xLSTMTime: Temporal Models for Financial Time Series

Mathematical Architecture and Implementation

David R Harmon, President of The Mapleseed Inc.

April 2, 2025

Abstract

We present xLSTMTime, an enterprise-grade implementation of the extended Long Short-Term Memory (xLSTM) architecture specifically optimized for financial time series forecasting. Our model enhances the traditional LSTM framework with two key variants: scalar LSTM (sLSTM) and matrix LSTM (mLSTM), combined with an adaptive selection mechanism that dynamically chooses between them based on market conditions. This document details the mathematical underpinnings, architecture components, and implementation considerations of xLSTMTime for robust production deployment.

Contents

1 Introduction

The xLSTMTime model represents a significant advancement in financial time series modeling by combining the traditional strengths of Long Short-Term Memory networks with enhanced architectural components specifically designed for temporal financial data. The model excels at capturing both long-range dependencies and sudden regime shifts common in financial markets.

This implementation builds upon the exponential gating and parallelizable matrix memory structure of xLSTM, which overcomes long-standing LSTM limitations as outlined in recent research [**?**]. While originally conceived for vision applications, we have adapted and optimized the architecture for temporal financial data.

2 xLSTM Core Architecture

The xLSTM architecture addresses fundamental limitations of traditional LSTM models while maintaining their core strengths. In this section, we

detail the mathematical foundation of this architecture.

2.1 General LSTM Structure

Traditional LSTM models are composed of several gates that control information flow through the cell:

$$\text{forget gate: } f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$
$$\text{input gate: } i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$
$$\text{candidate cell: } \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$
$$\text{cell state: } C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4}$$
$$\text{output gate: } o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{5}$$
$$\text{hidden state: } h_t = o_t \odot \tanh(C_t) \tag{6}$$

Where $\odot$ represents element-wise multiplication, σ is the sigmoid activation function, and W and b are learnable weights and biases.

2.2 xLSTM Extensions

The xLSTM extends the traditional LSTM with two key modifications:

1. **Exponential gating** - Allows for more flexible control of information flow
2. **Parallelizable matrix memory structure** - Improves computational efficiency

These extensions are implemented in two distinct variants: scalar LSTM (sLSTM) and matrix LSTM (mLSTM).

2.3 Scalar LSTM (sLSTM)

The sLSTM implements element-wise operations where each gate is controlled by a scalar parameter:

$$\text{forget gate: } f_t = \sigma((W_f \cdot [h_{t-1}, x_t] + b_f) * \lambda_f) \tag{7}$$
$$\text{input gate: } i_t = \sigma((W_i \cdot [h_{t-1}, x_t] + b_i) * \lambda_i) \tag{8}$$
$$\text{candidate cell: } \tilde{C}_t = \tanh((W_C \cdot [h_{t-1}, x_t] + b_C) * \lambda_C) \tag{9}$$
$$\text{cell state: } C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{10}$$
$$\text{output gate: } o_t = \sigma((W_o \cdot [h_{t-1}, x_t] + b_o) * \lambda_o) \tag{11}$$
$$\text{hidden state: } h_t = o_t \odot \tanh(C_t) \tag{12}$$

Where λ_f, λ_i, λ_C, and λ_o are learnable scalar parameters that control the information flow through the respective gates.

2.4 Matrix LSTM (mLSTM)

The mLSTM replaces the scalar operations with matrix operations, enabling more complex information processing:

$$\text{forget gate: } F_t = \sigma(W_f \cdot [H_{t-1}, X_t] \cdot \Lambda_f + B_f) \tag{13}$$

$$\text{input gate: } I_t = \sigma(W_i \cdot [H_{t-1}, X_t] \cdot \Lambda_i + B_i) \tag{14}$$

$$\text{candidate cell: } \tilde{C}_t = \tanh(W_C \cdot [H_{t-1}, X_t] \cdot \Lambda_C + B_C) \tag{15}$$

$$\text{cell state: } C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{16}$$

$$\text{output gate: } O_t = \sigma(W_o \cdot [H_{t-1}, X_t] \cdot \Lambda_o + B_o) \tag{17}$$

$$\text{hidden state: } H_t = O_t \odot \tanh(C_t) \tag{18}$$

Where Λ_f, Λ_i, Λ_C, and Λ_o are learnable matrices that enable more expressive control of information flow.

