CONTENTS

1. Abstract

Our project develops a hybrid deep-learning system for predicting stock prices of the top ten Indian equities, combining traditional machine-learning strength with modern sequence modeling. Beginning with robust data acquisition, we leverage both public datasets from platforms such as Kaggle and real-time feeds via Yahoo Finance and the NSE API to assemble a comprehensive time-series corpus. The raw data undergoes rigorous preprocessing and cleaning through Python libraries like NumPy and Pandas, ensuring consistency and handling missing entries after that it goes to K-means Clustering algo before it flows into our anomaly detection module. In this stage, a Long Short-Term Memory (LSTM) network, trained during stable market periods, serves to flag unusual patterns; once reconstruction error exceeds predefined thresholds, potential market shocks are identified and managed to prevent distorted forecasts.

Building on stable data, our system employs Temporal Convolutional Networks (TCNs) to capture both short-term momentum and long-range trends in stock movements. These convolutional encoders transform raw price sequences into compact embeddings, effectively summarizing market dynamics in a form amenable to downstream learning. For our core regression engine, we replace traditional gradient-boosting implementations with LightGBM, capitalizing on its histogram-based tree growth and native handling of categorical features to achieve faster training times and competitive accuracy. To promote transparency, we integrate SHAP-based explainability, which illuminates feature contributions and empowers users to understand why the model favors certain signals over others. By structuring our pipeline in modular stages—data ingestion, anomaly filtering, embedding extraction, and boosted regression—we maintain clarity, ease of maintenance, and the potential for rapid iteration.

For user interaction and deployment, we adopt a full-Python stack that balances simplicity with scalability. The frontend interface is built with Streamlit, allowing us to craft intuitive dashboards and input controls using minimal code, while the backend prediction service is exposed through FastAPI, offering asynchronous, high-performance REST endpoints and auto-generated documentation via Pydantic. This separation of concerns ensures that our

prediction logic can be tested independently of the UI, and that new features—such as continuous online learning to adapt to concept drift or integration of alternative models like Random Forest, SVR, or KNN—can be slotted in with minimal disruption. The resulting proof-of-concept not only serves as a demonstrable base model for academic evaluation but also lays a clear roadmap for future.

2. List of Figures

3. Introduction

Our stock-price forecasting engine is engineered from the ground up to be a no-nonsense, modular pipeline that balances clarity, adaptability, and performance. Rather than blindly stacking complexity, we employ a pragmatic architecture that only integrates advanced techniques when they demonstrably improve results. At its core, the system follows a streamlined flow that begins with comprehensive data ingestion and culminates in interpretable, high-accuracy price forecasts. We start by pulling in the most exhaustive dataset possible. This includes multi-year historical price data, corporate actions such as splits and dividends, and alternative sources like macroeconomic indicators and sentiment data extracted from financial news and social media. Live tick data is continuously streamed through resilient, fault-tolerant APIs to ensure the model always has access to the freshest signals. All incoming data flows into a unified preprocessing layer that standardizes formats, handles missing ticks, and corrects for anomalies such as corporate actions. During this step, we compute a suite of engineered features like rolling window statistics—including moving averages, historical volatility, and annualized performance metrics—giving our models a richer feature space to learn from.

Once the data is cleaned and engineered, we apply regime clustering to segment the time series into distinct market phases. Using unsupervised learning techniques like K-means, we partition the data into "stable" and "turbulent" regimes. This step is crucial, as training on heterogeneous market behaviour introduces noise that can degrade model performance. By isolating calmer regimes from periods of high volatility, we enable downstream models to learn more meaningful patterns within each segment. The LSTM-based anomaly detector plays a vital role in this process. We deploy an autoencoder trained on data from stable regimes to learn what "normal" looks like. Any significant deviation in reconstruction error acts as a red flag, signalling a potential regime shift. This lets us quarantine outliers or erratic swings before they propagate downstream and corrupt our price prediction engine. This layer essentially acts as a firewall, protecting the model from tainted or non-stationary input.

The heart of the forecasting module lies in a Temporal Convolutional Network (TCN), which processes sequential data from each regime to extract both short-term fluctuations and long-term trends. The TCN's ability to maintain memory across long sequences without the vanishing gradient problem makes it ideal for time series with complex, multiscale dependencies. The output of the TCN is passed through a bottleneck to produce dense embeddings that capture the essence of market behaviour in each regime. These embeddings are then fed into a LightGBM regressor, which is responsible for generating the final

price forecasts. LightGBM is chosen not only for its speed and accuracy but also for its robustness to overfitting and its ability to handle high-dimensional input spaces with missing values and collinear features.

To make sure the system adapts to ever-changing market conditions, we incorporate an online learning component using SGDRegressor. As new data arrives, this lightweight model acts as a real-time tuner, incrementally updating parameters to reflect the latest market dynamics. It's akin to a pit crew making rapid adjustments between laps, keeping the forecasting engine optimally calibrated without having to retrain the entire pipeline. This ensures that the system remains responsive and doesn't become stale or obsolete.

Quality control is baked into every layer of this architecture. We rigorously validate each module using walk-forward back testing and nested cross-validation to simulate real-world deployment scenarios. Ablation studies are routinely conducted to assess the marginal utility of each component. If a module fails to deliver a measurable boost in predictive accuracy, it gets removed. This approach guarantees that every part of the pipeline earns its place. There's no room for academic bloat or theoretical dead weight—only components that provide a real, data-backed edge make the cut.

The final product is a lean, transparent, and maintainable system designed to deliver consistent, high-fidelity forecasts. It's modular enough to swap out algorithms or data sources with minimal disruption, and it's rigorous enough to withstand the scrutiny of production-level deployment in high-stakes financial environments. We've stripped away the fluff and focused on what works—no kitchen-sink overengineering, just a sharp, adaptive pipeline built to dominate the forecasting game with clarity, speed, and precision.

4. Scope of the Work

The scope of work for this project outlines the modules and activities we will undertake to build our initial proof-of-concept hybrid forecasting system, as well as the aspects we deliberately defer to later phases. Our focus at this stage is on end-to-end data pipelines, core model development, and a minimal user interface to demonstrate functionality and performance. We will document each step thoroughly to ensure reproducibility and clarity for evaluation, while leaving room for enhancements and additional features in the "Future Enhancement" section of the report.

