



NED UNIVERSITY OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & IT

TSCS (with Specialization in Cybersecurity)

CT-361 Artificial Intelligence & Expert Systems

Complex Computing Problem

REPORT

Date: 10th May, 2025

Title: AlphaAI – A Multi-Agent Financial Analysis System

Group members:

Yasra Khan	CR-22005	(Group Lead)
Syed Ali Mujtaba Rizvi	CR-22011	
Barira Tariq	CR-22017	
Aleeza Mehmood	CR-22024	

ALPHA_{AI} – A MULTI-AGENT FINANCIAL ANALYSIS SYSTEM

Problem Statement

The stock market is influenced by various factors like news sentiment, financial health, technical indicators, and long-term fundamentals. Many investors struggle to make rational decisions due to information overload or lack of analysis tools.

Objective

To build an intelligent multi-agent system called **AlphaAI** that assists users in analyzing a company's stock by:

- Reading **news**
- Analyzing **stock price trends**
- Studying **long-term valuation**

This simulation mimics an **expert system** using **four AI agents**, each playing the role of an expert in its domain.

Research Area:

Natural Language Processing (NLP) for News Sentiment Analysis

- The `news_agent.py` uses **Google Search** to retrieve the stock-related news.
- Summarizes and filters top 4 relevant articles using **LLM-powered NLP** for clarity and relevance.
- This simulates a basic form of **sentiment estimation** based on headlines and summaries.

Technical Analysis Agent

- The `technical_agent.py` uses **Groq's LLaMA3 70B model** with `yfinance` tools.
- Implements standard algorithmic trading indicators:
 - **RSI, MACD, Moving Averages (50/200-day),**
 - **Support & Resistance, Fibonacci Retracements,** and
 - **Volume & Pattern Analysis.**
- Automates chart-based strategies to mimic human trader logic.

Fundamental Analysis Agent

- The `fundamental_agent.py` uses **Groq's LLaMA3 70B model**.
- Performs detailed **financial health evaluation** using:

- **Financial Statements and Key Ratios** (P/E, ROE, Debt-to-Equity, etc.),
 - **Growth Analysis, DCF Valuation,** and
 - **Competitor & Industry Benchmarking.**
- Utilizes yfinance tools to fetch and analyze company data.

Agent-Based Architecture

- Each analysis type (news, technical, fundamental) is handled by a **separate, modular agent**.
- This allows:
 - Clear **separation of concerns**,
 - Easy **scalability**
 - **Parallel development** or debugging of individual components.




Project Structure

AlphaAi/

```

|
|— main.py          <- Entry point of the project (runs everything)
|
|— agents/
|   |— financial_agent.py  <- Evaluates income, expenses, etc.
|   |— news_agent.py      <- Analyzes news for positive/negative words
|   |— technical_agent.py  <- Looks at RSI, MACD, trends
|   |— init.py           <- Empty, but required to mark Agents/ as a Python package so its
modules can be imported
|
|— Requirements
    |— requirements.txt <- prerequisite modules for this project
  
```

How the Multi-Agent Workflow Works

- **User interacts via the Phi Playground UI:**
 The user selects **one of the three available agents**:
 -  Technical Analyzer Agent
 -  News Search Agent
 -  Fundamental Analyzer Agent
 Then enters a **stock-related query** (e.g., "Analyze Apple Inc.").

- **Agent Execution Based on Selection:**
 - ◊ If **News Agent** is selected:
 - Fetches and summarizes the **latest news** about the queried company using Google Search.
 - ◊ If **Technical Agent** is selected:
 - Performs **technical stock analysis** using indicators like RSI, MACD, moving averages, and chart patterns.
 - ◊ If **Fundamental Agent** is selected:
 - Generates a **comprehensive financial analysis** report using company data (P/E, ROE, DCF, growth trends, etc.).
- **Agent Outputs Are Isolated & Contextual:**
 - Each agent works **independently**, responding only to the user's **query + selected agent type**

Benefits of This Interaction Model:

- Allows **user-driven control** of analysis focus.
- Keeps each agent **modular and independently testable**.
- Fits well within **multi-agent architectures** where specialized agents can be scaled or extended.

