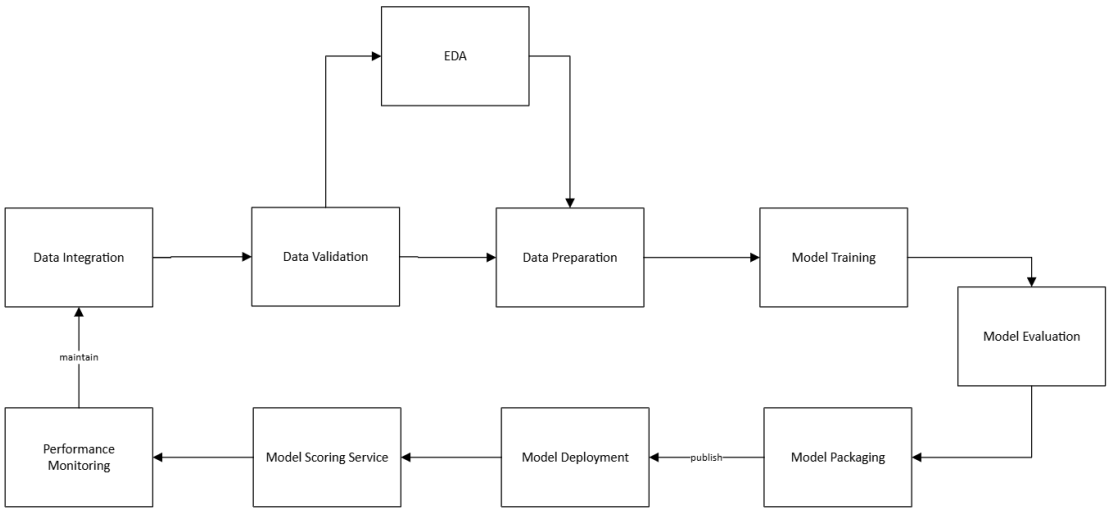
Challenge Extension: End-to-End System Design



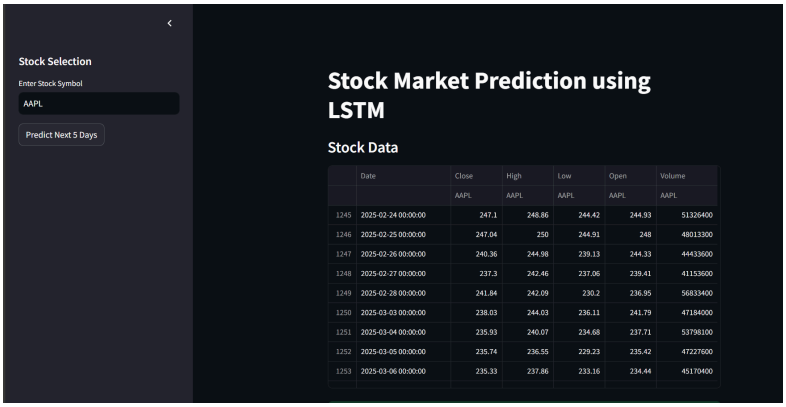
1. Data Collection and Ingestion

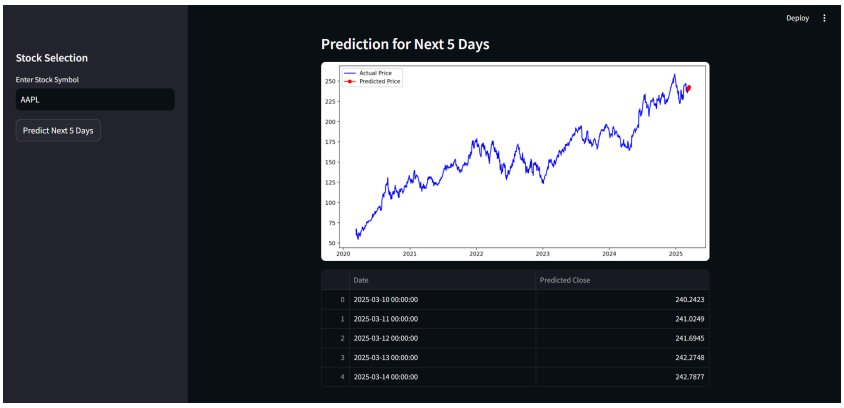
Accurate and consistent data collection forms the foundation of any stock price prediction model. This system collects both historical and real-time stock data using the Yahoo Finance API. Yahoo Finance was chosen because it provides reliable and free financial data with good coverage of both historical and real-time market data.

Why Yahoo Finance API Was Selected (for our Test End to End System):

- It offers free access to reliable financial data.
- It covers a wide range of stocks and historical data.
- It integrates well with Python-based systems, making it easy to set up and maintain.
- Other options like Alpha Vantage and Quandl were considered, but Yahoo Finance offered better data quality and no usage cost for basic use.

And This is Our Tested End to End System.(using streamlit)





Real-Time Data Handling:

- Real-time data collection is handled using **Kafka** as the message broker.
- Kafka was chosen over RabbitMQ because it supports higher throughput and better scalability.
- Kafka's ability to handle spikes in data flow ensures that the system can remain responsive even under heavy load.

Data Collection Process:

- Data is collected at regular intervals, including real-time updates.
- Key financial indicators collected: Open, High, Low, Close, and Volume.
- The focus is on the Close price because it has the highest predictive value.
- The data is automatically cleaned and stored in a structured format to maintain consistency.

