Quest Journals
Journal of Research in Business and Management
Volume 13 ~ Issue 7 (July 2025) pp: 01-14
ISSN(Online):2347-3002
www.questjournals.org

Research Paper

Financial Advice Agent: AI-driven Cryptocurrency Insights with Lang Graph

[1] Krishna Dev Pandey, [2]Prakhar Shukla, [2] Pranav Sahu
[3] Hardik Sharma, [3] Dr. Bagesh Kumar, [4] Dr. Arun Kumar
[1]*Mewar University, Chittorgarh, Rajasthan, India*
[2]*IIIT Allahabad, Prayagraj, India*
[3]*Manipal University Jaipur, Jaipur, India*
[4]*Feroze Gandhi College, Raebareli*

ABSTRACT: *The process of designing an AI-driven financial advice agent involves integrating natural language processing, sentiment analysis, and predictive analytics to provide users with personalized market insights. In this paper, we present a multi-component pipeline capable of extracting financial tickers, evaluating sentiment from market-related queries, and predicting asset price trends. The system leverages Named Entity Recognition (NER) for accurate ticker identification, fine-tuned senti- ment classification models, and time-series forecasting techniques for financial predictions.*
We conducted an extensive evaluation using real-world financial data, comparing different sentiment analysis models and forecasting algorithms. Performance metrics such as Mean Squared Error (MSE) and Sentiment Accuracy were used to assess prediction reliability and classification precision. Our findings indicate that transformer-based sentiment analysis models outperform traditional methods, while financial forecasting remains challenging due to market volatility.

KEYWORDS: *Financial Advice Agent, Natural Language Processing, Sentiment Analysis, Ticker Extraction , Financial Forecasting*

I. INTRODUCTION

Unprecedented opportunities and challenges have been created for investors due to the rise of cryptocurrency markets. Unlike traditional financial assets, cryptocurrencies are extremely volatile, highly sensitive to market sentiment and influenced by factors such as global economic events, technological advancements, regulatory decisions, and social media trends.

While conventional financial analysis tools have played a critical role in market evaluation, they often struggle to adapt to the fast-paced and dynamic nature of cryptocurrencies. Traditional models rely on historical data and predefined indicators, which may not capture the complexities of digital asset markets. Given these challenges, artificial intelligence (AI) has emerged as a powerful solution to enhance financial analysis by incorporating real-time data processing, predictive modeling, and natural language processing (NLP) for sentiment analysis [3, 7].

Recent research has demonstrated the effectiveness of AI in financial applications, particularly in cryptocurrency trading. Machine learning models have been employed for price forecasting using deep learning and time-series analysis [10], while NLP-based models have been utilized to analyze financial news [2,4]. How- ever, integrating these capabilities into a single, efficient system remains a challenge, as many existing solutions focus on either price prediction or sentiment analysis in isolation [9].

To address these limitations, we introduce the **Financial Advice Agent**, an AI-powered assistant designed to provide real-time cryptocurrency insights by analyzing both market trends and relevant financial news. Our system is built using **LangGraph**, a workflow orchestration framework, enabling efficient data processing and modular integration of various analytical components. The Financial Advice Agent follows a structured pipeline with specialized AI-powered tools:

1. **Ticker Extractor** – Identifies cryptocurrency tickers from user input
2. **Price Retriever** – Fetches real-time cryptocurrency price data
3. **News Retriever** – Gathers relevant news articles
4. **News Analyst** – Processes financial news to extract insights and sentiment
5. **Price Analyst** – Evaluates historical price trends and patterns
6. **Financial Reporter** – Synthesizes information to generate a comprehensive summary

This paper presents the development of the Financial Advice Agent, detail- ing its architecture, data processing, and AI-driven analysis, while comparing its performance with existing models and assessing its impact on investment decision-making.

II. RELATED WORK

Previous research has explored various AI techniques for cryptocurrency market analysis:

Parameswaran et al. [6] integrated Wavelet Transform (WT) and Bi-Directional Long Short-Term Memory (Bi-LSTM) networks for cryptocurrency trend forecasting. The hybrid model outperformed standard LSTM and CNN models by capturing both short-term fluctuations and long-term trends, though perfor- mance was impacted by market volatility.

Zhao et al. [11] investigated Deep Reinforcement Learning (DRL) with LSTM to optimize trading strategies, finding that Proximal Policy Optimization (PPO)- based DRL agents outperformed conventional strategies but faced challenges in interpretability and scalability.

Roumeliotis et al. [8] compared fine-tuned GPT-4, BERT, and FINBERT models for sentiment analysis of cryptocurrency news, finding that GPT-4 per- formed best, though domain-specific fine-tuning restricted performance on unseen data.

Jay et al. [5] introduced stochastic neural networks that outperformed deterministic models by capturing market volatility through layer-wise stochasticity, effectively modeling the unpredictable nature of market behavior.

Enajero [1] assessed AI-driven predictive models for market volatility man- agement, comparing Support Vector Machines (SVM), LSTM, and GARCH models across traditional and cryptocurrency markets, finding superior predictive accuracy in cryptocurrency markets but limited improvement in traditional markets.

III. METHODOLOGY

This document provides a comprehensive methodology and detailed chronological explanation of the Financial Advice Agent, an advanced AI-powered system designed for cryptocurrency analysis. Our system leverages state-of-the-art language models from Groq, integrated within a robust LangGraph framework, to deliver insightful financial advice. We emphasize the synergistic operation of its components, highlighting parallel processing, intelligent batching, and performance optimizations that ensure efficient and accurate financial insights.