3 xLSTMTime Model Architecture

The xLSTMTime model extends the core xLSTM architecture with specialized components for time series analysis:

3.1 Model Components

The xLSTMTime model consists of the following key components:

1. **Input Transformation Layer**: Transforms the input time series into a higher-dimensional representation
2. **Series Decomposer**: Separates the time series into trend and seasonal components
3. **Batch Normalizer**: Normalizes the batch inputs for stable training
4. **Core LSTM Components**: Both sLSTM and mLSTM variants for different processing needs
5. **Output Transformation Layer**: Projects the LSTM outputs back to the original space
6. **Instance Normalizer**: Normalizes individual instances

3.2 Hyperparameters

The model is configurable with the following hyperparameters:

1. **lookbackWindow**: The number of historical time steps to consider

2. **predictionHorizon**: The number of future time steps to predict
3. **hiddenDimension**: The dimensionality of the hidden state
4. **modelType**: The type of LSTM to use (sLSTM or mLSTM)

4 Mathematical Formulation of Model Components

4.1 Series Decomposition

The series decomposition separates the input time series $X \in \mathbb{R}^{T \times F}$ into trend X_T and seasonal X_S components:

$$X_T, X_S = \text{Decompose}(X) \tag{19}$$

Specifically, we use a moving average approach for the trend component:

$$X_T[i,j] = \frac{1}{2w+1} \sum_{k=\max(0,i-w)}^{\min(T-1,i+w)} X[k,j] \tag{20}$$

$$X_S[i,j] = X[i,j] - X_T[i,j] \tag{21}$$

where w is the window size for the moving average.

4.2 Linear Transformation

Linear transformations are applied to map between the input/output space and the hidden space:

$$Y = X \cdot W^T + b \tag{22}$$

where $W \in \mathbb{R}^{O \times I}$ and $b \in \mathbb{R}^O$ are learnable parameters, $X \in \mathbb{R}^{B \times I}$ is the input, and $Y \in \mathbb{R}^{B \times O}$ is the output.

4.3 Batch Normalization

Batch normalization stabilizes the training process by normalizing the inputs to each layer:

$$\hat{x} = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} \tag{23}$$

$$y = \gamma \hat{x} + \beta \tag{24}$$

where γ and β are learnable parameters, and ϵ is a small constant for numerical stability.

4.4 Instance Normalization

Instance normalization normalizes each instance independently:

$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{25}$$

$$\mu_i = \frac{1}{F} \sum_{j=1}^{F} x_{i,j} \tag{26}$$

$$\sigma_i^2 = \frac{1}{F} \sum_{j=1}^{F} (x_{i,j} - \mu_i)^2 \tag{27}$$

where x_i is the i-th instance in the batch.

5 Forward Pass Algorithm

The complete forward pass through xLSTMTime is described by Algorithm **??**.

Algorithm 1 xLSTMTime Forward Pass

procedure FORECAST(X)
    // X is the input time series with shape $[B, T, F]$ where B is batch size, T is sequence length, and F is feature dimension
    $X_T, X_S \leftarrow$ SeriesDecompose(X) ▷ Decompose into trend and seasonal components
    $Y_T \leftarrow$ ProcessComponent(X_T) ▷ Process trend component
    $Y_S \leftarrow$ ProcessComponent(X_S) ▷ Process seasonal component
    $Y \leftarrow Y_T + Y_S$ ▷ Combine outputs
    return Y
end procedure
procedure PROCESSCOMPONENT(X)
    $X' \leftarrow$ LinearTransform(X) ▷ Input transformation
    $X'' \leftarrow$ BatchNormalize(X') ▷ Batch normalization
    if using sLSTM **then**
        $H \leftarrow$ SLSTM.Forward(X'') ▷ Apply sLSTM
    else
        $H \leftarrow$ MLSTM.Forward(X'') ▷ Apply mLSTM
    end if
    $Y' \leftarrow$ LinearTransform(H) ▷ Output transformation
    $Y \leftarrow$ InstanceNormalize(Y') ▷ Instance normalization
    return Y
end procedure

