Volume 6 Issue 1 (March 2026)
DOI 10.24191/jaeds.v6i1.164
eISSN 2805-5756

Journal of Applied Engineering Design and Simulation

MLP Sliding-Window Forecasting for Electricity Load Prediction: A Multi-Scale Evaluation using an ONNX-based Java Framework

Farid Morsidi[1,*], Asma Hanee Ariffin[2], Rohaizah Abdul Wahid[2]
[1]Computing Department, Faculty of Computing & Meta-Technology, Universiti Pendidikan Sultan Idris, 35900 Tanjong Malim, Perak, Malaysia
[2]Department of Software Engineering and Smart Technology, Universiti Pendidikan Sultan Idris, 35900 Tanjong Malim, Perak, Malaysia
*corresponding author: farid.mors90@gmail.com

ABSTRACT

Deep learning models for time-series forecasting offer significant potential but face deployment challenges. This paper advances a modular framework for deploying ONNX models in Java, building on previous work. The research paper assesses the MLP sliding-window architecture through different time intervals which extends the previous LSTM-based system to establish primary performance standards that future research can use for comparison. The framework incorporates additional features for enhanced evaluation, including five metrics (MAE, RMSE, MAPE, SMAPE, R^2), automated archiving, and high-quality graphical exports. Evaluations using electricity consumption data (ETTh1) and benchmark datasets show that the MLP sliding-window model outperforms LSTM in terms of R^2, MAE, and MAPE, achieving scores of 0.9895, 0.5907, and 7.19%, respectively, indicating high accuracy across various domain. The framework also implements a custom threshold ($\varepsilon = 1e{-}10$) to handle near-zero values in MAPE calculation, ensuring reliable result storage for reproducibility. The findings reveal that simpler MLP designs can either match or exceed LSTM performance in specific scenarios while offering faster processing and easier implementation. Detailed comparisons and guidelines for deployment are included, along with insights into model selection criteria.

Keywords: Time-series forecasting; deep learning; ONNX; Multi-layer perceptron; LSTM

Abbreviations

ONNX	Open Neural Network Exchange
MLP	Multi-Layer Perceptron
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
MAPE	Mean Absolute Percentage Error
SMAPE	Symmetric Mean Absolute Percentage Error
R^2	Coefficient of Determination
ARIMA	Autoregressive Integrated Moving Average
ETTh1	Electric Transformer Temperature – Hourly
DJL	Deep Java Library
csv	comma separated value

1.0 INTRODUCTION

Time-series prediction is applied in most areas to make decision-making more accurate. This is especially true for the fields of environmental monitoring, energy management, financial trading, and manufacturing, where the benefits of accurate predictions can range from better resource positioning and risk reduction to higher efficiency. Such challenges lead to more reliance on deep learning [1], [2], [3] since traditional statistical approaches like ARIMA and smoothing have difficulty handling complex nonlinear patterns. Long Short-Term Memory (LSTM) networks, often referred to as the best technique for time-series data due to their ability to perfectly depict the complex dependencies, are the ones who have managed to overcome the vanishing gradient problem which is common in the standard recurrent networks [4]. Nevertheless, LSTMs are not without their drawbacks as they make the implementation more difficult since they require sequential processing and, they create problems in migrating from Python environments to the production. To tackle these difficulties, this study context created a modular Java application of ONNX Time Series Prediction Models by means of LSTM networks for precipitation estimation [4], thereby making a prototype to investigate the issues related to implementation. The proposed

Received on 24.01.2026
Accepted on 10.03.2026
Published on 30.03.2026

framework utilises the Deep Java Library (DJL) for inferencing across various platforms. The research work is mainly about the model selection and its performance assessment accompanied by a detailed review of the sliding window method against LSTM and MLP architectures across the entire datasets. The study outcome reveals some inadequacies in time-series forecasting research, mainly the lack of comparative studies and the ambiguity of performance metrics such as MAPE. The objective is to perform a very detailed visual and functional evaluation of LSTM and MLP models with respect to prediction accuracy, model performance across different temporal resolutions within the electricity domain, and practical implementation guidelines. The research mainly supports the prediction of the next 24 hours but also considers the future multi-day predictions and datasets with missing values. The sections cover the review of previous work, framework structure, methods, results, and finally, they present the main findings and directions for the future.

2.0 RELATED WORK

The landscape of time-series prediction has undergone a major shift, largely due to the integration of deep learning techniques. This progress has moved forecasting beyond the realm of classic statistical approaches, such as ARIMA models [1], [4], by enabling the capture of intricate, non-linear relationships within data. Long Short-Term Memory (LSTM) networks have gained considerable traction owing to their inherent architecture, which employs gating systems to effectively model dependencies that span extended periods, thus excelling with data exhibiting complicated temporal dynamics [5], [6]. Gated Recurrent Units (GRUs) offer a substitute that delivers similar effectiveness yet necessitates lower computational demands [7], [8]. Furthermore, transformer designs incorporating attention mechanisms are proving very effective for projects involving extremely long sequences. However, employing more straightforward Multi-Layer Perceptron (MLPs) alongside a rolling window approach has demonstrated performance comparable to state-of-the-art models, often distinguished by faster deployment times and simpler embedding into operational systems [9], [10], [11]. Investigations spanning various predictive domains—such as utility usage forecasting, ecological indicators, and capital market modelling—reliably show that the optimal architectural selection is heavily dependent on the unique characteristics of the dataset, the temporal extent of the observations, and the complexity inherent in the modelled dynamics; this means that a single dominant model cannot universally outperform all others in every context [12], [13], [14]. Standardized testing grounds, such as the ETTh1 (Electricity Transformer Temperature) dataset and various weather data collections, facilitate objective performance measurement. However, the bulk of current research tends to concentrate evaluation within a single subject area, thereby restricting understanding of how well these models can generalize their predictive power across different fields [15], [16].

Model performance evaluation and selection are done with the help of evaluation metrics. There are many traditional metrics (for instance, MAE, RMSE and MAPE) commonly used [11], [17], but MAPE is known to perform poorly about zero values [18] as well as having asymmetrical penalties, both of which are problematic for environmental data. Newer metrics like SMAPE and R^2 address the issues with MAPE and allow for the evaluation of multiple aspects of forecasting performance simultaneously [19]. SMAPE uses both the actual and predicted values to create a denominator of absolute value to create a more symmetrical error evaluation [20]. R^2 quantifies how much of the variance in forecasting is accounted for by the predictions made [19]. Recent studies have shown that there should be a move towards multi-metric evaluation [7], [11], [21], as evaluating with a single metric can lead to incorrect conclusions: different metrics will each highlight different aspects of forecasting ability, and it is possible to have a model that has a low RMSE but a poor-performing (percentage-based) model, while a different model has an excellent-performing (percentage-based) MAPE but a large amount of absolute error in magnitude.

The Open Neural Network Exchange (ONNX) format allows users to create and use machine learning models developed using different frameworks and programming languages [22]. This means that a model trained on one framework such as the widely revered PyTorch or TensorFlow can be executed in a different programming language by using the appropriate runtime [4]. Furthermore, deploying ML-based applications using Java provides many benefits for businesses, such as strong typing, mature toolsets, and diverse library ecosystems for integrating with existing systems. The Deep Java Library (DJL) provides a high-level API for making predictions with deep learning in Java, while also supporting several back-end technologies, including ONNX Runtime [23], [24].

