VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

____****____

BIG DATA ENGINEERING AND TECHNOLOGIES FOR ARTIFICIAL INTELLIGENCE

REPORT

TOPIC: REAL-TIME CRYPTO ANALYSIS AND PREDICTION

Supervisors: Dr. Trần Hồng Việt
M.Sc. Ngô Minh Hương

Students: Bùi Thanh Dân - 23020342
Phạm Tiến Dũng - 23020345
Vũ Nguyên Đan - 23020351

Hanoi, 2025

TASKS OF TEAM MEMBERS	
Full name	**Tasks / Responsibilities**
Bùi Thanh Dân	**- Design and build the overall system architecture (Big Data infrastructure, processing pipelines, Docker deployment).** **- Implement the base code for the project**: Kafka ingestion, Spark Streaming + Batch APIs, HDFS/HBase integration. - Design dataflow and optimize pipeline performance. - Integrate Apache Airflow into the system, setup periodic data-processing DAGs. - QA across the project: test both batch and stream layers. - Write Chapter 2 of the report, prepare slides, present.
Phạm Tiến Dũng	**- Develop the AI Chatbot system (Gemini LLM + NewsAPI + web scraping).** - Design NLP logic: intent recognition, language analysis, sentiment detection and LLM-based responses. - Upgrade the **Real-time Dashboard** (Flask + JavaScript UI), improve UX/UI. - Optimize the stream layer: real-time processing with Spark Streaming, tune writes to HBase. - Testing and refactoring backend and chatbot code. - Write Chapter 3 of the report, prepare slides, present.
Vũ Nguyên Đan	**- Build and optimize the Batch Layer**: historical data processing, compute 40+ technical indicators. **- Train, evaluate and optimize the XGBoost model** (1-hour + 1-day forecasting for 10 cryptocurrencies). - Design model outputs: prediction, confidence, volatility, momentum, technical signals. - **Develop the advanced Streamlit Dashboard**: KPIs, correlation heatmap, portfolio analytics, volatility/risk metrics. - Write Chapter 1 and appendices, write README, participate in presentations.

INTRODUCTION

The rapid development of blockchain technology has turned the cryptocurrency market into a dynamic financial domain, attracting millions of investors worldwide. This market generates massive amounts of data characteristic of Big Data — Volume, Velocity, Variety, Veracity and Value. However, high volatility and fast data generation pose a major challenge: how to efficiently and timely extract and analyze this data to support decision-making.

Against this background, the topic "Real-time crypto analysis and prediction" is carried out to design and deploy a system capable of handling data in both directions: real-time streaming and historical batch processing. Specifically, the group focuses on building a high-performance data collection pipeline; deploying a Stream Layer for immediate predictions and a Batch Layer for deep analysis and model training; and integrating a virtual assistant using a Large Language Model (LLM) to provide an interactive, information-rich interface.

The choice of this topic stems from practical needs when traditional methods cannot meet the speed and scale of cryptocurrency data. The Lambda architecture, with its ability to combine hot-path streaming and cold-path batch processing, is an appropriate solution. Alongside the development of artificial intelligence, especially LLMs, users can not only observe but also "converse" with the data, ask questions and receive synthesized analyses from multiple sources, improving decision effectiveness.

To ensure feasibility, the project focuses on price data (OHLCV) and news of several popular coins. The system is built on open-source platforms including Apache Kafka, Spark, HDFS, HBase, MongoDB and deployed using Docker. The research target is not only the analytical outcomes but also the performance and interoperability of system components in solving practical problems.

The group applies a systematic research process: from studying Lambda architecture theory and related technologies, to detailed dataflow design, and module-by-module implementation using Docker Compose to ensure consistency. Finally, the system is comprehensively tested to evaluate performance, latency and output quality.

The report consists of three main chapters: Chapter 1 presents the overview, challenges and theoretical foundations; Chapter 2 describes the architecture, technologies and data processing pipelines in detail; Chapter 3 summarizes results, evaluates limitations and proposes future work.

Contents

Chapter 1

OVERVIEW

1.1 Overview of the research area

1.1.1 Data in the cryptocurrency market

The cryptocurrency market is one of the most volatile domains today, where every change in price, volume or order state generates a high-frequency real-time data stream. Each recorded trade contributes to continuous time series reflecting the market's state in real time. This is an important characteristic of decentralized financial systems and high-frequency trading markets. Common data sources in the cryptocurrency market include:

- **Trade data:** Includes matched prices, traded volumes and timestamps. This data is updated at millisecond granularity, directly reflecting market buy/sell activity.
- **Order book data:** Shows bid/ask prices, market depth and liquidity. Order books change very rapidly and are widely used in high-frequency trading (HFT) models.
- **OHLCV data:** Includes open, high, low, close prices and traded volume for intervals (1 minute, 1 hour, 1 day, ...). This standardized format supports technical analysis and predictive modeling.
- **Unstructured data:** News, social media, announcements from exchanges or blockchain projects. This type of data is often unstructured and is used for sentiment analysis or event detection.

The common traits of crypto market data are **large volume**, **high velocity** and **continuous volatility**. This makes data collection, storage and analysis a significant technical challenge, requiring systems to be both scalable and low-latency.

1.1.2 Challenges in real-time analysis

Real-time analysis in the cryptocurrency domain faces several important challenges, requiring systems to ensure high processing speed, scalability and stability under severe market fluctuations.

- **High-velocity data:** Exchanges can generate thousands of events per second. This requires systems to process continuously with extremely low latency so that analytics and predictions remain timely.
- **Market instability:** Crypto prices can swing sharply in very short time frames. Predictive models are easily disrupted if data is not properly cleaned, normalized and synchronized.