In Scope

- Data Collection & Management
  - Assemble and version historical and near-real-time price series for the selected equities.
  - Design storage schemas and implement basic data quality checks and logging.
- Preprocessing & Cleaning
  - Handle missing or anomalous values through imputation and simple statistical filters.
  - Normalize and transform raw series into modeling-ready inputs.
- Clustering Stable market data
  - We will be using K-Means Clustering algo on the pre processed dataset to cluster the chunk of data where the market has good stability using the help of ARP & Rolling Average.
- Anomaly Detection Module
  - Develop a sequence-based filter that flags extreme deviations in stable periods.
  - Implement threshold-based logic to isolate potential market shocks.
- Feature Encoding & Embedding
  - Construct temporal encoders to summarize recent and long-range movements into fixed- length vectors.
  - Validate embedding quality via reconstruction or clustering diagnostics.
- Core Regression Model
  - Train and validate a gradient-boosting regressor as the primary predictor.
  - Optimize hyperparameters through systematic search and cross-validation.
  - Evaluate performance using back-testing metrics (RMSE, MAPE, directional accuracy).

- Model Explainability

  • Integrate a global feature-importance analysis to highlight key drivers.

  • Generate local explanations for individual predictions to support interpretability.
- Basic User Interface

  • Build a simple interactive dashboard allowing selection of stocks, date ranges, and prediction requests.

  • Display forecast results alongside historical price plots and explanation summaries.
- Testing & Documentation

  • Create unit tests for data pipelines and model inference routines.

  • Produce detailed process documentation and code comments.

While the above deliverables establish a fully functioning prototype, several advanced capabilities and auxiliary components fall outside our current timeline and resource allocation. These items will be addressed in subsequent project phases or marked as future enhancements.

Out of Scope

- Advanced Feature Engineering (e.g., alternative data sources such as sentiment or macro indicators)
- Ensemble & Stacking Architectures beyond the primary regressor
- Continuous Online Learning for real-time concept drift adaptation
- Custom Authentication or User Management in the dashboard
- High-Throughput Production Deployment (e.g., auto-scaling clusters, Kubernetes orchestration)
- Comprehensive Monitoring & Alerting Systems for live drift or performance degradation
- Multi-Page Web Application Frameworks or deep frontend customization

By clearly delineating what we will and will not cover in this initial phase, we ensure a focused effort on the core prediction pipeline and a lean, demonstrable system suitable for academic evaluation. Subsequent enhancements can then build upon this foundation without disrupting the integrity of our primary deliverables.

5.Survey of the previous works

Bansa**l** & et. Al(2022)

This paper provides a comparative analysis of five machine learning algorithms—KNN, Linear Regression, SVR, Decision Tree Regression, and LSTM—for stock price prediction. Using stock data from 12 Indian companies across a 7-year period, the study evaluates model performance using SMAPE, RMSE, and R^2 metrics. LSTM was found to significantly outperform traditional models, achieving the lowest average error rates. The paper emphasizes the limitations of conventional ML models in capturing the non-linear and temporal complexities of financial markets, advocating for a shift toward deep learning approaches like LSTM. The structured methodology of data preprocessing, model training, testing, and evaluation offered a practical baseline for our own model development and benchmarking.

Mehtarizadeh, Mansouri & et. al (2025)

This paper introduces a hybrid model that combines statistical and deep learning techniques—specifically, ARIMA-GARCH for trend decomposition and SCA-optimized LSTM for residual forecasting. The Sine-Cosine Algorithm (SCA) is used to fine-tune hyperparameters, significantly improving prediction accuracy over traditional LSTM and other hybrid variants. The model demonstrates notable gains in RMSE and R^2 metrics across multiple datasets. However, it also comes with high computational overhead and requires rigorous preprocessing. This paper influenced our decision to explore hybrid models that leverage both statistical and deep learning strengths, particularly in cases involving non-stationary and highly volatile market-data

Siyi Li and Sijie Xu (2024)

In this work, the authors combine GANs, GRU-based sequence modeling, Transformer-based attention, and Fin-BERT sentiment analysis to build a highly intelligent stock prediction framework. Variational Autoencoders (VAE) are used for feature denoising and compression, while Bayesian Optimization fine-tunes the architecture. The model achieves strong predictive performance across various tech stocks, although generalizability to other sectors remains a challenge. This paper influenced our use of attention mechanisms and the incorporation of sentiment data, guiding us toward more sophisticated and interpretable deep learning designs that considers both numerical & Textual Data

Yingcheng Xu, Yunfeng Zhang & et. al (2024)

This study proposes a nonlinear fusion architecture that integrates ARIMA-based denoising, ACNN for feature extraction, BiLSTM for sequence modeling, and a CNN-based discriminator within a GAN framework. Applied to Bank of China stock data, the model outperforms several benchmarks with impressive accuracy metrics like MAE, RMSE, and R^2. The paper highlights the effectiveness of combining traditional statistical tools with advanced deep learning components

