Data/Math (2026), **xx**:xx 1–23

Tutorial

Machine Learning Method in Economic Signal Prediction

FUCHIEN TSOU[1]

[1]Graduation Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan.

Keywords: Financial Time-Series Forecasting, Feature Optimization, Machine Learning

Abstract

Financial time-series forecasting remains a highly complex challenge due to inherent market volatility and low signal-to-noise ratios. This tutorial provides a comprehensive methodological framework for economic signal prediction, tracing the algorithmic evolution from foundational feature engineering to state-of-the-art generative architectures. The workflow initially establishes a robust data processing layer by transforming raw market prices into optimized multidimensional tensors utilizing classical technical indicators and correlation analysis. Subsequently, the tutorial explores the single-model paradigm, detailing the distinct mathematical advantages of deep recurrent networks, non-parametric structural signal decomposition, and tree-based tabular learning. Recognizing the constraints of isolated architectures, the paradigm shifts toward treating time-series data as language. By leveraging numerical tokenization pipelines and continuous likelihood modeling, pre-trained large language models are effectively adapted to function as powerful zero-shot forecasters. Finally, the tutorial synthesizes these standalone concepts into advanced hybrid frameworks. By integrating bidirectional sequential encoders, modified transformer attention mechanisms, and temporal convolutional networks, these unified systems capture multi-scale global dependencies while strictly preventing future data leakage. Additionally, the incorporation of domain-specific natural language processing for sentiment analysis alongside generative adversarial networks enables the extraction of unstructured market shocks. Ultimately, this tutorial systematically demonstrates how the strategic integration of optimized data preprocessing, discrete tokenization, and multimodal architectural design forms highly resilient predictive systems, offering quantitative developers a rigorous roadmap for navigating modern algorithmic trading environments.

Contents

1. Technical Indicators and Feature Engineering - Data Layer

In financial time-series forecasting, raw market data is often characterized by a low signal-to-noise ratio, non-stationarity, and extreme volatility. To address these challenges and enhance the predictive accuracy of machine learning architectures, data preprocessing and feature engineering serve as a critical foundational layer. A widely adopted framework involves calculating Technical Indicators from raw market prices to transform sequential price paths into highly informative structural signals.

Therefore, we will introduce some basic technical indicators definition before introducing machine learning architectures.

1.1. Mathematical Formulations of Basic Technical Indicators

To systematically capture different dimensions of market dynamics, such as trend changes, price momentum, and overbought/oversold conditions, several classical statistical indicators are computed using historical open, high, low, and close (OHLC) prices

1.1.1. Moving Average (MA)

Moving Average helps smooth out price data to create a trend-following indicator, making it easier to identify price trends. Which can be designed in different lengths of windows, such as MA_5, MA_{10}, MA_{20} etc.

$$\mathrm{MA}_T = \frac{1}{T}\sum_{i=0}^{T} C_i \tag{1.1}$$

Where

- T: The length of windows.
- C_i: The closing price of day i

1.1.2. Exponential Moving Average (EMA)

For an Exponential Moving Average, the calculation gives more weight to more recent prices and is defined as.

$$\mathrm{EMA}_T = \alpha C_i + (1-\alpha)\mathrm{EMA}_{T-1} \tag{1.2}$$

where

$$\alpha = \frac{2}{T+1} \tag{1.3}$$

- T: The length of windows.
- C_i: The closing price of day i

1.1.3. Moving Average Convergence Divergence (MACD)

The Moving Average Convergence Divergence is a momentum-based technical indicator that calculates the difference between a short window EMA and a long window EMA. Usually using the difference between EMA_{12} and EMA_{26}

$$\mathrm{MACD} = \mathrm{EMA}_{\mathrm{long}} - \mathrm{EMA}_{\mathrm{short}} \tag{1.4}$$

1.1.4. Relative Strength Index (RSI)

The Relative Strength Index is a momentum oscillator that measures the speed and magnitude of recent price movements to evaluate overbought or oversold conditions in a market. Usually using RSI_6, RSI_{12} and RSI_{14}.

$$\mathrm{RSI}_T = 100 - \left[\frac{100}{1+\left(\frac{\mathrm{AG}_T}{\mathrm{AL}_T}\right)}\right] \tag{1.5}$$

Where

- AG_T: The Average Gain in the over T days lookback window.
- AL_T: The Average Loss in the over T days lookback window.

1.1.5. Stochastic Oscillator (%K)

The Stochastic Oscillator is also momentum indicator comparing a specific closing price of a security to its price range over a given period T:

$$\%\mathrm{K} = \left[\frac{C_i - L_T}{H_T - L_T}\right] \times 100 \tag{1.6}$$

Where

- C_i: The current closing price.
- L_T: The Lowest prices within the last T days.
- H_T: The Highest prices within the last T days.

1.2. Feature Optimization and Tensor Transformation

Upon generating the comprehensive pool of Technical Indicators, a crucial optimization step is required before model training. Raw indicators cannot be fed blindly into deep neural networks, as highly collinear features (such as MA, EMA, and MACD) or irrelevant features can lead to overfitting and severe computational inefficiency.

To extract the most predictive inputs, Pearson correlation (IC) analysis is performed to calculate the linear relationship between each individual technical indicator and the subsequent closing price. Features demonstrating the strongest historical correlation are isolated to filter out background market noise. These highly optimized adaptive features are then structured and stacked into a final multidimensional technical tensor [1].

2. Individual Architectures

2.1. Deep Sequential Baselines: Recurrent Networks

Traditional time-series models like RNN often struggle to capture long-term temporal dependencies due to vanishing gradient problems in standard backpropagation. Long Short-Term Memory (LSTM) networks and their advanced variant, Bidirectional LSTMs (BiLSTM), resolve this by utilizing a specialized gating mechanism (comprising input, forget, and output gates) to retain information over long horizons.

While a standard LSTM processes inputs chronologically, a BiLSTM duplicates the recurrent layer to sweep data in both forward and backward time directions. This dual-directional approach allows the network to evaluate how past trajectories and future context concurrently impact current sequence steps, offering a robust baseline for dense sequential encoding.

2.1.1. Tensor Data Pipeline

Before feeding data into recurrent architectures, the optimized technical indicators and market variables processed in Chapter 1 must be strictly formatted into a three-dimensional tensor with the shape (Batch-Size, Time-Steps, Input-Features).

- Batch-Size: The number of training samples (e.g., historical trading sequences) processed simultaneously in a single iteration to stabilize gradient descent.
- Time-Steps (T): The length of the lookback historical window (e.g., using the past 30 days of market data to predict future movements).
- Input-Features (d): The total number of parallel channels, representing the raw market metrics (OHLCV) combined with the correlation-filtered Technical Indicators such as RSI and MACD.

When this 3-D tensor enters the recurrent network, the model does not look at the entire matrix at once. Instead, it sequentially unpacks the tensor along the Time-Steps axis. At each specific time step t (where $1 \leq t \leq T$), the network ingests a 1D feature vector $x_t \in \mathbb{R}^d$, representing the exact market snapshot of that specific moment. The mathematical mission of the recurrent cell is to map this continuous stream of vector inputs into an internal trajectory of memory states.