- **Diverse data sources:** Combining quantitative data like *price, volume* with unstructured data like *news, sentiment* complicates enrichment and feature extraction.
- **Scalability requirements:** Systems need horizontal scalability to handle continuously increasing data loads, especially during major market events (e.g., sudden spikes or drops).
- **Limitations of traditional models:** Classical statistical models often cannot keep up with the complex dynamics of crypto markets, motivating the use of ML/DL techniques to improve accuracy.

These challenges motivate the adoption of modern data processing architectures such as **Lambda Architecture**, and encourage applying ML/DL/AI to improve prediction accuracy and adaptability in crypto market analysis.

1.2 Theoretical background

1.2.1 Big Data characteristics and the Lambda Architecture

Big Data is often described by three core characteristics: **Volume**, **Velocity** and **Variety**. In the cryptocurrency market context, all three are present. In particular, **Velocity** is strong as price, volume and market events are updated continuously at millisecond granularity.

To handle high-velocity streams effectively, the **Lambda Architecture** is applied. This model combines the advantages of two processing approaches to ensure both high accuracy and real-time responsiveness.

- **Batch Layer:** processes large-scale historical data. This layer has higher latency but delivers reliable and accurate computations for aggregations and analytical models.
- **Speed Layer (Streaming Layer):** processes data as it arrives to produce immediate results, compensating for the batch layer's latency.
- **Serving Layer:** stores, aggregates and serves outputs to applications, models or user interfaces.

By combining these, Lambda Architecture allows systems to guarantee **accuracy** (from the batch layer) and **timeliness** (from the speed layer), making it suitable for real-time financial analytics and prediction tasks in crypto markets.

1.2.2 Big Data features and parallel processing

Big Data systems often rely on **distributed parallel processing**, which partitions data and distributes compute tasks across multiple nodes in a server cluster. This approach optimizes processing speed, increases fault tolerance, and allows systems to scale flexibly. Important characteristics include:

- **Parallel Processing:** Frameworks like Hadoop and Apache Spark enable breaking processing into small tasks that run in parallel to accelerate computation on large datasets.
- **Fault Tolerance:** Systems provide fault resilience via data replication and task retries, allowing pipelines to continue operating when some nodes fail.
- **Distributed Storage:** Data is stored and managed in distributed form (e.g., HDFS), improving scalability and reliability in big data environments.

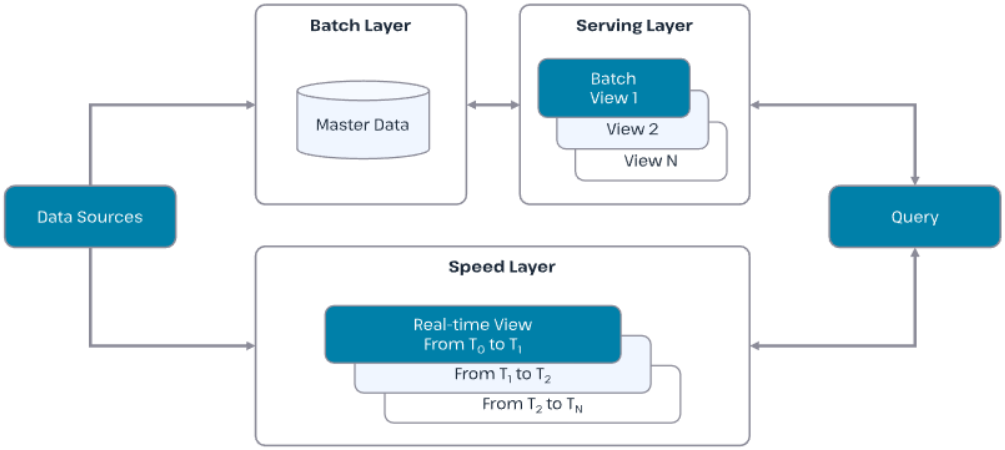

Figure 1.1: General overview of the Lambda architecture.

- **Scalability:** Systems can scale by adding compute or storage resources without changing application structure.

In this project, Apache Spark plays a central role in parallel processing for both **streaming data** from Kafka and **batch data** collected from Yahoo Finance. Combining these processing modes ensures the system is both responsive in real time and capable of high-performance historical analysis.

1.2.3 AI applications in financial analysis

Artificial intelligence (AI) and machine learning (ML) have become key technologies in modern financial analysis. With the ability to model nonlinear relationships and discover hidden patterns in noisy data, AI models often outperform traditional statistical methods. Notable research directions include:

- **Time-series forecasting:** Models like recurrent neural networks (RNN, LSTM) and ensemble algorithms (Random Forest, XGBoost) have shown strong performance in price and volatility forecasting, especially for nonlinear time series [1, 2].
- **Sentiment analysis:** Mining news, social media, forums and user behavior helps assess market sentiment. Studies show sentiment indices correlate significantly with price volatility across asset classes, including crypto [3].
- **Anomaly detection and risk:** Unsupervised models like Isolation Forest, Autoencoders, and clustering are used to detect anomalous trades, market shocks, or signs of price manipulation.
- **Portfolio optimization:** Heuristic algorithms and reinforcement learning are increasingly applied to support asset allocation and risk management under fast-changing market conditions.

Overall, AI plays an important role in improving forecasting accuracy, enhancing risk detection and extracting value from Big Data in modern finance.

1.2.4 Integrating price data and news in crypto analysis

In crypto markets, combining structured time-series price data with unstructured news and sentiment data is a comprehensive and effective approach. Price data reflects market state at each timestamp, while news provides qualitative context about sentiment, policy, events and other factors that can strongly impact price movements.