6. Workflow diagram

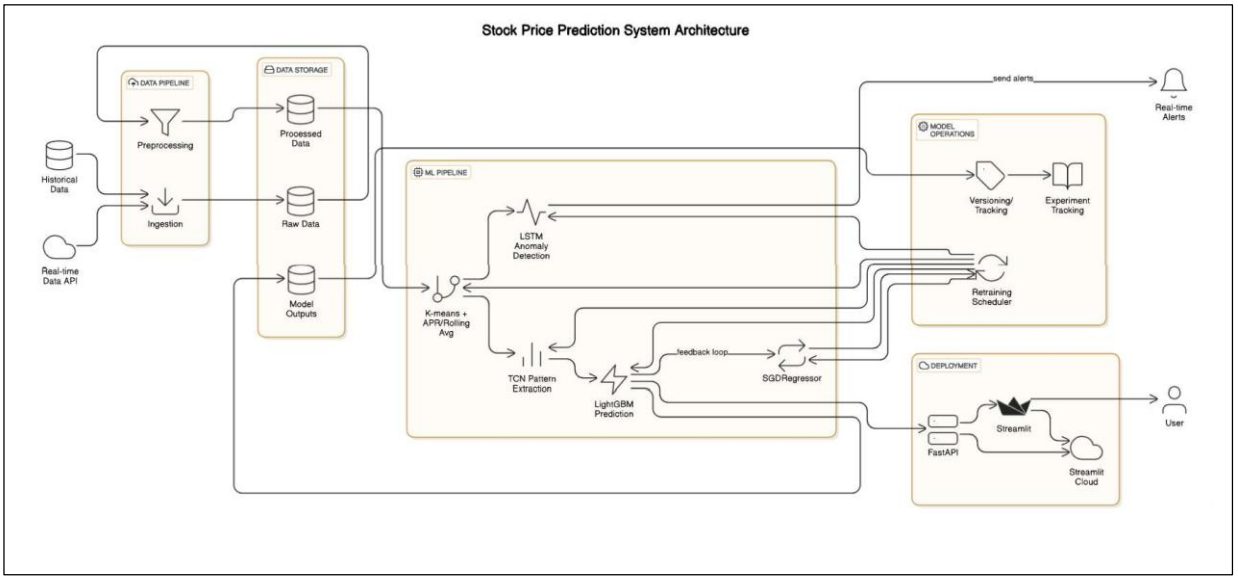

Figure 5.1 - Proposed Model Architecture

Our workflow executes an end-to-end hybrid stock-price prediction pipeline in six stages, each leveraging specialized technologies to ensure modularity, which can be seen in **Diagram -1** performance, and interpretability. We begin by collecting and versioning multi-source time series data, which lays the foundation for reproducible experiments. Next, we rigorously preprocess and clean the raw series to eliminate noise and prepare features, followed by an LSTM-based autoencoder that detects and excludes anomalous market shocks. A Temporal Convolutional Network then compresses cleaned sequences into fixed-length embeddings that capture both momentum and trend. These embeddings feed into a LightGBM regressor, whose predictions are interpreted via SHAP values. Finally, we expose our model through FastAPI and build a Streamlit dashboard, completing a full-Python stack for demonstration and future extensions.

Steps of to successfully execute the whole project:

1. Data_Collection

During the training phase, we batch-download and archive historical price series for selected stocks to build a comprehensive dataset that captures a wide range of market conditions. This ensures our models are trained on both stable and volatile periods before fine-tuning. Daily and intraday OHLCV data is fetched from multiple APIs including Upstox, Alpha Vantage, Yahoo Finance, NSE, BSE, and Breeze. The collected data is merged and aligned by timestamp, with missing values handled through forward or backward imputation. Cleaned time series are partitioned and stored in Parquet files, organized by date and ticker. Metadata such as source, fetch time, and row count is recorded for reproducibility and version control. Data integrity is validated using checks like monotonic timestamps and non-negative volumes.

In the deployment phase, the system continuously ingests live data in small increments, enabling real-time forecasting and adaptive model updates. The live stream mirrors the training setup but focuses exclusively on the most recent market information. API endpoints are polled at regular intervals—such as every minute or hour—and each new batch is preprocessed on the fly. This includes handling missing ticks, clipping extreme spikes, and applying the same scaling used during training. Fresh records are appended to a rolling window buffer to form the input sequence for inference. Incoming data and fetch errors are logged for monitoring and audit purposes, and the prediction pipeline is triggered immediately after each buffer update to deliver up-to-date forecasts.

2 . Data Preprocessing

During the training phase, data preprocessing plays a critical role in ensuring that historical financial data is clean, consistent, and suitable for model development. This involves multiple steps, starting with data cleaning to identify and handle missing or null values, ensuring the dataset is complete and accurate. Outlier detection and removal follow, addressing anomalies that could skew the model's learning process. Data transformation is performed to convert raw inputs into suitable formats, such as converting timestamps into datetime objects or encoding categorical variables. Feature scaling is then applied to normalize or standardize numerical features so that each one contributes equally to the

model's training. Finally, the dataset is split into training, validation, and testing sets to properly evaluate the model's performance and generalizability.

In a real-time deployment scenario, preprocessing must be both efficient and capable of handling streaming data to enable timely predictions. Real-time data ingestion is performed continuously from live sources with minimal latency. On-the-fly data cleaning is applied immediately as data arrives to correct inconsistencies or fill gaps, preserving data integrity. Real-time transformation is also carried out to ensure the data is formatted appropriately, such as converting values or encoding variables. Incremental feature scaling is used to adapt to new data dynamically without needing to retrain the entire model. Stream processing frameworks are utilized to manage data in motion, supporting real-time analytics and fast decision-making.

3.Clustering the Data

In the clustering stage, we'll isolate "stable" market regimes by grouping time-series observations based on summary statistics (e.g., annualized percentage return, rolling mean, rolling volatility) using K-means. First, we compute our features over a fixed window and scale them to unit variance. Then we run K-means—choosing the number of clusters via the elbow method or silhouette scores—to carve the data into distinct regimes. Finally, to validate cluster quality and assign a "stable" label to the low-volatility, steady-return clusters that feed into our LSTM anomaly detector.

4. Anomaly Detection & Refinement

During the training phase, anomaly detection is essential to identify and exclude irregular data points that could distort the model's learning process. This ensures the model is trained only on data that accurately represents typical market conditions. The process begins with data segmentation, where historical time series data is divided into manageable sequences for analysis. An unsupervised model, such as an LSTM autoencoder, is then trained on these sequences to learn the normal patterns in the data. After training, the model is used to reconstruct the input sequences, and the reconstruction error—i.e., the difference between the original and reconstructed data—is calculated. Based on these errors, anomaly scores are

assigned, with higher errors indicating potential anomalies. A threshold is then set for anomaly detection, often based on a number of standard deviations above the mean error. Data points that exceed this threshold are flagged as anomalies and excluded from the training dataset to avoid negatively influencing the model.

In a real-time deployment environment, anomaly detection must operate efficiently and handle streaming data to quickly identify irregularities. The system continuously monitors incoming data streams, applying the trained anomaly detection model to calculate real-time anomaly scores. To remain adaptive to evolving data patterns, the anomaly detection threshold is adjusted dynamically based on recent data behavior. When an anomaly is detected, immediate responses—such as alerts or automated mitigations—are triggered to address potential issues swiftly. A feedback loop is also integrated to refine the model based on newly detected anomalies and incoming data, thereby enhancing its accuracy and robustness over time.

5. Temporal Embeddings Extraction

Temporal embedding extraction is a pivotal component of the project, transforming sequential financial data into fixed-length vector representations that capture temporal dynamics. This process enhances the model's ability to learn from time-based patterns, improving predictive accuracy. During the training phase, historical time-series data is segmented into fixed-length sequences to maintain consistency in input dimensions. These sequences are processed using a Temporal Convolutional Network (TCN), which effectively models temporal dependencies. The TCN generates fixed-length embeddings that encapsulate the temporal characteristics of each sequence. These embeddings are then normalized to ensure uniformity and stored alongside the original sequences for subsequent model training and evaluation.

In a real-time deployment environment, temporal embeddings are generated on-the-fly from live financial data to enable immediate predictions. Incoming data is continuously ingested and buffered into sequences that match the training format. The same TCN model, trained offline, processes the buffered sequences to produce real-time embeddings. These embeddings are directly fed into the predictive model to generate up-to-date forecasts. The

system also monitors the embedding outputs for anomalies or shifts in data patterns. If significant drifts are detected, retraining or model adjustment procedures are triggered to maintain accuracy and reliability.

