Multi-Agent Financial Analysis System

Authors: Ramesh Dhanasekaran, Mostafa Zamani Turk, Victoria Dorn, Atul Aneja, and Juan Pablo Triana Martinez.

School: University of San Diego Shiley-Marcos School of Engineering

Course: AAI-520 Natural Language Processing and Generative AI

Professor Siamak, Aram. Ph. D.

October 15, 2025

Abstract:

With the rapid hardware development of graphical processing units (NVDIA, 2025), tensor processing units (Vahdat, A. 2025) in the big technological field, the computational ability of computers has increased exponentially, allowing for the building and creation of much more complex Transformer LLMs architectures, mixed assembly of experts, or localized regions for knowledge-specific tasks. This year, modules used to create agentic AI like ChatGPT, n8n, CrewAI, Langraph, and many more have now become another set of tools that a current machine learning engineer, AI engineer, Data Scientist, or even someone who performs a lot of data preprocessing and fundamental machine learning must master now to use this new source of knowledge. To demonstrate this, the following report talks about a multi-agent financial system built with CrewAI, whereby using 5 fundamental agents (Earnings, News, Market, Investment, and Critic), created using OpenAI API key, it's able to access the web into the Federal Reserve Bank of St Louis data, and News using their respective API keys, to generate a .txt file containing a through financial report indicating for a specific ticker stock weather to buy, sell, or hold.

Output:

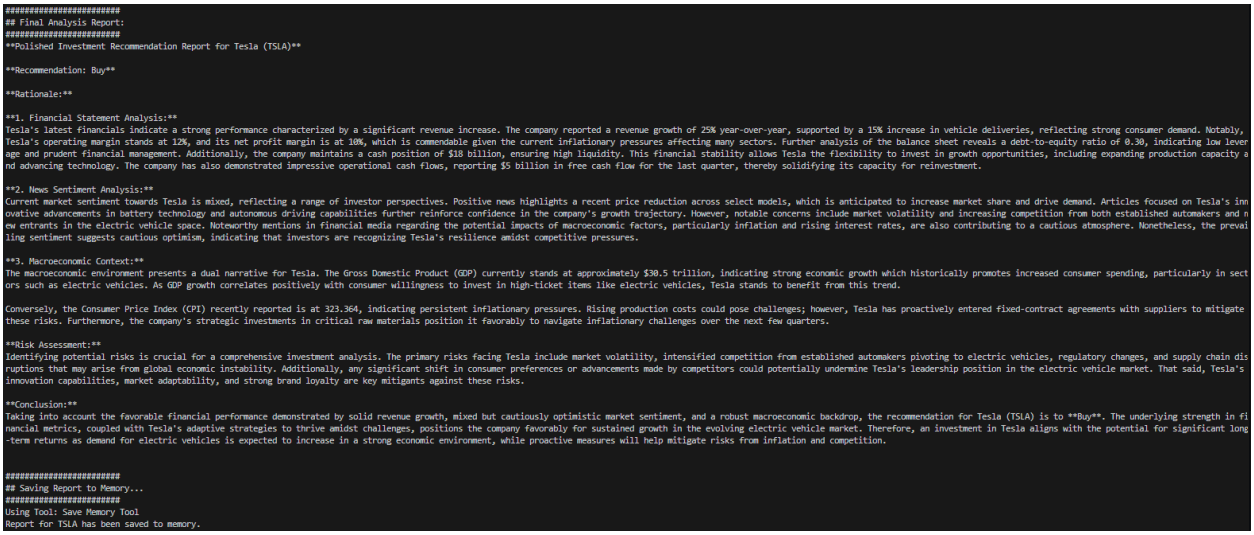

Image 1: Terminal Output of Final Report after executing the entire multi-agent pipeline

Setup:

The following must be performed once the project has been git cloned to a specific directory in your computer, the following is the link: [vdorn5/aai-520-final-project]

- Create a .env file containing OPENAI_API_KEY, NEWS_API_KEY, FRED_API_KEY.
- Pip install all requirements to your virtual environment: pip install -r requirements.txt
- Run the code with python src/orchestrator.py

This Framework works with paid credits of OpenAI. It's important to point that out before using it. From this point onwards, the terminal will be your best friend, showcasing how the LLM agents think, retrieve, and analyze each specific task.

Multi-Agent Flow Diagram:

The following is the entire multi-agent financial analysis system flow diagram of information.