Some notable findings indicate that:

- **News has a strong impact on short-term volatility**, especially in crypto markets which are sensitive to investor sentiment and frequent changes. [4]
- **Hybrid models** that combine price data and sentiment data often achieve better forecasting accuracy than models relying solely on quantitative data. [5]
- **Alternative data**, including news, social media and community engagement metrics, provide important supplementary signals that help models better understand market drivers. [6]

The integration process typically includes these steps:

1. Collect price data from exchanges;
2. Collect news via APIs or web crawling;
3. Preprocess and extract textual features using NLP techniques (BERT, TF-IDF, LSTM embeddings);
4. Synchronize both data sources by timestamp;
5. Build forecasting or risk assessment models.

This approach helps models not only reflect pure price movements but also understand the underlying causes of those movements, improving accuracy and reliability.

Chapter 2

SYSTEM AND DEPLOYED TECHNOLOGIES

2.1 Overall system architecture and dataflow

The system is designed according to the **Lambda Architecture**, leveraging the strengths of both data processing paradigms: *real-time streaming* and *batch processing*. This model allows the system to respond quickly to real-time market data while also aggregating and deeply analyzing historical data, and integrates news analysis using a Large Language Model (LLM). The system dataflow includes four main layers:

- **Ingestion Layer:**

  - Collect price and volume data from *Binance WebSocket API* (real-time) and historical data from *Yahoo Finance API*.

  - Data is pushed into *Apache Kafka*, which acts as a message broker, decoupling data sources from downstream processing layers.

- **Stream Layer:**

  - A pre-trained *XGBoost* model (trained in the batch layer) is loaded to produce **short-term predictions** for each cryptocurrency.

  - Processing results are written to *Apache HBase* to support fast queries and are exposed via a Flask API.

- **Batch Layer:**

  - PySpark periodically reads historical data from *HDFS*, performs cleaning, feature engineering, and retrains *XGBoost* models.

  - After training, new models are saved to *HDFS* and synchronized to the Stream Layer for real-time inference.

  - Aggregated statistics and long-term analysis results are stored in *MongoDB* for display through the Streamlit Dashboard.

- **Serving Layer:**

  - *Apache HBase* serves low-latency read requests from the real-time layer via the Flask API.

  - *MongoDB* stores batch results and long-term statistics, which are displayed through the Streamlit UI.

- **Addition**

  - **LLM (Large Language Model)** receives user questions and news data from *NewsAPI* via Flask, performs summarization, sentiment analysis and assesses market impact.

  - Analytical outputs are combined with price data and displayed in the Flask UI where users can interact via the AI chatbot.

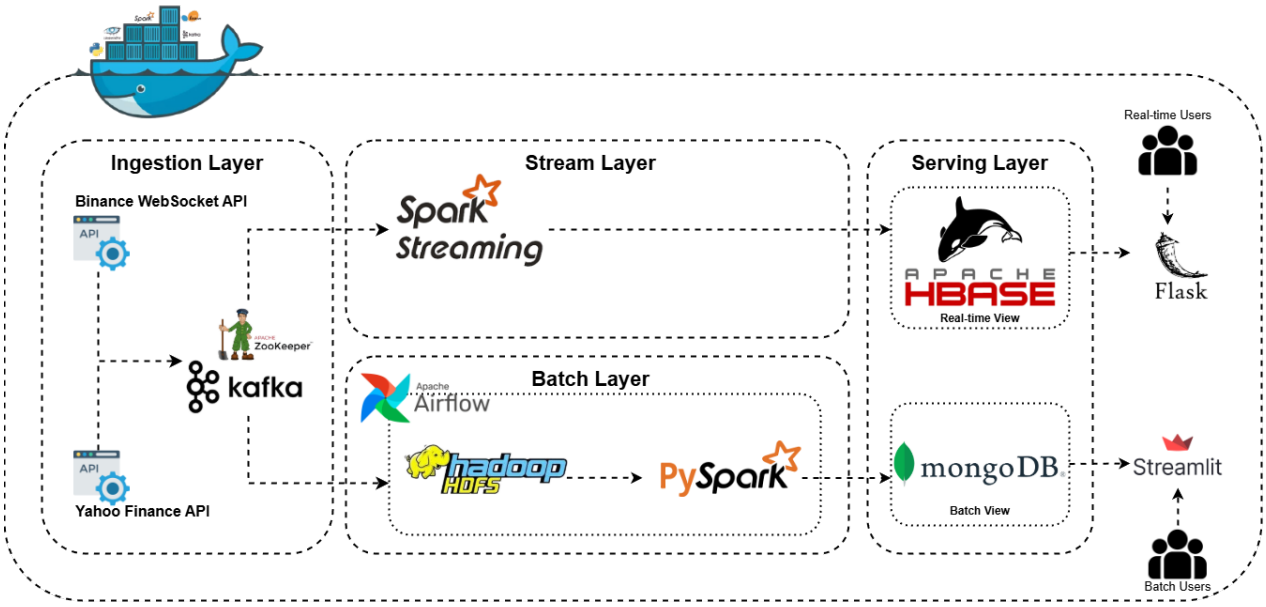

Figure 2.1: Overall system architecture (Lambda Architecture).

