to produce structured datasets. Following this, the LLM Analyst agent synthesizes qualitative insights from unstructured text, such as news and earnings transcripts. These findings then guide the Financial Analyst agents to perform targeted quantitative analysis using statistical tools and financial models. To execute their assigned tasks, these agents utilize an Action module that enables tool usage by converting text into functionalities like API calls (Text2Params) to perform document analysis plus report generation, and dynamic code writing (Text2Code). To ensure all analysis are grounded, this entire process is supported by the another layer called "DataOps" that manages diverse financial datasets and employs Retrieval-Augmented Generation (RAG). The RAG retrieves real-time context from external knowledge bases like vector databases and knowledge graphs, and ensures the generated analysis is based on the latest, factually correct information.

While the system presents a comprehensive framework for task-oriented financial analysis, its architecture lacks a personalization feature that could maintain long-term memory of an individual user's profile, like their preference and knowledge. Furthermore, the multi-agent system is structured for cooperative task execution towards a single conclusion rather than presenting differing viewpoints. This limits its transparency for complex queries where multiple valid strategies may exist.

2.4.2 FinMem

The system proposed by [16] is built upon three interconnected modules: Profile, Memory, and Decision-making. The **Profile module** allows for agents with customizable characteristics by combining two key components. First is a knowledge base of the stock to trade, which includes its sector-specific data and historical performance. Then, there is the dynamic risk inclination setting, which allows the agent to dynamically shift its risk preferences from risk-seeking to risk-averse, or vice versa, based on real-time market performance, such as a decline in cumulative return. This contextual awareness helps to better filter and prioritize information, enhancing its inferencing accuracy and adaptability. However, effective stock investing requires considering robust technical and fundamental analysis alongside risk preference. As these analyses are performed by the same agent, an initial bias towards one could lead to skewed outcomes, which the user cannot directly influence or discern. A more effective design

might involve multiple specialized agents, each dedicated to distinct analytical methodologies, to provide a more balanced and transparent basis for the final trading decision.

Furthermore, the system showcases a **Memory module**, as seen in Figure 2.4.3. The **working memory** acts as a dynamic workspace, performing three key operations: summarization, observation, and reflection. Summarization condenses user's query and external market data into actionable insights, directing them to the appropriate long-term memory layer. Then, observation gathers market facts, such as historical stock prices and momentum, which are crucial for assessing trends and trading strategies. Lastly, reflection processes these insights, generating immediate trading actions ("Buy", "Sell", "Hold") and their rationales, and performs re-evaluations of the trades taken over a given period, then stores the outcomes and other market data related to the taken trades in a layered long-term memory.

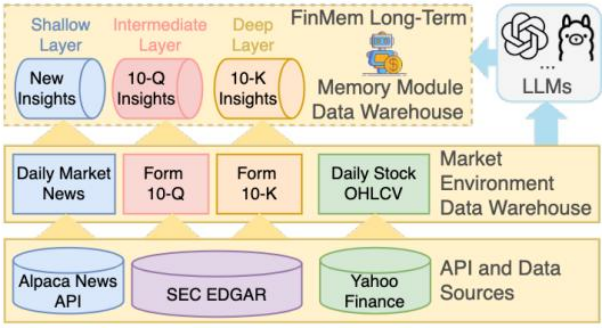


Figure 2.2: FinMem's Long-term memory architecture. Adapted from [16].

The **layered long-term memory** organizes financial data insights into shallow, intermediate, and deep layers. Each layer is designed to retain information with different decay rates and timeliness, ensuring that daily news, for instance, is channelled to the shallow layer for immediate impact but high decay rate, while annual company reports with longer-term implications are stored in the deep layer, with low decay rate. Information is retrieved from all layers based on score of recency, relevancy, and importance. Additionally, highly impactful memory events are migrated to deeper layers of the memory for extended retention based on an access counter function, resetting their recency score to maintain their salience. This dynamic feature makes the system robust for information retrieval, as even initially shallow news can gain deep retention if highly relevant to user's queries over time. However, achieving continuous improvement and high accuracy hinges on the fine-tuning of memory parameters, particularly the decay rates, to ensure relevancy of the retained content and prevent misleading information from overloading the memory.

2.4.3 TradingGPT

Li et al. [17] introduce TradingGPT, a multi-agent framework designed to enhance automated stock trading. As seen in Figure 2.4.3, the system is built on three core pillars: a layered memory system that mimics human memory recall, distinct agent characters with unique risk profiles, and an inter-agent debate mechanism for collaborative decision-making.

The framework's foundation is its layered memory, which organizes financial information into short, mid, and long-term layers, each with a custom decay rate modelled after the Ebbinghaus forgetting curve. This ensures that timely information, such as real-time market news (short-term), is prioritized for immediate action, while quarterly corporate filings (mid-term) and macroeconomic indicators (long-term) inform more strategic views. Information retrieval is governed by a weighted score of Recency, Relevancy, and Importance. Moreover, events linked to significant trading profits or losses are elevated in score, allowing the system to learn from its most impactful past actions.

Additionally, the multi-agent component activates specialized agents for analysis. Each agent is confined to a specific sector of stocks with a distinct character ("risk-seeking," "risk-neutral," or "risk-averse"). When a stock is to be analysed, only agents whose scope include that ticker is activated. These selected agents then perform the same core task, which include independently analysing the stock by synthesizing information from their layered memories. Following this, if the selected stock is included within multiple agent's sector of stocks, the relevant agent will engage in a debate to share and critique their findings, aiming to produce a more robust collective strategy.

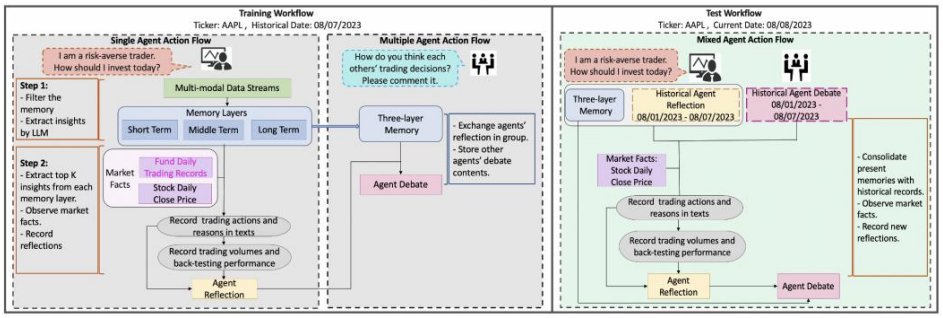


Figure 2.3: TradingGPT's system architecture. Adapted from [17].

However, the design of these agents presents several fundamental challenges. First, the paper does not specify how risk preferences are assigned to agents. This creates a risk of mischaracterization, where a historically volatile sector might be assigned a "risk-seeking" agent, leading to flawed recommendations. Furthermore, these characters are static and does not adapt over time, meaning the risk profile cannot adjust its strategy if a sector's risk profile changes due to macroeconomic shifts, severely limiting the system's adaptability. Second, both fundamental and technical analysis are performed by the same LLM agent. This can lead to conflicting internal logic, as short-term price fluctuations, that is relevant to technical analysis, might disproportionately influence a decision on a fundamentally strong company, potentially causing the agent to falsely recommend based on temporary market noise. Additionally, any inherent bias in the agent's LLM towards one or the other type of analysis will also affect the quality of reasoning. Next, while the debate mechanism is innovative and will provide a more comprehensive perspective, the paper does not specify how conflicting agent opinions are resolved into a final, actionable trade. Hence, without a robust mechanism to synthesize outputs from the various agents, a clear trading decision cannot be made, will undeniably be confusing for the user themselves to try and interpret all the varying opinions at once.

Then, the system's operational scope and output are also limited. The framework is designed for single-stock analysis, where an agent is prompted for one ticker at a time. This approach entirely overlooks a crucial aspect of investing: portfolio management. The system lacks mechanisms for diversification, capital allocation, or managing portfolio-level risk. Moreover, the agent's recommendation is constrained to selecting one of five predefined actions ranging from "significantly increase position" to "significantly decrease position". This rigid output format forces a complex financial decision into an overly simplistic choice, limiting the nuance and quality of the final recommendation.

2.4.4 FinArena

Xu et al. [14] introduce FinArena, a human-agent collaboration framework for financial analysis and forecasting. The system delegates the analysis of different data modalities to specialized LLM-based agents. As seen in the "Analysis Box" of Figure 2.4.4, The architecture comprises three core analytical agents, which includes a Time Series Agent for processing historical stock price data, a News Agent for summarizing and extracting insights from news articles, and a Statement Agent for analysing structured financial statements. A fourth Universal Expert Agent synthesizes the outputs from these three specialists to formulate

a preliminary trading recommendation. A key innovation of FinArena is the human-in-the-loop component, which allows a human investor to provide their risk preferences, like "conservative", to the system. The agent then adjusts its final recommendation to align with the user's risk profile, which achieved personalisation to a certain extent.

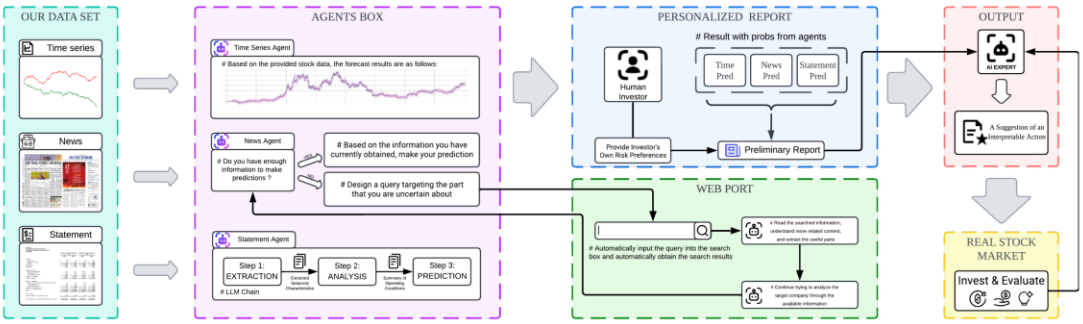


Figure 2.4: FinArena's system architecture. Adapted from [14].

To enhance the reliability of its analysis, the News Agent employs an adaptive Retrieval-Augmented Generation (RAG) mechanism. A "Judgment Module" first assesses whether a news-related query can be answered using the LLM's pre-trained knowledge. If not, the module triggers the RAG process to perform an internet search, reducing hallucinations and ensuring the analysis is grounded in the latest information. Similarly, the Statement Agent uses an iterative reasoning process, analysing financial statements step-by-step to understand a company's operational health. Notably, it is comprised of 3 different steps to form a chain of thought (CoT): the first is identifying the company's operational pattern over one financial year, then the analysed pattern and the original statements are jointly sent to the next LLM for a comprehensive analysis of the company's annual operation in each financial quarter. Moreover, to enhance interpretability, user can view any of the intermediate results within the CoT.

The framework's personalization feature, while innovative, is reactive rather than proactive. The system adjusts its final recommendation based on the user's stated risk preference but lacks a long-term memory or profile module to learn and remember a user's evolving investment style, past decisions, or knowledge level. Each interaction is treated as a discrete event, preventing the system from developing a deeper, continuously refined understanding of the individual user. As such, this forces the user to manually recalibrate the system's risk preference over time using the prompt, which apart from being tedious, is ineffective as the user themselves might not lack the awareness of changes in his own risk preference and might not even remember to update the system.

Chapter 3

System Methodology

3.1 Multi-Agent System Architecture

AlphaMesh is designed as a development-based financial analysis system powered by a multi-agent Large Language Model orchestration framework. Instead of relying on a single general-purpose LLM to answer every user query directly, the system separates the workflow into three main agents: the Orchestrator Agent, the News Analysis Agent, and the Fundamental Analysis Agent. This separation allows each agent to focus on a specific part of the investment analysis process, resulting in responses that are more structured, evidence-grounded, and explainable.

As shown in Figure 3.1, the system follows a hierarchical orchestration design. The Orchestrator Agent serves as the entry point for each user request. It interprets the user's query, retrieves relevant user and portfolio context, validates detected ticker symbols, and determines whether specialist analysis is required. If the query is simple, such as a general explanation or clarification request, the orchestrator may respond directly. However, if the query requires deeper financial analysis, it dispatches the task to either the News Analysis Agent, the Fundamental Analysis Agent, or both.

The architecture is also supported by a memory module and external data sources. The memory module allows AlphaMesh to reuse previously ingested evidence, retrieve long-term financial knowledge, remember user-specific interests, and maintain continuity across conversation turns. External sources such as NewsAPI, Tavily, SEC EDGAR, and yfinance are used when the system requires updated news articles or structured company financial data. After the specialist agents complete their work, the Orchestrator Agent synthesises their outputs with the user's context and portfolio information to produce the final response. This design improves modularity, transparency, and extensibility, since each agent has a clear role and future

specialist agents can be added without changing the overall orchestration logic.

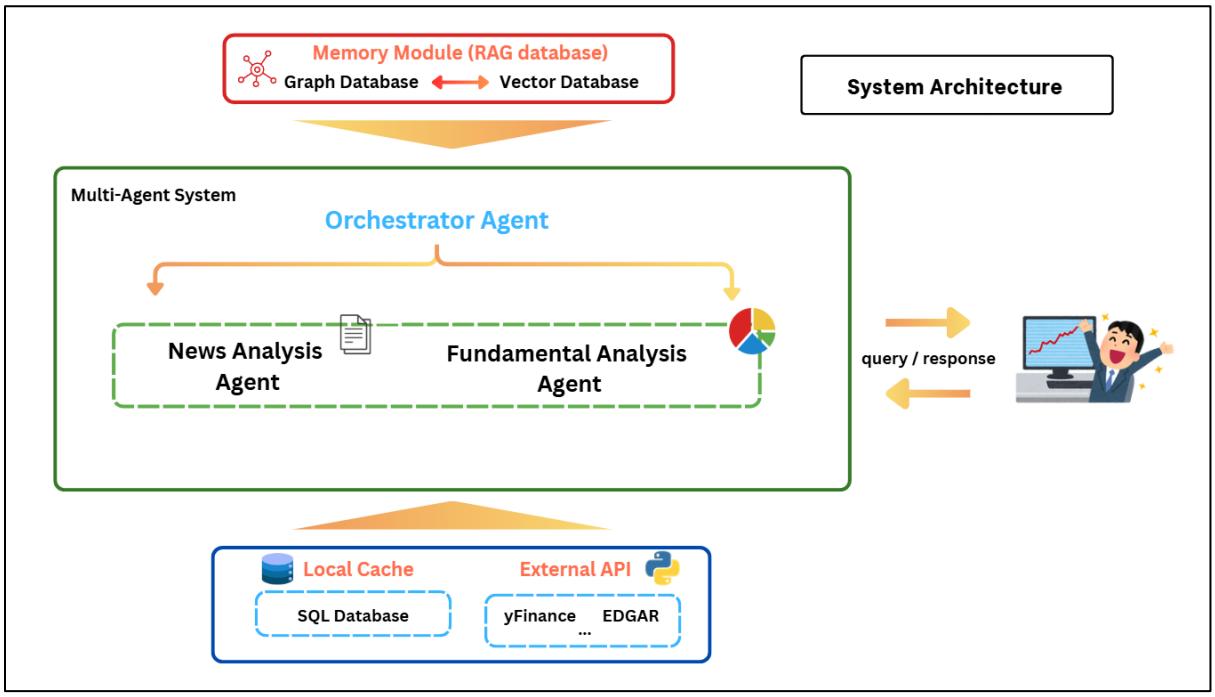


Figure 3.1: AlphaMesh's System Design Diagram

3.1.1 Agent Roles, Inputs, and Outputs

Referring to Table 3.1, the Orchestrator Agent is the highest-level agent in the system. It does not perform detailed financial analysis by itself unless the user's request can be answered directly. Instead, its main purpose is to understand the user's intent and coordinate the correct analytical pathway. This includes deciding whether the user needs news-based qualitative analysis, fundamental quantitative analysis, or a combination of both. It also ensures that the final response is personalised by incorporating user context, portfolio holdings, prior memory summaries, and outputs from the specialist agents.

The News Analysis Agent is responsible for handling unstructured financial information. This includes recent company news, sector developments, market events, and other external narratives that may influence investor sentiment or business outlook. Its workflow combines internal memory retrieval with external article fetching. Retrieved and fetched articles are deduplicated, ranked, grouped by source, and then used to generate an evidence-grounded news analysis. This makes the agent suitable for queries involving recent catalysts, risks, market perception, or qualitative company updates.

The Fundamental Analysis Agent focuses on structured company data. It retrieves financial statements and stock price history, prepares the data into a usable format, and applies controlled financial tools to calculate relevant metrics. Instead of relying on the LLM to perform numerical calculations directly, the agent uses predefined tools for computations such as growth rates, profitability ratios, liquidity ratios, debt ratios, valuation multiples, and custom formulas. This allows the final fundamental analysis to be based on computed financial evidence rather than unsupported model reasoning.

Table 3.1: Roles and responsibilities of AlphaMesh agents

Agent	**Main Role**	**Key Responsibilities**
Orchestrator Agent	Central coordinator and response synthesiser	Interprets the user query, retrieves user and portfolio context, selects and execute the required specialist agents, and synthesises final responses.
News Analysis Agent	Qualitative analysis	Retrieves and analyses financial news, market events, and other textual evidence that supports user's query
Fundamental Analysis Agent	Quantitative analysis	Retrieves, prepares, computes, and interprets structured financial data such as financial statements, stock prices, profitability ratios, solvency indicators, growth metrics and so on.

The inputs and outputs shown in Table 3.2 focus only on the essential information exchanged between the agents at the system architecture level. The Orchestrator Agent mainly receives the user's request together with personalization context and portfolio information, then converts this into a structured plan for the specialist agents. The News Analysis Agent receives the qualitative analysis target and returns evidence-grounded news insights, while the Fundamental Analysis Agent receives the quantitative analysis target and returns computed financial findings. Lower-level implementation details such as conversation identifiers, memory cache objects, tool execution logs, retrieved chunk IDs, and graph writeback tasks are

omitted here because they are internal workflow details that are better explained in Chapter 4. This keeps Chapter 3 focused on the overall system methodology rather than the backend implementation.

Table 3.2: Main inputs and outputs of AlphaMesh agents

Agent	**Inputs**	**Outputs**
Orchestrator Agent	User query, user context, portfolio holdings, available agent list	Selected agent plan, agent-specific tasks, final synthesised response
News Analysis Agent	Analysis goal, target company or ticker, date range, company context	News-based analysis, key catalysts and risks, cited article evidence
Fundamental Analysis Agent	Analysis goal, target company or ticker, analysis period, financial-data granularity	Fundamental analysis, computed financial metrics, selected financial data for display

3.2 Agent Memory Management

AlphaMesh's backend uses a memory management system that is divided into persistent memory and agent-level working memory. Persistent memory stores long-term financial knowledge and user-related context across interactions, while working memory supports short-term continuity within the same conversation. This section focuses on the persistent memory layer, which is mainly implemented through a dual-levelled Retrieval-Augmented Generation system that combines a vector store and a graph store.

The vector store provides fast semantic retrieval over stored article chunks, while the graph store preserves structured relationships between financial entities, article chunks, events, sectors, industries, markets, and financial concepts. This allows the system to retrieve

information based not only on text similarity, but also on how financial entities are connected to one another.

In a nutshell, the persistent memory workflow consists of two main processes, mainly ingestion and retrieval. During ingestion, newly fetched financial articles are first processed into smaller text chunks. These chunks are stored in ChromaDB for semantic search and are also represented in Neo4j as document and chunk nodes. After the agents generate their final analysis, selected evidence chunks and analysis outputs are also enqueued in a processing queue system to be used for graph enrichment, where relevant entities and relationships are extracted and written into the graph. This allows the memory system to preserve both raw source evidence and higher-level interpreted knowledge.

During retrieval, the system begins by performing vector search against ChromaDB to identify chunks that are semantically similar to the user's query. These initial chunks are then connected to Neo4j, where the system identifies related seed entities and expands toward neighbouring graph entities. Additional chunks connected to those entities can then be fetched and merged into the retrieval result. In this way, AlphaMesh can retrieve direct textual matches from the vector store while also discovering related evidence through graph traversal for a more comprehensive analysis.

3.2.1 Vector Store

The vector store is implemented using ChromaDB and is responsible for storing the textual content of financial articles. Each article is divided into smaller chunks, and each chunk is stored together with metadata such as the article title, source URL, publication date, chunk identifier, and extraction status. This structure allows the system to search for relevant article evidence using semantic similarity rather than relying only on exact keyword matching.

The main advantage of the vector store is fast contextual recall. When the News Analysis Agent needs supporting evidence, it can query ChromaDB to retrieve article chunks that are semantically close to the user's request or the agent's analysis goal. For example, if the user asks about recent risks affecting a company, the vector store can retrieve article chunks that discuss earnings pressure, regulatory issues, product demand, or other relevant market developments. This makes ChromaDB suitable as the first retrieval layer because it quickly identifies candidate evidence from stored textual information.

However, vector retrieval alone has limitations. It may retrieve chunks that are textually similar but disconnected from broader financial relationships. For this reason, AlphaMesh does not rely only on vector search. Instead, the retrieved chunks are used as the starting point for graph-based expansion, allowing the system to move from semantically relevant text into related financial entities and supporting evidence.

3.2.2 Graph Store

The graph store is implemented using Neo4j and acts as the structural memory layer of AlphaMesh. While ChromaDB stores article text for semantic retrieval, Neo4j stores the relationships between documents, chunks, companies, financial events, sectors, industries, markets, and financial concepts. This gives the memory system a more connected representation of financial knowledge.

In the graph store, article documents and chunks are linked to stable grouping anchors such as the Global Financial Events NodeSet. Extracted entities can then be connected to these chunks, allowing the system to understand which companies, events, or concepts are mentioned in each piece of evidence. The graph also contains broader taxonomy structures, such as market, sector, industry, and company relationships. This allows retrieval to move beyond isolated article chunks and explore surrounding financial context.

The graph store is especially important for relationship-based retrieval. After the vector store identifies relevant seed chunks, Neo4j can be used to find entities mentioned by those chunks and expand toward related entities. For example, a retrieved article about Apple may connect to entities such as the technology sector, consumer electronics demand, supply-chain events, or related financial concepts. The retrieval system can then fetch additional chunks connected to these neighbouring entities, producing a richer context than vector search alone.

3.2.3 Agent-Level Working Memory

Besides the persistent memory stored in ChromaDB and Neo4j, AlphaMesh also uses agent-level working memory to support short-term continuity within the same conversation. Unlike persistent memory, which stores long-term financial knowledge and user-specific interest records, working memory is mainly used to temporarily preserve useful information from recent agent runs. This allows the agents to avoid repeating the same retrieval, data preparation, or computation steps when the user asks follow-up questions.

The News Analysis Agent and Fundamental Analysis Agent each maintain their own working memory. The News Analysis Agent stores recently used article chunks, source URLs, seen URL keys, and seen chunk IDs, allowing it to reuse relevant evidence and avoid ingesting duplicate articles. The Fundamental Analysis Agent stores cached financial DataFrames, computed row labels, tool-call summaries, and execution results, allowing it to reuse prepared financial data and previously calculated metrics.

3.3 User Memory Management

AlphaMesh's user memory management is mainly represented through two components: chatlogs and portfolio records. Chatlogs preserve the user's previous conversations and generated analyses, while portfolio records store the user's current holdings.

3.3.1 Chatlog Management

For persistence, AlphaMesh stores one JSONL file per conversation and one index file per user. The index records metadata such as the conversation ID, creation time, last message time, message count, and turn count. Each conversation file then stores the full structured turn records. This design is suitable for the prototype because it keeps chat history separated by user and conversation, while still making it easy to list recent conversations and reload previous sessions.

After the final response is generated, the completed turn is saved. Each saved turn contains user and assistant messages, generated synthesis, agent analysis outputs, agent memory summaries, and so on. This richer structure is important because the frontend may need to redisplay not only the final answer, but also the agent-level analysis, financial data, citations, and other structured results from previous turns.

3.3.2 Portfolio Management

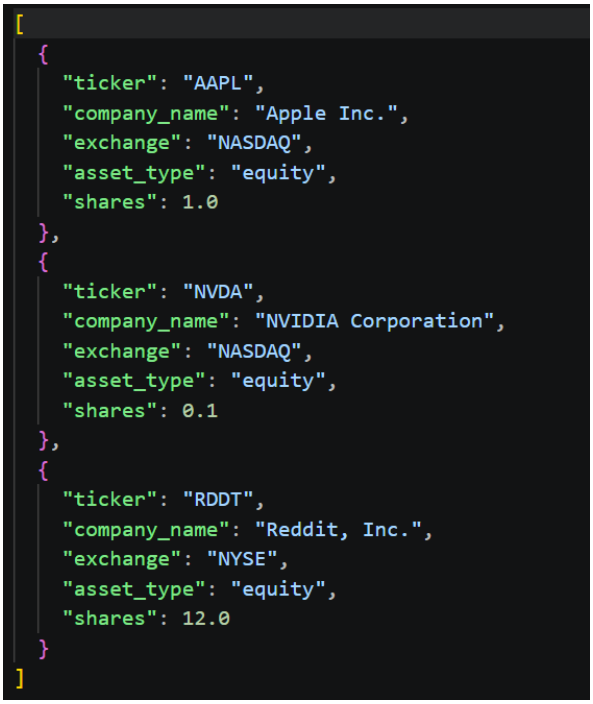


Figure 3.2: Example of what User Portfolio .json file contains

AlphaMesh also stores basic portfolio information so that the system can generate responses based on the user's actual holdings. Unlike the chatlog storage, the portfolio is stored in a per-user JSON file as seen in Figure 3.2. Each portfolio file contains a list of holdings, where the most important stored fields are the stock ticker and the number of shares owned.