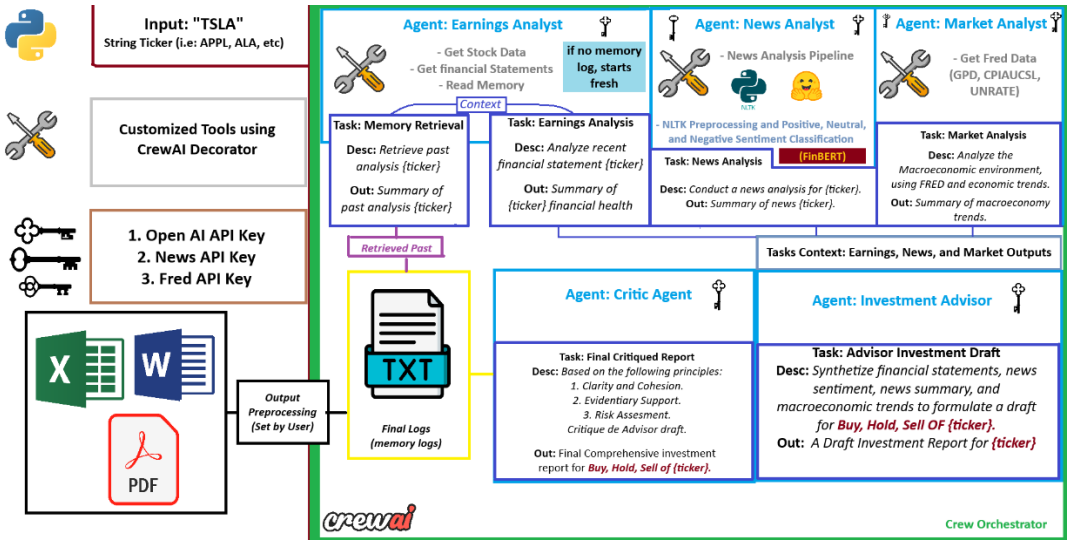

Image 2: Multi-Financial Agent system flow diagram

The interesting thing about this picture is that the output (i.e: final logs.txt), this file would be used as a memory for new runs for the earnings analyst agent to consider when re-running the entire framework again. This allows this system to "re-assess" passed financial reports and use them as context for the most updated research. The holding thing starts with a string input of a specific ticker, ending with the log output file, as shown in the following images:

```
if __name__ == "__main__":
    stock_ticker = "TSLA"
    print(f"🚀 Starting analysis for {stock_ticker}...")
    run_analysis(stock_ticker)
```

Image 3: String input ticker, TSLA for testing.

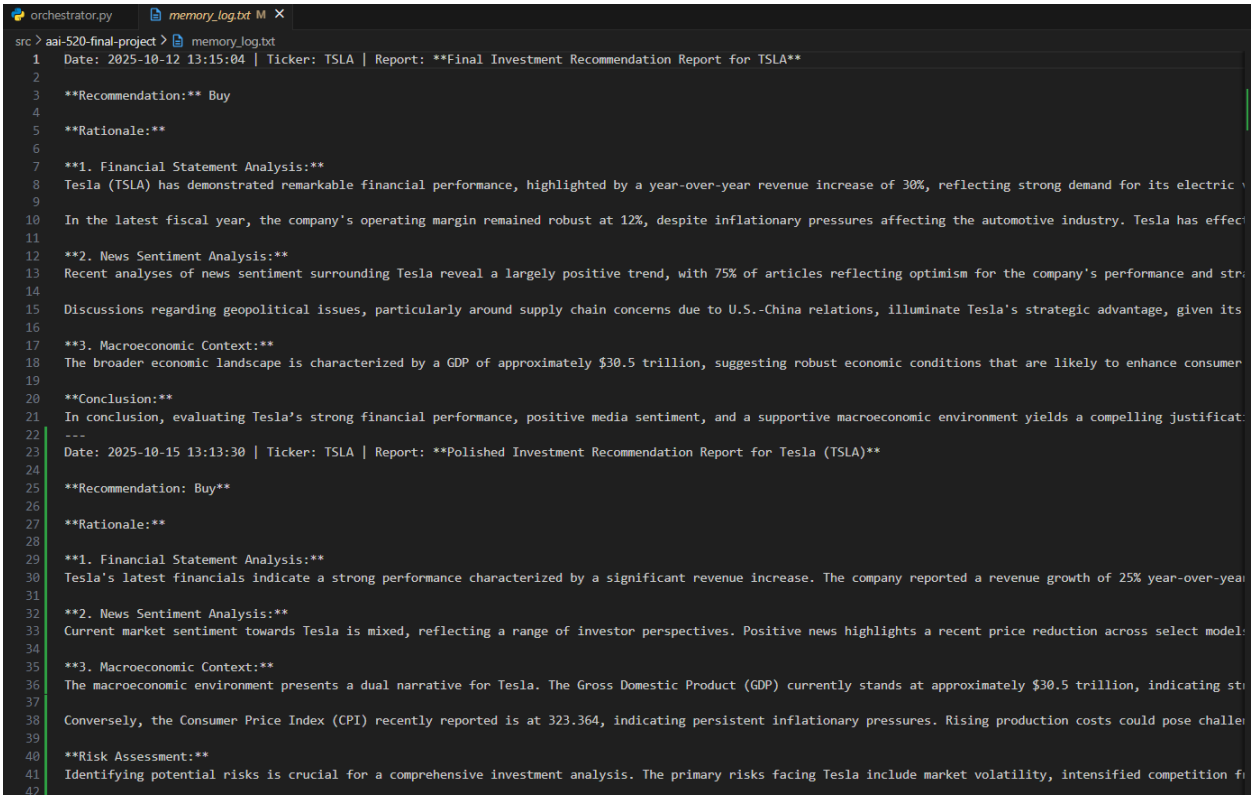

Image 4: memory_log.txt output file, past run 2025-10-12, current run, 2025-10-15.