2.1.2. Long-Short Term Memory Model (LSTM)

Traditional Recurrent Neural Networks (RNNs) suffer from vanishing and exploding gradients during backpropagation through time, making it mathematically impossible to retain historical dependencies

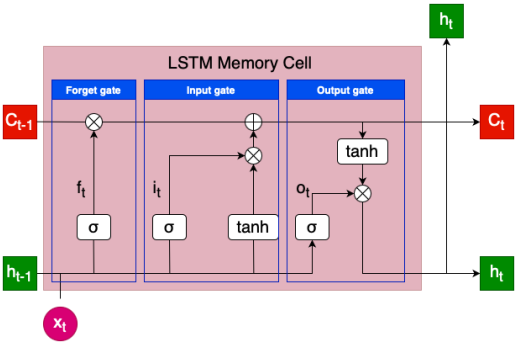

Figure 1. *LSTM Workflow Diagram.*

over long windows. To mitigate this constraint, the Long Short-Term Memory (LSTM) network introduces a specialized architectural cell governed by an explicit hidden state (h_t) and a cell state (c_t) [2], which acts as a dedicated internal conveyor belt for long-term memory. At each time step t, the LSTM cell ingests the current feature input x_t and the previous step's hidden state h_{t-1}. The architecture of the LSTM cell is illustrated in Figure 1. And the information flow is dynamically regulated by three learnable, non-linear gates:

1. Forget Gate (f_t): Determines what percentage of the long-term memory (c_{t-1}) should be discarded.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.1}$$

where

- f_t The output value of the Forget Gate, which has the range of 0 to 1.
- W_f: The weight function of Forget Gate.
- h_{t-1}: The output value of last time.
- x_t: The current input value.
- b_f: The bias of Forget Gate.
- σ: The sigmoid layer which squashes the values between 0 to 1.

2. Input Gate (i_t & $\tilde{c}_t$): Decides which new market insights from x_t should be written into the long-term cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.2}$$

Where

- i_t The output value of the Input Gate, which has the range of 0 to 1.
- W_i: The weight function of Input Gate.
- h_{t-1}: The output value of last time.
- x_t: The current input value.
- b_i: The bias of Input Gate.
- σ: The sigmoid layer which squashes the values between 0 to 1.

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{2.3}$$

Where

- $\tilde{c}_t$ The candidate cell state value.

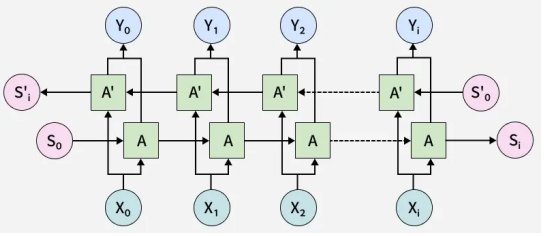

Figure 2. *BiLSTM Workflow Diagram where A and A' represent LSTM cell - GeeksforGeeks.*

- W_c: The weight function of candidate cell state.
- h_{t-1}: The output value of last time.
- x_t: The current input value.
- b_c: The bias of candidate cell state.
- tanh: The hyperbolic tangent function squashes the values between -1 to 1.

Then, Cell State Update (c_t) Combines the filtered past memory and the validated new insights to form the updated long-term context. Where $\odot$ represents Hadamard Product.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.4}$$

3. Output Gate (o_t & h_t): Generates the current hidden state h_t, which serves as both the explicit model output for step t and the short-term contextual memory passed to step $t + 1$.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.5}$$

Where

- o_t The output value of the Output Gate, which has the range of 0 to 1.
- W_o: The weight function of Output Gate.
- h_{t-1}: The output value of last time.
- x_t: The current input value.
- b_o: The bias of Output Gate.
- σ: The sigmoid layer which squashes the values between 0 to 1.

Then, update the current hidden state h_t

$$h_t = o_t \odot \tanh(c_t) \tag{2.6}$$

2.1.3. Bi-directional Long Short-Term Memory Model(BiLSTM)

In financial markets, price movements at time t are heavily influenced by local micro-trends and structural context that span across the entire temporal window. While a standard LSTM can only model information chronologically from left to right, the Bidirectional LSTM (BiLSTM) framework duplicates the internal hidden processing to scan the time series from both ends simultaneously [3].

As demonstrated in figure 2, for any given input x_t, the BiLSTM activates two independent LSTM processing channels directly from start point and reversely from end point.

- The Forward Layer ($\vec{h}_t$): Recurrently processes the sequence chronologically from $t = 1$ to $t = T$, capturing historical dependencies up to the current moment.
- The Backward Layer ($\overleftarrow{h}_t$): Recurrently processes the sequence in reverse chronological order from $t = T$ down to $t = 1$, capturing the latent future context surrounding that specific step.

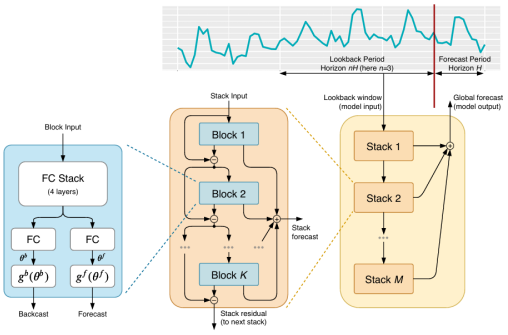

Figure 3. *N-BEATS Workflow Diagram.*

The final aggregated hidden state representation h_t^{bi} is constructed by concatenating the hidden state vectors from both networks

$$h^{bi} = [\vec{h}_t \parallel \overleftarrow{h}_t] \tag{2.7}$$

2.2. Structural Signal Decomposition: N-BEATS

Departing from recurrent operations entirely, Neural Basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS) relies purely on a deeply stacked [4], fully connected Multi-Layer Perceptron architecture. The core innovation lies in its doubly residual connection architecture. Each basic building block outputs a backcast (to reconstruct and remove the historical input signal) and a forecast (to predict future horizons). By subtracting the backcast from the input, the residual signal is isolated and passed to the next block, forcing downstream layers to focus exclusively on unexplained noise.

Furthermore, by enforcing specific polynomial and Fourier basis functions within its projection layers, N-BEATS elegantly decomposes a raw price line into highly interpretable, mathematically distinct Trend and Seasonality components without requiring any manual feature engineering.

2.2.1. Architecture

As illustrate in figure 3, the most basic unit of N-BEATS is a "Block" (blue ones) and multiple base "Blocks" will be connected in series into a "Stack" (Orange one). The entire architecture is assembled by chaining together multiple "Blocks" into a series of "Stacks".

