IJHSR
ijhighschoolresearch.org
RESEARCH ARTICLE

Benchmarking Distinct Deep Learning Architectures for Forecasting ARIMA-Synthesized Non-Stationary Time Series

Abdullah Shaikh

Jumeirah College, Dubai, UAE; abdshk21@gmail.com

ABSTRACT: Artificial Narrow Intelligence (ANI) systems now provide forecasting capabilities for non-stationary time series that match or exceed those of traditional statistical methods. Non-stationary processes are central to applications in finance, healthcare, and sports analytics, where accurate prediction is essential. To generate controlled datasets for benchmarking, we employed Autoregressive Integrated Moving Average (ARIMA) models as a classical framework for simulating non-stationary series. Using these sequences, we conducted a comparative evaluation of Long Short-Term Memory (LSTM) networks and Transformer architectures. Performance was quantified by mean squared error across a range of baseline block sizes, with data produced through custom Python pipelines. Results show that LSTMs consistently achieve lower prediction error than Transformers, with differences in mean squared error reaching statistical significance ($p < 0.05$). Transformer performance also degraded in the absence of moving average components, indicating sensitivity to residual noise. These findings suggest that LSTMs provide more stable and reliable forecasts for ARIMA-generated non-stationary data, while Transformers require additional constraints or modifications to achieve comparable robustness.

KEYWORDS: Robotics and Intelligent Machines, Machine Learning, Time Series Forecasting, Autoregressive Integrated Moving Average Models, Long Short-Term Memory, Transformer.

Introduction

Historically, classical Autoregressive Moving Average (ARMA) models assumed a stationary world, where statistical properties such as mean, variance, and autocorrelation remain constant over time. However, real-world data is rarely stationary since it exhibits stochastic volatility, regime shifts, long-term seasonality, evolving autocorrelation, and non-linear dependencies. These dynamics stretch beyond the capabilities of ARMA and its extensions, such as ARIMA (differencing) and SARIMA (seasonal). Nevertheless, recent advances in deep learning have offered promising alternatives for time series predictions. Chief amongst these are Long Short-Term Memory (LSTM) networks and Transformer-based models. In this study, we track how prediction errors evolve over block sizes, allowing us to characterize each model's stability and responsiveness in the face of non-stationary dynamics.

Traditional Statistical Models:

Diving deeper, models like ARIMA require data to be made stationary via transformations like differencing, which imposes rigidity and inhibits their capacity to adjust to volatile fluctuations. Furthermore, these models require extensive manual parameter tuning and frequently fail when confronted with noisy or sparse signals. In a comparative analysis of biomedical data, a Seasonal-ARIMA benchmark had a MAPE (Mean Absolute Percentage Error) of about 7.1%, whereas deep learning models had a MAPE of less than 3%.[1] Even more flexible statistical models, such as Meta's Prophet, outperformed ARIMA but struggled with the considerable unpredictability in patient vital signs, resulting in much greater RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) than neural networks.[1]

LSTM-Based Approaches:

In recent decades, LSTM networks have marked a paradigm shift in time series forecasting by introducing memory cells and gating mechanisms that selectively retain or discard past information. In a comparative study on the SSE 50 Index, it was found that an LSTM achieved an MAPE of 1.13%, thus significantly outperforming ARIMA in short-term forecasting accuracy, especially in the early prediction window.[2] Researchers who built upon these improvements had proposed a hybrid ensemble that integrates deep learning with gradient-boosted trees (XGBoost-DL). This model, when tested, achieved an even better accuracy, with an RMSE of 25.70 and an MAE of 19.82 on sunspot number forecasting. This model notably outperformed both SARIMA (RMSE: 54.11, MAE: 45.51) and a Transformer-based Informer model (RMSE: 29.90, MAE: 22.35).[3] The consistent reduction in error across these studies, from ARIMA to LSTM to XGBoost-DL, demonstrates the growing effectiveness of models that can flexibly learn complex, non-stationary dynamics.[3]

Transformer-Based Approaches:

Today, the latest evolution in time series forecasting is the introduction and development of Transformer-based models, consisting of a self-attention mechanism that is used to capture temporal relationships. Rather than processing inputs sequentially like an LSTM, a Transformer attends to all time steps simultaneously, learning which past observations are most relevant to forecasting the future. Transformers can learn

DOI: 10.36838/IJHSR87.71

seasonality, trends, and irregular patterns through positional encodings and attention weights, instead of assuming a fixed statistical structure. For example, recent studies introduced an Informer-based transformer for multi-domain forecasting, which considerably outperformed LSTM, GRU (Gated Recurrent Unit), and TCN (Temporal Convolutional Network) benchmarks by halving the MSE on traffic flow data (≈10.1 vs >20) and improving stock price forecast accuracy.[4] On the contrary, several other researchers note that for very long historical series or small datasets, a naive Transformer without architectural optimization may not automatically excel.[5] For example, they saw that a vanilla Transformer slightly underperformed an optimized LSTM on a decades-long stock dataset, with the LSTM yielding a lower RMSE (0.104 vs 0.232) for Hewlett–Packard stock forecasting.[5] However, the trend in recent research is that Transformer-based hybrid architectures tend to achieve the highest accuracy on non-stationary benchmarks. These include Temporal Fusion Transformers (which combine an LSTM encoder with multi-head attention for long-term features) and the Informer (which uses probabilistic sparse self-attention and a distillation mechanism to reduce redundant information).