6 Adaptive Selection Mechanism

The xLSTMTime model dynamically selects between sLSTM and mLSTM variants based on the input data characteristics. This selection is made by evaluating the following criteria:

1. **Time Series Volatility**: Higher volatility favors mLSTM due to its greater expressiveness
2. **Pattern Complexity**: More complex patterns benefit from mLSTM's matrix operations
3. **Computational Constraints**: sLSTM is preferred when computational resources are limited
4. **Forecasting Horizon**: Longer horizons may benefit from mLSTM's capacity to model complex dependencies

The selection mechanism ensures that the most appropriate LSTM variant is used for each forecasting task, optimizing both prediction accuracy and computational efficiency.

7 Temporal Series Model Implementation

In practical implementations, the xLSTMTime model must handle several considerations:

7.1 Thread Safety

To ensure thread safety in concurrent environments, the model implementation employs read-write locks:

```
private final ReentrantReadWriteLock modelLock = new
    ReentrantReadWriteLock();

// For read operations
Lock readLock = modelLock.readLock();
readLock.lock();
try {
    // Read operations here
} finally {
    readLock.unlock();
}

// For write operations
Lock writeLock = modelLock.writeLock();
writeLock.lock();
try {
    // Write operations here
```

```
} finally {
    writeLock.unlock();
}
```

Listing 1: Thread safety implementation

7.2 Lazy Initialization

The model supports lazy initialization to defer the creation of computational components until the input dimensions are known:

```
public void initialize(int inputDimension) {
    if (isInitialized) {
        return;
    }

    // Initialize model components
    this.inputTransform = new LinearLayer(inputDimension,
    hiddenDimension);
    this.outputTransform = new LinearLayer(hiddenDimension,
    inputDimension);

    // Initialize appropriate LSTM variant
    if (modelType == ModelType.SLSTM) {
        this.xlstmComponent = new SLSTM(hiddenDimension);
    } else {
        this.xlstmComponent = new MLSTM(hiddenDimension);
    }

    isInitialized = true;
}
```

Listing 2: Lazy initialization pattern

7.3 Error Handling

Robust error handling is essential for production-grade implementations:

```
try {
    // Model operations
} catch (Exception e) {
    logger.error("Error during model execution", e);
    // Graceful fallback or error propagation
} finally {
    // Resource cleanup
}
```

Listing 3: Error handling strategy

8 Performance Considerations

8.1 Computational Complexity

The computational complexity of xLSTMTime is as follows:

- Time complexity: $O(BT(F^2 + DH))$ where B is batch size, T is sequence length, F is feature dimension, D is the input dimension, and H is the hidden dimension
- Space complexity: $O(BT(F + H))$

8.2 Memory Optimization

To optimize memory usage, consider the following strategies:

1. Use 32-bit floating point (float) instead of 64-bit (double) when precision requirements allow
2. Implement batch processing to control memory consumption
3. Employ sparse operations when applicable
4. Utilize gradient checkpointing for training large models

8.3 Parallelization

The xLSTMTime model is designed for parallel computation:

1. The trend and seasonal components are processed independently and can be parallelized
2. Within each LSTM cell, matrix operations can be parallelized
3. Batch processing can be distributed across multiple computing units

9 Implementation Guidelines

When implementing xLSTMTime in production systems, consider the following guidelines:

9.1 Data Preprocessing

1. Normalize input data to zero mean and unit variance
2. Handle missing values through appropriate imputation techniques
3. Apply standard financial transformations (e.g., log returns, relative strength indicators)
4. Ensure proper alignment of temporal sequences