By changing only one string line called stock_ticker, the framework would give you a concise financial report. Without further ado, let's jump into each of the agents and their functionalities

Earnings Analyst Agent:

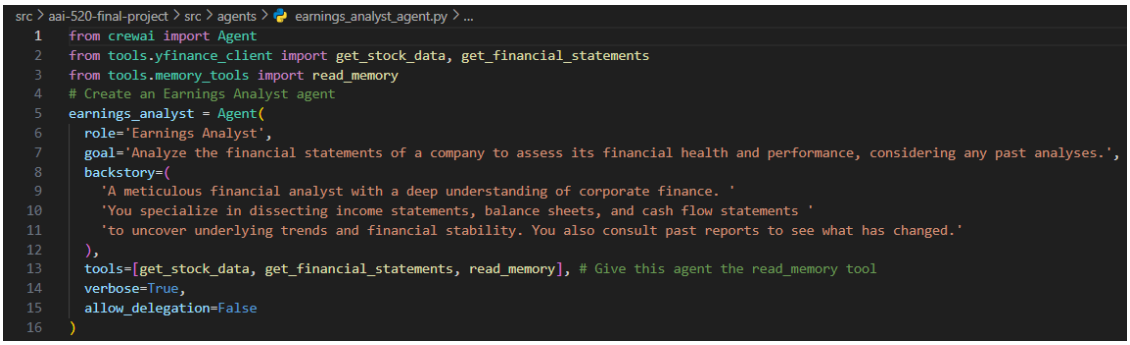

Image 5: Earnings Analyst Agent CrewAI code

As shown in Image 5, the earnings analyst agent has the goal to analyze financial statements, income statements, balance sheets, and cash flow statements to determine the financial health of a specific company. Interestingly, this agent has two tasks defined in Image 6:

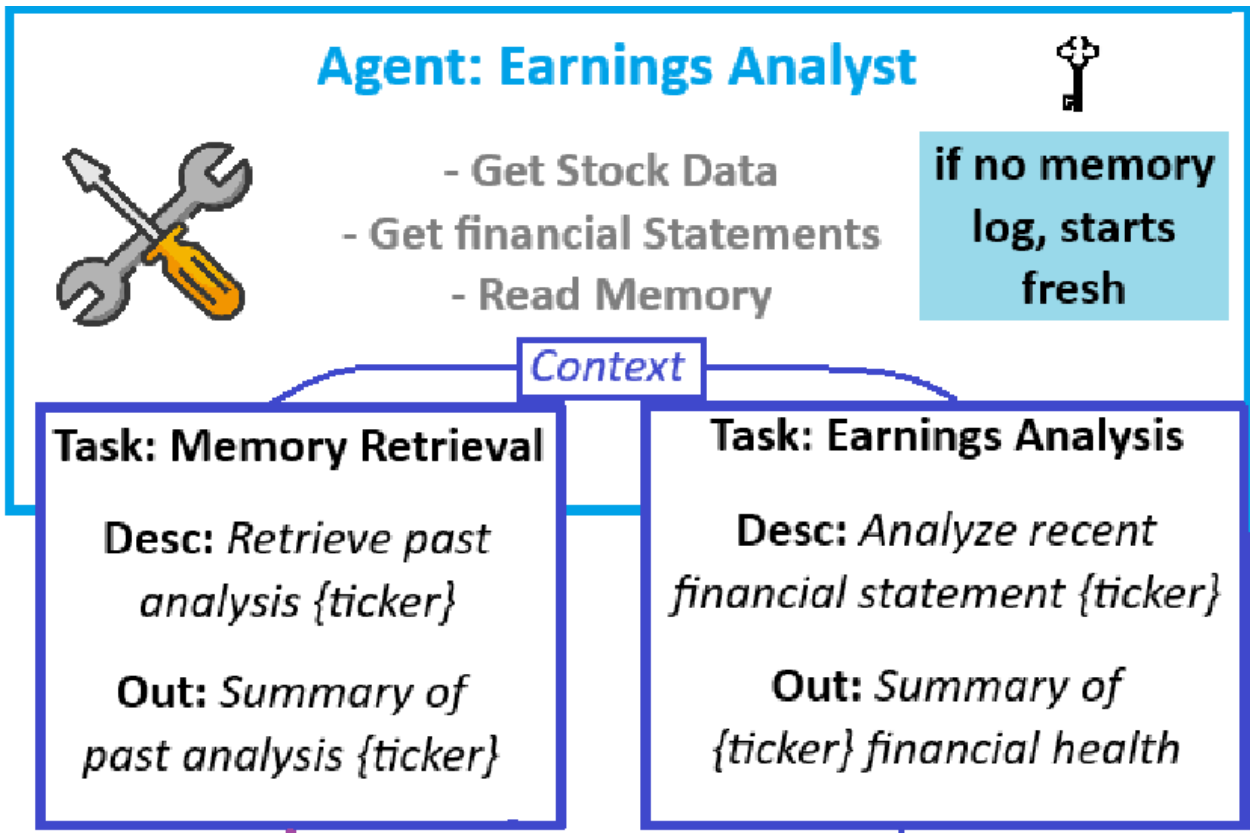

Image 6: Earnings Analyst Agent Input-Output Block Diagram. Assigne with specific tools and two different tasks, memory retrieval and earnings analysis.

Before we dive into the tasks, it's time to showcase the 3 customized tools this agent has:

```python
from crewai.tools import tool
import yfinance as yf
# --- Tool for Basic Stock Data ---
@tool("Stock Ticker Data Tool")
def get_stock_data(ticker: str) -> dict:
    """A tool to get basic financial data for a given stock ticker."""
    # Implementation remains the same...
    stock = yf.Ticker(ticker)
    info = stock.info
    return {
        "longName": info.get("longName"),
        "sector": info.get("sector"),
        "industry": info.get("industry"),
        "marketCap": info.get("marketCap"),
        "trailingPE": info.get("trailingPE"),
        "forwardPE": info.get("forwardPE"),
        "dividendYield": info.get("dividendYield"),
        "fiftyTwoWeekHigh": info.get("fiftyTwoWeekHigh"),
        "fiftyTwoWeekLow": info.get("fiftyTwoWeekLow"),
        "regularMarketPrice": info.get("regularMarketPrice")
    }
```

```python
# --- Tool for Financial Statements (for EarningsAnalyst) ---
@tool("Company Financial Statements Tool")
def get_financial_statements(ticker: str) -> str:
    """
    A tool to get the most recent annual financial statements (Income Statement,
    Balance Sheet, and Cash Flow) for a given stock ticker.
    """
    stock = yf.Ticker(ticker)

    # Fetch the most recent annual data
    income_statement = stock.income_stmt.iloc[:, 0]
    balance_sheet = stock.balance_sheet.iloc[:, 0]
    cash_flow = stock.cashflow.iloc[:, 0]

    # Format into a single string for the LLM
    report = f"""
    Income Statement:\n{income_statement.to_string()}\n
    Balance Sheet:\n{balance_sheet.to_string()}\n
    Cash Flow Statement:\n{cash_flow.to_string()}
    """
    return report
```

Image 7 & 8: Customized crewai tools using yfinance module to retrieve financial ticker info.

Using the yfinance module from Python, adapted with the tool's decorator from CrewAI, these two functions are used to get all of the financial information required from Yahoo Finance.Well… this is just for the earnings analysis task, what about the memory retrieval task?

```python
from crewai import Task

memory_retrieval_task = Task(
    description='Retrieve any past analysis for the stock {ticker} from the memory log using the Read Memory Tool.',
    expected_output='A summary of past analysis for {ticker}, or a statement that no prior analysis was found.',
    agent=None # Will be assigned to earnings_analyst
)

# Task for the Earnings Analyst
earnings_analysis_task = Task(
  description=(
    'Analyze the most recent annual financial statements for the stock {ticker}. '
    'First, consider any insights from prior analyses provided in the context. '
    'Then, examine the Income Statement, Balance Sheet, and Cash Flow Statement to identify '
    'key trends in revenue, profitability, debt, and cash flow. '
    'Provide a summary of the company\'s financial health.'
  ),
  expected_output=(
    'A concise summary of the company\'s financial health based on its latest '
    'annual financial statements, highlighting key trends and figures, and noting any changes from past analyses.'
  ),
  agent=None,
  context=[memory_retrieval_task]
)
```

Image 9: Financial Tasks for Earnings Analyst Agent.

This is where the read_memory customized tool comes into play; essentially, this will check whether a log file exists. If it does, it reads from it as input for the LLM to use in its financial analysis; otherwise, it starts a brand-new log file.

```python
from crewai.tools import tool


SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
PROJECT_ROOT = os.path.dirname(os.path.dirname(SCRIPT_DIR))
MEMORY_LOG_FILE = os.path.join(PROJECT_ROOT, "memory_log.txt")

# --- Tools for Memory ---
@tool("Read Memory Tool")
def read_memory(ticker: str) -> str:
    """A tool to read the memory log for any prior analysis on a given stock ticker."""
    try:
        with open(MEMORY_LOG_FILE, "r") as f:
            full_log = f.read()

        entries = full_log.split("\n---\n")

        relevant_memories = [entry for entry in entries if ticker.upper() in entry]

        if not relevant_memories:
            return f"No prior analysis found for {ticker}."

        return "\n---\n".join(relevant_memories)
    except FileNotFoundError:
        return "No memory log found. Starting fresh."
```

Image 10: Read Memory tool for Earnings Analyst Agent.

