**ARTIFICAL INTELLIGENCE**

**MINI PROJECT**

Multi-Agent Financial Analysis System Using Agentic AI

Mini Project Report

# 1. Objectives

• To develop a multi-agent AI system capable of providing real-time financial insights using web search and stock data analysis.

• To demonstrate the effectiveness of collaborative agentic AI in streamlining financial decision-making and information retrieval.

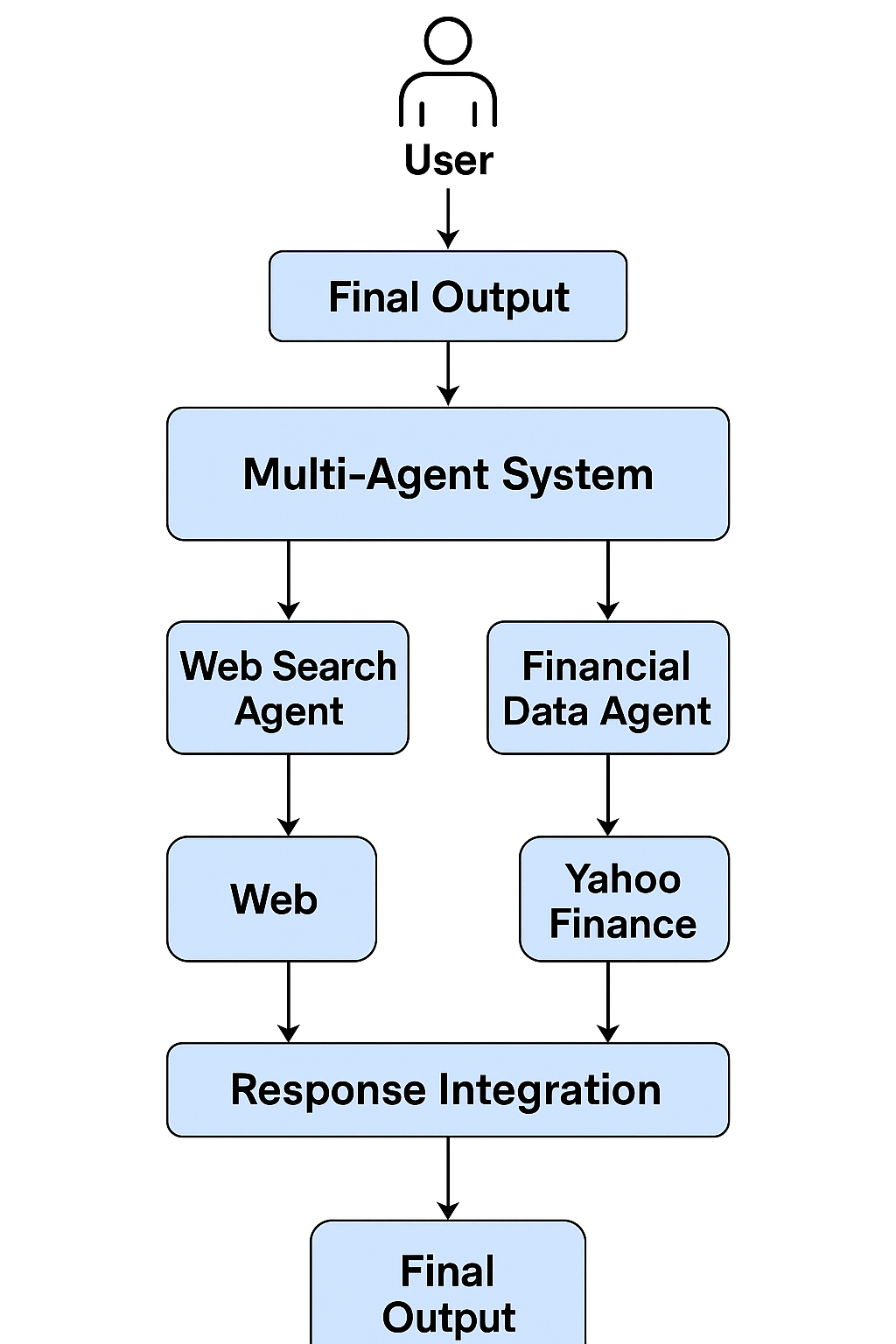
# 2. Abstract

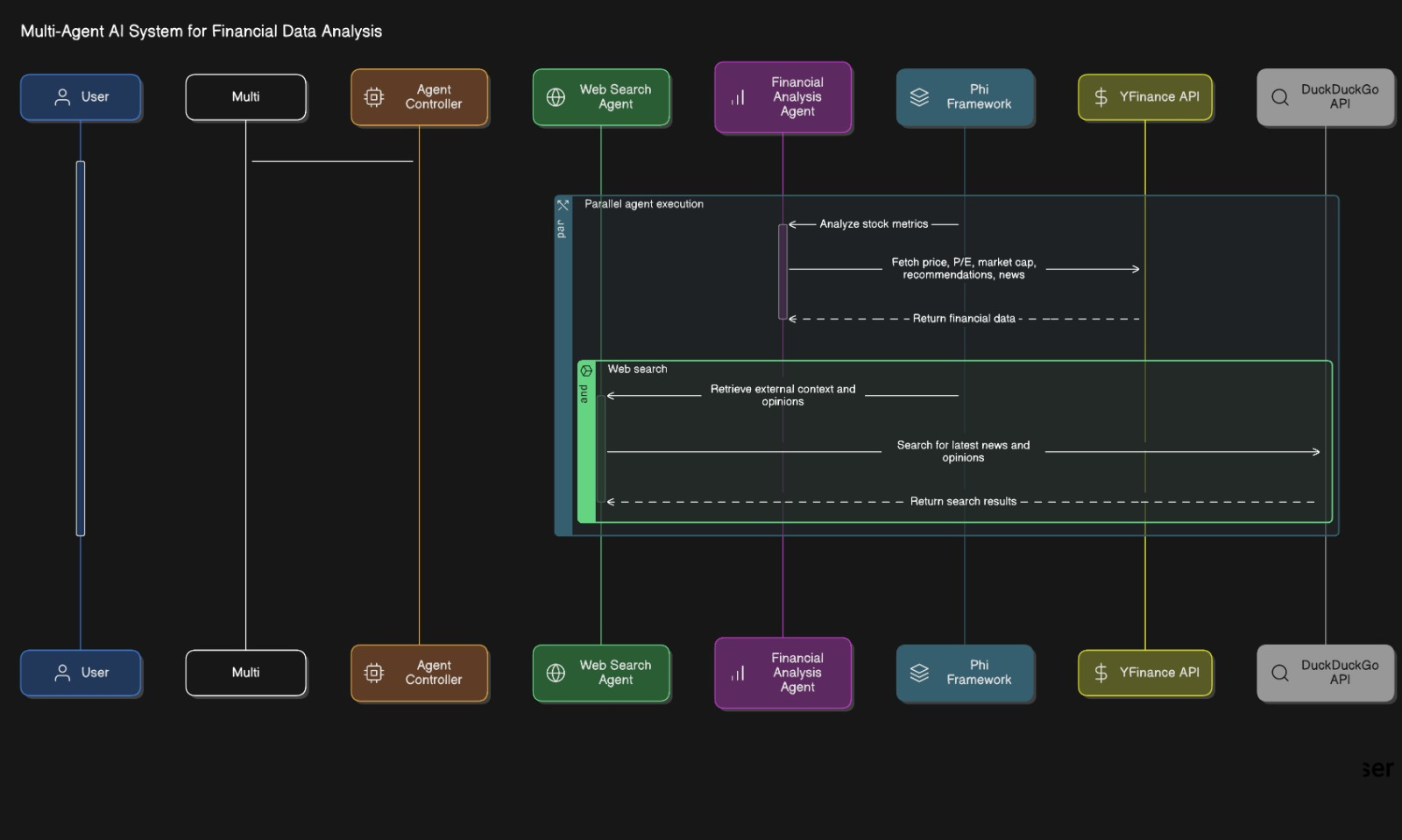
This mini project presents a multi-agent AI system designed to automate financial data analysis and information retrieval. Leveraging the capabilities of the Phi framework, this project integrates multiple specialized agents—specifically a web search agent and a financial analysis agent—to provide accurate and up-to-date insights into company stocks. The financial agent utilizes yfinance and other tools to extract key metrics such as current price, P/E ratio, market capitalization, analyst recommendations, and recent news. In parallel, the web search agent uses DuckDuckGo to gather additional context and external opinions, ensuring a broader view of the stock market.  
  