Limitations of Current Literature:

The central tension in the current body of literature is that the pursuit of predictive accuracy is in a world governed by uncertainty, non-linearity, and evolving dynamics. While a range of models has been proposed, they come with constraints that limit generalizability and robustness across diverse real-world settings. Some of these constraints are explored ahead.

Firstly, many forecasting studies use real-world historical data where the true data-generating process is unknown, making it hard to discern which model is fundamentally better, as it might be highly likely that the deep learning network is simply overfitting the data and not necessarily learning the underlying structures within the dataset itself.[6] As a result, reliable forecasting benchmarks must no longer rely on a single test set or raw error difference alone. Instead, the need is to use statistical significance tests such as the Diebold-Mariano (DM) test[7] or the Wilcoxon signed-rank test to determine whether performance differences between models are truly meaningful or just due to chance. The widespread use of these in literature is currently not evident.

Another limitation is the inconsistent way some studies compare LSTMs and Transformers. Authors frequently evaluate proposed models against baseline architectures without performing equivalent hyperparameter tuning or ensuring consistent experimental settings, resulting in uncontrolled comparisons.[8] For example, in a recent finance study comparing LSTM and Transformer models across multiple financial instruments, the Transformer's MAPE varied from 1.34% to 2.7%, while the LSTM's MAPE remained between 1.33% and 1.51%, highlighting the Transformer's sensitivity to training conditions and data volatility.[8]

These findings indicate that unless experimental conditions are carefully controlled, one might draw wrong conclusions about which model is better.

Furthermore, several studies rely only on aggregate error metrics like MSE, RMSE, or MAE to compare models, without analyzing how errors evolve over the forecast horizon. Whilst these metrics summarize overall accuracy, they mask a model's failure to extrapolate trends or seasonality beyond the training data. For example, contemporary research demonstrated that an LSTM trained on a non-stationary series (with an upward trend) can fit short-term points well but then completely underestimate the continued growth, resulting in much larger errors farther out in time.[7] They also saw that two models with identical MSE may have very different behaviors: one might be consistently slightly off, while another is spot-on for short horizons but significantly off for long horizons. This type of long-term evaluation is currently limited in the present-day literature.

Additionally, it is also common to see papers proclaiming one model as "better" overall without accounting for contextual variability, such as the application domain and noise levels. For example, a study compared ARIMA and LSTM models for disease incidence forecasting at different time scales and found that no one model is universally best.[10] The classical ARIMA outperformed LSTM on coarser time scales (monthly and weekly predictions), while LSTM excelled on the daily forecasts.[10] The M4 and M5 forecasting competitions that involve thousands of series from finance, economics, and demographics also showed that no single method dominated across all categories.[11] Indeed, ensemble approaches and hybrids tended to do best overall by precisely adapting to different patterns. Therefore, a critical limitation in the literature currently is the lack of results stratified by context (domain type, data characteristics, forecast horizon). Without overcoming this issue, one might incorrectly assume that a model that works in one setting will work everywhere.

Key Intervention:

To address some of these gaps, this study introduces a controlled experimental framework for evaluating the performance of LSTM and Transformer models on non-stationary time series data generated using ARIMA pipelines. Unlike prior work that often relies on static benchmarks or uncontrolled real-world datasets, this setup enables fine-grained control over the degree and type of non-stationarity by systematically varying ARIMA parameters and block sizes. Model performance is evaluated over a forecast horizon using time-resolved Mean Squared Error (MSE), allowing us to examine how predictive accuracy stabilizes for each model and configuration.

In line with our commitment to overcome, or at the very least, mitigate the aforementioned limitations of existing literature, we ensured that both models are trained under consistent experimental settings with equivalent hyperparameter tuning. In addition, the Wilcoxon signed-rank test is used to statistically verify whether observed performance differences are robust or attributable to random variation. While this study focuses on baseline architectures, it also lays the groundwork for future experimentation with ensemble or hybrid forecasting models that may better accommodate contextual variability, such as application domain and noise level.

Methods

Research Design:

The research design is structured as a comparative simulation study. We generated a synthetic time series using explicitly defined ARIMA parameters. This approach offers a rare epistemic advantage: the data-generating process is fully known, allowing us to isolate model performance from the uncertainties of unknown external dynamics.

Such simulation-based forecasting benchmarks have been gaining traction for their rigor and reproducibility, as highlighted in various studies,[7] which emphasize the dangers of over-relying on real-world data without ground truth or reproducible baselines as outlined in depth previously.[9] Our methodological choice is aligned with these works but diverges by adding an incremental layer of variability: across both block sizes (from 5 to 95) and ARIMA parameter configurations ((0,1,1), (1,1,1), (2,1,2), (2,2,0))

For the purpose of acquiring an in-depth understanding of the data generation process, one must first understand the core logic behind the ARIMA model itself, which consists of three hyperparameters (p,d,q):

1. p (Autoregressive (AR)): The number of lagged observations included in the model (i.e., how many past values influence the current value). Increasing p allows the model to capture longer memory effects, but increases the risk of overfitting.

$y_t = \phi 1 y_{t-1} + \phi 2 y_{t-2} + \varepsilon_t$ for AR Order 2

2. d (Integrated (I)): The number of times the series is differenced to achieve stationarity. Differencing helps remove trends and seasonality.

- o $y_t' = y_t - y_{t-1}$ is the first difference
- o $y_t'' = y_t' - y_{t-1}$ is the second difference

3. q (Moving Average (MA)): The number of lagged forecast errors used in the model. Increasing q lets the model account for short-term shocks or noise patterns.