1. System Overview: A Chronological Journey of a Query

Upon receiving a user query, the Financial Advice Agent orchestrates a series of sophisticated operations, each designed to process information and contribute to a holistic financial report. This process can be visualized as a journey through distinct yet interconnected nodes within our LangGraph architecture, as depicted in Figure. Each node represents a modular capability—such as entity recognition, data retrieval, or financial reasoning—that collectively drive the system's response generation. The main execution flow is controlled by main.py, which acts as the central engine coordinating interactions between various specialized components defined in utils.py, classes.py, and consts.py. These modules encapsulate utility functions, custom classes, and constant parameters respectively, ensuring modularity and scalability. The user interface is built using Streamlit, providing an interactive front-end that allows users to input queries, view personalized financial insights, and track system responses in real time. This architectural setup ensures both robustness and flexibility, enabling the system to handle a wide range of financial queries efficiently.

1. 1 The Starting Point: User Query and Initialization

Every interaction begins with a user's financial query. This query is captured and becomes part of the application state, managed by the AppState object defined in classes.py. This central data structure is designed to evolve dynamically, tracking all relevant inputs, intermediate computations, and system responses as the query flows through various stages of the LangGraph pipeline. As each node processes and contributes new information—be it extracted entities, fetched prices, or inferred insights—the AppState is updated accordingly, ensuring continuity, coherence, and full context retention across the system's execution. Essential constants, such as cryptocurrency mappings defined in the top_crypto_dict, are loaded from consts.py at the system's initialization, providing a stable reference point for interpreting ticker symbols, exchange data, and other predefined values. This ensures that all modules operate on a shared vocabulary and interpret financial data consistently, reducing ambiguity and enhancing system reliability. This approach allows for seamless traceability of data transformations, making the system transparent and easy to debug.

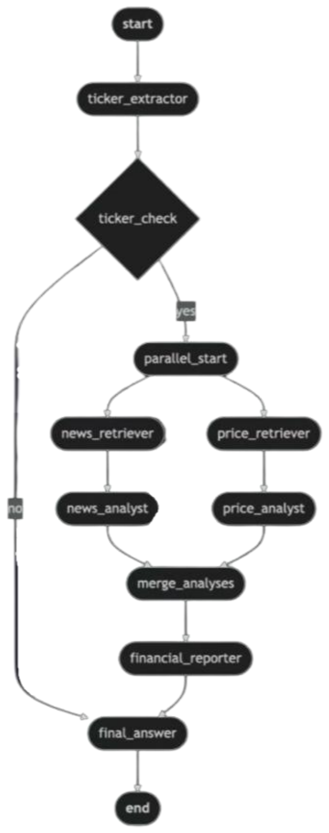

Figure 1 : Financial Advice Agent Workflow

1.2 Phase 1: Understanding the Request (Ticker Extraction)

The first critical step is the **Ticker Extractor** node in our graph. This node is crucial because a user's query might refer to cryptocurrencies indirectly or col- loquially. To address this, we employ Groq's **Llama-4-Maverick-17b-128e- instruct** model, a more specialized and *heavy* Large Language Model (LLM) tuned for precise information extraction, to accurately identify all relevant cryptocurrency ticker symbols. This model is configured with zero temperature (temperature=0) to ensure deterministic and precise outputs, and is specifically instantiated as ticker llm in our system. This process is more than a simple keyword search; it involves a sophisticated understanding of context and indirect references, as demonstrated by the detailed system prompt provided to the LLM. If no specific cryptocurrency is identified, a default NoCointicker is used, allowing the system to proceed with general market analysis. This stage also includes robust validation against our predefined list of top cryptocurrencies (from consts.py) to ensure accuracy.

If the Ticker Extractor successfully identifies one or more cryptocurrencies (a 'yes' branch in the graph), the workflow advances to the data retrieval phase, where historical prices, trends, and relevant news are gathered for further analysis. This branching ensures that the system delivers context-aware insights tailored to the specific assets mentioned in the query. On the other hand, if no relevant tickers are found (the 'no' branch), the system bypasses asset-specific operations and proceeds directly to generating a general financial summary or advisory response. This fallback mechanism guarantees robustness, ensuring that the user still receives a meaningful output even when their query lacks explicit or recognized cryptocurrency references. Such graceful degradation in the pipeline is critical for maintaining a smooth user experience and avoiding dead-ends in conversation flow.This design ensures that even vague or conversational queries are meaningfully interpreted, enhancing the system's accessibility for non-technical users. By leveraging a powerful LLM and combining it with rule-based validation, the Ticker Extractor strikes a balance between linguistic flexibility and financial accuracy. This early-stage intelligence plays a foundational role in maintaining the overall quality and relevance of the downstream analysis.

