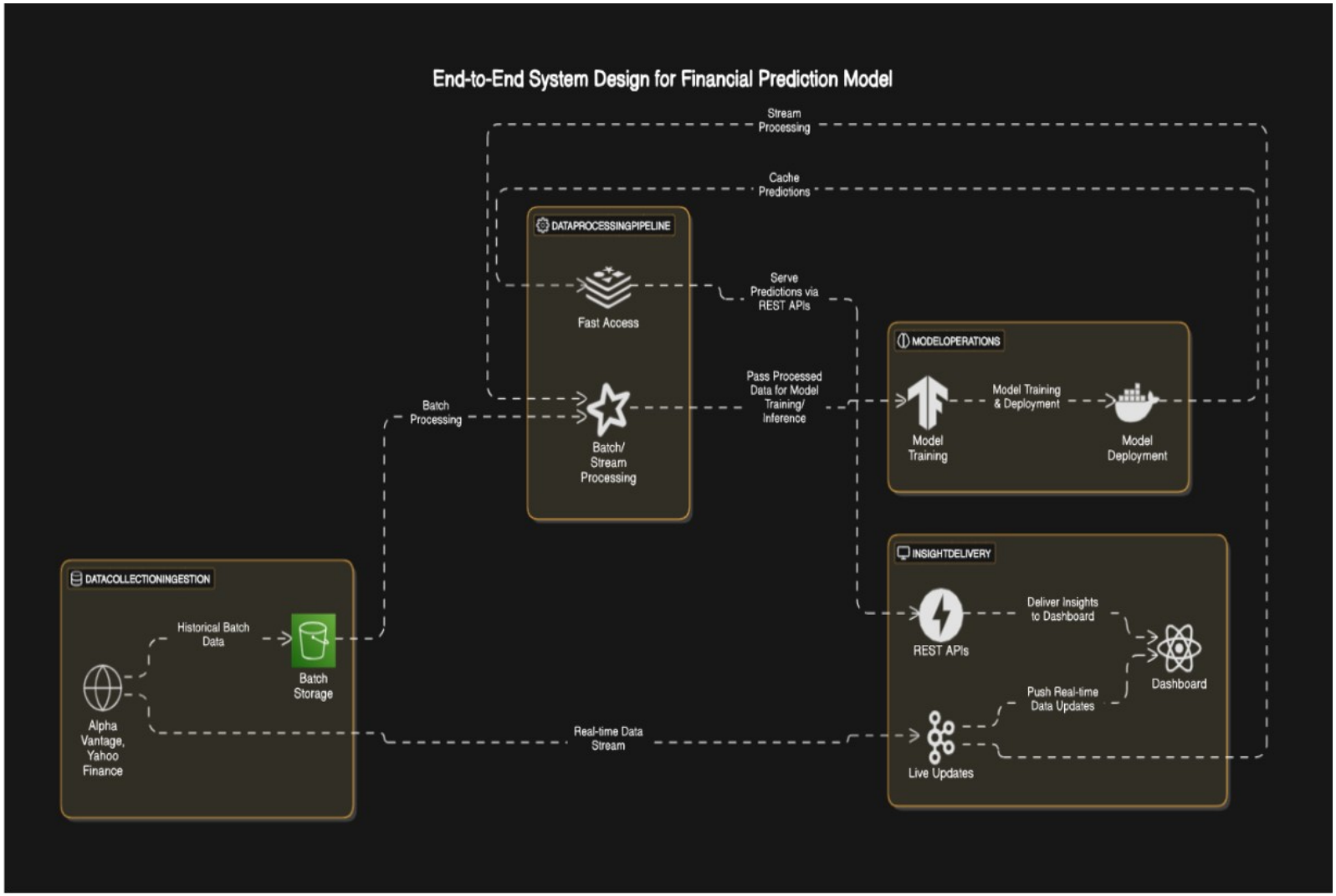
# Technical Report

## IntelliHack Task 04

**Project name:** Stock Price Prediction Challenge & Challenge Extension: End-to-End System Design

**GitHub Link:** https://github.com/KSDeshappriya/Intellihack_TetraNeurons_04.git

**Prepared by:** Team TetraNeurons


End-to-End System Design for Financial Prediction Model
Stream Processing
Cache Predictions
DATAPROCESSINGPIPELINE
Fast Access
Serve Predictions via REST APIs
Batch Processing
Batch/ Stream Processing
Pass Processed Data for Model Training/ Inference
MODELOPERATIONS
Model Training
Model Training & Deployment
Model Deployment
INSIGHTDELIVERY
REST APIs
Deliver Insights to Dashboard
Dashboard
Push Real-time Data Updates
Live Updates
DATACOLLECTIONINGESTION
Historical Batch Data
Batch Storage
Alpha Vantage, Yahoo Finance
Real-time Data Stream

# 1. Component Justification: Explanation of Technology Choices and Trade-offs

### 1.1 Data Collection & Ingestion

**Technologies**:

- **Financial APIs (Alpha Vantage, Yahoo Finance)**
- **AWS S3 (Batch Storage)**
- **Apache Kafka (Real-time Stream)**

**Why this choice?**

- **Financial APIs** provide easy access to historical and real-time market data, which is essential for building a financial prediction model.
- **AWS S3** is a reliable and scalable storage solution for large datasets, particularly for historical financial data, which is ingested in batches.
- **Apache Kafka** is chosen for real-time streaming due to its ability to handle large data volumes with low latency. This ensures that the financial prediction model can act on live market data as soon as it is available.

**Trade-offs**:

- **Financial APIs** may come with rate limits or data restrictions depending on the API provider, potentially limiting the amount of real-time data collected.
- **S3 storage** incurs storage costs, especially when handling large volumes of historical data.
- **Kafka** requires careful management to ensure the real-time stream is processed correctly, and scaling Kafka might require additional infrastructure.