$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1}$ for MA(1)

The general form then becomes: $\Phi(L)(1-L)^d y_t = \Theta(L)\varepsilon_t$

Where:

- y_t is the original time series
- $\nabla^d y_t$ is the d-th order differenced time series
- ε_t is white noise error
- L is the lag operator ($L y_t = y_{t-1}$)
- $\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L_p$
  - o The ϕ coefficients control how strongly past values at different lags influence the prediction.
- $\Theta(L) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L_q$
  - o Each θ term controls how much past forecast errors influence the current value.

Another key component of the ARIMA model is the Block Size. It is the length of the forecast horizon, which represents how many time steps ahead the predictions are generated before retraining or updating the model. With smaller block sizes, it is widely believed that the accuracy will be greater as the predictions are likely to remain closer to the training range. However, with larger block sizes, the opposite is generally expected.

In the fitted ARIMA models, the estimated parameters are expressed through autoregressive (AR) and moving average (MA) coefficients, which are conventionally represented as denominator and numerator polynomials, respectively. The AR coefficients capture how past observations of the series influence its present value, while the MA coefficients reflect how past forecast errors propagate forward through the model. Together, these define the recursive structure of the time series in the time domain and can also be interpreted in the frequency domain as a transfer function given by the ratio of the two polynomials. This formulation allows insight into stability, persistence, and the filtering behavior of the model across different frequencies. Coefficients were computed in Python using the statsmodels library and are reported in the Results section to ensure transparency of model fitting and to provide a basis for comparison with the neural architectures benchmarked in this study.

For each p,d,q configuration used in this research, 1000 time steps were generated using the statsmodels library in Python, with Gaussian white noise input. The generated series were non-stationary by construction but structurally transparent, facilitating reproducibility and interpretability. To mimic batch-based processing common in neural forecasting pipelines, the full series was split into sliding window blocks of varying sizes (5 to 95, in increments of 10). Each block pair (input, target) was used to train and test the models.

Model Architectures:

In order to understand the specific model architecture used, it is imperative to first acknowledge how vanilla LSTMs and Transformers work, including the mathematical equations behind them.

LSTMs: LSTMs contain a memory cell and gating mechanisms that regulate the flow of information (Figure 1). These gates control what the network remembers, updates, and outputs at each time step, allowing LSTMs to learn both short- and long-range patterns in sequential data. The three gates are forget, input, and output gates that interact with a central cell state (memory) and a hidden state (output).[12] Below are the equations numbered sequentially from 1 to 6.

1. Forget Gate: This gate decides how much of the previous memory C_{t-1} to keep. The sigmoid function σ outputs values between 0 and 1 (0 means forget completely, 1 means keep entirely)

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

2. Input Gate: This gate controls how much of the new input x_t should be written to the memory cell. Again, values close to 1 allow more new information in.

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

DOI: 10.36838/IJHSR87.71

3. Candidate Memory (New Info to Add): This creates a candidate for new memory content, based on the current input and previous hidden state, passed through a tanh function to squash values between −1 and 1.

$\tilde{c_t} = tanh\,(W_C \cdot [h_{t-1},x_t] + b_C)$

4. Update Cell State: The memory cell is updated by combining the retained old memory C_{t-1} and the new candidate information, each weighted by its respective gate.

$C_t = f_t * C_{t-1} + i_t * \tilde{c_t}$

5. Output Gate: This gate decides how much of the memory to show as output, again using a sigmoid filter.

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

6. Compute Hidden State (Output): Finally, the hidden state output is calculated by applying tanh to the updated memory and scaling it by the output gate.

$h_t = o_t \cdot tanh\,(C_t)$

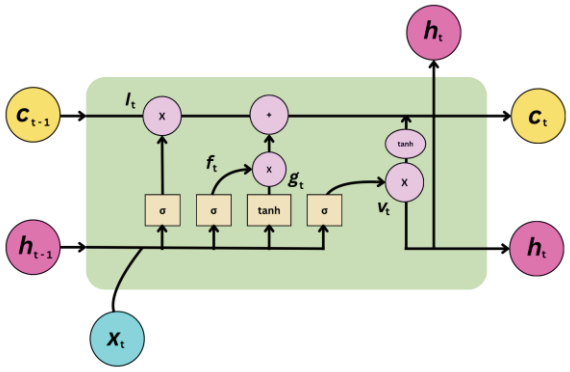

Figure 1: Diagrammatic representation of a vanilla LSTM cell. The figure illustrates how information flows through the forget gate (Equation 1), the input gate (Equation 2), and the output gate (Equation 5) to regulate updates to the cell state memory (Equation 4) and hidden state output (Equation 6). Each gate applies a sigmoid or tanh transformation to control what past information is retained and what new input is incorporated. This gating mechanism enables LSTMs to preserve long-term dependencies while flexibly adapting to short-term fluctuations, making them particularly effective for forecasting sequential and non-stationary data.

Transformers: In a transformer, there are two main parts: encoders and decoders (Figure 2). They process all steps in parallel using self-attention, allowing each input to weigh the importance of other inputs. Transformers learn patterns by passing this attention output through multilayer perceptrons, where residual connections and normalization ensure stable learning. The following steps elucidate the inner workings of a vanilla Transformer, like the one outlined by Vaswani *et al.*[13]

1. Input Embeddings and Positional Encoding: The raw input x_t is projected into a higher-dimensional space using weights W_e, and then a positional encoding is added to retain the order of the time steps.