Following embedding extraction, the core regression and explainability phase begins. In the training phase, cleaned and scaled feature matrices are combined with the generated TCN embeddings into a comprehensive training table. This table is then converted into LightGBM's internal dataset format to optimize performance. Key parameters are configured, including the regression objective, RMSE as the evaluation metric, and hyperparameters such as the number of leaves, learning rate, and tree depth. The model is trained using early stopping based on a validation split to avoid overfitting. Once trained, the model is serialized to disk in binary format for efficient loading during deployment. Additionally, global feature importance scores are extracted from the model to support interpretation and transparency.

In real-time deployment, the pre-trained LightGBM model is deserialized into memory at service startup. Incoming feature and embedding vectors are assembled and formatted identically to the training input. The model's predict function is invoked to generate forecasts in milliseconds. If needed, post-processing steps like inverse transformations are applied to convert predictions back to original price units. Each prediction request and its result are logged for audit purposes, while system performance is monitored to track latency and error rates, ensuring compliance with service-level agreements.

6.Deployment & testing

The deployment process for the hybrid stock-prediction app begins by organizing the monorepo structure. A single GitHub repository is created containing two subfolders: /frontend, which houses the Streamlit application, and /backend, which contains the FastAPI service. For the frontend, a new Web Service is set up in Render by connecting to the repository and specifying /frontend as the root directory. The build command is set as pip install -r requirements.txt, and the start command is defined as streamlit run app.py --server.port $PORT.

Similarly, the backend service is configured in Render by adding another Web Service and pointing it to the /backend directory. The build and start commands mirror those of the frontend, with the backend using uvicorn main:app --host 0.0.0.0 --port $PORT. Monorepo deployment filters are enabled in the settings of each service, allowing only relevant changes in respective file paths to trigger rebuilds. Environment variables and secrets—such as MODEL_PATH, API keys, or database URIs—are added under each service's Environment tab in the Render dashboard to secure sensitive configurations.

To enable seamless interaction between the frontend and backend, CORS (Cross-Origin Resource Sharing) is configured in FastAPI to allow requests originating from the Streamlit domain. After deploying both services, end-to-end connectivity is tested to ensure that the Streamlit frontend can successfully send POST requests to the FastAPI /predict endpoint and display the returned results. Finally, system performance is monitored using Render's built-in metrics and logs. These tools help track CPU usage, memory consumption, and response times, enabling dynamic scaling of service instance sizes or replicas as needed.

7.Continuous Learning

The algorithm used for continuous learning in our system is **SGDRegressor**. Continuous learning is not part of the offline training pipeline; instead, it is initiated only after the model goes live and begins serving predictions. In our workflow, this process is triggered immediately following the "API & Dashboard Deployment" phase, and it becomes an integral part of the ongoing "Monitoring & Maintenance" stage. At this point, the system continuously monitors incoming data and the performance of predictions, actively detecting any drift in inputs or outputs. Recent data is buffered in real time for potential retraining, and governance rules are applied to determine whether the model should be fine-tuned or fully retrained. If improvements are validated through testing, updated models are rolled out seamlessly without disrupting service. This post-deployment continuous learning mechanism ensures that the model adapts and evolves in response to changing market conditions.

7.Algortihms Used

This section outlines the core algorithms integrated into our hybrid stock price prediction system, highlighting their individual contributions and providing concise code references.

- Temporal Convovnutional Network(TCN)

We are using the Temporal Convolutional Network (TCN) in our stock price prediction project because it effectively captures temporal dependencies in sequential financial data. Unlike traditional RNNs or LSTMs, TCNs offer faster training, parallel processing, and better long-range memory, which are critical for modeling historical market trends. Their causal convolution structure ensures that predictions at any point are based only on past data, aligning perfectly with the nature of time-series forecasting. This makes TCNs a robust and efficient choice for generating temporal embeddings that enhance the accuracy of our predictive models.

Algorithm

INPUT: $B \in \mathbb{Z}^{n \times d}, \delta$;
OUTPUT: An LLL-reduced basis B;
[Gram-Schmidt] Compute Γ and $\|b_i^*\|, i = 1, \ldots, n$;
[Size Reduction] **SizeReduceBasis**(B,Γ,n,1);
Set $k = 2$;
[LLL iterations] WHILE $k \leq n$ DO
    IF $\|b_k^*\|^2 < (\delta - \Gamma_{k-1,k}^2)\|b_{k-1}^*\|^2$ THEN
        Swap $(B, \Gamma, k, k-1)$;
        SizeReduceBasis (B, Γ, n, k);
        IF $k > 2$ THEN $k \leftarrow k - 1$;
    ELSE
        $k \leftarrow k + 1$;