Summary: The main dataflow goes from input APIs (Binance WebSocket API, Yahoo Finance API) → Kafka, then is processed in parallel by the **Stream Layer** and the **Batch Layer**. Results are stored in respective databases: Apache HBase for the stream and MongoDB for the batch. Apache Airflow automates historical data collection (Yahoo Finance) and regular XGBoost model training. Data is displayed via two user interfaces. **Flask API** provides data for *real-time users* (from HBase) and integrates with the LLM and NewsAPI. Aggregated batch data (MongoDB) is provided to *batch users* through the dashboard.

2.2 Technologies used

Technology choices determine the system's success and scalability. This section presents core technology groups organized into three pillars: a Big Data platform for collection and storage; analytics, applications and user interfaces; and deployment technologies for operational efficiency and stability.

2.2.1 Big Data platform

The system is built on a Big Data platform to ensure the ability to collect, process and store large volumes of data in real time.
Main technologies include:

Apache Kafka: a message queue system that acts as an intermediary to transmit data between components. Kafka ingests streams from the *Binance WebSocket API* and distributes them to processing services (Spark Streaming, Flask API). Kafka ensures reliability, reduces latency and scales well.

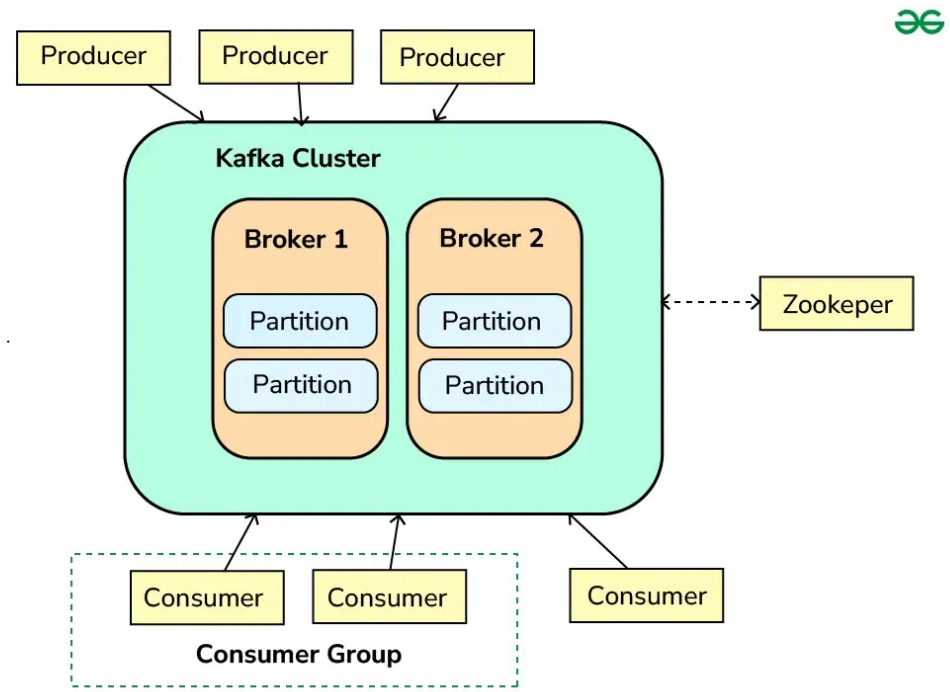

Figure 2.2: Basic architecture of Apache Kafka.

Apache Spark: a distributed data processing engine used for both the *batch layer* and the *stream layer*. Spark Streaming handles real-time crypto price streams, while Spark batch is used to train XGBoost models on historical data. Spark optimizes processing speed and integrates well with HDFS and HBase.

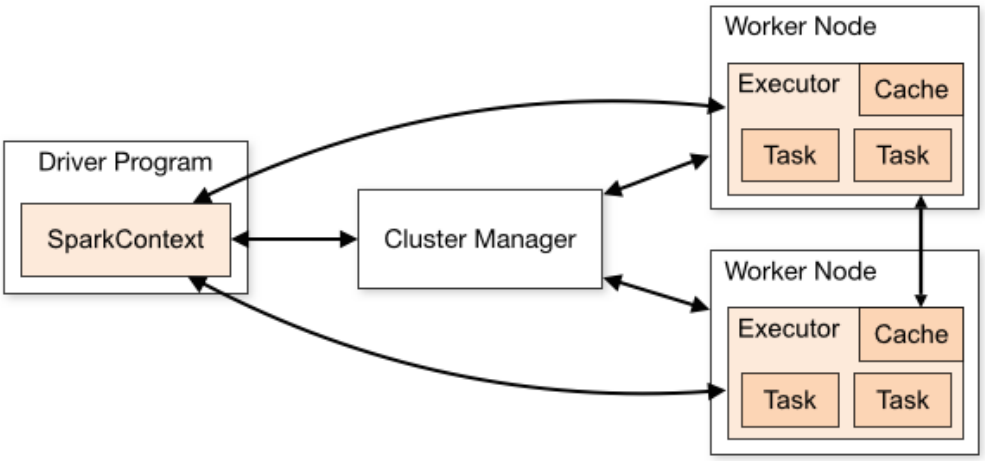

Figure 2.3: Data processing flow with Apache Spark.

Hadoop Distributed File System (HDFS): used to store historical crypto data collected from Yahoo Finance. HDFS ensures scalability, data safety, and supports efficient Spark read/write operations for model training.

Apache Airflow: an open-source platform for scheduling, managing and monitoring workflows as DAGs (Directed Acyclic Graphs). Airflow automates data processing pipelines and tracks job statuses via a web UI. In this project, Airflow automates historical data ingestion (HDFS and MongoDB) and model training tasks.