2.2.2. Basic Blocks

First, the input signal x_ℓ will pass through the standard full connection layer (FC Stack) with 4 layers, using RELU nonlinear activation functions.

$$h_{\ell,1} = \text{ReLU}(W_{\ell,1}x_\ell + b_{\ell,1}) \tag{2.8}$$

$$h_{\ell,2} = \text{ReLU}(W_{\ell,2}h_{\ell,2} + b_{\ell,2}) \tag{2.9}$$

$$h_{\ell,3} = \text{ReLU}(W_{\ell,3}h_{\ell,2} + b_{\ell,3}) \tag{2.10}$$

$$h_{\ell,4} = \text{ReLU}(W_{\ell,4}h_{\ell,3} + b_{\ell,4}) \tag{2.11}$$

Then, the network will calculate two sets of expansion coefficients through the LINEAR layer:

- Forward expansion coefficient θ_ℓ^f: Represents the reconstruction of historical input data by the model.
- Backward expansion coefficient θ_ℓ^b: Represents the model's reconstruction of past time points.

$$\theta_\ell^b = \text{LINEAR}_\ell^b(h_{\ell,4}) \tag{2.12}$$

$$\theta_\ell^f = \text{LINEAR}_\ell^f(h_{\ell,4}) \tag{2.13}$$

These expansion coefficients are mapped to actual temporal waveforms through dedicated basis functions:

$$\hat{y}_\ell = g_\ell^f(\theta_\ell^f) \quad \text{(Forecast)} \tag{2.14}$$

$$\hat{x}_\ell = g_\ell^b(\theta_\ell^b) \quad \text{(Backcast)} \tag{2.15}$$

Where $\hat{y}_l = \sum_{i=1}^{\dim(\theta_\ell^f)} \theta_{\ell,i}^f \mathbf{v}_i^f$ and $\hat{x}_\ell = \sum_{i=1}^{\dim(\theta_\ell^b)} \theta_{\ell,i}^b \mathbf{v}_i^b$. Here $\mathbf{v}_i^f$ and $\mathbf{v}_i^b$ are forecast and backcast basis vectors.

Basically, we will using neural network to learn basis vectors but it's not unexplainable. To address the problem of neural networks learning bases on their own leading to unexplainable results, researchers can directly set the functions of the base layer to specific mathematical forms.

- Trend model: In order to mimic the typically slowly changing monotonic function characteristics of a trend, the basis matrix is set as a low-degree polynomial matrix.
- Seasonality model: To simulate the cyclical fluctuation characteristics of seasonal patterns, the base matrix is set as a periodic Fourier series matrix S.

2.2.3. Backcast Residual Branch

Multiple blocks combined as stacks and it demonstrates the unique way its residual branches operate. The Backcast ($\hat{x}_l$) acts as the block's best estimation of the historical sequence it just analyzed.

To enforce the residual mechanism, the block subtracts this backcast from its original input to generate a cleaned residual signal.

$$x_{\ell+1} = x_\ell - \hat{x}_\ell \tag{2.16}$$

- The residual calculated $x_{\ell+1}$ here will be directly used as the input for the next block.
- It means that the backward residual branch is performing a 'sequential analysis' of the input signal, with each block trying to explain the left historical data that the previous block could not realize for.

2.2.4. Global Forecast Branch

When the input signal sequentially passes through all the blocks in the Stack, each block will generate its own partial forecast $\hat{y}_\ell$.

Concurrently, the final Global Forecast ($\hat{Y}$) of the system is obtained via a global additive aggregation of all partial forecasts across all layers.

$$\hat{Y} = \sum_{\ell} \hat{y}_{\ell} \tag{2.17}$$

2.3. Tree-Based Apex: LightGBM

In the contrast to neural network approaches, Light Gradient Boosting Machine (LightGBM) represents the apex of the tree-based machine learning [5]. Instead of map-reading raw sequences, While traditional GBDT implementations (such as XGBoost and pGBRT) have achieved state-of-the-art performance in many machine learning tasks, their training efficiency and scalability remain unsatisfactory when dealing with large-scale datasets and high-dimensional features.

The main computational bottleneck of traditional methods lies in the fact that, in order to evaluate every possible split point, the algorithm must scan all data samples to calculate the information gain, which results in the overall computational complexity being proportional to the number of features and samples.

To address this pain point, LightGBM innovatively proposes two core technologies: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). These two technologies address the issue by reducing the number of samples (#data) and the number of features (#features), respectively, successfully accelerating the training process of traditional GBDT by more than 20 times while maintaining near-lossless accuracy.

2.3.1. Overview: LightGBM vs Traditional GBDT

⋆ Gradient Boosting Decision Tree (GBDT)

- Core Idea: A sequential ensemble learning algorithm where decision trees are trained one after another.
- How it Works: Each new tree fits the negative gradients (residual errors) of the previous trees to continuously minimize the overall prediction error.
- The Problem: Traditional GBDT must scan every single data instance and feature to find the best split points, making it highly time-consuming with large-scale tabular data.

⋆ LightGBM

1. GOSS (Gradient-based One-Side Sampling) — Reduces Rows (#data)
   - Concept: Data instances with larger gradients (higher errors) play a more important role in computing information gain.
   - Method: It keeps all instances with large gradients and randomly samples a smaller percentage of instances with small gradients.
   - Result: It drastically decreases the data size while maintaining a highly accurate gain estimation.
2. EFB (Exclusive Feature Bundling) — Reduces Columns (#features)
   - Concept: High-dimensional tabular data is usually very sparse, meaning many features are mutually exclusive.
   - Method: It bundles these exclusive sparse features into a single dense feature bundle by adding offsets to their values.
   - Result: It avoids unnecessary computations for zero values, reducing the histogram building complexity from $O(\#data \times \#feature)$ to $O(\#data \times \#bundle)$.

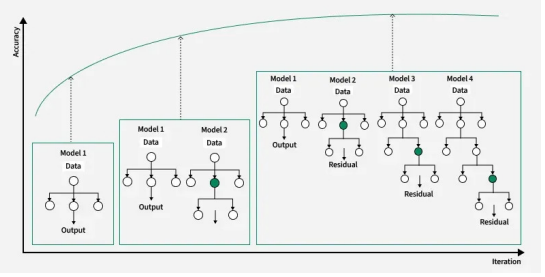

Figure 4. *LightGBM Decision Making Process and Accurcy - GeeksforGeeks.*

$$\text{LightGBM} = \text{GBDT} + \text{GOSS} + \text{EFB}$$

2.3.2. Histogram-based Algorithm

In terms of optimizing the underlying data structure, LightGBM abandons the traditional, time-consuming, and memory-intensive pre-sorted algorithm and instead develops based on the **Histogram-based Algorithm (Algorithm 1)**.

- Data Bucketing: This algorithm discretizes continuous feature values and puts them into a fixed number of bins. During training, these discrete bins are used directly to construct feature histograms.
- Complexity Reduction: This histogram structure reduces the complexity of histogram construction from pre-sorted to O(#data × #feature), while the complexity of finding the optimal split point is only $O(\#\text{bin} \times \#\text{feature})$. Since bin is much smaller than data, histogram building dominates the overall runtime.

