Financial Analysis: Multi-Agent with Open Source LLMs Using CrewAI and Ollama Models

Anoop Maurya · Follow

Published in Generative AI

19 min read · Oct 17, 2024

Photo by Marvin Meyer on Unsplash

Stuck behind a paywall? Read for Free!

Imagine John, a financial analyst who's always felt overwhelmed by the sheer volume of data needed to make sound investment decisions. In today's fast-paced financial world, analysts like him don't just need to look at numbers — they must understand market sentiment, analyze competitor performance, and predict risks, all while crafting an investment strategy that balances risk and reward. Doing this manually is no small task. But what if John could delegate these tasks to specialized agents, powered by AI, working together seamlessly? This is the world of multi-agent systems using CrewAI, integrated with Ollama's open-source Large Language Models (LLMs).

In this article, we'll walk through how John transformed his workflow by using CrewAI to build a multi-agent system that automated and enhanced his financial analysis tasks. Specifically, we'll explore the roles of various agents, each designed to focus on a specific part of the analysis, ultimately providing John with a

comprehensive investment strategy.

The Power of Multi-Agent Systems

At the core of this transformation lies the concept of **multi-agent systems**. Instead of a single AI performing all tasks, a multi-agent system breaks down the workflow into specialized components. In the context of financial analysis, this means you can assign specific tasks to different AI agents, each powered by Ollama's LLMs.

Each agent in the system is designed to perform a specific role, leveraging tools and models tailored for that task. These agents communicate with each other, sharing insights, cross-verifying data, and collaboratively reaching conclusions. CrewAI, the framework used here, provides the architecture that allows these agents to work together efficiently.

If you like the article and want to show some love:

- **Clap** 50 times — each one helps more than you think! 👏
- **Follow** me here on Medium and subscribe for free to catch my latest posts. 🫶
- Let's connect on **LinkedIn**, check out my projects on **GitHub**, and stay in touch on **Twitter**!

The Agents: Specialists in Financial Analysis

Let's break down the roles of the agents in John's financial analysis system. Each agent is not just an automated tool but rather a "virtual professional" with expertise in a specific area of finance:

1. **Stock Market Researcher**:
   This agent focuses on gathering detailed data about the stock market. Its goal is to analyze various metrics, from technical chart patterns to financial ratios, all while keeping an eye on competitors in the market. This agent uses advanced

analysis tools to uncover hidden trends that might not be immediately obvious.

2. **Financial Analyst:**
   Acting as the brain behind investment insights, this agent takes the raw data gathered by the Stock Market Researcher and synthesizes it. It provides detailed investment insights, including risk assessments, based on both technical and fundamental analysis. It is responsible for spotting potential risks and making sure all data is coherent and actionable.

3. **Sentiment Analyst:**
   In today's financial world, numbers alone aren't enough. Sentiment — the emotions and opinions driving the market — plays a huge role. The Sentiment Analyst agent focuses on gauging the overall mood of the market, using data from news sources and social media platforms to understand how public perception might affect the stock's future performance.

4. **Investment Strategist:**
   After gathering research data, performing risk assessments, and analyzing sentiment, the final step is to create a comprehensive investment strategy. The Investment Strategist agent takes into account all the gathered information and develops tailored strategies for different types of investors. It balances risk with reward and creates a game plan for how John (and his firm) should proceed with investments in the stock.

How CrewAI Powers the Collaboration

The CrewAI framework acts as the glue that binds these agents together. It orchestrates the process in a **sequential** manner, ensuring that each task builds on the results of the previous one. Here's how the collaboration flows:

1. **Research Task:**
   The first step in the process is to gather relevant data. The Stock Market Researcher agent performs technical and fundamental analysis on a given stock. This involves looking at stock prices, revenue growth, profit margins, and competitor performance. It delivers a comprehensive summary, which is then passed on to the next agent.

2. **Sentiment Task:**
   Once the data has been gathered, the Sentiment Analyst agent steps in to understand how the market feels about the stock. It examines news stories, social media trends, and expert opinions to determine the current sentiment around the company. This emotional context is essential for understanding potential market movements.

3. **Analysis Task:**
   The Financial Analyst agent combines the hard data from the Stock Market Researcher and the sentiment insights from the Sentiment Analyst. It performs a risk assessment, factoring in both numerical data and market sentiment, to provide a thorough analysis of the stock's potential. The outcome is a complete picture of the stock's performance and associated risks.

4. **Strategy Task:**
   Finally, the Investment Strategist agent takes all the gathered information and develops an investment strategy. It considers various investor profiles — some looking for high-risk, high-reward opportunities, while others may seek more conservative, stable investments. The strategy is adaptable, offering actionable insights for each type of investor.

NOW LET's BUILD IT:

STEP 1- TOOLS:

- `tools/comptiter_analysis.py`

```
import yfinance as yf
from crewai_tools import tool

@tool
def competitor_analysis(ticker: str, num_competitors: int = 3):
    """
    Perform competitor analysis for a given stock.
```

```
    Args:
        ticker (str): The stock ticker symbol.
        num_competitors (int): Number of top competitors to analyze.

    Returns:
        dict: Competitor analysis results.
    """
    stock = yf.Ticker(ticker)
    info = stock.info
    sector = info.get('sector')
    industry = info.get('industry')

    # Get competitors in the same industry
    industry_stocks = yf.Ticker(f"^{sector}").info.get('components', [])
    competitors = [comp for comp in industry_stocks if comp != ticker][:num_com

    competitor_data = []
    for comp in competitors:
        comp_stock = yf.Ticker(comp)
        comp_info = comp_stock.info
        competitor_data.append({
            "ticker": comp,
            "name": comp_info.get('longName'),
            "market_cap": comp_info.get('marketCap'),
            "pe_ratio": comp_info.get('trailingPE'),
            "revenue_growth": comp_info.get('revenueGrowth'),
            "profit_margins": comp_info.get('profitMargins')
        })

    return {
        "main_stock": ticker,
        "industry": industry,
        "competitors": competitor_data
    }
```

The `competitor_analysis` function is designed to perform a comparative analysis of a stock and its competitors within the same industry using Yahoo Finance data. Here's a complete breakdown of how it works:

Overview:

This function takes a stock's ticker symbol and the number of competitors to analyze as input. It then identifies competitors within the same industry, fetches financial

data for both the target stock and its competitors, and returns key metrics for comparison, such as market capitalization, P/E ratio, revenue growth, and profit margins. This analysis is valuable for understanding how the target stock performs relative to its peers, which can provide deeper insights for investment decisions.

