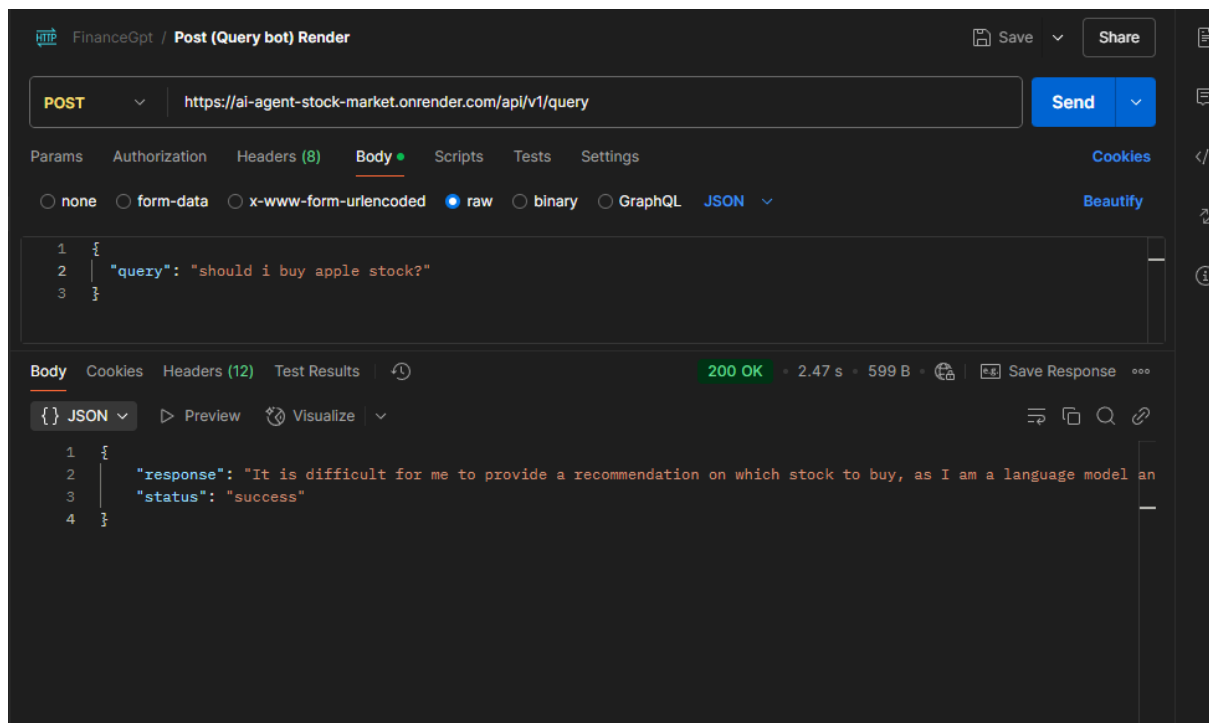


Stock Market AI Agent - Technical Documentation

Introduction and links

Github Repo : [saksham08jain/AI Agent Stock Market: Playing with AI agents for stock market](https://github.com/saksham08jain/AI-Agent-Stock-Market)

How to use deployed service : Hit the api <https://ai-agent-stock-market.onrender.com/api/v1/query>
As shown in example image



Caveats : Render is a free deployment service , server is spinned down with inactivity first hit to server will take a while , though before submitting this I put a new hugging face token , still its a free tier use of Mistral 7B and with ReACT prompt so tokens can be over pretty soon

Resume Link : [PDF SakshamJain_Resume.pdf](#)

