Ollscoil na Gaillimhe
University of Galway

AI STOCK ANALYSIS TOOL

College of Engineering and Science Bachelor of Science (Computer Science and Information Technology)

Skehan, Gavin
Computer Science and Information Technology

Table of Contents

Table of Contents 1
Chapter 1. Introduction 2
1.1 Document Outline 2
1.2 Project Background 3
1.3 Problem Statement 3
1.4 Solution Overview: Tradeskee 3
1.5 Project Objectives 4
1.6 Feasibility Study 4
1.5 Stakeholders 7
1.6 Project Management 7
Chapter 2. Requirements 9
2.1 Introduction 9
2.2 Functional Requirements 9
2.3 Non-Functional Requirements 10
2.4 Constraints and Assumptions 11
2.5 Technologies Rational 11
2.4 User Experience (U) and Interface (UI) Requirements 12
2.5 Development Considerations 13
2.6 Constraints 14
Chapter 3. System Design 15
3.1 System Architecture Overview 15
3.2 Data Layer Design 17
3.3 AI Analysis Layer 18
3.4 Frontend-Backend Integration 21
3.5 Design Justifications 24
3.6 Alignment With Requirements 26
3.7 Key Features and User Interaction 27
Chapter 4. Implementation 28
4.1 Introduction 28

4.2 Development Environment....28
4.3 Implementation Approach....30
4.4 Backend API Development....32
4.4.1 Data Collection....32
4.4.2 Data Cleaning and Filtering....33
4.4.3 Data Storage and Caching....33
4.4.4 Data Processing and Function Execution....34
4.5 Backend LLM Integration & Multi-Model Design....34
4.6 Challenges and Solutions....35
4.7 Summary....37
Chapter 5. UI, Testing and Evaluation....37
5.1 Introduction....37
5.2 UI Design....37
5.3 Testing and Debugging....38
5.4 Sample Output & Evaluation (DeepSeek AI Analysis) [2]....40
Chapter 6. Conclusion....45
6.1 Challenges....45
6.2 Lessons Learned....45
6.3 Future Work....45
6.4 Final Conclusion....45
Chapter 7. References....46

Chapter 1. Introduction

1.1 Document Outline

This document serves as a formal report for the module CT413, undertaken within the BSc Computer Science and Information Technology programme at the University of Galway. The project, titled Tradeskee, is a personal initiative inspired by a deep interest in financial markets and supported by technical skills' development during an internship in backend development and data science.

This document provides a comprehensive account of the projects' lifecycle, from conception and research through design and implementation to testing and evaluation. It also reflects on personal learning experiences and outlines future directions for the system. The core of this

project is not just technical implementation but also understanding and addressing a real-world user problem, enabling beginner investors to make more informed decisions through the use of AI.

1.2 Project Background

The idea of this project stemmed from the intersection of my technical development skills and my passion for finance, investing and economics. While my formal education has focused on computing, I have long been drawn to the complexity and strategy of the financial world, particularly how professionals assess and value companies using both fundamental and technical analysis.

During my internship, I worked with API-driven data collection and large language models (LLMs), including Random-Augmented Generation (RAG) exposure. Additionally, I consulted numerous academic papers and online resources to deepen my understanding of RAG and prompt engineering during placement. While this report focuses on practical implementation, it is important to know that there is extensive research in the area of RAG and LLM relationships.

Reading these papers inspired me to explore the intersection of finance and AI. This enabled me to envision an AI-powered tool capable of real-time market analysis, bridging the gap between casual interest and professional-grade insight by using AI as the connective tissue.

1.3 Problem Statement

New retail investors often face a steep learning curve. In fact, more than 70% of DIY investors lose money [14]. The volume of information available across Reddit forums, Discord groups, YouTube videos, blogs and TikTok is overwhelming and often misleading. These sources tend to lack transparency, structure and evidence-backed analysis. Lack of financial punctuality is what leads to high percentages of people losing money from poor investment decisions through poor financial psychology and buying based on unreputable sources.

Unlike professional investors who use rigorous data-backed approaches, beginners frequently make emotionally driven decisions without truly understanding key indicators or company fundamentals. As a result, many end up losing money early on, discouraging them from ever investing again. This is a problem.

Additionally, even the most powerful LLMs like Gemini or GPT-4 are limited by their static training data, they don’t have up-to-date knowledge of current stock prices, earnings releases or recent news events. Without external context their responses can be outdated or generic.

1.4 Solution Overview: Tradeskee

Tradeskee is a web-based, AI-powered financial assistant that combines live market data, retrieval-augmented generation (RAG), and LLM reasoning to deliver reliable, explainable stock analysis. Users can input a stock ticker and receive:

- Cleaned financial and technical data retrieved and storage via the Alpha Vantage API
- A custom RAG pipeline that injects historical, live and economic data into the LLM prompt for contextual analysis

- A two-stage LLM interaction which includes initial analysis and deep-thinking analysis (Gemini, DeepSeek)
- A React-based frontend that presents results in a modern, intuitive user interface (UI)

Tradeskee is designed to act as a learning tool, guiding beginner investors to begin thinking like professional investors, doing so using real data.

1.5 Project Objectives

The primary goal of this project is to build an AI-powered trading analysis tool through various technology utilization, including a multi-model approach, RAG adaptations, prompt engineering, deep-thinking explainable AI and more. The intelligent tool leverages AI to deeply analyze fundamental, technical and economic stock data.

Specifically, the system will:

1. Retrieve and process historical and live financial data (fundamental + technical) from reliable sources like Alpha Vantage.
2. Inject this data into prompts for LLMs, creating a lightweight but effective RAG pipeline without fine-tuning or retraining.
3. Use a two-stage LLM architecture where Gemini performs initial summarisation and DeepSeek provides a refined response with deeper insight.
4. Provide accessible, clear financial commentary for users through a web frontend.
5. Support education and exploration, helping users understand how financial analysts approach investing decisions.
6. Optimize performance and latency using caching, MongoDB for storage, and efficient API routing.
7. Remain extensible, allowing future versions to integrate more asset classes, sentiment data, or UI features.

1.6 Feasibility Study

This section evaluates the practical viability of the Tradeskee project by examining key feasibility factors: technical, economic, legal, and operational. It identifies whether the required tools, technologies, and resources are available and suitable for successful implementation.

1.6.1 Technical Feasibility

To ensure the technical feasibility of Tradeskee, I assessed the availability of data sources, AI tools, infrastructure requirements and overall project complexity.

Data Availability & Sources

The foundation of this project depends on the access to diverse financial data. Alpha Vantage is the primary API provider. Alpha Vantage provides an array of data to be fetched and manipulated. These datasets include:

- Fundamental Data: Includes metadata, financial statements (balance sheets, income statements, etc), earnings reports and macroeconomic indicators. These are crucial for evaluating a company's financial health, often ignored by beginners

- Technical Data: This covers trading volumes, open/closing prices, highs/lows, price movements, simple moving averages, exponential moving averages, etc. These metrics assist in identifying price patterns and short-term momentum.
- News & Sentiment: Alpha Vantage also provides financial news articles paired with sentiment score, giving insight into market perception and events influencing stock markets.

Alpha Vantage's monthly tiered pricing model affects project efficiency. During development, I used the free tier (25 API calls/day), which was sufficient for proof-to-concept testing, but limited during iterative experiments. Upgrading to the $50 tier (75 calls/minute) or higher would significantly reduce wait times and enable more comprehensive data collection.

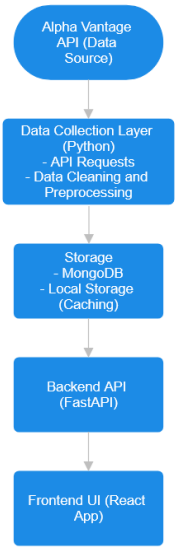


Figure 1: Data Acquisition Pipeline

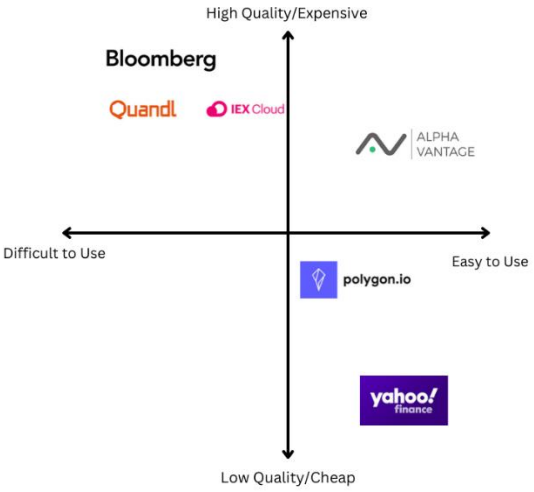


Figure 2: Feasibility Chart: Data API

AI Selection

The intelligence behind Tradeskee relies on the integration of local LLMs using Ollama, allowing analysis without depending on external API's or cloud-based LLMs. The choices reduce cost and increases data control. The current AI stack includes:

- Gemini: Performs the initial prompt interpretations and financial data summary
- DeepSeek: Process the previous LLM output to generate deeper, contextual financial insights, effectively creating a multi-stage analysis pipeline

This design mirrors Retrieval-Augmented Generation (RAG) architecture, where the structured and unstructured external data is injected into LLM prompts. Unlike traditional LLMs that solely rely on static pre-trained data, Tradeskee uses live financial data, ensuring insights are real-time and context-aware.

Infrastructure Requirements

The following technologies and resources are utilized:

- Development Stack: Python for backend development, FastAPI for API endpoints, React for frontend development and Ollama for LLM handling
- Compute Resources: Initial stages are handled locally using a GPU-enabled system. Future scalability will require cloud platforms such as AWS or Azure for deploying models and managing larger data streams
- Database & Storage: MongoDB (NoSQL) is used to persist financial data, offering flexibility for querying time-series datasets

1.4.2 Economic Feasibility

This aspect considers the financial sustainability of Tradeskee's development and long-term operation.

Cost Estimation

- API Usage: This project uses Alpha Vantage, with an MVP (Minimum Viable Product) tier costing $50/month. More extensive use cases would require the premium tier of $250/month.
- Compute & Hosting: Cloud deployment could incur additional costs ranging from $100 to $500/month depending on the provider and traffic
- Development Costs: As the sole developer, labour costs are reduced to zero, apart from personal investment
- Hosting and Maintenance: Future deployment to users would require budgeting from hosting servers and maintenance cycles

Revenue and Business Model

To monetize the tool, several pathways are being considered:

- Freemium Subscription Model: Provide limited access to financial insights for free, with premium unlocking detailed reports, advanced metrics and enhanced AI analysis
- AI Insights as a Service: Allow third-party platforms to integrate Tradeskee's stock analysis engine via API access
- Consulting Model: Offer personalized reports for users or small investment groups, including profiles, portfolio suggestions and earnings analysis

1.4.2 Legal & Ethical Feasibility

As the platform delivers financial insights, it must operate responsibly and legally within a sensitive regulatory landscape.

- Compliance Requirements: The app does not give direct financial advice but must still align with SEC (If deployed in the US) regulations or relevant regulations to avoid violations related to financial advising
- Data Privacy: With plans to target EU users, GDPR compliance is mandatory, especially if user profiles, financial preferences, or trading behaviour are stored
- Ethical AI Use: The system must communicate the limitations of AI-generated predictions. Transparency is key to ensuring users do not interpret output as guaranteed financial outcomes

1.4.3 Operational Feasibility

This section evaluates the practical usability and long-term maintenance of Tradeskee

- User Needs: The application is designed to cater to three core user groups:
    - Beginner Retail Investors: Need accessible and educational insights
    - Active Traders Looking for AI-powered augmentation for fast decision-making
    - Institutional Analysts: Could use Tradeskee's analytical pipeline to validate data-driven strategies
- Usability & Platform Design: A web interface ensures accessibility for non-technical users, while backend APIs can be integrated into custom tools or trading dashboards
- Scalability: MongoDB supports high-volume storage, and the system design accommodates batch data processing. However, real-time data streaming (WebSocket's for live price feeds) is flagged as future work

1.5 Stakeholders

The primary stakeholders of Tradeskee are aspiring investors and retail investors who want to transition from intuition-based investing to data-driven decision-making. The platform serves as both a learning resource and a real-time assistance for financial analysis, tailored to beginner-level comprehension but scalable to more advanced users.

As the developer and primary stakeholder, I aim to create a tool that also enhances my trading outcomes by combining AI interpretation with robust financial metrics.

Additionally, institutional users or financial analysts could benefit from the system's multi-model analysis pipeline, allowing for deeper sentiment and fundamental interpretations. Their use cases may differ (E.g. integrating with internal tools), but they could serve as important adopters or even early clients in a commercialized plan.

1.6 Project Management

Managing an ambitious technical project within an academic timeframe requires a balanced strategy. With guidance from my supervisor, the project was broken into clear, achievable milestones, allowing adaptability in scope as progress evolved. Feature prioritization was essential. Core functionalities such as data retrieval, AI prompting and frontend design were developed first. Once those were in place, enhancements such as multi-model pipelines and premium data insights were layered on.

Refactoring was a continuous process. Feedback-driven improvements were applied iteratively, adopting an agile workflow kept the project manageable but also ensured steady progress despite minor setbacks in data limitations and model inconsistencies.