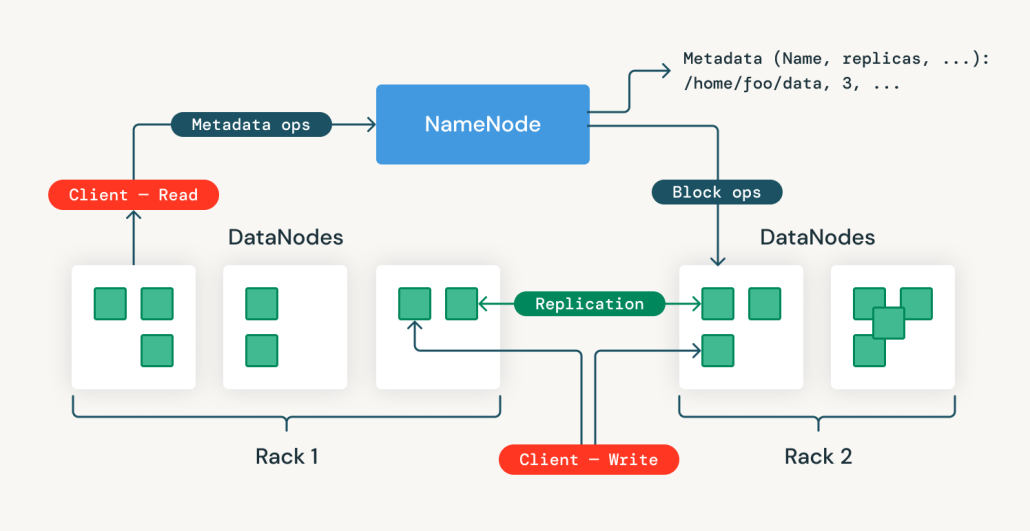

Figure 2.4: Basic architecture of HDFS.

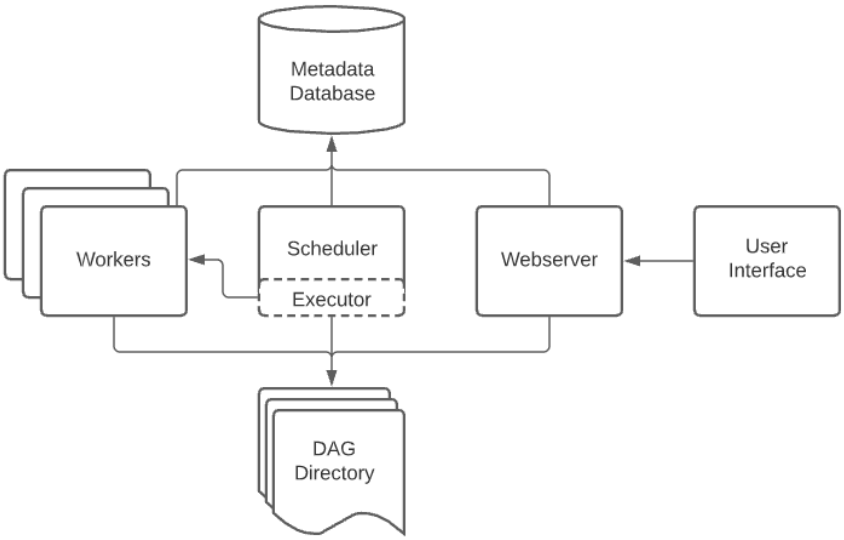

Figure 2.5: Basic architecture of Airflow.

Apache HBase: a NoSQL database optimized for random access and high throughput. In the system, HBase stores real-time data from Spark Streaming to support fast API queries and dashboard display.

MongoDB: used as the batch storage layer, holding aggregated statistics, long-term predictions and processed records. MongoDB provides flexible schema and supports display of aggregated data via Streamlit.

2.2.2 Analytics, applications and interfaces

1. **LLM (Large Language Model):** integrated to analyze news and articles related to the crypto market. The LLM extracts *sentiment*, assesses market trend signals from news sources and provides natural responses for the advisory chatbot. The system uses the *Google AI Studio* API for this functionality.

2. **NewsAPI:** a global news data source used to collect articles related to crypto and finance. These news items are preprocessed before being sent to the LLM for semantic analysis

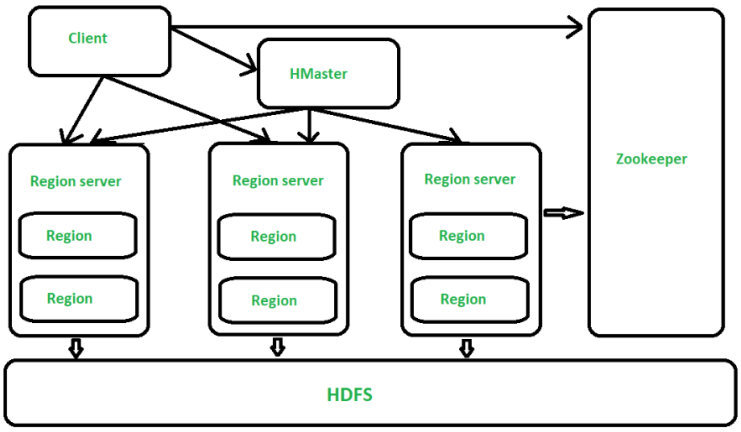

Figure 2.6: Basic architecture of Apache HBase.