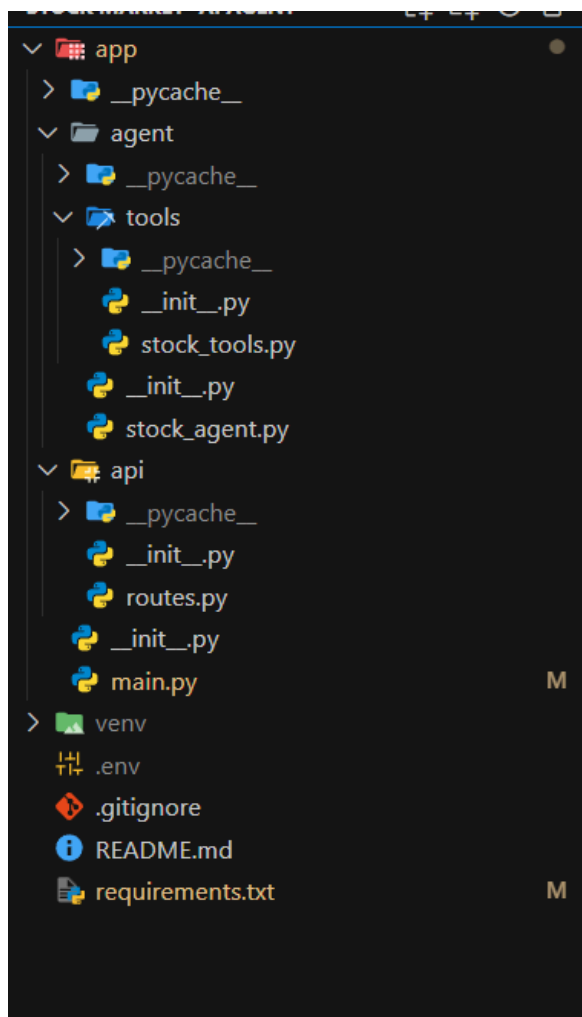
I used help from perplexity , chatgpt and claude for this assignment .

Project Overview

I've developed an AI-powered stock market analysis agent that responds to user queries about stock prices and provides technical analysis using LangChain, Mistral 7B, and Yahoo Finance data. This project demonstrates the integration of LLMs with financial data tools in a practical application

Technical Implementation

Directory Structure



Implementation Details

Stock Tools (stock_tools.py)

I created two main tools using Pydantic and LangChain typing:

1. StockPriceTool

- Uses Yahoo Finance ([yfinance](#)) to fetch stock price data
- Implements async functionality for better performance
- Returns current and historical pricing data based on query parameters

2. StockAnalysisTool

- Combines RSI and moving average to generate buy/sell/hold signals

Implemented custom regex parsing for tool arguments:

This regex extracts ticker and period from a single string input

```
match = re.search(r"ticker=(\w+),?\s*period=(\w\d+)", query)
ticker = match.group(1).upper() # First capture group - the ticker symbol
period = match.group(2)        # Second capture group - the period
```

This regex workaround was necessary because the REACT agent pattern had trouble with multiple separate parameters

AI Agent (stock_agent.py)

I chose to use Mistral 7B via Hugging Face API instead of OpenAI or Claude to avoid paid API costs:

Key Implementation Decisions:

- **ConversationBufferMemory:** Enables multi-turn interactions by maintaining chat history, allowing users to ask follow-up questions without repeating context. This is crucial for natural stock queries like "How has it performed this year?" after discussing a specific stock.
- **max_iterations=5:** Limits the number of reasoning cycles the agent can perform, preventing infinite loops if the LLM gets confused. Five iterations balances thoroughness with response time, allowing multi-step analyses (price check → technical analysis → recommendation) while preventing excessive API usage.
- **handle_parsing_errors=True:** Gives the agent another chance when it makes formatting mistakes with tool inputs rather than failing completely - especially important with Mistral 7B which sometimes produces incorrect parameter formats.

API Layer (api.py)

I chose FastAPI for several technical advantages:

FastAPI Technical Benefits:

- **Async Support:** Native async/await capabilities work seamlessly with my asynchronous stock data fetching functions
- **Performance:** Built on Starlette and Uvicorn for high throughput with minimal overhead
- **Automatic Documentation:** Generates interactive Swagger UI for testing endpoints
- **Dependency Injection:** The `Depends(get_agent)` function implements a singleton pattern, ensuring the agent is initialized only once and reused across requests, preventing redundant model loading

Deployment

I deployed the application on Render's free tier, which presented specific challenges:

- Server spins down after periods of inactivity, causing 30-60 second cold start delays
- Memory management issues with the ConversationBufferMemory in a serverless environment
- Had to replace the Hugging Face API key after deployment for the demo to work