Step-by-Step Breakdown:

1. **Importing the `yfinance` Library**

> *The function imports `yfinance`, which is used to fetch real-time stock and financial data directly from Yahoo Finance.*

2. Using the `@tool` Decorator

> *The `@tool` decorator indicates that this function will be used as a tool by one of the agents (likely the Stock Market Researcher agent) in the CrewAI framework. This allows the agent to perform competitor analysis as part of its role.*

3.Function Arguments:

> *The function takes two arguments:*
>
> *`ticker` (e.g., 'AAPL' for Apple) which specifies the stock that the agent is analyzing.*
>
> *`num_competitors`, an integer representing how many competitors the agent should include in the analysis. This defaults to 3 but can be changed depending on the depth of the analysis.*

4. Fetching Stock Information:

> *First, the function uses `yfinance` to create a `Ticker` object for the specified stock.*
>
> *The `.info` attribute is used to extract detailed information about the stock.*
>
> *It then pulls out the **sector** and **industry** to identify the broader market context in which the stock operates. This is important because it helps identify competitors operating in the same space.*

5. **Identifying Competitors:**

> *The function fetches other stocks in the same industry by creating a new `Ticker` object for the stock's sector. It uses the sector's financial index (by prepending `^` to the sector name) to get a list of stocks within that industry.*
>
> *It filters out the target stock from this list, ensuring only competitors are included.*
>
> *It limits the number of competitors to the specified `num_competitors`.*

6. **Fetching Competitor Data:**

> *This part of the function loops through the identified competitors, fetching each competitor's detailed financial information.*
>
> *For each competitor, the function collects key financial metrics:*
>
> ***Ticker**: The stock symbol of the competitor.*
>
> ***Name**: The full name of the competitor company.*
>
> ***Market Cap**: The total market capitalization (value of the company).*
>
> ***P/E Ratio**: Price-to-Earnings ratio, a key valuation metric.*
>
> ***Revenue Growth**: Growth rate of the company's revenue.*
>
> ***Profit Margins**: Profitability as a percentage of revenue.*

7.Returning the Analysis Results:

> *Finally, the function returns a dictionary containing:*
>
> *The **main stock** (i.e., the target stock's ticker symbol).*
>
> *The **industry** that the stock belongs to.*
>
> *A list of **competitor data**, including the key metrics for each competitor.*

- tools/risk_assessment_tool.py

```
import yfinance as yf
import numpy as np
from scipy import stats
from crewai_tools import tool

@tool
def risk_assessment(ticker: str, benchmark: str = "^GSPC", period: str = "5y"):
    """
    Perform risk assessment for a given stock.

    Args:
        ticker (str): The stock ticker symbol.
        benchmark (str): Benchmark index for comparison (default: S&P 500).
        period (str): Time period for analysis.

    Returns:
        dict: Risk assessment results.
    """
    stock = yf.Ticker(ticker)
    benchmark_index = yf.Ticker(benchmark)

    stock_data = stock.history(period=period)['Close']
    benchmark_data = benchmark_index.history(period=period)['Close']

    # Calculate returns
    stock_returns = stock_data.pct_change().dropna()
    benchmark_returns = benchmark_data.pct_change().dropna()

    # Calculate beta
    covariance = np.cov(stock_returns, benchmark_returns)[0][1]
    benchmark_variance = np.var(benchmark_returns)
    beta = covariance / benchmark_variance

    # Calculate Sharpe ratio
    risk_free_rate = 0.02  # Assume 2% risk-free rate
    excess_returns = stock_returns - risk_free_rate
    sharpe_ratio = np.sqrt(252) * excess_returns.mean() / excess_returns.std()

    # Calculate Value at Risk (VaR)
    var_95 = np.percentile(stock_returns, 5)
```

```
    # Calculate Maximum Drawdown
    cumulative_returns = (1 + stock_returns).cumprod()
    max_drawdown = (cumulative_returns.cummax() - cumulative_returns).max()

    return {
        "ticker": ticker,
        "beta": beta,
        "sharpe_ratio": sharpe_ratio,
        "value_at_risk_95": var_95,
        "max_drawdown": max_drawdown,
        "volatility": stock_returns.std() * np.sqrt(252)
    }
```

The `risk_assessment` function is designed to evaluate the financial risk associated with a particular stock by comparing it against a benchmark index (such as the S&P 500). This analysis provides key risk metrics such as **Beta**, **Sharpe Ratio**, **Value at Risk (VaR)**, **Maximum Drawdown**, and **Volatility**. Here's a detailed explanation of how the function works:

Step-by-Step Breakdown:

1. **Importing Required Libraries**

> *`yfinance` : Used to fetch historical stock and benchmark data from Yahoo Finance.*
>
> *`numpy (np)` : Utilized for mathematical calculations such as covariance and variance.*
>
> *`scipy.stats` : Used for advanced statistical computations (though in this case, not directly used).*