Framework

It is a clean, and modular multi-agent AI system using Phi Framework.

Using:

- LLMs (Groq) for smart reasoning,
- Data tools (yFinance, GoogleSearch) for external information,
- Phi agents to keep tasks separate but integrated,
- Playground UI for interaction.

Phi Framework: The Phi Framework is a lightweight Python framework designed for building, organizing, and deploying multi-agent AI systems that can interact with large language models (LLMs) and external tools. It's specifically useful for projects like chatbots, AI assistants, autonomous agents, or task automation tools.

Code and structure:

Main.py

```
from agents.news_agent import news_search_agent
from agents.technical_agent import technical_analyzer_agent
from agents.fundamental_agent import fundamental_analyzer_agent
```

```

from phi.playground import Playground, serve_playground_app
from dotenv import load_dotenv
import os
load_dotenv()

groq_api_key = os.getenv("GROQ_API_KEY")
phi_api_key = os.getenv("PHI_API_KEY")

news_agent = news_search_agent()
tech_agent = technical_analyzer_agent()
fundamental_agent = fundamental_analyzer_agent()

# Setup and serve the playground application
app = Playground(agents=[news_agent, tech_agent, fundamental_agent]).get_app()

if __name__ == "__main__":
    print("Starting the app...")
    serve_playground_app("main:app", reload=True)

```

This is the main.py file. It's like the brain that connects all agents together.

First, it loads the API keys securely from a .env file. Then, it creates three agents — one for news, one for technical analysis, and one for fundamentals. Finally, all of them are added into a playground — which is the interface users see. When we run this file, a web app opens, and the user can ask questions like ‘Analyze Tesla stock’ — and the selected agent generates the answer.

1. Imports all required agents and tools

- a. news_agent: Searches the stock-related news.
- b. technical_agent: Analyzes charts and stock movements.
- c. fundamental_agent: Analyzes the company's financials.

2. Importing Phi playground tools

- a. Playground: Provides a **built-in web UI** to talk with agents like a chatbot.
- b. serve_playground_app: Starts the web application using FastAPI + React.

3. Loading API keys Securely

- a. .env file stores your **secret keys** safely.
- b. os.getenv() reads them without exposing them in the code.

4. Creating agent instance

- a. Each agent is **initialized (started)** so they're ready to perform tasks.

5. Setting up the web interface

- a. All agents are passed into Playground.
- b. This creates a **web-based chatbot UI** where the user can ask questions like: “Should I invest in Tesla?”

6. Running the app

```
if __name__ == "__main__":
```

```
serve_playground_app("main:app", reload=True)
```

- If you run this Python file directly, it will:
- Start a local web server,
- Open the chatbot UI in your browser.
- reload=True means any code change automatically refreshes the app during development.

Fundamenta_agent.py

This script defines a **Fundamental Analyzer Agent** using the **Phi Framework** and **Groq's LLaMA 3 model**.

The agent uses **financial tools from Yahoo Finance** to evaluate a company's **financial health** and generate a full **fundamental analysis report**.

1. Importing required libraries:

```
2. import os
3. from dotenv import load_dotenv
4. from phi.agent import Agent
5. from phi.model.groq import Groq
6. from phi.tools.yfinance import YFinanceTools
```

- os: To access environment variables like your API key.
- dotenv.load_dotenv(): Loads secrets from a .env file (e.g., GROQ_API_KEY).
- Agent: Used to define a custom AI agent using the Phi framework.
- Groq: Connects to the **Groq cloud inference service**, which runs LLaMA 3 models.
- YFinanceTools: Provides access to stock data like financials, news, and key ratios using Yahoo Finance.

2. Load API Key Securely:

```
load_dotenv()
groq_api_key = os.getenv("GROQ_API_KEY")
```

- .env file stores secret keys securely (e.g., GROQ_API_KEY).
- os.getenv(...) loads the API key without hardcoding it in your script.

3. Define the fundamental_analyzer_agent function:

```
def fundamental_analyzer_agent():
```

This function **creates and returns** an AI agent specialized in **fundamental stock analysis**.