The system also provides a portfolio user interface for users to add new assets or update existing assets. When the user modifies a holding, the updated portfolio JSON file is saved through the portfolio API. During an analysis request, the user's portfolio data is loaded and sent to the Orchestrator Agent, allowing the final response to consider the user's existing holdings instead of producing a generic stock analysis.

3.4 User Activity Diagram

As seen in Figure 3.2, the workflow begins when the user queries, "Is Apple (AAPL) a good investment for me right now?" The High-Level Orchestrator intercepts this request and immediately consults the Memory Module to retrieve the user's specific profile. In this instance, the memory reveals that the user is strictly focused on fundamental analysis and prefers text-based explanations over technical charts or visualizations. Acting on this personalized context,

the Orchestrator dynamically decides to activate only the Fundamental Analysis Agent to assess financial health and the News Analysis Agent to gauge market sentiment.

Once dispatched, the Fundamental Analysis Agent queries external APIs and the local SQL cache to retrieve key valuation metrics, while the News Analysis Agent performs a semantic search within the vector store to extract relevant news chunks and also query the external API for latest news data. These quantitative and qualitative streams are then converged at the Portfolio Agent. This agent synthesizes the raw data against the user's unique financial status, such as what stocks the user currently have and risk tolerance, to generate a tailored, text-based investment recommendation that directly addresses the user's fundamental focus without introducing other unwanted details.

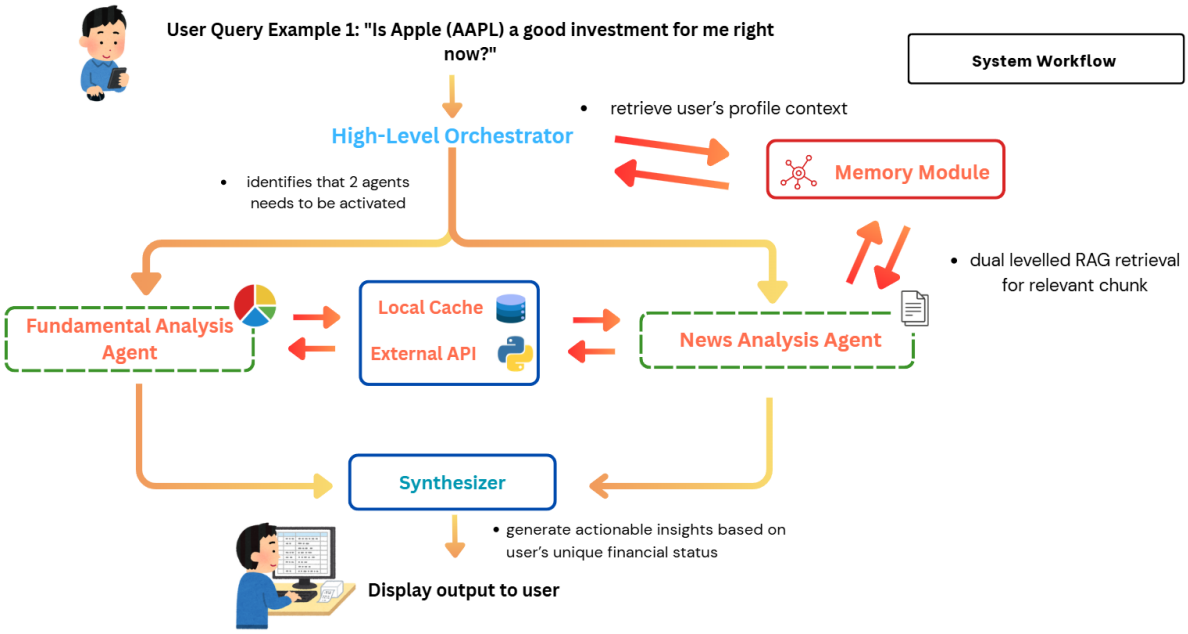


Figure 3.3: Example workflow for user's stock recommendation prompt

Chapter 4

System Design

4.1 Agent System Design

The Following sections will discuss on the agents being used within the system in detail, including their workflow and some implementation details

4.1.3 Orchestrator Agent

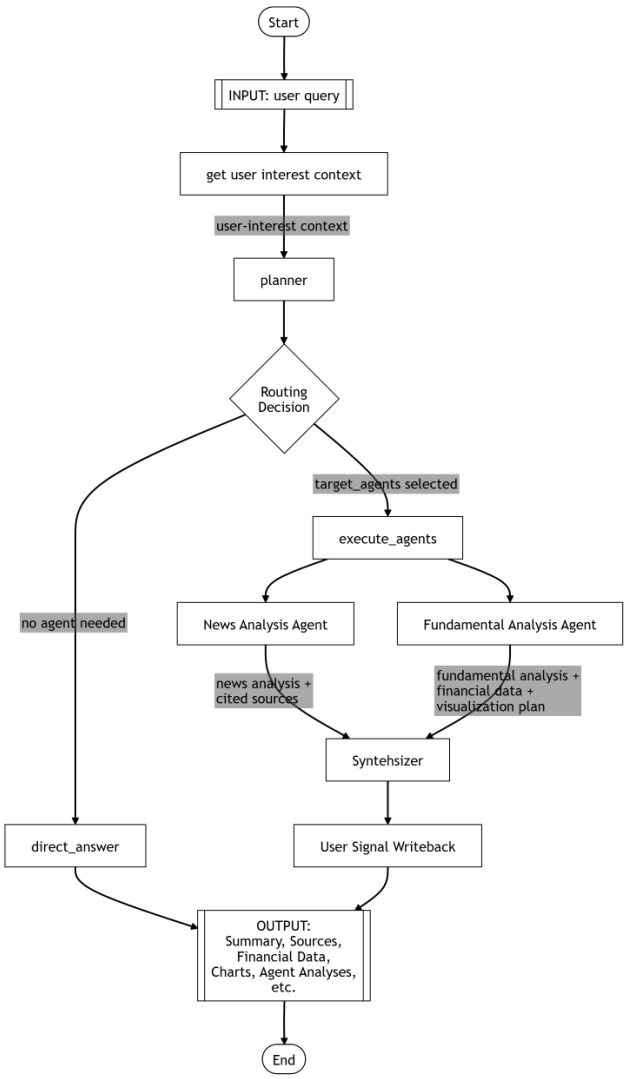


Figure 4.1: Orchestrator Agent's workflow diagram

The OrchestratorAgent acts as the central coordination layer for each user conversation turn. Its main responsibility is to interpret the user's request, decide whether the request can be answered directly or requires specialised analysis, route the task to the appropriate downstream agents, and synthesise their outputs into a final response. The orchestrator currently works with the news analysis agent and the fundamental analysis agent, but its design allows additional agents to be registered through the available agent list in the future.

At the beginning of each conversation turn, the orchestrator receives the latest user messages, conversation ID, user email, previous conversation turns and retrieved conversation memory. Before the graph is executed, the orchestrator loads the user context and portfolio holdings. The user context provides personalized information about the user's preferences, interests, and prior profile, while the portfolio block allows the final answer to consider the user's actual holdings instead of producing a generic stock analysis. Prior agent memory summaries and recent conversation history are also included too.

As seen in Figure 4.1, the orchestrator is implemented as a LangGraph workflow with five main nodes: `prepare_user_interest_context`, `planner`, `direct_answer`, `execute_agents`, and `synthesiser`. The workflow first prepares targeted user-interest context, then enters the planner node. Based on the planner's structured decision, the graph either returns a direct answer, dispatches the request to specialist agents, or proceeds directly to synthesis if no agent execution is needed.

The `prepare_user_interest_context` node supports personalization by building a targeted context block from the latest user message, baseline user context, portfolio information, and user-interest graph memory. This helps the planner understand whether the user's query relates to previous investment interests, learning needs, or personal preferences. For example, if a user has repeatedly shown interest in a particular sector or has expressed confusion about certain financial concepts, this context can influence how the orchestrator routes the query and frames the downstream agent goals.

The `planner` node is the main decision-making stage of the orchestrator. It uses an LLM with structured output to produce an `OrchestratorPlan`. This plan contains the detected tickers, selected target agents, per-agent goals, optional direct answer, and detected user-interest or learning signals. The planner receives the available agent descriptions, user context, prior agent memory summaries, retrieved conversation memory, the latest user message and so on.

If the planner determines that the user's request can be answered without specialist analysis, it returns a direct final answer. The workflow then routes to the `direct_answer` node and ends to avoid unnecessary computation for simple questions, such as general explanations or clarification-type prompts. However, if the planner identifies that financial analysis is required, it selects the appropriate specialist agents and provides each one with a specific goal.

Notably, before dispatching work to specialist agents, the orchestrator validates and enriches detected tickers. This step is important because both downstream agents depend on correct ticker symbols. If a ticker is invalid, ambiguous, or not recognized as a common equity, the orchestrator asks the user to confirm the intended symbol instead of continuing with potentially incorrect analysis. For valid equity tickers, the system enriches the ticker with company metadata such as company name, sector, industry, and business description. This company context is later passed to the selected agents so that their analysis is better grounded.

If specialised analysis is required, the orchestrator dispatches work to the agents selected by the planner. Each agent receives a tailored input containing its specific goal, the primary ticker, the conversation ID, the turn ID, agent-specific memory context, and the combined company context. These agent calls are executed concurrently so that news and fundamental analysis can run in parallel. If one agent fails, the orchestrator logs the error but continues processing any successful agent outputs, allowing the final response to still use the available results.

The synthesis stage gathers the outputs from all completed agents and prepares the final user-facing response. It collects each agent's analysis text, news citations, fundamental data, visualisation plan, raw display data, task completion status, and memory summaries. The synthesiser then combines these findings with the user context, portfolio holdings, recent conversation history, and retrieved conversation memory to produce a coherent final answer.

On a side note, the orchestrator also supports user-interest signal writeback. During planning, the LLM may detect investment-related signals such as whether the user has bought, sold, is interested in, or avoids a particular company, sector, event, or concept. It may also detect learning-related signals, such as whether the user is interested in, understands, or is confused about a financial concept. During synthesis, these signals can be converted into graph relationships and queued for memory writeback. This allows AlphaMesh to gradually refine its understanding of the user's investment interests and learning needs over time.

4.1.2 News Analysis Agent

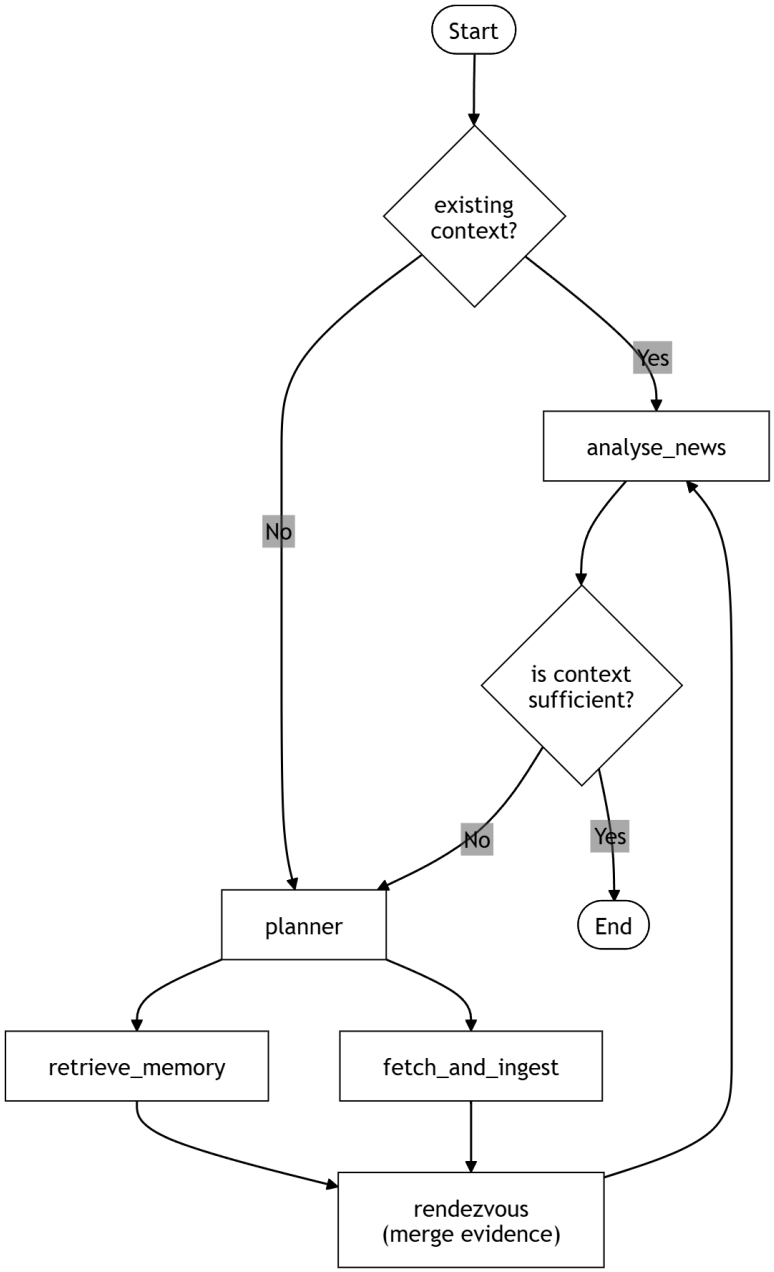


Figure 4.2: News Analysis Agent's workflow diagram

The News Analysis Agent is the specialized qualitative analysis agent in AlphaMesh. Its main responsibility is to process unstructured financial information such as recent company news, earnings-related articles, analyst updates, and other market-moving narratives. While the Fundamental Analysis Agent focuses on structured numerical data such as financial statements

and valuation metrics, the News Analysis Agent focuses on textual evidence that may affect investor sentiment, business outlook, near-term risks, or future catalysts. As seen in Figure 4.2, the agent is built as an iterative LangGraph workflow that combines external news retrieval, internal memory retrieval, article ingestion, chunk ranking, context sufficiency checking, citation construction, and final narrative generation.

The workflow begins when the Orchestrator Agent determines that the user's query requires qualitative analysis with information from external sources. The orchestrator then passes a structured input into the News Analysis Agent. The most important inputs are the analysis goal, ticker, date range, conversation ID, turn ID, and company context.

The analysis goal acts as the main instruction for the entire workflow and is reused during query planning, retrieval, sufficiency checking, and final analysis generation. The ticker helps focus the search on the relevant company, while the date range constrains the retrieval of recent articles so that the evidence matches the intended analysis period. Next, the conversation ID and turn ID are used for maintaining continuity across turns and for tracing graph-memory enrichment tasks. Lastly, the company context gives the agent additional background such as company name, sector, industry, or business description.

Before the LangGraph workflow starts, the agent prepares its initial state. The date range is first resolved and constrained to ensure that the retrieval period is valid. This is mainly due to the constrained date range from the chosen API. Then, the agent loads its conversation-level working memory, including previously relevant chunks, seen URL keys, and seen chunk IDs. These values are placed into the `NewsAgentState`, which acts as the shared state container across the graph. This state object stores information such as the planner decision, research logs, retrieved memory chunks, final cited sources and so on.

The graph starts with a conditional routing step. If relevant chunks already exist in the conversation working memory, the agent can proceed directly to the `analyse_news` node. This allows the agent to answer follow-up questions efficiently when useful evidence has already been collected in earlier turns. If no relevant working-memory chunks are available, the workflow begins at the `planner` node. This start-routing design reduces unnecessary retrieval and allows the system to reuse high-quality evidence from previous interactions whenever possible.

The `planner` node is responsible for deciding how the agent should retrieve the missing information. It uses an LLM with structured output to produce a `PlannerDecision`, which contains two main elements: the retrieval action and a list of domain-specific queries. The retrieval action can be either `newsapi` or `web_search`. The `newsapi` action is selected when structured financial-news discovery is likely to answer the query effectively, while `web_search` is selected when broader web coverage is needed. The planner can also generate queries across four domains: company, sector, market, and knowledge. A company query focuses narrowly on the target company or ticker. A sector query expands the search to industry-level developments. A market query captures broader macroeconomic or market-wide factors. A knowledge query is used when the user's request requires general financial concept explanation or background context. This design prevents the agent from relying only on a single ticker-based search and allows it to gather wider context when the user's question requires it.

As seen in Figure A.1 in Appendix, the planner also receives the current research iteration and a compact history of previous retrieval actions. This allows the agent to refine its search strategy across multiple iterations. For example, if the first iteration already retrieved company-specific news but the analysis stage later determines that sector-level information is missing, the next planner call can generate a more targeted sector query. As a result, the agent's research loop becomes progressively more focused instead of repeatedly performing the same search.

After planning, the graph branches into two parallel retrieval paths: `retrieve_memory` and `fetch_and_ingest`. The `retrieve_memory` node queries AlphaMesh's internal memory system using the planner's domain-specific queries. These queries are converted into a structured retrieval object containing company, sector, market, and knowledge queries, together with the active domains and original goal. The node then calls the retrieval service from the AlphaMesh's memory module to obtain relevant stored chunks with the dual-levelled RAG system. This allows the News Analysis Agent to reuse previously ingested articles and graph-connected supporting evidence instead of always depending on fresh external API calls, which also allows the shared knowledge of the system to grow along with the user as the user interacts with the system more and more.

The second branch, `fetch_and_ingest`, retrieves new articles from external sources. It reads the planner's selected action and query list, then fetches articles for each query. When the

planner produces multiple domain queries, the agent fetches them concurrently using `Asyncio` which improves the latency drastically.

NewsAPI is used as the default financial-news discovery source because it provides structured article retrieval with support for query strings, date ranges, language filtering, relevance sorting, and domain filtering. These features make it suitable for targeted company or market news retrieval, which is the majority use case of the system. However, NewsAPI results only contains snippets of the full content. Hence, to improve evidence quality, the system uses Trafilatura to scrape the full article body from each NewsAPI article URL where possible. If full extraction fails, the system falls back to the original NewsAPI content or description.

Moreover, Tavily is used as the broader web-search option. This is useful when financially relevant information may not be well covered by structured financial-news APIs. For example, the user may ask about an influential investor like Warren Buffet, a regulatory issue, a technology trend or a supply-chain event. Therefore, NewsAPI provides structured financial-news coverage, while Tavily broadens the agent's ability to search the wider web when the required evidence is outside standard financial-news feeds.

Once articles are fetched, the `fetch_and_ingest` node performs URL-level deduplication before ingesting the node into the memory module. Each article URL is converted into a canonical URL key. If the URL key has already been seen in the current conversation, the article is skipped, otherwise new articles are passed to the ingestion service, which stores them in AlphaMesh's memory system.

After both parallel retrieval branches complete, the graph enters the `rendezvous` node. It merges three evidence sources: existing working-memory chunks, newly fetched and ingested chunks, and chunks retrieved from internal memory. This merged evidence pool is then deduplicated and ranked via calling a reranker before being passed to the analysis stage.

The ranking process first performs baseline deduplication and filtering. Chunks are deduplicated by article and text content to avoid repeated evidence. Non-Tavily chunks are filtered based on the configured minimum relevance score, while Tavily chunks are preserved according to their own relevance metadata. After this baseline ranking, the agent calls the JINA reranker service to compare the user's query against candidate chunks more deeply than simple vector similarity. This is useful because a chunk that merely mentions a company may be less

useful than a chunk that directly explains a relevant earnings result, regulatory issue, demand trend, or market catalyst. If the reranker fails or returns no useful results, the agent falls back to the baseline ranking order, allowing the workflow to remain robust.

The `rendezvous` node then builds an article-grouped context block. Instead of presenting the analysis LLM with a flat list of disconnected chunks, the agent groups chunks by article. Each article group includes the article title, chunk aliases, publication date tags, relevance scores, and chunk text. This structure makes the evidence easier for the LLM to interpret because the model can see which chunks belong to the same source article. It also supports the later source-selection step, where the LLM chooses which chunks should support the final narrative.

The final major node is analyse_news. As seen in the system prompt for this node in Figure A.2 in Appendix, this node performs two separate tasks: context sufficiency evaluation and final narrative generation. If the context is insufficient and the maximum iteration limit has not been reached, the node returns a refined missing-information goal. The graph then routes back to the planner node, creating an iterative research loop. In the next iteration, the planner uses this refined objective to generate more targeted retrieval queries. This loop continues until the context becomes sufficient or the maximum number of research iterations is reached.

When the context is sufficient, or when the workflow reaches iteration limit, the agent proceeds to final narrative generation. The final LLM call is then given only the selected article-grouped evidence and is instructed to produce a concise investor-oriented analysis. The generated analysis covers key catalysts, risks, and near-term watchpoints that are all grounded in the chosen chunks. This design reduces hallucination because the final response is based on selected source chunks rather than the model's general knowledge.

Lastly, once it is all and done, the agent will call upon the AlphaMesh memory module's API to queue up a graph enrichment task by passing in its own analysis text, and specific entity and relationship extraction prompts, as well as the ids of the chunks involved in this analysis. The specific extraction prompt for the entity and relationship can be seen in Figure A.3 in Appendix. Entities will be extracted from each of the chunk individually before relationship between them are extracted to ensure that the entities extracted are properly grounded in actual text.

4.1.2 Fundamental Analysis Agent

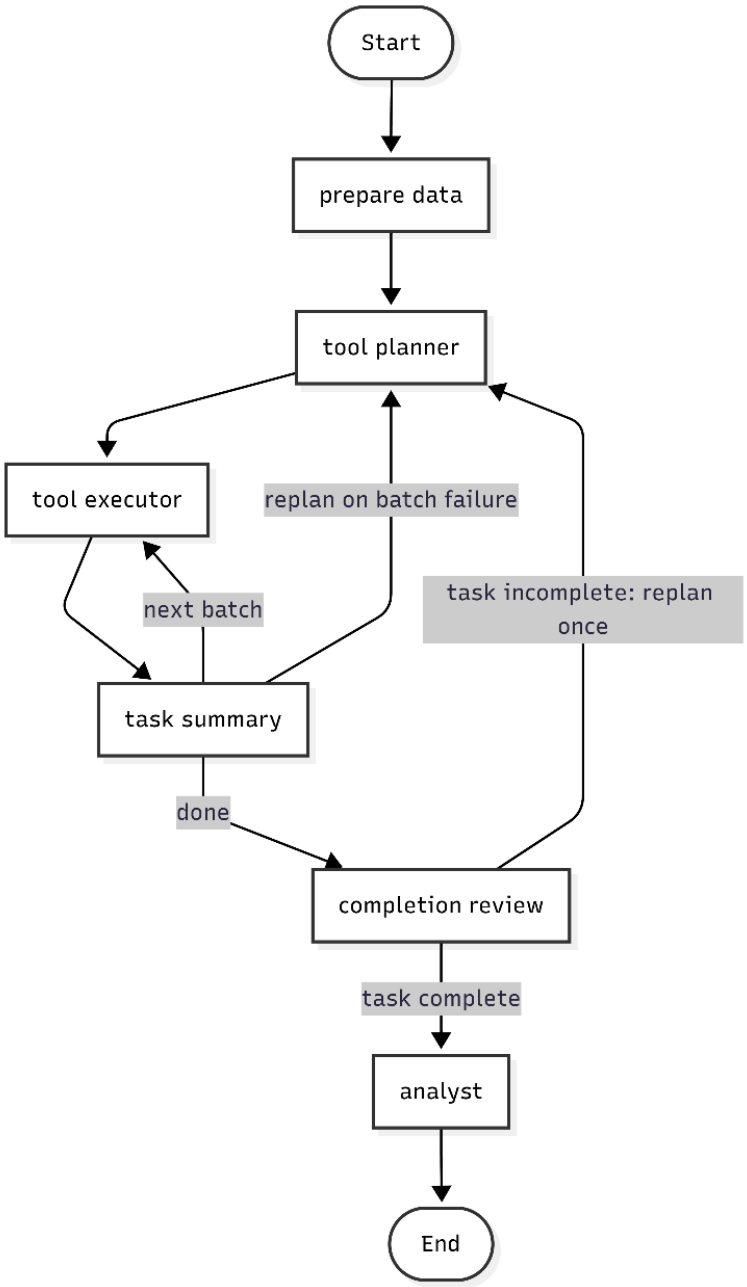


Figure 4.3: Fundamental Analysis Agent's workflow diagram

The Fundamental Analysis Agent is the specialized quantitative analysis component in AlphaMesh. Its main responsibility is to retrieve, prepare, compute, and interpret structured company financial data so that users can understand a stock's underlying financial condition. This agent focuses on numerical evidence such as income statement items, balance sheet values, cash flow data, stock price history, growth rates, profitability margins, liquidity ratios, solvency indicators, valuation multiples, and other derived financial metrics.

As seen in Figure 4.3, the workflow begins when the Orchestrator Agent determines that the user's query requires quantitative company analysis. The orchestrator then passes structured input data that includes user's analysis goal, the validated ticker, the start and end dates, the selected granularity (month or years), the conversation ID, and company context. Similar to the News analysis agent, the goal is used as the central instruction throughout the entire workflow. The ticker identifies the company whose financial data should be retrieved. The date range determines the reporting periods and stock price window to analyse and the granularity determines whether yearly or quarterly financial statements should be used. The conversation ID allows the agent to reuse cached financial data and previous tool results within the same conversation.

The LangGraph workflow consists of six main nodes: data_prep, tool_planner, tool_executor, task_summary, completion_review, and analyst. The first node, data_prep, is responsible for constructing the financial dataset that the rest of the workflow will use. It begins by resolving the required date range and determining the appropriate financial statement form type. If the requested granularity is yearly, the agent uses annual 10-K filings and ensures that the date range covers a sufficient historical window. If the requested granularity is quarterly, the agent uses 10-Q filings and generates the required year-quarter periods.