$x_t' = x_t W_e + \mathrm{PosEnc}(t)$

2. Scaled Dot-Product Attention: Each input is then attended to all others using a weighted sum of values V, where the weights are based on the similarity (dot product) between queries Q and keys K. The division by $\sqrt{dk}$ stabilizes gradients.

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V$$

3. Multi-Head Attention: Instead of computing attention once, it is done in parallel across multiple heads, each learning different relationships. The results are then concatenated and linearly transformed.

$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1,\ldots,\mathrm{head}_h)W^O$

4. Individual Attention Head: Each head uses its own set of learned weights to compute queries, keys, and values. This lets the model learn diverse attention patterns.

$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

5. Feedforward Neural Network (FFNN): After attention, each position passes through the same fully connected network. The non-linearity (ReLU) helps capture complex patterns.

$\mathrm{FFNN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2$

6. Residual Connection and Layer Normalization: A shortcut connection adds the input back to the sublayer output (either attention or FFNN), and the result is normalized. This helps stabilize and speed up training.

$\mathrm{Output} = \mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

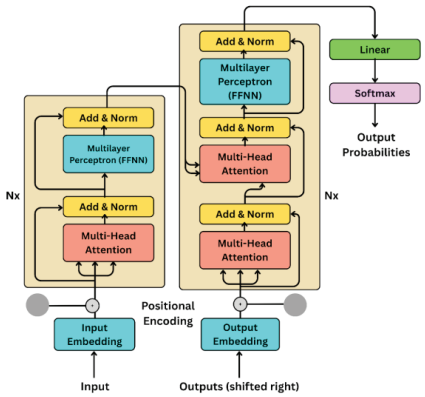

Figure 2: Diagrammatic representation of an encoder-decoder vanilla transformer, similar to the initial design presented by Vaswani *et al.* Figure highlights the parallel processing of all time steps via self-attention (Equations 2,3,4), where queries, keys, and values determine contextual relationships across the sequence. Multi-head attention allows the model to capture diverse dependencies, whilst residual connections and normalization stabilize learning. Positional encodings (Equation 1) supplement the absence of recurrence by embedding order information directly into the input.

Both models were trained on identical input–output configurations to ensure a fair comparison. Inputs consisted of single-feature time series segments of fixed block sizes (ranging from 5 to 95), with a forecasting horizon of 1. All models received the same preprocessed windowed data with no exogenous variables or pretrained components. The LSTM architecture followed a standard sequential design implemented in TensorFlow/Keras. It consisted of a single LSTM layer with 200 hidden units and ReLU activation, followed by a Dense layer with 50 ReLU units and a final Dense output layer with linear activation. No dropout was applied within the LSTM,

and the model was compiled using the Adam optimizer with a learning rate of 0.001 and an MSE as the loss function.

In our experiments, we used an encoder-only transformer, unlike the encoder-decoder Transformer in Figure 2. It is widely held true that encoders are more useful for natural language understanding, which makes them suitable for single-step regression forecasting, compared to decoders, which tend to be more apt for natural language generation instead. Our encoder-only transformer had two transformer blocks, each containing:

- Multi-head self-attention with 2 heads and head size = 64
- Layer normalization pre- and post-attention
- Dropout (0.2) applied to both attention outputs and feed-forward layers
- A feedforward sublayer using Conv1D with 64 filters and ReLU activation, followed by a linear Conv1D output layer.

Sequence outputs were aggregated using GlobalAveragePooling1D, followed by an MLP head with layers Dense(64, ReLU) → Dropout(0.2) → Dense(32, ReLU), and a final Dense output layer. Positional encodings were not used to retain simplicity. Architectural hyperparameters were selected to maintain a balance between model capacity and convergence stability, aligning with prior best practices. Both LSTM and Transformer models were trained using an 80/20 split on the ARIMA-generated datasets across all block sizes, with no separate validation set to maintain experimental consistency. Training was conducted for up to 50 epochs using the Adam optimizer (learning rate = 0.001, batch size = 64) and an MSE as the loss function. Early stopping with a patience of 10 was used to preserve generalization and retain the best model weights.

Analysis and Interrogation of Results:

For the evaluation process, we first compiled tables containing the Mean Squared Errors for each block size (from 5 to 95) for the 4 p,d,q parameters outlined earlier. In a nutshell, the MSE[14] measures how far the model's predictions are from the actual values. For each prediction, it finds the difference from the true value, squares it (to make all errors positive), and then takes the average of these squared differences. A lower MSE means the model is predicting more accurately.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- y_i: the actual (true) value at position i
- $\hat{y}_i$: the predicted value at position i
- n: the total number of predictions

Following this calculation, the MSE was then passed through the Wilcoxon Signed Rank Test, which is a non-parametric test, in order to confirm whether any difference in the MSEs was legitimate or simply random variation. As noted earlier in the introduction, we could have also chosen the Diebold-Mariano (DM) test due to its prevalence in existing literature. However, it is key to understand that the DM test assumes covariance stationarity (constant mean, constant variance, and an autocovariance dependent solely on the lag). This is not the case for our ARIMA-generated time series, and thus, the core assumption of the DM test would break down if we were to use it on our MSEs. Given that the Wilcoxon signed-rank test does not hold any such assumptions, it was deemed to be the more adept test to use. Not just that, but the use of the Wilcoxon signed rank test has been widely supported in various older[15] and contemporary studies.[16] This calculation was implemented using scipy.stats.wilcoxon. The following steps outline how the Wilcoxon signed rank test values are calculated mathematically.