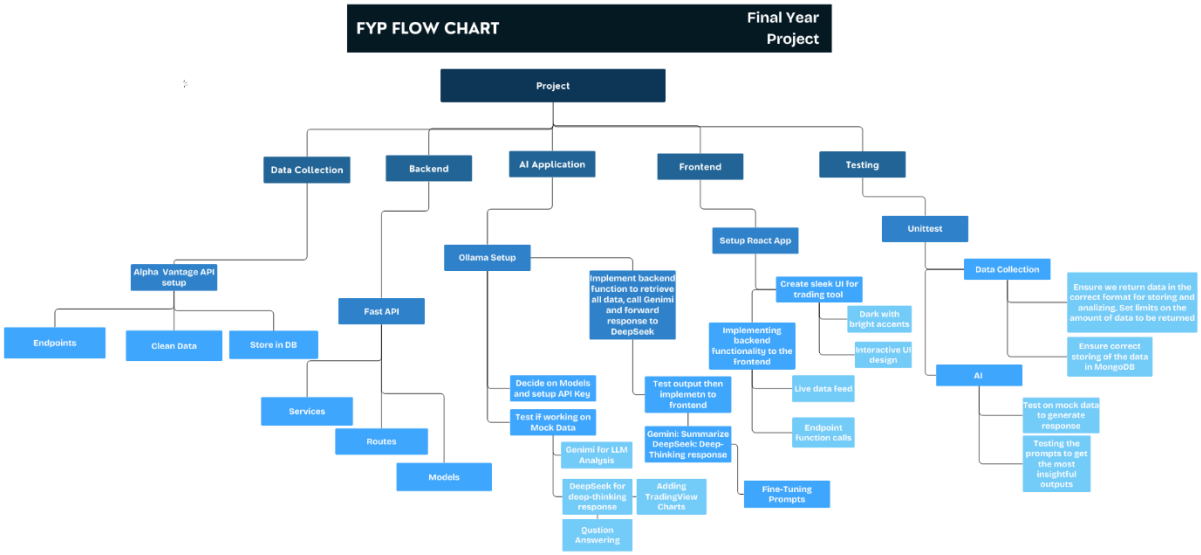


Figure 3: Project Management Flow

Git and GitHub

Version control was key to managing my project with so many moving parts. I used skills from my internship to successfully implement branching mechanisms into my project. Isolating features and bug fixes away from my development branch enabled smooth progression, also providing me with the ability to roll back or review changes made from the previous working branches. Regular clear and informative commits were used within feature branches when milestones were hit to ensure I had a checkpoint if the code crashed. Visual Studio Code was used as the Integrated Development Environment (IDE) and I utilized PyCharm's easily to use debugging and testing features.

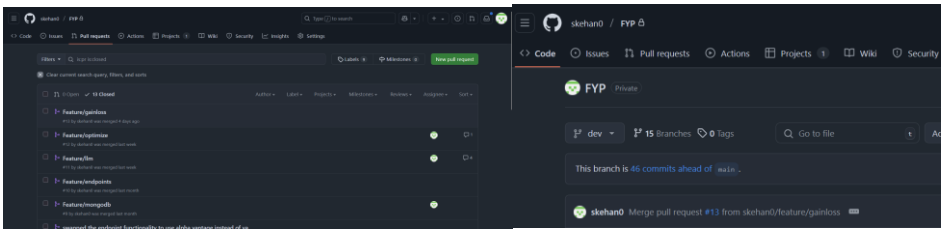


Figure 4: Pull Requests and Branching

Notion

Notion was used to document everything from task management, knowledge management, and progress management, acting as a hub for communication with my supervisor. Utilizing

Notion's capabilities to provide clear information separated into pages and sections for specific aspects of the project. Source to document all the progress and provide added information of code and screenshots from the application.

Agile Methodology

An agile methodology was adopted from the beginning of the project, from experience working in two-week sprints from my internship, this was the obvious choice. Initially adopting a two-week sprint plan and meetings in semester one allowed to me focus on the research and learning side of the project, setting achievable goals mainly to start making little progress on the project before semester two where I started weekly sprints and meetings with my supervisor, exercising sprint reviews and sprint plannings within the weekly sessions. Testing was carried out in tandem with each feature.

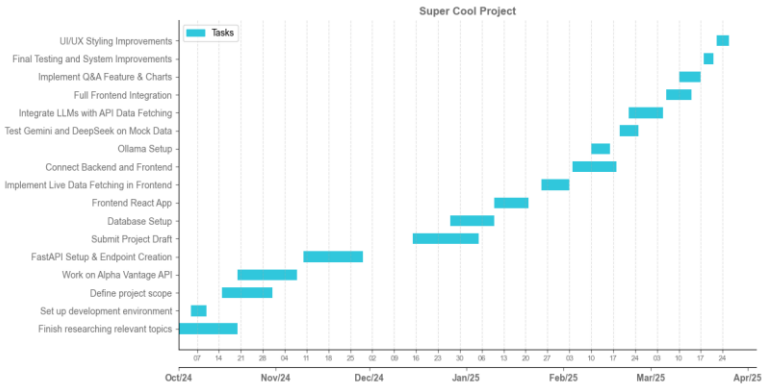


Table 1: Gantt Chart - Project Goals & Deadlines

Chapter 2. Requirements

2.1 Introduction

This chapter outlines the core requirements necessary for developing the AI-powered stock analyser. It details both the functional and non-functional requirements, as well as the desired user experience (UX) and user interface (UI) considerations. The development constraints and the key factors that influence implementation decisions are also discussed. These requirements ensure the system is efficient, scalable and user-friendly while delivering accurate financial insights. The user's needs were kept central throughout the project, focusing on accessibility, actionable insights and an intuitive experience.

2.2 Functional Requirements

The following functional requirements define the intended behaviour of the system in response to user and system events

2.2.1 Data Retrieval

- FR1: The system shall retrieve live stock data from external APIs at scheduled intervals (e.g. every hour)

- FR2: When data retrieval fails, the system shall log the error and retry up to 3 times before notifying the user of temporary unavailability
- FR3: Upon successful retrieval, the system shall store the stock data in MongoDB database under the appropriate collection for fast access

2.2.2 User Query and Analysis Processing

- FR4: When a user submits a stock, the system shall retrieve relevant data from the database and inject it into the AI prompt in the form of a cleaned data array
- FR5: The system shall process user queries using the Gemini AI model and return a summarised actionable insight
- FR6: When operating in offline mode, the system shall process user queries using Ollama running locally

2.2.3 API and Backend Functionality

- FR7: The system shall expose an API endpoint using FastAPI to handle incoming user requests and return analysis results
- FR8: When the API receives invalid or malformed requests, it shall respond with appropriate error messages

2.2.4 Data Storage and Management

- FR9: The system shall organize stock data into collections based on individual endpoints within MongoDB
- FR10: When new data is inserted, the system shall overwrite outdated entries to maintain data relevance

2.2.5 Model Selection and Flexibility

- FR11: The system shall allow switching between different AI models to accommodate performance or availability needs
- FR12: When Gemini AI is unavailable, the system shall def

2.3 Non-Functional Requirements

2.3.1 Performance

- NFR1: The system shall return AI-generated stock insights within 10 minutes for locally processed queries (Ollama), providing sufficient time to analysis and process information
- NFR2: API response time shall not exceed 3 seconds under standard load conditions
- NFR3: Data retrieval from external APIs shall complete within 3 second per request

2.3.2 Reliability and Availability

- NFR4: The system shall maintain at least 99% uptime during operational hours
- NFR5: In the event of external API downtime, the system shall continue to operate using cached or stored data

2.3.3 Scalability

- NFR6: The system architecture shall support the addition of new AI models and data sources with minimal configuration changes

- NFR7: MongoDB shall efficiently handle up to 100,000 records of stock data without performance degradation

2.3.4 Usability

- NFR8: The system interface shall follow modern UI design standards for clarity and ease of use
- NFR9: The system shall provide clear, user-friendly error messages and loading indicators during data processing

2.3.5 Security

- NFR10: The API shall validate all incoming requests to prevent injection attacks
- NFR11: Access to sensitive endpoints shall require API authentication tokens

2.4 Constraints and Assumptions

- C1: The system assumes stable internet connectivity for real-time data retrieval and API integration
- C2: The project assumes access to public or freemium stock data API's for initial development and testing
- C3: Ollama and other local AI models require sufficient local compute resources (minimum 8GB RAM, modern CPU)

2.5 Technologies Rational

- **Fast API [6]**
    - FastAPI is an open-source web framework for building high-performance APIs using Python. It was chosen for this project due to its ease of use, asynchronous capabilities and built-in support for automatic API documentation. FastAPI enables the development of data retrieval endpoints using the FastAPI router, providing a clear and modular project structure that enhances scalability and maintainability. FastAPI facilitates seamless integration with the frontend, simplifies testing and debugging, and ensures efficient backend development.
- **MongoDB [5]**
    - MongoDB was selected as the database due to prior experience and its flexible schema-less structure, which is ideal for handling dynamic financial data. Each data endpoint has a dedicated collection within the database, where retrieved stock data is stored for fast access and analysis.
- **Ollama (LLM Engine) [3]**
    - Ollama is an open-source tool that allows for running large language models (LLMs) locally and efficiently. It is ideal for developing a minimum viable product (MVP), as it allows offline inference without incurring costs from external API calls. This setup ensures low-latency responses, making Ollama a reliable component for AI-powered stock analysis.
- **Gemini AI [4]**
    - Gemini AI, developed by Google, is a powerful large language model (LLM) with multiple variations suited for different workloads. This project integrates

Gemini 4b (3.3GB), a lightweight yet efficient model that balances performance and resource efficiency. While more advanced models exist, Gemini 4b was selected for its speed and capability in handling stock market queries and trend analysis. The model receives live market data from the database, which is injected into the prompt along with the user query, essentially acting as part of a Retrieval-Augmented Generation (RAG) system.

- **DeepSeek [2]**
    - DeepSeek is a next-generation LLM designed for advanced natural language processing tasks. Integrated into the system for comparative analysis, DeepSeek enables evaluation of different model performances and contributes to ensuring accurate financial insights. It offers diverse capabilities, such as in-depth market summarisation and sentiment analysis, enhancing the system's overall analytical capabilities.

2.4 User Experience (U) and Interface (UI) Requirements

The success of the AI-powered stock analysis platform heavily depends on providing a modern and engaging user interface. The following requirements outline the desired user experience and visual design objectives for the system.

UX/UI Objectives:

- Deliver a clean, uncluttered dashboard for clear presentation of analysis results
- Enable users to search for and view stock analysis easy
- Display sentiment analysis results alongside core stock data for quick context
- Use visual indicators to highlight key insights and actionable data
- Ensure the design feels futuristic and sleek, balancing aesthetics with functional clarity
- Maintain dynamic responsiveness, keeping users informed during backend AI processes

The UX design prioritises smooth interactions while subtly showcasing backend complexity without overwhelming the user. Ensuring transparency during AI processing is critical for maintaining user engagement, especially given the response time of AI models and deep-thinking models.

Design Inspiration & Prototyping

Leading fintech platforms such as Crypto.com and Trading212 utilize dark themes with vibrant accents to evoke a premium, high-tech feel. Inspired by these designs, the application will adopt:

- A dark blue base for a modern, professional appearance
- Lighter blue accents for contrast and improved readability
- A minimalist layout to maintain clarity and allow for future design enhancements

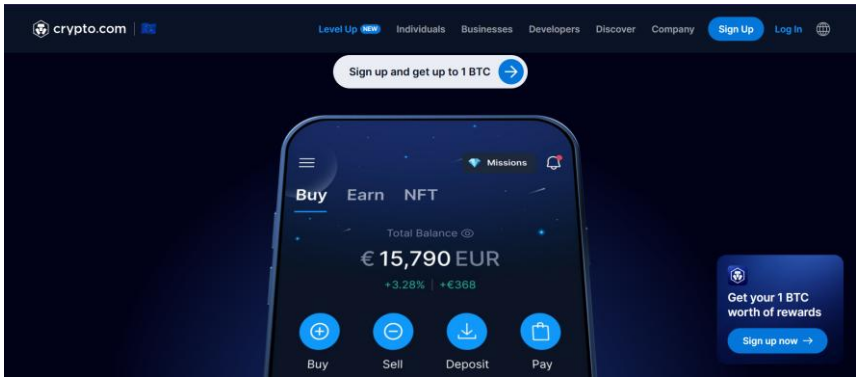


Figure 5: Crypto.com

Figure 6: Initial Frontend Prototype

To ensure a smooth user experience during LLM processing times, the interface will include progress indicators, reassuring users that processes are actively running. Additionally, simple animations and subtle visual feedback will enhance usability.

Planned UI/UX Features:

- Visible AI process indicators (e.g. loading animations, streamlines output)
- Option to save analysis in multiple file formats
- Real-time data updates where possible
- Simplistic, minimalistic design to avoid cognitive overload
- Mobile-responsive layout for accessibility across devices

2.5 Development Considerations

Scalability

The project is designed with future scalability in mind. It should not remain a standalone prototype but serve as a foundation for continuous feature expansion. To achieve this, the codebase will follow principles of modularity and encapsulation, ensuring new features can be integrated seamlessly without compromising the system's integrity.

Performance & Optimization

Minimizing latency and jitter is crucial for providing a responsive user experience. Several techniques are adopted to enhance performance:

- Caching mechanism: Temporarily store recently fetched data (Axios) to reduce redundant API calls

- Data pipeline: Implement a process to store retrieved data in a dedicated collection, allowing functions to check local data before making external API requests
- Model efficiency: Utilize the latest AI model versions for optimal accuracy. Where possible larger models will be tested to stress-test the system, with the flexibility to scale up to more powerful models in future iterations

Frontend Design

While the backend delivers the core functionality, the frontend plays a vital role in user engagement. A visually appealing interface, often favoured by traders, will use dark themes with contrasting accents to convey professionalism. The design will remain clean and minimalistic, ensuring smooth interaction and intuitive navigation.