In the previous published study [4], it was demonstrated that deploying ONNX models in Java for time-series forecasting is feasible through a framework that enables data loading, inference, visualization, and exporting results using a JavaFX user interface [4]. However, there were no comparisons of the results from using different types of ML models and datasets, and the evaluation metrics were not complete or comprehensive. This paper aims to address these issues by systematically comparing MLP and LSTM architectures using multiple datasets and adding more evaluation metrics, thus providing empirical data to support the selection of one architecture over another, as well as offering a production-ready framework for reproducible research and deployment of ML models. The electricity domain was chosen as the sole application area for two reasons: (i) the ETT (Electricity Transformer Temperature) datasets are widely used benchmarks that enable direct comparison with published results; and (ii) the prior framework [4] was designed and validated for electricity load data, making this domain

the natural extension for the comparative study. While cross-domain generalization is an important research direction, it is acknowledged as a limitation of the current study and is identified as a priority for future work.

3.0 METHODOLOGY

3.1 Framework architecture

Figure 1 depicts the arrangement of the system, featuring six linked elements managed by the *ForecastController* class. Information moves sequentially from the CSV data source, passing through substages for preparation, prediction computation, performance assessment, visual output, and finally, data extraction, thus forming a complete forecasting workflow.

The framework based on ONNX's modular construction is displayed in Figure 1 which consists of 6 pipeline components with a single central controller that coordinates all of them. Each of the components receive Data sequentially moving through the pipeline. The six pipeline components are as follows:

- The data loading module (*CsvTimeSeriesLoader*) reads the target variable defined in the user's CSV file within the column the user indicated. It validates that the numerical portion is complete and outputs a clean array of time-series data.
- The preprocessing module (*SlidingWindowService*) converts the array into (N-W) input-target pairs using a sliding window of N samples (e.g., W= 24), resulting in a feature matrix [N-W,W] and a target vector whose values are ready for inference.
- The model inference module (*OnnxForecastingModel*) applies mean-scale normalization to the input window, converts it into a tensor of NDArray type, and uses the DJL API to execute the ONNX model and return the predictions to the original scale.
- The evaluation module (*EvaluationService*) computes five additional metrics - Mean Absolute Error, Root Mean Squared Error, Mean Absolute Percentage Error, Symmetric Absolute Percentage Error and R-Squared for comparison to each actual value.

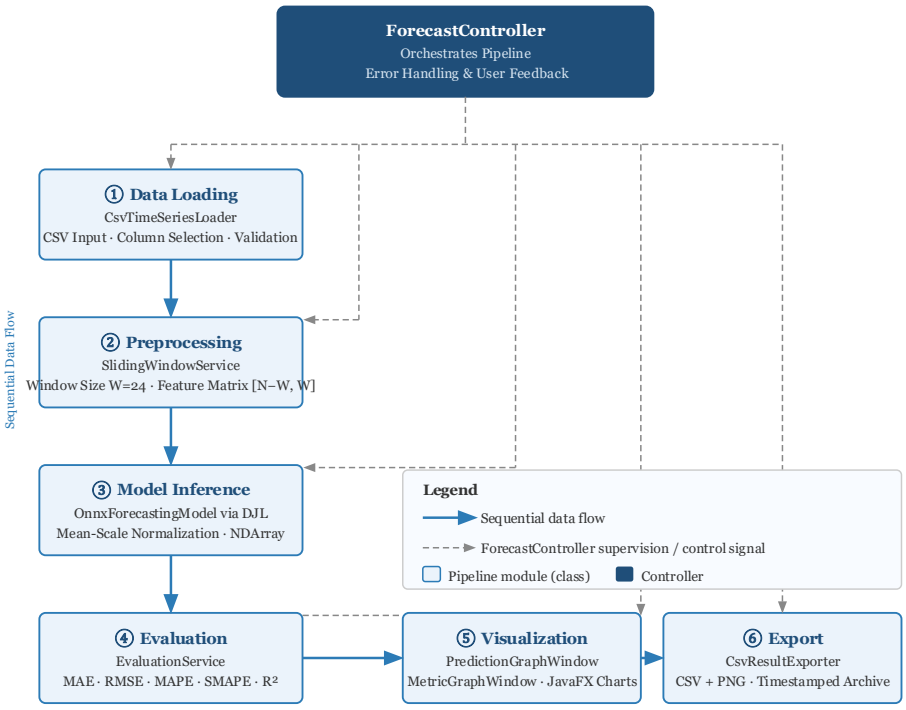


Figure 1. Modular architecture of the ONNX-based time-series forecasting framework