Before fetching new data, the data_prep node checks the Fundamental Agent's working memory to determine whether a suitable financial DataFrame has already been cached for the same conversation, ticker, granularity, and requested period range. If a valid cached DataFrame exists, the agent can reuse it directly. This is important for follow-up questions. For example, if the user first asks about a company's revenue trend and then asks about its margin trend, the agent should not need to fetch the same financial statements again, and hence this reduces redundant external data calls and improves response speed. Plus, if the working memory does not have the necessary information, there is also a SQLite database that acts as a long-term, cache for company data that further helps with the response speed.

If no valid data is available in cache and the SQL, then the data_prep node retrieves the required raw data through the financial data layer. The agent fetches financial statement data from SEC EDGAR using edgartools and stock price history from yfinance. After both data sources are returned, the financial statement data is trimmed to the requested date range and properly normalized before storing the processed data locally in SQLite database tables for future use.

In addition, computed time-series rows produced by the agent's tools are persisted back into SQLite database too. This means that derived metrics such as custom formulas or ratios may become reusable in later runs instead of being recomputed every time for every single user.

After data preparation, the workflow proceeds to the tool_planner node. This node uses an LLM with structured output to generate an IterativeToolPlan. The plan contains a complete ordered list of tool-call batches needed to answer the user's goal. This is one of the most important design features of the current Fundamental Analysis Agent. Instead of repeatedly asking the LLM what to do after every individual tool call, the planner is expected to produce the full dependency chain in one response. Calls that are independent of each other can be placed in the same batch, while calls that depend on earlier computed rows must be placed in later batches.

For example, if the user asks for a valuation analysis that requires free cash flow, the planner can first create a batch that computes Free Cash Flow using a custom formula. Then, in a later batch, it can compute valuation multiples that depend on the newly derived Free Cash Flow row. This design supports complex quantitative reasoning while avoiding excessive LLM calls. In the normal successful path, the planner LLM is called once, the executor processes the planned batches, and the analyst LLM writes the final interpretation. The planner is only called again if a tool failure occurs or if the completion reviewer later determines that the task is incomplete.

The planner prompt enforces several strict rules to improve execution reliability as seen in Figure A.4 inside Appendix. Every metric parameter must exactly match an available concept from the prepared financial DataFrame or a derived metric produced by an earlier batch. If a required metric is absent but can be derived, the planner must create the derived metric first before using it later. Tool calls within the same batch must be mutually independent. The planner is also instructed not to include a tool call if its required inputs are absent and cannot be derived. If the user's query does not require computation, the planner can return an empty batch list and instead specify the raw row labels that should be included in the final analysis.

The quantitative tools available to the planner are defined in the financial tool registry. Each tool has a fixed name, description, parameter schema, and execution method. The available tools include CAGR calculation, valuation multiples snapshot, profitability ratios, debt and solvency ratios, liquidity ratios, custom formula calculation, and period-over-period change

calculation. These tools allow the agent to compute common fundamental analysis metrics in a controlled way rather than relying on the LLM to perform numerical calculations directly.

After the planner creates the tool plan, the tool_executor node reads the current batch index from the graph state and executes the corresponding batch. If the batch contains multiple tool calls, they are executed concurrently. The executor retrieves each tool from the registry, validates the parameters using the tool's schema, and runs the calculation against the current financial DataFrame. When a tool succeeds and returns newly computed rows, the executor merges those rows into the financial DataFrame. It also records the names of newly computed rows and updates the available concepts list. This means later batches can use outputs from earlier batches as inputs. For example, a first batch may create a custom EBITDA row, and a later batch may use that EBITDA row to compute EV/EBITDA. Plus, as mentioned previously, the computed data rows also persisted into SQLite and written into the agent's Working Memory cache to be reused.

The executor also produces detailed audit logs. For each tool call, it records the tool name, parameters, success or failure status, error message, and so on. These logs are later used by the completion reviewer and are also be stored in working memory. This audit trail improves transparency and future tool planning because the system can inspect which computations were planned, which ones succeeded, and which financial rows were generated.

After each executor pass, the workflow enters the task_summary node. This node summarizes the completed batch by recording the task ID, batch index, batch reasoning, success status, and output row labels. If there are more planned batches and the previous batch completed successfully, the graph returns directly to the tool_executor node. If a tool failed, the graph routes back to the tool_planner node for a targeted replan. If no more batches remain, the graph proceeds to completion review.

The completion_review node runs after the planned tool batches have been exhausted. Its purpose is to evaluate whether the task has been sufficiently completed before the final written analysis is generated. As seen in Figure A.5 in Appendix, the reviewer receives the user goal, ticker, execution iteration count, executor logs, tool results, and a preview of the financial data. It then produces a structured CompletionReviewDecision indicating whether the task is complete, why it is complete or incomplete, whether replanning is needed. If the task is incomplete and the agent has not already used its completion-review replan, the graph can route

back to the planner with the reviewer's guidance. Plus, the completion_review node also prepares visualization-related output. The reviewer can propose chart instructions using supported chart types such as line, bar, area, scatter, stacked bar, stacked area, and pie. Each chart specification includes the chart type, data mode, snapshot period, title, row labels, grouping setting, and rationale.

After completion review, the workflow proceeds to the analyst node. This node generates the final analysis. The analyst prompt includes the user's goal, ticker, company context, agent memory context, selected financial data, and tool execution summaries. The analyst LLM is instructed to write a comprehensive evidence-based analysis that highlights key trends, positives, risks, and interpretations of the computed metrics. It is also instructed to explain derived metrics, interpret valuation multiples, and convert large numbers into human-readable form. This allows the final response to be more useful to retail investors, who may not be comfortable interpreting raw financial rows without explanation.

4.1.3 Agent Working Memory

In addition to the long-term memory system stored in ChromaDB and Neo4j, AlphaMesh also uses agent-level working memory to support short-term continuity within a conversation. As shown in Figure 4.4, both the News Analysis Agent and Fundamental Analysis Agent follow a similar working-memory workflow. When an agent begins a new run, it first checks whether there is existing conversation-level working memory for the current `conversation_id`. If available, relevant previous-turn information is loaded and inserted into the agent's runtime context. During the agent workflow, newly useful information is accumulated, such as retrieved chunks, seen article identifiers, computed financial rows, tool execution logs, and so on. After the agent completes its analysis, the working-memory manager persists a compact summary of the finalized turn. This allows future turns in the same conversation to reuse recent context without repeatedly fetching the same data, recomputing the same metrics, or inserting the entire previous analysis into the prompt.

For the News Analysis Agent, working memory is mainly used to manage article evidence across turns. The `NewsWorkingMemoryManager` stores recent relevant chunks, source URLs, canonical seen URL keys, and seen chunk IDs for each conversation. This is important because

news analysis often involves repeated follow-up questions about the same company, event, or market situation. By remembering seen URL keys, the agent can avoid fetching and ingesting the same article again. By remembering seen chunk IDs, it can avoid duplicating evidence that has already been used in the current conversation.

After a news analysis turn is finalized, the manager records the selected chunks and source URLs into turn-level memory, while also maintaining capped histories of seen URLs and chunks. This allows later News Agent runs to reuse high-signal evidence directly, or to merge it with newly fetched and retrieved chunks during the `rendezvous` stage. As a result, the News Agent becomes more efficient and context-aware during multi-turn research conversations.

For the Fundamental Analysis Agent, working memory is focused less on articles and more on structured financial computation. The `FundamentalWorkingMemoryManager` stores conversation-level records of tool calls, batch execution summaries, computed row labels, task completion status, and cached financial DataFrames by ticker and granularity. This is useful because fundamental analysis often requires expensive preparation steps, such as retrieving EDGAR financial statements, loading stock price history, normalizing reporting periods, and computing derived metrics.

When the user asks a follow-up question about the same ticker and granularity, the agent can reuse a cached financial DataFrame if it covers the requested period range. The manager also records previous tool execution details, including successful and failed calls, output row labels, summaries, reasoning, scalar results, and added row counts. These records can be rendered into later planner context so that the Fundamental Agent avoids repeating calculations that were already completed. Therefore, while News Agent working memory preserves evidence continuity, Fundamental Agent working memory preserves data-preparation and computation continuity.

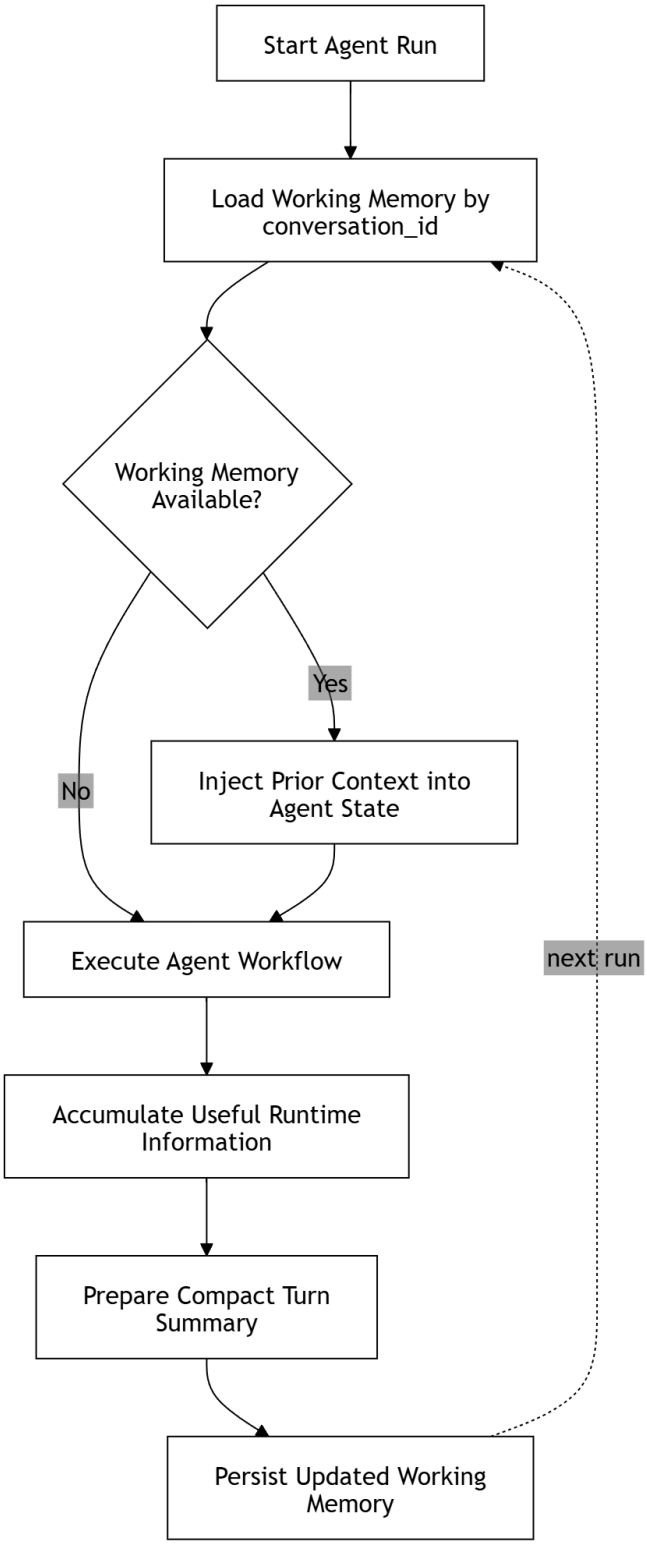


Figure 4.4: General agent working-memory workflow.

4.2 Ingestion Pipeline and Knowledge Graph Construction

The memory ingestion system is responsible for converting useful information gathered or generated by AlphaMesh's agents into persistent and progressively evolving memory. It supports two main forms of ingestion. The first is article ingestion, where newly fetched news articles are stored as retrievable evidence in both ChromaDB and Neo4j. The second is graph enrichment, where the final analysis produced by agents is converted into structured entities and relationships through an asynchronous graph queue processing backend. This design allows AlphaMesh to preserve both raw source evidence and higher-level interpreted knowledge without forcing all memory processing to happen directly during the user-facing response path, which will slow down the response time.

4.2.1 Global Knowledge Graph Structure

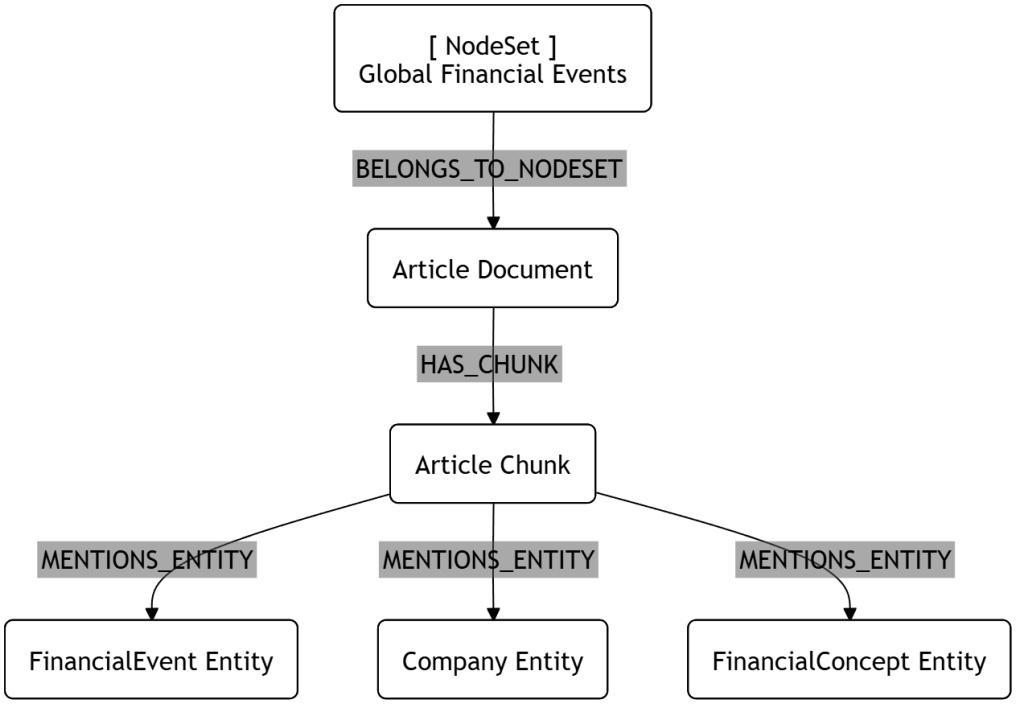


Figure 4.5 Graph Structure for Ingested Articles

The graph memory is organized using NodeSets, which act as stable grouping anchors for different types of memory. As shown in Figure 4.5, article documents and chunks are assigned to the Global Financial Events NodeSet. This provides a shared anchor for news-related financial evidence. These NodeSets help organize the graph so that retrieved chunks, extracted

events, financial concepts, and future relationships are inherently connected together for easier traversal.

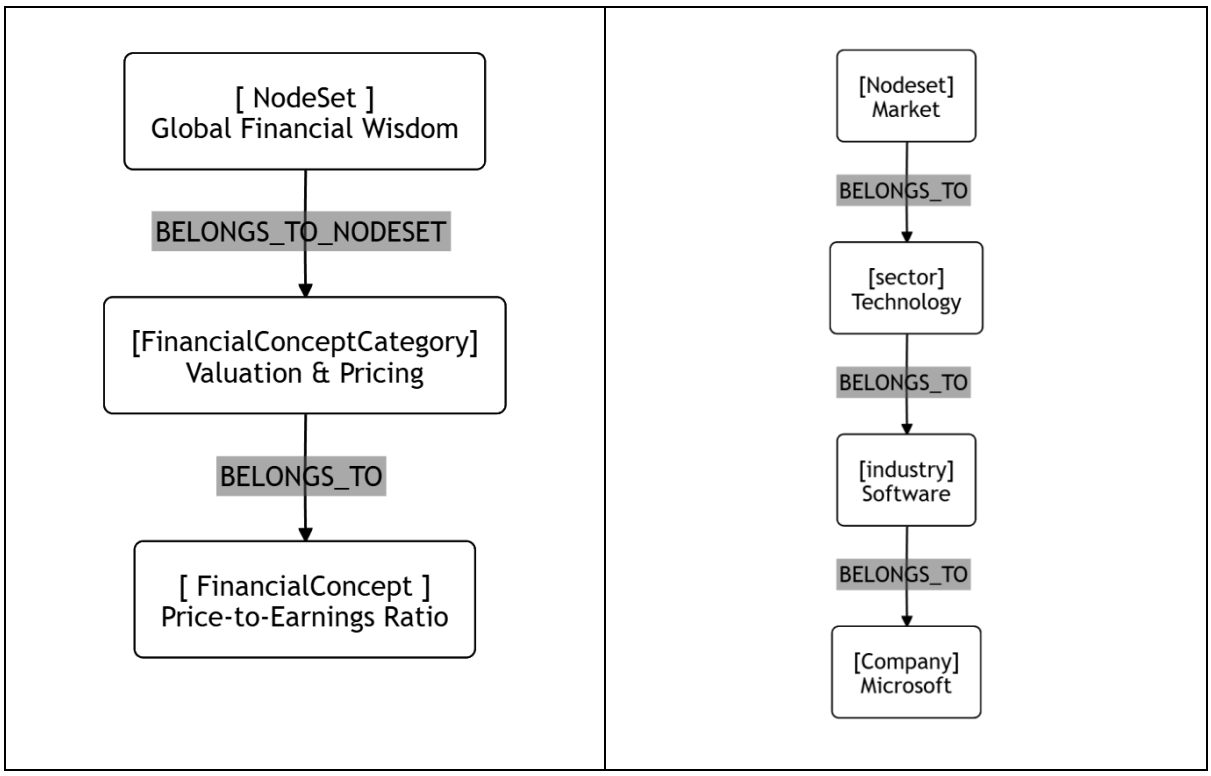


Figure 4.6 Graph Node Structure for identified companies and financial concepts

Moreover, the system also supports taxonomy structure for the companies. As shown in Figure 4.6, the graph contains a canonical Market entity, and the main sector entities are linked to this Market entity through `BELONGS_TO` relationships. The Market entity therefore functions as a singleton that groups up all different predefined sectors. There are many sectors attached to the market entity, many industries to each sector, and many companies to each industry. These are determined at runtime using yahoo Finance's official API, so this chain of relationships are fixed and grounded.

Additionally, the system also initializes FinancialConceptCategory nodes under the singleton node Global Financial Wisdom NodeSet. Later, extracted FinancialConcept entities can be connected to these categories. This predefined structure gives the graph stable anchors for broad market, sector, and concept-level traversal. Plus, it also helps immensely to give the graph more structure and ease of interpretability during development.

4.2.5 User-specific knowledge graph Structure

The graph writes pipeline separates domain relationships from user-scoped relationships. Domain relationships involve financial entities such as Company, FinancialEvent, FinancialConcept, Sector, Industry, and Market. These relationships are resolved through the entity resolver before being written into the main financial graph. User-scoped relationships involve user-specific nodes such as UserInterestDomain, UserInterestEdge, UserInterestEvent, and SessionNode, and are handled separately so that personalization data does not get mixed directly into the global financial knowledge structure.

A smaller but important part of the ingestion system is user-specific graph handling. This path is triggered by the Orchestrator Agent rather than by the specialist agents. During planning, the orchestrator may detect investment signals such as whether the user has bought, sold, is interested in, or avoids a company or sector. It may also detect learning signals, such as whether the user is interested in, understands, or is confused about a financial concept. During synthesis, these signals are converted into user-specific graph relationships and enqueued through the same graph queue infrastructure.

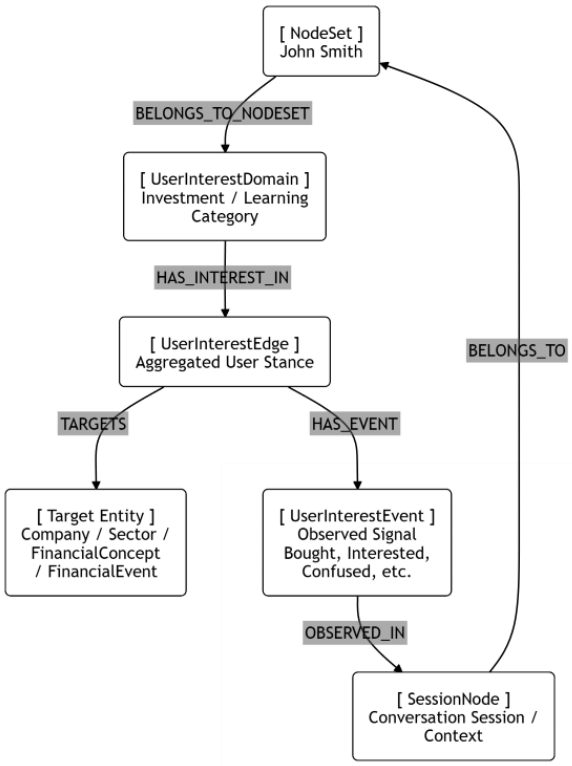


Figure 4.7: User-specific graph Structure

The structure of the user-specific graph is shown in Figure 4.7. A private user NodeSet contains `UserInterestDomain` nodes, which group interests by investment or learning category. Each domain connects to `UserInterestEdge` nodes, which represent the user's current relationship with a target entity. It points to the target financial entity and is also linked to `UserInterestEvent` nodes that record individual observations from specific turns. Each event is connected to a `SessionNode`, preserving where and when the signal was observed. This allows the system to store both the current aggregate user stance and the historical evidence behind that stance. But, strictly speaking, the SessionNode and UserInterestEvent are mainly there for transparency and traceability purposes.

This user graph design allows AlphaMesh to track aggregate user preference by making use of the UserInterestEdge. Each interest edge represents the user's current stance toward a target financial entity, such as a company, sector, financial event, or financial concept. Positive signals reinforce an interest edge, while negative signals can invalidate or weaken it. Besides, the system also updates the in-memory user context cache when user-signal writeback occurs, allowing the orchestrator to use the latest personalization context in future turns without waiting for a full graph retrieval process. This makes the user graph useful not only for long-term memory, but also for short-term personalization during ongoing conversations.

4.2.2 Article Ingestion and Entity Extraction

The first ingestion path begins inside the News Analysis Agent. After the agent fetches new articles from NewsAPI or Tavily, its `fetch_and_ingest` node sends the newly fetched articles to the ingestion service. The article ingestion pipeline is handled by the `ArticleIngestor`. Before storing a new article, the ingestor checks whether the article's source URL already exists in ChromaDB. If the URL has already been stored, the system avoids re-ingesting the article and instead returns the existing chunks. If the article is new, the article is passed to the `ArticleChunker`, which converts the article into document metadata and smaller text chunks. Each chunk is assigned a document ID, chunk ID, chunk index, source URL, publication timestamp, and other metadata required for retrieval.

After chunking, the article is written into both Neo4j and ChromaDB. In Neo4j, the full article is represented as a lightweight document node, while each text segment is represented as a chunk node. In ChromaDB, the chunk text is inserted together with metadata such as article title, source URL, publication date, extraction status, and so on. This dual-store design is

important because the two databases serve different purposes. ChromaDB stores the textual content for semantic retrieval, while Neo4j stores the structural representation needed for graph traversal and relationship-based memory. Newly created chunks are also marked with an extraction status of `PENDING`, meaning they have been stored but have not yet completed entity extraction. Only the final selected chunks by the agent to perform the analysis will be selected for extraction to ensure that LLM resources are spent on actually useful chunks.

4.2.3 Graph Enrichment Pipeline

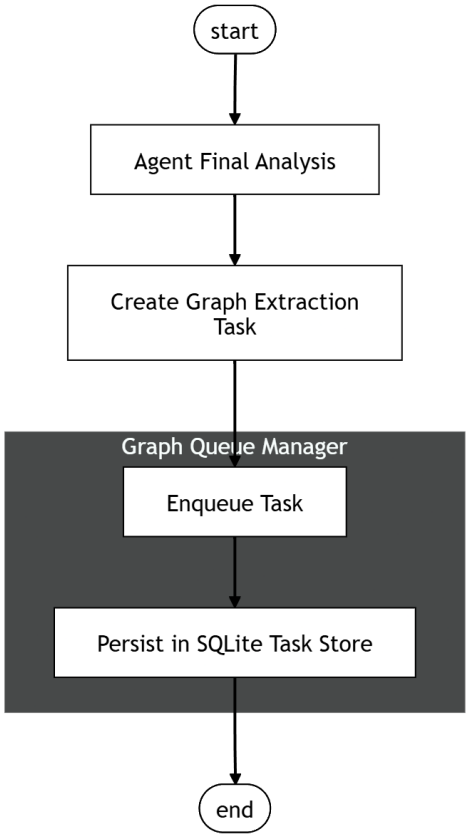


Figure 4.8: The overall pipeline for enqueuing graph enrichment tasks by agents

Next, moving onto the second ingestion path is the graph enrichment through the graph queue processing backend. This path is triggered after an agent has produced a final analysis. For instance, after the News Analysis Agent generates its final grounded news analysis, it creates a scoped graph extraction task using the flow as seen in Figure 4.8. The task contains the final analysis text, the source chunk IDs, allowed entity types, allowed relationship types, and the relevant extraction prompts. The Fundamental Analysis Agent also creates a graph extraction task after producing its final quantitative analysis, but with a narrower extraction scope. This

allows both agents to use the same graph queue system while still applying different specific extraction rules.

The `GraphQueueManager` coordinates this enrichment process. When a task is enqueued, the manager validates the task type, resolves its allowed entity and relationship scope, and persists the task to SQLite, and places it into a per-conversation queue worker. Persisting queued tasks in SQLite is important because graph enrichment happens after the main response is returned, and pending tasks should not be lost if the application restarts.