2. Using the `@tool` Decorator

> *The `@tool` decorator indicates that this function is a tool that can be used by one of the agents (such as the **Financial Analyst agent**) in the CrewAI framework to perform risk assessments as part of its analysis process.*

3. Function Arguments:

> `ticker` *(str): The stock symbol for which the risk is being assessed (e.g., 'AAPL' for Apple).*
>
> `benchmark` *(str): A stock market index for comparison (default is* `^GSPC` *, representing the S&P 500).*
>
> `period` *(str): The time period for historical data, defaulting to "5y" (five years).*

4. Fetching Stock and Benchmark Data:

> *The function retrieves historical closing price data for both the stock (* `ticker` *) and the benchmark index (* `benchmark` *).*
>
> *It uses the* `.history()` *method from* `yfinance` *to fetch data for the specified period (default: 5 years), focusing on the 'Close' price, which reflects the end-of-day price for both stock and index.*

5. **Calculating Daily Returns:**

> *The function calculates* ***percentage returns*** *for both the stock and the benchmark by using the* `.pct_change()` *method, which computes the daily return as the percentage change in the stock's price compared to the previous day.*
>
> `.dropna()` *ensures that any missing data is removed.*

6. **Calculating Beta:**

> ***Beta*** *measures the volatility of the stock relative to the benchmark. It is calculated using the covariance between the stock's returns and the benchmark's returns divided by the variance of the benchmark returns.*
>
> *If* ***Beta > 1****, the stock is more volatile than the benchmark. If* ***Beta < 1****, the stock is less volatile.*

7. **Calculating Sharpe Ratio:**

> *The* ***Sharpe Ratio*** *is a measure of risk-adjusted return, showing how much excess return is*

> *generated for each unit of risk. It compares the stock's returns above the risk-free rate (assumed to be 2%) to the volatility of the stock.*
>
> *The multiplication by* `np.sqrt(252)` *annualizes the Sharpe Ratio, assuming 252 trading days in a year.*

8. Calculating Value at Risk (VaR):

> ***Value at Risk (VaR)*** *estimates the potential loss a stock might face in a given period under normal market conditions, at a 95% confidence level. It uses the 5th percentile of daily returns to calculate the worst expected loss over a day with 95% confidence.*

9. Calculating Maximum Drawdown:

> ***Maximum Drawdown*** *is the largest drop in the stock's value from a peak to a trough over the period analyzed. It helps investors understand the stock's worst possible performance during a downturn.*
>
> *The function calculates cumulative returns and then identifies the maximum difference between the highest and lowest points.*

10. Calculating Volatility:

> ***Volatility*** *measures the standard deviation of the stock's daily returns, indicating how much the stock's price fluctuates. The function annualizes volatility by multiplying the standard deviation by the square root of 252 trading days in a year.*

11. Returning the Risk Metrics:

> *The function returns a dictionary containing all the calculated risk metrics:* ***Beta, Sharpe Ratio, Value at Risk (VaR),*** ***Maximum Drawdown****, and* ***Volatility****. This provides a comprehensive view of the stock's risk profile.*

- `tools\sentiment_analysis_tool.py`

This `sentiment_analysis` function performs sentiment analysis on recent news articles and social media (simulated) for a given stock. The purpose is to gauge the general mood or sentiment surrounding the stock, which could indicate how the market and investors perceive its future performance. Here's a detailed explanation of the code:

Step-by-Step Breakdown:

Importing Required Libraries:

> `yfinance` *: Used to fetch recent news articles related to the stock using Yahoo Finance.*
>
> `TextBlob` *: A natural language processing library used to perform sentiment analysis on text data (specifically, the news article titles).*
>
> `requests` *and* `BeautifulSoup` *: Though imported, they are not directly used in the code, but could be employed for web scraping if needed.*

Using the `@tool` Decorator:

> *The* `@tool` *decorator indicates that this function can be used by a CrewAI agent, such as a* ***Market Sentiment Analyst****, for automated analysis of stock sentiment.*

Function Arguments:

> `ticker` *(str): The stock ticker symbol for which sentiment analysis is being performed (e.g., 'AAPL' for Apple).*

Fetching Recent News Articles:

> *The function fetches recent news articles for the stock using the* `.news` *attribute provided by the* `yfinance` *library. This retrieves a list of articles related to the stock, containing details like the article title, source, and link.*

Performing Sentiment Analysis:

> *The function loops through the 5 most recent news articles.*

> *The article's **title** is extracted, and **TextBlob** is used to calculate the sentiment polarity of the title.*
>
> ***Polarity** ranges from -1 (negative sentiment) to +1 (positive sentiment).*
>
> *For each article title, the sentiment is calculated and stored in the `sentiments` list.*

Calculating Average Sentiment:

> *The function calculates the average sentiment of the 5 most recent news articles. If no articles are available, the sentiment is set to 0.*

Simulating Social Media Sentiment:

> *Since actual social media sentiment analysis isn't implemented, the function calls `simulate_social_sentiment()` to generate a random sentiment score for social media. In a real-world scenario, this function would use APIs like Twitter or StockTwits to gather social media sentiment for the stock.*

Simulating Social Media Sentiment (Function):

> *This function simulates social media sentiment by returning a random value between -1 (negative sentiment) and +1 (positive sentiment).*
>
> *In practice, this could be replaced by actual sentiment data from social media APIs.*

Returning Sentiment Results:

The function returns a dictionary containing:

> ***ticker**: The stock symbol.*
>
> ***news_sentiment**: The average sentiment score based on news articles.*
>
> ***social_sentiment**: The simulated sentiment score from social media.*
>
> ***overall_sentiment**: The average of the news and social media sentiments, giving a combined sentiment score for the stock.*

- `tools/yf_fundamental_analysis.py`

This `yf_fundamental_analysis` function performs a **comprehensive fundamental analysis** on a given stock ticker using data from Yahoo Finance (`yfinance`). It retrieves key financial information, including ratios, growth metrics, and more, to assess the financial health and future prospects of the company.