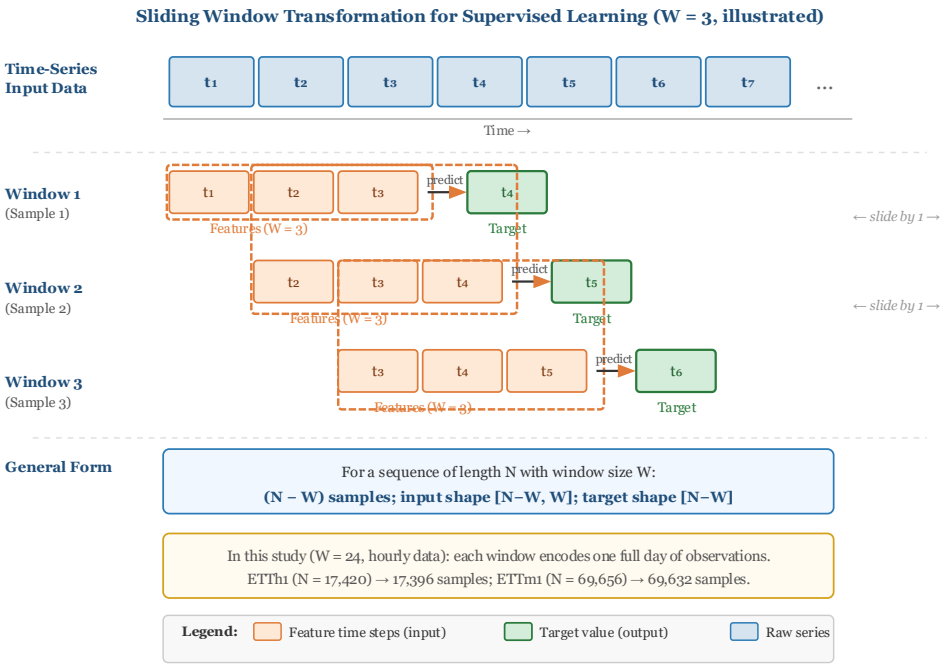


Figure 2. Sliding window transformation for supervised learning format

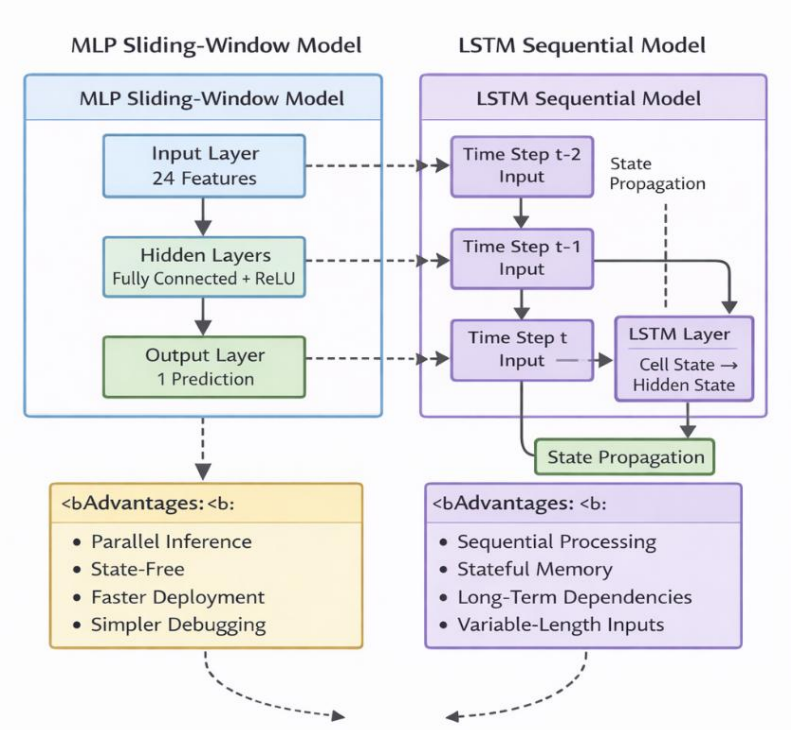


Figure 3. Architectural comparison between MLP sliding window and LSTM models

Figure 2 shows the illustration of sliding window approach with window size W=3. Each window uses W consecutive historical observations as features to predict the next time step (target). For hourly data with W=24, this captures daily periodic patterns essential for time-series forecasting. The modular design approach of the framework also applies to how each of its six components handles the same concerns independently of each other, thus enabling the framework to be maintained, extended, and reused easily at various points in the forecasting process. Figure 3 shows the complete structure of the system and the flow of information between each of its components.

As illustrated in Figure 2, sliding window transformation is the main pre-processing step in preparing raw time series data for use in both model architectures. *The SlidingWindowService* produces (N - W) non-overlapping input-target pairs by moving the sliding window across the time series of length N by one sample at a time. Each input is a W-sized array of consecutive observations used as feature inputs to the model, while the target is the corresponding observation immediately following the window. The result creates an input feature matrix of shape [N - W, W] and a target vector of length N - W, which are both prepped for supervised learning. W is chosen as 24 for both datasets so that a 24-hour time window of hourly observations (ETTh1) captures a full day's worth of observations and thus the insolubility of diurnal patterns in electricity consumption. W is also 24 for 15-minute data sets (ETTm1), producing a 6-hour time window with prior sub-daily variation, but with a finer resolution than the hourly data set at W = 24. The transformation can be computed using $O(n)$ in terms of runtime and memory usage. In the case of ETTm1, N = 69,656 and therefore results in 69,632 samples, meaning that, if there is an ETTm1 of size N, there is a possibility of producing 69,632 total samples after just one iteration through all available data. By only needing one iteration through N, this represents an efficient approach for designing a dataset, like the one described in this research framework, to reduce time and memory deployment when working with distinguishable amounts of data.

Windows are defined as fixed-size samples of the raw input sequence and provide a means for connecting Figure 2 to Figure 3. Both figures represent different representations of this same sample. In Figure 2, the raw input sequence has been restructured as fixed-sized windows (a maximum of W in size) and become fixed-feature vectors for the MLP. However, for the LSTM, the LSTM receives W size samples sequentially in time and uses each window to process the entire input sequence with respect to the previous input, maintaining a cumulative amount of hidden state from the beginning to the end of processing. Therefore, although the window step is architecturally neutral and the same amount of processing occurs across both architectures, the architectural differences between the MLP and LSTM mean that the two architectures use the window step quite differently with respect to the actual temporal context in the raw input sequence. For example, each window through an MLP defines a discrete bounding limit of the entire temporal context for that input sample, while the LSTM model uses the order of samples to determine all previous and subsequent temporal context based on the entire sample.

Based on Figure 3, there is structural differences between MLP (feedforward, window-based) and LSTM (recurrent, sequential) architectures. MLP processes fixed-size windows independently, enabling parallel inference, while LSTM maintains hidden state across time steps to capture long-term dependencies. Architecturally, MLP and LSTM architectures differ because MLP is a feedforward architecture, which uses fixed windows to process its input [6], [11], whereas LSTM is a recurrent neural network architecture that could use sequential data [8], [25]. In addition, while MLP utilizes discrete, fixed-size segments of input data that have no memory of prior segments and therefore can be computed at once, LSTM maintains an internal state that changes over time based on the input to the model and thus can represent long-range dependencies between multiple time points [1], [8].

The part of the framework that deals with data loading offers the *CsvTimeSeriesLoader* class as one way of getting time-series data in a flexible manner, and the input can be buffered from many columns. The user can indicate which column(s) contain the variable(s) of interest, and the component checks the data qualitatively to the extent that it claims to provide the temporal sequences as domain objects. The pre-processing component uses the *SlidingWindowService* to convert time-series data into a form suitable for supervised learning. The sliding window technique is used by this service to derive from a sequence of length N and window size W forming (N-W) examples where each example consists of W recent historical values as features and the next value as the target. The dimensionality of the feature matrix is [N-W, W] whereas the target vector has length N-W, both are ready for supervised learning.

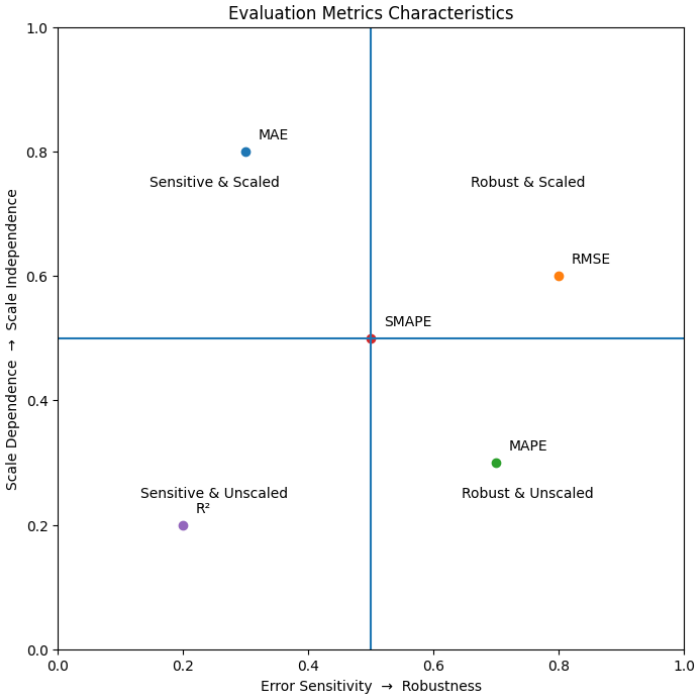


Figure 4. Complementary evaluation metric for comprehensive assessment

The evaluation of the five models shown in Figure 4 baselines the five metrics of MAE, RMSE, MAPE, SMAPE, and R^2 that guide in determining the performance of the proposed model on various data types. The inference of the model is carried out through the class *OnnxForecastingModel*, which enables the deployment of the ONNX model using the DJL. Mean-scale normalization is the method used for data normalization, where the input parameters in the CSV files refer to $x\_scaled = (x\text{-}mean)/scale$. The *NDManager*, the tensor operation manager, is used and a Predictor with *NoopTranslator* is employed for direct inference. The *EvaluationService* measures the five metrics which are: MAE indicating average deviation, RMSE informing penalty-weighted evaluation, MAPE with special handling for near-zero values, SMAPE for zero-heavy distributions, and R^2 for variance explained. MAPE does not consider instances below $\varepsilon = 1e\text{-}10$ where ε is a small numerical tolerance introduced to avoid division-by-zero errors. The implementation system uses this threshold because it is developed as a custom solution instead of adopting standard limits. On the other hand, SMAPE considers both actual and predicted values. JavaFX components are responsible for the visualization; *PredictionGraphWindow* displays a dual-line chart of the actual and the predicted values while the *MetricGraphWindow* shows the evaluation metrics as bars for quick performance assessment in a bar chart manner. Both users are permitted to export high-resolution images with customizable scale factors, where the 2× default is appropriate for publication. The outcomes are collected by the *CsvResultExporter* which generates CSV files for raw predictions and evaluation metrics, time-stamped accordingly. The *ForecastController* module also takes care of data flow and error management, letting users indicate external information sources with a single enumeration-based parameter that is quite versatile and allows easy switching between sources. The class further takes care of the output folder organization, visual and exporting timelines signalling, user feedback via JavaFX dialogs, and the implementation of extensive error handling with thorough diagnostics. This setup facilitates both interactive research usage and integration.