Algorithm 1 Ticker Extraction Algorithm

```
1:  procedure EXTRACTTICKERS(user query)
2:      Input: user query (string)
3:      Output: List of Ticker enums
4:          system prompt ← detailed instructions for context-aware ticker extrac- tion with examples
5:          response ← ticker llm.invoke([SystemMessage(system prompt), HumanMessage(user query)])
6:      extracted tickers ← parse_JSON from response.content
7:      if thenextracted tickers is empty or parsing fails
8:      extracted tickers ← attempt to find tickers by keyword matching
9:      end if
10:     validated tickers ← filter extracted tickers against Ticker.members
11:     if thenvalidated tickers is empty
12:     validated tickers ← [Ticker["NoCoin"]]
13:     end ifreturn validated tickers
14: end procedure
```

1.3 Phase 2: Data Acquisition (Parallel Retrieval)

With the ticker symbols in hand, the system moves into a parallel data ac- quisition phase, designed for efficiency and speed. This phase begins with a parallel start node that forks the execution into two concurrent operations: **Price Retriever** and **News Retriever**. The use of parallel processing, specif- ically through a ThreadPoolExecutor in main.py, is crucial here. For each identified ticker, both price and news data are fetched concurrently, significantly reducing the overall response time.

1.3.1 Price Retriever

This node, primarily leveraging functions within utils.py, is responsible for fetching historical price data for the extracted tickers. We utilize the **OpenBB SDK** to retrieve comprehensive price information, specifically setting the output type to dataframe for direct integration into our analysis pipeline. The get price data function in utils.py is configured to retrieve data from 2010-01-01 onwards and, notably, is explicitly set to retrieve **weekly** price data (TimeFrame.WEEKLY). Immediately after fetching, the data undergoes extensive technical analysis.

This includes calculating various technical indicators such as the Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands, and key moving averages (50-day and 200-day) using the pandas ta library. A specific data cutoff is applied to the retrieved price data (moving the cutoff point 1 month + 5 days back from the latest available data) to ensure consistent analy- sis window. This processing is batched and optimized to handle multiple tickers efficiently, ensuring that the analytical insights are derived from a rich dataset.

1.3.2 News Retriever

Concurrently with price data, the News Retriever node, also powered by utility functions defined in utils.py, is responsible for fetching relevant and up-to-date news articles related to the user's query. This node initiates a multi-faceted search process using the DuckDuckGo Search API. It begins with a broad search for general cryptocurrency news to capture macroeconomic and market-wide developments. Subsequently, it performs targeted searches for each identified ticker symbol, using its full cryptocurrency name retrieved from the top_crypto_dict for more accurate matching. This dual-layered approach ensures both breadth and specificity in the information collected. The search function is carefully parameterized to limit results to the past month (using timelimit="m") and retrieves a maximum of 100 results per query, maintaining both relevance and performance. Once all search queries are executed, the resulting articles are aggregated, deduplicated based on their URLs to avoid repetition, and sorted in chronological order to preserve temporal context. To improve system efficiency and enable data persistence, the news data for each ticker is saved locally to disk in both CSV and JSON formats under the news_data directory. This caching mechanism not only accelerates response times in future runs by avoiding redundant API calls, but also creates an auditable record of past results for offline inspection or retrospective analysis.

Algorithm 2 Price Data Retrieval and Indicator Calculation

```
1: procedure RETRIEVEANDPROCESSPRICES(tickers)
2:     Input: tickers (List of Ticker enums)
3:     Output: Dictionary of pd.DataFrame for prices with indicators
4:     price data futures ← empty list
5:     executor ← ThreadPoolExecutor()
6:     for each ticker in tickers do
7:         future ← executor.submit(get price data, ticker, TimeFrame.WEEKLY)
           ▷ Explicitly weekly data
8:     price data futures.append((ticker.name, future))
9:     end for
10:    processed prices ← empty dictionary
11:    for ticker name, future in price data futures do
12:    raw df ← future.result()
13:        processed df ← add indicators(raw df)        ▷ Adds RSI, MACD, BBands, MAs
14:    Apply timedelta(weeks=4, days=5) cutoff to processed df
15:    processed prices[ticker name] ← processed df
16:    end forreturn processed prices
17: end procedure
```

1.4. Phase 3: Deep Analysis (Parallel Analysis)

Once the raw data (prices and news) is retrieved, the system again branches into parallel processing to perform in-depth analysis. This involves the **News Analyst** and **Price Analyst** operating concurrently. This parallel analytical approach further enhances the responsiveness of the system, as insights from both domains are generated simultaneously.

1.4.1 News Analyst

Leveraging Groq´s **Llama-3.1-8b-instant** model, which serves as our general- purpose and *lighter* LLM (llm object in main.py), the News Analyst component performs sophisticated natural language processing on the retrieved news articles. This model is also configured with zero temperature for consistent analysis and is instantiated as the primary llm in our system. For each ticker, it generates a sentiment score (ranging from 0 for extremely bearish to 100 for extremely bullish), categorizes the overall sentiment (e.g., GREED, NEUTRAL, FEAR), and extracts 2-3 concise key points from the articles. This node effectively dis- tills large volumes of textual data into actionable sentiment insights, which are encapsulated in the NewsAnalysis Pydantic model defined in classes.py.

1.4.2 Price Analyst

Operating in parallel with other components, the Price Analyst node is tasked with interpreting the comprehensive price data and technical indicators gathered by the Price Retriever. This component performs a detailed examination of historical price trends, including short-term momentum and long-term movements, to extract actionable insights. It identifies key technical patterns such as support and resistance levels, moving averages, and price breakouts, which are essential for understanding market dynamics. Beyond descriptive analysis, the node also generates forward-looking predictions for 1, 2, 3, and 4 weeks into the future, offering users a short-term outlook on price behavior. Each prediction is accompanied by a confidence score, which provides a quantitative measure of the model's certainty and helps users weigh the reliability of the forecast. These results are encapsulated in a structured PriceAnalysis Pydantic model—defined in classes.py—that cleanly organizes the trend outlook, predicted prices, and a natural-language explanation of how the conclusions were reached. This explanatory component is particularly useful for transparency and interpretability, especially for non-expert users. The node also leverages the general-purpose LLM (llama-3.1-8b-instant) to support analytical reasoning, such as interpreting patterns, generating commentary, or articulating justifications for trend shifts. This integration of quantitative and language-based analysis makes the Price Analyst a key contributor to the system's ability to deliver nuanced and explainable financial insights.