Overview of the Function

The function does the following:

1. **Fetch Stock Information**: Retrieves fundamental data (financials, balance sheet, and cash flow) using `yfinance` .
2. **Calculate Financial Ratios and Growth Metrics**: It computes important ratios such as current ratio, debt-to-equity ratio, return on equity (ROE), return on assets (ROA), revenue and net income growth, and free cash flow.
3. **Handle Missing Data**: If any financial data is missing or unavailable, it handles exceptions and sets the metrics to `None` .
4. **Return Comprehensive Data**: The function compiles all the gathered information and calculated metrics into a dictionary, providing a detailed fundamental analysis.

```
import yfinance as yf
from crewai_tools import tool

@tool
def yf_fundamental_analysis(ticker: str):
    """
    Perform comprehensive fundamental analysis on a given stock ticker.

    Args:
        ticker (str): The stock ticker symbol.

    Returns:
```

```
        dict: Comprehensive fundamental analysis results.
    """
    stock = yf.Ticker(ticker)
    info = stock.info
    financials = stock.financials
    balance_sheet = stock.balance_sheet
    cash_flow = stock.cashflow

    # Calculate additional financial ratios
    try:
        current_ratio = balance_sheet.loc['Total Current Assets'].iloc[-1] / ba
        debt_to_equity = balance_sheet.loc['Total Liabilities'].iloc[-1] / bala
        roe = financials.loc['Net Income'].iloc[-1] / balance_sheet.loc['Total
        roa = financials.loc['Net Income'].iloc[-1] / balance_sheet.loc['Total

        # Calculate growth rates
        revenue_growth = (financials.loc['Total Revenue'].iloc[-1] - financials
        net_income_growth = (financials.loc['Net Income'].iloc[-1] - financials

        # Free Cash Flow calculation
        fcf = cash_flow.loc['Operating Cash Flow'].iloc[-1] - cash_flow.loc['Ca
    except:
        current_ratio = debt_to_equity = roe = roa = revenue_growth = net_incom

    return {
        "ticker": ticker,
        "company_name": info.get('longName'),
        "sector": info.get('sector'),
        "industry": info.get('industry'),
        "market_cap": info.get('marketCap'),
        "pe_ratio": info.get('trailingPE'),
        "forward_pe": info.get('forwardPE'),
        "peg_ratio": info.get('pegRatio'),
        "price_to_book": info.get('priceToBook'),
        "dividend_yield": info.get('dividendYield'),
        "beta": info.get('beta'),
        "52_week_high": info.get('fiftyTwoWeekHigh'),
        "52_week_low": info.get('fiftyTwoWeekLow'),
        "current_ratio": current_ratio,
        "debt_to_equity": debt_to_equity,
        "return_on_equity": roe,
        "return_on_assets": roa,
        "revenue_growth": revenue_growth,
        "net_income_growth": net_income_growth,
        "free_cash_flow": fcf,
        "analyst_recommendation": info.get('recommendationKey'),
        "target_price": info.get('targetMeanPrice')
    }
```

- `tools/yf_tech_analysis_tool.py`

code is designed to perform advanced technical analysis on stock data using **Yahoo Finance** and technical analysis libraries like **`ta`**. It calculates various technical indicators, identifies potential support and resistance levels, and detects common chart patterns like **head and shoulders**, **double top**, and **double bottom**. Let me summarize and explain each part:

1. Stock Data Retrieval with `yfinance`:

- The function `yf_tech_analysis(ticker: str, period: str)` retrieves historical stock data for a given ticker and period using the `yfinance` library.
- The `history` method fetches price data (open, high, low, close, volume) for the specified period.

2. Technical Indicators with `ta`:

- **`add_all_ta_features`** from the `ta` library computes various technical analysis indicators, such as moving averages, RSI, MACD, Bollinger Bands, etc.
- The **`dropna`** function removes rows with missing data, ensuring calculations use clean data.

3. Custom Indicators:

- **Volatility**: Calculated as the standard deviation of the daily returns over the past 20 days, annualized.
- **Momentum**: Measures the price difference between the current close and the close 20 days ago.

4. Support and Resistance Levels:

- Using `find_peaks` from `scipy.signal`, the code identifies local maxima (peaks) as potential **resistance levels** and local minima (troughs) as potential **support levels**.

5. Chart Pattern Detection:

- The code includes pattern recognition functions for detecting **head and shoulders**, **double top**, and **double bottom** patterns:
- `is_head_and_shoulders` : Detects if the price pattern resembles a head and shoulders.
- `is_double_top` : Identifies if two peaks are close enough in price to form a double top.
- `is_double_bottom` : Identifies two troughs that are close in price to form a double bottom.