1. Firstly, the Wilcoxon Signed-Rank Test calculates the difference (D_i) in the performance scores (MSE) for each paired forecast (LSTM and Transformer).
$D_i = x_i - y_i$

2. Then it removes any zero differences, ranks the absolute values of the remaining differences ($|D_i|$), and assigns the original signs (+ or –) back to each rank. It then sums the positive and negative ranks separately, and the smaller of these sums becomes the test statistic (W).

$$W = \min\left(\sum_{R_i>0} R_i\,,\ \sum_{R_i<0} |R_i|\right)$$

3. The formula below (Z) standardizes the test statistic W so it can be compared to a normal distribution.

$$Z = \frac{W - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}}$$

4. Then, the two-tailed p-value for a standard normal distribution can be calculated using the formula below. It represents the probability that a Z-score is at least as extreme as the observed value |z|, multiplied by 2 to account for both tails of the distribution.
$p = 2 \cdot P(Z \geq |z|)$

5. Finally, a statistically significant result is one where the p-value is below the actual significance level, indicating that one model consistently outperforms the other across the paired forecasts and that the results are not random.

Beyond the use of the MSE, we are also noting 2 other key metrics: MAE and MAPE.[17] These 2 metrics allow us to compare model performance in relative percentage terms and interpret the typical deviation better if needed.

On one hand, the MAE measures the average of the absolute differences between predicted values and actual values. The formula it utilizes is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

On the other hand, the MAPE expresses prediction error as a percentage of the actual values. It tells us how large our errors are relative to the true values. The key equation here is:

DOI: 10.36838/IJHSR87.71

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Additional Implementation Details:

For ARIMA, we employed a train–test split rather than introducing an additional validation set. In classical time-series modeling, the primary safeguard against overfitting lies in ensuring that model residuals remain well-behaved and that the fitted coefficients generalize across the observed series. Because ARIMA relies on sequential dependence, introducing a validation split would require withholding contiguous future values, which are more appropriately reserved for final out-of-sample evaluation. Moreover, the models exhibited stable convergence and consistent performance during training, reducing the need for iterative hyperparameter tuning through a separate validation set. Randomization of the data was not applied, as preserving temporal ordering is essential for autoregressive structures. Thus, the simpler train–test scheme was sufficient to evaluate forecasting performance while maintaining methodological rigor.

Also, all experiments were implemented in Python 3.10 and additional NumPy, SciPy and statsmodels libraries were used for time series data generation. Training and evaluation were conducted on Google Colab Pro, leveraging an NVIDIA T4 GPU for accelerated computation. Random seeds were fixed (np.random.seed(42)) at all stages to ensure reproducibility across runs and models. Visualizations were rendered using Matplotlib.

Results

To recap, for each configuration of the ARIMA Model, we evaluate model performance over varying block sizes (lag lengths) using Testing MSE as the primary metric. All experiments maintain d≥1 to ensure non-stationarity, in alignment with the study's focus. Additionally, the code was configured to follow a 'greedy' approach[18] where the number of epochs was not fixed, but rather depended on the point at which the validation loss plateaued or began to increase.

For reference, the ARIMA coefficients (AR and MA parts) for each configuration were:

- (0,1,1)
  - o Generated MA (Numerator) Coefficients: [1. -1.5]
  - o Generated AR (Denominator) Coefficients: [1.]
- (1,1,1)
  - o Generated MA (Numerator) Coefficients: [1. -1.5]
  - o Generated AR (Denominator) Coefficients: [1. -0.8]
- (2,1,2)
  - o Generated MA (Numerator) Coefficients: [1. 0. -2.25]
  - o Generated AR (Denominator) Coefficients: [1. 0. -0.64]
- (2,2,0)
  - o Generated MA (Numerator) Coefficients: [1.]
  - o Generated AR (Denominator) Coefficients: [1. 0. -0.64]

As mentioned earlier, the experiment incorporates the Wilcoxon Signed Rank Test to assess the significance of the results. This is explored further below:

Table 1: Wilcoxon signed-rank test results comparing LSTM and Transformer prediction errors across ARIMA configurations. Negative Z-scores indicate cases where LSTMs achieved lower mean squared errors than Transformers, while extremely small p-values ($p < 0.05$) confirm that these differences are statistically significant rather than random variation.

Dataset	W (Wilcoxon)	Z-Score	P-value
0,1,1	0	-8.862	3.90E-18
1,1,1	255	-7.805	5.95E-15
2,1,2	1920	-2.08	3.75E-02
2,2,0	970	-5.347	8.96E-08

The results in Table 1 indicate statistically significant differences in prediction errors between LSTM and Transformer models across all ARIMA configurations.

When interpreting the results seen in Figure 3, we can note the following for varying block size ranges:

• Small Block Sizes (5-25): LSTM consistently outperforms the Transformer in 3 out of 4 ARIMA configurations. It achieves particularly low MSE values in complex setups like ARIMA(2,1,2) and ARIMA(2,2,0), with averages around 0.028. In contrast, Transformer performance is highly unstable in these settings, with MSEs ranging between 0.015 and 0.315.

• Medium Block Sizes (35-65): LSTM models demonstrate consistent and stable performance across all ARIMA configurations, though without achieving the lowest errors, with MSE values typically ranging from 0.045 to 0.079 on average. Transformers show erratic behavior, performing well under ARIMA(2,1,2) with a low MSE of 0.019, but yielding significantly higher errors in ARIMA(0,1,1) and ARIMA(2,2,0), where MSE exceeds 0.09 and even 0.17.