3.2.1 Model architecture

LSTM (long short-term memory) networks and MLP (multilayer perceptron) networks were chosen for this study because of their opposing modes of temporal modeling and the potential of each type of network for deployment in production environments. The primary reason for selecting LSTM as an established recurrent architecture is that it has demonstrated its ability to model long-range dependencies between sequences of data, which formed the basis for the previous research framework [4]. An MLP was selected as a simpler and more deployment-friendly alternative to LSTMs, since using an MLP with a sliding window eliminates the need for a sequential processing mechanism inherent in recurrent architectures. Transformer/attention architectures were excluded from this study due to their high computational complexity and complicated deployment processes when compared to the focus of this study on lightweight ONNX-deployable architectures using the Java programming language. GRU architectures were not included and examined as part of this study even though their computational complexity is very similar to LSTM architectures. This will allow the investigation to focus on contrasting recurrent versus feedforward architectures.

3.2.1 LSTM model

The LSTM model architecture from our previous work [4] consists of multiple stacked LSTM layers followed by fully connected layers. The model processes sequential input directly without windowing, leveraging LSTM's inherent ability to maintain hidden state across time steps. The architecture captures long-term dependencies through cell state propagation and gating mechanisms (input, forget, and output gates). Model training utilized PyTorch framework with Adam optimizer and mean squared error loss, followed by export to ONNX format for deployment.

3.2.2 MLP sliding-window model

The MLP model employs a feedforward architecture processing fixed-size input window. The network consists of:

i. Input layer: Accepts W features representing W consecutive historical time steps
ii. Hidden layers: Multiple fully connected layers with ReLU activation functions
iii. Output layer: Single neuron producing the next time step prediction

The MLP model does not have recurrent connections like LSTM; thus, it treats each window as an independent sample. While an MLP has no internal memory to represent temporal relationships, the sliding window input establishes temporal dependencies implicitly. An MLP will receive W consecutive time steps as a single input to the model from which it will learn temporal relationships. The model therefore learns patterns over adjacent time steps during training (for example, diurnal cycles using W=24); however, the model does not carry a hidden state through each of the windows like an LSTM would do. The use of a sliding window is an effective method for modelling regularly sampled data where short-range dependencies remain within a single window, but it could miss long-range-cross-window dependencies that an LSTM could capture. The MLP model also has several advantages due to this structure: (i) MLP can perform inference on all windows in parallel; (ii) MLP has a much simpler structure, containing fewer parameters; (iii) the MLP model converges to a training solution faster than LSTM; and (iv) MLP can be deployed without managing state. The sliding window with size W=24 captures daily hour-by-hour patterning - this occurs across most forecasting domains. When standardising input using mean and scale values from the appropriate dataset, the numerical results stay numerically stable as well as improving the speed at which it will converge during training.

3.3 Datasets

The test showed that the assessments of three data sets depend on the frequency of sampling affected, and not only how accurate forecasts made by one of the models would be but also how the differences in behaviour between data sources could be adjusted to produce equivalent models from the same data. In order to assist in comparing other models operating under the same conditions but producing forecasts on different datasets, ETTh2 (**Electricity Transformer Temperature - Hourly, Station 2**) was selected as a reference dataset for the investigation due to the fact it uses the same method of modelling as ETTh1 and has the same variable (oil temperature, OT) and is compiled from a different transformer/station. Data for ETTh2 spans over nearly two years ranging from July 2016 to June 2018 which had a total number of observations equal to 17,420 (one for each hour). Comparing and contrasting the two different datasets (ETTh1 and ETTm1) allow for an objective analysis of how well the derived forecasts from ETTh1 and ETTm1 promote or detract from the accuracy of the derived forecasts from all three datasets, and whether variations in temporal resolution between datasets will impact the size or number of windows required to generate an accurate prediction (forecast). The change in temporal resolution from hours to 15-minute intervals will provide validation for using this method of evaluating the forecasting accuracy across a range of different industrial time series with typical monitoring intervals. Table 1 summarises some of the main characteristics of each dataset and relay the nature of how temporal resolution will affect the required width/number of windows to generate accurate forecasting results. Additionally, all three datasets use oil temperature as the target variable and therefore can all be compared to each other in a fair and neutral manner, and by doing this, establish the temporal dimension of the experiments as the primary independent variable for the experiment.

Table 1: Key characteristics of each dataset

Dataset	Domain	Samples	Frequency	Target Variable	Column Index
ETTh2	Electricity	17,420	Hourly	Oil Temperature	2
ETTh1	Electricity	17,420	Hourly	Oil Temperature	7
ETTm1	Electricity	69,680	Hourly	Oil Temperature	7

i. The dataset ETTh1 (**Electric Transformer Temperature – Hourly**) includes about 17000 hourly entries, True Time Series (IRS) of the Electricity Transformer Temperature between July 2016 and July 2018 and is one of the largest datasets in its domain. The ETTh1 dataset consists of 7 load variables; High Useful (HUFL), High Useless (HULL), Mid-Useful (MUFL), Mid-Useless (MULL), Low Useful (LUFL), Low Useless (LULL) and Oil Temperature (OT). The Oil Temperature column is used as a target value while predicting Oil Temperature over time using a computer model; therefore, ETTh1 provides an established benchmark against existing models and demonstrates typical repetitive patterns of use with electricity.

ii. The ETTm1 is a same class electrical transformer dataset consisted of multiple observations with a much greater amount of temporal detail than ETTh1 (about four times more often than ETTh1). In addition to this flight window, the ETTm1 data contains the same features and oil temperature targets as the ETTh1 data; however, with an increased amount of time per record (15 minutes) in the ETTm1 dataset, this allows for much more precision in sampling frequency while sacrificing depth of historical records (just six hours' worth) to achieve this. ETTm1 captures only six hours of the historical context for producing the next 24 timesteps of data compared to the 24 hours' worth of historical context that ETT1 captures. Therefore, this type of configuration is representative of modern industrial IoT-enabled managed infrastructures where sensors are set at significantly shorter reporting frequencies than the hour-long intervals in conventional systems.

3.4 Evaluation metrics

This study uses five complementary measures to assess forecasting performance, ensuring that model quality is viewed from multiple perspectives.

i. **Mean Absolute Error (MAE):** It is the average of the absolute differences between predicted and observed values. MAE provides an easily understood indication of how far predictions can be wrong in the same units as those of the target variable [20]. MAE treats all errors equally regardless of direction; thus, it is interpretable and robust to data scale [26].

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{1}$$

ii. **Root Mean Square Error (RMSE):** It extends assessment of MAE by calculating the square root of average squared differences hence penalizing larger errors more heavily than MAE due to squaring operations. RMSE has the same units as the target variable and is particularly sensitive to outliers [20]; therefore, it becomes useful in identifying models that occasionally produce large deviations [11], [17].