2.3.3. Gradient-based One-Side Sampling (GOSS)

The core idea of **Gradient-based One-Side Sampling) (Algorithm 2)** (GOSS) is to use the magnitude of the gradient of a sample as a metric for the importance of the data. In GBDT, although there are no native sample weights like in AdaBoost, the absolute value of the gradient directly reflects the residual error: samples with larger gradients represent under-trained instances and contribute more to the information gain; while samples with smaller gradients represent those that are perfectly fitted.

In a standard GBDT, if we let O be the training dataset at a fixed node in the decision tree, then the variance gain of feature j at the split point d is defined as

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{\left(\sum_{\{x_i \in O : x_{ij} \le d\}} g_i\right)^2}{n^j_{l|O}(d)} + \frac{\left(\sum_{\{x_i \in O : x_{ij} > d\}} g_i\right)^2}{n^j_{r|O}(d)}\right) \quad \textbf{(GBDT)} \tag{2.18}$$

Where

- $n_O = \sum I[x_i \in O]$: # of samples.
- $n^j_{l|O}(d)$: # of sample after splitting on the left side.
- $n^j_{r|O}(d)$: # of sample after splitting on the right side.

To reduce the sample size, GOSS first sorts the data in descending order of absolute gradient value, retaining the first $a \times 100\%$ of large gradient samples (set A). Then, from the remaining $(1-a) \times 100\%$ of small gradient samples (set A^c), $b \times |A^c|$ samples (set B) are randomly selected. To avoid disrupting

Algorithm 1 Histogram-based Algorithm

```
Input : I: training data,d: max depth
Input : m : feature dimension
nodeSet ← {0} ▷ tree nodes in current level
rowSet ← {{0, 1, 2, ...}} ▷ data indices in tree nodes
for i = 1 to d do
    for node in nodeSet do
        usedRows ← rowSet[node]
        for k = 1 to m do
            H ← new Histogram()
            ▷ Build histogram
            for j in usedRows do
                bin ← I.f[k][j].bin
                H[bin].y ← H[bin].y + I.y[j]
                H[bin].n ← H[bin].n + 1
            end for
            Find the best split on histogram H.
            ...
        end for
    end for
    Update rowSet and nodeSet according to the best split points.
end for
```

the original data distribution when calculating information gain, GOSS introduces a constant multiplier $\frac{1-a}{b}$ for the small gradient samples to compensate for the weights. The corrected estimated variance gain $\tilde{V}_j(d)$ is given by the following formula:

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n_r^j(d)}\right) \quad \textbf{(GOSS)} \qquad (2.19)$$

2.3.4. Exclusive Feature Bundling (EFB)

In large-scale practical applications, high-dimensional tabular data is often accompanied by extremely high sparsity. Many features are mutually exclusive, meaning they rarely take non-zero values simultaneously (e.g., features after one-hot encoding). The purpose of Exclusive Feature Bundling (EFB) is to securely bind these mutually exclusive features into a single exclusive feature bundle, thereby effectively reducing the feature dimensionality.

The paper proves that dividing features into the minimum number of mutually exclusive bundles is an NP-hard problem. Therefore, EFB transforms it into a graph coloring problem and uses the **Greedy Bundling (Algorithm 3)** to find an approximate solution, while allowing a small maximal conflict rate (gamma) to achieve more extreme feature compression.

After determining the feature groups, to allow the merged feature bundle to re-identify the original feature values, EFB adds offsets to isolate the histogram intervals (discrete bins) of each feature. For example, if the original range of feature A is $[0, 10)$ and the original range of feature B is $[0, 20)$, the algorithm will add an offset of 10 to feature B, moving its range to $[10, 30)$, and finally merge them into

Algorithm 2 Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models ← { }, fact ← $\frac{1-a}{b}$
topN ← a×len(I) , randN ← b × len(I)
for i = 1 to d **do**
    preds ← models.predict(I)
    g ← $loss$(I, preds), w ← 1,1,...
    sorted ← GetSortedIndices(abs(g))
    topSet ← sorted[1:topN]
    randSet ← RandomPick(sorted[topN:len(I)],randN)
    usedSet ← topSet + randSet
    w[randSet] × = fact ▷ Assign weight $fact$ to the
    small gradient data.
    newModel ← L(I[usedSet], - g[usedSet], w[usedSet])
    models.append(newModel)
end for

a comprehensive feature package with a range of $[0, 30]$. Last, merge by **Merge Exclusive Features (Algorithm 4)**

Algorithm 3 Greedy Bundling Algorithm

Input: F: features, K: max conflict count
Construct graph G
searchOrder ← G.sortByDegree()
bundles ← { }, bundlesConflict ← { }
for for i in $searchOrder$ **do**
    needNew ← True
    for for j = 1 to $len(bundles)$ **do**
        cnt ← ConflictCnt(bundles[j],F[i])
        if $cnt + bundlesConflict[i] \leq K$ **then**
            bundles[j].add(F[i]), needNew ← False
            break
        end if
    end for
    if $needNew$ **then**
        Add $F[i]$ as a new bundle to $bundles$
    end if
end for
Output: $bundles$

Algorithm 4 Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges $\leftarrow \{0\}$, totalBin $\leftarrow 0$
for f in F **do**
    totalBin += f.numBin
    binRanges.append(totalBin)
end for
newBin $\leftarrow$ new Bin(numData)
for $i = 1$ to $numData$ **do**
    newBin[i] $\leftarrow 0$
    for $j = 1$ to $len(F)$ **do**
        if $F[j].bin[i] \neq 0$ **then** then
            newBin[i] $\leftarrow$ F[j].bin[i] + binRanges[j]
        end if
    end for
end for
Output: $newBin$, $binRanges$

3. The Large Language Model Shift

Traditional quantitative finance interprets economic signals strictly as continuous, numeric vectors. However, the modern state-of-the-art frontier introduces a radical conceptual shift: framing financial time-series data as a language modeling task. Because Large Language Models (LLMs) are built to represent highly complex probability distributions over sequences, they are theoretically well-suited to discover underlying temporal patterns without requiring task-specific architecture changes. By treating sequential numbers identically to text characters, pre-trained generative architectures can act as powerful zero-shot time-series forecasters, outperforming many popular deep learning baselines on target datasets without any additional historical training.

3.1. Tokenization Process

If we want to framing financial time-series data as a language, Tokenization as a critical preprocessing layer to transform continuous price trajectories into discrete, structured symbolic inputs. This discretization process provides two major advantages in quantitative workflows:

1. Noise Reduction: By grouping data points or mapping continuous values into distinct semantic bins, tokenization filters out microscopic market noise and random fluctuations, allowing downstream models to focus on structural trends.
2. Architecture Synergy: Converting time-series arrays into standardized tokens allows developers to bypass restrictive task-specific networks and directly leverage powerful, pre-trained attention backbones or ready Large Language Models (LLMs) for multi-step forecasting.

In practice, modern time-series tokenization diverges into two distinct methods: neural-embedded patching (PCIE), which maps continuous sub-series blocks into dense latent vectors within trained neural networks, and string-based text encoding (LLMTIME), which formats raw digits directly into character strings to manipulate existing natural language models such as GPT, Claude, Gemini, etc.