• Large Block Sizes (75-95): LSTM models consistently achieve their lowest errors, most notably under ARIMA(2,1,2) with an average MSE of 0.014. Additionally, the Transformer model's performance can also be seen to improve.

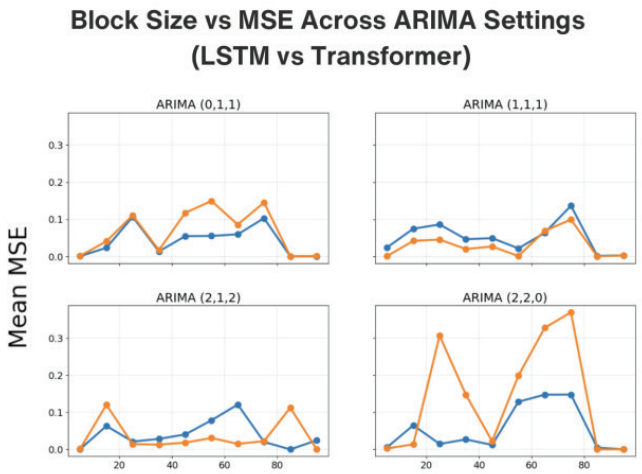

Figure 3: Mean testing MSE vs. block size for LSTM and Transformer models across different ARIMA configurations. Each subplot uses a consistent x and y-axis scale to enable direct visual comparison. The curves show that LSTMs generally maintain stable performance as block size increases, with error values clustering within a narrow band. In contrast, Transformer performance fluctuates sharply depending on block size and ARIMA parameters, with notable instability in (0,1,1) and (2,2,0).

Moreover, as seen below, Figure 4 displays the ΔMSE (LSTM MSE – Transformer MSE) across block sizes for all four ARIMA configurations, highlighting comparative model performance. Across most block sizes and configurations, negative ΔMSE values dominate, indicating that LSTM outperforms Transformer more frequently. The performance gap is especially pronounced under ARIMA(2,2,0), where LSTM achieves up to 0.29 lower MSE than Transformer, particularly in the small- to mid-range block sizes. ARIMA(2,1,2) shows more balanced results, with LSTM and Transformer alternating in dominance depending on block size. For ARIMA(1,1,1), Transformer generally maintains a slight edge throughout, while ARIMA(0,1,1) demonstrates the narrowest margins and highest variability in relative performance.

Figure 4: Difference in mean testing MSE (ΔMSE = LSTM – Transformer) across block sizes for each ARIMA configuration. Negative ΔMSE values indicate LSTM superiority, while positive values reflect cases where the Transformer performs better. These patterns underscore how LSTM resilience stems from its recurrent inductive bias, while Transformer performance is highly configuration-sensitive.

Beyond evaluating block size dependencies, it was imperative to investigate how variations in the ARIMA (p,d,q) parameters influence the performance. Across configurations, as seen in Figure 5, the LSTM consistently demonstrated a smooth and resilient performance curve. In contrast, the Transformer's performance was notably more configuration-sensitive. It only matched or slightly outperformed the LSTM under the baseline ARIMA(1,1,1) setting, which offers a balanced temporal dynamic. However, under more complex conditions, particularly with increased differencing (d) or autoregressive terms (p), its performance deteriorated significantly. In the ARIMA(0,1,1) configuration, where the autoregressive term is eliminated, LSTM performance remains relatively stable. However, Transformer models exhibit a moderate increase in MSE and greater variability across block sizes. Then, in the ARIMA(2,2,0) configuration, which removes the MA component while increasing differencing and autoregression, the most significant decline in Transformer performance takes place. Starkly high MSE values indicate a lack of adaptability to noise-driven or shock-driven memory structures. Lastly, in higher complexity configurations (a greater number of coefficients go into the ARMA), like ARIMA(2,1,2), both models exhibit improved performance, with LSTM maintaining a slight edge (although the difference in performance is negligible).

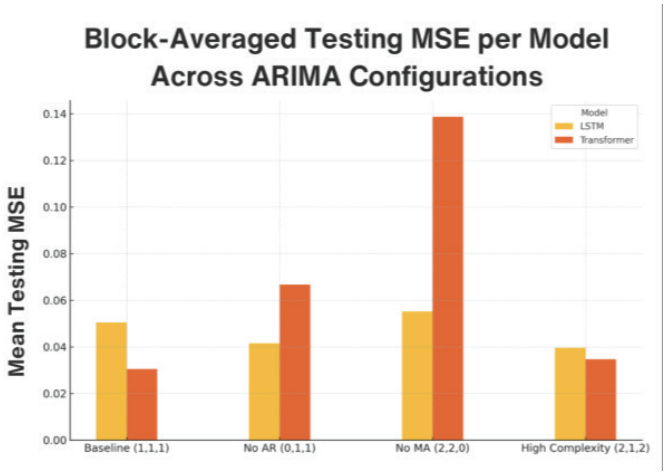

Figure 5: Average testing MSE of LSTM and Transformer models across four ARIMA configurations. When autoregressive or moving average components are removed, as in (0,1,1) and (2,2,0), Transformer performance deteriorates substantially while LSTMs remain stable. In the higher-complexity (2,1,2) setting, both models improve, though LSTM maintains a slight edge. This figure illustrates that while Transformers can excel in balanced conditions, they are far more sensitive to shifts in ARIMA dynamics, whereas LSTMs display consistent adaptability.