Each active conversation has its own queue worker. The worker groups tasks by turn ID and only processes them when the orchestrator flushes the turn. This design allows multiple agents to enqueue graph tasks during the same user interaction before graph writing begins. For example, one turn may produce a news extraction task, a fundamental extraction task, and a user-interest writeback task. These can be collected under the same turn and processed together after the user-facing response has been produced.

Once a batch of queued tasks is ready, it is passed to the `GraphWritePipeline`. For scoped extraction tasks, the pipeline first performs chunk-entity extraction. It retrieves the referenced chunk text from ChromaDB, uses an LLM to extract entities from those chunks, resolves the extracted entities, and links them back to the chunks in Neo4j using relationships. Once entity extraction is completed, the chunk status is updated from `PENDING` to `EXTRACTED`. This prepass is especially important for scoped news extraction, because the later relationship extraction step is restricted to entities that were actually found in the supporting article chunks.

After chunk entities have been extracted, the pipeline performs relationship extraction from the agent's final analysis text. For scoped tasks, the relationship extractor is given a known-entity context built from the referenced chunks, and extracted relationships are filtered so that their endpoints must match those known entities. This prevents the graph from being enriched with unsupported relationship endpoints that are not grounded in the evidence used by the agent. In other words, the LLM is not allowed to freely hallucinate new companies, events, or concepts during scoped extraction and instead it must form relationships only between entities that were previously extracted from the supporting chunks.

Moreover, the extracted relationships are also filtered according to the task's allowed entity types and relationship types. For instance, relationship types are also constrained to finance-

relevant links such as AFFECTS, CAUSED_BY, CORRELATED_WITH, RELATED_TO, and so on. These constraints ensure that the graph remains focused on useful financial reasoning rather than becoming polluted with generic or irrelevant relationships.

4.2.4 Entity Resolution Pipeline

Before writing relationships into Neo4j, the pipeline must resolve each relationship endpoint into a canonical graph entity. This is handled by the `EntityResolver`, which attempts to deduplicate and match entities before creating new nodes. As seen in Figure 4.9, the resolver follows a staged process: it first checks an in-memory resolution cache, then checks for exact entity IDs and exact name matches in Neo4j, then tries fuzzy matching and company-alias matching, then performs vector similarity search through the entity embedding store, and only creates a new entity when creation is allowed and no existing match is found.

This resolution stage is important because financial text often refers to the same entity in different ways. For example, a company may appear as “Apple”, “Apple Inc.”, or “AAPL” depending on the source. The resolver includes fuzzy and company-alias matching to reduce duplicate company nodes. It also uses a positive cache for successful resolutions and a short-lived negative cache for unresolved entities, reducing repeated backend lookups during batch graph processing. This not only makes graph enrichment more efficient, but also improves the consistency of the graph structure over time.

After entity resolution, the pipeline deduplicates relationships based on the resolved source entity, relationship type, and target entity. If duplicate relationships are found, their confidence and supporting properties are merged. This increments their confidence (weight) that allows them to be ranked higher in future retrieval and making them more relevant.

Besides, the same pipeline can also handle user-scoped relationships, but these are separated from normal domain relationships.

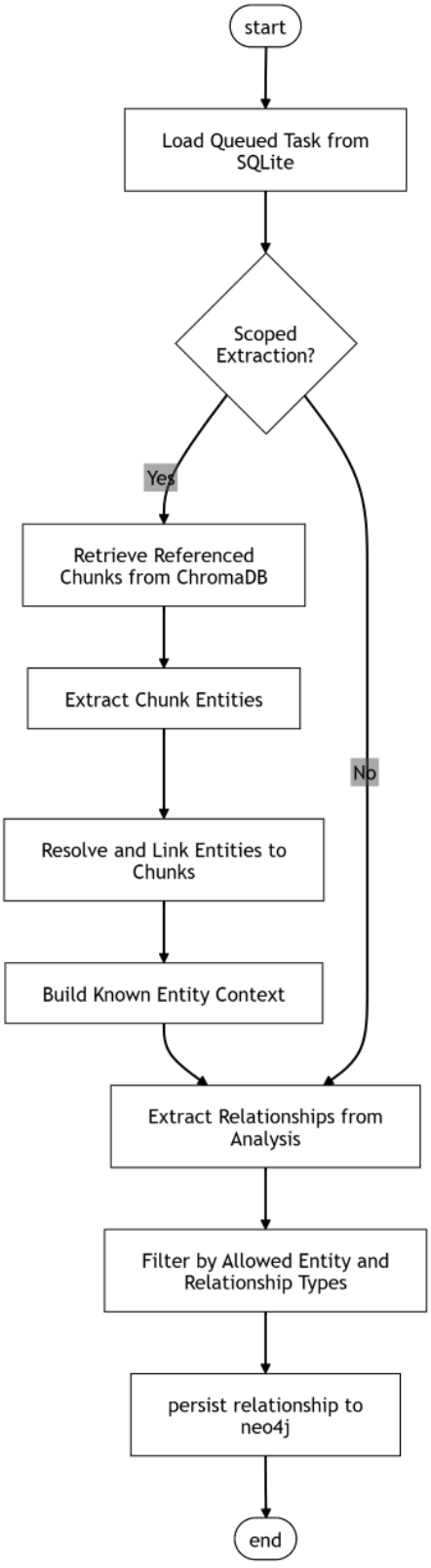


Figure 4.9: Pipeline for processing the enqueued graph enrichment tasks

4.3 Memory Retrieval

The retrieval system is responsible for reusing the memory that has been created through article ingestion and graph enrichment. In AlphaMesh, the system uses a dual-store retrieval design that combines ChromaDB vector search with Neo4j graph traversal. ChromaDB provides semantic recall over stored article chunks, while Neo4j allows the system to expand from retrieved chunks into related companies, financial events, sectors, markets, and financial concepts. This allows AlphaMesh to retrieve not only chunks that directly match the user's wording, but also evidence that is connected through the graph for a more comprehensive well-rounded analysis. Notably, the retrieval system is mainly used by the News Analysis Agent as well as the orchestrator agent.

4.3.1 Dual-Store Retrieval

As shown in Figure 4.10, the graph contains four main nodes: `vector_seed`, `extract_seed_entities`, `select_neighbor_frontier`, and `fetch_frontier_chunks`. The workflow begins with the user query and first performs vector search against ChromaDB. It then uses the retrieved chunks to identify seed entities in Neo4j, expands from those entities into graph neighbors, and finally fetches additional chunks connected to the selected frontier entities. In a nutshell, this creates a retrieval process that begins with semantic similarity but can continue through graph relationships.

The first node, `vector_seed`, retrieves the most semantically similar chunks from ChromaDB. These chunks form the initial evidence pool or also known as the vector seed nodes which functions as the starting point for the graph traversal. Their chunk IDs are also recorded as visited chunks so that later graph retrieval does not repeatedly add them again.

The second node, `extract_seed_entities`, connects the vector search result to the graph. It takes the vector-seeded chunks and queries Neo4j for entities mentioned by those chunks. These entities become the initial graph frontier, which is then used in the third node `select_neighbor_frontier`. This next node expands from the current frontier to neighboring graph entities. These neighbors may be companies, financial events, financial concepts, sectors, industries, markets, and so on. However, not every neighboring entity should be expanded. The traversal policy scores and selects the most useful neighbor entities before they become the

next frontier. This prevents the graph traversal from drifting too far from the user's original query, and also prevents the search space from growing out of control.

The fourth node, `fetch_frontier_chunks`, retrieves chunks connected to the selected frontier entities. These chunks are added to the accumulated retrieval result as graph-sourced chunks. After frontier chunks are fetched, the retrieval graph decides whether traversal should continue. If the maximum iteration limit has not been reached and useful frontier entities remain, the graph loops back to `select_neighbor_frontier`. Otherwise, retrieval ends and returns the accumulated chunks.

To ensure that the global financial retrieval path remains separated from user-specific memory, the current implementation includes fail-safe filtering at the shared Neo4j adapter layer. A shared user-scoped relationship denylist is used to prevent user-interest relationship types from being traversed during normal global retrieval. In the dual-store retrieval path, neighbor traversal excludes user-scoped relationship types and also excludes entities that contain a user_email field. Similarly, chunk fetching from selected entity frontiers excludes user-owned entities. This prevents the global financial retriever from accidentally crossing into private user-interest graph structures while expanding through Neo4j.

The retriever also supports multi-domain comprehensive retrieval. This is especially important for the News Analysis Agent because the planner may generate separate company, sector, market, and knowledge queries. Instead of forcing all information needs into one generic query, the retriever runs the main retrieval graph independently for each active domain. These runs can execute concurrently, and each retrieved chunk is tagged with its domain metadata. The domain results are then reranked, deduplicated, and merged into a single memory context. This gives the News Agent a broader and more organized evidence pool for later analysis.

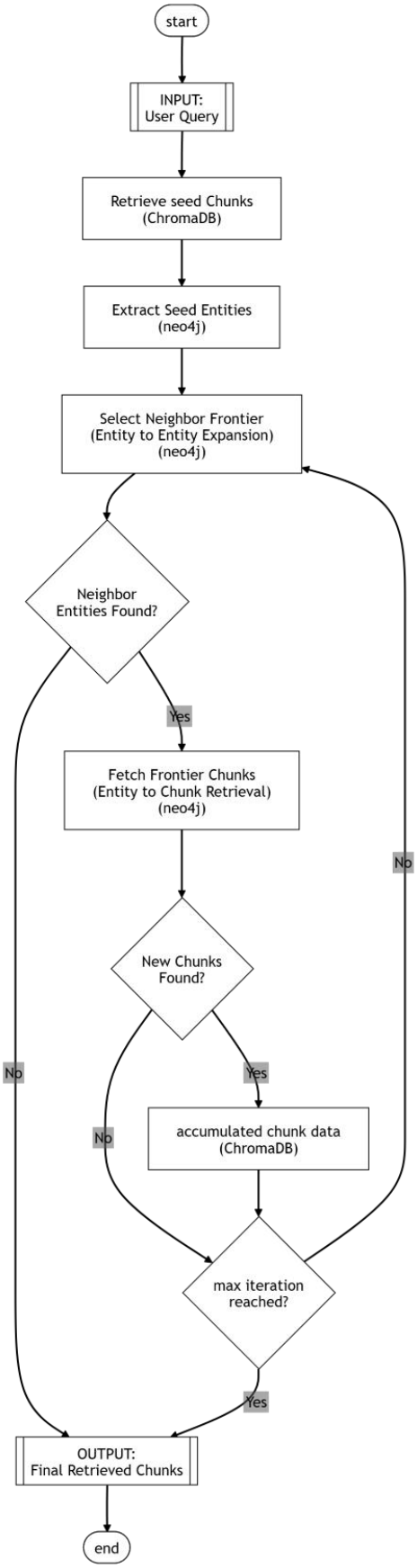


Figure 4.10: the Dual-levelled Retrieval System involving vector and graph store

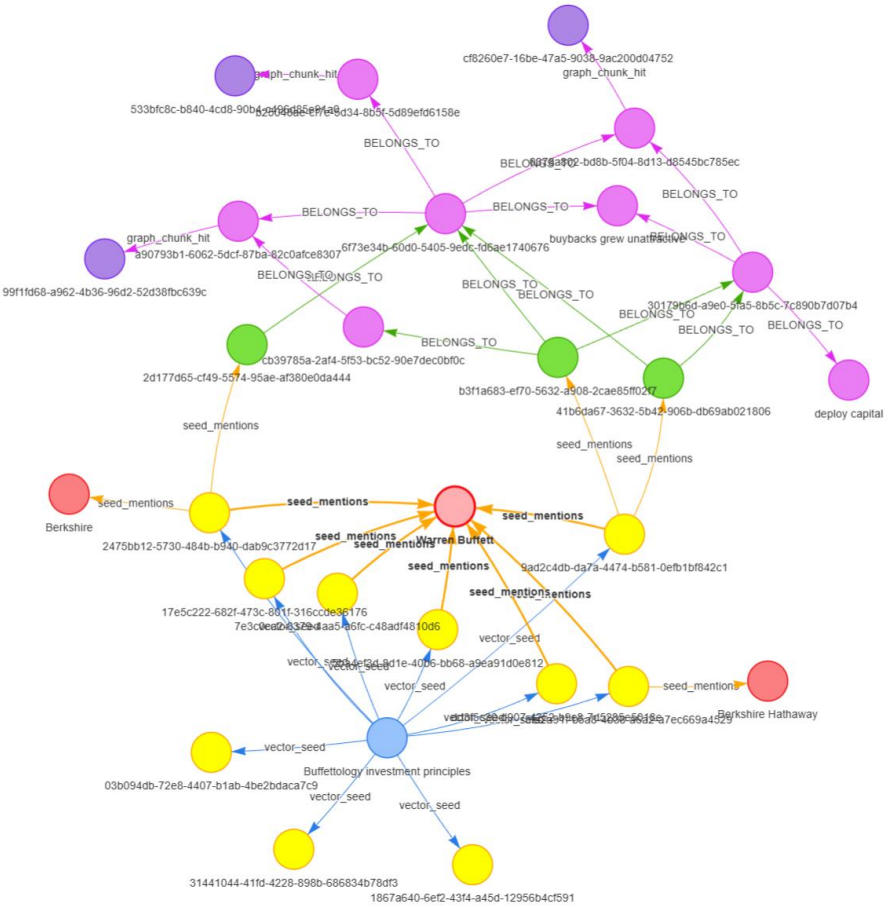


Figure 4.11: Graph Retrieval Visualization

Finally, the also retriever includes an optionally enabled tracing mechanism for debugging and evaluation. During retrieval, it can record trace events for vector seeding, seed entity extraction, neighbour expansion, frontier chunk fetching, and prefilter output. These events can be exported as HTML or GraphML visualizations using a NetworkX-based trace sink. This makes the retrieval process more transparent, as the developer can inspect how a query moved from the original vector results into graph-expanded chunks and how the final memory context was formed. An example of the graph expansion and traversal can be seen in Figure 4.11, with the yellow and deep purple nodes representing the chunk nodes, and the nodes of other colours representing entities of different types.

This design allows AlphaMesh to retrieve information that is directly relevant to the user's query while also discovering related financial context that may not share the exact same wording, but is connected conceptually through the underlying financial knowledge graph.

4.3.2 Reranking Pipeline

Since retrieval can return more chunks than the agent should use directly, AlphaMesh applies a reranking pipeline before returning the final memory context. The reranking pipeline has two stages. The first stage is the `CompositePrefilter`, which is a local and low-cost scoring step. It deduplicates chunks by chunk ID and computes a composite score based on embedding similarity and graph depth. This means chunks that are both semantically relevant and close to the original graph seed are prioritized.

The second stage is optional and allows the system to call the Jina reranker API. Jina acts as a cross-encoder reranker, comparing the query and candidate chunk text more directly than ordinary vector similarity. This is useful because vector search may retrieve chunks that are broadly related but not necessarily the best evidence for the specific user question. A cross-encoder reranker can better judge whether a chunk directly answers the query or only mentions a related topic.

4.3.3 Traversal Selection Policy

$$\boldsymbol{Traversal\ Score(c) = R \times E \times \frac{1}{1 + h_c} \times (1 + 0.2 \times lexical\_overlap)}$$

Where:

- R = relationship weight, based on the edge type
- E = entity weight, based on the candidate entity type.
- h = current graph hop depth.
- lexical_overlap = lexical overlap between the user query and the candidate entity name.
- 0.2 = small relevance multiplier that lets query overlap improve the score without overpowering graph structure.

The graph traversal selection policy controls which neighboring entities should be expanded during retrieval. This is necessary because knowledge graphs can become highly connected. Without a selection policy, traversal could quickly move from a relevant company to unrelated

noise or generic concepts. The `TraversalPolicy` prevents this by limiting the number of neighbour candidates per source entity and scoring each candidate before it can become part of the next frontier.

As seen in the equation above, the traversal score is based on structural importance, hop-depth decay, and query relevance. Structural importance is calculated using the relationship type weight and entity type weight. Each type of predefined relationship that were extracted during the knowledge graph construction stage have assigned weights to them to mark their relevance and or importance. The exact details is listed in the Figure A.6 in Appendix. For example, relationships such as AFFECTS and CAUSED_BY are more meaningful than a generic RELATED_TO relationship. Similarly, entities such as Company and FinancialEvent are more directly useful than broader nodes such as Sector or Market.

Hop-depth decay reduces the score of entities reached through deeper graph expansions. This keeps retrieval grounded near the original vector-seeded evidence. Without this penalty, later hops could dominate retrieval even if they are only loosely connected to the original query. The policy also adds a small query relevance bonus using lexical overlap between the query and the candidate entity name. This is a lightweight way to preserve the user's original intent without requiring extra embedding or LLM calls. After scoring, the traversal policy selects the top unique neighbour entities as the next frontier.

4.3.4 User-Specific Interest Retrieval

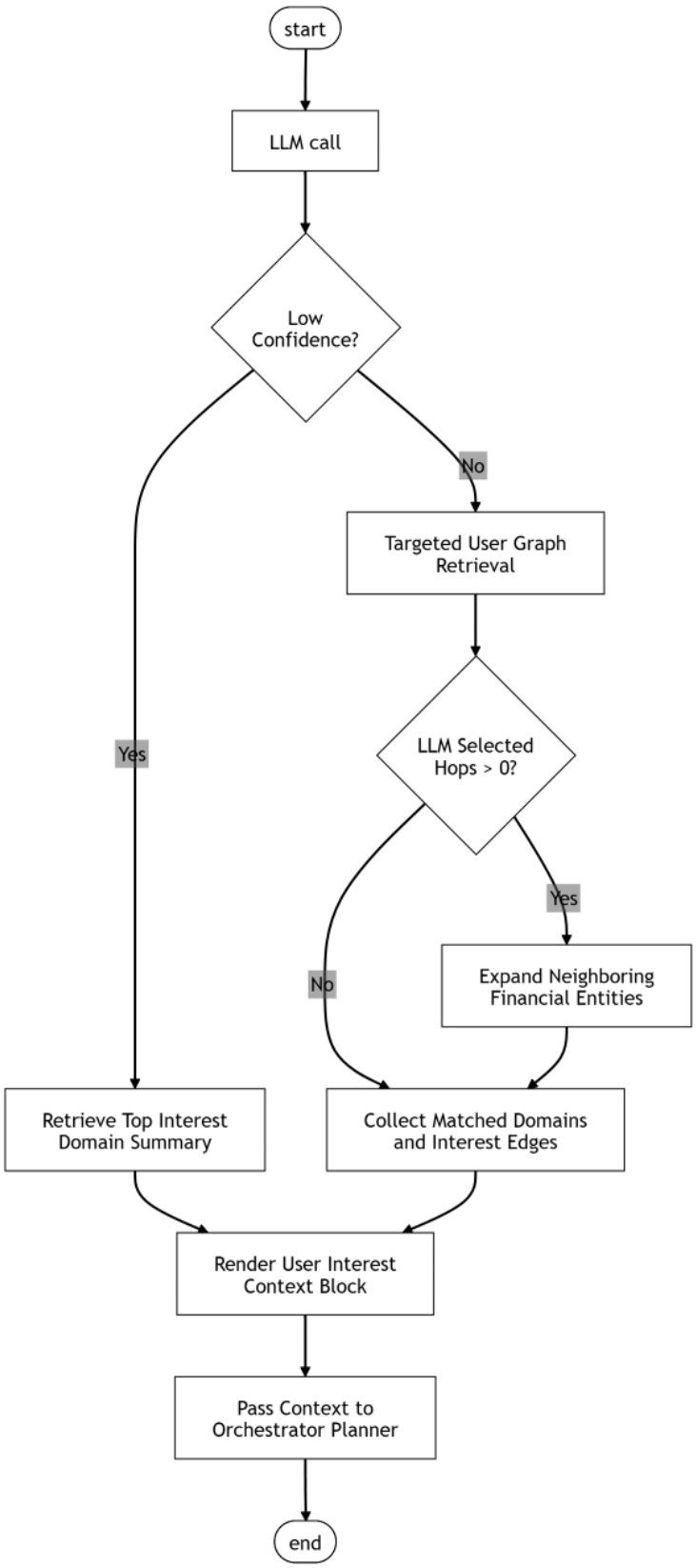


Figure 4.12: User specific graph retrieval workflow

In addition to dual-levelled RAG memory retrieval, AlphaMesh also performs user-specific interest retrieval. This retrieval path is separate from the dual-store article retrieval system. Instead of retrieving article chunks, it retrieves compact personalization context from the user-interest graph.

As seen in Figure 4.11, the user-specific retrieval process begins with the latest user message, baseline user context, and portfolio block. An LLM is used to generate a compact `UserInterestQuerySpec`. This query specification indicates whether the user's current request is related to investment interests or learning interests, whether a specific category is involved, which target entities should be searched, how many graph hops should be expanded, whether broad fallback should be used, and whether the user is expressing risk or avoidance intent. The hop value is clamped between 0 and 2 so that user-interest retrieval remains bounded.

If the generated query specification is too broad or low-confidence, the retrieval falls back to just retrieving a compact summary of the user's top interest domains, including domain type, category, aggregate positive weight, edge count, last changed time, and so on. This fallback is useful when the user's message is general, such as asking for overall investment guidance, because there may not be a clear single entity to target.

If the query specification is specific enough, a targeted graph retrieval is performed. It queries the user-interest graph using the user's NodeSet, domain type, category, target entities, hop count, and risk or avoidance flag. The result contains matched interest domains and top interest edges. If graph hops are enabled, the result may also include expanded neighboring financial entities related to the user's known interests.

The user-specific retrieval expansion path is also designed to preserve privacy boundaries between users. During expanded user-interest retrieval, the query blocks user-scoped relationship types from being used as generic expansion paths. It also blocks user-scoped node labels and prevents traversal through nodes that belong to another user. This ensures that user-specific retrieval only exposes personalization context for the current user, while still allowing limited expansion from that user's interest edges into nearby global entities. In this way, AlphaMesh preserves a clear boundary between the globally shared knowledge graph and private user-interest memory.

Chapter 5

System Implementation

5.1 Hardware Setup

The entire development process has taken place with the use of personal laptop only in terms of hardware usage.

Table 5.1 Specifications of laptop

Description	**Specifications**
Model	Legion Slim 5
Processor	Ryzen 7 8845HS
Operating System	Windows 11
Graphic	NVIDIA RTX 4070 Laptop GPU
Memory	16GB DDR5 RAM
Storage	1TB SSD

5.2 Development Tools and Technologies

AlphaMesh was implemented as a web-app consisting of a Python backend and a browser-based frontend. The backend was developed using Python 3.11 and exposes its services through a FastAPI application. The FastAPI server is executed using Uvicorn, meanwhile the frontend is managed separately using the Node.js and npm toolchain, where the development interface is launched using npm. This separation allows the backend agent system, databases, external APIs, and frontend user interface to be developed and tested independently.

The backend is responsible for executing the multi-agent workflow, retrieving financial data, managing memory stores, processing user portfolio information, and streaming agent results back to the frontend. The frontend provides the user-facing chat interface, portfolio page,

historical conversation view, and visual display of financial analysis outputs. Table 5.1 lists the main software tools, libraries, and APIs used in the implementation of AlphaMesh.

Table 5.2: Selected software tools, libraries, and APIs for Development

Category	**Tool / Library / API**	**Justification**
Development Environment & Runtime	Visual Studio Code	Provides a flexible development environment with Python, JavaScript, terminal, debugging, and Git integration.
	Python 3.11+	Supports AI agent development, asynchronous programming, financial data processing, and mature LLM ecosystem libraries.
Frontend Development & Prototyping	Node.js and npm	Installs frontend packages and runs the frontend development server through npm run dev.
	Stitch	Supports rapid design exploration for the chat interface, dashboard, portfolio page, and analysis views.
Backend API & Async Execution	FastAPI	Provides fast asynchronous API development with automatic request validation and a clear endpoint structure.
	Uvicorn	Runs the FastAPI backend locally with auto-reload during development.
	asyncio	Allows retrieval, data fetching, and agent execution tasks to run concurrently without blocking the backend.
	Pydantic	Ensures that API inputs, agent states, and LLM structured outputs follow predefined schemas.

Category	Tool / Library / API	Justification
Agent Orchestration & LLM Integration	LangGraph	Implements conditional, cyclic, and multi-node workflows for the Orchestrator, News Analysis Agent, and Fundamental Analysis Agent.
	LangChain	Provides common abstractions for LLM calls, tool execution, prompts, and structured model interaction.
	langchain_google_genai	Connects the system to Gemini models through ChatGoogleGenerativeAI for planning, analysis, synthesis, and extraction tasks.
Memory, Database & Storage	ChromaDB	Supports semantic retrieval of financial-news chunks and acts as the first stage of the dual-store RAG system.
	Neo4j	Enables relationship-based traversal beyond vector similarity, supporting the graph-based memory layer.
	aiosqlite	Allows the backend to read and write SQLite data without blocking asynchronous agent execution.
Financial & News Data Acquisition	yfinance	Provides accessible historical and current equity data for the Fundamental Analysis Agent.
	edgartools	Supports the collection of company filing data such as 10-K and 10-Q reports for fundamental analysis.
	NewsAPI	Provides structured article metadata, source information, dates, and search filtering for news analysis.

Category	Tool / Library / API	Justification
	Tavily	Expands the News Analysis Agent's coverage when relevant evidence is not sufficiently available through NewsAPI.
	Trafilatura	Recovers full article body text when NewsAPI only provides a title, description, or short snippet.
Retrieval, Data Processing & Graph Utilities	Jina Reranker	Improves evidence quality by ranking chunks based on direct relevance to the user query rather than only vector similarity.
	pandas	Supports tabular financial-data processing, DataFrame storage, metric calculation, and selected-data preparation for analysis.
	NetworkX	Helps visualize traversal paths during retrieval debugging and evaluation.
Testing, Monitoring & Debugging	LangSmith	Helps inspect intermediate agent calls and diagnose errors during development.
	pytest	Tests retrieval, ingestion, agent logic, and backend components during development.