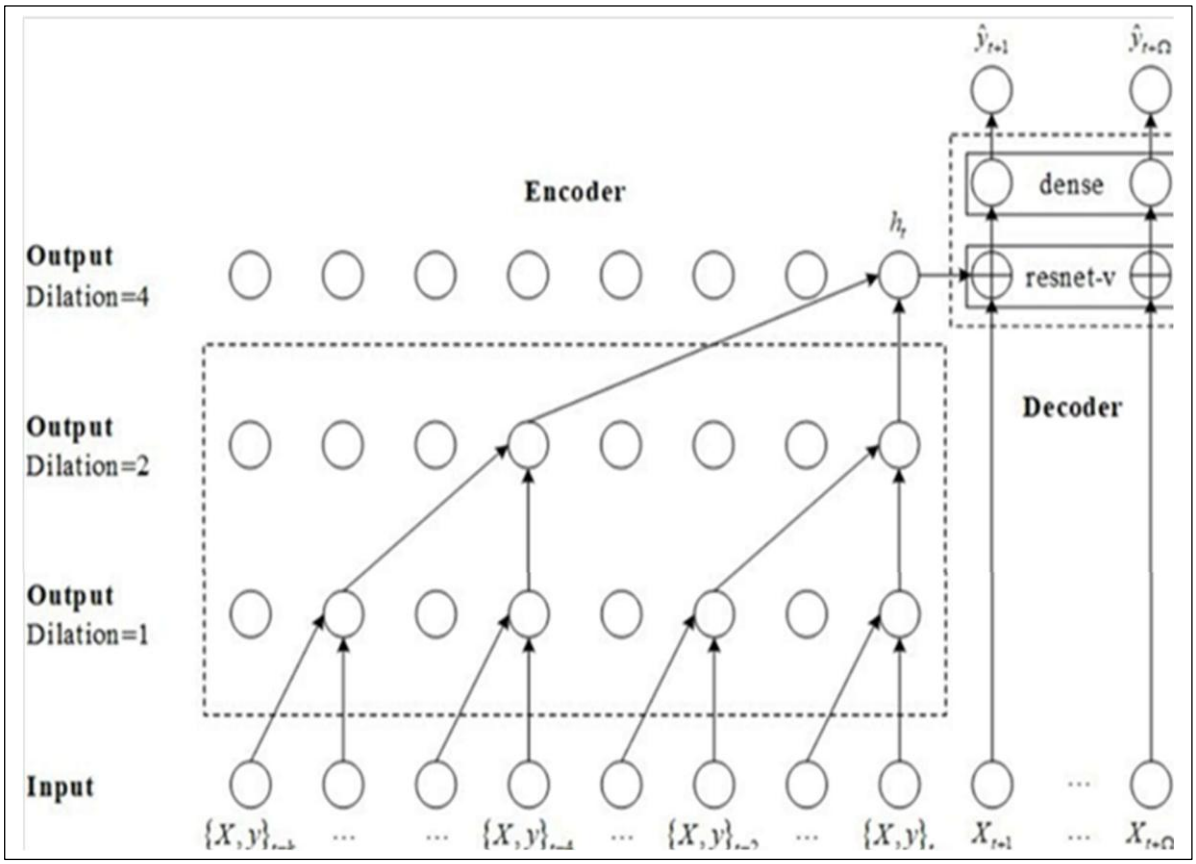

Figure 5.2 – TCN Architecture

- **LightBGM**

We are using the **LightGBM** algorithm in our stock price prediction project due to its **high efficiency, scalability, and accuracy** in handling structured, tabular data. LightGBM excels at capturing non-linear **relationships** between features and is optimized for speed and memory usage, which is crucial for real-time financial forecasting. Its **gradient boosting framework** allows for robust model performance, even with large datasets and high-dimensional feature spaces. Additionally, LightGBM provides **built-in support for feature importance**, which enhances model explainability—an important aspect when making financial decisions.

Algortihm

The LightGBM algorithm
Input:
Training data: D = {(χ1, y1), (χ2, y2), ..., (χN, yN)}, χi ∈χ, χ ⊆ R,
yi∈{−1,+1}; loss function: L(y, θ (χ));
Iterations:
M; Big gradient data sampling ratio: a; slight gradient data sampling ratio: b;
1: Combine features that are mutually exclusive (i.e., features never simultaneously accept nonzero values) of χi, i = {1, ...,N} by the exclusive feature bundling (EFB) technique;
2: Set $\theta_0(\chi) = arg\ min_c \sum_i^N L(y_i, c)$;
3: For m = 1 to M do
4: Calculate gradient absolute values:

$$r_i = \left| \frac{\partial L(y_i, \theta(x_i))}{\partial \theta(x_i)} \right|_{\theta(x)=\theta_{m-1}(x)}, i = \{1, \ldots, N\}$$

5: Resample data set using gradient-based one-side sampling (GOSS) process:
$topN = a \times len(D)$; $randN = b \times len(D)$;
$sorted = GetSortedIndices(abs(r))$;
$A = sorted[\ 1 : topN]$; $B = RandomPick(sorted[\ topN : len(D)]\ , randN)$;
$\hat{D} = A + B$;
6: Calculate information gains:

$$V_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} r_i + \frac{1-a}{b}\sum_{x_i \in B_l} r_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} r_i + \frac{1-a}{b}\sum_{x_i \in B_r} r_i\right)^2}{n_r^j(d)} \right)$$

7: Develop a new decision tree $\theta_m(x)'$ on set D'
8: Update $\theta_m(\chi) = \theta_{m-1}(\chi) + \theta_m(\chi)$
9: End for
10: Return $\tilde{\theta}(x) = \theta_M(x)$

- K-Means Clustering

We are using the **K-Nearest Neighbors (KNN)** algorithm in our stock price prediction project as a **simple yet effective baseline** for comparison and short-term pattern recognition. KNN makes predictions based on the **similarity of recent data points** to historical patterns, which is useful in capturing **local trends** in stock movements. It is a **non-parametric and instance-based** learning method, meaning it makes no assumptions about the underlying data distribution and adapts naturally to new data. Although not as scalable as other models, KNN offers **interpretability and ease of implementation**, making it valuable for benchmarking and exploratory analysis in our system.

Algorithm

```
for i ← 1 to n do Visited [i] ← false
Initialize the list Path with s
Visited [s] ← true
Current ← s
for i ← 2 to n do
    Find the lowest element in row current and unmarked column j containing the
    element.
    Current ← j
    Visited[j] ← true
    Add j to the end of list Path
Add s to the end of list Path
return Path
```

- LSTM(Long Short Term Memory)

We are using the Long Short-Term Memory (LSTM) model in our stock price prediction project because it is specifically designed to learn from sequential and time-dependent **data**. LSTM networks excel at capturing long-term dependencies and handling the vanishing gradient problem better than traditional RNNs, making them well-suited for modeling financial time series. Their ability to retain relevant information over extended sequences allows for more accurate forecasting of complex stock price patterns. LSTMs are particularly valuable when market behavior depends on trends or signals spread across time, offering deeper insights and **improved prediction stability** in dynamic market environments.

Algortihm

1	Input: Previous samples and present input is given to LSTM
2	Randomly initialize the weights (W's) and bias (b's) values
3	The forget gate: $f_t = \text{Sigmoid}\ \{\sum_{t=1}^{N} w_f h_{t-1} + x_t + b_f\}$
4	The input gate: $I_t = \text{sigmoid}\ \{\sum_{t=1}^{N} w_i h_{t-1}\, x_t w_i + b_i\}$
5	The candidate value: $\tilde{ct} = \text{Tanh}\ \{\sum_{t=1}^{N} w_c h_{t-1} + w_c x_t + b_c\}$
6	Cell state is updated c_{t-1} to c_t
7	Cell state $c_t = \{c_{t-1} * f_t\} + \{i_t * \tilde{ct}\}$
8	LSTM is updated by Output gate: $O_t = \text{Sigmoid}\ \{\sum_{t=1}^{N} w_o h_{t-1} + w_o x_t + b_o\}$
9	Output: Estimated next cell state c_{t+1}

working of the algorithm

LSTM architectures involves the memory cell which is controlled by three gates: the *input gate*, the *forget gate* and the *output gate*. These gates decide what information to add to, remove from and output from the memory cell. Here is a Diagram of it architecture.

- **Input gate**: Controls what information is added to the memory cell.

- **Forget gate**: Determines what information is removed from the memory cell.
- **Output gate**: Controls what information is output from the memory cell.