It is also essential to note that although MAE and MAPE were computed during experimentation, as outlined in the Methods section, they were not included in the main results for several reasons. Firstly, the models were trained to minimize MSE loss, so using MSE as the primary evaluation metric ensured direct alignment with the training objective. Additionally, relying on a single metric allowed for consistent and fair comparisons across different models and ARIMA configurations. Moreover, MAPE can become unstable or undefined when actual values are close to zero, a limitation encountered in almost all ARIMA configurations that use a block size of 85 and 95, making it less reliable for comprehensive reporting.

■ Discussion

Across all experimental configurations, the core insight was that vanilla LSTM models demonstrate greater resilience and adaptability than vanilla Transformers. Figure 4 reveals that the reason why LSTMs are better suited for sequential data is due to their recurrent design, which allows them to preserve contextual information over time and adapt robustly to changes in AR and MA components. Conversely, the Transformer, which relies on self-attention mechanisms rather than recurrence, exhibits greater sensitivity to the nature of the time series. Its performance peaks in the ARIMA(1,1,1) configuration, where both AR and MA components are present in moderate quantities. However, as the structure becomes skewed, either by removing AR in (0,1,1) or MA in (2,2,0), the Transformer struggles to adapt. Notably, the absence of the MA component results in a substantial performance drop, indicating potential difficulty in capturing residual-based or noise-driven correlations. These findings suggest that Transformers may lack the

DOI: 10.36838/IJHSR87.71

inductive bias necessary to generalize across varying degrees of non-stationarity.

The observed performance patterns offer several implications for practitioners working with non-stationary time series in domains such as finance, climate science, energy forecasting, and economics. First, the consistent superiority of LSTM models suggests that they are likely to perform better in real-world scenarios where there is a combination of noise-heavy settings, regime shifts, seasonal drifts, or shock-driven fluctuations. In contrast, Transformers may be appropriate in environments where the data exhibits balanced or well-structured dependencies, such as periodic sensor signals or clean transactional logs. However, their heightened sensitivity to block size and time series structure implies a greater need for configuration tuning and prior knowledge of the data-generating process.

Moreover, across all ARIMA configurations, the choice of block size (i.e., input sequence length) emerged as a key determinant of forecasting performance. LSTM models displayed a characteristic low U-shaped error curve, with degraded performance at both extremes while achieving optimal accuracy in the mid-to-high block size range. This suggests that LSTMs benefit from a balanced input window: one that is long but not so long that noise or redundant information accumulates. For Transformers, the relationship with block size was less stable and more configuration-dependent. In some settings, such as ARIMA(1,1,1), longer block sizes improved performance, whereas in others, like ARIMA(2,2,0), performance worsened with increasing sequence length. This inconsistency highlights the need for careful block size tuning. From a practical standpoint, these results position block size as a first-order hyperparameter, on par with learning rate or architecture depth. Selecting an inappropriate sequence length can lead to substantial drops in predictive accuracy, particularly for Transformer models. Therefore, automated tuning strategies or block-size-adaptive architectures may be worth exploring in future work.

While the experimental framework provides valuable insights into model behavior under controlled non-stationary settings, several limitations must be acknowledged. First, the analysis compares only baseline implementations of LSTM and Transformer architectures, without exploring architectural variants (e.g., BiLSTM, Informer, or Transformer-XL) that may offer enhanced performance on time series data. Additionally, only a fixed forecasting horizon and evaluation metric were used for comparison. Incorporating other perspectives, such as long-horizon degradation analysis, time-weighted metrics, or probabilistic error measures, could yield richer model diagnostics. Secondly, the statistical testing approach, while robust in comparing paired samples, relies on a finite set of repeated runs and does not fully capture within-run stochasticity. This may underrepresent variance in model training outcomes, particularly for Transformer models, which are known to be sensitive to initialization and convergence dynamics. Addressing these limitations in future work will help to strengthen the reliability and applicability of the study's conclusions.

Additionally, further investigation is warranted into the modeling of forecast error over time. The future direction of research would be to characterize how prediction errors evolve across the forecast horizon and attempt to fit polynomial models to time-resolved MSE curves. This may offer deeper insight into extrapolation dynamics and would directly address the open question of whether errors can be predictably modeled as a function of time in non-stationary settings. Given the sensitivity of both architectures to input sequence length, integrating dynamic lag windows or attention masks may lead to more stable and efficient forecasting pipelines. These extensions would further bridge theoretical understanding with practical forecasting performance.

Conclusion

In conclusion, these findings carry important implications for the broader research community working on deep learning for time series forecasting. While Transformers have recently garnered widespread attention for their success in natural language and vision tasks, this study reaffirms that their advantages do not straightforwardly translate to non-stationary time series prediction, especially under noisy or structurally imbalanced conditions. Our approach complements emerging studies that seek to reintroduce domain structure into neural forecasting (e.g., AR-Net,[19] N-BEATS[20]), positioning our work as both a caution and a contribution: cautioning against overgeneralizing Transformer utility in non-stationary domains, and contributing new methodology to diagnose temporal robustness in deep sequence models.

Acknowledgments

I would like to extend my heartfelt gratitude to Dr. Eric Sakk, Associate Professor of Computer Science at Morgan State University, for his mentorship and scholarly insight throughout this research. His deep expertise in artificial intelligence and time series analysis provided a valuable intellectual framework, particularly during the initial stages of experimental design. Moreover, Dr. Sakk's thoughtful feedback and academic encouragement greatly enhanced the rigor and clarity of my approach. I am truly grateful for his support and commitment to guiding young researchers in the pursuit of meaningful scientific inquiry. Additionally, I attest that the ideas, graphics, and writing in this paper are entirely my own.