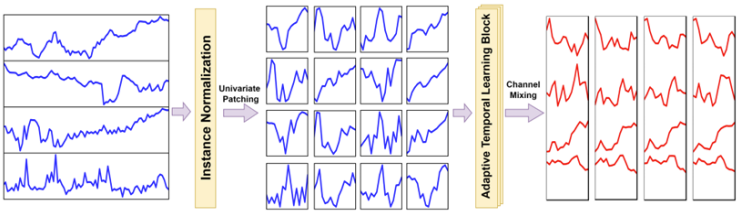

Figure 5. *PCIE Tokenization Process.*

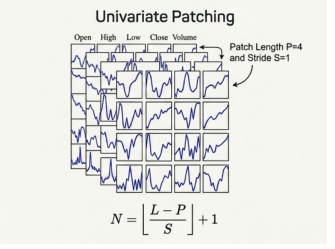

Figure 6. *PCIE - Univariate Patching Visualize.*

3.1.1. Patched Channel Integration Encoder (PCIE)

The Patched Channel Integration Encoder (PCIE) model introduces a highly effective, architecture-driven tokenization framework specifically tailored to handle multi-step stock price direct estimation and stock percentage return estimation simultaneously [6].

Unlike traditional methods that flatten the signal relationships or treat the financial tensor as completely isolated variables, the PCIE tokenization process models financial time-series by mapping local temporal patterns and complex cross-channel correlations directly into a high-dimensional latent embedding space.

1. **Univariate Patching**: Instead of processing the entire time series point-by-point, the model takes an input data set of length L containing M different financial channels (series) and splits each individual series into smaller segments called "patches", illustrated in **Figure 6**.

$$N = \left\lfloor \frac{L-P}{S} \right\rfloor + 1 \tag{3.1}$$

   Where

   - N: Total number of patches in each series.
   - L: Total length of the original input time series.
   - P: Fixed length of each patch.
   - S: Stride during segmentation.
   - If a series is not perfectly divisible, padding is added to the ends of the series.

2. **Adaptive Temporal Learning (ATL)**: After cutting the inputs into localized patches, each individual channel i patch vector of size $1 \times P$ ($X_P^{(i)}$) must be mapped to a latent token feature vector vector of size $1 \times d$ ($X_{d\_patch}^{(i)}$). Because distinct financial assets and technical indicators feature highly volatile distribution properties, the ATL block dynamically evaluates three parallel structural mappings to generate optimal learned representations and generate the optimal results, illustrated in **Figure 7**.

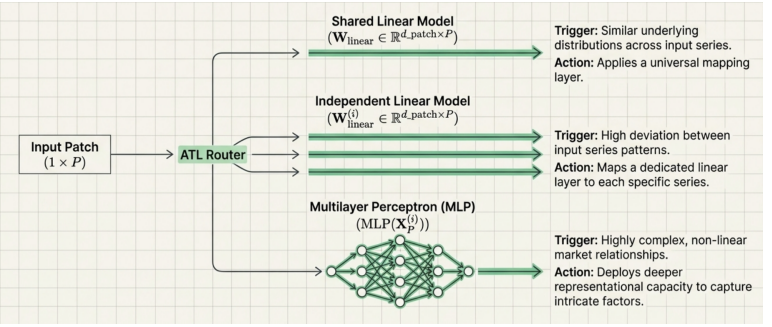

Figure 7. *PCIE - Feature Adaptive Temporal Learning Process.*

- **Shared Linear Layer**: Employs a single shared linear weight layer across all channels, highly effective when asset classes follow symmetric underlying tracking distributions.

$$X^{(i)}_{d\_patch} = W_{linear} \cdot X^{(i)}_P + b_{linear} \quad \text{where } W_{linear} \in \mathbb{R}^{d\_patch \times P} \tag{3.2}$$

- **Series-Independent Layer**: Allocates a dedicated, completely separate linear projection matrix $W^{(i)}_{linear}$ for each unique input channel i. This layer avoids interference errors when data fields exhibit massive deviations in statistical traits.

$$X^{(i)}_{d\_patch} = W^{(i)}_{linear} \cdot X^{(i)}_P + b^{(i)}_{linear} \tag{3.3}$$

- **Multilayer Perceptron**: Leverages a deeper non-linear neural network map to encode highly complex, non-linear dependencies across market intervals.

$$X^{(i)}_{d\_patch} = \text{MLP}(X^{(i)}_P) \tag{3.4}$$

3. **Channel Mixing**: Last, Aggregates independent univariate components to construct the multi-dimensional token array. Since financial indicators cannot be viewed as isolated variables. To solve the structural timeline bias, a learnable additive position encoding matrix ($W_{pos} \in \mathbb{R}^{d\_model \times N}$) is blended into the tensor. The resulting token feature size scales up to $d\_model = d\_patch \times M$, mapping the multi-variable integration into a single Financial Token.

$$X_{d\_model} = \text{Flatten}\left(X^{(1)}_{d\_patch}, \ldots, X^{(M)}_{d\_patch}\right) + W_{pos} \tag{3.5}$$

3.1.2. LLMTIME

While architecture-driven frameworks like PCIE restructure time-series data using neural networks, LLMTIME introduces a purely data-driven approach. The core philosophy is elegantly simple: by encoding continuous numerical time-series values into standardized strings of text, time-series forecasting can be framed directly as a next-token prediction task. This unlocks the ability to use pre-trained, off-the-shelf Large Language Models (such as GPT-3 or LLaMA-2) as zero-shot forecasters [7], illustrated at a high level in **Figure 8**.

However, standard Large Language Models (LLMs) struggle with numbers because of Byte-Pair Encoding (BPE). Instead of seeing a number as a coherent mathematical value, BPE chops it into arbitrary chunks based on text frequency (e.g., 42235630 becomes [422], [35], [630]). Because a tiny numerical change can completely rewrite how the number is split, the model cannot recognize numerical patterns, learn arithmetic, or track time-series steps reliably. We solve this problem by following procedure.