A unique aspect of this system is its agent orchestration. The multi-agent controller efficiently combines outputs from individual agents into a cohesive, human-readable summary that is both informative and interactive. With modular design principles, the project is easily extendable to include more agents for different domains or tasks in the future.  
  
The tool has immense potential for deployment in real-world environments, such as financial portals, research platforms, or trading dashboards. Future scope includes integration with advanced LLMs, support for multiple data sources, and server deployment via hosted URLs. The system is a strong demonstration of the potential of agentic AI in the financial domain and beyond.

# 3. Data Source Link

* Yahoo Finance API (via yfinance): <https://www.yahoofinanceapi.com/>
* Groq API : <https://groq.com//>
* Gemini API: <https://ai.google.dev/>

# 4. Architecture Diagram



z

# 5.Bio-Inspired Algorithm (Stepwise Explanation)

Inspired by stigmergy (from nature, e.g., ant colonies), the system operates in the following steps:

1. Initialization: Each agent is initialized with a unique role and toolset.

2. Task Distribution: The multi-agent controller decides which agent handles which part of the query.

3. Local Processing: Financial Agent fetches stock data and analysis; Web Search Agent retrieves articles and trends.

4. Information Aggregation: The controller merges results into one response.

5. Adaptation Loop: User input can refine the query, mimicking environmental feedback.

# 6. Code

from phi.agent import Agent

from phi.model.google import Gemini

from phi.tools.yfinance import YFinanceTools

from phi.tools.duckduckgo import DuckDuckGo

import os

from dotenv import load\_dotenv

import yfinance as yf

import inspect

# Load environment variables

load\_dotenv()

Gemini.api\_key = os.getenv("AIzaSyDwW51XtnY95xmBTz0kCOVVbdZZdm3zQkg")

## Web Search Agent

web\_search\_agent = Agent(

    name="Web Search Agent",

    role="Search the web for accurate and up-to-date financial information",

    model=Gemini(id="gemini-2.0-flash-exp"),

    tools=[DuckDuckGo()],

    instructions=[

        "1. Perform targeted searches to find the most relevant and recent financial information.",

        "2. Always verify the credibility of sources before including them in the response.",

        "3. Summarize the information concisely and include direct links to the sources.",

        "4. If no relevant information is found, state this clearly and suggest alternative search terms.",

    ],

    show\_tool\_calls=True,

    markdown=True,

)

## Financial Agent

financial\_agent = Agent(

    name="Finance AI Agent",

    model=Gemini(id="gemini-2.0-flash-exp"),

    tools=[

        YFinanceTools(

            stock\_price=True,

            stock\_fundamentals=True,

            analyst\_recommendations=True,

            company\_news=True,

        )

    ],

    instructions=[

        "1. Use tables to display stock data for better readability.",

        "2. Always include the following details for any stock:",

        "   - Current price",

        "   - Key fundamentals (e.g., P/E ratio, market cap)",

        "   - Latest analyst recommendations (Buy/Hold/Sell)",

        "   - Recent company news (last 7 days)",

        "3. If data is unavailable, explain why and suggest alternative tools or sources.",

        "4. Keep the response concise and avoid unnecessary details.",

    ],

    show\_tool\_calls=True,

    markdown=True,

)

## Multi-AI Agent

multi\_ai\_agent = Agent(

    model=Gemini(id="gemini-2.0-flash-exp"),

    team=[web\_search\_agent, financial\_agent],

    instructions=[

        "1. Use the web search agent to find the latest financial news and updates.",

        "2. Use the financial agent to analyze stock data and provide structured insights.",

        "3. Always combine the results from both agents into a single, cohesive response.",

        "4. Follow these formatting guidelines:",

        "   - Use headings to separate sections (e.g., 'Latest News', 'Stock Analysis').",

        "   - Use tables for numerical data.",

        "   - Include clickable links for all sources.",

        "5. If the user's query is unclear, ask for clarification before proceeding.",

    ],

    show\_tool\_calls=True,

    markdown=True,

)

"""

# Ask the user for a query at runtime

user\_query = input("Enter your query: ")

# Run the agent with the user's input

multi\_ai\_agent.print\_response(user\_query, stream=True)

"""