2. Data Processing Pipeline

After data is collected, it is processed to remove inconsistencies and prepare it for model training.

Handling Missing Values:

- Missing values in the Close column were filled using a 5-day rolling mean.
- A shorter window (e.g., 3 days) caused the model to become unstable.
- A longer window (e.g., 10 days) missed short-term patterns, reducing accuracy.

- The 5-day window was selected because it provided the best balance between stability and capturing short-term trends.

Feature Scaling:

- The data was scaled using MinMaxScaler to keep values between 0 and 1.
- MinMaxScaler was selected because it preserves the original data distribution while ensuring that all values fit within a fixed range.
- LSTM models are sensitive to input scale, so using MinMaxScaler improved model convergence and accuracy.
- StandardScaler and RobustScale**r** were tested, but they distorted the data's range and reduced prediction accuracy.

Data Storage:

- Processed data is stored in a PostgreSQL database.
- PostgreSQL was selected because it supports time-series indexing and allows efficient storage and retrieval of large datasets.
- MySQL and MongoDB were considered but PostgreSQL provided better query performance and structured data handling.

3. Model Operations

The important thing of the system is the predictive model, which is designed to capture complex time-series patterns and generate accurate future stock price predictions.

Model Framework:

- The model is built using TensorFlow due to its strong support for deep learning and production-level deployment.
- TensorFlow was selected over PyTorch because it provides better support for GPU acceleration and integrates well with Kubernetes for deployment.

Model Architecture:

- The final model uses two LSTM layers:
    - First LSTM layer with 128 units and `return_sequences=True` to pass information to the next layer.
    - Second LSTM layer with 64 units and `return_sequences=False` to capture long-term dependencies.
    - A dense layer with 25 neurons for improved feature extraction.
    - A final dense layer with 1 neuron to predict the closing price.

Learning Rate and Optimization:

- The model is optimized using the Adam optimizer with a learning rate of 0.001.
- Adam was selected because it adjusts the learning rate dynamically, helping the model train faster and generalize better.
- RMSprop and ReduceLROnPlateau were tested, but they resulted in slower convergence and lower predictive accuracy.
- Adam provided the best balance between training speed and prediction accuracy.

Dropout and Batch Normalization:

- Dropout (0.2) and batch normalization layers were added to prevent overfitting.
- Dropout randomly disables neurons during training, improving model generalization.
- Batch normalization reduces internal covariate shifts and stabilizes the learning process.
- However, these layers increased training time without improving accuracy, so they were removed from the final model.

Deployment:

- The model is deployed using Kubernetes for automatic scaling and container-based deployment.
- Kubernetes was chosen over Docker Compose because it offers better load balancing and high availability.
- TensorFlow Serving was considered but lacked flexibility compared to Kubernetes.

Performance Monitoring:

- TensorBoard is used to monitor training loss, accuracy, and learning rate adjustments in real time.
- TensorBoard helps identify overfitting and instability early, making it easier to adjust the model if needed.

4. Insight Delivery

Predictions must be accessible and easy to understand for users to make informed decisions.

User Interface:

- Predictions are delivered through a dashboard built with Streamlit.
- Streamlit was chosen because it allows fast deployment and easy integration with TensorFlow models.

- The dashboard provides an interactive platform where users can select stock symbols and view predictions.

Data Visualization:

- Prediction results are plotted using Matplotlib and Seaborn.
- These libraries provide high-quality and customizable plots for better visualization.
- The dashboard shows both actual closing prices and predicted values for the next 5 days.

Real-Time Updates:

- The system uses WebSocket for real-time updates.
- WebSocket was chosen over HTTP because it supports full-duplex communication.

5. System Considerations

- **Scalability:** Kubernetes supports horizontal scaling to handle high user traffic.
- **Reliability:** Backup data sources like Alpha Vantage ensure data availability.
- **Latency:** GPU acceleration improves model inference speed and real-time responsiveness.
- **Cost:** Using AWS spot instances reduces GPU costs, and open-source tools like TensorFlow, Streamlit, and PostgreSQL minimize licensing fees.

6. Data Flow Explanation

The system follows a structured flow of data from collection to insight delivery:

- **Data Collection:** Stock data is collected using Yahoo Finance API and streamed using Kafka.
- **Data Processing:** Missing values are filled, and data is scaled using MinMaxScaler.
- **Storage:** Processed data is stored in PostgreSQL.
- **Model Prediction:** The last 60 days of data are fed into the LSTM model for predictions.
- **Insight Delivery:** Predictions are displayed on the Streamlit dashboard and updated in real time using WebSocket.

7. Challenge Analysis and Solutions

Developing and deploying the system involved several challenges:

- **Data Source Limitations:** Yahoo Finance has rate limits that may restrict data flow. Backup sources like Alpha Vantage are integrated to solve this.

- **Model Overfitting:** The model initially overfitted on training data. Dropout and batch normalization were added to reduce overfitting.
- **High Latency:** Model inference was slow under high traffic. GPU acceleration and increased batch size were used to improve response time.
- **Prediction Drift:** The model's accuracy decreased over time as market patterns changed. Regular model retraining helps maintain predictive accuracy.
- **Infrastructure Costs:** Running a GPU model continuously increases cost. Using AWS spot instances helps reduce costs.