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2}$$

iii. **MAPE (Mean Absolute Percentage Error)**: It is a percentage-based error metric relative to actual input values. In the context of comparing two disparate sets of data, it offers a means for comparing values using a scale-independent benchmark (relative to the total). In the current research study, we overcame the most significant limitation of MAPE when calculating the error using zero and near-zero cases by utilising a threshold, ε = 1e-10; thus, we did not include any sample points with actual values below that threshold. This approach to thresholding is especially critical for any dataset that contains a significant number of zeros [20].

$$\mathrm{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\% \tag{3}$$

iv. **SMAPE (Symmetric Mean Absolute Percentage Error)**: It resolves additional problems with MAPE by including the sum of the absolute actual values and predicted values in the denominator. By doing this, SMAPE provides a more equal treatment of underestimates and overestimates in addition to being better able to accommodate zero values [20]. Through the symmetric formulation of SMAPE, the penalisation of errors is proportional regardless of whether the prediction is greater or less than the actual value.

$$\text{SMAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2} \times 100\% \tag{4}$$

v. **Coefficient of Determination (R^2)**: It measures how much the actual values vary in relation to the predicted ones, where values approaching one (1) are indicative of a better fit. R^2 is also a better measure of accuracy than traditional error metrics, as it is a standard and scale-independent measure that allows for comparisons across studies. It is determined by subtracting one (1) from the ratio of the residual sum of squares to the total sum of squares, which entails comparing the model fit to that of a simple mean value prediction. The coupling of R^2 with metrics such as MAE and RMSE, which evaluate absolute error magnitude in different ways, and MAPE and SMAPE, which deal with model accuracy by handling zeros in a unique way, leads to the emergence of a comprehensive approach to performance evaluation. These metrics, in concert, deter over-optimization based on a single criterion, thus providing a well-rounded evaluation of a forecasting model's performance and error attributes, consequently leading to a rigorously assessed model quality [27].

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{5}$$

$$SS_{\text{res}} = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{6}$$

$$SS_{\text{tot}} = \sum_{i=1}^{N}(y_i - \bar{y})^2 \tag{7}$$

Where SS_{res} represents the squared sum of the residual error, and SS_{tot} defines the squared sum of the observed data.

3.5 Implementation details

- Programming Environment: The framework is implemented in Java 11, leveraging JavaFX 23.0.1 for graphical interface components. This choice enables cross-platform deployment and integration with enterprise systems.
- Deep Learning Inference: This study utilises Deep Java Library (DJL) for ONNX model inference. DJL provides an abstraction layer over multiple backend engines, with our implementation using ONNX Runtime for execution. Models are loaded through DJL's Model API with *NoopTranslator* for direct tensor operations.
- Data Processing: CSV parsing employs Java's *BufferedReader* with custom parsing logic for flexible column selection and header handling. The sliding window generation utilizes standard Java arrays for efficiency in memory-intensive operations.
- Visualization: Chart generation uses JavaFX's *LineChart* and *BarChart* components with *NumberAxis* and *CategoryAxis* configurations. High-resolution image export employs SnapshotParameters with configurable *Transform.scale*() operations, producing PNG files through *SwingFXUtils* and *ImageIO*.
- File Management: The framework implements timestamp-based naming using *LocalDateTime* with *DateTimeFormatter* pattern "yyyyMMdd_HHmmss". Output organization follows a structured directory layout with separate folders for CSV results and graph images.
- Dataset Configuration: An enumeration-based system enables dataset switching through a single constant modification. Each dataset configuration encapsulates file path, column index, and display name, promoting maintainability and reducing configuration errors.

- Error Handling: Comprehensive exception handling with user-friendly error dialogs ensures robust operation. The framework validates file existence before processing and provides detailed error messages including file paths and troubleshooting suggestions.

4.0 EXPERIMENTAL SETUP

This section describes the experimental design which evaluates the MLP sliding-window model through three electricity datasets that use different time intervals. Section 4.1 details the data preprocessing and transformation procedures, including the sliding window configuration and normalization strategy. Section 4.2 describes the model training and ONNX export process. Section 4.3 describes the process for conducting inference and performing evaluation tests. the hardware and software setup used in all experiments is described in Section 4.4, while Section 4.5 establishes the standard protocol used for cross-dataset analysis. All experiments use the same preprocessing procedures, evaluation metrics, and inference environment to ensure fair and reproducible comparisons.

4.1 Data preprocessing and transformation

All datasets undergo consistent pre-processing to ensure fair comparison and optimal model performance. The complete time-series is first loaded from CSV files using appropriate column indices for target variables, followed by transformation into supervised learning format through the sliding window approach. This study employs a fixed window size of W=24-time steps, selected to capture daily patterns in hourly data or approximately six-hour patterns in 15-minute interval data. For a time-series of length N, this process generates (N-24) samples where each consists of 24 consecutive historical observations as input features and the immediate next value as the prediction target.

The MLP model requires input normalization for numerical stability and convergence, achieved through mean-scale standardization where each feature is transformed as $x\_normalized = (x-\mu)/\sigma$. The normalization parameters μ (mean) and σ (standard deviation) are computed from the training set and saved as separate CSV files during model training. During inference, these pre-computed parameters are loaded and applied consistently to maintain the same feature distribution the model encountered during training, ensuring predictions are made on properly scaled data without requiring access to the original training set. For model training conducted in PyTorch, the study employs temporal splitting where earlier observations form the training set and later observations constitute the test set, preserving sequential ordering to simulate real-world deployment where models predict future values based on historical data. Every one of the datasets underwent an identical 80-20 temporal split. For both ETTh2 (original) and ETTh1, there are a total of 13,936 as well as 3,484 test samples, respectively. For ETTm1, there are 55,744 as well as 13,936 test samples. Each of these datasets have the early observations being part of the training set, with only the later observations included within the testing set. This preserves the order in which they are recorded, essentially allowing the models to be tested the same way that they would be when being used for the purpose of predicting future results.

4.2 Model training and export

The LSTM model training from our previous work utilised PyTorch framework with Adam optimizer (learning rate 0.001), batch size of 32, and 100 epochs with early stopping based on validation loss. Mean Squared Error serves as the loss function, and the trained model is exported to ONNX format using PyTorch's *torch.onnx.export* functionality, preserving the complete computational graph for inference. The MLP sliding-window model follows similar training protocols with Adam optimizer, initial learning rate 0.001 with scheduling, batch size 64, and 150 epochs with early stopping. Upon convergence, both the model architecture and normalization parameters computed from training data are exported—the former as ONNX format for framework-agnostic deployment, and the latter as CSV files essential for proper inference-time pre-processing. This ONNX export process captures model architecture, learned weights, and computational operations in a standardized intermediate representation executable by any ONNX-compatible runtime, including DJL's ONNX backend used in the applied proprietary Java framework.

4.3 Inference pipeline and evaluation

Figure 5 relays the systematic workflow from data loading through model inference to result export. The pipeline includes data validation, sliding window transformation, ONNX model execution via DJL, five-metric computation, visualization generation, and timestamped archival of results. The inference pipeline executes a systematic five-stage process for each dataset evaluation.