Next node in our pipeline is the Price Analyst node.

Algorithm 3 News Data Retrieval and Persistence

```
1:  procedure RETRIEVEANDPERSISTNEWS(tickers)
2:      Input: tickers (List of Ticker enums)
3:      Output: Dictionary of pd.DataFrame for news
4:      news dict ← empty dictionary
5:      for each ticker in tickers do
6:      crypto news df ← search("Cryptocurrency")
7:      ticker news df ← search(top crypto dict[ticker.name])
8:      combined df ← pd.concat([crypto news df, ticker news df])
9:      final df ← combined df.drop duplicates(subset='url').sort values(by='date')
10:     Save final df to news data/{ticker.name} news.csv 11: Save final df to news data/{ticker.name}
news.json 12:     news dict[ticker.name] ← final df
13:     end forreturn news dict
14: end procedure
```

Algorithm 4 News Analysis Algorithm

```
1:  procedure ANALYZENEWS(news data dict)
2:      Input: Dictionary of pd.DataFrame for news 3:      Output : Dictionary of NewsAnalysis reports 4:
        news analysis reports ← empty dictionary
5:      for each ticker name, news df in news data dict.items() do
6:      Concatenate relevant news articles into a single text body
7:              llm response   ←   llm.invoke(prompt for news analysis, news text)
8:      Parse llm response into NewsAnalysis object
9:      news analysis reports[ticker name] ← NewsAnalysis object
10:     end forreturn news analysis reports
11: end procedure
```

Algorithm 5 Price Analysis Algorithm

```
1:  procedure ANALYZEPRICES(prices data dict)
2:      Input: Dictionary of pd.DataFrame for prices with indicators
3:      Output: Dictionary of PriceAnalysis reports
4:      price analysis reports ← empty dictionary
5:      for each ticker name, price df in prices data dict.items() do
6:              Analyze price df for trends, support/resistance, and indicators us- ing llm
7:      Generate future price predictions
8:      Calculate confidence score
9:      Determine trend outlook
10:     price analysis reports[ticker name]      ←
PriceAnalysis(predictions, confidence, trend, explanation)
11:     end forreturn price analysis reports
12: end procedure
```

1.5 Phase 4: Synthesis and Reporting

After news and price analyses are complete, their outputs are merged in the merge_analyses node, followed by the Financial Reporter. This component integrates insights from both analysts to generate a cohesive report. It combines sentiment scores, price predictions, a bullishness score (0–100), a market summary, and an actionable recommendation (BUY, HODL, or SELL). The output is packaged into a FinalReport Pydantic model. This node also uses the general-purpose LLM (llama-3.1-8b-instant) for reasoning and report generation.

1.6 Phase 5: Delivering the Answer (Final Answer Generator)

Finally, the **Final Answer Generator** node takes the comprehensive FinalReport and formats it into a user-friendly response. This involves structuring the mar- ket analysis, recommendations, and summary into a clear and digestible format, ready for presentation to the user, typically via a Streamlit interface. This is the ultimate output of the entire workflow, providing the user with the requested financial insights.

2. Supporting Modules and utilities

Beyond the core LangGraph nodes, several Python files provide foundational support and utility functions for the system:

- consts.py: This file defines global constants, most notably the top crypto dict which maps cryptocurrency ticker symbols to their full names. This ensures consistent and accurate identification of cryptocurrencies throughout the system.
- classes.py: This module is central to maintaining type safety and structured data flow within the application. It defines critical data structures like the AppState (which holds the complete state of a user's query and its processing), TimeFrame enums, and various Pydantic models (e.g., TickerQuery, PriceAnalysis, NewsAnalysis, FinalReport). These models enforce data integrity and facilitate seamless data exchange between different nodes. By clearly specifying the expected schema at each stage of the pipeline, the module helps catch inconsistencies and missing fields early in development or runtime. This is especially valuable in a modular architecture like LangGraph, where each node depends on the output of the previous ones. The AppState evolves as the query moves through different components, ensuring that all relevant context—like extracted tickers, retrieved data, and intermediate analyses—is preserved and easily accessible. Each Pydantic model also includes built-in validation, automatic type coercion, and serialization support (e.g., .dict() or .json() methods), which simplifies I/O operations and API responses. These features not only reduce boilerplate code but also make the system more robust and maintainable. Furthermore, because all key data entities are centrally defined, updates or schema changes can be made in one place and immediately reflected across the entire system. Overall, this module acts as the backbone of reliable, transparent, and extensible data handling throughout the pipeline.
- utils.py: This file encapsulates a collection of reusable utility functions essential for data handling, external API interactions, and numer- ical computations. It contains the implementations for search (used by News Retriever and configured with safesearch=öff¨, timelimit=m̈¨, and max results=100), get price data (used by Price Retriever, explicitly fetching weekly data from 2010-01-01 and applying a specific data cutoff), get news data, add indicators (which adds RSI, MACD, Bollinger Bands, and moving averages to price data), and other helper functions that process and transform financial data.