News Analyst Agent:

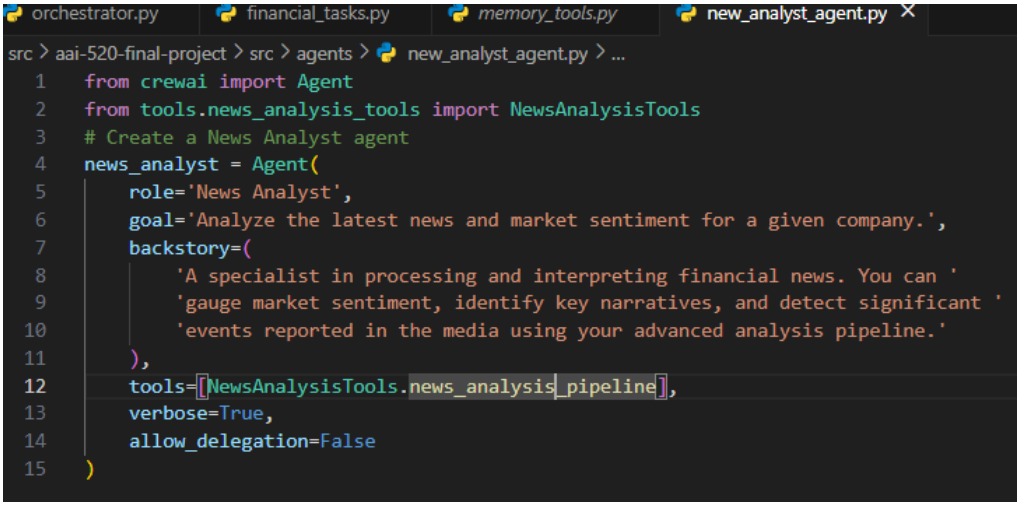

Image 11: News Analysis Agent using CrewAI code

Once the Earnings analyst agent has retrieve a summary of the financial health of the desired ticker company, the next agent would be to retrieve the global news of that specific ticker company. This is where the News analysis agent comes into play:

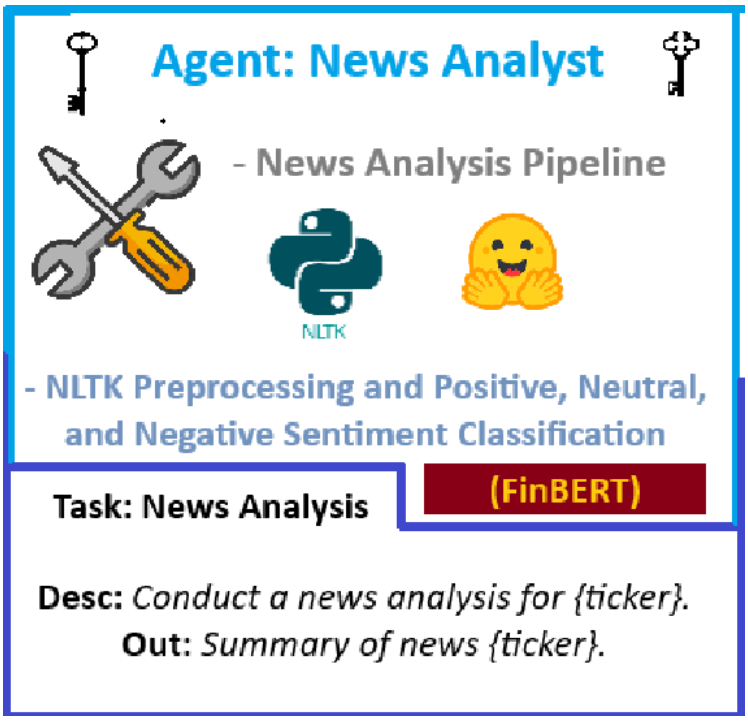

Image 12: News Analysis Agent Block Diagram

```python
# Task for the News Analyst
news_analysis_task = Task(
    description=(
        'Conduct a comprehensive news analysis for the company associated with the ticker {ticker}.'
        'Use your advanced pipeline to determine the overall market sentiment. '
        'The final output should be a summary of the sentiment analysis.'
    ),
    expected_output=(
        'A summary of the recent news sentiment (positive, negative, or neutral) '
        'surrounding the company.'
    ),
    agent=None
)
```