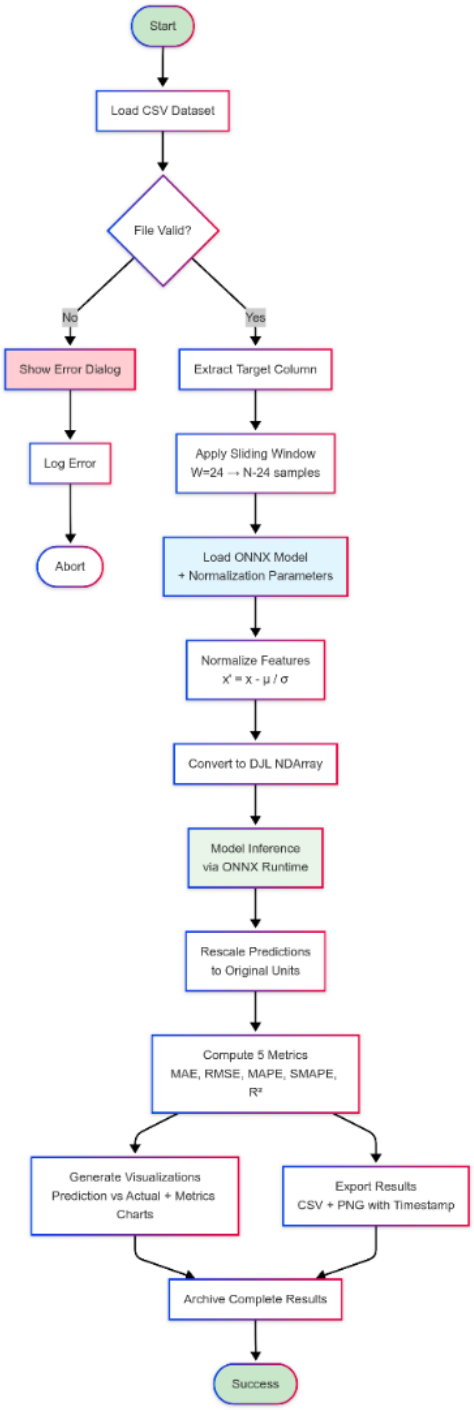


Figure 5. End-to-end inference pipeline for ONNX model evaluation

First, the framework loads the specified CSV file and extracts the target variable column based on configured indices, with validation ensuring no missing values and appropriate numeric formatting. Second, the *SlidingWindowService* transforms the loaded time-series into input-output pairs using 24-step windows, producing an input matrix [*N*-24, 24] and target vector [*N*-24]. Third, the *OnnxForecastingModel* loads the ONNX model file and normalization parameters, then for each input window applies mean-scale normalization, converts normalized data to DJL NDArray tensors, executes model inference through ONNX Runtime, and collects predictions in original scale. Fourth, predicted and actual values are passed to *EvaluationService* for computation of all five metrics, with MAPE calculation automatically skipping samples where actual values fall below threshold ε = 1e-10. Finally, results are visualized through JavaFX windows showing prediction time-series and metric bar charts, while simultaneously exporting CSV files containing index, actual values, predictions, and metrics alongside high-resolution PNG images of both visualizations, all with timestamps for archival and comparison.

4.4 Experimental configuration and reproducibility

Experiments are carried out using an Intel Core i7-13700HX CPU @ 2.1GHz with 16GB RAM in CPU-only mode for deployment without any specialized hardware. The software environment is made up of Windows 11, OpenJDK 11, JavaFX 23.0.1, DJL 0.21.0, ONNX Runtime 1.14.0 through DJL, and Eclipse IDE 2025-03. The datasets evaluated are the original dataset (src/data/dataset.csv), ETTh1 (src/data/ETTh1.csv), and ETTm1 (src/data/ETTm1.csv), all having a fixed window size of W=24. Measures guaranteeing reproducibility are taken which includes deterministic ONNX inference and fixed normalization parameters. Timestamped archiving and detailed logging are the means through which key metrics are tracked for every run. The performance metrics collected consist of MAE, RMSE, MAPE, SMAPE, and R^2, alongside visualizations and CSV exports of the prediction errors. To conduct a fair comparison between the LSTM and MLP models, the same evaluation conditions are applied consistently, including the same test sets, pre-processing procedures, and evaluation metrics, thus enabling a meaningful analysis of the model architecture and temporal scale effects in the datasets.

4.5 Comparative analysis protocol

To enable fair comparison between LSTM and MLP models, we maintain consistent evaluation conditions:

i. **Same Data**: Both models are evaluated on identical test sets from each dataset
ii. **Same Pre-processing**: Windowing and normalization follow the same procedures
iii. **Same Metrics**: All five-evaluation metrics are computed identically for both architectures
iv. **Same Inference Environment**: DJL/ONNX Runtime provides consistent execution for both model types.

This controlled setup isolates model architecture as the primary variable, enabling meaningful performance comparisons and generalization analysis across domains.

5.0 RESULTS AND ANALYSIS

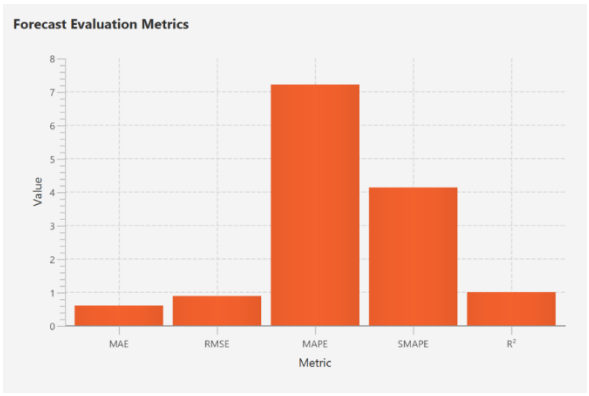
5.1 Quantitative performance results

This study conducts comprehensive quantitative evaluation of the MLP sliding-window model across three datasets representing the same domain (electricity systems) at different temporal resolutions. Table 2 summarizes the forecasting performance using all five-evaluation metrics. Results demonstrate excellent forecasting performance across all datasets, with accuracy improving at higher temporal frequencies. R^2 values exceed 0.98 for all datasets; specifically, ETTm1 has $R^2 = 0.9969$, signifying 99.69% variance explained. ETTm1 metrics include MAE = 0.3237, RMSE = 0.4770, MAPE = 3.85%, and SMAPE = 2.54%. Original and ETTh1 datasets yield nearly identical results, underscoring model generalization across electricity systems.

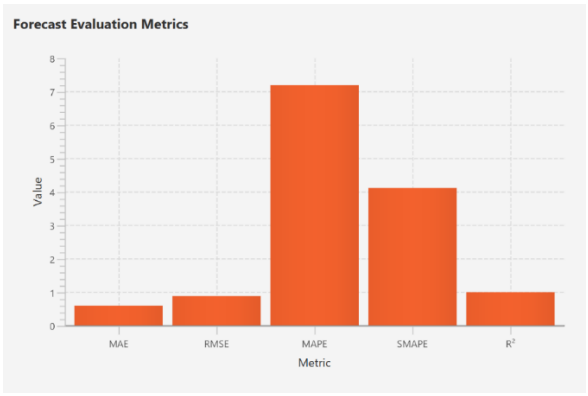
The characteristics of the dataset are addressed in Figure 8, and they clearly indicate a focus on the ETTh1, ETTm1 (electricity transformer temperature), and the original datasets. Variations in sample size, frequency, and domain characteristics make it easier for models to generalize for the purpose of forecasting. The change from hourly to 15-minute intervals is noticed by the dramatic drop in MAE of 45% and MAPE of 46% that bode well for short-term forecasting. The hourly datasets show a roughly constant MAE of about 0.59, while the ETTm1 dataset clearly demonstrates the 0.3237 MAE, which is decreased MAE due to better time resolution. The RMSE measures are very similar to MAE, in this case-almost 0.88 for hourly data. MAPE signifies strong accuracy, especially in hourly datasets with 7.2%, and ETTm1 with 3.85%. SMAPE values confirm this, with hourly data at about 4.1% and ETTm1 at 2.54%. R^2 values show very good model fitting, with hourly datasets reaching $R^2 = 0.9895$. The analysis of the performance concluded that the MLP sliding window forecasting method is optimal for hourly data. ETTm1's increased sampling frequency leads to better prediction synchronization, which not only strengthens accuracy across datasets but also points to the same error distributions; thus, confirming the model's ability to deal with different temporal resolutions.