Challenges Encountered

1. Regex Parameter Parsing

Initially tried to implement tools that take multiple parameters, but the REACT prompt framework had issues parsing them correctly. My regex solution extracts multiple parameters from a single string:

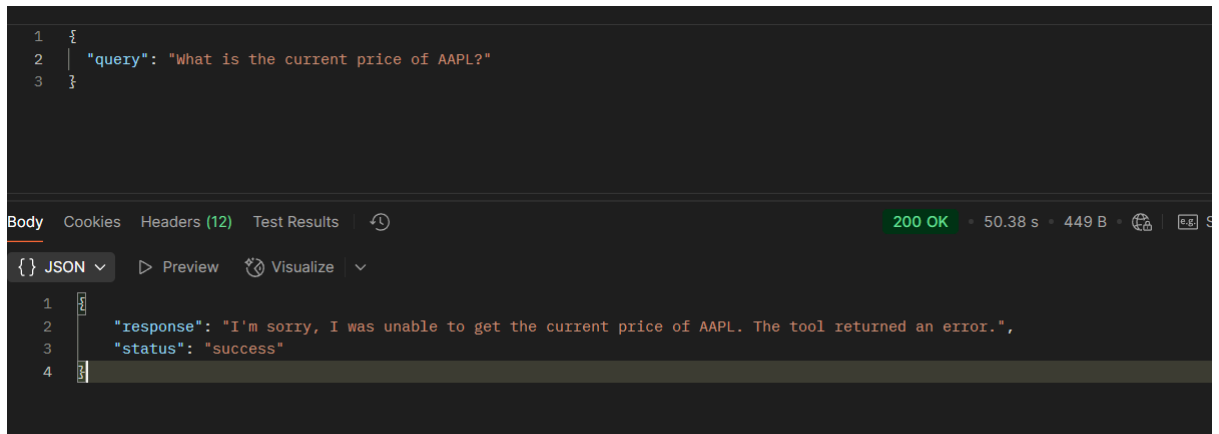
```
re.search(r"ticker=(\w+),?\s*period=(\w\d+)", query)
```

This regex matches inputs like "ticker=AAPL, period=1m" or "ticker=MSFT period=6m", making it flexible for various input formats while working within the constraints of the REACT framework.

2. Ticker Symbol Recognition Issues

Discovered an interesting bug where:

- Using "AAPL" in query would fail



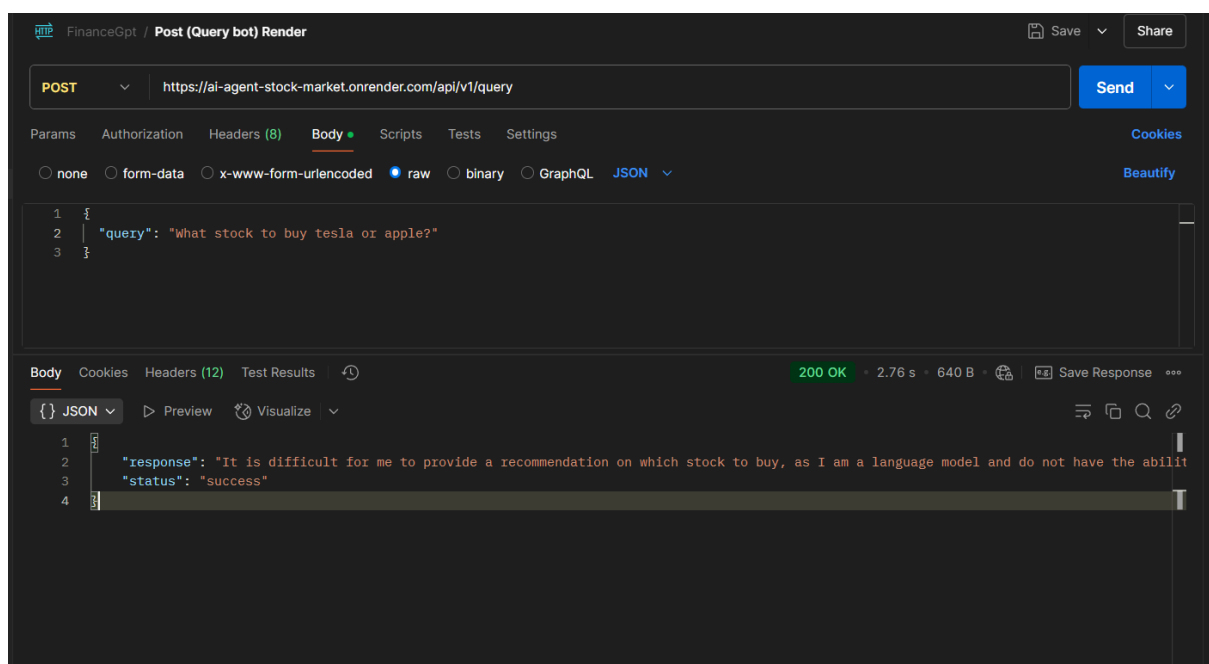
```
1 {
2   "query": "What is the current price of AAPL?"
3 }
```

