Building a Multi-Agent System for Real-Time Financial Analysis: A Comprehensive Tutorial

In this tutorial, we'll walk through the process of building a sophisticated multi-agent system for real-time financial analysis using the Swarms framework. This system is designed for financial analysts and developer analysts who want to leverage AI and multiple data sources to gain deeper insights into stock performance, market trends, and economic indicators.

Before we dive into the code, let's briefly introduce the Swarms framework. Swarms is an innovative open-source project that simplifies the creation and management of AI agents. It's particularly well-suited for complex tasks like financial analysis, where multiple specialized agents can work together to provide comprehensive insights.

For more information and to contribute to the project, visit the [Swarms GitHub repository](https://github.com/kyegomez/swarms). We highly recommend exploring the documentation for a deeper understanding of Swarms' capabilities.

Additional resources:

- [Swarms Discord](https://discord.com/servers/agora-999382051935506503) for community discussions
- [Swarms Twitter](https://x.com/swarms_corp) for updates
- [Swarms Spotify](https://open.spotify.com/show/2HLiswhmUaMdjHC8AUHcCF?si=c831ef10c5ef4994) for podcasts
- [Swarms Blog](https://medium.com/@kyeg) for in-depth articles
- [Swarms Website](https://swarms.xyz) for an overview of the project

Now, let's break down our financial analysis system step by step.

Step 1: Setting Up the Environment

First install the necessary packages:

```bash
$ pip3 install -U swarms yfiance swarm_models fredapi pandas
```

First, we need to set up our environment and import the necessary libraries:

```python
import os
import time
from datetime import datetime, timedelta
import yfinance as yf
import requests
from fredapi import Fred
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from swarms import Agent, AgentRearrange
from swarm_models import OpenAIChat
import logging
from dotenv import load_dotenv
import asyncio
```

```
import aiohttp
from ratelimit import limits, sleep_and_retry

# Load environment variables
load_dotenv()

# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# API keys
POLYGON_API_KEY = os.getenv('POLYGON_API_KEY')
FRED_API_KEY = os.getenv('FRED_API_KEY')
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')

# Initialize FRED client
fred_client = Fred(api_key=FRED_API_KEY)

# Polygon API base URL
POLYGON_BASE_URL = "https://api.polygon.io"
```

This section sets up our environment, imports necessary libraries, and initializes our API keys and clients. We're using `dotenv` to securely manage our API keys, and we've set up logging to track the execution of our script.

Step 2: Implementing Rate Limiting

To respect API rate limits, we implement rate limiting decorators:

```python
@sleep_and_retry
@limits(calls=5, period=60)  # Adjust these values based on your Polygon API tier
async def call_polygon_api(session, endpoint, params=None):
    url = f"{POLYGON_BASE_URL}{endpoint}"
    params = params or {}
    params['apiKey'] = POLYGON_API_KEY
    async with session.get(url, params=params) as response:
        response.raise_for_status()
        return await response.json()

@sleep_and_retry
@limits(calls=120, period=60)  # FRED allows 120 requests per minute
def call_fred_api(func, *args, **kwargs):
    return func(*args, **kwargs)
```

These decorators ensure that we don't exceed the rate limits for our API calls. The `call_polygon_api` function is designed to work with asynchronous code, while `call_fred_api` is a wrapper for synchronous FRED API calls.

Step 3: Implementing Data Fetching Functions

Next, we implement functions to fetch data from various sources:

Yahoo Finance Integration

```python
async def get_yahoo_finance_data(session, ticker, period="1d", interval="1m"):
    try:
        stock = yf.Ticker(ticker)
        hist = await asyncio.to_thread(stock.history, period=period, interval=interval)
        info = await asyncio.to_thread(lambda: stock.info)
        return hist, info
    except Exception as e:
        logger.error(f"Error fetching Yahoo Finance data for {ticker}: {e}")
        return None, None

async def get_yahoo_finance_realtime(session, ticker):
    try:
        stock = yf.Ticker(ticker)
        return await asyncio.to_thread(lambda: stock.fast_info)
    except Exception as e:
        logger.error(f"Error fetching Yahoo Finance realtime data for {ticker}: {e}")
        return None
```

These functions fetch historical and real-time data from Yahoo Finance. We use `asyncio.to_thread`

to run the synchronous `yfinance` functions in a separate thread, allowing our main event loop to continue running.