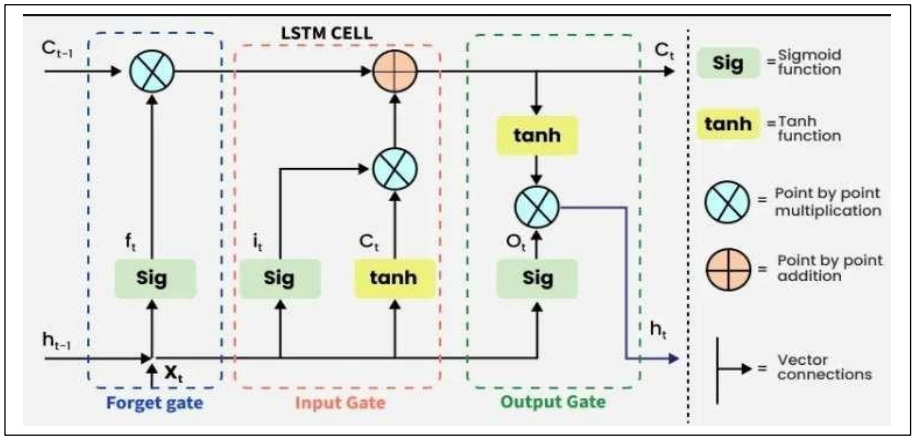

Figure 5.3–_LSTM Architecture

- Forget Gate The information that is no longer useful in the cell state is removed with the forget gate.
  - Equation Used = $ft=\sigma(Wf\cdot[ht-1,xt]+bf)$
- Input Gate The addition of useful information to the cell state is done by the input gate
  - Equation Used = 2 equations used here
    - $it=\sigma(Wi\cdot[ht-1,xt]+bi)$ and
    - $C\hat{}t=\tanh(Wc\cdot[ht-1,xt]+bc)C\hat{}t=tanh(Wc\cdot[ht-1,xt]+bc)$
- Output Gate The task of extracting useful information from the current cell state to be presented as output is done by the output gate
  - Equation used = $ot=\sigma(Wo\cdot[ht-1,xt]+bo)$

8. Hardware & Software Details

a. **Additional Hardware (if required)**

To support efficient model training, real-time inference, and data storage, the following hardware components may be a "good to go".

- High-Performance GPU
  - **Name**: NVIDIA GeForce RTX 3090
  - **Purpose**: Accelerate deep-learning model training (LSTM, TCN) and embedding extraction
  - **Specification**: 24 GB GDDR6X VRAM, 10,496 CUDA cores, 936 GB/s memory bandwidth
  - **Model Built**: Founders Edition (or equivalent aftermarket)
  - **Price (Approx.)**: ₹1,40,000 – ₹1,60,000
- Multi-Core CPU Server
  - **Name**: Intel Xeon Silver 4214R-based Workstation
  - **Purpose**: Host FastAPI backend, handle data preprocessing pipelines, and coordinate ensemble training
  - **Specification**: 12 cores / 24 threads @ 2.4 GHz base (3.5 GHz turbo), 64 GB DDR4 ECC RAM, dual 1 TB NVMe SSD
  - **Model Built**: Dell PowerEdge T140 (configured)
  - **Price (Approx.)**: ₹1,50,000 – ₹1,80,000
- High-Capacity SSD Storage
  - **Name**: Samsung 970 EVO Plus 2 TB NVMe SSD
  - **Purpose**: Store historical datasets, model checkpoints, and backtest results with low I/O latency
  - **Specification**: Read/write speeds up to 3,500 / 3,300 MB/s

        - **Model Built**: MZ-V7S2T0BW
        - **Price (Approx.)**: ₹18,000 – ₹22,000
    - Optional Network Attached Storage (NAS)
        - **Name**: Synology DS920+
        - **Purpose**: Centralized team data repository and automated backups for reproducibility
        - **Specification**: 4-bay chassis, Intel Celeron J4125, 4 GB DDR4 (expandable), supports up to 64 TB RAID
        - **Model Built**: DS920+ with four HDD slots
        - **Price (Approx.)**: ₹50,000 (chassis) + HDDs (~₹12,000 × 4)

b. **Software requirements:**

Below is the complete list of software packages, frameworks, and tools required to develop, train, deploy, and maintain our hybrid stock-price prediction system.

- **Python 3.8+**
    - **Purpose:** Core programming language for data processing, model development, API, and UI
    - **License:** Open Source (PSF License)
    - **Link:** https://www.python.org/downloads/
- **NumPy**
    - **Purpose:** Numerical computing, array operations, and linear algebra foundations
    - **License:** Open Source (BSD)
    - **Link:** https://numpy.org/
- **Pandas**
    - **Purpose:** Data manipulation, cleaning, and transformation of time-series
    - **License:** Open Source (BSD)
    - **Link:** https://pandas.pydata.org/
- **scikit-learn**
    - **Purpose:** Traditional ML algorithms (e.g., Random Forest, SVR,

kNN), preprocessing, model evaluation

- **License:** Open Source (BSD)
- **Link:** https://scikit-learn.org/

- **LightGBM**
  - **Purpose:** High-performance gradient-boosting regressor for core predictions
  - **License:** Open Source (MIT)
  - **Link:** https://github.com/Microsoft/LightGBM
- **TensorFlow** *(or PyTorch)*
  - **Purpose:** Build and train deep-learning modules (LSTM autoencoder, TCN)
  - **License:** Open Source (Apache 2.0)
  - **Link TensorFlow:** https://www.tensorflow.org/
  - **Link PyTorch:** https://pytorch.org/
- **SHAP**
  - **Purpose:** Explainability framework to compute feature-importance (global and local)
  - **License:** Open Source (MIT)
  - **Link:** https://github.com/slundberg/shap
- **FastAPI**
  - **Purpose:** High-performance ASGI web framework for serving prediction APIs
  - **License:** Open Source (MIT)
  - **Link:** https://fastapi.tiangolo.com/
- **Uvicorn**
  - **Purpose:** ASGI server for running FastAPI applications in production
  - **License:** Open Source (BSD)
  - **Link:** https://www.uvicorn.org/
- **Streamlit**
  - **Purpose:** Rapid development of interactive dashboards as the project frontend
  - **License:** Open Source (Apache 2.0)
  - **Link:** https://streamlit.io/
- **yfinance**
  - **Purpose:** Python wrapper for Yahoo Finance API to fetch historical and

real- time price data

- **License:** Open Source (MIT)
- **Link:** https://pypi.org/project/yfinance/

- **nsepy**
  - **Purpose:** API client for retrieving Indian stock data from NSE
  - **License:** Open Source (MIT)
  - **Link:** https://pypi.org/project/nsepy/
- **Jupyter Notebook / JupyterLab**
  - **Purpose:** Interactive environment for exploratory analysis and prototyping
  - **License:** Open Source (Modified BSD)
  - **Link:** https://jupyter.org/
- **Git**
  - **Purpose:** Version control for source code, collaboration, and CI/CD integration
  - **License:** Open Source (GPL v2)
  - **Link:** https://git-scm.com/
- **GitHub / GitLab**
  - **Purpose:** Remote code hosting, issue tracking, and CI/CD pipelines (e.g., GitHub Actions)
  - **License:** Free tier available; Public repos open source
  - **Link GitHub:** https://github.com/
  - **Link GitLab:** https://gitlab.com/
- **Render.com**
  - **Purpose:** Git-driven cloud platform for hosting web services and static sites
  - **License:** Freemium (proprietary platform with free tier)
  - **Link:** **https://render.com/**
- **Visual Studio Code** *(optional IDE)*
  - **Purpose:** Code editor with rich extensions for Python, Docker, and Git integration
  - **License:** Free (MIT)
  - **Link:** https://code.visualstudio.com/.