2.6 Constraints

Technical Constraints

- API limitations: The Alpha Vantage free tier restricts usage to 25 calls per day, requiring efficient data handling and minimal redundant calls
- Computational resources: Development is restricted to local environments without cloud-based GPUs. Larger LLM models exceeded local hardware capacity, necessitating the use of smaller models
- Adaptation of APIs: Initial integration with Yahoo Finance API proved insufficient, prompting a switch to Alpha Vantage for better compatibility
- Testing limitations: To manage API quotas, testing was conducted using smaller datasets to maximise test coverage without exhausting daily limits

Despite these constraints, the challenges fostered growth in problem-solving, debugging and system testing throughout the project lifecycle.

Time Constraints

Balancing academic commitments with project development posed significant challenges. Time management strategies included:

- Allocating dedicated weekly time blocks for project work
- Utilizing Christmas break to compensate for lost time during exam period
- Operating on a bi-weekly sprint basis in semester one and a weekly sprint basis in semester two.

Unexpected setbacks including complex bugs and steep learning curves with unfamiliar technologies, impacted progress. However, perseverance, proactive learning and consistent communication with the project supervisor proved essential for overcoming these hurdles.

Regulatory & Ethical Constraints

Working with AI systems necessitates a strong understanding of ethical responsibilities:

- Ensuring transparency by clearly labelling AI-generated outputs
- Avoiding misuse of AI, particularly in sensitive financial contexts
- Complying with relevant data privacy and usage policies to maintain user trust

Reference [7]: Ethical guidelines followed during AI development

Chapter 3. System Design

3.1 System Architecture Overview

The system is designed as an end-to-end pipeline that begins with a single user action, pressing the 'Analyze' button. Upon user input of the desired stock ticker and time period, the following flow is executed:

1. Cache Check: The backend first checks the local in-memory cache to see if recent data exists for the requested stock. Cached data is stored temporarily for one hour to reduce unnecessary API calls and mitigate rate-limit restrictions.
2. Database Check: If the data is not present in the cache, the system queries the MongoDB database. The data is organized into dedicated collections based on type (e.g. balance sheets, metadata) to maintain clarity and efficiency.
3. API Call (Alpha Vantage): If the required data is missing from both the cache and the database, the system makes a call to the Alpha Vantage API to fetch fresh data. Upon retrieval, the data undergoes cleaning and filtering before being stored in the appropriate MongoDB collection for future use.
4. Data Preparation for Analysis: Once data is retrieved, it is loaded into the stock data array, preparing it for AI-powered analysis.
5. LLM Analysis Pipeline: The prepared data is passed through a two-stage AI analysis process:
    a. Initial Analysis: Processed by Gemini for summarization and initial analysis.
    b. Refinement and Deep-Analysis: The outputs from Gemini and the fetched data are then passed to DeepSeek, which refines and expands the analysis, offering deeper insights, utilizing its human-like cognitive thinking along with carefully crafted prompt engineering techniques.
6. Frontend Delivery: The final analysis results are sent to the React frontend, where they are rendered in a user-friendly, trader-focused dashboard. The interface updates in near-real-time, providing the user with actionable insights efficiently.

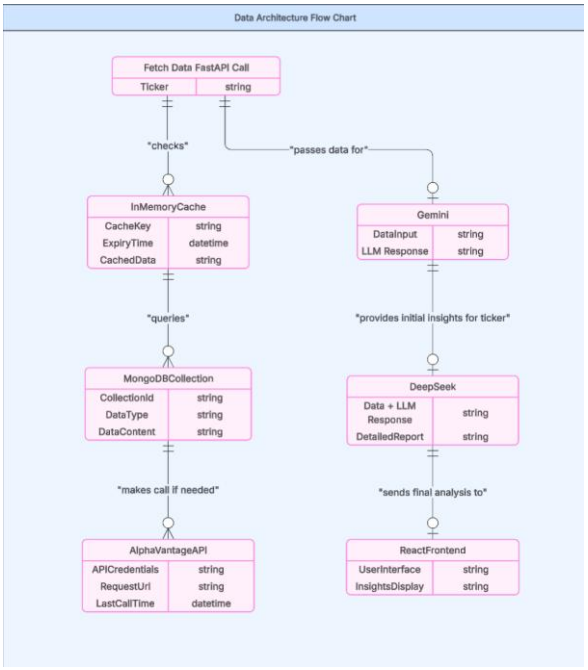


Figure 7: Data Architecture Flow Chart

Gemini and DeepSeek are running locally using Ollama, this feature is easily scalable to boost capabilities by running on the cloud and using more data-intensive, powerful LLMs. I am running the models shown below. Utilizing experience using generative AI and prompt engineering to improve basic outputs into more professional and informative responses. Prompt engineering is 'the process of crafting prompts to get the right output from a model is called prompt engineering'. [9] Using techniques such as giving the LLM a role to adopt, such as 'financial advisor', allocating relevant data, and detailing how you want the answer to be presented are some examples of improving prompt engineering. Prompt Engineering is used to provide context and data for the LLMs. Gemini delivers a response, the response is sent to the deep-thinking model, DeepSeek along with the original data for a final analysis, also using prompt engineering to improve the output responses.

The reasoning for this is to provide an extra layer of information for improved analysis, it resembles multi-agent systems (MAS) where multiple interacting intelligent agents can cooperate, sharing information to achieve a goal that can be too complex for a single agent. My system lacks a back-and-forth interaction between the LLMs, but DeepSeek is tasked with optimizing/refining Gemini's initial analysis, much like a junior and senior financial analyst. Deep-thinking models are used to solve more complex tasks following a process of reinforcement learning, a cognitive human-like thinking process given to LLMs, which enables the models to think deeply on their own before answering the question. [10]. Implementing this component into the project highlights the capabilities of AI currently. Resulting in a processing tool with extreme capabilities that the smartest humans cannot process with this level of detail and accuracy. Once the analysis is begins, TradingView [11] will fetch and display the relevant interactive chart for the stock. Once finished, the user can prompt Gemini a general query about the analysis (input data as reference) or a general question such as 'What does SMA stand for'. The purpose of this, is to enable simple queries to be answered, maintaining user engagement and highlighting AI educational aspect.

```
C:\Users\skeha>ollama list
NAME              ID              SIZE      MODIFIED
gemma3:4b         c0494fe00251    3.3 GB    2 days ago

deepseek-r1:7b    0a8c26691023    4.7 GB    2 weeks ago
```

Figure 8: Ollama LLM Models

3.2 Data Layer Design

The data layer forms the backbone of the application, ensuring efficient. Reliable and rate-limit aware handling of financial data. The goal is to balance performance, storage optimization and accuracy of the analysis pipeline

3.2.1 Data Collection

- API Integration
    - The system integrates with Alpha Vantage API to fetch financial data, including balance sheets, cash flows and metadata. Axios is used as the HTTP client for streamlined requests
- Caching Strategy
    - To optimize performance and mange API rate limits, a local in-memory caching system is implemented. Fetched data is cached for 1 hour, significantly reducing redundant API calls during active calls when possible

3.2.2 Data Cleaning & Filtering

- Filtering Logic
    - Only essential data points required for analysis are retained. Irrelevant metadata and incomplete records are discarded to ensure data quality and avoid noisy input into the AI pipeline
- Why Filtering?
    - Clean data enhances LLM performance, reduces processing time and prevents hallucinations or incorrect outputs from the models. For example, outdated or null values are excluded to maintain integrity of analysis
    - Each financial report is saved in its own collection. This separation simplifies querying and maintains clarity, especially as data volumes grow

3.2.3 Data Storage

- MongoDB Schema Choice:
    - MongoDB is used for its flexibility in handling unstructured and semi-structured data. Data is stored in distinct collections
- Benefits of the Approach
    - Easy scalability for additional data types (real-time stock quotes)
    - Faster access and querying due to clear data segmentation
    - Preparedness for future migrations or integrations with external analytical tools

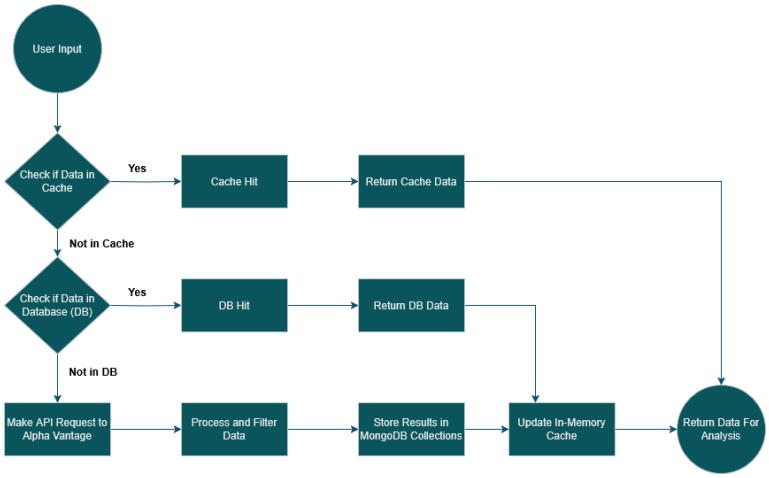


Figure 9: Static Data Function Data Handling Flow Chart

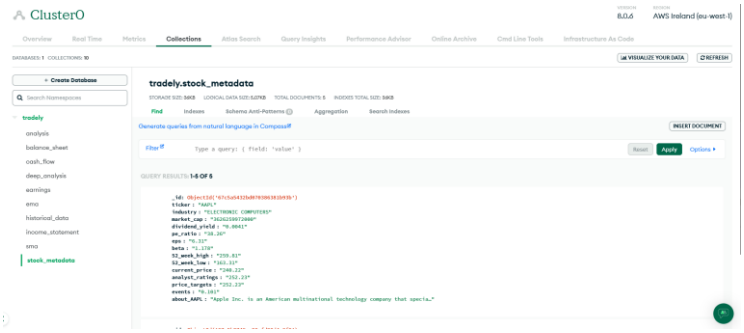


Figure 10: MongoDB Collections

3.3 AI Analysis Layer

The AI analysis layer is the intelligent core of the system, responsible for transforming raw financial data into actionable insights. It leverages a multi-step process that incorporates Retrieval-Augmented Generation (RAG), local LLM models and carefully engineered prompts to ensure accurate and ethical analysis.

3.3.1 Retrieval-Augmented Generation (RAG)

- Data Injection Pipeline
  - Once financial data is cleaned and processed, it is injected into the prompt templates used by the LLMs. The pipeline structures the data for optimal readability by the models, ensuring relevant financial figures (such as total assets, total liabilities, profitability and debt levels) are correctly referenced during generation
- Benefit
  - RAG helps mitigate LLM hallucinations by grounding responses in real data, improving the accuracy of generated insights
  - RAG removes the need for re-training models on up-to-date data
  - Unlike traditional RAG, we are constantly fetching real-time data for analysis where the original method updates with more static data

3.3.2 LLM Interaction

Model Sequence:

The pipeline employs a two-stage LLM process to enhance analysis depth and report quality:

1. Gemini Model: Acts as the first-pass analysis engine. Gemini interprets the cleaned financial data and generates a structured preliminary financial report, ensuring accuracy and coherence from the outset.
2. DeepSeek Model: The output from Gemini, along with the original cleaned data, is fed into DeepSeek for deeper analysis. By leveraging both the raw data and the initial LLM-generated insights, DeepSeek enhances the clarity, depth, and professional standard of the final financial report.

Rationale for this Setup:

Gemini's advanced capabilities deliver robust initial outputs with strong interpretive power, setting a reliable foundation. DeepSeek, known for its human-like reasoning and deep cognitive capabilities, adds a second layer of refinement, producing analysis that feels closer to expert human commentary. This layered approach ensures both precision and narrative quality in the final report.

3.3.3 Prompt Engineering

Prompt Engineering is a critical component of the systems' AI pipeline, it enables LLMs to produce accurate, insightful, structured and contextually appropriate financial analysis.

Adopting OpenAI's Greg Brockman's recommended *'Anatomy of an o1 Prompt'* [12], which emphasizes clear instructions, role definition and structured input/output formats to maximize LLM effectiveness. Using some extra prompt engineering practices such as system role and RAG, we can really empower the LLMs to exceed expectations. After all, the output will only be as good as the prompt engineering behind it.