6. Return Output:

- The function returns a dictionary with:
- Current price
- 50-day and 200-day simple moving averages (SMA)
- RSI (Relative Strength Index)
- MACD (Moving Average Convergence Divergence)
- Bollinger Bands
- Average True Range (ATR)
- Volatility, Momentum, Support and Resistance Levels

Detected chart patterns

STEP-2: Desiging The Crew

```
from crewai import Agent, Task, Crew, Process
from langchain.llms import Ollama
from tools.yf_tech_analysis_tool import yf_tech_analysis
from tools.yf_fundamental_analysis_tool import yf_fundamental_analysis
from tools.sentiment_analysis_tool import sentiment_analysis
from tools.competitor_analysis_tool import competitor_analysis
from tools.risk_assessment_tool import risk_assessment

def create_crew(stock_symbol):
    # Initialize Ollama LLM
    llm = Ollama(model="tinyllama")  # Make sure you have the llama2 model inst

    # Define Agents
    researcher = Agent(
        role='Stock Market Researcher',
        goal='Gather and analyze comprehensive data about the stock',
        backstory="You're an experienced stock market researcher with a keen ey
        tools=[yf_tech_analysis, yf_fundamental_analysis, competitor_analysis],
        llm=llm
    )

    analyst = Agent(
        role='Financial Analyst',
        goal='Analyze the gathered data and provide investment insights',
        backstory="You're a seasoned financial analyst known for your accurate
        tools=[yf_tech_analysis, yf_fundamental_analysis, risk_assessment],
        llm=llm
    )

    sentiment_analyst = Agent(
        role='Sentiment Analyst',
        goal='Analyze market sentiment and its potential impact on the stock',
        backstory="You're an expert in behavioral finance and sentiment analysi
        tools=[sentiment_analysis],
        llm=llm
    )

    strategist = Agent(
        role='Investment Strategist',
        goal='Develop a comprehensive investment strategy based on all availabl
```

```
            backstory="You're a renowned investment strategist known for creating t
            tools=[],
            llm=llm
        )

        # Define Tasks
        research_task = Task(
            description=f"Research {stock_symbol} using advanced technical and fund
            agent=researcher
        )

        sentiment_task = Task(
            description=f"Analyze the market sentiment for {stock_symbol} using new
            agent=sentiment_analyst
        )

        analysis_task = Task(
            description=f"Synthesize the research data and sentiment analysis for {
            agent=analyst
        )

        strategy_task = Task(
            description=f"Based on all the gathered information about {stock_symbol
            agent=strategist
        )

        # Create Crew
        crew = Crew(
            agents=[researcher, sentiment_analyst, analyst, strategist],
            tasks=[research_task, sentiment_task, analysis_task, strategy_task],
            process=Process.sequential
        )

        return crew

    def run_analysis(stock_symbol):
        crew = create_crew(stock_symbol)
        result = crew.kickoff()
        return result
```

Breakdown of the Code:

Agent Initialization:

> *You define four agents, each with a specific focus:*

> ***Researcher:*** *Gathers technical and fundamental data.*
>
> ***Analyst:*** *Synthesizes the data and conducts risk assessments.*
>
> ***Sentiment Analyst:*** *Analyzes market sentiment from news and social media.*
>
> ***Strategist:*** *Develops investment strategies based on the gathered information.*

Tools Used:

> *Each agent has access to relevant tools:*
>
> ***Technical and Fundamental Analysis Tools:*** `yf_tech_analysis` *and* `yf_fundamental_analysis` *.*
>
> ***Competitor Analysis:*** `competitor_analysis` *.*
>
> ***Sentiment Analysis Tool:*** `sentiment_analysis` *.*
>
> ***Risk Assessment Tool:*** `risk_assessment` *.*

Task Definition:

> *You define tasks for each agent, describing what each agent is supposed to do:*
>
> ***Research Task:*** *Involves gathering data and metrics.*
>
> ***Sentiment Task:*** *Evaluates market sentiment.*
>
> ***Analysis Task:*** *Synthesizes information and assesses risks.*
>
> ***Strategy Task:*** *Develops an investment strategy based on all gathered data.*

Creating the Crew:

> *A* `Crew` *object is created with all defined agents and tasks. The crew executes the tasks sequentially.*

Execution:

> *The `run_analysis` function creates the crew and kicks off the analysis process for the specified stock symbol.*

STEP-3: Streamlit And Main Code Flow

```
import streamlit as st
import yfinance as yf
import plotly.graph_objs as go
from crew import run_analysis
import json

def main():
    st.set_page_config(layout="wide")
    st.title("AI-Powered Advanced Stock Analysis")

    # User input
    stock_symbol = st.text_input("Enter stock symbol (e.g., AAPL):", "AAPL")

    if st.button("Analyze Stock"):
        # Run CrewAI analysis
        with st.spinner("Performing comprehensive stock analysis..."):
            result = run_analysis(stock_symbol)

        # Parse the result
        analysis = json.loads(result)

        # Display analysis result
        st.header("AI Analysis Report")

        col1, col2 = st.columns(2)

        with col1:
            st.subheader("Technical Analysis")
            st.write(analysis.get('technical_analysis', 'No technical analysis

            st.subheader("Chart Patterns")
            st.write(analysis.get('chart_patterns', 'No chart patterns identifi

        with col2:
            st.subheader("Fundamental Analysis")
```

```
        st.write(analysis.get('fundamental_analysis', 'No fundamental analy

        st.subheader("Sentiment Analysis")
        st.write(analysis.get('sentiment_analysis', 'No sentiment analysis

    st.subheader("Risk Assessment")
    st.write(analysis.get('risk_assessment', 'No risk assessment available'

    st.subheader("Competitor Analysis")
    st.write(analysis.get('competitor_analysis', 'No competitor analysis av

    st.subheader("Investment Strategy")
    st.write(analysis.get('investment_strategy', 'No investment strategy av

    # Fetch stock data for chart
    stock = yf.Ticker(stock_symbol)
    hist = stock.history(period="1y")

    # Create interactive chart
    fig = go.Figure()
    fig.add_trace(go.Candlestick(x=hist.index,
                                 open=hist['Open'],
                                 high=hist['High'],
                                 low=hist['Low'],
                                 close=hist['Close'],
                                 name='Price'))

    # Add volume bars
    fig.add_trace(go.Bar(x=hist.index, y=hist['Volume'], name='Volume', yax

    # Add moving averages
    fig.add_trace(go.Scatter(x=hist.index, y=hist['Close'].rolling(window=5
    fig.add_trace(go.Scatter(x=hist.index, y=hist['Close'].rolling(window=2

    fig.update_layout(
        title=f"{stock_symbol} Stock Analysis",
        yaxis_title='Price',
        yaxis2=dict(title='Volume', overlaying='y', side='right'),
        xaxis_rangeslider_visible=False
    )

    st.plotly_chart(fig, use_container_width=True)

    # Display key statistics
    st.subheader("Key Statistics")
    info = stock.info
    col1, col2, col3 = st.columns(3)
    with col1:
```

```
                st.metric("Market Cap", f"${info.get('marketCap', 'N/A'):,}")
                st.metric("P/E Ratio", round(info.get('trailingPE', 0), 2))
            with col2:
                st.metric("52 Week High", f"${info.get('fiftyTwoWeekHigh', 0):,.2f}
                st.metric("52 Week Low", f"${info.get('fiftyTwoWeekLow', 0):,.2f}")
            with col3:
                st.metric("Dividend Yield", f"{info.get('dividendYield', 0):.2%}")
                st.metric("Beta", round(info.get('beta', 0), 2))

    if __name__ == "__main__":
        main()
```