Body Cookies Headers (12) Test Results 200 OK • 50.38 s • 449 B

{ } JSON Preview Visualize

```
1 {
2   "response": "I'm sorry, I was unable to get the current price of AAPL. The tool returned an error.",
3   "status": "success"
4 }
```

- Using "Apple" in query would work correctly



FinanceGpt / Post (Query bot) Render Save Share

POST https://ai-agent-stock-market.onrender.com/api/v1/query Send

Params Authorization Headers (8) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "query": "What stock to buy tesla or apple?"
3 }
```

Body Cookies Headers (12) Test Results 200 OK • 2.76 s • 640 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "response": "It is difficult for me to provide a recommendation on which stock to buy, as I am a language model and do not have the ability to provide financial advice.",
3   "status": "success"
4 }
```

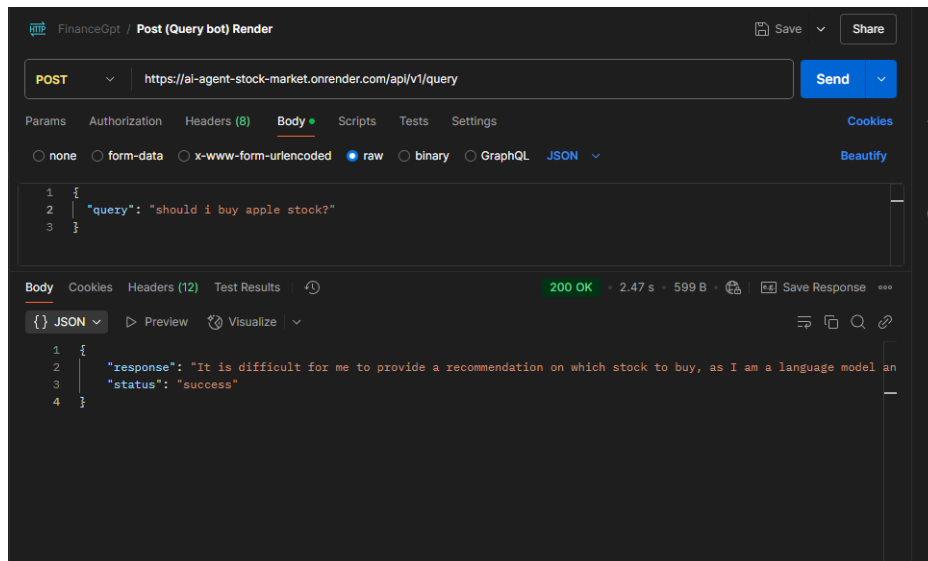
This inconsistency stems from how Mistral 7B parses and formats tool inputs. The model sometimes struggled with exact ticker symbols but worked better with company names, suggesting that more robust symbol normalization would improve reliability.