Polygon.io Integration

```python
async def get_polygon_realtime_data(session, ticker):
    try:
        trades = await call_polygon_api(session, f"/v2/last/trade/{ticker}")
        quotes = await call_polygon_api(session, f"/v2/last/nbbo/{ticker}")
        return trades, quotes
    except Exception as e:
        logger.error(f"Error fetching Polygon.io realtime data for {ticker}: {e}")
        return None, None

async def get_polygon_news(session, ticker, limit=10):
    try:
        news = await call_polygon_api(session, f"/v2/reference/news", params={"ticker": ticker, "limit": limit})
        return news.get('results', [])
    except Exception as e:
        logger.error(f"Error fetching Polygon.io news for {ticker}: {e}")
        return []
```

These functions fetch real-time trade and quote data, as well as news articles from Polygon.io. We

use our `call_polygon_api` function to make these requests, ensuring we respect rate limits.

FRED Integration

```python
async def get_fred_data(session, series_id, start_date, end_date):
    try:
        data = await asyncio.to_thread(call_fred_api, fred_client.get_series, series_id, start_date, end_date)
        return data
    except Exception as e:
        logger.error(f"Error fetching FRED data for {series_id}: {e}")
        return None

async def get_fred_realtime(session, series_ids):
    try:
        data = {}
        for series_id in series_ids:
            series = await asyncio.to_thread(call_fred_api, fred_client.get_series, series_id)
            data[series_id] = series.iloc[-1]  # Get the most recent value
        return data
    except Exception as e:
        logger.error(f"Error fetching FRED realtime data: {e}")
        return {}
```

These functions fetch historical and real-time economic data from FRED. Again, we use `asyncio.to_thread` to run the synchronous FRED API calls in a separate thread.

Step 4: Creating Specialized Agents

Now we create our specialized agents using the Swarms framework:

```python
stock_agent = Agent(
    agent_name="StockAgent",
     system_prompt="""You are an expert stock analyst. Your task is to analyze real-time stock data and provide insights.
    Consider price movements, trading volume, and any available company information.
    Provide a concise summary of the stock's current status and any notable trends or events.""",
    llm=OpenAIChat(api_key=OPENAI_API_KEY),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)

market_agent = Agent(
    agent_name="MarketAgent",
      system_prompt="""You are a market analysis expert. Your task is to analyze overall market conditions using real-time data.
    Consider major indices, sector performance, and market-wide trends.
```

```
    Provide a concise summary of current market conditions and any significant developments.""",
    llm=OpenAIChat(api_key=OPENAI_API_KEY),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)

macro_agent = Agent(
    agent_name="MacroAgent",
        system_prompt="""You are a macroeconomic analysis expert. Your task is to analyze key economic indicators and provide insights on the overall economic situation.
    Consider GDP growth, inflation rates, unemployment figures, and other relevant economic data.
      Provide a concise summary of the current economic situation and any potential impacts on financial markets.""",
    llm=OpenAIChat(api_key=OPENAI_API_KEY),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)

news_agent = Agent(
    agent_name="NewsAgent",
     system_prompt="""You are a financial news analyst. Your task is to analyze recent news articles related to specific stocks or the overall market.
```

```
    Consider the potential impact of news events on stock prices or market trends.
    Provide a concise summary of key news items and their potential market implications.""",
    llm=OpenAIChat(api_key=OPENAI_API_KEY),
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)
```

Each agent is specialized in a different aspect of financial analysis. The `system_prompt` for each agent defines its role and the type of analysis it should perform.

Step 5: Building the Multi-Agent System

We then combine our specialized agents into a multi-agent system:

```python
agents = [stock_agent, market_agent, macro_agent, news_agent]
flow = "StockAgent -> MarketAgent -> MacroAgent -> NewsAgent"

agent_system = AgentRearrange(agents=agents, flow=flow)
```

The `flow` variable defines the order in which our agents will process information. This allows for a logical progression from specific stock analysis to broader market and economic analysis.

Step 6: Implementing Real-Time Analysis

Now we implement our main analysis function:

```python
async def real_time_analysis(session, ticker):
    logger.info(f"Starting real-time analysis for {ticker}")

    # Fetch real-time data
    yf_data, yf_info = await get_yahoo_finance_data(session, ticker)
    yf_realtime = await get_yahoo_finance_realtime(session, ticker)
    polygon_trades, polygon_quotes = await get_polygon_realtime_data(session, ticker)
    polygon_news = await get_polygon_news(session, ticker)
    fred_data = await get_fred_realtime(session, ['GDP', 'UNRATE', 'CPIAUCSL'])

    # Prepare input for the multi-agent system
    input_data = f"""
    Yahoo Finance Data:
    {yf_realtime}

    Recent Stock History:
    {yf_data.tail().to_string() if yf_data is not None else 'Data unavailable'}

    Polygon.io Trade Data:
    {polygon_trades}
```

```
    Polygon.io Quote Data:
    {polygon_quotes}

    Recent News:
    {polygon_news[:3] if polygon_news else 'No recent news available'}

    Economic Indicators:
    {fred_data}

    Analyze this real-time financial data for {ticker}. Provide insights on the stock's performance, overall market conditions, relevant economic factors, and any significant news that might impact the stock or market.
    """

    # Run the multi-agent analysis
    try:
        analysis = agent_system.run(input_data)
        logger.info(f"Analysis completed for {ticker}")
        return analysis
    except Exception as e:
        logger.error(f"Error during multi-agent analysis for {ticker}: {e}")
        return f"Error during analysis: {e}"
```