Prompt Structure

- System Role Definition
    - What I like to do is, at the beginning of each prompt, the model is assigned a role. For example, "You are a financial analyst providing insights for retail investors". This helps set the tone and level of depth expected in the response.
    - Clear role definitions minimize ambiguity and help focus the model on the correct perspective, avoiding generic responses
- Input Data Injection
    - Cleaned and structured financial data is dynamically injected into the prompt to provide context. This ensures the model's analysis is grounded in real, up-to-date data
- O1 Prompt
    - Goal
        - Purpose: State the precise goal of the analysis or output. This focuses the LLM on producing responses that are aligned with the user's expectations

        - "Generate a financial report that includes performance metrics, investment recommendations and risk analysis based on the latest earnings report"
        - Efficiency: A well-defined goal prevents the model from generating unnecessary details or irrelevant information, streamlining the output.
    - Return Format
        - Purpose: Define the structure and format of the output. Ensuring the results is easy to digest and align with professional analysis standards
        - "Present your findings under the following headings in a bulleted list…"
        - Performance: By guiding the format in advance, the prompt avoids unnecessary formatting errors, making it easier to analyze the output and integrate it into decision-making processes
    - Warnings
        - Purpose: In situations where data might be ambiguous or incomplete, adding a cautionary message ensures that the model doesn't make incorrect assumptions
        - "Ensure the report is clearly labelled as AI-generated to abide by the laws regarding AI use in the EU"
        - "If data is missing for any key metrics, explicitly note this and do not attempt to fill in gaps without solid assumptions"
        - Efficiency: Helps the LLM avoid making unsupported inferences, maintaining accuracy and reliability in financial decision-making, and also acts as a debugger to handle missing data
    - Context
        - Purpose: Provide any background context necessary for the analysis. This can include industry trends, historical data, or other relevant factors that the model needs to consider
        - "Based on the current economic downturn, adjust the analysis to factor in market volatility"
        - "Consider the potential impact of Trump's tariffs in the analysis"
        - Performance: Properly contextualized input helps the model adapt its response to dynamic financial conditions, yielding more refined insights.
        - Dynamic Prompting may be added as a future feature (allowing the user to inject current contextual factors)

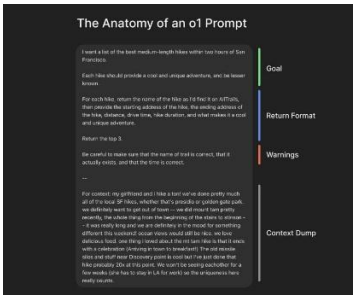


Figure 11: 'Anatomy of an o1 Prompt'

```
def perform_analysis(stock_data: dict) -> str:
    """
    Perform analysis on the stock data.
    """
    sections = [
        "Metadata",
        "Historical Data", "Income Statement", "Balance Sheet", "Cash Flow", "Earnings",
        "SMA", "EMA",
        "News"
    ]
    analysis = """
        goal = Conduct a concise financial analysis of {ticker} based on recent market trends, historical data, and technical indicator

        Return Format:
            - Summary: A brief overview of the stock's current trend.
            - Key Financial Indicators: Price movements, moving averages (SMA/EMA).
            - Market Sentiment: A summary of recent news and sentiment trends.
            - Risk Factors: Highlight any volatility, earnings reports, or macroeconomic risks.

        Warnings:
            - Do not provide direct financial advice.
            - Ensure the response is fact-based and avoids speculation.
            - Keep the response concise, focusing on actionable insights without exceeding 2000 words.

        Context:
            - Stock Data: {stock_data}
            - Technical Indicators: Provide insights based on available technical data.
            - Sentiment Analysis: Summarize the sentiment around recent news articles.
            - Macroeconomic Trends: Highlight broader economic influences impacting this stock.
    """
```

Figure 12: Example: Data Injected Into Prompt

3.4 Frontend-Backend Integration

The frontend and backend are connected through a series of API endpoints, enabling seamless communication and real-time data flow between the components. This architecture enables the frontend to deliver a responsive and data-rich user experience powered by backend data processing and analysis services. This section will detail the API endpoint design, state management strategy, real-time updates and user interaction flow.

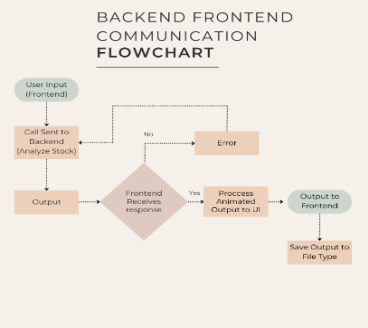


Figure 13: Simple Backend & Frontend Communication Flow

3.4.1 API Endpoints

The backend exposes a RESTful API that serves as the primary communication bridge between the client and the server. These endpoints facilitate data requests for stock information, financial statements, news, market movements (top gainers/losers), and AI-generated financial analysis.

```
url = f"https://www.alphavantage.co/query?function=OVERVIEW&symbol={ticker}&apikey={API_KEY}"
data = await make_request(url)
```

Key Endpoints Table

Endpoint	Description
GET /api/analyze_all/{ticker}	Primary endpoint: Triggers full data analysis pipeline for the given ticker. Aggregates metadata, financials, LLM-generated insights, and visual data into one response.
GET /api/all-stock-data/{ticker}	Practicing encapsulation: calls all relevant endpoints to retrieve a full snapshot of the stock data.

GET /api/metadata/{ticker}	Retrieves cached or live metadata: company profile, market cap, dividend yield, EPS, PE ratio, highs/lows, events, etc.
GET /api/historical/{ticker}	Retrieves stored historical data (weekly) or calls API. Includes highs/lows, volumes, prices, adjusted close, dividends.
GET /api/news/{ticker}	Retrieves cached news or calls API. Cleans and stores in feed array, updates cache. Data includes title, summary, pub Date, URL, thumbnail, sentiment score, sentiment label.
GET /api/live-news-headlines	Frontend endpoint for displaying live news articles. Adds dynamic feel to UI with clickable links and thumbnails.
GET /api/income/{ticker}	Checks cache, then DB, else calls API to fetch income statement. Stored in cache (temp) and DB (static data).
GET /api/balance/{ticker}	Checks cache, then DB, else calls API to fetch income statement. Stored in cache (temp) and DB (static data).
GET /api/cashflow/{ticker}	Checks cache, then DB, else calls API to fetch income statement. Stored in cache (temp) and DB (static data).
GET /api/earnings/{ticker}	Checks cache, then DB, else calls API to fetch income statement. Stored in cache (temp) and DB (static data).
GET /api/sma/{ticker}	Simple Moving Average: checks cache or retrieves from API. No need to store long-term.
GET /api/ema/{ticker}	Exponential Moving Average: checks cache or retrieves from API. Short-term indicator, no long-term storage required.
GET /api/live-market-prices	Frontend endpoint for displaying live market prices of major stocks. Shows price and percentage changes in colour. Compares to price 5 days ago.
GET /api/top-gainers-losers	Live market data endpoint, called in when the frontend is initialized to display a sleek table highlighting the top gainers/losers of that trading day. Uses caching.
POST /api/ask-question	Post request that enables the user to ask the LLM a trading related question in the UI, this enhances UX
GET /api/status	Simple endpoint for checking if connection to MongoDB is active

Table 2: Key Endpoints

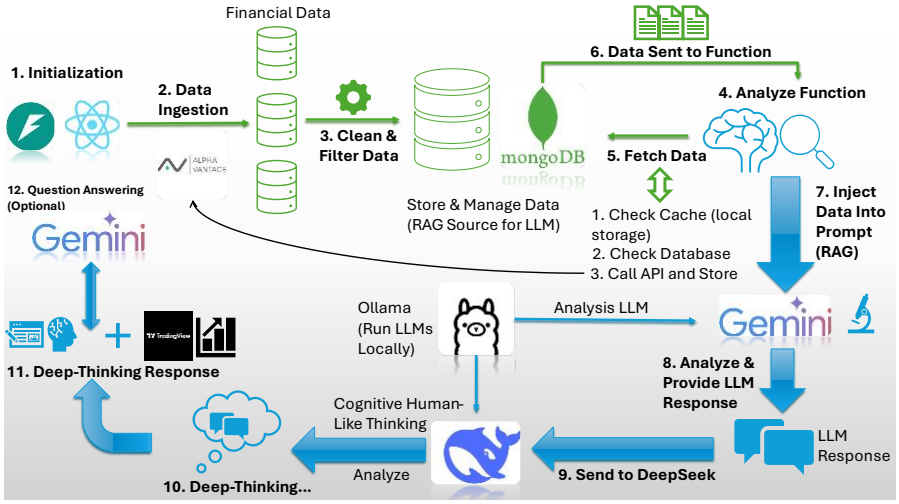


Figure 14: Analyze All Backend Function Architecture

3.4.2 Real-Time Updates and User Interaction Flow

Real-Time Updates

The application integrates real-time data streams to enhance the user experience and ensure up-to-date financial insights. While some data is static (like financial statements – updated annually), other components are refreshed dynamically.

- Live Market Prices (GET /api/live-market-prices)
    - The frontend fetches this endpoint at regular intervals (e.g., every 5–10 seconds). Displays live price movements, price changes over time, and percentage changes. Prices are visually coded (green for gains, red for losses) to improve user recognition of market movements.
- Live News Headlines (GET /api/live-news-headlines)
    - The frontend periodically calls this endpoint to fetch the latest financial news. Users can click to read more, and arrows/buttons allow them to navigate through the latest articles. News sentiment is also embedded (positive/negative/neutral) for quick analysis.
- Frontend Polling Strategy
    - The frontend employs interval-based polling for specific endpoints like live prices and headlines, ensuring timely updates without overwhelming the server.
- Caching Mechanism for Real-Time Data
    - For live data, short-term caching is employed to balance freshness and performance. Cache expiry is set low (e.g., 30–60 seconds) to avoid excessive API calls while still maintaining responsiveness.

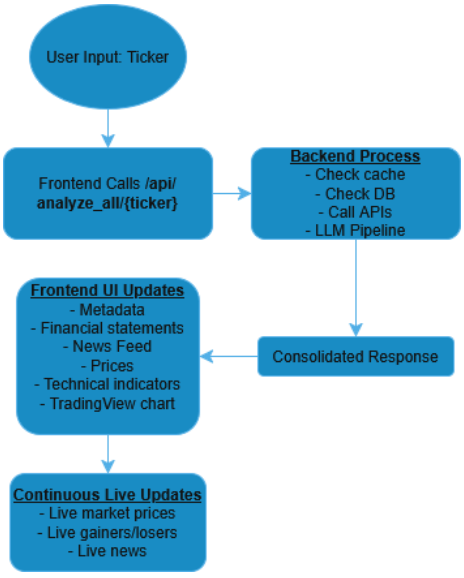


Figure 15: Analyze All Flow Chart (Frontend-Backend)

3.5 Design Justifications

3.5.1 Why MongoDB over PostgreSQL?

- Schema Flexibility for Financial Data
  - Financial datasets are irregular and not nested, companies have varying depths of reports and metadata. MongoDB's document-oriented design perfectly fits JSON structure directly from APIs without complex transformations
- Semi-Structured Data Storage
  - Financial APIs return highly nested JSON and MongoDB stores this natively, eliminating the need to flatten or overly normalize the data
- Scalability and Deeper Agility
  - As the platform grows, MongoDB allows for horizontal scaling, replica sets and fast prototyping, which are critical for iterative development
- Avoiding Complex Joins
  - SQL excels at relational data, but for hierarchical API responses, MongoDB simplifies queries and reduces development time
- Conclusion: MongoDB offers schema flexibility, fast development and seamless scaling for financial analytics use cases.

3.5.2 Why Ollama vs Cloud Based?

- Cost Control
  - Cloud APIs charge per token and per request, Ollama runs models locally, eliminating ongoing API costs
- Low Latency Responses
  - Local inference means no network latency, enabling faster multi-model workflows (Gemini – DeepSeek)
- Data Privacy & Security
  - Keeping sensitive financial data local reduces compliance risks and improves trustworthiness
- Offline Capability
  - Ollama supports full offline operation, which is valuable for demos or environments with restricted connectivity
- Conclusion: Ollama ensures predictable costs, faster responses, and full data control over cloud LLM services

3.5.3 Why Local LLMs Despite Hardware Limitations?

- Customisability and Experimentation
  - Local models allow full control over prompt design, temperature and fine-tuning, offering flexibility not always available with managed services
- Independent of External APIs
  - No reliance on third-party uptime or API limitations, ensuring greater system robustness
- Performance with Optimized Models
  - Despite hardware limitations, quantised models (4-bit) allow efficient inference even on consumer-grade GPUs or CPUs
- Security & Data Privacy

- Sensitive financial data never leaves the local environment, aligning with best practices for data protection
- Conclusion: Local LLMs provide experimentation freedom, security and sufficient performance when models are optimized

3.5.4 Why RAG?

- Contextual Accuracy
  - RAG pipelines enhance LLM outputs by grounding answers in real, up-to-date financial data, improving factual accuracy
- Scalability
  - RAG allows handling vast amounts of company data without requiring retraining the model itself
- Reduced Hallucination Risk
  - By injecting retrieved data directly into prompts, the model relies less on internal assumptions and more on verified facts
- Dynamic Knowledge Base
  - The database can be updated independently of the model, allowing real-time improvements to responses without additional training
- Conclusion: The RAG implemented bridges the gap between static model knowledge and static external sources through the retrieval of dynamic real-world financial data.