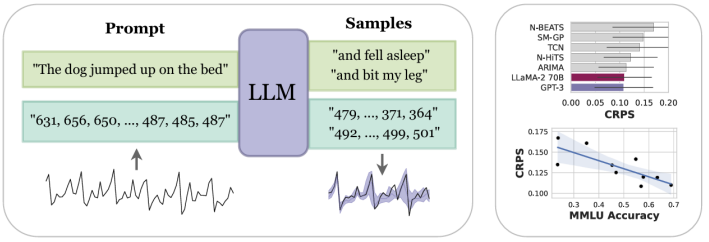

Figure 8. *LLMTIME, a method for time series forecasting with large language models (LLMs) by encoding numbers as text and sampling possible extrapolations as text completions. LLMTIME can outperform many popular time series methods without any training on the target dataset (i.e. zero-shot). The performance of LLMTIME also scales with the power of the underlying base model.*

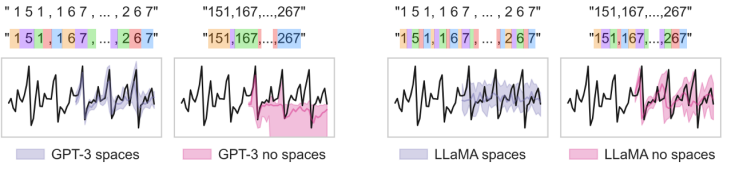

Figure 9. *LLMTIME - GPT vs LLaMa.*

- **Fixed Precision Truncation**: Raw continuous floats are truncated to a fixed decimal precision (typically 2 or 3 digits), and redundant decimal points are completely dropped to optimize the model's context window efficiency (e.g., 12.30 becomes the integer string "1230").
- **Multi-Ordered Percentile Rescaling**: Inputs are rescaled via an affine transformation ($x_t \mapsto \frac{x_t - b}{a}$) using a validation-tuned offset b and a strict α-percentile a (instead of the absolute maximum value). This exposes the LLM to shifts in digit lengths, enabling it to extrapolate future values far beyond historical boundaries.
- **Digit Separation Mechanics (GPT vs. LLaMA)**:
  1. For GPT models: Spaces are manually injected between every single digit, and commas separate time steps (e.g., [151, 167] becomes "1 5 1, 1 6 7,"). This forces Byte-Pair Encoding (BPE) tokenizers to assign exactly one separate token per digit, neutralizing chunking errors during sampling.
  2. For LLaMA models: Because newer open-source architectures natively tokenize numbers into individual digits by default, values are encoded compactly without added spaces ("151,167,") to prevent nuisance inputs and preserve sequence distribution consistency.
  3. Results shown in **Figure 9**.
- **Autoregressive Token Generation**: Predictions are drawn as standardized next-token completions conditioned exclusively on historical prompts ($p_\theta(u_j \mid u_{0:j-1})$). Generating multiple sample paths (e.g., 20 samples) via temperature scaling and nucleus sampling yields stable point-wise medians and probabilistic quantiles.
- **Continuous Likelihood Mapping**: Modeling individual digits sets up an implicit hierarchical softmax distribution over a base-B vocabulary, mapping values into B^n disjoint numeric bins of width B^{-n}. By placing a uniform distribution across each discrete bin, LLMTIME transforms discrete token probabilities into continuous probability densities ($\log p(x) = \log p_k + n \log B$). This provides high-resolution uncertainty quantification across asymmetric or multimodal market environments.

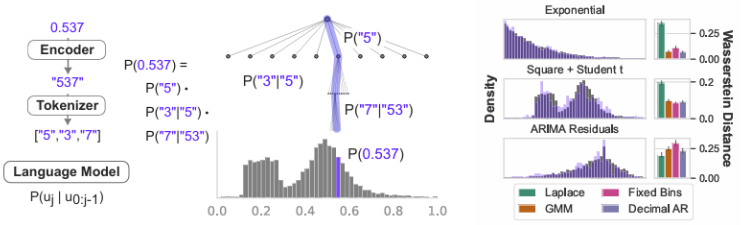

***Figure 10. Left**: Autoregressive models treating digit sequences as hierarchical softmax distributions—combined with uniform bucket distributions—create highly expressive continuous domain models. **Right**: Simple RNNs trained on string numbers can fit complex, multimodal, or heavy-tailed distributions, outperforming traditional methods like GMMs or fixed binning in Wasserstein distance.*

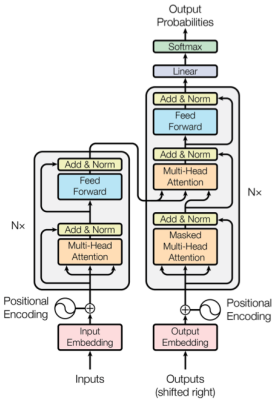

***Figure 11.** The Transformer - model architecture.*

3.2. Transformer

To process the discrete financial tokens generated in the previous section, modern quantitative pipelines rely heavily on the Transformer framework, illustrate in **Figure 11**. Originally introduced by Google in 2017 for machine translation [8], the Transformer completely discards traditional recurrent or convolutional structures in favor of a pure Multi-Head Self-Attention mechanism.

By dynamically mapping geometric relationships between all time steps simultaneously, the model bypasses the sequential processing bottlenecks of traditional LSTMs, achieving superior parallelization, robust gradient stability, and a global receptive field that can access historical macro-trends regardless of temporal distance.

Note: Given that the Transformer has become the cornerstone of modern artificial intelligence and generative large language models, its generic inner workings, such as the exact mathematical scaling of Query, Key, and Value (Q,K,V) matrices or structural layer normalization, have been documented across thousands of mainstream AI platforms. Therefore, this tutorial will intentionally bypass an over-detailed deep dive into standard Transformer theory. Instead, our technical focus will remain strictly scoped to how to strategically adapt and modify its encoder-decoder topology to optimize zero-shot multi-step economic signal predictions.

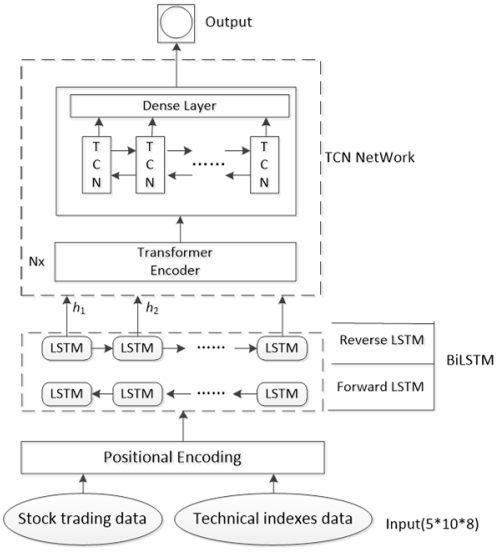

Figure 12. *BiLSTM-MTRAN-TCN model structure diagram.*

4. Hybrid Architectures - The Integration Layer

While standalone architectures offer powerful mathematical baselines, financial time-series data contains highly fragmented signals that no single algorithmic model can perfectly encapsulate. Standalone models are naturally constrained by their unique mathematical assumptions: LSTMs/BiLSTMs excel at retaining local chronological sequential dependencies but scale poorly over long horizons by the forget gate; TCNs leverage parallel dilated convolutions to efficiently capture wide receptive fields without future information leakage; Transformers exploit self-attention to dynamically prioritize multi-dimensional feature alignments; and deep generative frameworks like VAEs and GANs excel at high-dimensional denoising and probabilistic scenario synthesis.

To achieve maximum predictive accuracy, modern SOTA paradigms deploy Hybrid Frameworks that strategically combine these models, allowing each component to handle the specific signal dimension it is mathematically optimized for.