References

1. Ni, H.; Meng, S.; Geng, X.; Li, P.; Li, Z.; Chen, X.; Wang, X.; Zhang, S. Time Series Modeling for Heart Rate Prediction: From ARIMA to Transformers; 2024; pp 584–589.
2. Hao, J.; Li, X.; Lv, Y.; Xiao, N. A Comparative Study on SSE 50 Index Prediction Based on ARIMA Model and LSTM Neural Network. In *2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*; IEEE: Taiyuan, China, 2021; pp 216–219. https://doi.org/10.1109/mlbdbi54094.2021.00049.
3. Dang, Y.; Chen, Z.; Li, H.; Shu, H. A Comparative Study of Non-Deep Learning, Deep Learning, and Ensemble Learning Methods for Sunspot Number Prediction. *Applied Artificial Intelligence* **2022**, *36* (1). https://doi.org/10.1080/08839514.2022.2074129.

4. Bhogade, V.; Nithya, B. Time Series Forecasting Using Transformer Neural Network. *International Journal of Computers and Applications* **2024**, *46* (10), 880–888. https://doi.org/10.1080/1206212x.2024.2396321.
5. Sonata, I.; Heryadi, Y. Comparison of LSTM and Transformer for Time Series Data Forecasting. In *2024 7th International Conference on Informatics and Computational Sciences (ICICoS), IEEE: Semarang, Indonesia*, 2024; pp 491–495. https://doi.org/10.1109/icicos62600.2024.10636892.
6. Hewamalage, H.; Ackermann, K.; Bergmeir, C. Forecast Evaluation for Data Scientists: Common Pitfalls and Best Practices. *Data Min Knowl Disc* **2023**, *37* (2), 788–832. https://doi.org/10.1007/s10618-022-00894-5.
7. Klyushin, D. Nonparametric Method for Estimation of Forecasting Models Using Small Samples. https://ceur-ws.org/Vol-3312/paper11.pdf (accessed 2025-07-23).
8. Bączek, J.; Zhylko, D.; Titericz, G.; Darabi, S.; Puget, J.-F.; Putterman, I.; Majchrowski, D.; Gupta, A.; Kranen, K.; Morkisz, P. TSPP: A Unified Benchmarking Tool for Time-Series Forecasting. arXiv January 8, 2024. https://doi.org/10.48550/arXiv.2312.17100.
9. Ruiru, D. K.; Jouandeau, N.; Odhiambo, D. LSTM versus Transformers: A Practical Comparison of Deep Learning Models for Trading Financial Instruments. In *Proceedings of the 16th International Joint Conference on Computational Intelligence*; SCITEPRESS - Science and Technology Publications: Porto, Portugal, 2024; pp 543–549. https://doi.org/10.5220/0012981100003837.
10. Zhang, R.; Song, H.; Chen, Q.; Wang, Y.; Wang, S.; Li, Y. Comparison of ARIMA and LSTM for Prediction of Hemorrhagic Fever at Different Time Scales in China. *PLoS ONE* **2022**, *17* (1), e0262009. https://doi.org/10.1371/journal.pone.0262009.
11. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V. M5 Accuracy Competition: Results, Findings, and Conclusions. *International Journal of Forecasting* **2022**, *38* (4), 1346–1364. https://doi.org/10.1016/j.ijforecast.2021.11.013.
12. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. Neural Computation 1997, 9 (8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.
13. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. arXiv August 2, 2023. https://doi.org/10.48550/arXiv.1706.03762.
14. James, G.; Witten, D.; Hastie, T.; Tibshirani, R.; Taylor, J. *An Introduction to Statistical Learning with Applications in Python*; 2023.
15. Dems˘ar, J.; Demsar, J. Statistical Comparisons of Classifiers over Multiple Data Sets (2006).
16. Nguyen, T. S.; Nguyen, D. M. D.; Nguyen, V. T. Empirical Comparison of Lightweight Forecasting Models for Seasonal and Non-Seasonal Time Series. arXiv 2025. https://doi.org/10.48550/ARXIV.2505.01163.
17. Adhikari, R.; Agrawal, R. K. An Introductory Study on Time Series Modeling and Forecasting. **2013**. https://doi.org/10.48550/ARXIV.1302.6613.
18. Greedy Layer-Wise Training of Deep Networks. *Advances in Neural Information Processing Systems 19* **2007**, 153–160. https://doi.org/10.7551/mitpress/7503.003.0024.
19. Triebe, O.; Laptev, N.; Rajagopal, R. AR-Net: A Simple Auto-Regressive Neural Network for Time-Series. arXiv November 27, 2019. https://doi.org/10.48550/arXiv.1911.12436.
20. Oreshkin, B. N.; Carpov, D.; Chapados, N.; Bengio, Y. N-BEATS: Neural Basis Expansion Analysis for Interpretable Time Series Forecasting. arXiv February 20, 2020. https://doi.org/10.48550/arXiv.1905.10437.

Authors

Abdullah Shaikh, a rising senior from Jumeirah College, Dubai, is a dynamic innovator leading a Gen Z financial literacy movement by pioneering AI-driven financial tools and hosting insightful entrepreneurship podcasts. His research explores how deep learning techniques can aid in time series forecasting, especially for financial applications. Beyond academic pursuits, Abdullah has won global debate and STEM competitions, where he blends technical acumen with a passion for impact.

DOI: 10.36838/IJHSR87.71