4. Create and return the Agent:

```
return Agent(
    name="Fundamental Analyzer Agent",
    model=Groq(id="llama3-groq-70b-8192-tool-use-preview", api_key=groq_api_key)
```

- a) name="Fundamental Analyzer Agent": Names the agent.
- b) model=Groq(...): Uses **Groq's LLaMA 3 (70B) model** with tool-use capability.

5. Add tools for financial data:

```
tools=[YFinanceTools(stock_price=True, analyst_recommendations=True,
company_info=True, company_news=True)],
```

This enables the agent to fetch data via Yahoo Finance:

- **Stock Price**
- **Analyst Recommendations**
- **Company Info**
- **News Headlines**

These tools allow the LLM to answer with real data, not just generic text.

6. Define instructions for the agent:

```
instructions=["Conduct fundamental analysis of the stock. Include:\n"
• "Financial Statements (3 Years): Review key highlights.\n"
• "Key Ratios: P/E, P/B, P/S, PEG, Debt-to-Equity, etc.\n"
• "Competitive Position: Strengths and market advantages.\n"
• "Management Effectiveness: Assess ROE and capital allocation.\n"
• "Growth Trends: Revenue and earnings trajectory.\n"
• "Growth Catalysts & Risks (2-3 Years): Identify key drivers and
challenges.\n"
• "DCF Valuation: Include assumptions and insights.\n"
• "Competitor & Industry Comparison: Analyze against peers and averages."
],
```

This gives the agent a clear step-by-step **task list** for conducting the analysis:

- Reviews financial statements
- Calculates ratios like **P/E, Debt-to-Equity**
- Evaluates **management** and **growth potential**
- Runs a basic **Discounted Cash Flow (DCF)** model
- Compares with **competitors/industry**

7. Extra Settings:

```
8. show_tool_calls=True,
9. markdown=True,
```

- a) show_tool_calls=True: Shows which financial tool (e.g., Yahoo Finance) the agent used.
- b) markdown=True: Formats the output nicely (bold, headers, bullet points).

News_agent.py

This script defines a **News Search Agent** that uses Google Search tools and Groq's AI model to find and summarize the **4 most relevant news items** about a given stock or company.

1. Import required libraries:

```
2. import os
3. from dotenv import load_dotenv
4. from phi.agent import Agent
5. from phi.model.groq import Groq
6. from phi.tools.googlesearch import GoogleSearch
```

- a) os: Used to access environment variables like API keys.
- b) load_dotenv(): Loads sensitive data from a .env file.
- c) Agent: Class from Phi framework to define AI agents.
- d) Groq: Connects the agent to the LLaMA 3 model running on Groq cloud.
- e) GoogleSearch: Adds web search capability using Google Search API.

2. Load API Key Securely:

```
load_dotenv()
groq_api_key = os.getenv("GROQ_API_KEY")
```

- a) .env file securely stores the **Groq API key**.
- b) os.getenv("GROQ_API_KEY") retrieves it without exposing it in the code.

3. Define the news_search_agent function:

```
def news_search_agent():
```

This function **returns an AI agent** focused on searching and summarizing **news**.

4. Return a configured Agent:

```
return Agent(
    name="News Search Agent",
    model=Groq(id="llama3-groq-70b-8192-tool-use-preview", api_key=groq_api_key)
```

- a) name="News Search Agent": Gives the agent a readable identity.
- b) model=Groq(...): Uses **LLaMA 3 (70B)** via Groq for fast and accurate language processing.

5. Add Google Search as a Tool:

```
tools=[GoogleSearch()]
```

- a) Adds **real-time search capability** to the agent.
- b) Enables the agent to search Google for information.

6. Add description and instructions:


```
description="You are a news agent that helps users find the latest news.",
instructions=[
    "User Query: Respond with 4 of the latest news items on the given stock.\n",
    "Search Process: Retrieve 10 recent news items in English, then select the top 4
unique and relevant items.\n",
    "Output: Provide concise and clear summaries of the selected news items."
],
```

- a) description: Gives the AI a role — **finding news**.
- b) instructions: Gives it a detailed **task list**:

- Perform a **Google search** for the stock.
- Fetch up to **10 news items**.
- Filter down to the **top 4 most relevant**.
- Return **brief summaries** of those 4.