c. **Dataset to be used :**

This hardware setup ensures that both compute-intensive training and low-latency inference are handled smoothly, with room for future scaling and data redundancy.

1. **<u>Dataset – I</u>**
    a. **Source of the data : YahooFinance API**
    b. **Type of the data : Time series dataset**
    c. **Number of features : 9 [Date, Symbol, Series, Prev Close, Open, High, Low, Last, Close]**
    d. **Glimpse of the data :**

Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close
2021-04-26	ASIANPAINT	EQ	2517.95	2530.0	2575.00	2530.00	2558.0	2557.90
2021-04-27	ASIANPAINT	EQ	2557.90	2545.0	2579.90	2534.00	2571.0	2574.35

2. **<u>Dataset – III</u>**
    a. **Source of the data : Kaggle**
    b. **Type of the data : Historical dataset**
    c. **Number of features : 10 [Date, Symbol, Series, Prev Close, Open, High, Low, Last, Close, VWAP]**
    d. **Glimpse of the data :**

Date Date	Symbol Symbol	Series Type of security	Prev Close Previous day's close price	Open Open price of day	High Highest price in day	Low Lowest price in day	Last Last traded price in day	Close Close price of day	VWAP Volume Weighted Average Price
2007-11-27 2021-04-30	ADANIPORTS 69% MUNDRAPORT 31%	1 unique value	108 1.31k	108 1.31k	110 1.32k	106 1.27k	108 1.31k	108 1.31k	108 1.3k
2007-11-27	MUNDRAPORT	EQ	440.0	770.0	1050.0	770.0	959.0	962.9	984.72
2007-11-28	MUNDRAPORT	EQ	962.9	984.0	990.0	874.0	885.0	893.9	941.38
2007-11-29	MUNDRAPORT	EQ	893.9	909.0	914.75	841.0	887.0	884.2	888.09
2007-11-30	MUNDRAPORT	EQ	884.2	890.0	958.0	890.0	929.0	921.55	929.17
2007-12-03	MUNDRAPORT	EQ	921.55	939.75	995.0	922.0	980.0	969.3	965.65

3. **Dataset – IV**
    a. **Souce of the data : Quandl**
    b. **Type of the data : historical**
    c. **Number of total features : 9**
    d. **Glimpse of the data :**

Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close
2021-04-26	TCS	EQ	3109.50	3102.05	3153.00	3088.05	3100.05	3100.80
2021-04-27	TCS	EQ	3100.80	3106.00	3136.10	3103.00	3132.75	3132.00

4. **Dataset – V**
    a. **Source of the data : Breeze API for ICICI Bank**
    b. **Type of the data : Time-Series Data**
    c. **Number of total features : 9 [Date, Symbol, Series, Prev Close, Open, High, Low, Last, Close]**
    d. **Glimpse of the data**

Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close
2021-04-26	ICICIBANK	EQ	569.95	602.00	605.50	588.0	588.7	591.10
2021-04-27	ICICIBANK	EQ	591.10	593.25	601.95	591.1	599.0	598.75

5. **Dataset – VI**
    a. **Source of the data : BSE API**
    b. **Type of the data : Time Series**

c. **Number of total features : 9 [Date, Symbol, Series, Prev Close, Open, High, Low, Last, Close]**

d. **Glimpse of the data :**

Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close
2021-04-26	TATASTEEL	EQ	925.60	935.0	956.00	930.05	942.5	940.75
2021-04-27	TATASTEEL	EQ	940.75	948.3	983.00	944.30	982.0	977.75

9. Roadmap for Future Work

Semester	**Proposed Work**
7th Sem	1. **Enhanced Anomaly & Statistical Filtering** – Combine LSTM autoencoder with rolling z- score and EWMA to flag regime shifts

	dynamically. 2. **ARIMA-GARCH Residual Modeling** – Fit ARIMA-GARCH on detrended series to capture heteroskedasticity and integrate residual forecasts with our LightGBM outputs. 3. **Transformer Embeddings Prototype** – Implement a mini-Transformers module for sequence encoding; compare against TCN on key metrics. 4. **Sentiment Data Ingestion** – Build pipelines to fetch and preprocess news sentiment via FinBERT; design feature-fusion tests with existing technicals.

8th Sem	1. **GAN-LSTM Hybrid Architecture** – Develop a GRU-GAN generator/discriminator to augment rare market scenarios; retrain LSTM on enriched data. 2. **Metaheuristic Hyperparameter Tuning** – Apply Sine-Cosine Algorithm or Bayesian Optimization for joint tuning of LSTM, ARIMA, and LightGBM parameters. 3. **Reinforcement Learning Integration** – Prototype an RL agent that uses our forecast distribution to optimize entry/exit rules under transaction costs. 4. **Scalable Cloud Infrastructure & Monitoring** – Migrate services to auto-scaling serverless containers; deploy monitoring dashboards for drift detection and automated retraining triggers.

10. Future Scope of the Project

Building on our hybrid GAN-LSTM architecture and GAN-enhanced fusion models, we identify key limitations from prior research and outline how our approach addresses them, where it still falls short, and our plans for future improvement.

1. Generalizability Across Stocks

Limitation in Prior Works: Most GAN-based models (e.g., TSGAN on Bank of China data) focus on a single stock or sector, leading to overfitting and poor transferability to other equities. How Our Model Addresses It: We train on a diversified top-10 Indian stock universe, covering multiple sectors, and use embeddings from a TCN to capture common temporal patterns across assets.

Remaining Limitation: Our LightGBM core still tailors splits and leaf structures to the specific distribution of our 10 stocks, which may not generalize to smaller-cap or overseas equities.

Planned Improvement:

- Introduce meta-learning layers that adapt tree structures to new tickers with minimal fine- tuning.
- Build a transfer learning pipeline: pretrain the TCN on a broad index (e.g., NIFTY 50), then fine-tune on individual stocks.