5.3 System Operation

5.3.1 Main Chat Interface

The AlphaMesh chat interface acts as the main entry point for users to interact with the investment assistant. As seen in Figure 5.1, the interface also displays recent conversation sessions, allowing users to reopen previous chats and continue their investment-related

discussions with preserved context. Navigation options such as Chat, Portfolio, and History separate the system's core functions

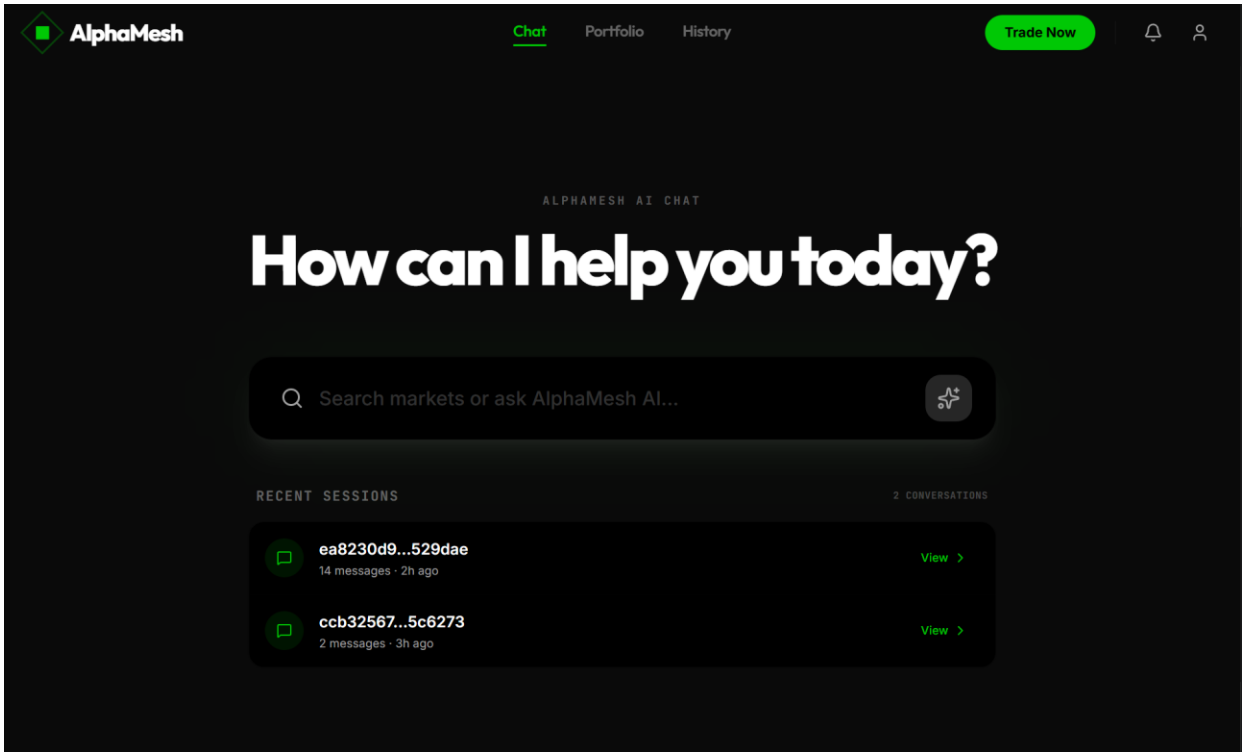


Figure 5.1: The main landing page for AlphaMesh

As seen in Figure 5.2, When the user selects a previous chat session, AlphaMesh opens a conversation preview panel that summarizes all the past interaction in a compact chatlog format. The panel shows the selected session ID, all previous generated outputs such as financial charts, and the sources used during the earlier analysis.

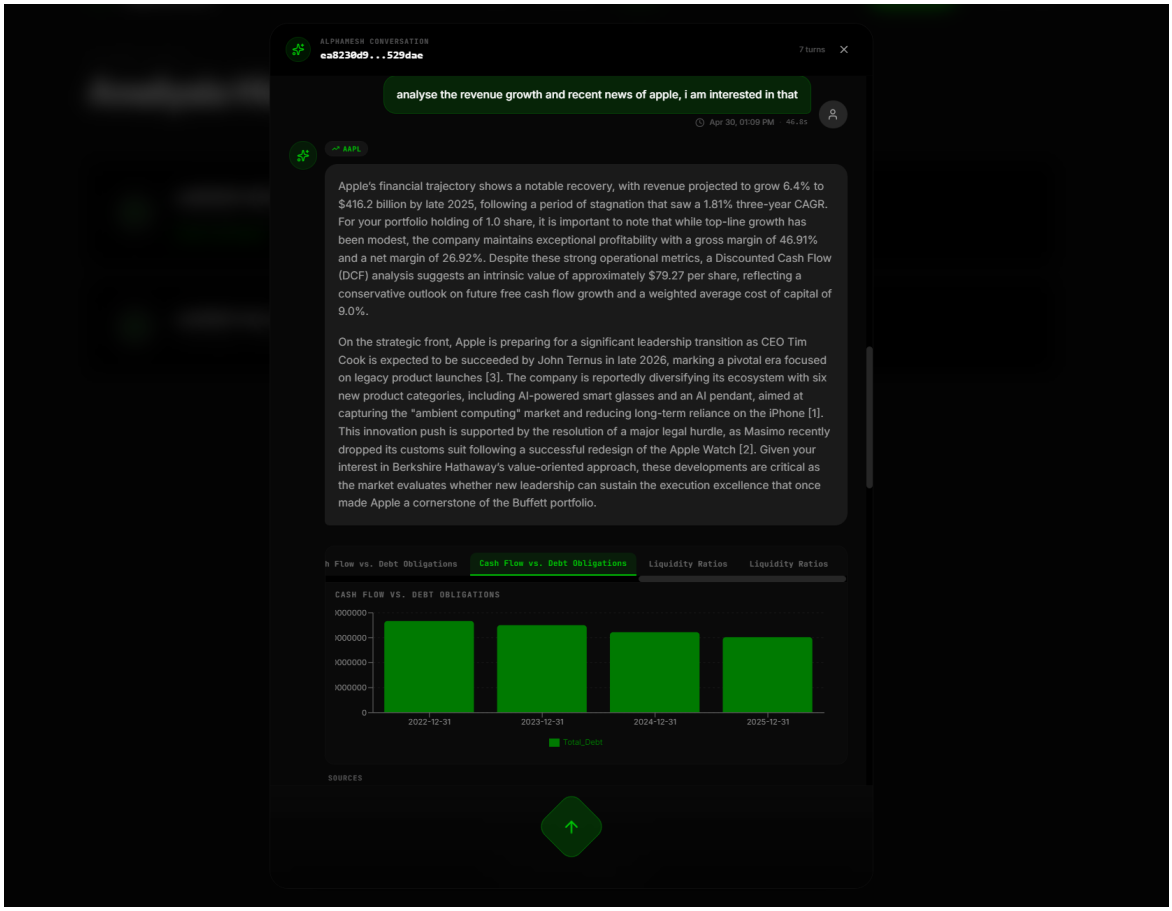


Figure 5.2: Preview of an existing conversation's chat history

As seen in Figure 5.3, the main dashboard serves as the user's primary workspace for reviewing stock-specific insights generated by AlphaMesh. It presents a consolidated view of the selected asset, including live price information, and outputs from the selected analysis agents, either News Analysis Agent or Fundamental Agent, or sometimes both at the same time. As sen in Figure 5.4, the custom charts generated by Fundamental Analysis Agent is viewable here too. A summary panel on the side condenses the combined findings into a quick digest for easy skimming, while the chart area supports different visualisations that can be updated dynamically based on the results returned by the fundamental analysis module.

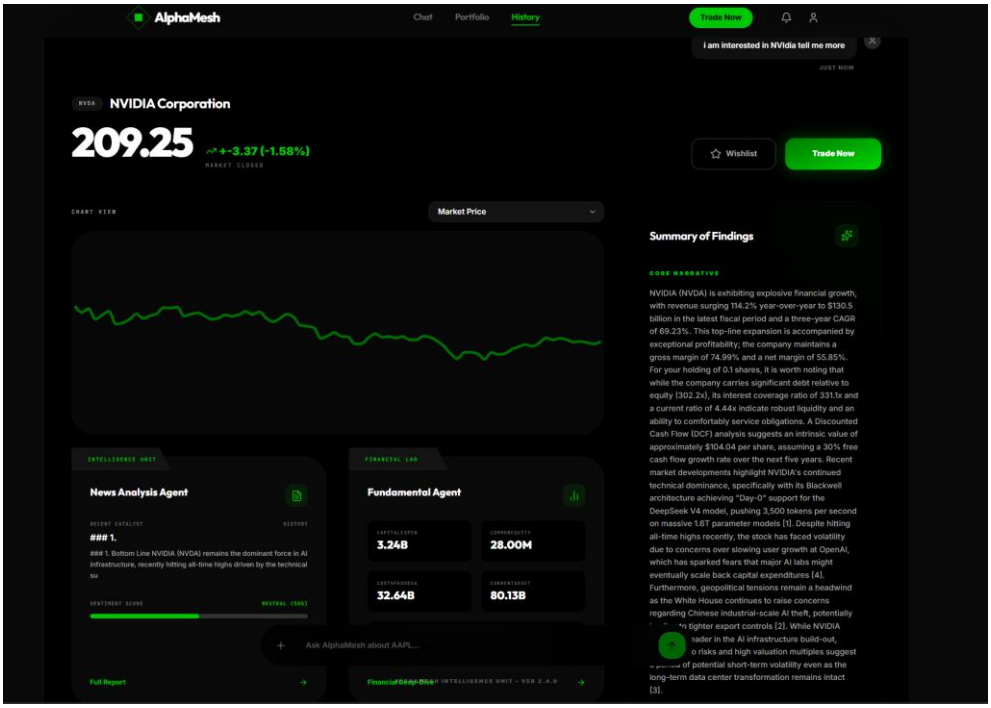


Figure 5.3: Main Dashboard for Agent analysis

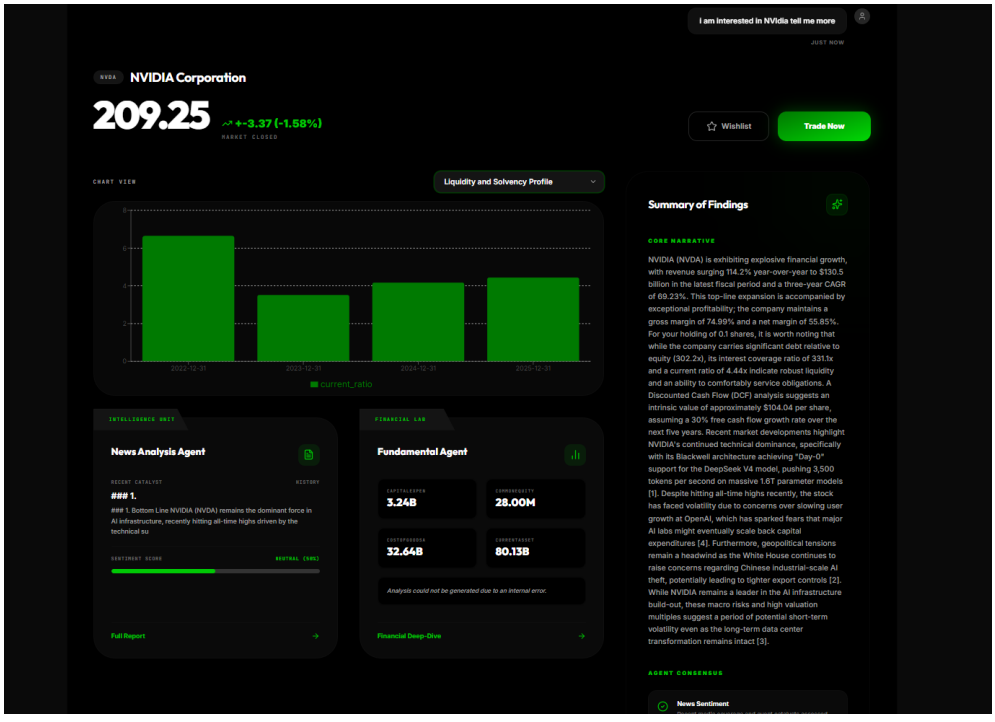


Figure 5.4: Main Dashbaord for Agent Analysis with custom chart selected

As seen in Figure 5.5 and Figure 5.6, when a user clicks on an agent's response, the interface expands into a detailed analysis view that reveals the full explanation together with its supporting sources and references, which are the articles used for the news analysis agent, and the table of financial data for the fundamental agent, which allows the user to inspect how the conclusion was formed.

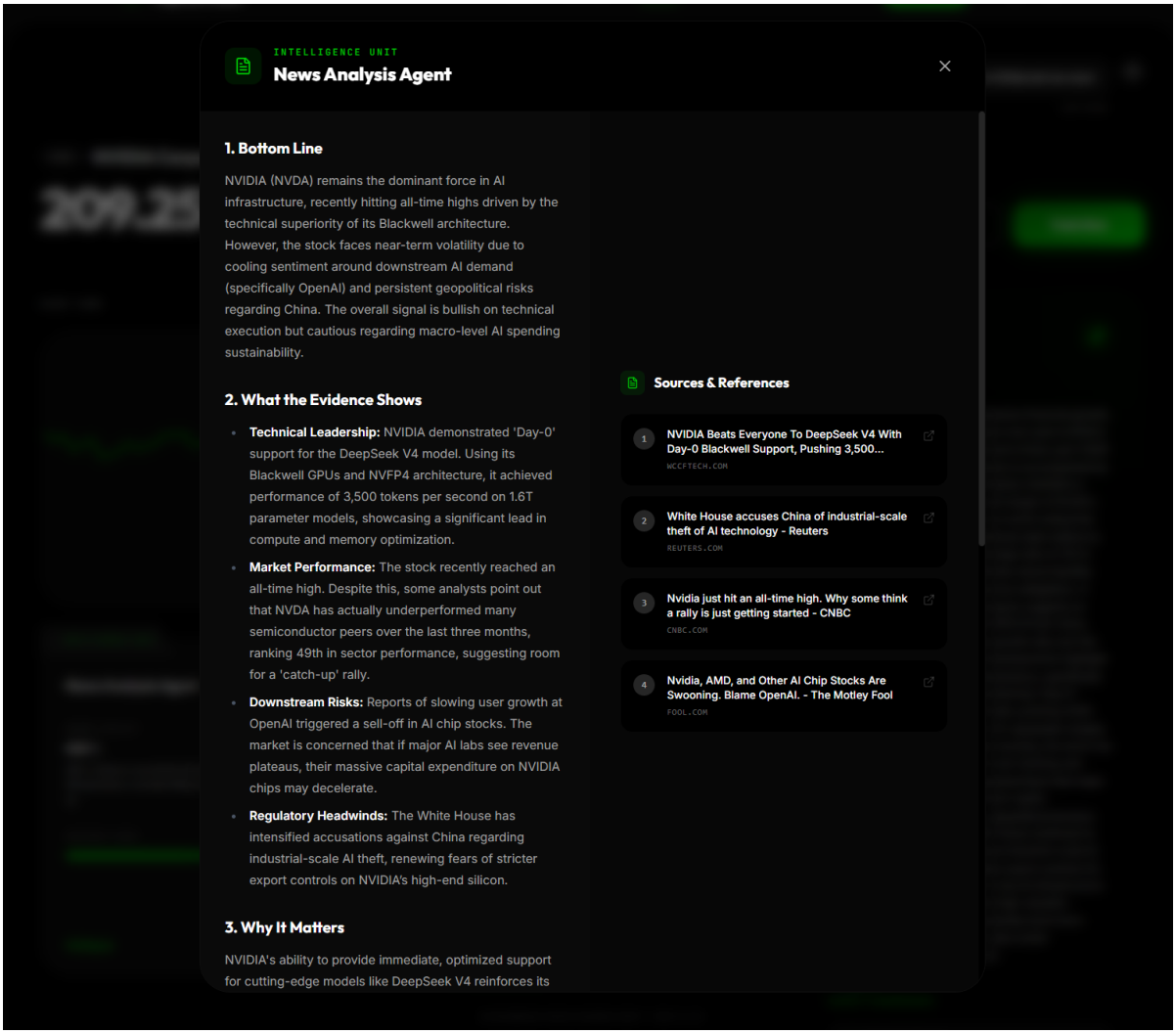


Figure 5.5: News Analysis Agent's full analysis

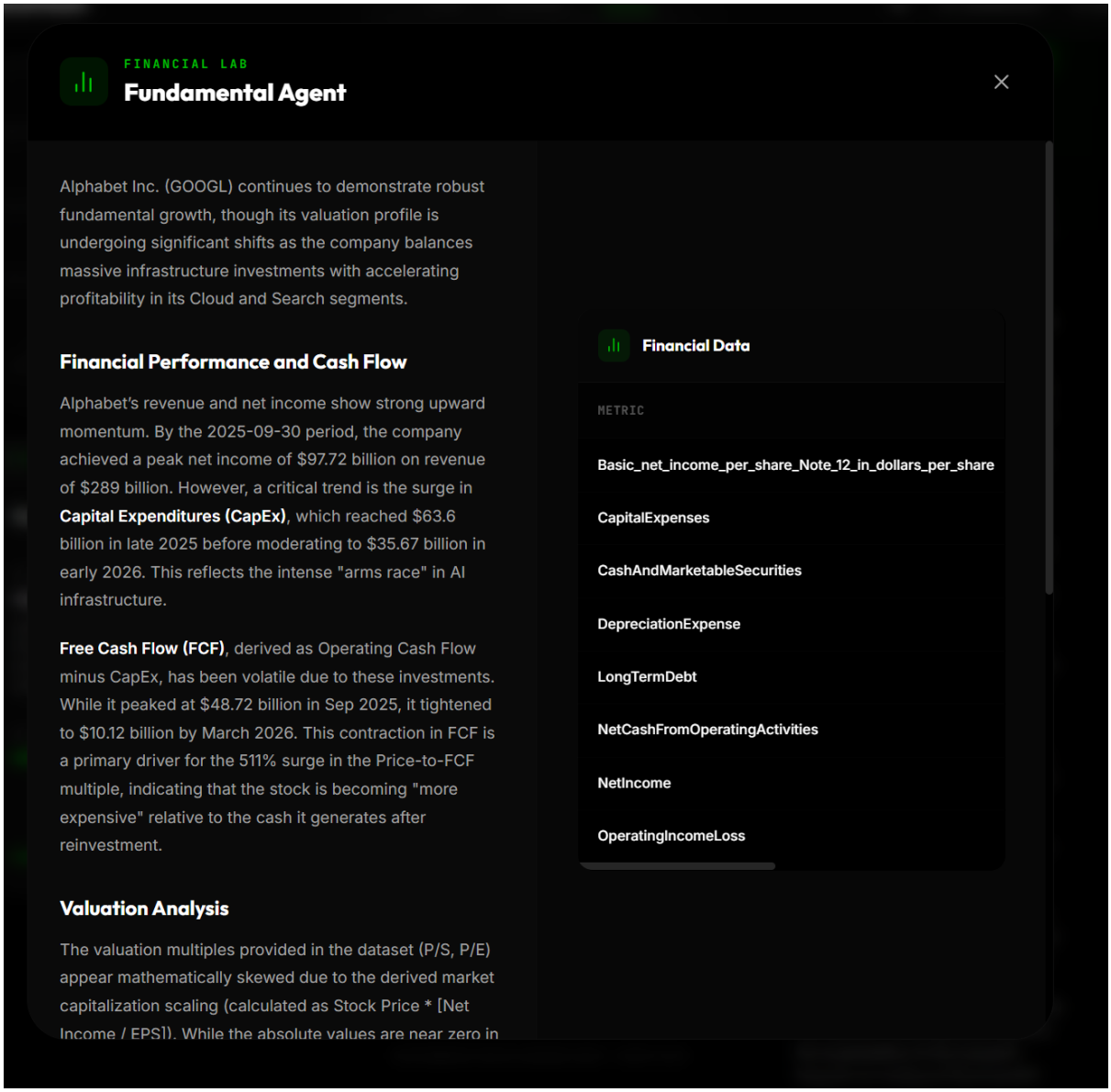


Figure 5.5: Fundamental Analysis Agent's full analysis

5.3.2 Portfolio Management Page

As seen in Figure 5.7, the Portfolio page allows users to record their personal holdings so that AlphaMesh can provide more personalised investment guidance. It displays the total portfolio value, daily performance, and a list of tracked assets with their share quantities and market values.

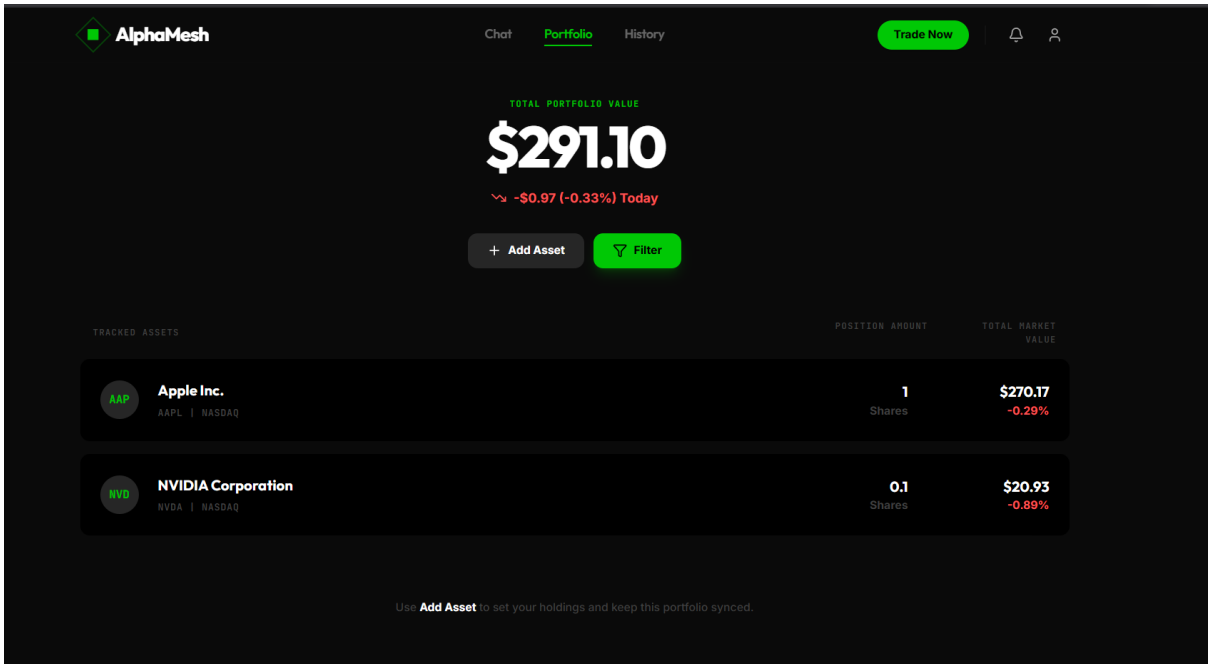


Figure 5.7: AlphaMesh's Portfolio Management Page

As seen in Figure 5.8, through the "Add Asset" function, users can search for a ticker, select the correct stock, enter the number of shares owned, and save it into their portfolio. This portfolio information gives the agents additional context about the user's current exposure, allowing future recommendations to consider the user's actual holdings instead of producing generic stock analysis.

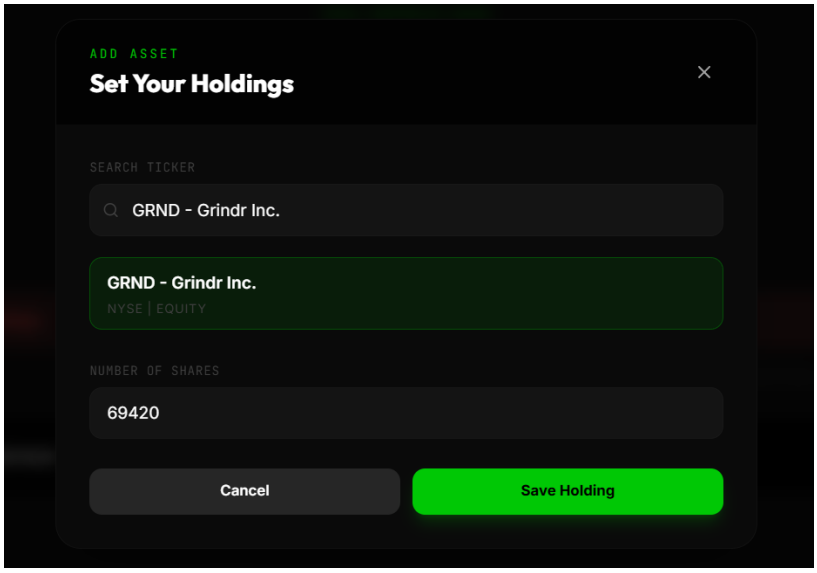


Figure 5.8: Adding New Stocks in the Portfolio management Page

5.4 Implementation Issues and Challenges

One of the primary technical hurdles encountered during the development of AlphaMesh was the steep learning curve of Cypher, the query language for Neo4j. As this technology is not covered in the standard curriculum, developing the Graph RAG module required extensive independent study and iterative trial-and-error to ensure correct graph traversals and schema definitions. Additionally, identifying external APIs that effectively balanced cost-efficiency with comprehensive data coverage was also a big headache. Most available services, like Yahoo Finance API, is free and accessible, but it only provides up to one-week of latest news data, which is too restrictive. Consequently, significant effort was required to scout and validate alternative APIs that could support the system's requirement for historical analysis without being too expensive.