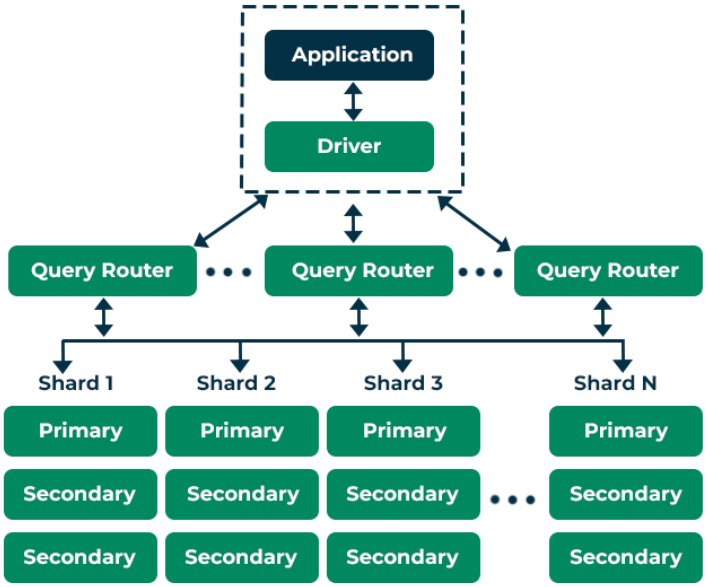

Figure 2.7: Architecture of MongoDB.

and summarization.

3. **Flask:** a lightweight Python web framework used to build REST APIs and real-time interfaces. Flask mediates between processing services, receiving user or dashboard requests, querying *HBase* and returning instantaneous results such as current prices, short-term predictions, or news analysis from the *LLM*.

4. **Streamlit:** an analytics and visualization platform used in the *batch layer*. The Streamlit app visualizes aggregated statistics, long-term predictions and indicators stored in *MongoDB*. This interface supports analysis of overall market trends, complementing Flask which focuses on real-time data.

2.2.3 Deployment technologies

The system is deployed entirely using **Docker Compose**, allowing simulation and operation of a distributed environment with a single configuration file. Each component — *Kafka*, *Spark*, *Airflow*, *HBase*, *MongoDB*, *Flask*, and *Streamlit* — is packaged in an independent container to ensure isolation, flexibility and easy scaling.
Docker Compose automates initialization, sets up inter-service networking, and manages container startup order. This deployment model optimizes resource usage and enables consistent deployments across environments, from local machines to cloud platforms.

2.3 Processing pipeline, deployment and results

This section details the data processing pipeline, system deployment and evaluation results. The three key steps include data collection and prediction, system deployment and integration, and final results with in-depth evaluation.

2.3.1 Data collection, processing and prediction

The system collects data simultaneously from three sources:

- **Binance WebSocket API:** provides real-time price and volume data for 10 cryptocurrencies.
- **Yahoo Finance API:** provides historical (batch) data used for model training.
- **NewsAPI:** gathers news and articles related to the crypto market as input to the LLM. This API is called when a user queries the chatbot via Flask; the returned news/articles are then fed into the AI chatbot.

Dataflow after collection:

- Real-time data stored in HBase is passed through a pre-trained *XGBoost* model (interval=1h) to produce short-term predictions.
- PySpark in the batch layer processes historical data, trains *XGBoost* models (for short- and long-term forecasts), then syncs models to the stream layer for inference.

In addition to quantitative XGBoost models, the system integrates a **LLM** that analyzes news (sentiment analysis, topic extraction) and provides contextual support for predictions and market analysis. Outputs from both model types are combined to form a synthesized market outlook.

2.3.2 Deployment and integration

The system is fully deployed using **Docker Compose**, ensuring each service runs independently yet connects over an internal network:

- **Kafka, Spark, Airflow, HBase, MongoDB** run as independent containers handling ingestion, processing and storage.
- **Flask API** provides REST and real-time websocket interfaces and integrates the AI chatbot for user interactions.

- **Streamlit Dashboard** displays aggregated data, analytical charts, and batch-layer indicators to support long-term trend analysis.

Docker containers are orchestrated with dependency-aware startup order (e.g., infrastructure services like Kafka, Zookeeper, Airflow → HDFS → databases (HBase, MongoDB) → processing applications and APIs (Flask)). The system is stable, scalable and compatible with cloud deployment.

2.3.3 Results and evaluation

1. Real-time dashboard: The Flask UI displays crypto data, real-time price charts, short-term predicted prices and an AI chatbot. Users can ask questions; the chatbot automatically queries the most recent news for the day to better understand market trends and provide more accurate responses.