2. Model Complexity & Deployment Overhead

Limitation in Prior Works: Architectures like SCA-LSTM-ARIMA-GARCH combine multiple time-series models, yielding high computational and preprocessing costs.

How Our Model Addresses It: We replace ARIMA-GARCH residual modules with a single LightGBM layer and streamline preprocessing via batched Pandas pipelines.

Remaining Limitation: Training the LSTM autoencoder and TCN still demands substantial GPU memory and offline compute time.

Planned Improvement:

- Swap in lightweight Transformer blocks with sparse attention for embeddings,

reducing parameter count.

- Explore knowledge distillation to compress the LSTM/TCN into a smaller student model for faster in-process inference.

3. Sensitivity to Market Shocks & Anomalies

Limitation in Prior Works: Fixed thresholds in LSTM or simple statistical filters miss regime- shifts or produce false positives during volatility spikes .

How Our Model Addresses It: We implement a hybrid anomaly detector combining LSTM reconstruction with rolling-window z-score filters to catch both abrupt crashes and gradual drifts. Remaining Limitation: Static window sizes and threshold formulas still lag sudden, unprecedented events (e.g., pandemic-driven crashes).

Planned Improvement:

Incorporate adaptive thresholding using Extreme Value Theory (EVT) to dynamically adjust anomaly cutoffs.

- Add a self-supervised change-point detection module to retrain thresholds when new regimes are detected.

4. Interpretability and Uncertainty Quantification

Limitation in Prior Works: GAN-based forecasts offer little insight into drivers of predictions or confidence intervals, making them hard to trust in live trading .

How Our Model Addresses It: We integrate SHAP explanations for the LightGBM stage and expose per-feature importances to users.

Remaining Limitation: We still lack probabilistic outputs or measures of prediction uncertainty.

Planned Improvement:

- Extend the regression head to quantile LightGBM models or Bayesian LightGBM variants to output prediction intervals.
- Overlay Monte Carlo dropout in the Transformer-lite encoder to approximate embedding uncertainty.

5. Real-World Trading Frictions

Limitation in Prior Works: Nearly all academic models ignore transaction costs, latency, and liquidity constraints when forecasting .

How Our Model Addresses It: Currently, we flag anomalies to avoid extreme trades but still assume frictionless execution.

Remaining Limitation: Predicted price changes don't account for slippage or order-size impact.

Planned Improvement:

- Simulate order books and market impact models in back testing to penalize large predicted moves.
- Integrate a reinforcement-learning agent that uses our forecasts but optimizes net P&L after costs.

6. Integrating Expanded Data Sources for Global Coverage

Limitation in Prior Works: Nearly all of the mentioned previous works on this topic used only focused on single market + hard coded the ticker list and sometimes even batch only pulls means no real time data.

How Our Model Addresses It: Our data-collection stage is built around reusable functions that take a list of tickers and API clients.

Remaining Limitation: We ingest OHLCV only for our top-10 Indian stocks via a fixed set of APIs, so adding new tickers requires manual code changes and separate data pulls, which doesn't scale.

Planned Improvement:

- Move ticker definitions into a YAML/JSON config so new markets (e.g., US, EU, ASEAN equities) can be enabled by editing a file, not code.
- Replace sequential pulls with a distributed ingestion layer (e.g., Apache Kafka or AWS Kinesis) that fans out requests across workers and back-pressures on rate limits, ensuring high throughput

11.Conclusion

This project presents a comprehensive approach to stock price prediction by combining multiple advanced methodologies in a unified, real-time architecture. Leveraging Temporal Convolutional Networks (TCNs) for embedding extraction, LightGBM for high-performance regression, LSTM for long-term dependency modeling, KNN for short-term similarity-based baselining, and GAN-based architectures for synthetic data generation and robustness, the system is designed to perform reliably under dynamic market conditions. The inclusion of anomaly detection, continuous learning, and real-time monitoring ensures that the model adapts to shifts in data distributions and maintains accuracy post-deployment.

We gratefully acknowledge the foundational research that guided the development of this work. The first paper, *"Stock Market Prediction with High Accuracy using Machine Learning Techniques"* by Malti Bansal et al. (2022, Procedia Computer Science), offered comparative insights into traditional ML models like KNN, LR, SVR, DTR, and LSTM, highlighting the superior accuracy of LSTM for stock forecasting. The second paper, *"Stock price prediction with SCA-LSTM network and Statistical model ARIMA-GARCH"* by Mehtarizadeh et al. (2025, Journal of Supercomputing), introduced hybrid modeling with optimization techniques, which influenced our architecture's direction toward model integration and hyperparameter tuning. The third study, *"Enhancing Stock Price Prediction Using GANs and Transformer-Based Attention Mechanisms"* by Li and Xu (2024, Empirical Economics), inspired the inclusion of sentiment-aware GAN models and attention layers for deeper feature learning. Lastly, the fourth paper, *"GAN-Enhanced Nonlinear Fusion Model for Stock Price Prediction"* by Xu et al. (2024, Int. J. of Computational Intelligence Systems), contributed to our understanding of nonlinear fusion using ARIMA, ACNN, and BiLSTM, enriching our approach to feature fusion and sequence modeling.

Together, these works provided invaluable theoretical and empirical grounding that helped shape the architecture, methodology, and evaluation strategies of our system. We sincerely thank the authors and publishers for making their work publicly available and contributing to the advancement of intelligent financial forecasting.

.

12.References:

[1] M. Bansal, A. Goyal, and A. Choudhary, "Stock Market Prediction with High Accuracy using Machine Learning Techniques," *Procedia Computer Science*, vol. 215, pp. 247–265, 2022. https://doi.org/10.1016/j.procs.2022.12.028

[2] H. Mehtarizadeh, N. Mansouri, B. M. H. Zade, and M. M. Hosseini, "Stock price prediction with SCA-LSTM network and Statistical model ARIMA-GARCH," *The Journal of Supercomputing*, vol. 81, p. 366, 2025. https://doi.org/10.1007/s11227-024-06775-6

[3] S. Li and S. Xu, "Enhancing Stock Price Prediction Using GANs and Transformer-Based Attention Mechanisms," *Empirical Economics*, vol. 68, pp. 373–403, 2024. https://doi.org/10.1007/s00181-024-02644-6

[4] Y. Xu, Y. Zhang, P. Liu, Q. Zhang, and Y. Zuck, "GAN-Enhanced Nonlinear Fusion Model for Stock Price Prediction," *International Journal of Computational Intelligence Systems*, 2024. https://doi.org/10.1007/s44196-023-00394-4