Image 13: News Analysis Task using CrewAI code

This agent ability to understand news, is done by a customized CrewAI tool that uses a sentiment classification transformer called FinBERT, as well as NLTK preprocessing steps to ensure the sentiment classification transformer are proper. The result? A list of preprocessed articles containing title, link, content, and sentiment of the article found.

```python
@staticmethod
def _preprocess_text(text: str) -> str:
    """Internal method to clean and normalize text."""
    if not text:
        return ""
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Tokenize and remove stopwords
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(text)
    filtered_text = [w for w in word_tokens if not w in stop_words]
    return " ".join(filtered_text)
```

Image 14: Preprocessing text function using NLTK

```python
@staticmethod
def _classify_sentiment(text: str) -> str:
    """Internal method to classify sentiment using FinBERT."""
    if not text:
        return "Neutral"
    try:
        tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert")
        model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert")

        inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512)
        with torch.no_grad():
            logits = model(**inputs).logits

        scores = {k: v for k, v in zip(model.config.id2label.values(), torch.softmax(logits, dim=0).tolist())}
        # Return the sentiment with the highest score
        return max(scores, key=scores.get)
    except Exception as e:
        # Fallback in case of model error
        return f"Sentiment analysis failed: {e}"
```

Image 15: Transformer sentiment classification of articles: Negative, Neutral, Positive.

Both functions, and the news_analysis_pipeline customized CrewAI tool are contained in news_analysis_tools.py, inside the tools folder, for further understanding.

Market Analysis Agent:

```python
from crewai import Agent
from tools.fred_client import get_fred_data

# Create a Market Analyst agent
market_analyst = Agent(
    role='Macroeconomic Analyst',
    goal='Provide macroeconomic context for the stock analysis.',
    backstory=(
        'An economist with expertise in tracking and interpreting broad economic indicators. '
        'Your insights help frame the company\'s performance within the larger economic picture.'
    ),
    tools=[get_fred_data],
    verbose=True,
    allow_delegation=False
)
```

Image 16: Market Analysis Agent using CrewAI code.

Well, we have the financial health summary of the ticker company, as well as the news of the ticker company. What's missing now? In this case, the macroeconomic trends help understand the ticker company's performance within the macro-economy of the world.

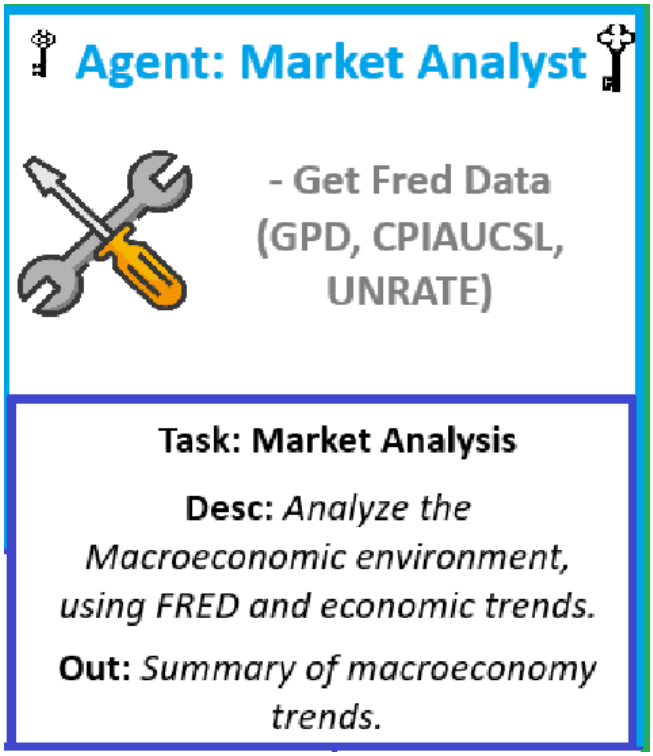

Image 17: Market Analysis Block Diagram of Tasks, and more.

```python
# Task for the Market Analyst
market_analysis_task = Task(
    description=(
        'Analyze the current macroeconomic environment. Fetch the latest data for '
        'Gross Domestic Product (GDP) and the Consumer Price Index (CPIAUCSL). '
        'Summarize the current economic trends and their potential impact on the stock market '
        'and the company with ticker {ticker}.'
    ),
    expected_output=(
        'A summary of the current macroeconomic trends (e.g., inflation, economic growth) '
        'and a brief analysis of their potential impact on the company.'
    ),
    agent=None
)
```

Image 18: CrewAI Task to perform market analysis.