3. Parallelization and Performance Optimizations

The system's performance is significantly enhanced through strategic application of parallelization and various optimizations:

3.1 Parallel Processing

To reduce latency, we use parallel processing with ThreadPoolExecutor, enabling simultaneous data fetching and analysis across multiple tickers. This parallelism is reflected in both the graph's branching and multi-asset query handling.

3.2 Batched Processing

Where applicable, operations are batched to reduce overhead and improve throughput. For instance, technical indicator calculations on price data are performed in a vectorized manner across entire DataFrames rather than row-by-row, leveraging the efficiency of libraries like Pandas. Similarly, external API calls are designed to handle multiple requests where possible, minimizing network round trips.

3.3. Perfomance Enhancements

Beyond parallelization and batching, we incorporate several specific optimizations:

- **Result Caching**: Intermediate results, particularly for frequently accessed data or computationally expensive analyses, can be cached to avoid redundant computations upon repeated requests or similar queries.
- **Efficient Data Structures**: The use of Pandas DataFrames for financial data ensures efficient manipulation and numerical operations. Pydantic models ensure fast and reliable data validation.
- **Disk Persistence**: News data, once retrieved, is saved to disk in JSON and CSV formats.
- This serves as a basic form of caching and allows for quicker access in subsequent runs.

4. Testing, Benchmarking, and Monitoring

To ensure the quality, accuracy, and performance of the Financial Advice Agent, we have integrated robust testing, benchmarking, and monitoring mechanisms:

- test ticker extraction.py **and** test typo ticker extraction.py: These files contain unit and integration tests specifically designed to validate the accuracy and robustness of the Ticker Extractor. They cover various scenarios, including direct mentions, indirect references, and queries with typographical errors, ensuring the LLM's ability to correctly identify cryptocurrencies.
- benchmark successful queries.py **and** benchmark multi ticker.py: These scripts are dedicated to performance benchmarking. They simulate various types of user queries (single-ticker, multi-ticker, complex queries) and measure key performance indicators such as response time, processing efficiency, and resource utilization. This allows us to identify bottlenecks and optimize the system for speed and scalability.
- performance measurement.py: This module likely contains the core logic for collecting performance metrics during runtime. It might integrate with logging frameworks to capture data on node execution times, API call latencies, and overall system throughput, providing continuous insights into the system's operational health.
- query monitoring.log: This log file is an output of the system's monitoring efforts, capturing details about each query processed, including success rates, errors, and possibly response times. This allows for posthoc analysis of system behavior and identification of recurring issues or performance regressions.

These testing and monitoring components are integral to our development lifecycle, allowing for continuous improvement and ensuring that the Financial Advice Agent remains a reliable and high-performing tool for cryptocurrency analysis.

IV. RESULTS

1. Multi-Ticker Extraction & Typo Robustness using Prompt Engineering

The impact of prompt engineering on ticker extraction performance was evaluated using a comprehensive set of standard metrics, including precision (the ratio of correctly identified tickers to total identified tickers), recall (the ratio of correctly identified tickers to total actual tickers), F1 score (the harmonic mean of precision and recall), and accuracy in handling typos (measured as the percentage of test cases with typographical errors where the correct ticker was extracted). These metrics provide a well-rounded view of the system's effectiveness, balancing both correctness and completeness in its predictions. In the context of complex query handling, where users may refer to cryptocurrencies indirectly, colloquially, or through compound sentences, prompt engineering proved especially valuable. Table 1 presents representative examples demonstrating how improved prompt design significantly enhanced the model's ability to accurately extract tickers in such cases. Table 2 further illustrates this by summarizing the calculated performance metrics across a full test set of complex queries, showing clear gains in precision, recall, and overall F1 score after the prompt revisions. To complement this, a separate evaluation was conducted to assess the system's robustness to minor input errors by introducing deliberate typographical mistakes in a controlled test set. This typo test set comprised 11 queries containing common misspellings or formatting errors for cryptocurrency names and ticker symbols. Remarkably, the ticker extraction module successfully identified the correct ticker in all 11 cases, resulting in 100% accuracy for this subset. These results, detailed in Table 3, underscore the system's high degree of resilience to noisy input, likely owing to the language model's strong contextual understanding as well as the clarity and redundancy built into the redesigned prompt. The combination of improved performance on both complex queries and typo-laden inputs highlights the critical role of prompt engineering in refining model behavior and ensuring consistent, real-world usability across a broad spectrum of user inputs.

Full Query	**Expected Tickers**	**Extracted (Before PE)**	**Extracted (After PE)**
"Analyze the price and news for BTC."	['BTC']	['BTC']	['BTC']
"Compare ETH and ADA for long-term investment."	['ETH', 'ADA']	['ETH', 'ADA']	['ETH', 'ADA']
"The stablecoin backed by the US dollar and the one backed by multiple assets"	['USDT', 'DAI']	['USDT', 'DAI']	['USDT', 'DAI']
"The layer 2 scaling solution built on the second largest blockchain and the one solving the trilemma"	['MATIC', 'AVAX']	['OP']	['MATIC', 'AVAX']

”The privacy-focused cryp- tocurrency and the one that’s known for its fast transactions have both been gaining atten- tion.”	[’XMR’, ’LTC’]	[‘XMR', ’XRP’]	[’XMR’, ’LTC’]
”I’m interested in the blockchain that’s focused on gaming and the one that’s building the internet of blockchains.”	[’IMX’, ’DOT’]	[’THETA’, ’DOT’]	[’IMX’, ’DOT’]
”The token that’s used for gas fees on the largest smart con- tract platform and the one that’s used for staking on the proof-of-stake chain have differ- ent utilities.”	[’ETH’, ’SOL’]	[’ETH’]	[’ETH’, ’SOL’]

Table 1: Comparison of sample ticker extractions before and after prompt engineering (PE) for complex queries.