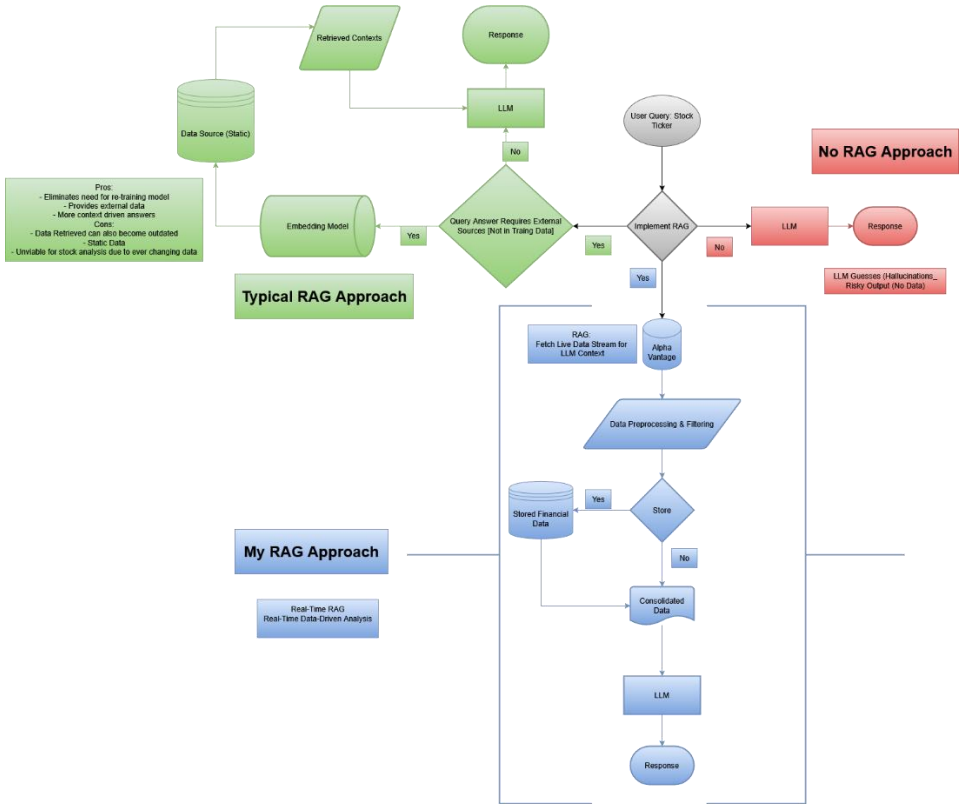


Figure 16: No RAG vs Typical RAG vs Tradeskee RAG

Feature	No RAG (LLM Only)	Typical RAG (Static Data)	Your RAG (Live Data + Alpha Vantage)
Data Source	None (LLM relies on pretrained knowledge)	Fixed dataset (e.g., articles, old financials)	Real-time API (Alpha Vantage) + live feeds
Accuracy	High risk of hallucinations	Outdated/mismatched data	Real-time, market-aligned
Retrieval Mechanism	None	Static database lookup	Dynamic fetch → Preprocess→ Vector DB
Response Quality	Generic, unreliable for stocks	Limited by dataset freshness	Precise, data-driven insights
Implementation Complexity	Low (just LLM)	Medium (static DB + embeddings)	High (API + preprocessing + retrieval)
Best For	General chat (no real-time needs)	Historical analysis (fixed timeframe)	Live trading, trend analysis, forecasts
Tools Needed	LLM (e.g., GPT-4)	LLM + Vector DB (Quadrant)	LLM + Alpha Vantage API + Vector DB

Table 3: No RAG vs Typical RAG vs Tradeskee RAG

3.5.5 *Why a Multi-Agent System?*

- Specialization of Roles
  - By assigning specific tasks to different LLMs (Gemini for structured summarisation, DeepSeek for deeper insights), we leverage the strengths of each model in a complementary manner
- Enhanced Output Quality
  - The first agent (Gemini) provides a clean, structured base analysis, while the second agent (DeepSeek) refines this with humanlike critical thinking and improved narrative flow
- Exploring Model Strengths
  - Different models excel in different tasks: Gemini's structured output complements DeepSeek's creative, human-like reasoning
- Future Expandability
  - The architecture allows easy expansion, e.g., introducing sentiment agents, risk analysis agents, or regulatory compliance agents
- Conclusion: The multi-agent design maximizes model strengths, delivering richer, higher-quality financial analysis

3.6 Alignment With Requirements

The design choices reflect a balance between performance, user experience and transparency. Each component, from the data retrieval to the LLM analysis and frontend integration was selected to meet the identified project requirements while ensuring flexibility for future enhancements.

Real-Time Updates	Live API integrations for market prices and news headlines: endpoints /api/live-market-prices | live-news-headlines | top-gainers-losers
LLM-Generated Insights	Local LLMs (DeepSeek, Gemini via Ollama) generate in-depth stock analysis and respond to user queries dynamically.
Multi-Source Data Integration	System pulls data from Alpha Vantage, news APIs, and cached financial databases, consolidated via the /api/analyze_all/{ticker} endpoint.
User-Friendly Interface	Clean frontend design with chat bubble summaries, real-time notifications, interactive charts, and file export options for ease of use.
Exportable Reports	Analysis results can be exported to various formats (PDF, TXT) directly from the frontend, allowing users to save and share insights.
Explainable AI (XAI)	Transparent data presentation with clear breakdown of analysis steps and multi-model outputs for user trust.
Performance & Cost Efficiency	A local-first approach using Ollama for LLM hosting avoids cloud inference costs; caching and lightweight API calls optimize system responsiveness.
Scalability & Flexibility	Modular backend design with clear RESTful endpoints allows easy future expansion (e.g., adding more indicators, new LLMs, or external APIs).
Enhanced User Engagement	Integrated Q&A feature with Gemini assistant to answer follow-up stock questions, increasing platform interactivity and session duration.
Data Transparency & Traceability	Financial data endpoints include metadata, historical trends, and sentiment scores, allowing users to cross-reference AI outputs.

Table 4: Requirements and Design Implementation

3.7 Key Features and User Interaction

3.7.1 User Interaction and Response Handling

Users have full control over the stock analysis process by simply entering a stock ticker symbol (AAPL, AMZN). Upon initiating the analysis, they receive real-time notifications as DeepSeek begins the output analysis. Once completed, the response is streamlined in an iterative process to the screen within a chat-bubble format, complemented by interactive charts for visual analysis. Users can also save the analysis to multiple file formats (PDF, TXT, CSV, JSON), promoting flexibility and data portability.

3.7.2 Question and Answering (Q&A)

This feature enhances user engagement by enabling natural language questions about stocks directly within the platform. Powered by Gemini LLM hosted on Ollama and acting as a ‘stock market enthusiast’, the assistant allows users to explore further insights, validate findings, or ask follow-up questions based on prior analysis. Cross—referencing LLM-

generated answers with real-time financial data builds user trust and increases session duration.

3.7.3 Explainable AI (XAI) & Multi-Agent Approach

Inspired by concepts from Explainable AI (XAI), this system ensures transparency in decision-making by clearly displaying how outputs are generated, increasing user trust. While not a traditional multi-agent system, the layered design, where different LLMs handle specialized tasks (e.g. DeepSeek for deep-thinking analysis), mimics a subset of a Multi-Agent System (MAS). The LLMs provide a functional output from the interaction between two LLM models, this opens the possibilities of exploring this concept further.

Chapter 4. Implementation

4.1 Introduction

In this section, I will showcase the functionality of the web application, how the backend and frontend communicate and the analysis pipeline.

4.2 Development Environment

The development environment for this project was selected based on experience, performance, flexibility and ease of integration with modern AI tools and data sources. Since the project required running local LLMs, real-time data retrieval and dynamic frontend updates both hardware and software choices were important.

Hardware Environment

- Laptop Specs:
  - Processor: AMD Ryzen Pro 5
  - Memory: 16GB RAM
  - Storage: 1TB SSD
  - OS: Windows 11

Running Local LLMs such as DeepSeek and Gemini via Ollama requires RAM availability. These models were suitable for my model specs and project requirements. While not as powerful as using cloud GPU servers, this architecture enabled faster iteration and offline testing. These can easily be upgraded depending on hardware specs or by swapping to cloud GPU.

Tool/Technology	Purpose	Description
Python	Backend Development	FastAPI server, API Integration, LLM Handling
FastAPI	Web Framework	Lightweight, Async Support, Easy Documentation
MongoDB	NoSQL Database	Flexible Schema, Fast Caching for API Responses
Ollama	Local LLM Deployment Tool	Allows Swapping and Running Other LLM Models
Gemini	Powerful LLM	Initial AI Analysis
DeepSeek	Deep-Thinking LLM	Primary Model for Analysis Generation
VSCode	IDE	Primary Integrated Development Environment
Postman	API Testing	Test Endpoints During Development
React.js	Frontend Development	Building Interactive User Interface
Axios	API Calls from Frontend	Handles HTTP Requests to Backend Endpoints
TradingView Lightweight Charts	Interactive Chart Visualization	Renders Interactive Financial Charts
Git & GitHub	Version Control	Project Versioning (feature branching, commits)

Table 5: Software Stack

4.2.1 Environment Management

- Python Virtual Environment: Used to manage dependencies and isolate project packages
- .env File: Stores API keys (e.g. Alpha Vantage) and sensitive configurations (Mongo URI)
- Pre-commit Hooks: Basic linting and formatting extensions

4.2.2 API Keys & External Services

The project integrates with multiple external APIs, requiring secure handling of credentials.

- Alpha Vantage: Fundamental, technical and economic stock data
- Ollama Models: Gemini and DeepSeek for combined detailed analysis

4.2.3 Development Workflow

- Backend was mainly developed and tested before frontend, proceeding to become a simultaneous development process, using VSCode split terminals for running both servers concurrently
- API endpoints were incrementally tested using Pytest and Postman
- Frontend used hot reload for rapid UI iterations
- Git branching strategy followed feature-based development for clarity

4.2.4 Development Approach

- Agile Iteration: Features were developed incrementally, starting from core API integrations to advanced frontend visualizations and local LLM processing.
- Test-Driven Development (TDD): API endpoints were tested using Postman and automated tests before frontend integration.
- Local-first Development: Focused on local execution to ensure independence from cloud resources and control over data flow.
- Modular Architecture: Backend services are broken down by responsibility (e.g., /api/metadata/, /api/analyze_all/), which improves maintainability and debugging.

4.2.5 Challenges in Setup

- Initial performance limitations when running multiple local models, mitigated by downsizing LLM model from high memory Mistral to Gemini

4.3 Implementation Approach

The development of this system followed an agile approach, setting clear sprint goals achievable within the weekly or bi-weekly period. this must incorporate testing, reviewing pull my pull request and documentation. Allowing flexibility propelled rapid prototyping and incremental improvements throughout the development lifecycle.

Step 1: Environment Setup

- Local Development Environment
    - Set up git repository and clone to local environment
    - Create a requirements file for libraries and dependencies (iteratively adding more) – install with pip install -r requirements.txt
    - Set up Python backend with FastAPI
    - Create .env file for Alpha Vantage and start initial development
- Version Control
    - Initialised project to maintain a full version using Git and GitHub
    - Regular commits ensured rollback
    - Branching supports isolation and testing while maintaining a working branch
    - Conduct pull request reviews independently

Step 2: Backend API Development

- Core API Design:
    - Designed RESTful API endpoints to handle different responsibilities

    - Followed clean architecture principles, following stock models, routes and services architecture flow
- Data Aggregation Logic
    - Developed data pipelines to separate the stock data endpoints into dedicated folders depending on the type of data (e.g. financial indicators)
    - Implemented caching mechanisms for efficiency, reducing reliance on Live API calls
- Live Market Endpoints
    - Designing endpoints for frontend integration to improve UI/UX
    - Following the same design patterns as the ticker-specific endpoints
- Testing
    - Pytest for testing individual components
    - Using mocking in some areas to avoid the need for extensive API calls (limited calls)

Step 3: LLM Integration & Multi-Model Architecture

- Local LLM Setup: Integrated Ollama to run multiple models locally (Gemini & DeepSeek)
- LLM flows:
    - Gemini: Initial stock data analysis and Q&A assistant
    - DeepSeek: In-depth deep-thinking analysis generation
- Prompt Engineering
    - Designed structured prompts for clarity and optimal model output
    - Implemented structured prompts following best techniques for optimal AI outputs
    - Implemented RAG flow to supply a data stream of live data to the LLMs. Resulting in an immeasurable fine-tuned output
- Explainability Layer
    - Ensured outputs included transparent reasoning by showing data references alongside LLM responses

Step 4: Frontend Development

- Frontend Framework
    - Built using React.js for dynamic, responsive UI
    - Styles components for clean, user-friendly design
- API Integration
    - Connected frontend to backend endpoints, enabling real-time data retrieval, visualization and displaying user requests
- Visualization
    - Integrated interactive TradingView Lightweight Charts to display specified stock
    - Enabled stock comparison, indicator visualization and much more

Step 5: User Interaction Flow and UX Enhancements

- Interactive Chat Bubble
    - Designed chat-style interaction for LLM outputs, improving readability
- Real-Time Notifications
    - Implemented frontend alerts when the analysis request is sent and completed
- Downloadable Reports
    - Enabled users to export analysis in multiple formats (PDF, CSV, TXT, JSON)

Step 6: Testing and Refinement

- Backend testing
    - Used Postman API and unit tests to validate and debug API responses
    - Mocking data to test the inner workings of functions
    - Logging, break-points, debugging and debug print statements to solve bugs
- Frontend Testing
    - Mocking live data to remove API limitations and enable continuous UI responsiveness and data rendering
    - Adding debug statements to pinpoint issues between frontend/backend communication
- User Feedback Iteration
    - Collected feedback from peers and supervisors to refine UX and improve performance

```python
@pytest.mark.asyncio
async def test_send_prompt_to_llm(self):
    prompt = """
    Analyze the following stock data:

    Metadata: {
        "ticker": "AAPL",
        "industry": "ELECTRONIC COMPUTERS",
        "market_cap": "3278873821000",
        "dividend_yield": "0.0046",
        "pe_ratio": "34.59",
        "eps": "6.31",
        "beta": "1.178",
        "52_week_high": "259.81",
        "52_week_low": "163.31",
        "current_price": "231.92",
        "analyst_ratings": "252.59"
    }
    """
    response = await send_prompt_to_llm(prompt)
    assert response is not None, "LLM response should not be None"
    print("LLM response:", response)
```

Figure 17: Testing LLM Communication

4.4 Backend API Development

The backend development comprises the following responsibilities: data retrieval, data cleaning and filtering, data storage and LLM analysis process. It also enables Q&A and live data for UI design.

4.4.1 Data Collection

To collect the data from Alpha Vantage API, we first had to get our API key and store it privately in the .env file.

The structure:

- Models: Defines structure of the function output

- Routes: Stores endpoints to fetch data
- Services: Contains the functions that execute API calls

Each endpoint has a unique URL, including the symbol parameter for stock-specific requests. Often, the functions include a limiting factor to ensure we don't return too much data.

```
url = f"https://www.alphavantage.co/query?function=OVERVIEW&symbol={ticker}&apikey={API_KEY}"
data = await make_request(url)
```

Asynchronous functions (async) enable the application to handle multiple operations, essential for I/O bound tasks (database queries, API calls).