3. Deployment Inconsistencies

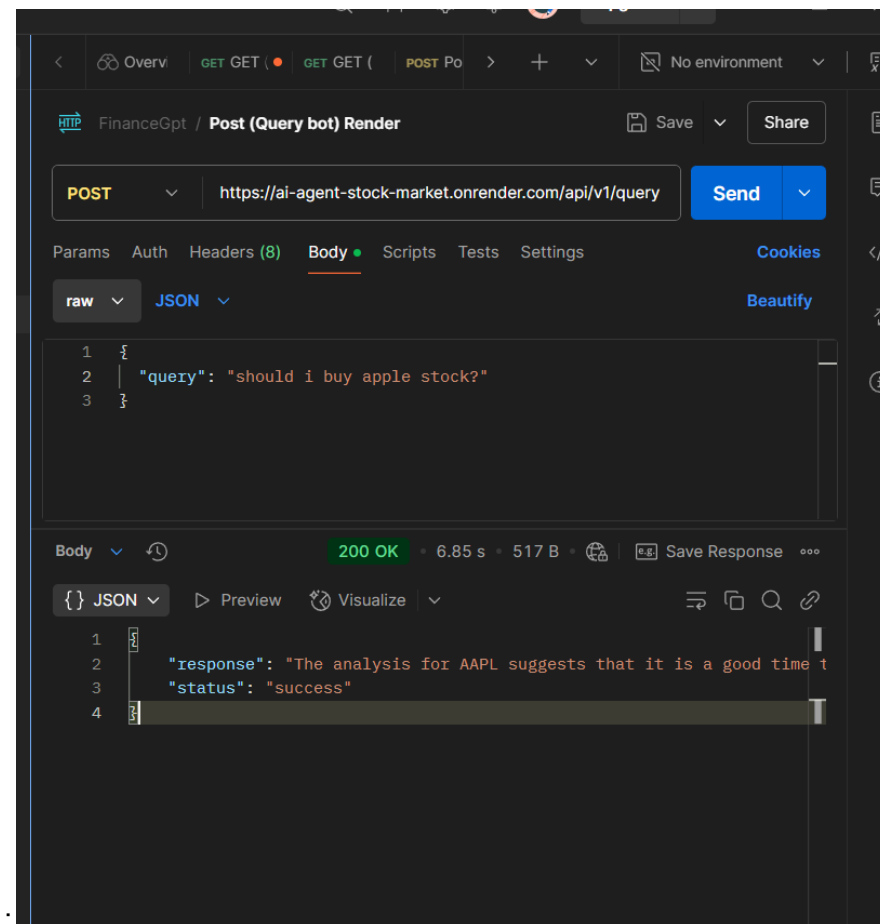
Experienced strange behavior where:

- Same code worked in development but failed on deployment
- Clearing cache and redeploying solved this issue

Before clearing cache :



After:



This was likely related to:

- ConversationBufferMemory persistence issues in serverless environments
- API rate limiting from Hugging Face

- Render memory issues for free tier access
- Potential memory leaks in the LangChain agent over multiple requests

4. LLM Response Quality

Mistral 7B sometimes provides inconsistent responses to similar queries. The temperature setting of 0.2 helped improve consistency but didn't eliminate all quality issues. This represents a tradeoff between using free open-source models versus paid commercial APIs.

Technical Insights

1. **LangChain Agent Architecture:** The REACT prompting pattern works effectively for simple stock queries but has limitations with complex parameter handling.
2. **FastAPI vs. Alternatives:** FastAPI's performance advantages and async support made it ideal for this application compared to Flask or Django, especially considering the asynchronous nature of stock data fetching.
3. **Memory Management:** ConversationBufferMemory works well for maintaining context but creates challenges in serverless environments with cold starts. Alternative memory strategies like database-backed memory might be more suitable for production.
4. **Deployment Architecture:** Free-tier serverless deployments introduce significant constraints for LLM applications, particularly around cold starts and memory persistence.

Future Improvements

1. Switch to a more robust LLM like OpenAI or Claude for production use
2. Implement better parsing of stock symbols with fallback mechanisms (try ticker first, then company name)
3. Add database-backed memory instead of in-memory conversation buffer
4. Add caching layer for stock data to reduce API calls and improve response time
5. Implement proper error handling for various edge cases in stock queries
6. Consider a dedicated server instead of serverless for more consistent performance

Conclusion

I thoroughly enjoyed this project even though i have my doubts about its practical use cases in the finance domain or real world scenarios. Thanks for this opportunity to work on AI agents , the experience itself was worth it.