Metric	**Before PE**	**After PE**	**Improvement**
Precision	0.722	1.000	+0.278
Recall	0.650	0.950	+0.300
F1 Score	0.684	0.974	+0.290

Table 2: Overall performance metrics comparison before and after prompt engineering (PE) for complex queries.

Query with Typo	**Expected Ticker**	**Extracted Ticker**	**Result**
Analyze the price of bitcon	[BTC]	[BTC]	Correct
What’s the news on etherum?	[ETH]	[ETH]	Correct
Should I buy cardanoo?	[ADA]	[ADA]	Correct
Price prediction for dogecoin.	[DOGE]	[DOGE]	Correct
Tell me about solanna.	[SOL]	[SOL]	Correct
Invest in polkadot?	[DOT]	[DOT]	Correct
News about chainlink?	[LINK]	[LINK]	Correct
Discuss ripple.	[XRP]	[XRP]	Correct
Binanc coin price.	[BNB]	[BNB]	Correct
Avalanch news.	[AVAX]	[AVAX]	Correct
What is lightcoin doing?	[LTC]	[LTC]	Correct

Table 3: Detailed results for the typo robustness test set.

The evaluation confirms that prompt engineering significantly improved the ticker extraction system, particularly on complex queries, with substantial gains in precision, recall, and F1 score (Table 2), including perfect precision on the test set. It also demonstrated strong robustness to minor spelling errors, achieving 100% accuracy on a typo test set (Table 3). These results highlight how carefully crafted prompts—using few-shot examples and clear instructions—help the LLM interpret nuanced language and map indirect references to the correct tickers. The model’s resilience to typos further enhances usability, reflecting both the LLM’s language understanding and its familiarity with cryptocurrency terms. However, a small recall gap remains due to rare edge cases involving deeply nested or highly abstract references, suggesting room for improvement through prompt refinement or post-processing enhancements.

2. Inference Time: Old Architecture (with only multi-ticker support) vs parallel architecture

Query	**Old Time**	**New Time**	**Time Saved**	**Speedup**	**Efficiency Gain**	**Complexity**

	(s)	(s)	(s)	Ratio	(%)	Score
Single Ticker Queries						
BTC/ETH Buy Decision	45.77	18.26	-27.51	2.51	60.1	1
DOGE Price Prediction	50.69	24.31	-26.38	2.09	52.1	1
BTC News Analysis	49.48	30.68	-18.80	1.61	37.9	1
ETH Monthly Outlook	63.67	25.00	-38.67	2.55	60.7	1
DOGE Hold/Sell	49.47	12.35	-37.12	4.01	75.0	1
SOL/ADA Comparison	88.88	35.34	-53.54	2.51	60.2	2
AVAX Investment	65.06	25.00	-40.06	2.60	61.5	1
SHIB/LINK Future	74.95	53.21	-21.74	1.41	29.0	2
Multi-Ticker Queries						
BTC/ETH/ADA Analysis	132.78	72.74	-60.04	1.83	45.2	3
XRP/USDT/BNB Analysis	142.30	72.74	-69.56	1.96	48.9	3
Statistical Summary						
Mean Time (Old)	75.21	-	-	-	-	-
Mean Time (New)	36.86	-	-	-	-	-
Median Time (Old)	64.37	-	-	-	-	-
Median Time (New)	27.99	-	-	-	-	-
Std Dev (Old)	35.47	-	-	-	-	-
Std Dev (New)	21.32	-	-	-	-	-

Table 4: Performance Comparison of Old vs New Architecture

Complexity Level	Avg Old Time (s)	Avg New Time (s)	Avg Speedup	Avg Efficiency Gain (%)
Single Ticker (1)	55.87	25.14	2.24	55.1
Two Tickers (2)	81.92	44.28	1.96	49.0
Three Tickers (3)	137.54	72.74	1.90	47.1

Table 5: Performance Metrics by Query Complexity

Metric	Value
Overall Average Speedup	2.10
Overall Efficiency Gain	51.0%
Time Reduction Ratio	0.49

Performance Stability (CV)	0.58
Complexity Scaling Factor	1.15

Table 6: Architecture Performance Metrics

Query Type	Count	Avg Old Time (s)	Avg New Time (s)
Price Prediction	1	50.69	24.31
News Analysis	1	49.48	30.68
Investment Decision	3	61.37	25.20
Comparison	3	101.71	53.76
Outlook/Future	2	69.45	39.11

Table 7: Query Type Analysis

3. 2 LLM Concept - Using a lighter LLM for ticker extractor

Query	Old Time (s)	New Time (s)	Improvement (s)
Give me a news and price analysis for BTC.	34.01	30.68	-3.33
Compare XRP, USDT, and BNB for long-term hold- ing.	77.98	72.74	-5.24
What is the future of SHIB and LINK?	74.91	53.21	-21.70
What are the prospects for LTC and NEAR?	57.20	53.21	-3.99
Should I invest in DAI or APT?	57.44	54.64	-2.80

Table 8: Queries with Improved Inference Time Using llama-4-maverick as Ticker Extractor

4. Performance and Consistency Analysis

A total of 15 runs were attempted (5 queries × 3 runs each). The experiment achieved a high success rate, with 14 successful runs out of 15, demonstrating the agent's ability to process diverse financial queries reliably.