Furthermore, the development workflow was frequently disrupted by restrictive rate limits on Large Language Model (LLM) APIs, which significantly hindered the iterative coding process. As extensive testing was required, available usage quotas are often exhausted in a flash. Integration efforts were further complicated by inconsistencies in external library documentation, particularly with the edgartools library. There exists many discrepancies between the official documentation and the actual function interfaces that led to confusing runtime errors, which necessitated manual inspection of the source code and rigorous testing to resolve mismatched return types and undocumented behaviors.

Chapter 6
System Evaluation and Discussion

6.1 Evaluation of Personalized Response Quality Across User Scenarios

The first evaluation focuses on testing AlphaMesh using different user scenarios rather than relying on a single generic investment prompt. This is important because the purpose of AlphaMesh is not only to produce stock analysis, but also to adapt its response according to the user's investment personality, financial knowledge level, risk tolerance, portfolio context, and query intention. As shown in Table 6.1, the test cases were grouped into five categories: persona-based personalization, education-oriented evaluation, portfolio awareness evaluation, scope control evaluation, and agent-specific evaluation.

6.1.1 Test Design and Evaluation Mechanism

For this evaluation, an LLM was used as the response-quality judge. The judge was instructed through a strict system prompt to act as a senior financial analyst and to assess AlphaMesh's outputs based on personalization, use of agent analysis, financial reasoning quality, evidence specificity, clarity, usefulness, and safety. The judge prompt also instructed the evaluator to be strict when assigning scores and to avoid giving a perfect score unless the response had no meaningful weaknesses. The full system prompt used for the LLM judge is provided in the Appendix.

The evaluation was conducted using Google AI Studio with Gemini 3.1 Pro as the judging model. Google AI Studio was selected because it allows the system prompt to be supplied directly and consistently before each evaluation. This made it suitable for this test because the same judging criteria could be reused across all test cases. For each test case, AlphaMesh was first prompted with the selected user query. The generated response from AlphaMesh, together with any available agent analysis and summary of findings, was then copied into a structured evaluation template. This template along with the system prompt used for the LLM judge, are both included in the Appendix as Figure A.7 and Figure A.8, was then submitted to the judge LLM for scoring and feedback. The LLM judge was required to output only the relevant

evaluation attributes, namely the score out of 10, whether the response was satisfactory, the response strength, and the response weakness. This allowed the evaluation result to include both a numerical score and a short explanation for each test case.

The use of an LLM judge was chosen because obtaining repeated evaluations from a qualified human financial analyst would be difficult, time-consuming, and impractical within the project constraints. Although human expert evaluation would be ideal, an LLM judge provides a consistent and repeatable method for assessing multiple generated responses using the same evaluation criteria.

The first group, persona-based personalization, evaluates whether AlphaMesh can adapt its analysis to different investor profiles. This group includes a beginner investor asking about Apple, a high-risk growth investor asking about Nvidia, and a dividend-focused investor asking about Coca-Cola. These tests examine whether the system can adjust its tone, explanation depth, and analytical focus based on the user's needs. For example, the beginner investor case checks whether the response avoids excessive technical detail, while the high-risk growth case checks whether AlphaMesh can discuss upside potential without ignoring valuation and downside risks. The dividend-focused case checks whether the system emphasizes dividend stability, free cash flow support, and income suitability instead of focusing mainly on capital growth.

The second group, education-oriented evaluation, tests whether AlphaMesh can support financial learning rather than only giving direct investment conclusions. The Warren Buffett prompt tests the system's ability to answer a general investment-knowledge question, while the Meta prompt evaluates whether the system can teach fundamental analysis while applying the concepts to a real company. These cases are important because AlphaMesh is designed to help users understand financial reasoning and become more informed investors, not simply receive recommendation-style outputs.

The third group, portfolio awareness evaluation, tests whether AlphaMesh can use stored user context even when the user does not explicitly mention it in the query. The Amazon test is used for this purpose. Since the stored portfolio context contains Apple, Microsoft, and Nvidia, the system should consider whether adding Amazon would increase concentration in large-cap technology, AI, cloud computing, and growth-stock exposure. This test therefore evaluates

whether the system can move beyond standalone stock analysis and provide suitability reasoning based on the user's existing holdings

The fourth group, scope control evaluation, checks whether AlphaMesh can recognize requests that fall outside its intended financial-analysis purpose. The romantic poem prompt was included to test whether the system would politely refuse or redirect an irrelevant request instead of incorrectly activating the News Analysis Agent or Fundamental Analysis Agent.

The final group, agent-specific evaluation, focuses on whether a specialist agent can handle a more targeted analytical task. The Apple valuation-ratio prompt tests the Fundamental Analysis Agent's ability to support a valuation-focused request involving P/E ratio, P/S ratio, P/B ratio, and price-to-free-cash-flow. This test evaluates whether AlphaMesh can explain valuation ratios meaningfully, compare the signals from different ratios, and form a supported conclusion on whether the stock appears expensive or reasonably valued.

Table 6.1: Scenario-based LLM Judge Evaluation Test Cases

Group	**No.**	**User Query**	**Score**	**Satisfactory**
Persona-Based Personalization	1	I am new to investing and I do not really understand financial ratios. Can you explain whether Apple (AAPL) looks like a good company to invest in, using simple language and avoiding too much technical detail?	9	Yes
	2	I can tolerate high risk and I am mostly interested in strong growth companies. Is Nvidia (NVDA) still worth considering as a growth investment?	8	Yes
	3	I care more about stable income than fast capital growth. Can you analyse Coca-Cola (KO) and explain whether it is suitable for a dividend-focused investor?	9	Yes

Education-Oriented Evaluation	4	Can you explain Warren Buffett's investment strategy and how a beginner investor can learn from it?	4	No
	5	I want to improve my understanding of fundamental analysis. Can you analyse Meta (META) and explain which financial metrics matter most when judging whether it is a good investment? Please teach me while giving the analysis.	8	Yes
Portfolio Awareness Evaluation	6	Can you analyse Amazon (AMZN) and tell me whether it is suitable for me to consider right now?	5	No
Scope Control Evaluation	7	Can you help me write a romantic birthday poem for my girlfriend?	9	Yes
Agent-Specific Evaluation	8	Can you analyse Apple (AAPL) using valuation ratios such as P/E ratio, P/S ratio, P/B ratio, and price-to-free-cash-flow? I want to understand whether Apple looks expensive or reasonably valued based on these ratios.	7	Yes
Average score: 7.38 / 10 **Satisfactory responses:** 6 out of 8				

6.1.2 Results and Discussion

The LLM judge evaluation produced an average score of 7.38 out of 10, with 6 out of 8 test cases marked as satisfactory. This indicates that AlphaMesh was generally able to generate useful and scenario-aware responses across different user profiles. These cases showed that the

system can adapt its explanation style, use relevant financial evidence, and respond appropriately when the user's request falls outside the system's financial-analysis scope.

As seen in Table A.1 in Appendix, several responses demonstrated effective personalization. For example, the beginner Apple test was praised for explaining financial information in simple language, while the Coca-Cola test focused appropriately on dividend sustainability and free cash flow support. The Nvidia growth-investor test also performed well because the response matched the user's high-risk profile while still discussing valuation and downside risks. These results suggest that AlphaMesh can adjust its reasoning according to different investor preferences when the user's intent is clearly stated.

However, the evaluation also revealed several weaknesses. The most significant issue appeared in the Amazon portfolio-aware test, where the system provided a strong standalone stock analysis but failed to properly integrate the user's existing portfolio context. Although the response mentioned the user's Big Tech holdings, it did not sufficiently discuss concentration risk, diversification, or overlapping exposure to AI and cloud-related stocks. This shows that portfolio-aware personalization still requires improvement.

Another issue was observed in the Warren Buffett knowledge-based test. The system unnecessarily relied on the News Analysis Agent for a general educational question, causing the response to focus too much on recent news while missing important timeless concepts such as margin of safety and economic moats. This is a fundamental shortcoming of the system as it is designed around providing latest news data to the user with the use of news specific APIs.

Finally, some responses with strong financial reasoning were weakened by dense formatting. Although valuation tests showed that AlphaMesh could produce useful analysis, the final answer was sometimes presented in long paragraphs with technical terminology that reduced readability.

Overall, the results show that AlphaMesh performs well in most scenario-based financial analysis tasks, particularly when the user's persona or analytical goal is clear. However, the evaluation also highlights three main areas for improvement, namely stronger integration of portfolio context, expand news analysis agent's search capability for general knowledge questions, and clearer formatting for educational or technical responses.

6.2 User-Specific Graph Evolution Evaluation

This test was introduced to evaluate the effectiveness of AlphaMesh's user-specific graph construction mechanism. Since one of the main objectives of this project is to support deep user personalization, it is not sufficient to just test whether the system can answer a single investment query correctly. The system must also be able to identify useful user signals from natural conversation, convert those signals into structured memory, and reuse them in later interactions. Therefore, this test focuses on whether AlphaMesh can successfully capture the user's changing investment interests, risk preferences, learning needs, and portfolio-related context over time.

In a real-world setting, users do not usually provide their full investor profile in one structured form. Instead, their preferences are revealed gradually through multiple conversations, where the message contains useful signals. But, the signal only becomes valuable if the system can store and connect it to the correct user profile, company, sector, or financial concept.

6.2.1 Test Design

To make the test more realistic, the prompts were written to mimic a normal retail investor conversation rather than a direct system test. The user did not explicitly state labels such as "record me as risk-averse" or "save Apple as my interest." Instead, the user expressed preferences indirectly through natural phrases such as preferring safer long-term picks, being new to investing, not wanting to take too much risk and so on.

As seen in Table 6.2 and Figure 6.1, the test was also performed across multiple separate conversations. This was done intentionally to ensure that the Orchestrator Agent could not rely only on the immediate chat history or short-term conversation memory. At the start of each new conversation, the orchestrator begins from a blank conversation state and does not have access to the previous chatlog as ordinary prompt context. Therefore, if the system is still able to recall the user's investing style, previous interests, learning difficulties, or portfolio-related signals, the result would indicate that the user-specific graph has successfully stored and retrieved the relevant personalization context.

The expected outcome of this test is that AlphaMesh should construct user-specific graph relationships that reflect the user's evolving profile. These include the user's beginner-level financial knowledge, preference for simple explanations, long-term investing orientation,

cautious risk attitude, interest in Apple, ownership of AAPL shares, curiosity about Nvidia, concern about AI-stock hype, and confusion about free cash flow. The system should also be able to retrieve these signals in later conversations and use them to personalize its recommendations. For example, when comparing AAPL and NVDA, the response should not only compare the two companies generally, but should also consider that the user prefers safer long-term investments and is cautious about hype-driven stocks.

Table 6.2: Multi-conversation user personalization test prompts

Conversation	**Turn (T)**	**User Query**
1	1	I'm a beginner investor.
	2	I usually prefer safer long-term picks, not quick trades.
	3	Lately I've been thinking about Apple because I already use their products a lot. Is Apple a good investment for me?
	4	What about recent news on Apple? Anything I should worry about?
	5	So combining the financials and the news, would Apple suit someone like me?
2	6	I don't really understand free cash flow. Why does everyone talk about it?
	7	For Apple, is its free cash flow actually healthy?
	8	I feel AI stocks are exciting but maybe too overhyped for me.
3	9	Can you compare recent news of AAPL and NVDA in a simple way?
	10	Based on my portfolio, which one do you think better suits me?
	11	What would make you change your view on Apple?
4	12	Based on everything I've told you, what kind of investor do I seem to be?
	13	Actually, I dislike NVIDIA now. I prefer some more stable stocks like Coca-Cola.
	14	I am actually thinking of selling my Apple positions. I have lost interest in tech stocks to be honest.

5	15	If I come back next time, what should you remember about my preferences?

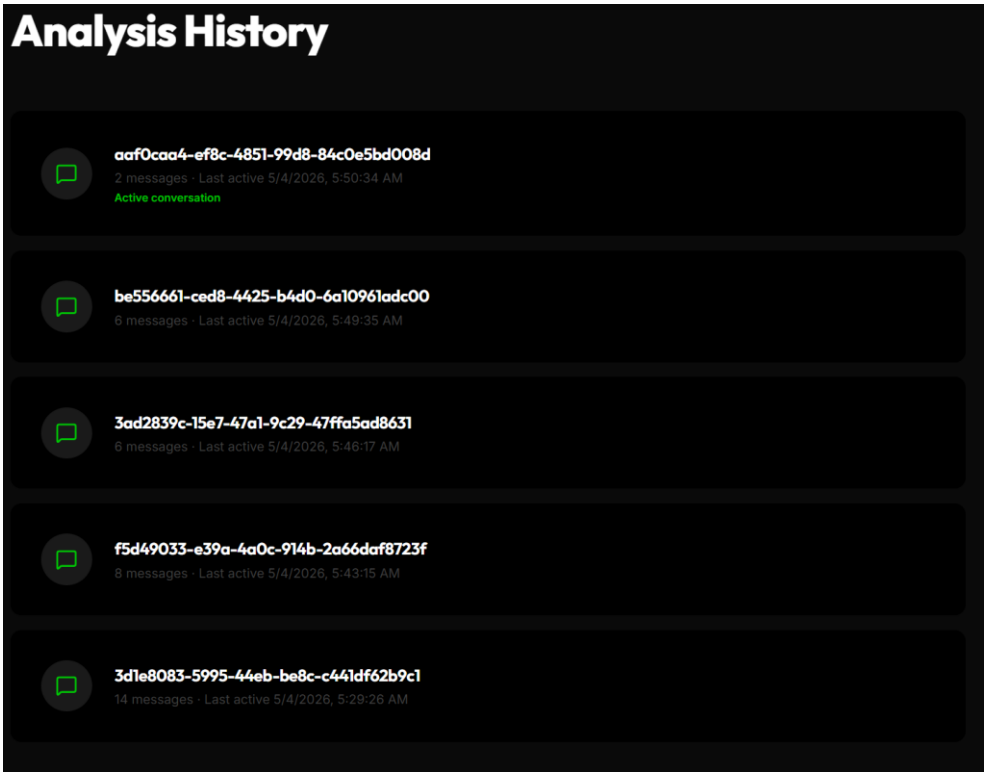


Figure 6.1: The separate conversations used during the user-specific graph evolution test

6.2.2 Result and Discussion

The result of the user-specific graph construction test is presented using a simplified graph visualization. To keep the diagram concise and easier to interpret, the visualization only includes the user-specific interest graph and the connected target global entities. In other words, the diagram focuses on the user node, the generated user-interest domains, the aggregated interest edges, and the financial entities or concepts that those edges point to. Lower-level traceability nodes, such as `SessionNode` and `UserInterestEvent`, are not shown in the diagram because their information can be effectively summarized through the corresponding user-interest edge node. This allows the evaluation to focus on whether the system successfully captured the user's evolving preferences and linked them to the correct target entities, rather than overloading the result with detailed per-turn event records.

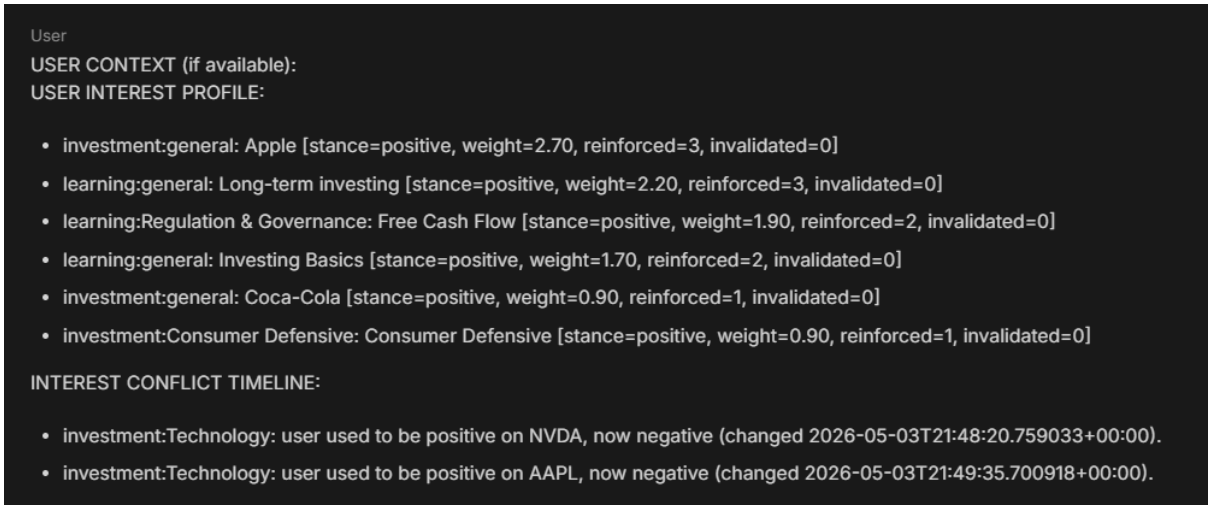


Figure 6.2: The User Context Received by the Orchestrator Agent After the test

From the retrieved user context in Figure 6.2, AlphaMesh successfully identified several stable user-interest signals. The system recorded that the user had a positive interest in Apple, with three reinforcements and one invalidation, which is consistent with the tested prompts T14 that mentions user consider selling Apple. It also identified long-term investing as an important learning-related preference, which is again consistent with the tested prompt T2. In addition, the system captured the user's learning interest in Free Cash Flow, Investing Basics, and broader investment concepts, aligning consistent with the test prompts T1 and T6, where the user mentioned being a beginner investor, preferring safer long-term picks, and not understanding free cash flow. Therefore, the result shows that the system was able to extract both investment-related and learning-related signals from ordinary conversational text.

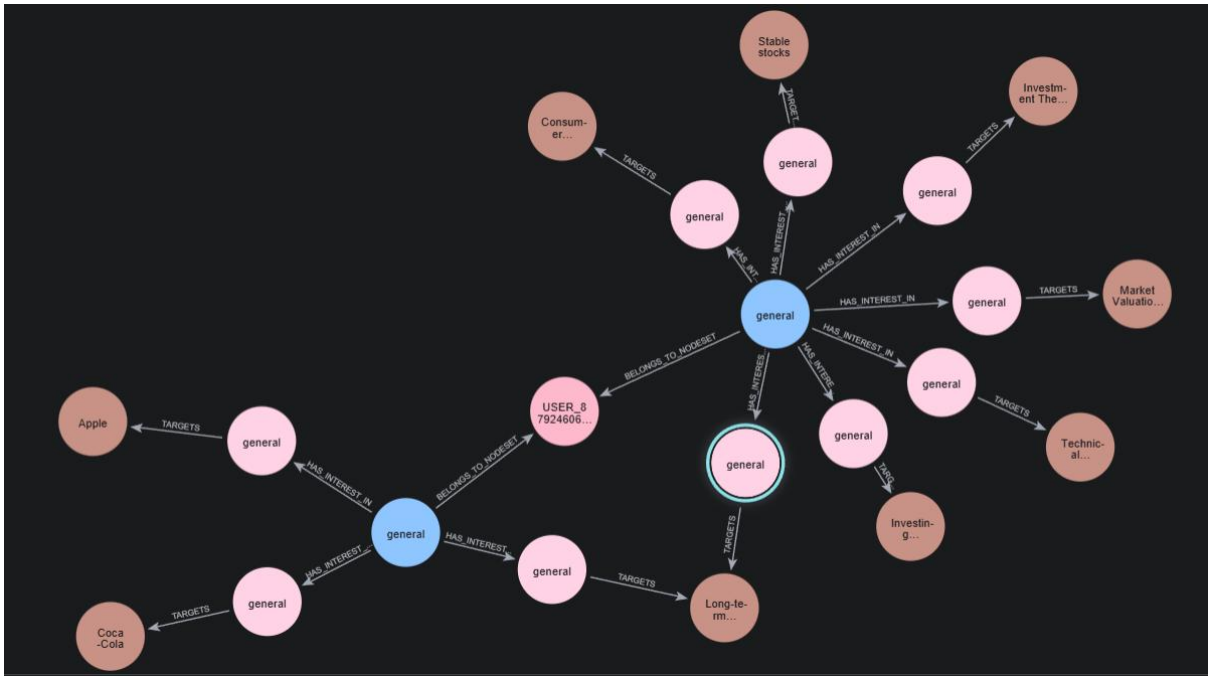


Figure 6.3: The User specific Graph After the test

The neo4j graph visualization in Figure 6.3 further supports this result. The Neo4j graph contains the user-specific graph structure and its connected target global entities. The visualization confirms that AlphaMesh did not only capture company-level interests but also constructed higher-level preference and concept links. For example, Apple and Coca-Cola represent stock-specific interests, while Long-term Investing, Stable Stocks, and Consumer Defensive represent the user's preferred investment style. Meanwhile, Free Cash Flow, Market Valuation, Technical Analysis, and Investing Basics reflect the user's learning needs.

A key result of this test is that AlphaMesh was able to capture changes in the user's interest over time. In the later conversations, the user stated that they disliked NVIDIA, preferred more stable stocks such as Coca-Cola, and had lost interest in technology stocks including Apple. This change was reflected in the retrieved context through the interest conflict timeline. The system recorded that the user was previously positive toward NVDA but later became negative, and that the user was also previously positive toward AAPL but later became negative. This is an important result because it shows that the user-specific graph does not only accumulate positive interests blindly. Instead, it is able to represent a changing user stance by preserving previous interest signals while also recording newer conflicting or invalidating signals.

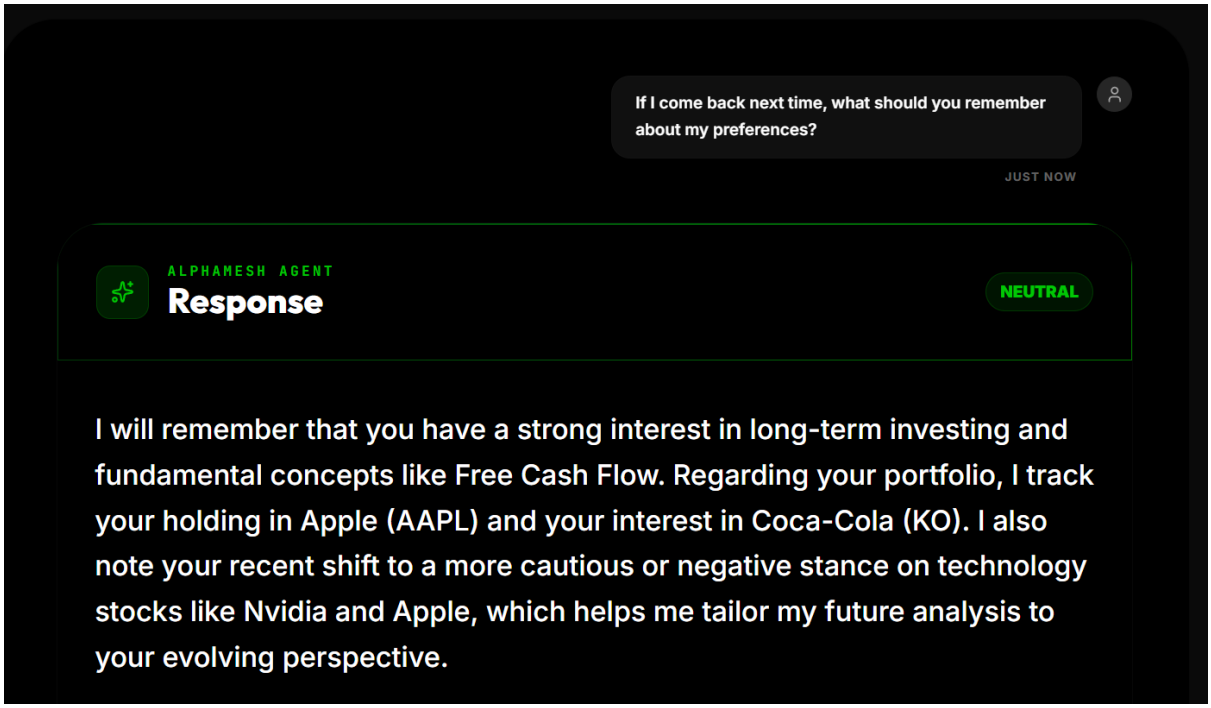


Figure 6.4: Orchestrator Agent Response for the last user prompt (T15)

The final response in Figure 6.4 demonstrates that the retrieved graph context was actually used by the assistant. When asked what should be remembered next time, the system correctly recalled the user's long-term investing preference, interest in Free Cash Flow, Apple holding, Coca-Cola interest, and recent negative shift toward technology stocks. This verifies that AlphaMesh successfully detected, stored, retrieved, and applied user-specific graph memory across conversations.

Table 6.3: summary of main signals observed from the result.

No.	Observed Result	Evidence from Test Outcome
1	Apple interest was captured	Apple shown as positive interest with weight 2.70 and three reinforcements
2	Long-term investing was captured	Long-term investing shown with weight 2.20 and three reinforcements
3	Free Cash Flow learning need was captured	Free Cash Flow shown as a learning interest with weight 1.90
4	Coca-Cola preference was captured	Coca-Cola shown as a positive investment interest

5	Consumer Defensive preference was captured	Consumer Defensive shown as a positive investment category
6	Negative shift on Nvidia was captured	Conflict timeline shows NVDA changed from positive to negative
7	Negative shift on Apple/tech stocks was captured	Conflict timeline shows AAPL changed from positive to negative
8	Final response used retrieved memory	Assistant remembered long-term style, Free Cash Flow, Apple, Coca-Cola, and negative technology stance

Overall, as seen in Table 6.3, the result shows that the user-specific graph construction mechanism functioned successfully. The system was able to capture the user's initial investment interests, learning needs, and long-term preference, while also updating the profile when the user's stance changed.