4.1. Spatio-Temporal Hybrid: BILSTM-MTRAN-TCN

While classical architectures offer strong baselines, they individually suffer from structural bottlenecks when applied to highly volatile stock market data. Transformers are exceptional at capturing global distance information but exhibit inherent weaknesses in extracting local sequence dependencies. Conversely, while BiLSTMs are highly effective at capturing bidirectional sequential data, they are prone to rapid gradient decay when processing extremely long financial time series, leading to the loss of important feature information. Furthermore, many traditional models suffer from timeliness issues, meaning their predictive stability degrades across different historical market periods.

To resolve these bottlenecks, the BILSTM-MTRAN-TCN architecture proposes a novel spatio-temporal hybrid framework [9]. The overall architecture sequentially processes 3D input tensors through a Positional Encoding layer, a BiLSTM feature extractor, an improved Transformer Encoder (MTRAN), and finally a Temporal Convolutional Network (TCN) coupled with a Dense layer to output the final stock price, demonstrated in **Figure 12**.

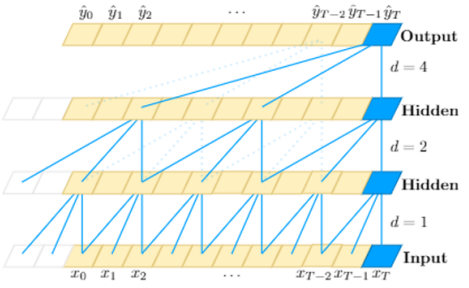

Figure 13. *Time Convolutional Neural Network (TCN).*

4.1.1. The BiLSTM Base

Instead of feeding raw market tensors directly into the attention mechanism, this framework utilizes a BiLSTM layer as the foundational sequence extractor. The BiLSTM effectively sweeps the input data to capture both forward and backward chronological information. By preprocessing the sequence data through the BiLSTM, the model stabilizes the learning process and prevents the rapid gradient decay and context explosion that typically plague ultra-long time-series analysis. This ensures that critical local sequence contexts are preserved and structurally reinforced before reaching the downstream attention modules.

4.1.2. The MTRAN Core

The Modified Transformer (MTRAN) core inherits the powerful Multi-Head Self-Attention mechanism from the standard Transformer Encoder. Once the BiLSTM extracts the local sequence features, the MTRAN calculates the mathematical importance of each element within the multi-source feature fusion. This mechanism prioritizes crucial feature information related to stock price fluctuations while ignoring irrelevant background noise. By dynamically assigning attention weights across different aspects of the time pattern, the MTRAN core successfully captures both short-term and long-term global dependencies that the BiLSTM alone cannot fully encapsulate.

4.1.3. The TCN Decoder

To replace the heavy and complex standard Transformer decoder, the architecture introduces a Temporal Convolutional Network (TCN). The TCN layer is fundamentally built upon three mechanisms [10], illustrated in **Figure 13**:

1. **Causal Convolutions**: Ensuring that the output at time T is convolved exclusively with elements that occurred prior to time T, strictly preventing future information leakage (a fatal flaw in many quant models, look-forward bias).

$$F(s) = \sum_{i=0}^{k-1} f(i) x_{s-di} \tag{4.1}$$

   Where

   - x: Time Series Data input.
   - s: The time point currently calculated.
   - d: Dilation factor.

2. **Dilated Convolutions**: Allowing the network to exponentially expand its receptive field to capture longer dependency information without having to stack an excessive number of hidden layers.

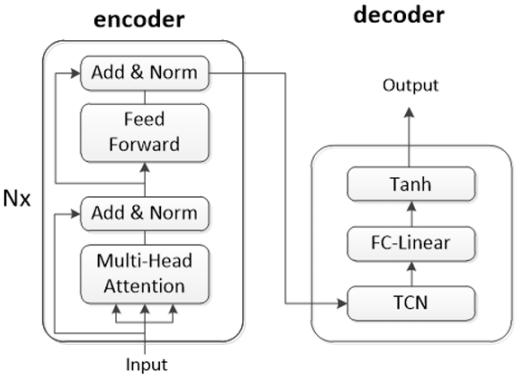

Figure 14. *Modified overall structure of transformer (MTRAN-TCN).*

3. **Residual Connections**: Reducing the network depth and parameter count, which effectively prevents network degradation, stabilizes gradient updates, and significantly improves the model's generalization ability and training efficiency.

4.1.4. MTRAN-TCN vs Traditional Transformer

The MTRAN-TCN is not merely a combination of models; it requires a surgical modification of the original Transformer (which was originally designed for Natural Language Processing tasks like machine translation). The structural differences are stark, as illustrated in **Figure 14**:

- **Removal of Input Embedding**: Standard Transformers use an Input Embedding module to vectorize text words into continuous vectors. Since stock data is already numerical, this text-vectorization step is completely removed in MTRAN-TCN to optimize efficiency.
- **Relocation of Positional Encoding**: Instead of feeding directly into the Transformer, the Positional Encoding module is moved to the very front of the pipeline, preceding the BiLSTM layer, to provide explicit temporal coordinates to the recurrent network.
- **Complete Decoder Replacement**: A traditional Transformer decoder utilizes complex masked multi-head attention and encoder-decoder attention to generate text sequentially. MTRAN-TCN strips away this entire original decoder, replacing it with the highly efficient TCN layer followed by a Fully Connected (Dense) layer and a Tanh activation function. This transforms the model from a sequential text generator into a direct, multi-step numerical forecaster.

4.2. Generative & Denoising Hybrid: FinBERT-VAE-GAN

Traditional quantitative forecasting paradigms are often highly constrained because they rely entirely on structured numeric market data, completely ignoring the profound, non-linear shocks induced by unstructured text anomalies like financial news and social media investor sentiment. Furthermore, standalone neural networks that model financial asset prices directly are prone to extreme overfitting and mode collapse due to the low signal-to-noise ratio, intense non-stationarity, and high-frequency background volatility inherent in capital markets.

To overcome these systemic vulnerabilities, this framework introduces a Multimodal Generative Denoising Hybrid architecture [11], as illustrated in **Figure 15**. The system resolves these bottlenecks by stacking a domain-specific financial natural language processing backbone (FinBERT), a Variational Autoencoder (VAE) for latent feature compressed representation and noise filtering, and a Generative

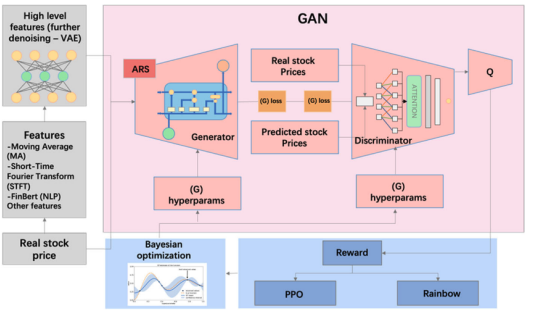

Figure 15. *FinBERT-VAE-GAN model structure diagram.*

Adversarial Network (GAN) optimized via transformer-based attention modules to generate robust, high-fidelity probabilistic scenario forecasts.