Table 2: Performance across Datasets at Different Temporal Scales

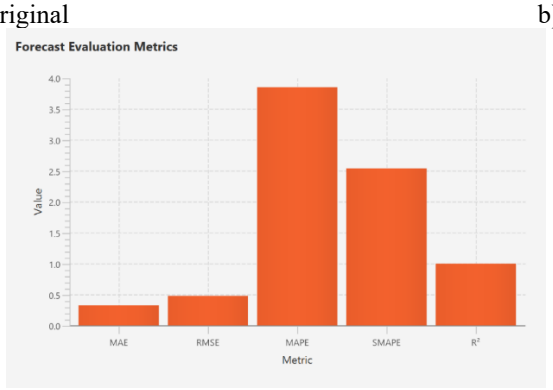
Dataset	Frequency	Samples	MAE	RMSE	MAPE (%)	SMAPE (%)	R^2
Original	Hourly	17,396	0.5921	0.8783	7.20	4.12	0.9895
ETTh1	Hourly	17,396	0.8764	7.19	4.11	4.11	0.9895
ETTm1	15-minute	69,680	0.3237	0.4770	3.85	2.54	0.9969



a) Original

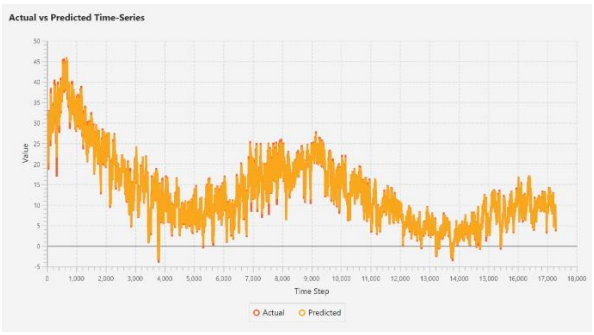


b) ETTH1

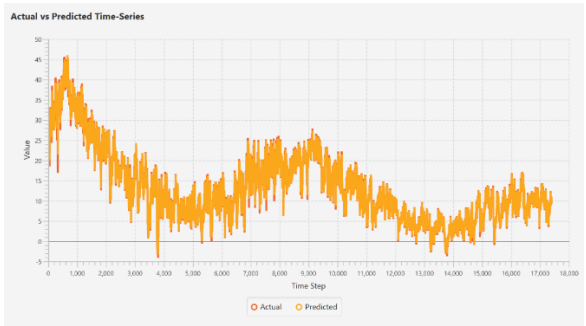


c) ETTM1

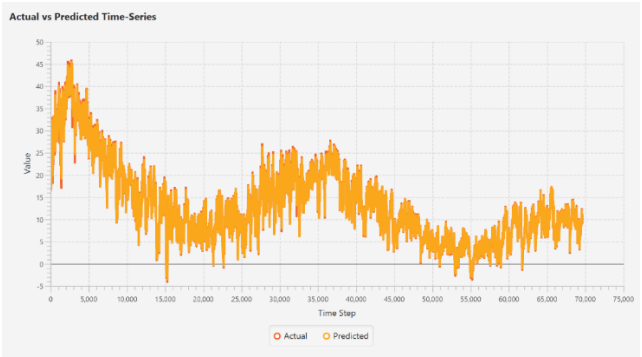
Figure 6. Result of the forecast evaluation metrics for the three compared datasets



a) Original



b) ETTH1



c) ETTM1

Figure 7. Comparative output between actual versus predicted time-series data for the 3 datasets

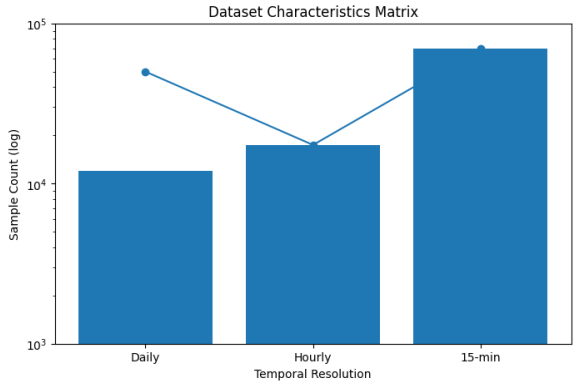


Figure 8. Characteristics of evaluation datasets across domains and temporal resolutions

6.0 DISCUSSION

The framework extensions provided thorough evaluations over various temporal scales, employing five metrics (MAE, RMSE, MAPE, SMAPE, R^2) that ensure accurate performance representation without criterion optimization bias. SMAPE reflected about 57% of MAPE, showing no systematic bias. Utilizing a timestamp-based archiving enabled effective tracking of performance advancements across three datasets, particularly with increased frequency. High-resolution visualizations at 2× scaling enhanced graph quality for effective comparisons. An enumeration-based dataset configuration allowed for quick adjustments and minimized errors essential for evaluating temporal scale variations. MLPs achieved excellent performance, with R^2 reaching 0.9969, demonstrating that simple architectures can excel in time-series applications. They provided near-optimal accuracy and facilitated complete parallelization during inference, unlike sequential LSTMs. The analysis showed that using 15-minute data over hourly data significantly reduced MAE by 45% and improved MAPE by 46%, emphasizing the importance of current high-resolution data. MLP sliding-window models effectively forecasted one-step-ahead in regularly sampled data, while LSTMs catered to variable-length sequences with long-term dependencies. The findings illustrated better results from higher temporal proximity within a 24-timestep window. Key metrics indicated stable error characteristics across sampling frequencies, with ETTm1 achieving MAE = 0.32, RMSE = 0.48, MAPE = 3.85%, SMAPE = 2.54%, and R^2 = 0.9969. The Java implementation showed scalability evaluating 69,680 ETTm1 samples, underscoring the importance of high-resolution data for enhanced forecasting accuracy and calling for further research into diverse datasets. There is an important limitation of this study, namely direct benchmarking of LSTM applied to the electricity datasets evaluated in this analysis has not been accomplished. The LSTM used in the previous framework was trained on multivariate Malaysian rain data; therefore, it is not architecturally compatible with the univariate oil temperature prediction task performed for this study. As such, retraining a compatible LSTM on the ETT datasets is a priority for future work; this will enable the originally intended direct comparison of MLP versus LSTM.

7.0 CONCLUSION AND FUTURE WORK

This article presents a time-series forecasting framework based on Java and ONNX that allows for the comparison of various models and datasets. The work foundation builds upon an earlier LSTM architecture and makes use of the sliding window technique for the generation of multi-layer perceptron (MLP) models for the purpose of rainfall forecasting. The MLP makes good use of the advantages of well-spaced data, attaining high R^2 values and low errors, but they could not perform well at all with uneven, zero-inflated datasets. An evaluation module for performance measurement and a timestamping system for keeping track of results are two of the features of the framework that come along with its capabilities. The results emphasize very strongly that the choice of processing and the way of representing the data are the two factors that can either make or break a model, but at the same time they also bring to light the situation where it is very difficult to migrate a machine learning prototype to operational use. A few limitations are mentioned, such as the limited scope of the dataset and the absence of multi-step forecasting techniques. The authors urge the research community to carry out more empirical evaluations and to collaborate to develop the open-source solutions that are needed for the complex forecasting scenarios across different industries.

ACKNOWLEDGEMENT

The authors gratefully acknowledge Universiti Pendidikan Sultan Idris which provided support for this research and subsequent publications. The article received positive feedback from peers who reviewed it and their assessment of the work was acknowledged with gratitude. The research received no financial backing from any funding organization which included public and private and not-for-profit organizations. The authors will release

the source code for the Java-based forecasting toolkit after the official publication of the article which will be accessible through GitHub account at https://github.com/blackcontractor90 to support reproducible research and future development work.