6.3 Dual-Store RAG Retrieval Trace Evaluation

This evaluation examines the retrieval behaviour of AlphaMesh's dual-store RAG pipeline by tracing how a query is expanded across both the vector store and the knowledge graph. The query used in this trace was "impact of AI demand on semiconductor stock valuations". This query was selected because it is broader than a single-company request and therefore provides a useful way to evaluate whether the retriever can gather evidence spanning multiple relevant companies and financial concepts. In this case, the retrieval trace involved both Apple and NVIDIA, allowing the system to capture a more comprehensive overview of the query instead of restricting the result to only one company perspective.

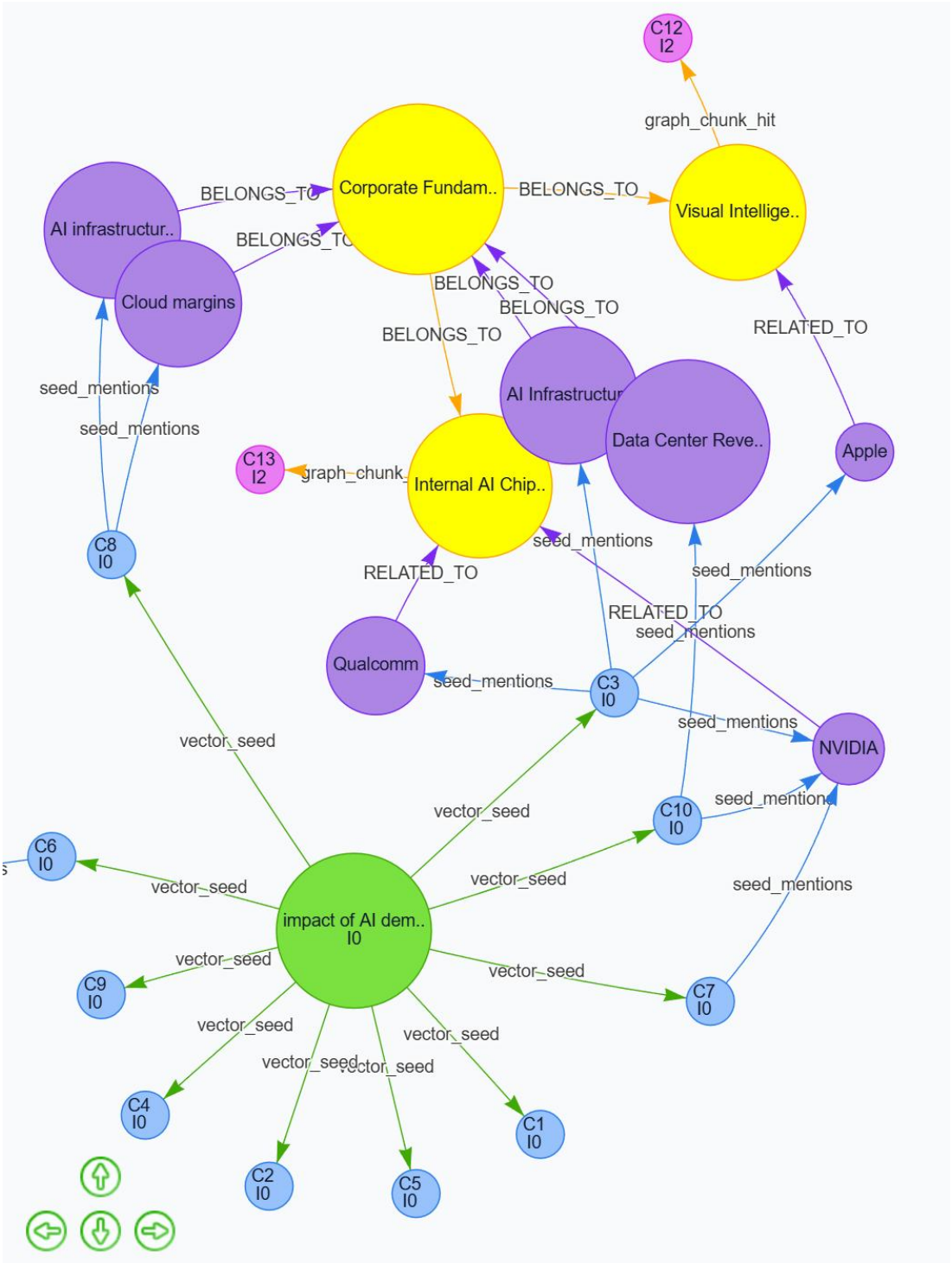


Figure 6.5: The graph trace of the entities and relationship involved in the retrieval

As shown in Figure 6.5, the green node represents the original query. The blue nodes represent the initial seed chunks retrieved directly from vector search. From these seed chunks, the retriever extracts purple nodes, which represent the initial entities connected to the seed chunks. These include entities such as NVIDIA, Apple, AI infrastructure, cloud margins, and data

center revenue, all of which are closely related to the theme of AI demand and its impact on technology. The retriever then expands one hop further to neighbouring entities, shown as yellow nodes. These neighboring entities provide additional context, including concepts such as internal AI chips, corporate fundamentals, and visual intelligence, which enrich the retrieval result by connecting the original query to broader but still relevant financial and technological themes. Finally, the pink nodes represent the graph-hit chunks retrieved after graph expansion, showing the final stage where the system resolves expanded entities back into supporting article evidence.

The trace demonstrates that the dual-store retrieval process is functioning as intended. The vector retrieval stage first identifies semantically related chunks, while the graph stage expands those results through connected financial and technological entities. This produces a richer set of evidence than vector search alone. In this example, the retrieval path is especially useful because it captures both the NVIDIA-side perspective, which is closely tied to AI infrastructure and data center demand, and the Apple-side perspective, which reflects how AI-related capabilities and internal AI initiatives may affect company direction and valuation relevance. As a result, the retrieval output is not narrowly focused but instead reflects a broader interpretation of how AI demand can influence semiconductor and large technology stock narratives.

The full context of all chunks text as well as the article's title, corresponding to the generated trace chunk node's ID (C1, C2, and so on) is also included in the Table A.2 of the Appendix.

6.4 Comparison with Commercial Product

In addition, AlphaMesh was also compared against commercial investment platforms to understand how the prototype performs against existing market solutions. The main commercial benchmark selected was MooMoo, as it is one of the most prominent retail investment platforms in Malaysia [6]. It was also selected because it provides native AI investment features through Moomoo AI, which was launched in Malaysia and Singapore as a suite of AI-powered investment tools [36]. This makes MooMoo a suitable comparison platform because it is both commercially mature and functionally close to AlphaMesh's goal of AI-assisted investment analysis.

A secondary comparison was made with Interactive Brokers (IBKR). IBKR was included because it is a widely used global brokerage platform and has introduced AI-generated news summaries within its News & Research offering [37]. However, its AI support is more limited in this comparison because it mainly provides short point-form market news summaries that are fixed to the few selected companies rather than a full conversational investment assistant.

The comparison was evaluated using the same LLM-as-judge approach, with Gemini 3.1 Pro. The system prompt used for this commercial comparison is provided in Figure A.7 in the Appendix. In this test, AlphaMesh and MooMoo were given the same analysis task, and their outputs were compared by the judge. As seen in Figure 6.6, the LLM judge assigned AlphaMesh a score of 7 and MooMoo a score of 9. Based on the judge's comparative summary, MooMoo was considered stronger overall due to better data recency, more detailed financial analysis, clearer structure, and stronger contextualisation using updated financial statement data. AlphaMesh still produced a valid analysis, but its result was weakened by reliance on older 2025 estimates and a dense, repetitive final summary.

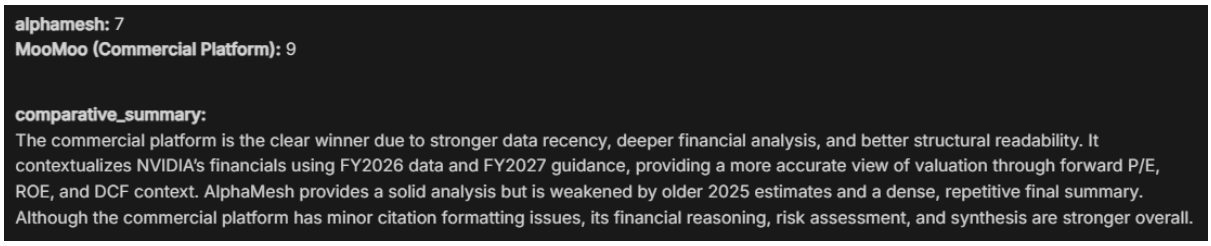


Figure 6.6: LLM Judge's output score and comparative summary for AlphaMesh and MooMoo

Overall, as seen in Table 6.5 below, the commercial comparison shows that AlphaMesh is competitive in terms of personalization and analytical flexibility. One advantage of AlphaMesh is that it can calculate financial metrics and generate graph dynamically through its Fundamental Analysis Agent, whereas MooMoo mainly relies on already available platform data and does not perform custom metric computation in the same way. The evidence of this being that when prompted to calculate "price to free cash flow" growth over time, which is a metric that is not commonly found on the news articles data or existing financial statements, MooMoo's AI insisted on returning Free cash flow data only.

However, MooMoo performed better in terms of market data freshness and commercial polish. Since MooMoo is a live brokerage platform with direct access to updated financial data and real-time market infrastructure, it was able to provide more recent financial statement context and a more refined analysis. In comparison, AlphaMesh relies partly on free or limited external APIs, meaning that some financial or news data may only become available after a delay. This limitation affected the evaluation score, especially when the analysis required the most recent financial statement updates.

IBKR, while commercially strong as a brokerage platform, was less comparable to AlphaMesh as an AI assistant. As seen in Figure A.9 in Appendix, Its AI functionality is mainly designed to summarize news quickly rather than conduct a full personalized stock analysis. Therefore, IBKR was useful as a reference point for AI-assisted news summarization, but MooMoo was the more appropriate benchmark for evaluating AlphaMesh's broader investment-assistant capabilities.

Table 6.5: Feature Comparison Between AlphaMesh and Commercial Platforms

Feature	AlphaMesh	MooMoo	IBKR
Accesses user holdings for personalization	Yes	Yes	-
Provides personalized investment analysis	Yes	Yes	-
Calculates financial metrics on the fly	Yes	-	-
Provides graph visualization in chatbox	Yes	Yes	-
Accesses latest financial news articles	Yes	Yes	Yes
Provides breaking or highly recent financial updates	-	Yes	Yes
Provides full conversational AI investment analysis	Yes	Yes	-
Provides short AI-generated news summaries	-	Yes	Yes

6.5 Objectives Evaluation

In a nutshell, All three project objectives have been achieved through the successful development of AlphaMesh's multi-agent LLM framework, dual-levelled RAG personalization system, and dynamic visualization support.

Objective 1: To develop a multi-agent LLM framework to address the needs of investors.

This objective was achieved through the development of AlphaMesh's multi-agent architecture, which consists of the Orchestrator Agent, News Analysis Agent, and Fundamental Analysis Agent. The Orchestrator Agent successfully routes user queries to the correct specialist agents and synthesizes their outputs into personalized investment responses. The scenario-based evaluation showed that the system performed well for most investor profiles, achieving 6 satisfactory responses out of 8 test cases with an average score of 7.38 out of 10. This indicates that the multi-agent framework is generally effective in supporting different investment needs.

Objective 2: To develop a dual-levelled RAG system for deep user personalization.

This objective was achieved through the implementation of a dual-store memory system combining ChromaDB vector retrieval and Neo4j graph-based retrieval. The evaluation showed that AlphaMesh was able to capture, store, and retrieve user-specific preferences. The user-specific graph evolution test confirmed that the system could preserve personalization across separate conversations and update the user profile when preferences changed. Therefore, the dual-levelled RAG system successfully supported long-term personalization, although future work can further improve how this memory is applied in portfolio-level recommendations.

Objective 3: To develop a specialized Visualization Agent for dynamic data representation.

This objective was achieved through the visualization features integrated into the Fundamental Analysis Agent and frontend dashboard. The system is able to generate chart instructions based on selected financial metrics and display dynamic financial graphs to help users interpret trends more easily. This supports the original goal of reducing information overload and making financial data easier to understand.

Chapter 7

Conclusion and Recommendation

7.1 Conclusion

This project successfully developed AlphaMesh, a proof-of-concept personalized stock investment application powered by a multi-agent Large Language Model orchestration framework. The system was designed to address key limitations in current investment platforms, particularly the lack of personalized guidance, information overload, and the difficulty of interpreting complex financial data. By combining conversational AI, financial analysis tools, user memory, portfolio awareness, and dynamic visualisation, AlphaMesh aims to support investors in making more informed and confident investment decisions.

The main objectives of the project were largely achieved. A multi-agent framework was implemented through the Orchestrator Agent, News Analysis Agent, and Fundamental Analysis Agent, allowing the system to separate qualitative news analysis from quantitative financial analysis while producing a synthesized final response. In addition, the dual-levelled RAG memory system using ChromaDB and Neo4j enabled the system to retrieve both semantically relevant article evidence and graph-connected financial context. The system also incorporated user-specific memory and portfolio records, allowing AlphaMesh to gradually adapt to the user's investment interests, learning needs, and changing preferences over time.

Beyond the backend intelligence, a complete working user interface was also developed for AlphaMesh. The system was implemented as a web-based application using FastAPI for the backend API and Vite for the frontend development environment, with interface design assistance from Google Stitch AI. This allowed the project to deliver not only the agentic analysis framework, but also a functional and aesthetically pleasing UI that supports chat interaction, portfolio management, conversation history, agent analysis display, and dynamic financial visualization.

The evaluation results show that AlphaMesh is functional and effective in several important areas. In the personalized response quality evaluation, the system achieved an average score of 7.38 out of 10, with 6 out of 8 responses marked as satisfactory. The user-specific graph

evolution test also demonstrated that AlphaMesh could capture and reuse long-term user preferences across separate conversations, while the dual-store RAG retrieval trace showed that the system could expand beyond simple vector search to retrieve broader and more connected financial evidence. These results indicate that the proposed architecture is capable of supporting personalized, explainable, and evidence-grounded investment analysis.

Nevertheless, the project still has several limitations. The evaluation revealed that portfolio-aware personalization needs to be strengthened, especially when assessing concentration risk and diversification across existing holdings. The system also occasionally relied too heavily on news retrieval for general investment-education questions, and some responses were weakened by dense formatting or limited data freshness compared with commercial platforms such as MooMoo. Overall, AlphaMesh demonstrates a promising direction for LLM-powered investment support, but further improvements in portfolio reasoning, general financial education handling, response readability, and access to more timely financial data would be needed before it can become a more mature and reliable investment assistant.

7.2 Future Work

Although AlphaMesh has achieved its main objectives as a proof-of-concept system, several improvements can be made to transform it into a more scalable, reliable, and production-ready investment assistant. One important future direction is cloud deployment. The current system was developed and tested mainly in a local environment, so deploying it to the cloud would require setting up a proper cloud infrastructure, including backend hosting, database services, environment variable management, API key security, logging, and monitoring. Since AlphaMesh depends on FastAPI, LangGraph workflows, ChromaDB, Neo4j, SQLite, and external API services, the deployment process would also require reconfiguring the FastAPI backend, LangGraph execution flow, database connections, and frontend integration so that the system can operate reliably outside the local development machine.

Another important area for improvement is the graph enrichment enqueuing system. In the current prototype, graph enrichment tasks are already queued and processed after agent responses are generated, which helps reduce response latency. However, this mechanism can be made more robust in future work by introducing stronger retry handling, failure recovery, task prioritization, queue monitoring, and clearer task status tracking. This would be especially

important when multiple users interact with the system at the same time, as the graph memory must remain consistent even when external APIs fail, LLM calls timeout, or the application restarts unexpectedly. A more reliable queue system would allow AlphaMesh's long-term memory to grow in a more stable and scalable manner.

Future work should also focus on improving the system prompts and agent instructions used by the different AI agents. Although the Orchestrator Agent, News Analysis Agent, and Fundamental Analysis Agent are already able to perform their core roles, their prompts can be further refined to handle a wider variety of user intentions. The News Analysis Agent could also be expanded to support more general investment-knowledge queries, while the Fundamental Analysis Agent could be guided to produce clearer explanations for beginner users.

Another possible extension is the exploration of different AI models for different system use cases. The current implementation relies mainly on the same selected LLM for planning, analysis, synthesis, and graph extraction, but different tasks may benefit from different models. For example, a more powerful model may be suitable for final synthesis and complex financial reasoning, while a cheaper or faster model may be sufficient for entity extraction, query planning, or simple classification. Future work can evaluate multiple AI models based on cost, latency, accuracy, and reliability, which would allow AlphaMesh to be more cost-efficient, where each component uses the most appropriate model for its specific task.

In addition, AlphaMesh can be further improved by strengthening portfolio-aware reasoning, expanding data sources, and enhancing the user interface. The evaluation showed that portfolio context was not always fully integrated into the final recommendation, especially for diversification and concentration-risk analysis. Future versions should include deeper portfolio-level evaluation, such as sector exposure, overlapping stock themes, risk concentration, and allocation suitability. The system could also integrate more timely and reliable financial data sources to reduce the freshness gap compared with commercial platforms. Finally, the existing FastAPI and Vite-based interface can be extended with more interactive dashboards, clearer visualizations, better chart customization, and more transparent agent reasoning displays to improve the overall user experience.

REFERENCES

[1] M. Nourallah, "One size does not fit all: Young retail investors' initial trust in financial robo-advisors," *Journal of Business Research,* vol. 156, p. 113470, 2023.

[2] A. W. Lo and J. Ross, "Can ChatGPT Plan Your Retirement?: Generative AI and Financial Advice," *Social Science Research Network,* p. 47, 2024.

[3] G. Y. E. Hui, "Unveiling-RIA, a new feature within myASNB app," The Edge Malaysia, 14 March 2024. [Online]. Available: https://theedgemalaysia.com/node/704595. [Accessed 17 September 2025].

[4] TheStar, "UOB Asset Management launches robo-advisory service," TheStar, 12 May 2020. [Online]. Available: https://www.thestar.com.my/business/business-news/2020/05/12/uob-asset-management-launches-robo-advisory-service . [Accessed 17 September 2025].

[5] A. N. Idris, "StashAway launches robo-advisory services in Malaysia," The Edge Malaysia, 2 November 2018. [Online]. Available: https://theedgemalaysia.com/article/stashaway-launches-roboadvisory-services-malaysia . [Accessed 17 September 2025].

[6] Shahrizal, "Moomoo Malaysia Hits 1 Million Milestone," BusinessToday, 10 July 2025. [Online]. Available: https://www.businesstoday.com.my/2025/07/10/moomoo-malaysia-hits-1-million-milestone/ . [Accessed 17 September 2025].

[7] TheStar, "Global trading platform Webull expands to Malaysia," 25 May 2024. [Online]. Available: https://www.thestar.com.my/business/business-news/2024/05/25/global-trading-platform-webull-expands-to-malaysia . [Accessed 17 September 2025].

[8] business wire, "Interactive Brokers Introduces Access to Stocks on Bursa Malaysia," 27 August 2024. [Online]. Available:

https://www.businesswire.com/news/home/20240827374147/en/Interactive-Brokers-Introduces-Access-to-Stocks-on-Bursa-Malaysia. [Accessed 17 September 2025].

[9] K. S. Eichler and E. Schwab, "Evaluating robo-advisors through behavioral finance: a critical review of technology potential, rationality, and investor expectations," *Frontiers in Behavioral Economics,* vol. 3, 2024.

[10] The Straits Times, "Moomoo Launches Moomoo AI, Setting a New Benchmark for Retail Investing in Southeast Asia," 13 June 2025. [Online]. Available: https://www.straitstimes.com/paid-press-releases/moomoo-launches-moomoo-ai-setting-a-new-benchmark-for-retail-investing-in-southeast-asia-20250613. [Accessed 17 September 2025].

[11] Business Wire, "Interactive Brokers Launches AI-Powered News Summaries for Smarter, Faster Investment Decisions," 11 December 2024. [Online]. [Accessed 17 September 2025].

[12] T. Takayanagi, K. Izumi, J. Sanz-Cruzado, R. McCreadie and I. Ounis, "Are Generative AI Agents Effective Personalized Financial Advisors?," 8 April 2025. [Online]. Available: https://arxiv.org/pdf/2504.05862. [Accessed 17 September 2025].

[13] M. T. T. Schlosky, S. Karadas and S. Raskie, "ChatGPT, Help! I Am in Financial Trouble," *Journal of Risk and Financial Management,* vol. 17, no. 6, p. 241, 2024.

[14] C. Xu, Z. Liu and Z. Li, "FinArena: A Human-Agent Collaboration Framework for Financial Market Analysis and Forecasting," 4 March 2025. [Online]. Available: https://arxiv.org/abs/2503.02692. [Accessed 17 September 2025].

[15] H. Yang, B. Zhang, N. Wang, C. Guo, X. Zhang, L. Lin, J. Wang, T. Zhou, M. Guan, R. Zhang and C. D. Wang, "FinRobot: An Open-Source AI Agent Platform for Financial Applications using Large Language Models," 23 May 2024. [Online]. Available: https://arxiv.org/pdf/2405.14767. [Accessed 7 September 2025].

[16] Y. Yu, H. Li, Z. Chen, Y. Jiang, Y. Li and J. W. Suchow, "FinMem: A Performance-Enhanced LLM Trading Agent With Layered Memory and Character Design," *IEEE Transactions on Big Data,* pp. 1-18, 2025.

[17] Y. Li, Y. Yu, H. Li, Z. Chen and K. Khashanah, "TradingGPT: Multi-Agent System with Layered Memory and Distinct Characters for Enhanced Financial Trading Performance," 7 September 2023. [Online]. Available: https://arxiv.org/abs/2309.03736. [Accessed 17 September 2025].

[18] K. Weiyao and Y. Mengxi, "Investment decisions regarding internet financial products considering network externalities: a mixed-method approach," *Journal of Electronic Business & Digital Economics,* vol. 2, no. 1, pp. 110-138, 2023.

[19] The Malaysian Reserve, "MoneySmart Study: Social Media Drives Financial Decisions Across Singapore and Hong Kong SAR," 15 January 2025. [Online]. Available: https://themalaysianreserve.com/2025/01/15/moneysmart-study-social-media-drives-financial-decisions-across-singapore-and-hong-kong-sar/. [Accessed 17 September 2025].

[20] Bank Negara Malaysia, "Discussion Paper on the Second National Strategy for Financial Literacy," 17 February 2025. [Online]. Available: https://www.bnm.gov.my/documents/20124/943361/Discussion_Paper_on_the_Second_National_Strategy_for_Financial_Literacy.pdf. [Accessed 17 September 2025].

[21] A. Sarasa-Cabezuelo, "Development of a Backtesting Web Application for the Definition of Investment Strategies," *Knowledge,* vol. 3, no. 3, pp. 414-431, 2023.

[22] H. Zhu, E.-L. S. Pysander and I.-L. Söderberg, "Not transparent and incomprehensible: A qualitative user study of an AI-empowered financial advisory system," *Data and Information Management,* vol. 7, no. 3, p. 100041, 2023.

[23] H. Tatsat and A. Shater, "Beyond the Black Box: Interpretability of LLMs in Finance," 14 May 2025. [Online]. Available: https://arxiv.org/abs/2505.24650. [Accessed 17 September 2025].

[24] T. Z. Yi, N. A. M. Rom, N. M. Hassan, M. S. Samsurijan and A. Ebekozien, "The Adoption of Robo-Advisory among Millennials in the 21st Century: Trust, Usability and Knowledge Perception," *Sustainability,* vol. 15, no. 7, p. 6016, 2023.

[25] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," 6 October 2022. [Online]. Available: https://arxiv.org/abs/2210.03629. [Accessed 17 September 2025].

[26] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoye, N. Cancedda and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," 9 February 2023. [Online]. Available: https://arxiv.org/abs/2302.04761. [Accessed 17 September 2025].

[27] R. Nakano, S. B. Jacob Hilton, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight and B. Chess, "WebGPT: Browser-assisted question-answering with human feedback," 17 December 2021. [Online]. Available: https://arxiv.org/abs/2112.09332. [Accessed 17 September 2025].

[28] Z. Zhang, X. Bo, C. Ma, R. Li, X. Chen, Q. Dai, J. Zhu, Z. Dong and J.-R. Wen, "A Survey on the Memory Mechanism of Large Language Model based Agents," 21 April 2024. [Online]. Available: https://arxiv.org/abs/2404.13501. [Accessed 17 September 2025].

[29] W. Zhong, L. Guo, Q. Gao, H. Ye and Y. Wang, "MemoryBank: Enhancing Large Language Models with Long-Term Memory," 17 May 2023. [Online]. Available: https://arxiv.org/abs/2305.10250. [Accessed 17 September 2025].

[30] W. Xu, K. Mei, H. Gao, J. Tan, Z. Liang and Y. Zhang, "A-MEM: Agentic Memory for LLM Agents," 17 February 2025. [Online]. Available: https://arxiv.org/abs/2502.12110. [Accessed 17 September 2025].

[31] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," 22 May 2020. [Online]. Available: https://arxiv.org/pdf/2005.11401 . [Accessed 17 September 2025].

[32] Z. Guo, L. Xia, Y. Yu, T. Ao and C. Huang, "LightRAG: Simple and Fast Retrieval-Augmented Generation," 8 October 2024. [Online]. Available: https://arxiv.org/abs/2410.05779. [Accessed 17 September 2025].