Breakdown of the Code:

Streamlit Configuration:

> *Sets the layout to wide and defines the title for the application.*

User Input:

> *A text input field allows users to enter a stock symbol, with a default value of "AAPL."*
>
> *A button triggers the analysis when clicked.*

Running the Analysis:

> *When the "Analyze Stock" button is pressed, a loading spinner is displayed while the analysis runs using the `run_analysis` function.*

Parsing the Results:

> *The analysis results are parsed from JSON format and displayed in the application.*
>
> *Results are divided into sections, such as technical analysis, fundamental analysis, sentiment analysis, risk assessment, competitor analysis, and investment strategy.*

Stock Data Visualization:

> *Fetches historical stock data for the last year using the `yfinance` library.*
>
> *Creates an interactive candlestick chart using `Plotly` to visualize stock prices and volume*

> *over time.*
>
> *Adds moving averages for better trend analysis.*

Displaying Key Statistics:

> *Displays key statistics like market cap, P/E ratio, 52-week high/low, dividend yield, and beta using Streamlit's metric components for a clean presentation.*

The Advantages of Using Open-Source LLMs

One of the most significant benefits of John's multi-agent system is its use of **open-source LLMs from Ollama.** These models are highly customizable, allowing each agent to be tailored specifically for financial tasks. Unlike proprietary AI models that often come with heavy costs and limitations, open-source LLMs are flexible, allowing for continuous improvement without the burden of licensing fees.

Ollama's LLMs are powerful enough to handle a range of financial tasks, from natural language processing (NLP) for sentiment analysis to handling numerical data in financial reports. Fine-tuning these models to work within CrewAI ensures that each agent delivers highly accurate and relevant results.

The Transformation: A Day in John's Life Post-CrewAI

Now, let's bring John back into the picture. Before adopting CrewAI and Ollama models, John's typical workday was filled with repetitive tasks — manually extracting data, reading through endless news articles, and crunching numbers in Excel. It was a grind that left little room for strategic thinking.

After setting up his multi-agent system, everything changed. Now, John simply feeds the stock symbol he's interested in into the system. Within minutes, the agents work their magic:

- The **Stock Market Researcher** provides a detailed breakdown of the stock's financial health and competitor performance.
- The **Sentiment Analyst** delivers insights into the market's mood, highlighting whether the stock is in favor or under scrutiny.
- The **Financial Analyst** ties everything together, offering a clear picture of the stock's risks and potential gains.
- Finally, the **Investment Strategist** crafts a tailored investment strategy, helping John make decisions quickly and confidently.

Instead of manually handling each of these tasks, John now spends his time making high-level decisions and fine-tuning strategies — things that truly require human intuition and expertise.

Conclusion

By integrating CrewAI's multi-agent system with Ollama's open-source LLMs, John completely transformed his approach to financial analysis. The system allows multiple agents to collaborate, each focusing on a specialized task, making the

entire process more efficient, accurate, and comprehensive. For any financial analyst or investment firm, leveraging such a system can mean the difference between getting bogged down by data and making intelligent, well-informed investment decisions.

In the fast-moving world of finance, the ability to delegate tasks to specialized agents that work together is a game-changer. By embracing CrewAI and Ollama models, analysts like John can finally focus on what matters most — crafting insightful, strategic decisions that lead to success.

Additional Resource:

Official Website: https://www.crewai.com/
Official GitHub:https://github.com/joaomdmoura/crewAI
Complete Code:https://github.com/imanoop7/Financial-Analysis--Multi-Agent-Open-Source-LLM
My GitHub: https://github.com/imanoop7
LinkedIn: www.linkedin.com/in/anoop-maurya-908499148
X: https://x.com/imanoop_7

Generative AI

This story is published on Generative AI. Connect with us on LinkedIn and follow Zeniteq to stay in the loop with the latest AI stories.

Subscribe to our newsletter and YouTube channel to stay updated with the latest news and updates on generative AI. Let's shape the future of AI together!

Crew Ai | Ollama | Open Source | Ai Agent | Financial Analysis

Follow

Published in Generative AI

42K Followers · Last published 20 hours ago

All the latest news and updates on the rapidly evolving field of Generative AI space. From cutting-edge research and developments in LLMs, text-to-image generators, to real-world applications, and the impact of generative AI on various industries.

Follow

Written by Anoop Maurya

1.4K Followers · 39 Following

Data Scientist

Responses (5)

Sebastien M. Laignel

What are your thoughts?