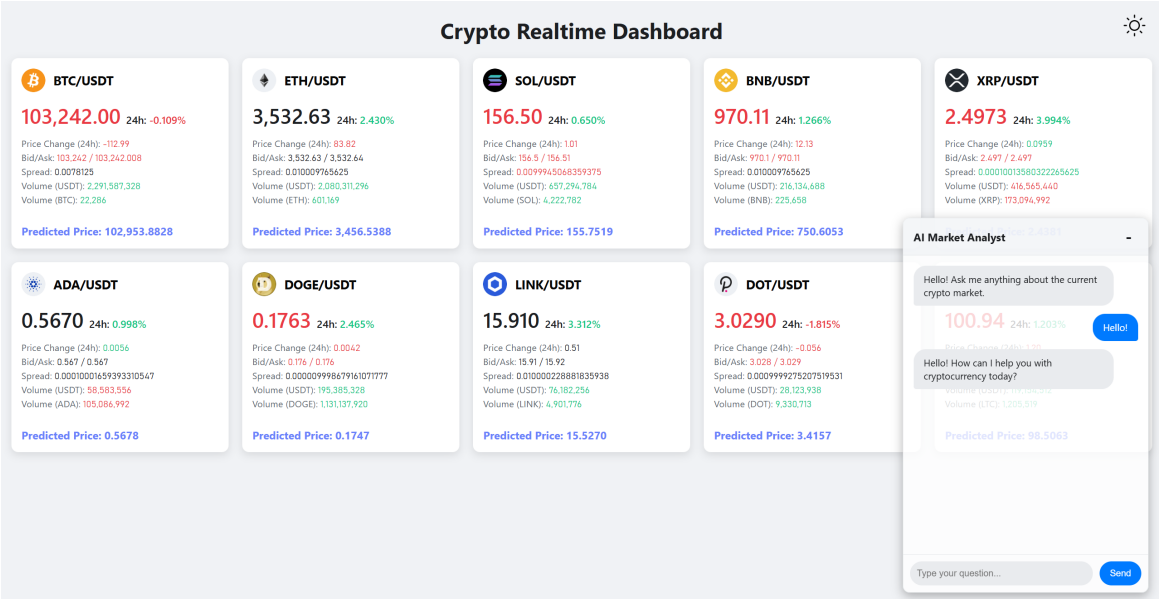

Figure 2.8: Real-time dashboard interface 1.

2. Batch dashboard (Streamlit): Streamlit shows historical data and aggregated statistics (average price, trading volume, weekly/monthly trends). Data is retrieved from MongoDB — the repository of batch results. Airflow updates this data daily depending on initial user settings.

3. Performance:

- Average latency in the real-time pipeline: $<$ **10 seconds** and small latency from Binance WebSocket API.
- Batch training speed: XGBoost training averages **2 minutes for 2 years of data per coin**.
- Scalability: the system can add new coin pairs or services without changing core architecture.

4. Overall assessment: The system operates stably, with low latency, quick responses and provides comprehensive information for both short-term traders and long-term analysts. Results indicate that combining **XGBoost (quantitative)** and **LLM (qualitative)** improves analysis accuracy and persuasiveness.

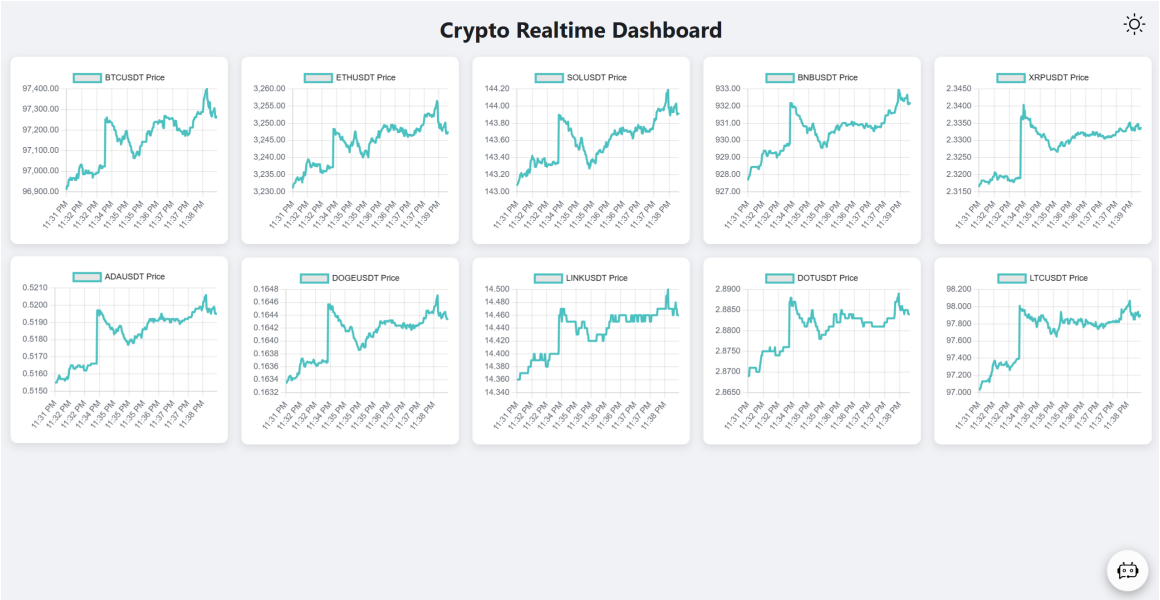

Figure 2.9: Real-time dashboard interface 2.

Figure 2.10: Batch dashboard interface.

Chapter 3

CONCLUSION AND FUTURE DIRECTIONS

3.1 Conclusion

3.1.1 Summary of content and achieved results

Within the scope of this project, an end-to-end platform for analyzing and forecasting the cryptocurrency market was fully built, integrating all steps from real-time data collection, processing and storage, model training, to visualization and interactive user interfaces.
Key achievements include:

- **Built a real-time data collection system** from Binance WebSocket, routed through Kafka and processed with Spark Structured Streaming.
- **Designed a real-time data store** using HBase, enabling fast timestamp-based queries and durable storage of price data.
- **Implemented a batch pipeline** using PySpark + Yahoo Finance:
  - Collected 1h/1d data for 2 years and computed 40+ technical indicators.
  - Trained XGBoost models for each cryptocurrency.
  - Produced forecasts, confidence scores, cumulative volatility, ... and stored results in HDFS and MongoDB.
- **Built an analytics dashboard** with Streamlit:
  - Displays KPIs, risk distributions, price forecasts, asset comparisons, correlation heatmaps and portfolio suggestions.
  - Monitors system status (model coverage, data quality...).
  - Users can download aggregated statistics as `.csv` files from Streamlit.
- **Developed a market analysis chatbot**, integrating:
  - News via NewsAPI + trafilatura.
  - Real-time technical analysis based on HBase data.
  - Natural language processing with Gemini, intent and sentiment recognition.
  - Expert-style response generation for market commentary.