This function fetches data from all our sources, prepares it as input for our multi-agent system, and

then runs the analysis. The result is a comprehensive analysis of the stock, considering individual performance, market conditions, economic factors, and relevant news.

Step 7: Implementing Advanced Use Cases

We then implement more advanced analysis functions:

Compare Stocks

```python
async def compare_stocks(session, tickers):
    results = {}
    for ticker in tickers:
        results[ticker] = await real_time_analysis(session, ticker)

    comparison_prompt = f"""
    Compare the following stocks based on the provided analyses:
    {results}

    Highlight key differences and similarities. Provide a ranking of these stocks based on their current performance and future prospects.
    """

    try:
        comparison = agent_system.run(comparison_prompt)
        logger.info(f"Stock comparison completed for {tickers}")
```

```
        return comparison
    except Exception as e:
        logger.error(f"Error during stock comparison: {e}")
        return f"Error during comparison: {e}"
```

This function compares multiple stocks by running a real-time analysis on each and then prompting our multi-agent system to compare the results.

Sector Analysis

```python
async def sector_analysis(session, sector):
    sector_stocks = {
        'Technology': ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'NVDA'],
        'Finance': ['JPM', 'BAC', 'WFC', 'C', 'GS'],
        'Healthcare': ['JNJ', 'UNH', 'PFE', 'ABT', 'MRK'],
        'Consumer Goods': ['PG', 'KO', 'PEP', 'COST', 'WMT'],
        'Energy': ['XOM', 'CVX', 'COP', 'SLB', 'EOG']
    }

    if sector not in sector_stocks:
        return f"Sector '{sector}' not found. Available sectors: {', '.join(sector_stocks.keys())}"

    stocks = sector_stocks[sector][:5]
```

```
    sector_data = {}
    for stock in stocks:
        sector_data[stock] = await real_time_analysis(session, stock)

    sector_prompt = f"""
    Analyze the {sector} sector based on the following data from its top stocks:
    {sector_data}

    Provide insights on:
    1. Overall sector performance
    2. Key trends within the sector
    3. Top performing stocks and why they're outperforming
    4. Any challenges or opportunities facing the sector
    """

    try:
        analysis = agent_system.run(sector_prompt)
        logger.info(f"Sector analysis completed for {sector}")
        return analysis
    except Exception as e:
        logger.error(f"Error during sector analysis for {sector}: {e}")
        return f"Error during sector analysis: {e}"
```

This function analyzes an entire sector by running real-time analysis on its top stocks and then prompting our multi-agent system to provide sector-wide insights.

Economic Impact Analysis

```python
async def economic_impact_analysis(session, indicator, threshold):
    # Fetch historical data for the indicator
    end_date = datetime.now().strftime('%Y-%m-%d')
    start_date = (datetime.now() - timedelta(days=365)).strftime('%Y-%m-%d')
    indicator_data = await get_fred_data(session, indicator, start_date, end_date)

    if indicator_data is None or len(indicator_data) < 2:
        return f"Insufficient data for indicator {indicator}"

    # Check if the latest value crosses the threshold
    latest_value = indicator_data.iloc[-1]
    previous_value = indicator_data.iloc[-2]
     crossed_threshold = (latest_value > threshold and previous_value <= threshold) or (latest_value < threshold and previous_value >= threshold)

    if crossed_threshold:
        impact_prompt = f"""
         The economic indicator {indicator} has crossed the threshold of {threshold}. Its current value is {latest_value}.

        Historical data:
        {indicator_data.tail().to_string()}
```

```
        Analyze the potential impacts of this change on:
        1. Overall economic conditions
        2. Different market
        2. Different market sectors
        3. Specific types of stocks (e.g., growth vs. value)
        4. Other economic indicators

        Provide a comprehensive analysis of the potential consequences and any recommended actions for investors.
        """

        try:
            analysis = agent_system.run(impact_prompt)
            logger.info(f"Economic impact analysis completed for {indicator}")
            return analysis
        except Exception as e:
            logger.error(f"Error during economic impact analysis for {indicator}: {e}")
            return f"Error during economic impact analysis: {e}"
    else:
        return f"The {indicator} indicator has not crossed the threshold of {threshold}. Current value: {latest_value}"
```

This function analyzes the potential impact of significant changes in economic indicators. It fetches historical data, checks if a threshold has been crossed, and if so, prompts our multi-agent system to

provide a comprehensive analysis of the potential consequences.