# Example Query

multi\_ai\_agent.print\_response(

    "Summarize analyst recommendations and share the latest news for IRFC.",

    stream=True,

)

def get\_user\_query():

    #Interactively collect user input for a detailed query.

    user\_query = input("Enter your query: ")

    while True:

        # Run the agent with streaming enabled

        response = multi\_ai\_agent.run(user\_query, stream=True)

        # Stream and print response chunk by chunk

        for chunk in response:

            print(chunk.content, end="", flush=True)  # Print the streamed response in real time

        print("\n")  # Add a newline after the response

        # Ask if more clarification is needed

        follow\_up = input("\nDo you want to add more details? (Type your response or 'quit' to proceed): ")

        if follow\_up.lower() == "quit":

            break  # Stop asking for more details

         #  Instead of appending, \*\*replace\*\* the query with the new details

        user\_query = follow\_up  # Reset the query to only the new follow-up input

    return user\_query

# Get the refined user query

final\_query = get\_user\_query()

# Stream the final response

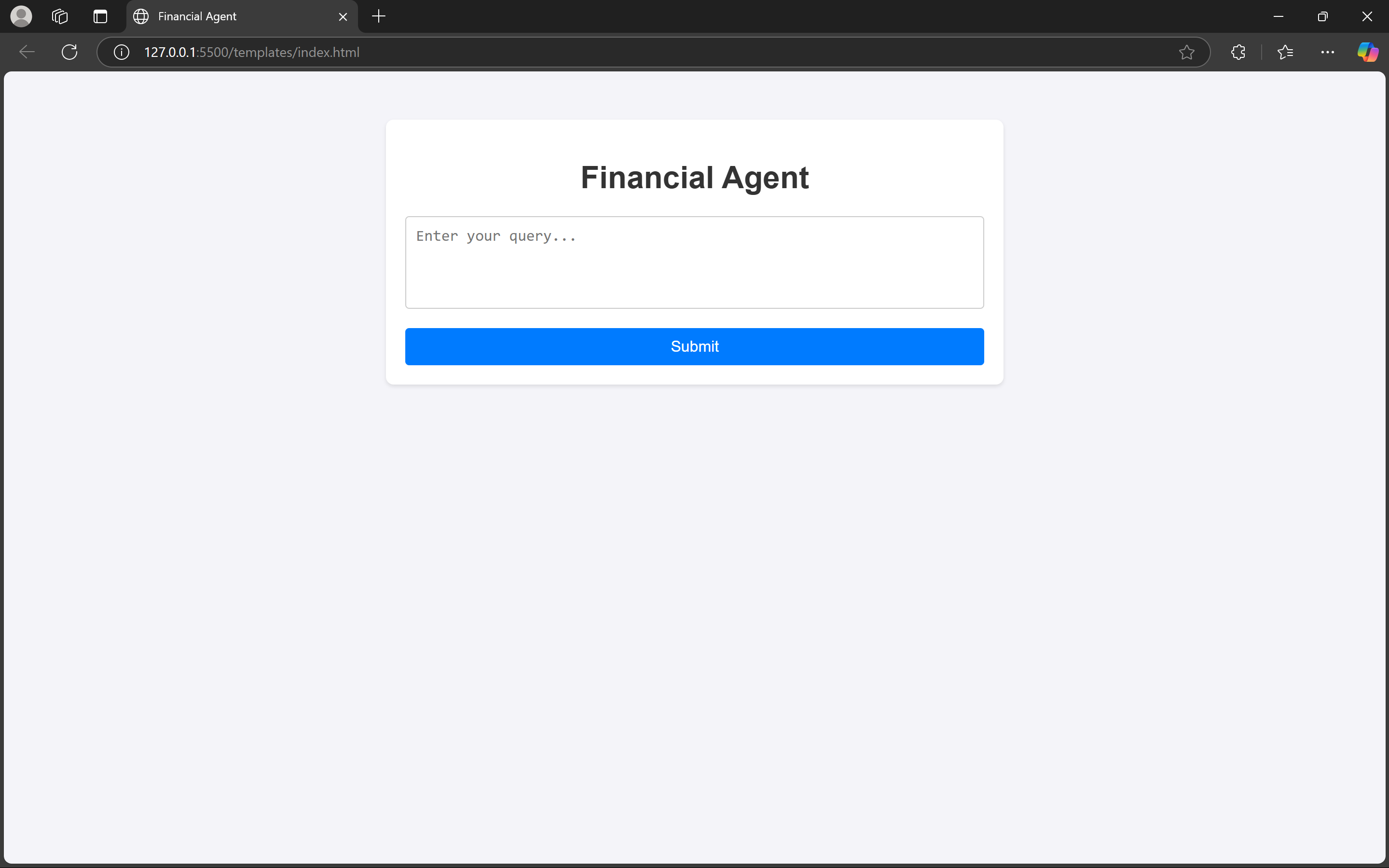
print("\nFinal Response:")