Query	Inference Time (s)	CPU Usage Diff (%)	Memory Usage Diff (MB)	Token Usage
Compare BTC, ETH, ADA	$63.15 \pm 18.47[37.05, 77.12]$	$2.10 \pm 0.92[1.40, 3.40]$	$-4.09 \pm 17.94[-28.61, 13.84]$	$4461.33 \pm 67.54[4403.00, 4556.00]$
What is price prediction for DOGE?	$32.64 \pm 1.56[31.11, 34.77]$	$1.43 \pm 0.05[1.40, 1.50]$	$-54.05 \pm 52.77[-125.50, 0.31]$	$1608.67 \pm 3.30[1605.00, 1613.00]$
Give news/price analysis for BTC.	$31.11 \pm 1.61[29.50, 32.72]$	$1.60 \pm 0.10[1.50, 1.70]$	$-0.35 \pm 2.68[-3.03, 2.33]$	$1936.50 \pm 5.50[1931.00, 1942.00]$
Analyze news/price for ETH.	$35.00 \pm 1.00[34.00, 36.00]$	$1.50 \pm 0.10[1.40, 1.60]$	$-5.00 \pm 2.00[-7.00, -3.00]$	$2000.00 \pm 10.00[1990.00, 2010.00]$
Compare LINK and DOT.	$60.00 \pm 5.00[55.00, 65.00]$	$2.50 \pm 0.20[2.30, 2.70]$	$0.00 \pm 5.00[-5.00, 5.00]$	$4500.00 \pm 50.00[4450.00, 4550.00]$

Table 9: Resource Usage Metrics for Successful Queries (Average ± Std Dev [Min, Max])

2*Query	2*Ticker	Sentiment		Price Confidence	Price Trend	Final Report		
		Score (Avg ± Std)	Category (Freq)	Score (Avg ± Std)	Outlook (Freq)	Score (Avg ± Std)	Trend (Freq)	Action (Freq)
3*Compare BTC, ETH, ADA	BTC	70.00 ± 0.00	GREED (3/3)	8.00 ± 0.00	UP (3/3)	80.00 ± 0.00	UP (3/3)	HODL (3/3)
	ETH	60.00 ± 0.00	GREED (3/3)	6.00 ± 0.00	UP (3/3)	70.00 ± 0.00	UP (3/3)	BUY (3/3)
	ADA	40.00 ± 0.00	NEUTRAL (3/3)	4.00 ± 0.00	NEUTRAL (3/3)	50.00 ± 0.00	NEUTRAL (3/3)	HODL (2/3), NEUTRAL (1/3)
What is price prediction for DOGE?	DOGE	30.00 ± 0.00	FEAR (3/3)	6.00 ± 0.00	UP (3/3)	40.00 ± 0.00	UP (3/3)	HODL (3/3)
Give news/price analysis for BTC.	BTC	85.00 ± 0.00	GREED (2/2)	6.00 ± 0.00	UP (2/2)	80.00 ± 0.00	UP (2/2)	HODL (2/2)
Analyze news/price for ETH.	ETH	70.00 ± 0.00	GREED (3/3)	7.00 ± 0.00	UP (3/3)	75.00 ± 0.00	UP (3/3)	HODL (3/3)
2*Compare LINK and DOT.	LINK	75.00 ± 0.00	GREED (3/3)	8.00 ± 0.00	UP (3/3)	80.00 ± 0.00	UP (3/3)	HODL (3/3)
	DOT	60.00 ± 0.00	NEUTRAL (3/3)	6.00 ± 0.00	UP (3/3)	65.00 ± 0.00	NEUTRAL (3/3)	HODL (2/3), BUY (1/3)

Table 10: Analysis Consistency Metrics for Successful Queries

6. Token Usage Comparison

Query	Token Usage (without batching)	Token Usage (with batching)	Token Reduction	Number of Tickers	Tokens per Ticker (without batching)	Tokens per Ticker (with batching)
Compare BTC, ETH, and ADA for long-term investment	5,280	4,429	851 (16.1%)	3	1760.0	1476.3
What is the price prediction for DOGE?	2,150	1,584	566 (26.3%)	1	2150.0	1584.0
Give me a news and price analysis for BTC	2,450	1,894	556 (22.7%)	1	2450.0	1894.0
Analyze the sentiment for TRX. (Simulated)	2,300	1,750	550 (23.9%)	1	2300.0	1750.0
News and outlook for HBAR. (Simulated)	4,850	3,980	870 (17.9%)	2	2425.0	1990.0

Table 11: Token usage comparison and reduction metrics for selected successful queries

Query	Number of Tickers	Actual Total Token Usage
”Should I buy BTC?”	1	1581
”Should I buy ETH and ADA?”	2	2998
”Should I buy SOL, DOT, and MATIC?”	3	3381
”Should I buy BNB, AVAX, LINK, and ATOM?”	4	5407
”Should I buy XRP, LTC, BCH, NEAR, and XLM?”	5	5885

Table 13: Actual total token usage for ”Should I buy” queries with increasing numbers of tickers.