The make request helper function is an asynchronous function that sends the HTTP GET request to the Alpha Vantage API. It is used inside the functions that call the API.

```
# Helper function for API requests with retries
async def make_request(url: str, retries: int = 3, backoff_factor: float = 0.5):
    """Handles API requests and rate limit errors with retries."""
    for attempt in range(retries):
        response = requests.get(url)
        if response.status_code == 429:
            if attempt < retries - 1:
                await asyncio.sleep(backoff_factor * (2 ** attempt))
                continue
            raise HTTPException(status_code=429, detail="Rate limit exceeded. Please try again later.")
        if response.status_code != 200:
            raise HTTPException(status_code=response.status_code, detail="Failed to fetch data from Alpha Vantage.")
        return response.json()
    raise HTTPException(status_code=500, detail="Failed to fetch data after multiple attempts.")
```

Figure 18: Make Request Code

4.4.2 Data Cleaning and Filtering

Cleaning and filtering the data is a crucial step to ensure that we only return relevant data. Alpha Vantage returns a wide range of raw data that can add unnecessary noise. Efficient extraction is a key step in LLM analysis. Setting strict limiting parameters on certain functions to reduce data is another good practice.

```
cleaned_news = [
    {
        "article": index + 1,
        "title": item.get("title", "N/A"),
        "summary": item.get("summary", "N/A"),
        "pubDate": item.get("time_published", "N/A"),
        "url": item.get("url", "N/A"),
        "thumbnail": item.get("banner_image", "N/A"),
        "sentimentScore": item.get("overall_sentiment_score", "N/A"),
        "sentimentLabel": item.get("overall_sentiment_label", "N/A"),
    }
    for index, item in enumerate(news_data[:limit])
]
```

Figure 19: Cleaning News Endpoint

4.4.3 Data Storage and Caching

MongoDB stores the data retrieved from the API in a dedicated collection. The functions check the database for the data before calling the API, this helps save computational costs by generating fewer calls to the API. To set this up we first have to get the API key by making a MongoDB account, setting up the database and initializing the collections. The Database Manager class handles all read-write operations.

Caching is a temporary local storage technique used to prevent excessive API calls and also reduce computational costs.

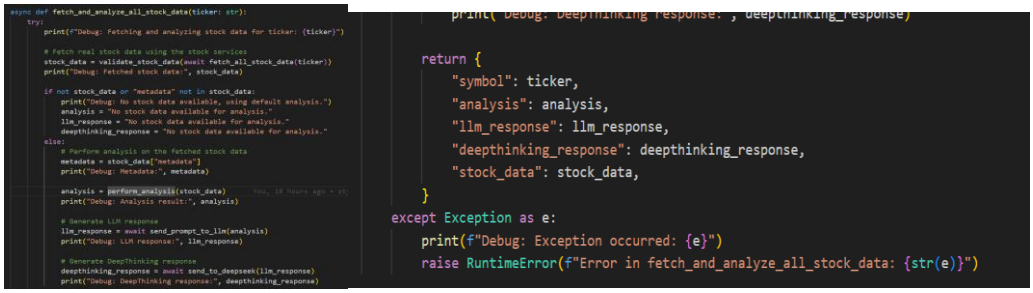
```python
# Fetch Income Statement
async def fetch_income_statement(ticker: str, limit: int = 1):
    cache_key = f"{ticker}_{limit}"
    if cache_key in income_statement_cache:
        return income_statement_cache[cache_key]
```

```python
# Caches with a TTL of 1 hour and a max size of 100 items
metadata_cache = TTLCache(maxsize=100, ttl=3600)
historical_data_cache = TTLCache(maxsize=100, ttl=3600)
news_cache = TTLCache(maxsize=100, ttl=3600)
income_statement_cache = TTLCache(maxsize=100, ttl=3600)
balance_sheet_cache = TTLCache(maxsize=100, ttl=3600)
cash_flow_cache = TTLCache(maxsize=100, ttl=3600)
earnings_cache = TTLCache(maxsize=100, ttl=3600)
sma_cache = TTLCache(maxsize=100, ttl=3600)
ema_cache = TTLCache(maxsize=100, ttl=3600)
top_gainers_losers_cache = TTLCache(maxsize=100, ttl=3600)
market_data_cache = {"data": {}, "last_updated": 0}
```

Figure 20: Caching Implementation

4.4.4 Data Processing and Function Execution

The main function for the backend combines all the individual functions and encapsulates them into one function. When the analysis button is pressed, it triggers the pipeline displayed in the diagram from the previous section. The data is fetched and passed to the LLM. The perform analysis function enables Gemini to take in the data and be prompted to analyze the stock before we call the send prompt to DeepSeek and then output the deep-thinking response.



```python
        return {
            "symbol": ticker,
            "analysis": analysis,
            "llm_response": llm_response,
            "deepthinking_response": deepthinking_response,
            "stock_data": stock_data,
        }
    except Exception as e:
        print(f"Debug: Exception occurred: {e}")
        raise RuntimeError(f"Error in fetch_and_analyze_all_stock_data: {str(e)}")
```

Figure 21: Analyze All Function

4.5 Backend LLM Integration & Multi-Model Design

The multi-model architecture empowers the platform to fully leverage collected financial data to provide AI-driven analysis. This is the selling point of the application.

4.5.1 LLM Role

The LLM receives pre-fetched and validated data from the backend. The integration focuses on seamless data flow into the LLM prompts for contextual data, providing data without gaps or noise, which will increase the efficiency of the LLM analysis. The prompts provided to the LLMs provide clear instructions for a successful output response.

To implement these models, we have to pull the models from Ollama to our local machine, where they will run in the background on a local Ollama server. In the code, we specify the model for each task (e.g. initial analysis, deep thinking, Q&A). Ollama is secure, cost-effective and fast, this is great for this project.

4.5.2 Prompt Generation

The prompts are carefully constructed with dedicated analysis functions, unique prompts are created for each process. Clear instructions and context are embedded within each prompt.

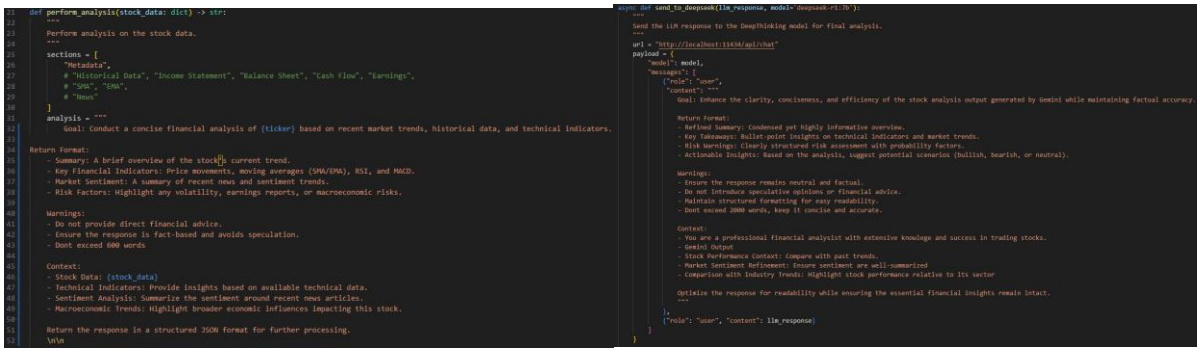


Figure 22: Prompt engineering for Gemini and DeepSeek response

4.5.3 LLM Integration and Asynchronous Communication

The send_prompt_to_llm() function sends a formatted prompt to Gemini, as specified in the model within the parameters, asynchronously using the httpx library. The use of asynchronous functions ensures that long-running I/O operations, such as API calls, don't block the main application thread. This is essential for handling real-time or near real-time requests such as the live data feeds.

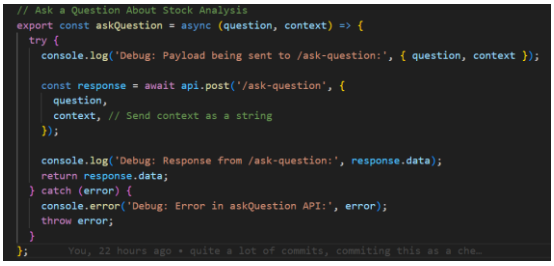
4.5.4 Model Flexibility

In this project, we use Gemini and DeepSeek via Ollama. The LLMs can be easily scaled in power and capabilities through Ollama, but it would also be relatively simple to implement paid premium services (e.g. Open AI GPT-4, Claude, etc)

4.5.4 Q&A with LLM

The Q&A feature is an essential addition that enhances user engagement by allowing users to ask stock-related questions without leaving the platform. The primary goal is to enable user retention and increase session duration by providing an intuitive assistant feature.

To achieve this, I integrated a Gemini LLM via Ollama, assigning it the role of a stock market enthusiast. This allows the model to act as an external assistant, capable of filling knowledge gaps or answering follow-up questions based on the generated stock analysis. Users can also cross-reference the LLM's response with the retrieved financial data, ensuring an interactive and educational experience.



```
// Ask a Question About Stock Analysis
export const askQuestion = async (question, context) => {
  try {
    console.log('Debug: Payload being sent to /ask-question:', { question, context });

    const response = await api.post('/ask-question', {
      question,
      context, // Send context as a string
    });

    console.log('Debug: Response from /ask-question:', response.data);
    return response.data;
  } catch (error) {
    console.error('Debug: Error in askQuestion API:', error);
    throw error;
  }
};      You, 22 hours ago • quite a lot of commits, commiting this as a che…
```

Figure 23: Q&A Frontend & Backend Implementation

4.6 Challenges and Solutions

Throughout the development of this project, several technical and design challenges emerged. These ranged from hardware limitations with local LLM deployment to synchronization complexities between multi-agent flows. Proactively identifying these issues early in the development cycle allowed me to dedicate time to solving issues and pushing out expected

features to following sprints. I will only accept a pull request if the testing is completed as part of the feature.

The table below summarizes the key challenges encountered and the solutions that were implemented to address them.

Challenge	Description	Solution Implemented
Running Local LLMs (Hardware Limitations)	Running LLMs like DeepSeek and Gemini locally was resource-intensive and occasionally caused performance drops.	Optimised Ollama setup for better resource management. Prioritised smaller, specialised models and leveraged caching of responses to reduce repeated inference load.
API Rate Limits and Data Freshness	External APIs like Alpha Vantage had strict rate limits, risking incomplete data during high request volumes.	Implemented caching and database storage to reduce redundant API calls. Prioritised critical data and queued non-urgent requests.
Synchronizing Multi-Agent LLM Outputs	Coordinating responses between analysis LLMs and assistant LLM (Gemini) without overlapping or confusing outputs.	Designed a controlled flow where Gemini & DeepSeek analysis processes first, followed by Gemini Q&A. Used clear state management and endpoint sequencing.
Frontend Responsiveness with Real-Time Data	Handling multiple real-time feeds (news, prices) slowed down the frontend and affected UX.	Introduced loading states and asynchronous data fetching.
Data Consistency Between Cache and Live Data	Differences between cached data and live updates led to inconsistencies in analysis outputs.	Implemented cache invalidation strategies and clear timestamping for data freshness. Indicated data sources and timestamps to users.

Explaining Model Outputs (XAI Challenge)	LLM responses initially lacked transparency, risking user trust.	Added structured prompts to ensure the model explains reasoning. Integrated references to financial indicators and data sources in the response.
Integration Complexity Between Frontend and Backend	Managing many API endpoints and ensuring seamless data flow to frontend components.	Used centralised API service in the frontend (React) and consolidated backend response structures for ease of integration.
Time Constraints and Scope Management	Large feature set with limited development time risked unfinished components.	Prioritised core features (Analyze All, Q&A) and postponed optional extras for future development. Focused on system stability.

Table 6: Challenges & Solutions

4.7 Summary

This chapter detailed the practical implementation of the stock analysis system, from initial setup to multi-agent LLM integration and real-time data streaming. Each component was carefully selected and integrated to ensure alignment with the original design objectives.

The development process also revealed several challenges, which were successfully addressed through thoughtful design choices, optimisation techniques and clear separation of responsibilities between components.

Overall, the implementation demonstrates a functional and scalable architecture that effectively meets the project's technical goals and user requirements. This foundation provides room for future enhancements, including model fine-tuning, user personalisation and advanced data visualization.

Chapter 5. UI, Testing and Evaluation

5.1 Introduction

In this section, I will present the UI design, testing methodologies and evaluation of the system's outputs. The goal is to demonstrate the application's usability, stability and performance through systematic testing and visual outputs.

5.2 UI Design

Below is a showcase of the main page of the application. It is an open-page intuitive design with everything you need and nothing you don't. Enables user requests for analyzing stocks with interactive charts for comparison and a Q&A feature for filling knowledge gaps. Live

market data pulled from backend Alpha Vantage endpoints. The ace logo symbolizes high quality and excellence, creating a good addition to the service for marketability.