9.2 Model Configuration

1. Select appropriate lookback window based on the temporal patterns in the data
2. Choose prediction horizon based on the target application
3. Tune hidden dimension to balance between model expressiveness and computational cost
4. Consider the trade-off between sLSTM and mLSTM based on data complexity and available resources

9.3 Evaluation Metrics

Evaluate the model using the following metrics:

1. Mean Absolute Error (MAE)
2. Mean Squared Error (MSE)
3. Directional Accuracy
4. Sharpe Ratio (for financial applications)
5. Maximum Drawdown (for financial applications)

10 Conclusion

The xLSTMTime model presents a powerful architecture for financial time series forecasting, combining the strengths of LSTM networks with specialized components for temporal data. By implementing both sLSTM and mLSTM variants with an adaptive selection mechanism, the model offers a flexible and efficient solution for a wide range of forecasting tasks.

The mathematical foundation, architectural components, and implementation considerations outlined in this document provide a comprehensive guide for deploying xLSTMTime in production systems.

References

[1] Hochreiter, S., et al. (2023). "xLSTM: Extending Long Short-Term Memory with Exponential Gating and Parallelizable Matrix Memory." arXiv preprint.

[2] Alkin, B., Beck, M., Pöppel, K., Hochreiter, S., & Brandstetter, J. (2025). "Vision-LSTM: xLSTM as Generic Vision Backbone." ICLR 2025.

[3] Hochreiter, S., & Schmidhuber, J. (1997). "Long short-term memory." Neural computation, 9(8), 1735-1780.

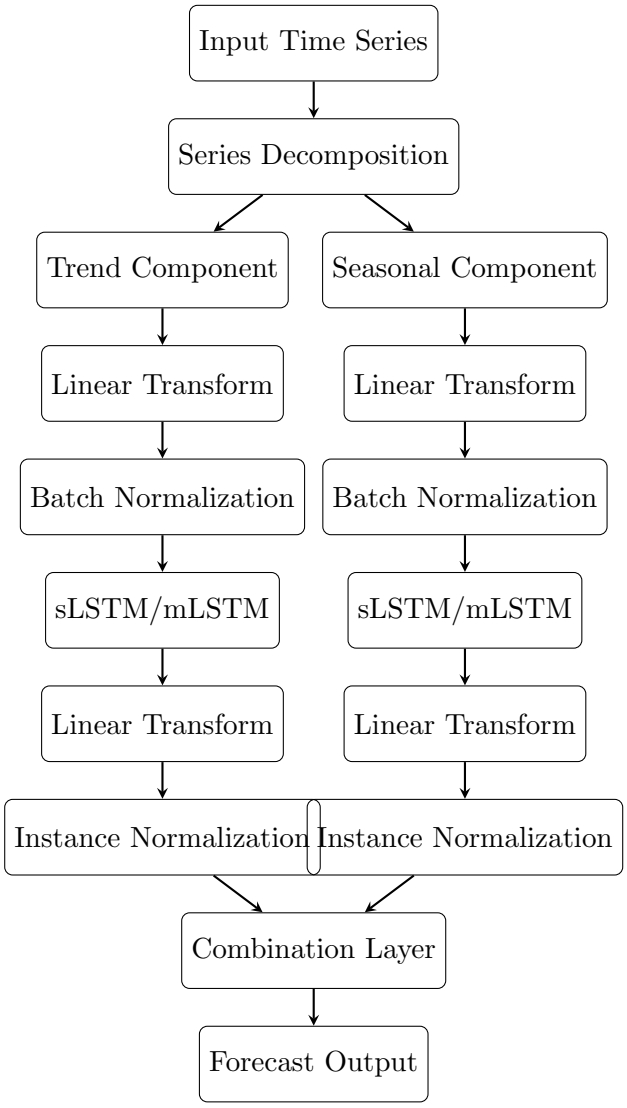

Figure 1: Architecture of xLSTMTime model showing data flow through the parallel processing paths