### 1.2 Data Processing Pipeline

**Technologies**:

- **Apache Spark (Batch and Stream Processing)**
- **Redis (Cache)**

**Why this choice?**

- **Apache Spark** is well-suited for large-scale data processing, capable of handling both batch and streaming data. It allows us to process historical data (from S3) and real-time data (from Kafka) efficiently in parallel, ensuring that both streams can feed into the model without delays.
- **Redis** is chosen for caching processed data and model predictions. It helps reduce latency when serving predictions by storing frequently accessed data in memory.

**Trade-offs**:

- **Apache Spark** may require significant computational resources, which can add to the cost, particularly when processing large datasets in real-time.
- **Redis** operates in memory, which limits the amount of data it can store compared to traditional databases. It is critical to monitor memory usage and optimize caching strategies to prevent overflow or loss of important data.

### 1.3 Model Operations

**Technologies**:

- **TensorFlow (Model Training)**
- **TensorFlow Serving (Model Deployment)**

**Why this choice?**

- **TensorFlow** is a robust and widely used machine learning framework that supports both model training and inference. It is suitable for handling the complex patterns found in financial data.
- **TensorFlow Serving** is designed specifically for deploying machine learning models in production environments. It supports features like batching and versioning, which are essential for a scalable and efficient deployment pipeline.

**Trade-offs**:

- **TensorFlow** requires careful tuning of the model and extensive resources for training. Its complexity might require expertise, particularly in financial modeling.
- **TensorFlow Serving** may face scalability challenges when serving a high volume of requests unless auto-scaling and load balancing are properly implemented.

### 1.4 Insight Delivery

**Technologies**:

- **FastAPI (Serving Predictions)**
- **ReactJS (Dashboard)**
- **Kafka (Real-time Updates)**

**Why this choice?**

- **FastAPI** is known for its high performance in serving REST APIs and its ease of integration with machine learning models. It provides a fast and reliable mechanism to serve predictions to external clients.
- **ReactJS** is a popular framework for building responsive and interactive user interfaces. In this case, it would be used to build a dashboard for analysts to interact with real-time financial predictions and data.
- **Kafka** helps push real-time updates to the frontend, ensuring that the dashboard always reflects the latest data and predictions.

**Trade-offs**:

- **FastAPI** may require additional optimizations for high-traffic environments, especially when serving predictions at scale.
- **ReactJS** requires regular maintenance and testing to ensure the dashboard is responsive and user-friendly.
- **Kafka** needs careful management to avoid delays or loss of messages in real-time data updates.

## 2. Data Flow Explanation

1. **Ingestion**:
    - **Real-time data** is fetched from **financial APIs** and streamed into **Apache Kafka**.
    - **Historical data** is collected in **AWS S3** as a batch storage mechanism.
2. **Processing**:
    - **Batch Processing**: Historical data from **S3** is ingested into **Apache Spark** for cleaning and transformation before being passed to **TensorFlow** for model training.
    - **Stream Processing**: Real-time data from **Kafka** is processed immediately by **Apache Spark**, transformed, and passed to **TensorFlow** for inference.
    - Processed data is stored in **Redis** for fast access to predictions and frequently used data.
3. **Model Training**:
    - The **TensorFlow** model is trained on the processed data from both real-time and historical sources.
    - **TensorFlow Serving** is used for deploying the trained model and serving predictions for future inputs.
4. **Prediction Serving**:
    - **FastAPI** serves the model's predictions through REST APIs, allowing external applications to query the model.
    - Predictions are stored in **Redis** for faster retrieval on subsequent requests.
5. **Dashboard Updates**:
    - **ReactJS** displays the predictions and insights on an interactive dashboard.
    - **Kafka** is used to stream real-time updates to the dashboard, ensuring that it reflects the most recent market changes.

## 3. Challenge Analysis: Identify 3-5 Potential Challenges and Propose Mitigation Strategies

### 3.1 Real-time & Batch Data Synchronization

- **Challenge**: Ensuring that the real-time data from Kafka is properly aligned with the historical data processed from AWS S3. If synchronization isn't handled correctly, predictions might be based on incomplete or outdated information.
- **Solution**: Use a unified data processing pipeline where batch and real-time data are processed in parallel but stored together in a way that ensures consistency. Implement data versioning and time stamps to manage the flow of data correctly.

### 3.2 Model Drift

- **Challenge**: Over time, the financial model may degrade as market conditions change (known as model drift), leading to less accurate predictions.

- **Solution**: Implement a model monitoring system that tracks prediction accuracy and automatically triggers **model retraining** when performance degrades. Periodically retrain the model with the most recent data to maintain its accuracy.

### 3.3 API Scalability

- **Challenge**: The FastAPI service might face scaling challenges under heavy load, especially when serving predictions at scale.
- **Solution**: Implement **auto-scaling** for the FastAPI service and use **load balancing** to distribute traffic evenly. Caching predictions in **Redis** can also reduce API load and improve response times.

### 3.4 Data Security

- **Challenge**: Financial data is sensitive, and ensuring its privacy and security during transmission and storage is critical.
- **Solution**: Encrypt data in-transit (using HTTPS) and at-rest (using encryption in S3 and Redis). Implement strict **access control** policies to ensure only authorized personnel can access the data and model.

### 3.5 Real-time Data Latency

- **Challenge**: Minimizing the latency between the ingestion of real-time data and the delivery of predictions to users.
- **Solution**: Optimize the **Kafka** and **Spark** processing pipelines for low-latency data streaming. Use **Redis** to cache frequent predictions and reduce the time it takes to serve responses.