Step 8: Running the Analysis

Finally, we implement our main function to run all of our analyses:

```python
async def main():
    async with aiohttp.ClientSession() as session:
        # Example usage
        analysis_result = await real_time_analysis(session, 'AAPL')
        print("Single Stock Analysis:")
        print(analysis_result)

        comparison_result = await compare_stocks(session, ['AAPL', 'GOOGL', 'MSFT'])
        print("\nStock Comparison:")
        print(comparison_result)

        tech_sector_analysis = await sector_analysis(session, 'Technology')
        print("\nTechnology Sector Analysis:")
        print(tech_sector_analysis)

        gdp_impact = await economic_impact_analysis(session, 'GDP', 22000)
        print("\nEconomic Impact Analysis:")
        print(gdp_impact)
```

```
if __name__ == "__main__":
    asyncio.run(main())
```

This `main` function demonstrates how to use all of our analysis functions. It runs a single stock analysis, compares multiple stocks, performs a sector analysis, and conducts an economic impact analysis.

Conclusion and Next Steps

This tutorial has walked you through the process of building a sophisticated multi-agent system for real-time financial analysis using the Swarms framework. Here's a summary of what we've accomplished:

1. Set up our environment and API connections
2. Implemented rate limiting to respect API constraints
3. Created functions to fetch data from multiple sources (Yahoo Finance, Polygon.io, FRED)
4. Designed specialized AI agents for different aspects of financial analysis
5. Combined these agents into a multi-agent system
6. Implemented advanced analysis functions including stock comparison, sector analysis, and economic impact analysis

This system provides a powerful foundation for financial analysis, but there's always room for expansion and improvement. Here are some potential next steps:

1. **Expand data sources**: Consider integrating additional financial data providers for even more

comprehensive analysis.

2. **Enhance agent specialization**: You could create more specialized agents, such as a technical analysis agent or a sentiment analysis agent for social media data.

3. **Implement a user interface**: Consider building a web interface or dashboard to make the system more user-friendly for non-technical analysts.

4. **Add visualization capabilities**: Integrate data visualization tools to help interpret complex financial data more easily.

5. **Implement a backtesting system**: Develop a system to evaluate your multi-agent system's performance on historical data.

6. **Explore advanced AI models**: The Swarms framework supports various AI models. Experiment with different models to see which performs best for your specific use case.

7. **Implement real-time monitoring**: Set up a system to continuously monitor markets and alert you to significant changes or opportunities.

Remember, the Swarms framework is a powerful and flexible tool that can be adapted to a wide range of complex tasks beyond just financial analysis. We encourage you to explore the [Swarms GitHub repository](https://github.com/kyegomez/swarms) for more examples and inspiration.

For more in-depth discussions and community support, consider joining the [Swarms Discord](https://discord.com/servers/agora-999382051935506503). You can also stay updated with

the latest developments by following [Swarms on Twitter](https://x.com/swarms_corp).

If you're interested in learning more about AI and its applications in various fields, check out the [Swarms Spotify podcast](https://open.spotify.com/show/2HLiswhmUaMdjHC8AUHcCF?si=c831ef10c5ef4994) and the [Swarms Blog](https://medium.com/@kyeg) for insightful articles and discussions.

Lastly, don't forget to visit the [Swarms Website](https://swarms.xyz) for a comprehensive overview of the project and its capabilities.

By leveraging the power of multi-agent AI systems, you're well-equipped to navigate the complex world of financial markets. Happy analyzing!

Swarm Resources:

* [Swarms Github](https://github.com/kyegomez/swarms)
* [Swarms Discord](https://discord.com/servers/agora-999382051935506503)
* [Swarms Twitter](https://x.com/swarms_corp)
* [Swarms Spotify](https://open.spotify.com/show/2HLiswhmUaMdjHC8AUHcCF?si=c831ef10c5ef4994)
* [Swarms Blog](https://medium.com/@kyeg)
* [Swarms Website](https://swarms.xyz)