AUTHORS CONTRIBUTION

Farid Morsidi (Conceptualisation; Methodology; Validation; Formal analysis; Data curation; Formal analysis; Investigation; Resources; Software; Visualisation; Writing - original draft; Writing - review & editing), Asma Hanee Ariffin (Conceptualisation; Methodology; Validation), Rohaizah Abdul Wahid (Conceptualisation; Methodology; Validation; Resources).

DECLARATION OF COMPETING OF INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] Y. Wang, P. Jia, Z. Shu, K. Liu, and A. Rashid, "Multidimensional precipitation index prediction based on C NN -LSTM hybrid framework," 2025, doi: https://doi.org/10.48550/arXiv.2504.20442.

[2] M. Asif, M. M. Kuglitsch, I. Pelivan, and R. Albano, "Review and Intercomparison of Machine Learning Applications for Short-term Flood Forecasting," *Water Resour. Manag.*, pp. 1971–1991, 2025, doi: 10.1007/s11269-025-04093-x.

[3] A. Poghosyan *et al.*, "An enterprise time series forecasting system for cloud applications using transfer learning," *Sensors*, vol. 21, no. 5, pp. 1–28, 2021, doi: 10.3390/s21051590.

[4] F. Morsidi, "A Modular Java-Based Framework for Deploying ONNX Time-Series Forecasting Models : A Rainfall Prediction Case Study," *J. Appl. Eng. Des. Simul.*, vol. 5, no. 2, pp. 90–104, 2025, doi: 10.24191/jaeds.v5i2.144.

[5] S. D. Latif *et al.*, "Assessing rainfall prediction models: Exploring the advantages of machine learning and remote sensing approaches," *Alexandria Eng. J.*, vol. 82, no. May, pp. 16–25, 2023, doi: 10.1016/j.aej.2023.09.060.

[6] P. Kanchan, "Rainfall Analysis and Forecasting Using Deep Learning Technique," *J. Informatics Electr. Electron. Eng.*, vol. 2, no. 2, pp. 1–11, 2021, doi: 10.54060/jieee/002.02.015.

[7] D. Sun, J. Wu, H. Huang, R. Wang, F. Liang, and H. Xinhua, "Prediction of Short-time rainfall based on deep learning," *Math. Probl. Eng.*, vol. 2021, 2021, doi: 10.1155/2021/6664413.

[8] A. Bhardwaj, "Time Series Forecasting with Recurrent Neural Networks: An In-depth Analysis Time Series Forecasting with Recurrent Neural Networks: An In-depth Analysis and Comparative Study," *EDU J. Int. Aff. Res. (EJIAR), ISS*, vol. 2, no. 4, 2024, [Online]. Available: https://edupublications.com/index.php/ejiar/article/view/36

[9] R. Rosly, M. Man, A. Ngah, and N. S. A. Manan, "Multi-classifier models to improve the accuracy of fish landing application," *Int. J. Adv. Technol. Eng. Explor.*, vol. 11, no. 111, pp. 145–159, 2024, doi: 10.19101/IJATEE.2023.10102060.

[10] M. Koklu and I. A. Ozkan, "Multiclass classification of dry beans using computer vision and machine learning techniques," *Comput. Electron. Agric.*, vol. 174, p. 105507, 2020, doi: https://doi.org/10.1016/j.compag.2020.105507.

[11] D. Endalie, G. Haile, and W. Taye, "Deep learning model for daily rainfall prediction: Case study of Jimma, Ethiopia," *Water Supply*, vol. 22, no. 3, pp. 3448–3461, 2022, doi: 10.2166/WS.2021.391.

[12] R. Taylor, "Machine Learning Techniques for Fish Breeding Decision Making," *2023 Wellingt. Fac. Eng. Symp.*, pp. 1–12, 2023, [Online]. Available: https://ojs.victoria.ac.nz/wfes/article/view/8422

[13] S. M. M. Islam, S. I. Bani, and R. Ghosh, "Content-based Fish Classification Using Combination of Machine Learning Methods," *Int. J. Inf. Technol. Comput. Sci.*, vol. 13, no. 1, pp. 62–68, 2021, doi: 10.5815/ijitcs.2021.01.05.

[14] P. Han, C. Du, J. Chen, and X. Du, "Minimizing Monetary Costs for Deadline Constrained Workflows in Cloud Environments," *IEEE Access*, vol. 8, pp. 25060–25074, 2020, doi: 10.1109/ACCESS.2020.2971351.

[15] M. Bacevicius and A. Paulauskaite-Taraseviciene, "Machine Learning Algorithms for Raw and Unbalanced Intrusion Detection Data in a Multi-Class Classification Problem," *Appl. Sci.*, vol. 13, no. 12, 2023, doi: 10.3390/app13127328.

[16] F. M. Javed Mehedi Shamrat *et al.*, "LungNet22: A Fine-Tuned Model for Multiclass Classification and Prediction of Lung Disease Using X-ray Images," *J. Pers. Med.*, vol. 12, no. 5, 2022, doi: 10.3390/jpm12050680.

[17] R. A. Adewoyin, P. Dueben, P. Watson, Y. He, and R. Dutta, "TRU-NET: A deep learning approach to high resolution prediction of rainfall," *Mach. Learn.*, vol. 110, no. 8, pp. 2035–2062, 2021, doi:

10.1007/s10994-021-06022-6.
[18] I. K. Opara, U. L. Opara, J. A. Okolie, and O. A. Fawole, “Machine Learning Application in Horticulture and Prospects for Predicting Fresh Produce Losses and Waste: A Review,” *Plants*, vol. 13, no. 9, pp. 1–21, 2024, doi: 10.3390/plants13091200.
[19] V. Atashi and H. T. Gorji, “Enhanced flood prediction using LSTM and climate parameters: Multi-station analysis of snowmelt-induced flooding in the Red River of the North,” *J. Hydroinformatics*, vol. 27, no. 2, pp. 245–260, 2025, doi: 10.2166/hydro.2025.236.
[20] A. Fayyazbakhsh, T. Kienberger, and J. Vopava-Wrienz, “Comparative Analysis of Load Profile Forecasting: LSTM, SVR, and Ensemble Approaches for Singular and Cumulative Load Categories,” *Smart Cities*, vol. 8, no. 2, 2025, doi: 10.3390/smartcities8020065.
[21] P. Hess and N. Boers, “Deep Learning for Improving Numerical Weather Prediction of Heavy Rainfall,” *J. Adv. Model. Earth Syst.*, vol. 14, no. 3, pp. 1–11, 2022, doi: 10.1029/2021MS002765.
[22] Deep Java Library (DJL), “ONNX runtime engine for DJL,” 2024.
[23] AWS Machine Learning Blog, “Simplified MLOps with Deep Java Library and ONNX,” Jun. 2023.
[24] Deep Java Library (DJL), “Time series forecasting extension,” 2024.
[25] E. Salcedo, “Graph Learning-based Regional Heavy Rainfall Prediction Using Low-Cost Rain Gauges,” 2024, doi: 10.1109/LA-CCI62337.2024.10814868.
[26] A. E. Yilmaz and H. Demirhan, “Weighted kappa measures for ordinal multi-class classification performance,” *Appl. Soft Comput.*, vol. 134, p. 110020, 2023, doi: 10.1016/j.asoc.2023.110020.
[27] T. Indravattana, “Chula Digital Collections Solving multi-depot open pickup and delivery problem with capacity and distance restrictions by ant colony optimization algorithm Solving Multi - depot Open Pickup and Delivery Problem with Capacity and Distance Restrictions by A,” 2023.