Gilad Bar-Ilan
Oct 19, 2024

did you use older version of crewai in purpose? as your tasks don't call for "expected output"

9 1 reply Reply

Swalk
Nov 19, 2024

Liked, followed and commenting for the algo
Send more like this

5 Reply

Xavier Ghyssens
Mar 6

Nice idea, but the tools is no more working, it seems that some dependecies are outdated

Reply

See all responses

More from Anoop Maurya and Generative AI

Anoop Maurya

Ollama-OCR Now Supports PDFs! 🚀

Stuck behind a paywall? Read for Free!

Mar 9 625

In Generative AI by Ritvik Nayak

5 ChatGPT Hacks That You've Never Heard of (But Need to Know)

These Simple ChatGPT Hacks Will Change Your Life — It's Not Clickbait

2.1K 64

In Generative AI by Cezary Gesikowski

Microsoft's AI Midlife Crisis — Why Satya Nadella's Caution Should Terrify Silicon Valley

The company that missed the internet, fumbled mobile, and played catch-up on the cloud now wants us to pump the brakes on AI hype.

Feb 28 720 43

Anoop Maurya

Ollama-OCR: Now Available as a Python Package!

Stuck behind a paywall? Read for Free!

Dec 2, 2024 2.9K 17

See all from Anoop Maurya

See all from Generative AI

Recommended from Medium

Anil Chandra Naidu Matcha

AI Influencer Automation to earn $10k/month for free

Have you seen the model in picture above anytime ? This instagram model makes more than $10000 per month. And what's crazy, it's not even...

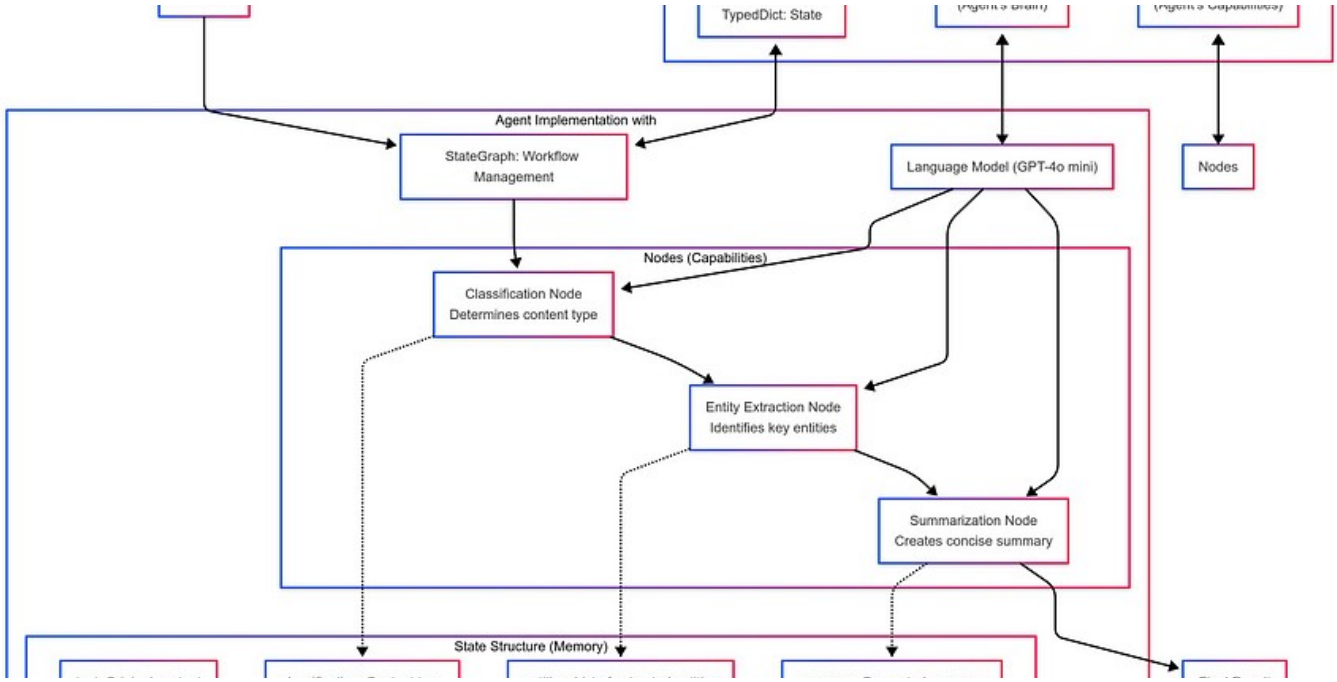
Feb 11 347 8

In Towards AI by Krishan Walia

Google's Gemma-3 Fine-Tuning Made Simple: Create Custom AI Models with Python and Unsloth

Gemma 3 could be the model, you have been looking for. Learn to fine-tune it for all your needs!