4.2.1. Sentiment Integration via FinBERT

Unstructured textual alternative data from reputable platforms such as Seeking Alpha is scraped and passed through a rigorous cleaning and tokenization pipeline. Because daily news occurrences are highly irregular, where certain periods experience multiple articles while other trading days lack any coverage, a weighted average approach is executed to normalize the impact of daily news density and ensure equitable feature distribution.

FinBERT, a bidirectionally trained transformer backbone optimized specifically for financial lexicons, processes the cleaned text through named entity recognition to filter content based on target corporations [12]. It maps the unstructured data into continuous probability distributions across negative, positive, and neutral sentiments. These generated sentiment scores serve as robust quantitative anchors that directly reflect market mood and investor expectations, acting as critical multimodal feature inputs for downstream sequential modeling. The workflow of FinBERT demonstrated in **Figure 16**.

4.2.2. Unsupervised Denoising via VAE

The multi-modal feature pool—comprising raw technical indicators, Short-Time Fourier Transform (STFT) periodic frequencies, and the FinBERT sentiment vectors is concatenated into a comprehensive feature matrix. This dense matrix is ingested by the VAE's Probabilistic Encoder, which is initialized with a self-attention module to capture global dependencies before projecting the features through dense linear layers with ReLU activations.

The network parameterizes a multi-variate Gaussian distribution within a compressed, lower-dimensional latent space using explicit mean (μ) and variance (σ^2) vectors via the reparameterization trick, ensuring differentiability during backpropagation. By minimizing the Evidence Lower Bound (ELBO), which contains the reconstruction error term and a latent space distribution regularization term via Kullback-Leibler (KL) divergence, the VAE effectively strips away random high-frequency market noise and isolates the true underlying structural representations of the economic signals [13].

4.2.3. Attention-Driven Learning via GANs

The compressed, denoised latent representation generated by the VAE is concatenated with original indicators to serve as the input for the Generative Adversarial Network (GAN) pipeline [14], as illustrated in **Figure 18**. The Generator (G) employs multiple Gated Recurrent Unit (GRU) layers, as illustrated in **Figure 17**, to model sequential dependencies and synthesize plausible multi-day future stock price trajectories, aims to deceive the opposing network. Concurrently, the Discriminator (D) utilizes a

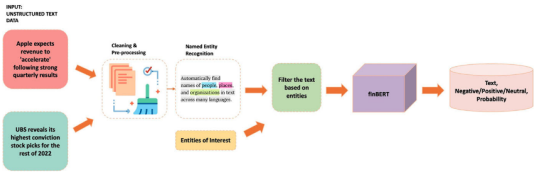

Figure 16. *Workflow of FinBERT.*

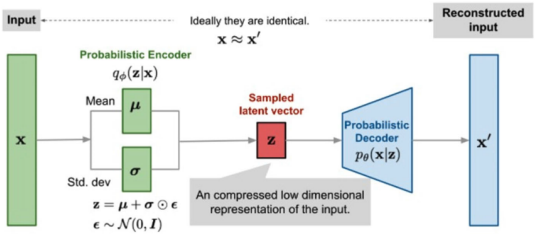

Figure 17. *Workflow of GRU.*

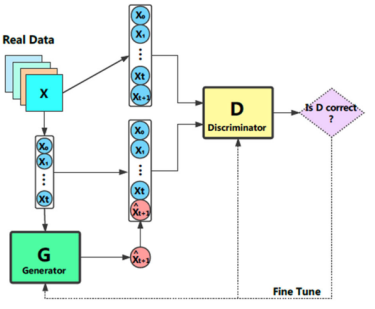

Figure 18. *Workflow of GANs.*

sequence of multi-layered convolutional structures to extract intricate patterns within the time series to differentiate genuine samples from synthetic ones.

Critically, a Transformer-based multi-head self-attention mechanism is embedded into the Discriminator. This attention mechanism allows the network to focus on the most vital feature representations and dynamically weigh the mathematical importance of different time steps in the input sequence. The entire adversarial minimax optimization is fine-tuned using reward-based reinforcement learning structures (such as PPO or Rainbow) and Bayesian optimization minimizing Root Mean Squared Error (RMSE) to achieve highly accurate, risk-adjusted forecasting profiles.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x_{\text{real}} \sim P_{\text{real}}}[\log D(x_{\text{real}})] + \mathbb{E}_{z \sim P_z}[\log(1 - D(G(z)))] \tag{4.2}$$

5. Conclusion

The landscape of financial time-series forecasting has undergone a profound transformation, evolving from simplistic statistical methods to highly sophisticated, multimodal deep learning frameworks. This tutorial has systematically outlined the necessary architectural progression required to build resilient

predictive systems. We established that robust feature engineering—whether through classical technical indicators or advanced tensor transformations—remains the critical foundation for mitigating market noise and preparing data pipelines.

By examining the single-model paradigm, we demonstrated that different standalone architectures excel at distinct mathematical tasks: recurrent networks capture local sequential momentum, pure MLP structures decompose highly interpretable trends, and tree-based algorithms efficiently navigate complex tabular data. However, the true breakthrough in modern financial forecasting lies in the generative shift. Framing continuous numerical time-series as language through precise tokenization mechanisms allows quantitative developers to harness the unprecedented zero-shot extrapolation capabilities of Large Language Models (LLMs), bypassing the need for extensive task-specific retraining.

Ultimately, the most powerful predictive solutions are realized through advanced hybrid frameworks. By surgically combining the structural self-attention of Transformers, the causal rigor of Temporal Convolutional Networks, and the unstructured text processing of models like FinBERT, developers can construct holistic systems that simultaneously evaluate quantitative price movements and qualitative market sentiment. As the quantitative industry continues to pivot toward massive Time-Series Foundation Models, mastering this deep integration of signal processing, discrete tokenization, and generative attention will be the defining competitive advantage for the next generation of algorithmic developers.

6. References

1. Agrawal, M., Khan, A. U., & Shukla, P. K. (2019). Stock price prediction using technical indicators: a predictive model using optimal deep learning. Learning, 6(2), 7.
2. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735-1780.
3. Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18(5-6), 602-610.
4. Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv preprint arXiv:1905.10437.
5. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 30.
6. Zhu, Z., et al. (2024). Tokenizing Stock Prices for Enhanced Multi-Step Forecast and Prediction. International Conference on Neural Information Processing (ICONIP). Springer.
7. Gruver, N., et al. (2023). Large language models are zero-shot time series forecasters. Advances in Neural Information Processing Systems, 36, 19622-19635.
8. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.
9. Wang, S. (2023). A Stock Price Prediction Method Based on BILSTM and Improved Transformer. IEEE Access, 11, 104211-104223.
10. Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2016). Temporal convolutional networks for action recognition and video analysis. arXiv preprint arXiv:1611.05267.
11. Li, S., & Xu, S. (2025). Enhancing stock price prediction using GANs and transformer-based attention mechanisms. Empirical Economics, 68, 373-403.
12. Araci, D. (2019). FinBERT: Financial sentiment analysis with pre-trained language models. arXiv preprint arXiv:1908.10063.
13. Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.
14. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27.