To perform this task, it's important to create a Federal Reserve of St. Louis API key, added to the .env file, which would be used to access the economic database to retrieve the GPD, Consumer Price Index for All Urban Consumers, and the Civilian Unemployment Rate. These are all added to a customized CrewAI tool that would be used by the Market Analyst Agent.

```python
# tools/analysis_tools.py
from crewai.tools import tool
from fredapi import Fred

# --- Tool for FRED Economic Data (for MarketAnalyst) ---
@tool("FRED Economic Data Tool")
def get_fred_data(series_id: str, limit: int = 5) -> str:
    """
    A tool to fetch economic data from the FRED API for a given series ID.
    Common series IDs include:
    - 'GDP': Gross Domestic Product
    - 'CPIAUCSL': Consumer Price Index for All Urban Consumers
    - 'UNRATE': Civilian Unemployment Rate
    """
    try:
        # The Fred class automatically looks for the 'FRED_API_KEY' environment variable.
        fred = Fred()
        data = fred.get_series(series_id).tail(limit)
        return data.to_string()
    except Exception as e:
        # The error message from the library is very descriptive, so we'll return it.
        return f"Error fetching FRED data: {e}"
```

Image 19: CrewAI customized tool to obtain the Federal Reserve St. Louis Data.

```python
# # agents/financial_agents.py

from crewai import Agent
# Create an Investment Advisor agent
investment_advisor = Agent(
  role='Investment Advisor',
  goal='Synthesize all analyses to produce a draft investment recommendation report.',
  backstory=(
    'A seasoned financial advisor who combines quantitative data, news sentiment, and macroeconomic '
    'trends to formulate a preliminary investment recommendation.'
  ),
  verbose=True,
  allow_delegation=False
)
```

Image 20: CrewAI Agent to perform investment advisor draft

Once we have the Earnings, News, and Market summaries of the ticker company; coming from the earnings, news, and market analyst agents; the next step is to pass these as context for the advisor agent to create a draft investment report. This agent has the goal to synthesize financial statements, news sentiments, macroeconomic, and past logs, in order to create a DRAFT investment recommendation to either buy, hold, or sell the desired ticker stock company.

Image 21: Block Diagram for Investment CrewAI agent

```python
# Task for the Investment Advisor (Draft Generation)
advisory_draft_task = Task(
  description=(
    'Synthesize the financial statement analysis, news sentiment analysis, and macroeconomic '
    'context. Based on all this information, formulate a DRAFT investment recommendation '
    '(Buy, Hold, or Sell) for {ticker}. Provide a detailed justification for your recommendation, '
    'referencing specific data points from all three analyses.'
  ),
  expected_output=(
    'A draft investment report with a clear recommendation (Buy, Hold, or Sell) '
    'and a detailed rationale that integrates insights from the financial statements, '
    'news sentiment, and macroeconomic environment.'
  ),
  agent=None,
  context=[earnings_analysis_task, news_analysis_task, market_analysis_task] # Corrected
)
```

Image 22: CrewAI Task for Advisory Investment Agent

The output would be a DRAFT report, which would be passed on to the final critic agent.

Critic Agent:

src > aai-520-final-project > src > agents > critic_agent.py > ...

```python
from crewai import Agent
# Create the Critic Agent
critic_agent = Agent(
    role='Quality Assurance Critic',
    goal='Critique and refine the investment report to ensure it is comprehensive, accurate, and actionable.',
    backstory=(
        'A highly experienced investment strategist with a keen eye for detail. You are tasked with '
        'reviewing financial reports to identify any gaps, logical inconsistencies, or unsupported claims. '
        'Your feedback is crucial for producing a final, high-quality investment recommendation.'
    ),
    verbose=True,
    allow_delegation=False
)
```

Image 23: CrewAI Critic Agent

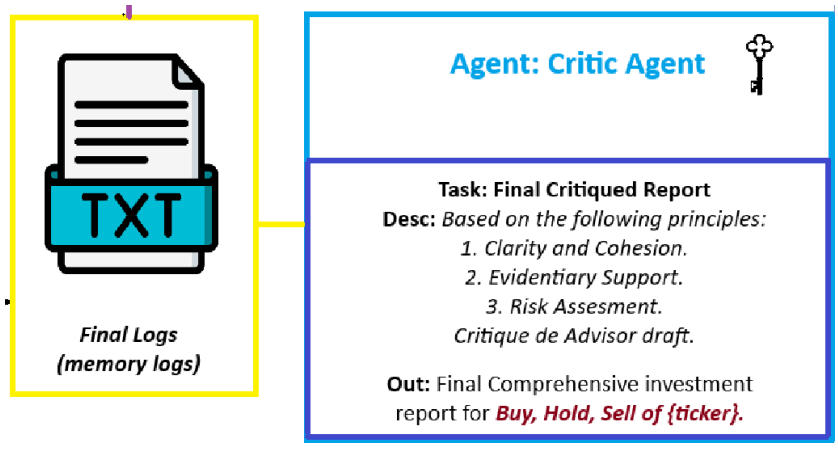

Image 24: CrewAI Critic Agent Flow Diagram

```python
# Task for the Critic Agent (Final Report Generation)
report_critique_task = Task(
    description=(
        'Review the DRAFT investment report provided in the context. Your role is to act as a '
        'skeptical quality assurance analyst. Critique the report based on the following principles:\n'
        '1. Clarity and Cohesion: Is the main investment thesis clear and easy to understand?\n'
        '2. Evidentiary Support: Are all claims backed by specific data points from the analyses?\n'
        '3. Risk Assessment: Does the report adequately identify and discuss the primary risks?\n\n'
        'After your critique, produce a FINAL, polished investment report that incorporates your feedback '
        'and provides a definitive recommendation for {ticker}.'
    ),
    expected_output=(
        'A final, comprehensive, and polished investment report with a clear recommendation '
        '(Buy, Hold, or Sell) and a detailed, well-supported rationale that has been '
        'critically reviewed and refined.'
    ),
    agent=None,
    context=[advisory_draft_task] # Corrected
)
```