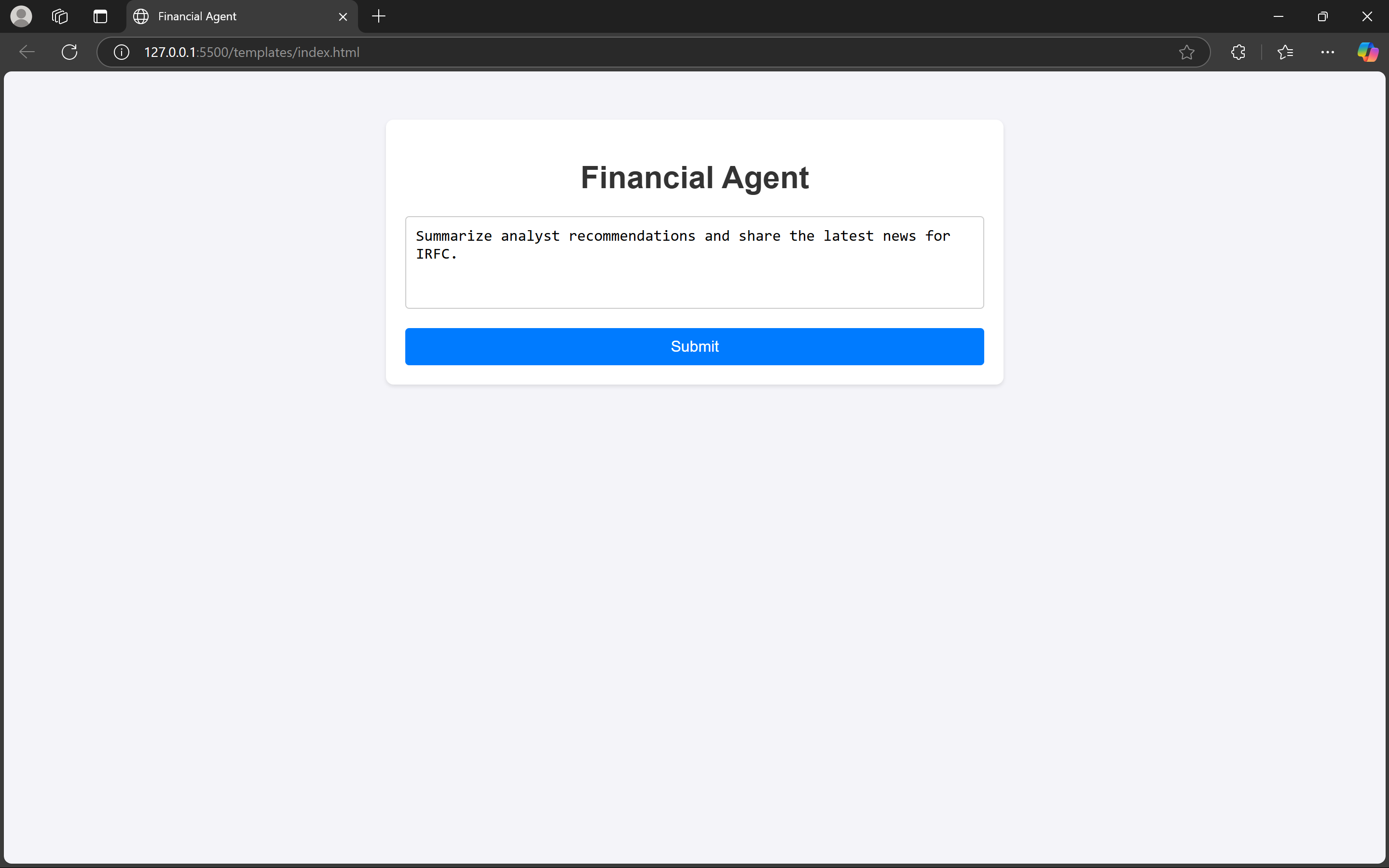
final\_response = multi\_ai\_agent.run(final\_query, stream=True)