The completed system demonstrates the ability to combine Big Data, Machine Learning, Data Visualization and NLP within a unified, smoothly operating and scalable architecture.

3.1.2 Evaluation and limitations

Although many important results were achieved, the system still has several limitations:

(1) Regarding forecasting models

- XGBoost is strong but not optimal for highly nonlinear, high-volatility time series like crypto.
- Deep learning architectures specialized for time-series (LSTM, TCN, Temporal Fusion Transformer) were not explored.
- MAPE and RMSE vary widely across coins due to differences in volume/volatility.

(2) Regarding data

- Yahoo Finance data may be misaligned with the live market, affecting training.
- On-chain data and sentiment sources — important for crypto — have not been integrated.
- HBase is not yet optimized for complex queries (e.g., long-range aggregation).

(3) Regarding system architecture

- The system currently runs only locally via Docker.
- High hardware requirements for local deployment (recommended RAM 16–32GB).
- Missing full monitoring (Prometheus, Grafana).
- Some components depend on local configuration and are not fully containerized.

(4) Regarding interface and UX

- The Streamlit dashboard lacks advanced customization such as multi-user support and access control.
- The chatbot sometimes produces overly long responses or over-analyzes when news is scarce.

These limitations suggest many directions for future improvement.

3.2 Future directions

In the future, the system can be extended and upgraded along the following directions:

(1) Upgrade machine learning models

- Apply deep learning architectures specialized for time series [7, 8]:
  - LSTM / BI-LSTM
  - Temporal Convolutional Network
  - Transformer for Time Series
  - NeuralProphet
- Merge real-time data from HBase with batch data to create hybrid models.
- Automate hyperparameter tuning using Optuna or Hyperopt [9].

(2) Expand data sources

- Integrate additional data sources [10]:
  - On-chain (Glassnode, Dune Analytics)
  - Social sentiment (Twitter, Reddit)
  - Orderbook depth for liquidity analysis
- Store real-time data partitioned by day/month to speed up queries.

(3) Harden Big Data architecture

- Deploy the system to cloud (GCP/AWS) or Kubernetes to remove local Docker dependency.
- Move heavy services (Spark, Kafka, HBase) to managed cloud clusters to reduce local hardware requirements.
- Integrate Prometheus + Grafana for monitoring [11]:
  - Kafka lag
  - Spark microbatch latency
  - HBase write/read metrics
- Use MinIO or S3 as HDFS replacements to enable cloud-native architecture.

(4) Upgrade dashboard and chatbot

- **Dashboard:**
  - Optimize UI/UX, support real-time updates via WebSocket.
  - Enable timeline comparisons across multiple cryptocurrencies.
- **Chatbot:**
  - Add long-term memory:
    - Per user
    - Per market history
  - Allow automatic PDF report generation on demand.
  - Integrate voice or Telegram bot interfaces.

(5) Practical deployment directions

- Deploy the system to cloud (GCP/AWS/DigitalOcean).
- Expose a public API for reading real-time price data and forecasts.
- Create a ”paper trading” module to test strategies based on predictions.

Bibliography

[1] G. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[2] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[3] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, no. 1, pp. 1–8, 2011.

[4] D. Shen, W. Zhang, and X. Xiong, "The impact of news on cryptocurrency markets," *Finance Research Letters*, vol. 30, pp. 1–7, 2019.

[5] A. Mittal and A. Goel, "Stock prediction using news sentiment analysis," in *Proc. IEEE/WIC/ACM International Conference on Web Intelligence*, 2012.

[6] J. Liew and T. Budavari, "The use of alternative data in investment management: A practitioner's guide," *The Journal of Financial Data Science*, vol. 1, no. 1, pp. 10–23, 2019.

[7] B. Lim *et al.*, "Temporal Fusion Transformers for interpretable multi-horizon time series forecasting," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[8] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[9] T. Akiba *et al.*, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

[10] F. Victor, "Measuring Ethereum-based assets," in *Financial Cryptography and Data Security*, 2021.

[11] B. Turner, *Monitoring with Prometheus*. O'Reilly Media, 2020.

[12] Documentation: Apache Kafka, Apache Spark, HBase, MongoDB, Streamlit, NewsAPI, OpenAI API.

APPENDIX

A. Source code and supplementary materials

All source code and documentation are available at here.
Demo videos of the system: batch dashboard, real-time dashboard.

B. Useful commands

1. Docker

```
docker compose up --build -d
docker compose down
docker logs <service>
docker ps
docker compose restart <service>
```

2. Airflow

```
docker exec airflow airflow dags trigger batch_pipeline
docker exec airflow airflow dags list-runs -d batch_pipeline
```

3. HDFS

```
docker exec namenode hadoop fs -ls /crypto/yahoo
docker exec namenode hadoop fs -rm /crypto/yahoo/btcusdt/*
```

4. HBase

```
docker exec hbase hbase shell
list
scan 'crypto_prices', {LIMIT => 5}
```

5. MongoDB

```
docker exec -it mongodb mongosh
use crypto_batch
db.predictions.find().pretty()
```

6. Full System Reset

```
1. Stop all containers:
docker compose down

2. Delete all persistent volumes:
docker volume prune
# Press Y when prompted

# If you want to delete only project-specific volumes:
docker compose down -v

3. Rebuild all images:
docker compose build --no-cache

4. Start everything again:
docker compose up -d
```

C. API Endpoints

- GET /api/crypto/{symbol}
- GET /api/crypto/history/{symbol}
- GET /api/crypto/predictions/{symbol}
- POST /api/chatbot/ask