Mar 15 325 1

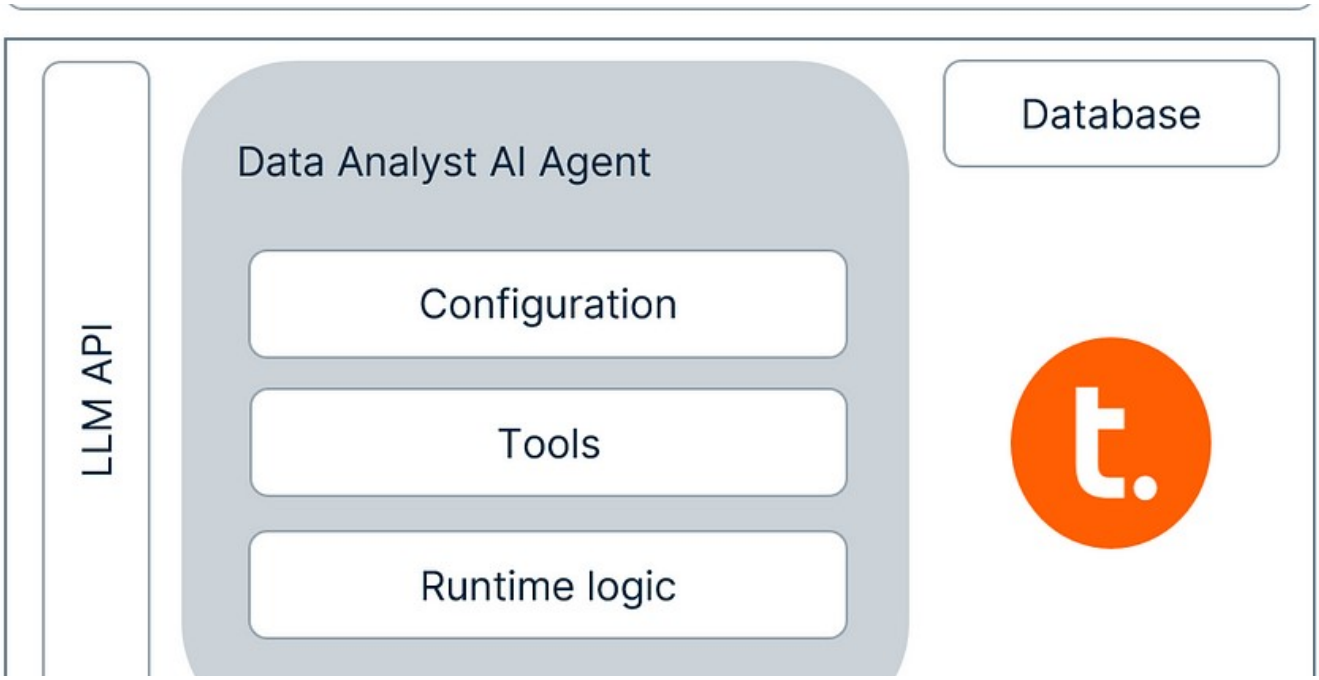


In Data Science Collective by Paolo Perrone

The Complete Guide to Building Your First AI Agent with LangGraph. (It's Easier Than You Think)

Three months into building my first commercial AI agent, everything collapsed during the client demo.

Mar 11 2.2K 50

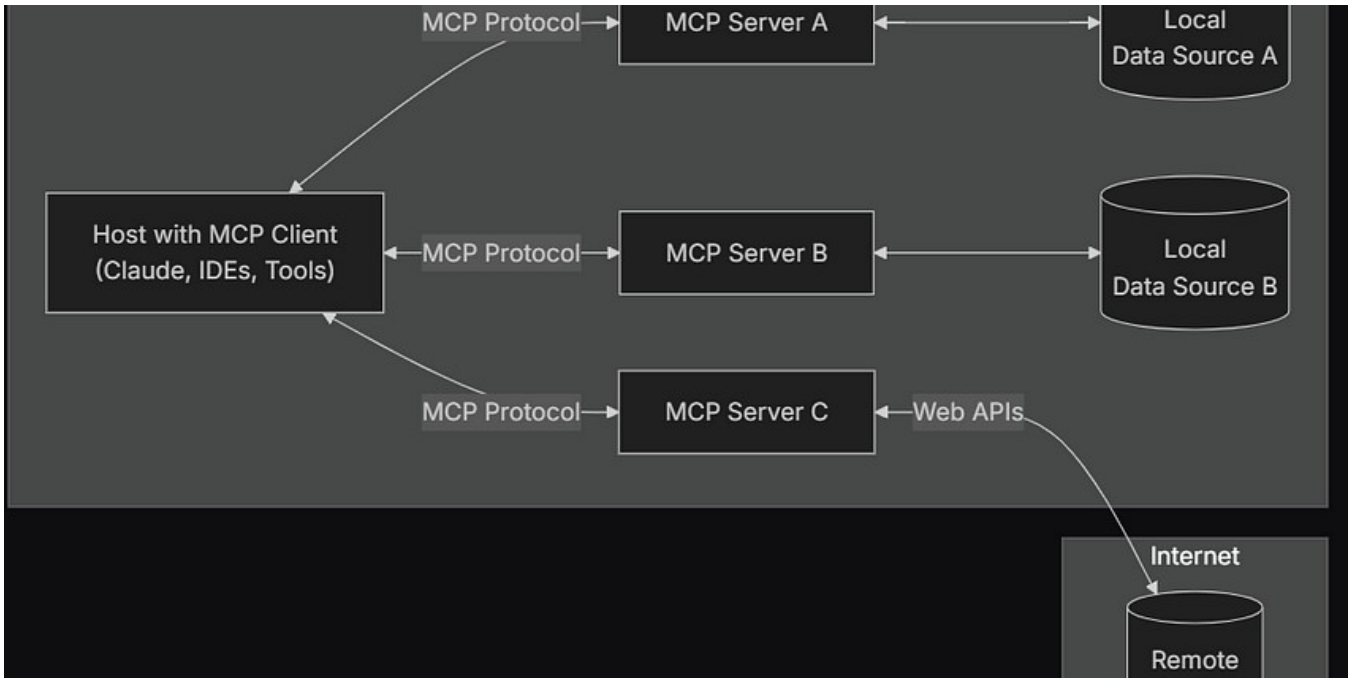


In Teradata by Daniel Herrera

Build a Data Analyst AI Agent from Scratch

As presented in the Open Data Science Conference AI Builders Summit 2025

Feb 7 306 5



In Data And Beyond by TONI RAMCHANDANI

The Model Context Protocol (MCP): The Ultimate Guide

Introduction to the Model Context Protocol (MCP)

178 4

Prabhudev Guntur

Unlocking AI Power: Run OpenManus Without an OpenAI Key!

Introduction

Mar 11 64 4

See more recommendations