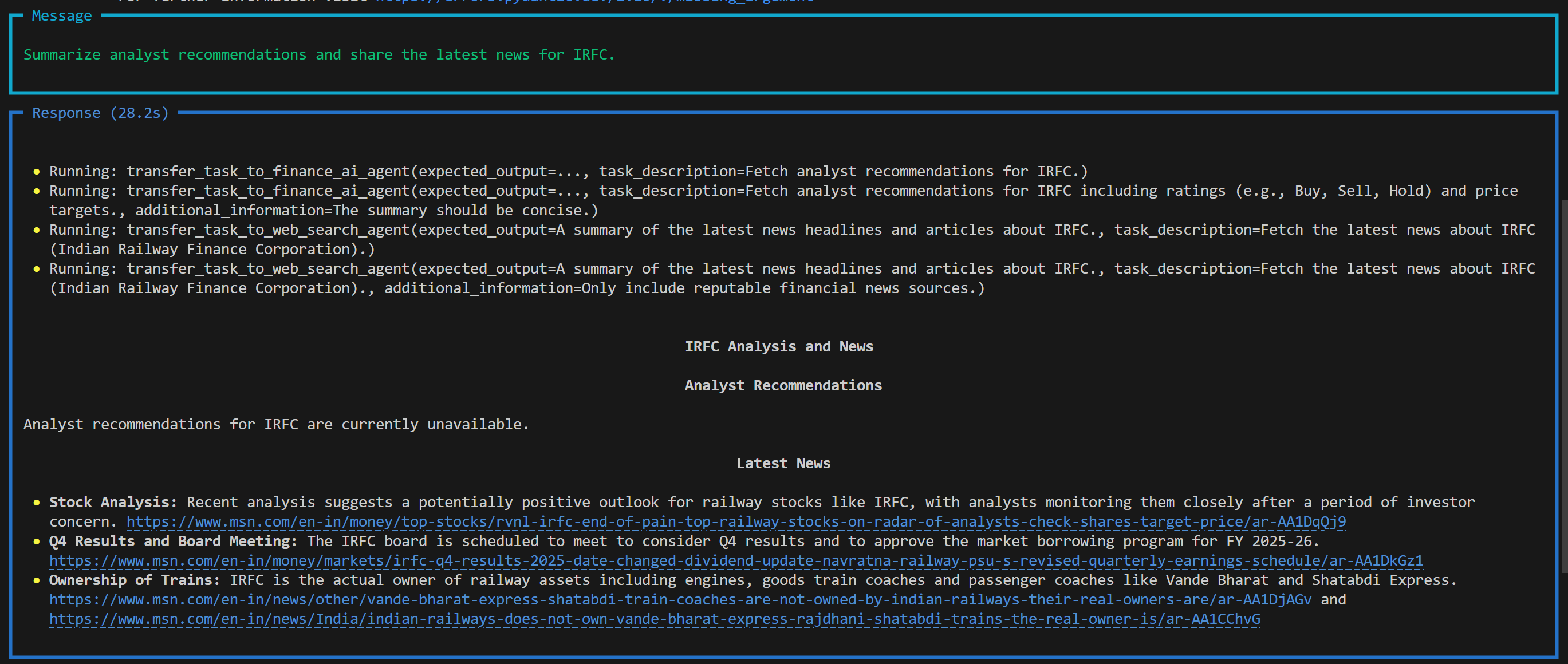
for chunk in final\_response:

    print(chunk.content, end="", flush=True)  # Stream the final response in real-time

# 7. Output







# 8. Conlclusion

This project demonstrates the power of multi-agent AI systems in transforming financial data processing. By using separate agents for data extraction and contextual analysis, the system provides a more holistic view of market trends. The modular and extensible architecture opens up future opportunities for integration with trading platforms, server deployment, and expansion to other domains such as legal or healthcare analysis. The success of this prototype paves the way for a robust, intelligent, agentic AI ecosystem.