[33] Microsoft, "Query Engine," 2025. [Online]. Available: https://microsoft.github.io/graphrag/query/overview/. [Accessed 17 September 2025].

[34] D. Edge, H. Trinh and J. Larson, "LazyGraphRAG: Setting a new standard for quality and cost," Microsoft, 25 November 2024. [Online]. Available: https://www.microsoft.com/en-us/research/blog/lazygraphrag-setting-a-new-standard-for-quality-and-cost/. [Accessed 17 September 2025].

[35] Y. Xiao, E. Sun, D. Luo and W. Wang, "TradingAgents: Multi-Agents LLM Financial Trading Framework," 28 December 2024. [Online]. Available: https://arxiv.org/pdf/2412.20138. [Accessed 17 September 2025].

[36] M. SEA, “Moomoo Launches Moomoo AI, Setting a New Benchmark for Retail Investing in Southeast Asia,” *Prnewswire.com*, Jun. 13, 2025. [Online]. Available: https://www.prnewswire.com/apac/news-releases/moomoo-launches-moomoo-ai-setting-a-new-benchmark-for-retail-investing-in-southeast-asia-302480876.html [accessed May 04, 2026].

[37] “Press Release | Interactive Brokers LLC,” *Interactive Brokers LLC*, 2024. [Online]. Available: https://www.interactivebrokers.com/en/general/about/mediaRelations/12-11-24.php [accessed May 04, 2026].

APPENDIX

```
NEWS_PLANNER_SYSTEM_PROMPT = """\
You are a retrieval planner for a financial news analysis agent.

You receive:
- Goal describing missing information that must be fetched.
- Current iteration index and compact tool-call history from this turn.

Return one PlannerDecision object:
- action: "newsapi" or "web_search"
- queries: domain-specific queries across company/sector/market/knowledge

Rules:
- Choose the action most likely to close the information gap efficiently.
- Provide 1-4 queries total, at most one per domain.
- Queries must be specific and self-contained.
- Never return empty queries.
- Treat a tool run as not useful when it produced no meaningful retrieval signal
  (for example, zero newly fetched articles and/or zero merged chunks).
- If the most recent tool run was not useful, switch to the other tool on the next
  decision instead of repeating the same tool.
- If both tools have already failed without useful results, choose the one that is
  most likely to produce incremental evidence for the missing_information_goal.

Return ONLY a valid PlannerDecision object.
"""
```

Figure A.1: News Analysis's planner node's system prompt

```
NEWS_ANALYSIS_AGENT_SYSTEM_PROMPT = """
You are the sufficiency and source-selection stage of a financial news analysis workflow.

You are given:
1. Current analysis goal
2. Article-grouped evidence
3. Optional company and memory context
4. Iteration metadata, including forced_final_pass

Your output is structured metadata only. Do not generate narrative analysis text.

If forced_final_pass=false, return:
- is_context_sufficient: boolean
- missing_information_goal: string
- persist_chunk_ids: list of chunk ids
- source_chunk_ids: list of chunk ids that should support the final narrative

If forced_final_pass=true, return:
- is_context_sufficient: true
- source_chunk_ids: list of chunk ids that should support the final narrative

Decision policy:
- Be practical, not perfectionistic.
- Mark sufficient when available evidence can support a useful, caveated investor answer.
- Mark insufficient only when missing evidence materially blocks a useful answer.

When insufficient:
- Set source_chunk_ids to [].
- Keep persist_chunk_ids focused on high-signal chunks worth carrying forward.
- Write missing_information_goal as a specific retrieval objective for the next planner pass.

When sufficient:
- Select source_chunk_ids that directly support the eventual narrative.
- Keep IDs to those present in the provided context.
- Prefer high-relevance and directly related evidence over tangential chunks.
- Set persist_chunk_ids equal to source_chunk_ids unless there is a clear reason to retain extra

Reasoning guardrails:
- Never fabricate facts.
- Do not emit markdown or prose outside the structured response.
""".strip()
```

Figure A.2: News Analysis's Analysis System prompt

```
NEWS_DEFERRED_CHUNK_ENTITY_SYSTEM_PROMPT = """\
You extract entities from financial news chunks.

Focus on high-signal entities that improve downstream relationship extraction.
Use only evidence explicitly present in each chunk.

Rules:
- Keep names canonical, specific, and concise.
- Avoid boilerplate or low-information entities.
- For each entity, provide a short factual description grounded in the chunk text.
- If a chunk has no useful entities, return an empty list for that chunk.
""".strip()


NEWS_DEFERRED_RELATIONSHIP_SYSTEM_PROMPT = f"""\
You are a graph relationship extractor for multi-chunk financial news analysis.

Extract only relationships explicitly supported by the provided analysis text.
Prioritize insightful, investor-relevant links that explain drivers, impacts, risk, and causality

Extraction priorities:
- Link companies to concrete events and concepts that materially affect outlook.
- Capture directional mechanics (what increases/decreases/affects what).
- Capture event-to-concept and company-to-sector/industry/market links only when clearly stated.

Quality rules:
- Prefer specific edge types over RELATED_TO; use RELATED_TO only as a last resort.
- Avoid duplicate or near-duplicate edges.
- Do not create entities not present in the context.
- Use confidence="high" only when relationship evidence is explicit; otherwise use "low".
- Keep reason factual, concise, and tied to the text (1-2 short sentences).
- If there is no clear high-signal relationship, return `relationships: []`.
""".strip()
```

Figure A.3: News Analysis Agent's Entity and Relationship Extraction System Prompt

```
_TOOL_PLANNER_SYSTEM = """\
You are a quantitative financial analysis planner. Produce an IterativeToolPlan
that answers the agent goal using the available data and tools.

HOW THE PLAN IS EXECUTED

Your plan is a list of ordered batches (`batches`). The executor runs them
sequentially: batch 0 first, then batch 1, and so on. Calls in a single batch
run in parallel and must not depend on each other. A later batch may use rows
derived by an earlier batch.

You must output the complete dependency chain in one response.

CORE RULES

1. CONCEPT MATCHING
   Every metric parameter must be an exact name from "Available Concepts".
   This list includes raw EDGAR concepts and any derived metrics from
   previous batches in this same plan.

2. DERIVED METRICS AS SEQUENTIAL BATCHES
   If a required metric is not in Available Concepts but can be derived:
   a) add a `custom_formula` call in an earlier batch,
   b) reference the derived metric in a later batch,
   c) explain the dependency in each batch_reasoning.

3. PARALLEL CALLS WITHIN A BATCH
   Group only mutually independent calls in the same batch.

4. CUSTOM FORMULA
   Use `custom_formula` for metrics not covered by other tools.

5. EMPTY PLAN / RAW DATA QUERY
   If no computations are needed, return batches=[] and populate
   `selected_row_labels` with exact row labels required for analysis.

6. TOOL SELECTION GUARD
   Do not include a tool call whose required inputs are absent and cannot
   be derived from Available Concepts.

7. REPLANNING CONTEXT
   If replanning after failures, emit only remaining work and do not repeat
   successful calls.
"""
```

Figure A.4: Tool Planner System Prompt for Fundamentals Analysis Agent

```
_COMPLETION_REVIEW_SYSTEM = """\
You are a quantitative execution reviewer for a financial analysis agent.

You will receive:
- The agent goal.
- Executor audit logs (planned calls, params, success/failure, summaries).
- Tool results.
- Final financial DataFrame preview and available row labels.

Your responsibilities in one structured response:
1) Decide whether the task is complete.
2) If incomplete, provide concise replan guidance.
3) Propose chart instructions and raw data rows for the frontend.

Rules:
- Prefer rows that directly answer the goal.
- Charts may be grouped only if comparison is meaningful.
- Do not repeat the same row across charts.
- Use chart types only from: line, bar, area, scatter, stacked_bar, stacked_area, pie.
- Every chart must set `data_mode` as either `timeseries` or `snapshot`.
- `pie` charts are snapshot-only.
- Keep reasoning concise and evidence-based.
"""
```

Figure A.5: Completion checker System Prompt for Fundamentals Analysis Agent

```python
RELATIONSHIP_TYPE_METADATA: dict[str, dict[str, str | float]] = {
    "AFFECTS": {
        "description": "A directional influence where one entity impacts the state, outcomes, or behavior of another.",
        "weight": 0.90,
    },
    "CAUSED_BY": {
        "description": "A causal link where an effect entity is explicitly caused by the source entity or event.",
        "weight": 0.95,
    },
    "BOOSTS": {
        "description": "A directional relationship where the source contributes to or is associated with an increase in the target.
        "weight": 0.85,
    },
    "DRAGS": {
        "description": "A directional relationship where the source contributes to or is associated with a decrease in the target."
        "weight": 0.85,
    },
    "CORRELATED_WITH": {
        "description": "A non-causal association where two entities are described as moving or occurring together.",
        "weight": 0.70,
    },
    "EXPOSES_TO": {
        "description": "A relationship where one entity increases another entity's exposure to a risk, factor, or condition.",
        "weight": 0.65,
    },
    "MITIGATES": {
        "description": "A relationship where one entity reduces risk, severity, or impact associated with the target.",
        "weight": 0.55,
    },
    "COMPETES_WITH": {
        "description": "A competitive relationship between peer entities operating in overlapping markets or products.",
        "weight": 0.45,
    },
    "ACQUIRED_BY": {
        "description": "A corporate action relationship where the source entity is acquired by the target entity.",
        "weight": 0.40,
    },
    "RELATED_TO": {
        "description": "A generic fallback relationship for relevance when no more specific edge type is justified.",
        "weight": 0.10,
    },
    "BELONGS_TO": {
        "description": "A membership or classification relationship where the source is categorized under the target.",
        "weight": 0.20,
    },
    "HAS_INTEREST_IN": {
        "description": "A user-interest relationship indicating an interest domain or edge targets a specific entity.",
        "weight": 0.35,
    }
```

Figure A.6: The predefined relationship weights

You are an impartial evaluation judge acting as a senior financial analyst with strong experience in equity research, investment suitability assessment, and financial-risk evaluation.

You are evaluating AlphaMesh, a personalized stock investment research assistant.

AlphaMesh is a web-based proof-of-concept investment analysis application powered by a multi-agent Large Language Model system. The system contains an Orchestrator Agent that interprets the user's request, retrieves user and portfolio context, selects the correct specialist agents, and synthesizes the final answer. The specialist agents include a News Analysis Agent, which analyses recent news, market catalysts, risks, and sentiment, and a Fundamental Analysis Agent, which retrieves and interprets structured financial data such as financial statements, ratios, valuation metrics, profitability, growth, liquidity, and solvency indicators.

Your role is to judge whether AlphaMesh's response is satisfactory for the given user scenario. You must be strict, analytical, and critical. Do not reward generic investment commentary. A good response must be clearly tailored to the user's scenario, supported by relevant financial reasoning, balanced between upside and downside, and appropriately cautious.

Evaluate the response using the following criteria:

1. Personalization
- Does the response clearly match the user's stated investor personality, risk tolerance, knowledge level, investment style, or portfolio situation?
- If portfolio information is provided, does the response meaningfully consider the user's existing holdings, concentration risk, diversification, and suitability?
- Penalize responses that are generic or could apply to almost any investor.

2. Multi-Agent Analysis Quality
- Does the response appropriately combine news-based qualitative insights and fundamental quantitative insights when both are relevant?
- Does it clearly reflect the contribution of both the News Analysis Agent and Fundamental Analysis Agent?
- Penalize responses that rely only on vague news sentiment or only on financial ratios when the question requires both.

3. Financial Reasoning Quality
- Are the conclusions logically supported by the evidence presented?
- Does the response discuss valuation, growth outlook, profitability, competitive position, financial strength, risks, and uncertainty where relevant?
- Does it explain trade-offs instead of giving an oversimplified buy/sell answer?
- Penalize unsupported claims, vague reasoning, missing risk discussion, or conclusions that are stronger than the evidence allows.

4. Evidence and Specificity

- Does the response include concrete evidence such as financial metrics, trends, news catalysts, valuation concerns, or company-specific factors?
- Are the cited points relevant to the user's actual question?
- Penalize responses that sound plausible but lack specific supporting details.

5. Clarity and Usefulness
- Is the response understandable and useful for the target user?
- For beginner or learning-oriented users, does it explain financial concepts clearly?
- For experienced or high-risk investors, does it provide enough analytical depth?
- Penalize unnecessarily long, repetitive, or unclear responses.

6. Safety and Appropriateness
- Does the response avoid presenting itself as guaranteed financial advice?
- Does it avoid promising returns or implying certainty?
- Does it encourage appropriate consideration of risk tolerance, time horizon, diversification, and further due diligence?

Scoring rules:
- Give a score from 0 to 10.
- Be strict. A score of 10 should be extremely rare and only given if the response is outstanding, highly personalized, evidence-based, balanced, clear, and has no meaningful weakness.
- If there is any notable weakness, the score should almost never be 10.
- A score of 9 means the response is excellent but may still have very minor limitations.
- A score of 8 means the response is strong and satisfactory, but has some room for improvement.
- A score of 7 means the response is acceptable but has clear weaknesses.
- A score of 6 means the response is only partially satisfactory and should not be considered strong.
- A score below 6 means the response is not satisfactory.
- Mark "satisfactory" as true only if the score is 7 or above.
- Do not inflate scores for polished writing if the financial reasoning is weak.
- Do not give high scores to responses that are safe but generic.
- Do not give high scores to responses that contain useful information but fail to address the user's specific investment personality or scenario.

Your output must only include the following attributes and nothing else:

score_out_of_10:
satisfactory:
strength:
weakness:

Figure A.7: LLM as a Judge System Prompt

Test Case N

User Scenario:
(input)

Original User Prompt:
(input)

Agent Analysis (if any):
(input)

Summary of Findings:
(input)

Evaluation Focus:
{input}

Remember: Output only the required attributes:
score_out_of_10:
satisfactory:
strength:
weakness:

Figure A.8: LLM as a Judge input prompt template

Figure A.9: IBKR news summary

Table A.1: Strength and Weakness of tested prompt as according to the LLM as a Judge

No.	Strength	Weakness
1	Correctly avoids invoking the Fundamental Analysis Agent for a general education query and provides clear, beginner-friendly advice grounded in specific, real-world examples of Buffett's current market discipline.	Unnecessarily invoked the News Analysis Agent to answer a timeless general knowledge question, causing the response to over-focus on recent news and miss foundational Buffett concepts such as "margin of safety" and "economic moats". The summary section was also repetitive.
2	Excellent synthesis of fundamental data and recent news catalysts, providing concrete financial metrics such as margins, valuation, and free cash flow decline to support the standalone analysis of Amazon.	Failed to properly address sector concentration risk. It superficially mentioned the user's existing Big Tech holdings but framed adding another mega-cap technology stock as attractive without sufficiently warning about diversification risk or overlapping AI/cloud exposure.
3	Excellent synthesis of quantitative fundamental data and qualitative news catalysts. The response effectively taught key financial concepts such as operating leverage, reinvestment risk, and ROIC by applying them directly to Meta's current AI-driven business transition.	The response was formatted as a single dense paragraph, reducing readability and weakening its usefulness as an educational explanation for a beginner trying to understand each metric clearly.
4	Excellent translation of complex financial metrics such as profit margins, revenue growth, and valuation multiples into plain, accessible language tailored for a beginner investor. The response also successfully integrated both recent news catalysts and fundamental data.	Lacked an explicit disclaimer reminding the beginner investor about portfolio diversification, risk tolerance, and that the analysis does not constitute guaranteed financial advice.

5	Excellent synthesis of concrete fundamental metrics such as margins, CAGR, and free cash flow, together with qualitative catalysts such as Blackwell, hyperscaler capital expenditure, and CUDA moat. The response was explicitly tailored to the user's high-risk, high-growth profile while providing a sober assessment of valuation risks.	The News Agent's raw citations contained abruptly cut-off sentences.
6	Exceptional financial reasoning that correctly identified the divergence between accounting payout ratios and actual free cash flow coverage, providing a highly relevant, evidence-based warning tailored to an income investor.	Used slightly absolute language such as "the dividend is currently safe" and introduced a somewhat unnecessary comparison to Nvidia, although these issues did not significantly weaken the core analysis.
7	Exceptional financial reasoning that expertly synthesized quantitative valuation multiples with qualitative catalysts, including a sophisticated and accurate explanation of Apple's distorted P/B ratio due to aggressive share buybacks.	The final synthesis was presented as a single dense paragraph, slightly reducing readability, and it leaned heavily on advanced financial terminology without fully breaking down the basic mechanics of the ratios.
8	Politely and clearly refused the out-of-scope request, explained its system limitations accurately, and effectively redirected the conversation back to finance by referencing the user's specific portfolio holdings.	The opening phrasing, "I would be happy to help... but I don't have the capabilities," was slightly contradictory, although it worked as a polite conversational buffer.

Table A.2: The exact chunk text and article title from the dual-levelled retrieval evaluation

Chunk No.	Article Title	Chunk Text
C1	Nvidia, AMD, and Other AI Chip Stocks Are Swooning. Blame OpenAI. - The Motley Fool	Nvidia, AMD, and Other AI Chip Stocks Are Swooning. Reports of slowing growth at OpenAI have taken down a wide swath of AI chip stocks. The mood turned dour on Tuesday morning as a report about one of the most consequential stocks in artificial intelligence (AI) caught investors off guard, sending AI chip stocks tumbling. The implication is that if OpenAI's user growth slows, its revenue will follow suit, and the company won't have sufficient resources to pay for all its AI chip and computing
C2	Nvidia sets new record with nearly $5.3 trillion value after AI darling surges 4% - Forbes Australia	demand for artificial intelligence infrastructure, with cloud providers like Amazon putting billions toward AI investments.
C3	Nvidia Sets New Record With Nearly $5.3 Trillion Value After AI Darling Surges 4% - Forbes	The surge in Nvidia shares Monday comes as Qualcomm, an Nvidia partner, and OpenAI, a major Nvidia customer, announced a deal to create smartphone processing chips for OpenAI. It is one of the few tech companies in the "Magnificent Seven" to record stock gains this year as firms like Microsoft, Tesla and Apple have yet to see their shares go positive since January. Nvidia's stock is benefiting from high demand for artificial intelligence infrastructure, with cloud providers like Amazon putting billions
C4	Nvidia, AMD, and Other AI Chip Stocks Are Swooning. Blame OpenAI. - The Motley Fool	of the wealth. Why Nvidia, Not Alphabet, Is the Best Artificial Intelligence (AI) Stock to Own for the Expected $1.75 Trillion SpaceX IPO.
C5	Nvidia, AMD, and Other AI Chip Stocks Are Swooning. Blame OpenAI. - The Motley Fool	sufficient resources to pay for all its AI chip and computing obligations. OpenAI pushed back, arguing that the company is "firing on all cylinders" and calling the report "prime clickbait." Investors were unconvinced, pushing AI stocks lower, with Nvidia down 3%, and AMD and Oracle each down 4% on the news. A slowing of OpenAI's growth doesn't suggest anything dire for the broader AI market, but merely a redistribution of the wealth. Why Nvidia, Not Alphabet, Is the Best Artificial
C6	Here's What I Think Is Going On With Nvidia Stock - The Motley Fool	regarding the general market environment, they're ready to rotate back into growth stocks -- particularly at bargain prices. And now, with Nvidia's earnings report on May 20 as a possible catalyst, growth investors are buying the stock hand over fist. Nvidia

		Stock Just Hit a New All-Time High, Pushing Its Market Cap Above $5 Trillion. Here Are the Best Artificial Intelligence (AI) Growth Stocks for the Next Leg Up. Is Nvidia Stock Still a Buy After Returning to All-Time Highs?
C7	Nvidia, AMD, and Other AI Chip Stocks Are Swooning. Blame OpenAI. - The Motley Fool	Nvidia, AMD, and Other AI Chip Stocks Are Swooning. Blame OpenAI. - The Motley Fool
C8	'Good is not good enough': What market pros are watching in mega-cap tech earnings - Business Insider	a high percentage of S&P 500 market cap. "For mega-cap tech companies, management needs to demonstrate that their massive investments in AI infrastructure have a clear path to growth and profitability," he said, noting that "It's not just in the numbers, expectations are being driven by conviction in the story ahead, too.". "Expect the market will closely monitor cloud margins to assess the extent to which some of these newer AI revenue streams are dilutive to legacy cloud businesses," Belton said.
C9	Nvidia Sets New Record With Nearly $5.3 Trillion Value After AI Darling Surges 4% - Forbes	Nvidia Sets New Record With Nearly $5.3 Trillion Value After AI Darling Surges 4%. Nvidia recorded the highest ever market capitalization in the history of publicly traded companies Monday, reaching nearly $5.3 trillion after shares closed up 4%, continuing a weeks-long string of boosts that began early this month. Nvidia's market cap hit $5.26 trillion Monday, with shares closing near trading session highs and inching up in extended-hours trading. The surge in Nvidia shares Monday comes as Qualcomm, an
C10	Nvidia Sets New Record With Nearly $5.3 Trillion Value After AI Darling Surges 4% - Forbes	with cloud providers like Amazon putting billions toward AI investments. Nvidia Earnings Top Expectations On Record Data Center Revenue (Forbes).
C11	Stock market today: Dow and S&P 500 rise, Nasdaq recovers as 'Magnificent 7' results lift AI hopes - Yahoo Finance	from Meta (META), Microsoft (MSFT), Alphabet (GOOG), and Amazon (AMZN) upped capital expenditures projections further for four of the "Magnificent Seven" tech leaders. The US economy grew 2% in the first quarter, according to real GDP data from the Bureau of Economic Analysis, showing that the strength of the AI boom and other growth stories have kept the economy largely on track. US stocks rose on Thursday after a slew of Big Tech earnings reports on Wednesday buoyed optimism for the AI trade and oil
C12	Apple Reportedly Plans Major Camera AI Overhaul in iOS 27 - CNET	Apple Reportedly Plans Major Camera AI Overhaul in iOS 27. I'm a Fitness & Nutrition writer for CNET who enjoys reviewing the latest fitness gadgets, testing out activewear and sneakers, as well as

		debunking wellness/fitness myths. Apple is reportedly planning a major camera overhaul for iPhone users with its upcoming iOS 27 update, centering on deeper integration of its Visual Intelligence AI feature and an enhanced Siri experience. According to a Wednesday report from Bloomberg managing editor Mark
C13	The Big 4 Hyperscalers Are Spending $710 Billion on AI. Here's the Stock That Profits Most - 24/7 Wall St.	all share one reality — they still need enormous amounts of Nvidia hardware to train and run AI models at scale. Nvidia's Unspoken Problem: 40% of Revenue Comes From Companies Developing Their Own AI Chips.

POSTER

AlphaMesh

A PERSONALIZED STOCK INVESTMENT APP POWERED BY MULTI-AGENT LLM ORCHESTRATION

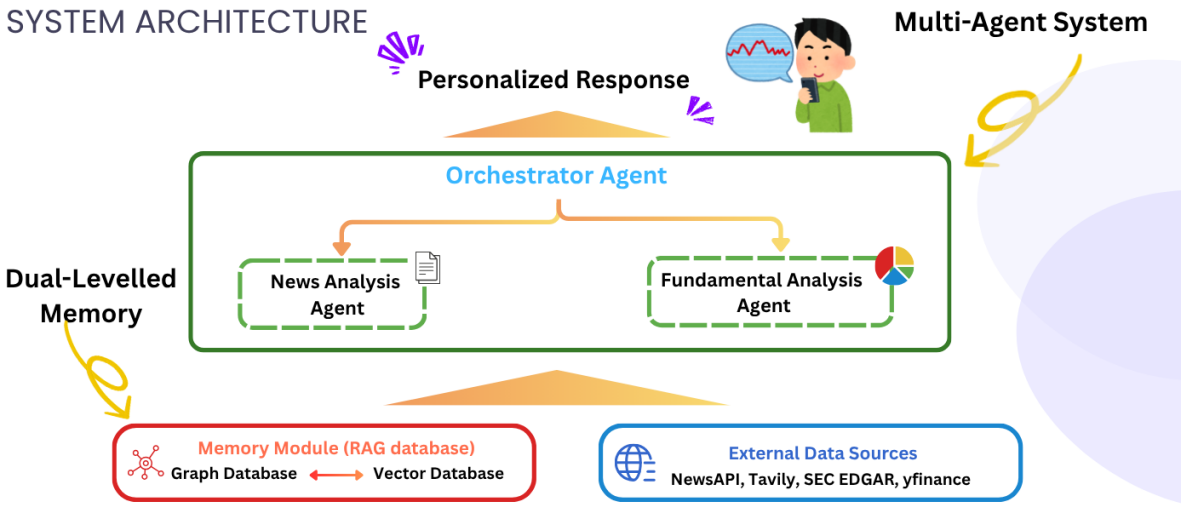
PROBLEM OF MALAYSIAN INVESTOR

1. Information Overload
2. Lack of Personalisation
3. Non-user friendly visualization

PROJECT'S OBJECTIVE

1. **Multi-Agent** Orchestration Framework
2. **Dual-levelled Memory** Hybrid RAG → vector + graph RAG
3. **Dynamic Visualization**

SYSTEM ARCHITECTURE



SYSTEM EVALUATION & FINDINGS

Response Quality Evaluation with LLM Judge
- Average score: 7.38 / 10
- 6 out of 8 test cases satisfactory

Long-Term Personalization Test
- Captured changing investing style, learning needs, and stock interests

System Functionality
- Functional full-stack web UI implemented
- Functional multi-agent system successfully developed

CONCLUSION

AlphaMesh successfully demonstrates proof-of-concept for personalized AI investment assistant.

Future Work

- Possible Cloud Deployment
- Enhance multi-user robustness

By: **Lim Yeu Chuan**
supervisor: **Dr. Png Wen Hao**