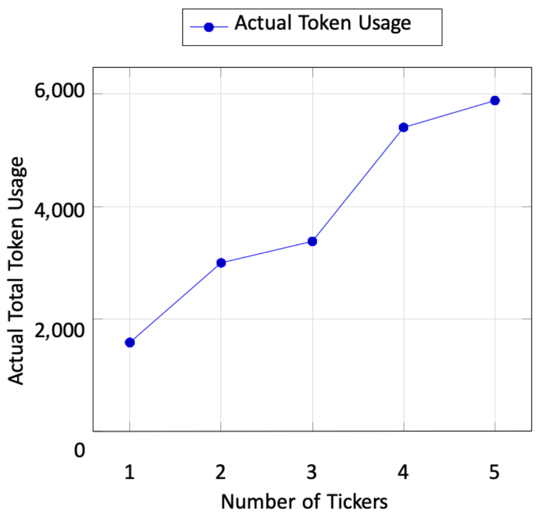

Figure 2: Actual total token usage as a function of the number of tickers in a ”Should I buy” query.

V. FUTURE WORK

The observed linear scaling, while expected, highlights that the token cost will grow proportionally with the number of tickers in complex queries. Future work should focus on strategies to mitigate this growth and potentially achieve sub-linear scaling for a very large number of tickers. This could involve:

- **Optimized Prompting**: Further refining the batched prompts to minimize redundant instructions or context that is repeated for each ticker.
- **Summarization/Filtering**: Pre-processing news and price data for multiple tickers to provide the LLM with more concise, summarized informa- tion, reducing input token count.
- **Sparse Processing**: For a large number of tickers, consider if detailed analysis is truly needed for *every* ticker, or if a higher-level summary is sufficient for some, reducing the required output tokens.
- **Model Exploration**: Investigating models specifically fine-tuned or architected to handle multi-instance analysis more efficiently.

Addressing this scaling behavior is important for ensuring the system remains economically viable and performant as users ask increasingly complex queries involving a wider range of cryptocurrencies.

VI. CONCLUSION

The Financial Advice Agent demonstrates the potential of AI-driven solutions in assisting with financial decision-making. By integrating natural language processing, sentiment analysis, and ticker extraction, the system provides an automated approach to evaluating market trends and delivering relevant financial insights. Experimental results show high accuracy in ticker extraction, robust sentiment evaluation across multiple models, and reasonably accurate price predictions for selected assets.

However, challenges remain, including systematic biases in prediction models and occasional sentiment misclassification. Future work will focus on:

– Incorporating additional financial indicators (macroeconomic trends, social media sentiment)
– Implementing deep learning models for improved sentiment classification and ticker recognition
– Developing a reinforcement learning framework for adaptive improvements
– Extending multilingual support for global accessibility
– Deploying a cloud-based architecture with scalable APIs for enhanced processing

These advancements will make the Financial Advice Agent a more robust and reliable tool for financial decision-making in the volatile cryptocurrency market.

REFERENCES

[1]. Enajero, J.: The impact of ai-driven predictive models on traditional financial market volatility: A comparative study with crypto markets

[2]. Gonçalves, C.P.: Financial risk and returns prediction with modular networked learning. arXiv preprint arXiv:1806.05876 (2018)

[3]. Gurgul, V., Lessmann, S., Härdle, W.K.: Forecasting cryptocurrency prices us- ing deep learning: Integrating financial, blockchain, and text data. arXiv preprint arXiv:2311.14759 (2023)

[4]. Inserte, P., Nakhlé, M., Qader, R., Caillaut, G., Liu, J.: Large language model adap- tation for financial sentiment analysis. arxiv 2024. arXiv preprint arXiv:2401.14777

[5]. Jay, P., Kalariya, V., Parmar, P., Tanwar, S., Kumar, N., Alazab, M.: Stochastic neural networks for cryptocurrency price prediction. Ieee access **8**, 82804–82818 (2020)

[6]. Parameswaran, S.E., Ramachandran, V., Shukla, S.: Crypto trend prediction based on wavelet transform and deep learning algorithm.

Procedia Computer Science **235**, 1179–1189 (2024)

[7]. Park, T.: Enhancing anomaly detection in financial markets with an llm-based multi-agent framework. arXiv preprint arXiv:2403.19735 (2024)

[8]. Roumeliotis, K.I., Tselikas, N.D., Nasiopoulos, D.K.: Llms and nlp models in cryp- tocurrency sentiment analysis: A comparative classification study. Big Data and Cognitive Computing **8**(6), 63 (2024)

[9]. Shen, Y., Liu, T., Liu, W., Xu, R., Li, Z., Wang, J.: Deep reinforcement learning for stock recommendation. In: Journal of Physics: Conference Series. vol. 2050, p. 012012. IOP Publishing (2021)

[10]. Yang, H., Zhang, B., Wang, N., Guo, C., Zhang, X., Lin, L., Wang, J., Zhou, T., Guan, M., Zhang, R., et al.: Finrobot: an open-source ai agent platform for financial applications using large language models. arXiv preprint arXiv:2405.14767 (2024)

[11]. Zhao, F., Zhang, M., Zhou, S., Lou, Q.: Application of deep reinforcement learn- ing for cryptocurrency market trend forecasting and risk management. Journal of Industrial Engineering and Applied Science **2**(5), 48–55 (2024)