Image 25: CrewAI Task for Critique Agent

Finally, once we have the final draft of the financial summary, the last step is to create a critique agent that would assess its content, clarity and cohesion, evidence, and risk assessment, to create a FINAL polished investment report for either sell, hold, or buy the desired ticker stock. This FINAL report is saved inside memory_log.txt file, as shown in Image 4, using the following code inside orchestrator.py:

```python
# Kick off the crew's work
final_report = financial_crew.kickoff(inputs={'ticker': ticker})

print("\n\n########################")
print("## Final Analysis Report:")
print("########################")
print(final_report)

# Manually save the final report to the memory log
print("\n\n########################")
print("## Saving Report to Memory...")
print("########################")
save_status = save_memory.run(report=final_report, ticker=ticker)
print(save_status)
```

Image 26: Final Code to save FINAL Investment report

Discussion and Final Conclusion:

Although we have created a functional multi-financial-agent system, there is still room for improvement in order to improve the quality of the investment report generated. These are:

- **News Analyst Agent:** The news_analysis_pipeline performed efficiently, leveraging NLP preprocessing using nltk and FinBERT, a transformer-based sentiment classifier optimized for financial text analysis. To ensure concise context handling, the FinBERT model was applied to preprocessed content previews limited to 200 tokens, balancing performance and interpretability.
- **Web Search Integration:** In addition to the current news_analysis_pipeline (powered by SerperDev), other web search tools can be integrated in future iterations to broaden data sources and improve retrieval robustness.
- **LLM Configuration:** Since the llm argument was not explicitly defined in agent initialization, all agents currently default to GPT-4, which still delivers outstanding output quality. This setup provides consistency while ensuring high reasoning capability across all tasks.
- **Future LLM Customization:** A configuration file could be introduced to allow dynamic assignment of specific LLMs per agent, enabling fine-tuned control over cost, performance, and domain expertise—potentially integrating frameworks like Ollama for hosting open-source local models.
- **Agent Design and Logic:** The backstories of the agents can be further expanded to enrich contextual reasoning and improve task alignment. The task logic and pipeline structure are solid and well-defined, effectively supporting the crew's overall workflow.
- **Execution Process:** The crew currently operates under Process.Sequential, which ensures task clarity and reproducibility within time constraints. However, transitioning to concurrent or hierarchical processing could enhance scalability and efficiency.
- **API Key and Hosting Considerations:** The current implementation is enclosed with the OPENAI_API_KEY, which provides reliable hosted inference. Future extensions could include an optional configuration for self-hosted or hybrid deployment, leveraging open-source LLMs and local inference environments for greater flexibility.

With all of these future implementations, we can certainly create a development-level product, able to provide proof of concept of proper financial analysis of markets using agentic AI. Of course, the ability to use other modules, like n8n, ChatGPT Agentic AI, NVIDIA agentic Kit, or langraph would highly depend on the user constraints, as well as required latency and functional objectives.

References:

1. Vahdat, A. (2025). *Ironwood: The first Google TPU for the age of inference*. Google Cloud Blog. https://blog.google/products/google-cloud/ironwood-tpu-age-of-inference
2. NVIDIA. (2025). *NVIDIA launches family of open reasoning AI models for developers and enterprises to build agentic AI platforms*. NVIDIA Newsroom. https://nvidianews.nvidia.com/news/nvidia-launches-family-of-open-reasoning-ai-models-for-developers-and-enterprises-to-build-agentic-ai-platforms
3. CrewAI. (2025). *LLMs*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/concepts/llms#openai CrewAI Documentation
4. CrewAI. (2025). *Agents*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/concepts/agents CrewAI Documentation
5. CrewAI. (2025). *Tasks*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/concepts/tasks CrewAI Documentation
6. CrewAI. (2025). *Crews*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/concepts/crews CrewAI Documentation
7. CrewAI. (2025). *Processes*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/concepts/processes CrewAI Documentation
8. CrewAI. (2025). *Tools*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/concepts/tools CrewAI Documentation
9. CrewAI. (2025). *Create custom tools*. In *CrewAI Documentation*. Retrieved October 15, 2025, from https://docs.crewai.com/en/learn/create-custom-tools CrewAI Documentation

Thank you for your time, patience, and teaching professors at the University of San Diego.