7. Display options:

```
show_tool_calls=True,
markdown=True,
```

- a) show_tool_calls=True: Displays which tool (like Google Search) was used.
- b) markdown=True: Formats the output with headers, lists, bold, etc.

Technical_agent.py

This agent uses Groq's AI model and Yahoo Finance tools to analyze **stock charts**, **indicators**, and **price patterns** for technical trading insights.

1. Import Required Libraries:

```
2.
3. import os
4. from dotenv import load_dotenv
5. from phi.agent import Agent
6. from phi.model.groq import Groq
7. from phi.tools.yfinance import YFinanceTools
8.
```

- a) os: Access environment variables like API keys.
- b) load_dotenv(): Loads keys securely from a .env file.
- c) Agent: Base class from the **Phi framework** to define intelligent agents.
- d) YFinanceTools: Adds tools to fetch stock data like prices, indicators, and charts.

2. Load API Key Securely:

```
# Load environment variables
load_dotenv()
groq_api_key = os.getenv("GROQ_API_KEY")
```

- a) Reads your .env file to securely access the groq_api_key.
- b) Keeps your API key hidden from the codebase for security.

3. Define the Technical Analyzer Agent:

```
def technical_analyzer_agent():
```

- a) This function creates and returns an AI-powered stock chart analyst.

4. Configure the Agent:

```
return Agent(
    name="Technical Analyzer Agent",
    model=Groq(id="llama3-70b-8192", api_key=groq_api_key),
```

- a) name: Gives a readable name to the agent.
- b) model: Uses Groq Model, a fast and powerful AI model.

5. Add Tools for Real-Time Stock Data:

```
tools=[YFinanceTools(stock_price=True, technical_indicators=True,
historical_prices=True)]
```

Adds access to:

- technical_indicators: RSI, MACD, etc.
- historical_prices: Past price data used for trends and patterns

6 Define the Agent's Role & Task:

```
• description="You are a technical analysis agent that helps users analyze stock prices
and trends.",
• instructions=["Perform technical analysis on the stock. Include:\n"
• "Moving Averages (1 Year): 50-day & 200-day, with crossovers.\n"
• "Support & Resistance: 3 levels each, with significance.\n"
• "Volume Analysis (3 Months): Trends and anomalies.\n"
• "RSI & MACD: Compute and interpret signals.\n"
• "Fibonacci Levels: Calculate and analyze.\n"
• "Chart Patterns (6 Months): Identify 3 key patterns.\n"
• "Sector Comparison: Contrast with sector averages."
• ],
```

- a) description: Explains the **purpose** of the agent (chart-based analysis).
- b) instructions: Tells the model **what to analyze**, including:
 - **Moving Averages:** 50-day and 200-day trends and golden/death crossovers
 - **Support & Resistance:** Key price levels
 - **Volume:** Spikes, trends, unusual activity

- **RSI & MACD:** Momentum indicators to spot overbought/sold signals
- **Fibonacci Retracement:** Predict support/resistance using ratios
- **Chart Patterns:** Patterns like head-and-shoulders, flags, etc.
- **Sector Comparison:** How the stock performs vs. its industry

9. Output Settings:

```
10. show_tool_calls=True,
11.     markdown=True,
```

- a) `show_tool_calls=True`: Shows which tools the agent used (e.g., yFinance).
- b) `markdown=True`: Formats output in a readable way with headings, lists, etc.

Conclusion

AlphaAI demonstrates how powerful AI agents can work collaboratively in a financial analysis setting. By dividing the workload into specialized agents (News, Technical, Fundamental), the system mimics how human analysts approach investing—with research, data interpretation, and contextual awareness.

This modular, agent-based design leads to:

- **Deeper insights** from different angles (news, charts, financials).
- **Faster decision-making** for investors, powered by LLMs.
- **Scalability** — new agents can be added later, such as ESG (Environmental, Social, Governance) or Social Media sentiment agents.

This project proves the effectiveness of multi-agent AI architecture, not just in finance, but in any domain requiring diverse intelligence like healthcare, law, or supply chain.

Appendix

YouTube: <https://youtu.be/3VQQTVPe6Zo>

GitHub: <https://github.com/MujtabaRizvi01/AIES-CCP-Project.git>