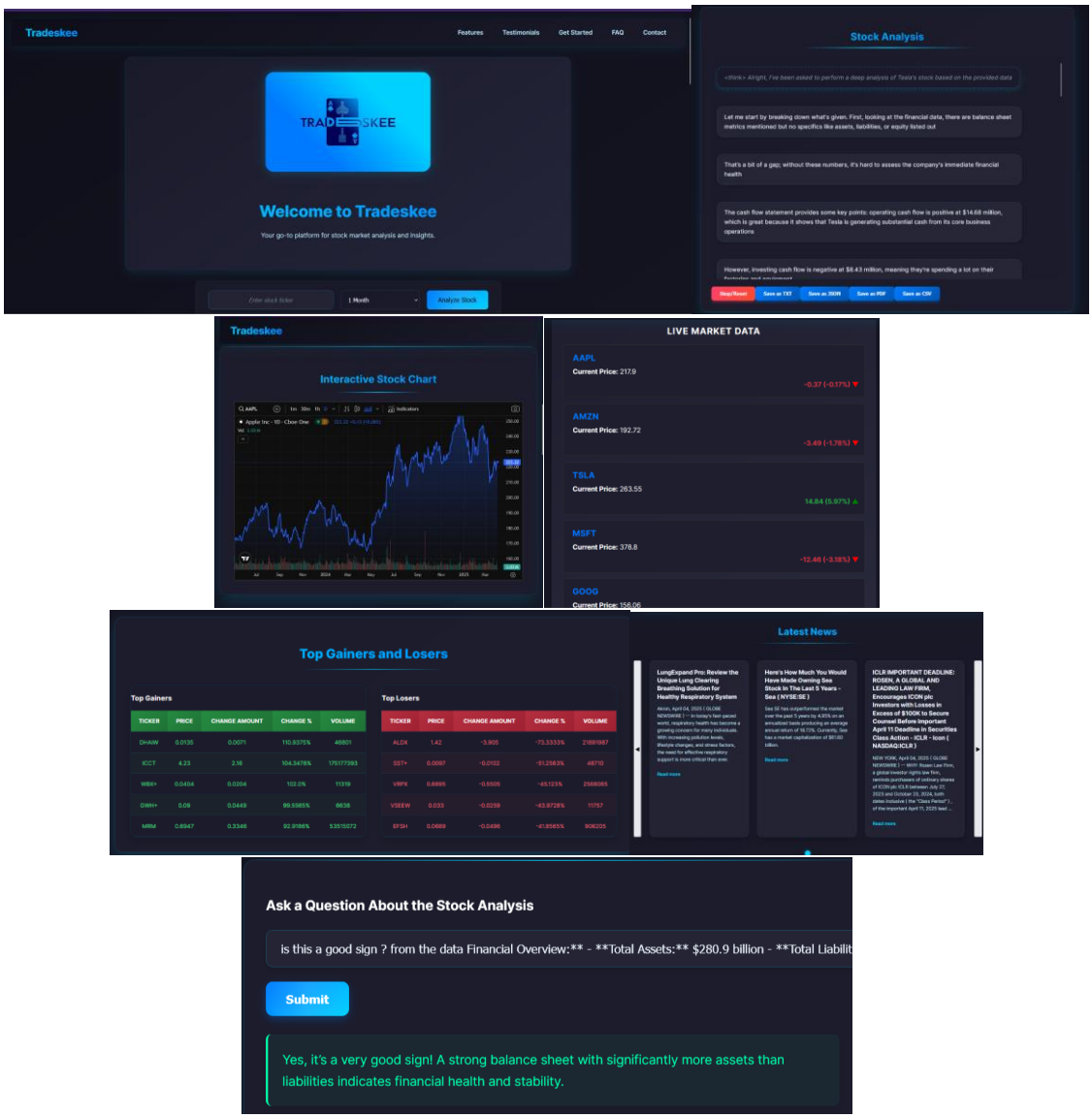


Figure 24: Tradeskee Home

5.3 Testing and Debugging

Thorough testing and debugging were essential in ensuring the system's reliability, efficiency, and accuracy. The testing strategy incorporated both unit testing and integration testing across different system components, including data collection, processing, and LLM functionality.

Mocking is a critical testing technique that simulates API responses, allowing developers to verify function behaviour without making actual API calls. This approach was particularly useful in testing the analysis function, ensuring it processed data correctly under various conditions while minimizing API usage and latency.

Below is an example of mocking the analysis function to simulate Alpha Vantage data retrieval without making real API requests:

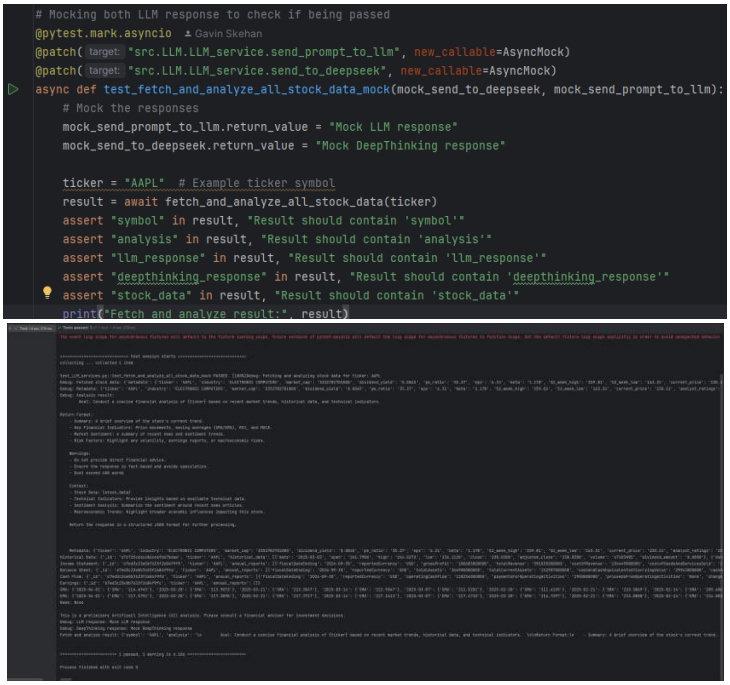


Figure 25: Mocking Analyze Function Test

5.3.1 Testing Strategies

- Unit Testing: Individual functions were tested to ensure they produced the expected outputs. This was especially important for data collection and LLM processing functions
- Database Testing: The MongoDB connection was tested to verify data retrieval, storage and caching mechanisms
- Frontend Debugging: Debugging tools such as print statements, breakpoints and browser developer tools were used to trace UI issues
- API Testing: Postman was used to validate API endpoints, ensuring correct data retrieval and response formatting
- Asynchronous Testing: Pytest with Pytest mark asyncio was used to test asynchronous functions, ensuring smooth execution without blocking operations
- Prompt Testing: Extensively running the analysis with variations of the prompt messaging to find the best outputs

By systematically applying these testing methodologies, I ensured that potential issues were identified and resolved early, leading to a more stable and efficient system.

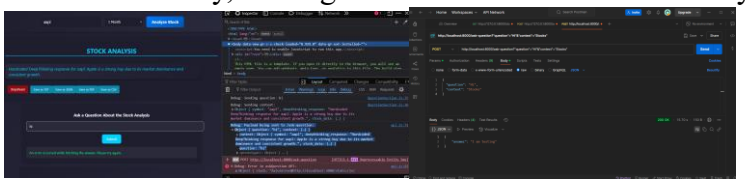


Figure 26: Testing and Debugging Q&A

Figures : Mocking Results to Test Frontend UI

Figures : Pytest without Mocking

5.4 Sample Output & Evaluation (DeepSeek AI Analysis) [2]

Test 1 – Tesla [Summary Results]

1. Financial Health Overview
Cash Flow Statement:
Positive operating cash flow of $14.68 million: Indicates strong generation from core business operations.
Negative investing cash flow of -$8.43 million: Suggests significant expenditures on growth initiatives, such as factories and equipment.
Slightly negative financing cash flow of -$1.34 million: This may indicate that Tesla is using debt or equity to fund these investments.
Lack of Financial Metrics:
Missing key metrics like **assets, liabilities, and equity**: This makes it difficult to comprehensively assess the company's financial leverage and overall health comprehensively.

---

2. Technical Indicators
Moving Averages:
Both **SMA** (Simple Moving Average) and **EMA** (Exponential Moving Average) are trending upwards, suggesting an uptrend in stock price.
The positive trend confirmed by rising averages is a good sign for Tesla's short-term performance.

---

3. News Sentiment Analysis
Battery Swap with Nio:
Sentiment: Neutral.
Impact: Potential collaboration enhancing Tesla's market position, but without significant directional impact on stock.
Musk's Stock Sales:

Sentiment: Bearish.
z
Impact: Musk's stock sales may signal issues with investor confidence, potentially affecting Tesla's valuation and future prospects.
Elon Musk's Twitter Affiliation:
Sentiment: Bearish.
Impact: Musk's negative association with Twitter could damage his public image and, in turn, Tesla's reputation and investor sentiment.

---

4. Overall Assessment
Positive Factors:
Strong **operating cash flow** and positive technical indicators (**SMA** and **EMA** trends).
Negative Factors:
Mixed **news sentiment**: Neutral on the battery swap and negative on Musk's stock sales and Twitter affiliation.
Gap in Data:
Missing **balance sheet details** prevent a full financial health assessment.

Key Takeaways
Price Action: Positive trend indicated by rising **SMA**/ **EMA**, suggesting potential upside. However, caution is needed due to mixed investor sentiment.
Valuation Concerns: Negative news surrounding stock sales and Musk's activities may weigh on Tesla's valuation despite positive cash flow.
Market Position: Tesla has strategic advantages, such as the potential from collaborations like the **battery swap**, but these are tempered by **Musk-related risks**.

Investment Considerations
Comparative Analysis:
A comparison of Tesla's performance with competitors (e.g., **Ford**, **Waymo**) would provide a better context for its market position and competitive advantage.
Granular Financial Data:
Accessing full financial statements is essential for a more comprehensive analysis beyond the high-level metrics provided.

Conclusion
While Tesla demonstrates positive signs of growth through strong cash flow and positive technical indicators, the mixed sentiment in the news introduces risks. Investors should conduct further due diligence and compare Tesla with its competitors to make an informed investment decision.

Figure 27: Summary output: Tesla DeepSeek RAG Output

Test 2 – BYDDF – Chinese EV Company [Summary Results]

1. Company Overview
BYD Group: BYDDY represents a Chinese electric vehicle (EV) manufacturer based in Shenzhen.
Market Position: A leading player in the EV industry, particularly dominant in China, with a growing global presence through partnerships like BYD Group HOLDCo.

---

2. Market Position
Global EV Market Growth: The global EV market has seen a 31% year-on-year growth, which serves as a significant tailwind for BYD.
China's Dominance: Despite strong global growth, declines in China's passenger car sales by 20% may impact BYD's primary revenue stream, as China is the largest EV market.

---

3. Financial Performance
Revenue Growth: Specific growth figures or percentage increases could be added to highlight BYD's financial health.
Profitability: Include net income and profit margin data to assess profitability trends.
Debt Levels: Debt metrics, if available, should be mentioned. High debt levels could pose a risk.

---

4. Key Risks
China Market Headwinds: It's important to investigate the reasons behind the decline in sales, such as economic slowdowns or increased competition from domestic EV players.
Competition: BYD faces competition from global players like **Tesla** and **Volkswagen**, as well as other emerging domestic EV manufacturers.
Liquidity Constraints: Being an OTC stock, BYDDY faces liquidity risks compared to stocks listed on exchanges like NASDAQ. This could impact trading and investor confidence.

---

5. Actionable Insights
Financial Health Review: Conduct a detailed analysis of recent financial reports to assess revenue growth, profit margins, and debt levels.
Market Monitoring: Stay informed on industry developments, such as M&A activities, partnerships, and strategic investments in EV technology.
Liquidity Assessment: Take into account the volatility and liquidity challenges associated with trading in the OTC market and its impact on investment stability.

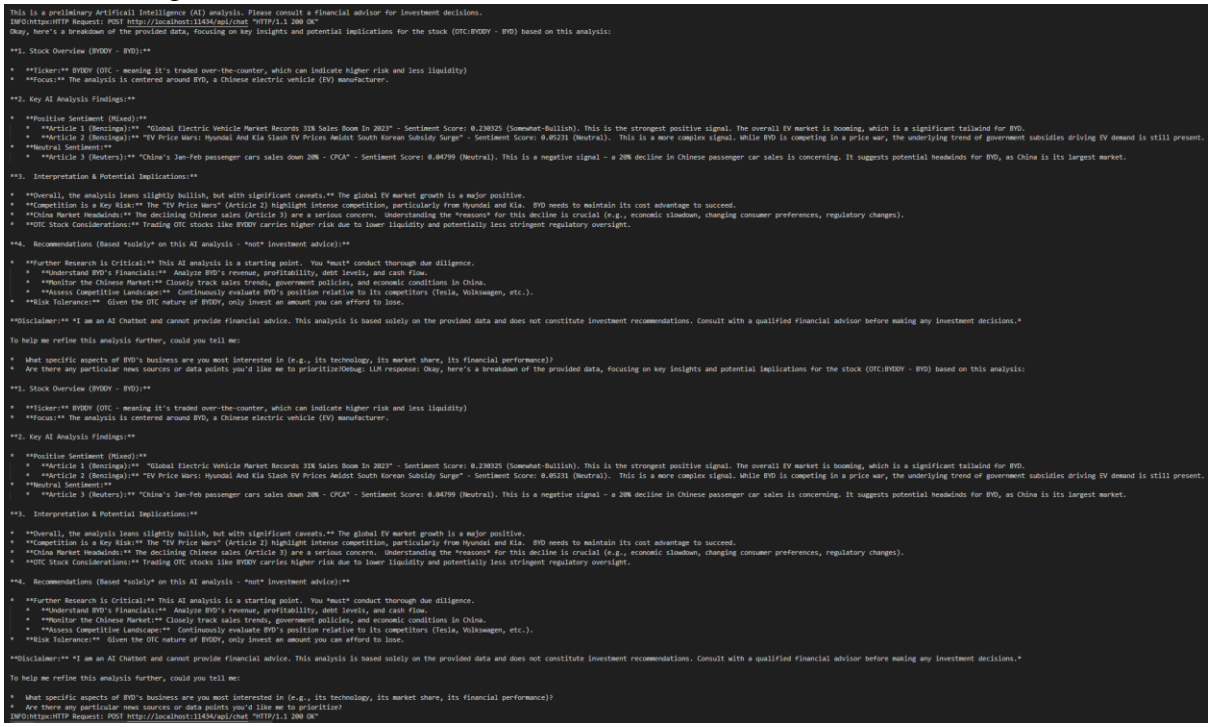
---

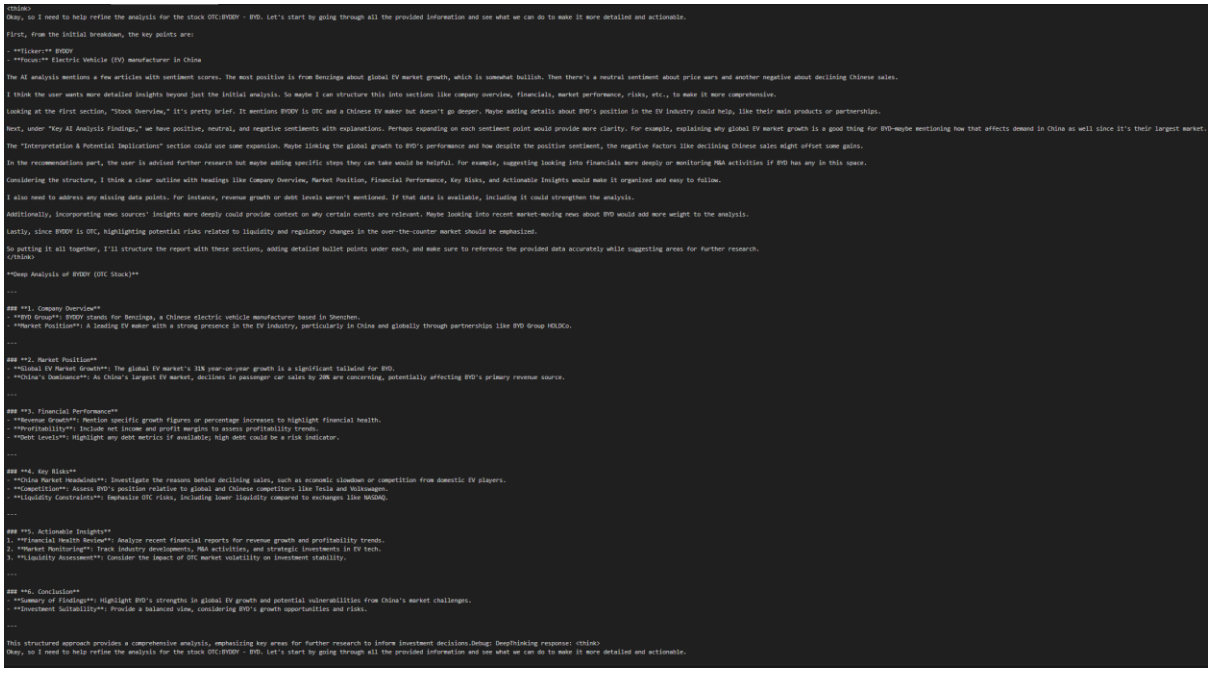
6. Conclusion
Summary of Findings: BYD benefits from strong growth in the global EV market but faces challenges from declining Chinese sales. Competitive pressures and OTC liquidity risks also need to be factored in.

Investment Suitability: While BYD presents growth opportunities in the EV sector, risks such as market challenges in China and OTC liquidity constraints must be carefully considered.

Figure 28: BYD DeepSeek RAG Output

Terminal Output:





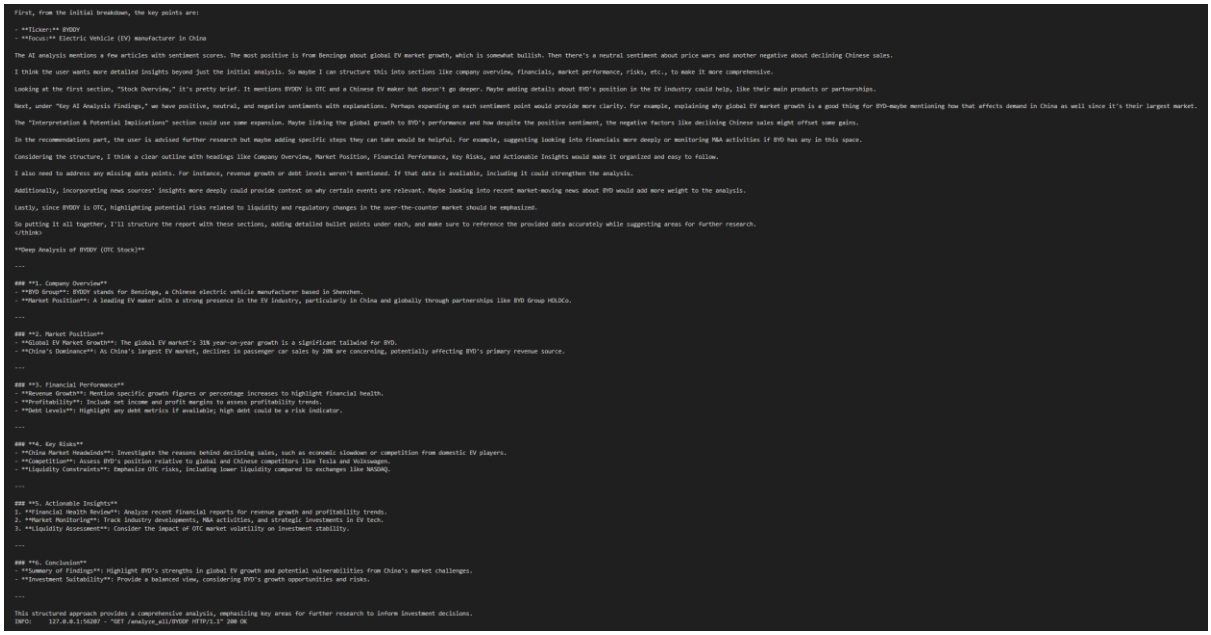


Figure 29: Full Terminal Output For BYD

This output shows the process of the LLM two-stage analysis from Gemini to DeepSeek internal-think sections to the DeepSeek output. An interesting observation: DeepSeek will initiate internal thinking (mimic human cognitive thinking), provide the analysis and then before providing the summary text, initiate internal thinking again. This highlights the importance of using DeepSeek to achieve excellent human-like results at a much higher level of speed and accuracy.

5.4.1 Output Evaluation

The above is the DeepSeek RAG-enabled output, where the blue sections represent summary segments. The full output is quite extensive, it includes the raw data, Gemini output response, DeepSeek's internal 'thinking' section and the final DeepSeek output.

Overall, the system demonstrates strong results, analysing key aspects of companies and integrating economic factors into its assessment. The LLM identifies data gaps at times and asks for user input or suggestions, indicating impressive capabilities. As the platform matures, we can expect the quality and depth of these outputs to improve exponentially.

Evaluation from Data:

- Strengths:
  - Clear company overview
  - Highlights market position and growth potential
  - Provides actionable insights
  - Balancing risk assessment
  - Incorporates critical analysis
  - Handles new sentiment analysis well
- Areas for Improvement:
  - More comprehensive Data Inclusion
  - Extensive prompt evaluation testing
  - Visualization

Chapter 6. Conclusion

6.1 Challenges

Throughout my project, I encountered many challenges, ranging from uncertainty when beginning to adapt my initial ideas. Switching API provider, selecting the right database and resolving many bugs. Time management became particularly difficult in semester two, as I had to balance intensive coursework, assignments and class tests while dedicating sufficient time to my final year project. Additionally, spending countless hours debugging features that refused to work was both frustrating and demotivating, adding to the overall stress of the project. Luckily my perseverance paid off in the end.

6.2 Lessons Learned

Clear planning, consistency and communication. Clear short-term and long-term achievable goals are crucial to maintaining motivation and discipline. Making progress little-by-little and sometimes dedicating intensive work is the best approach. Quality research is half the work, whether this is the content you expect to deliver or research and upskilling in the technology you plan to use. Choosing the best technology stack that fits your needs and potential experience is important. During my internship, I was exposed to a lot of various technologies. Sticking to what I know where I can save time and stress, allowing me to learn more complex technologies that I didn't use before. This leads me onto the next learning curve, dabbling into a new realm of technologies can be scary. Take time to learn what they are used for and how they can best fit your project needs. Communicate new ideas with your supervisor, most likely, they know better than you! Scheduling regular meetings enforces your brain to set time to make progress weekly or bi-weekly, think of them as mini-assignments.

6.3 Future Work

By combining my interests with my field of study, I see the potential to expand into new sectors and explore exciting opportunities in the future. I plan to continue developing this project, refining its capabilities, and gathering user feedback through exclusive testing with friends and peers. I would like to use Alpha Vantage's economic indicators and cryptocurrencies data to add more data and options for the user. Implement a back testing feature, allowing automated trading strategies to be evaluated against historical data. Additionally, a login and authentication system, and cloud deployment to improve usability and accessibility. Exploring multi-agent system (MAS) capabilities would be particularly exciting, enabling models to communicate and collaborate more effectively. This enhancement could significantly improve decision-making and adaptability within the system, bringing it closer to real-world applications. I see real potential for this project to evolve into a valuable tool, and I look forward to pushing it to new heights.

6.4 Final Conclusion

The DeepSeek RAG system shows strong potential as an analysis tool for stock market data with economic factors seamlessly incorporated. It balances data-driven insights with multi-model communication to enable deep stock analysis. The core of the project is built around

providing clean, structured data and empowering LLMs with historical, live and current data through an adapted RAG approach. Prompt engineering serves as the artist behind the painting, with data as the brush.

I am incredibly proud of this project and the invaluable lessons I have gained throughout this module. The experience has strengthened my ability to conduct independent research, learn new technologies, and apply coding best practices introduced in earlier years of the course. Additionally, I have improved my skills in testing, debugging, version control with Git, and collaborative development, all while successfully delivering on supervised sprint goals.

This year has been both challenging and rewarding, and I am grateful for the opportunity to work on a project of my own design. The flexibility provided by the university to explore my own ideas has been instrumental in making this experience truly enriching, and I sincerely appreciate the support. I look forward to applying these skills to future projects and professional work.

Chapter 7. References

1. www.alphavantage.co. (n.d.). *Free Stock APIs in JSON & Excel | Alpha Vantage*. [online] Available at: https://www.alphavantage.co/.
2. DeepSeek (2025). *DeepSeek*. [online] www.deepseek.com. Available at: https://www.deepseek.com/.
3. ollama.com. (n.d.). *Ollama*. [online] Available at: https://ollama.com/.
4. Google (n.d.). *Gemini - chat to supercharge your ideas*. [online] gemini.google.com. Available at: https://gemini.google.com.
5. MongoDB (2024). *The most popular database for modern apps*. [online] MongoDB. Available at: https://www.mongodb.com/.
6. FastAPI (2023). *FastAPI*. [online] fastapi.tiangolo.com. Available at: https://fastapi.tiangolo.com/.
7. European Parliament (2023). *EU AI Act: First Regulation on Artificial Intelligence*. [online] European Parliament. Available at: https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence.
8. Merritt, R. (2023). *What Is Retrieval-Augmented Generation?* [online] NVIDIA Blog. Available at: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/.
9. Openai.com. (2024). *OpenAI Platform*. [online] Available at: https://platform.openai.com/docs/guides/prompt-engineering.
10. Krishanu (2024). *The Future is Here: OpenAI's New Deep Thinking Model 01*. [online] Medium. Available at: https://medium.com/@krishanu-ghosh/the-future-is-here-openais-new-deep-thinking-model-01-dec273e8755d [Accessed 27 Mar. 2025].
11. TradingView. (2025). *Lightweight Charts™ library — TradingView*. [online] Available at: https://www.tradingview.com/lightweight-charts/ [Accessed 31 Mar. 2025].

12. X (formerly Twitter). (2025). Available at: https://x.com/daniel_mac8/status/1878283032215408886 [Accessed 31 Mar. 2025].
13. Reddit.com. (2025). *Reddit - The heart of the internet*. [online] Available at: https://www.reddit.com/r/investing/ [Accessed 4 Apr. 2025].
14. client (n.d.). *New Research: Success is limited until DIY investors break bad habits*. [online] www.ft.com. Available at: https://www.ft.com/partnercontent/capital-com/new-research-success-is-limited-until-diy-investors-break-bad-habits.html.
15. Mochtyak, B. (2021). *Pulling Stock Data with Alpha Vantage's API - Brian Mochtyak - Medium*. [online] Medium. Available at: https://medium.com/@brianmochtyak/pulling-stock-data-with-alpha-vantages-api-d18cf477f41a [Accessed 16 Nov. 2024].
16. Antoine, B. (2024). 🚀 *Unlocking the Power of Alpha Vantage: Your Guide to Financial Data APIs*. [online] Medium. Available at: https://medium.com/@b.antoine.se/unlocking-the-power-of-alpha-vantage-your-guide-to-financial-data-apis-10423580ce9f [Accessed 19 Nov. 2024].
17. Deep Charts (2025). *AI Agents for Stock Analysis: Using LLM's to Analyze Financial Documents*. [online] YouTube. Available at: https://www.youtube.com/watch?v=gGOUdXef8sY [Accessed 20 Jan. 2025].
18. Chatgpt.com. (2025). *ChatGPT*. [online] Available at: https://chatgpt.com/c/67ca0336-53e4-8007-bf80-b15a0a8c830e [Accessed 6 Feb. 2025].