Volume 5 Issue 2 (September 2025)
DOI 10.24191/jaeds.v5i2.144
eISSN 2805-5756

Journal of Applied Engineering Design & Simulation

A Modular Java-Based Framework for Deploying ONNX Time-Series Forecasting Models: A Rainfall Prediction Case Study

Farid Morsidi[1,*]
[1]Computing Department, Faculty of Computing & Meta-Technology, Universiti Pendidikan Sultan Idris, 35900 Tanjong Malim, Perak, Malaysia.
[*]corresponding author: farid.mors90@gmail.com

ABSTRACT

Bringing deep learning models for time-series forecasting into production has its challenges in the real world, and one of these challenges is transitioning from a Python-based development environment to platform-agnostic software. This paper proposed a modular extensible and easy-to-use Java-based framework for implementing deep learning time-series forecasting models, using simulated rainfall forecasting as a specific case study. One of the benefits of the case study of rainfall forecasting was the number of datasets available to work with and its relevance to environmental monitoring. The framework is able to seamlessly implement Long Short-Term Memory (LSTM) models that were trained in PyTorch and exported in the ONNX (Open Neural Network Exchange) format running inference with DJL (Deep Java Library). Users benefit from the ability to convert Java-native data to DJL compatible tensors via a custom Translator module in real time or batch for prediction. The framework also provides an easy-to-use graphical user interface (GUI) built in JavaFX to allow users to import CSV datasets to predict, visualize results, and export outputs without any advanced programming experience. In the rainfall forecasting case imposed for the case study analysis, the predictive accuracy was limited by the dataset; however, the main purpose of the work was to develop a reusable, accessible, and extensible deployment platform for ONNX-based deep-learning time-series models. The framework provides the foundation to allow for practical use of machine-learning workflows in a variety of applications, including environmental management, logistics, and industrial automation. The modularity of the framework and cross-platform development help to fill the gap with existing deployment technologies which offer a scalable pathway of operationalizing machine learning in practice with modern models such as deep learning. The proposed framework provides accessible solution between advanced model development and deployment covering wide use cases of machine learning.

Keywords: Rainfall prediction; Long Short-Term Memory (LSTM); Deep learning; Time-series forecasting; Deep Java Library (DJL)

Abbreviations

LSTM	Long Short-Term Memory
ONNX	Open Neural Network Exchange
csv	Comma-separated value
LSTM	Long Short-Term Memory
DJL	Deep Java Library

1.0 INTRODUCTION

Rainfall prediction remains an enduring challenge in the field of meteorological science because of the dynamism, nonlinearity, and multivariate characteristics associated with atmospheric systems [1], [2]. Weather is changeable and it depends on the interaction of a multitude of variables, including temperature, humidity, wind, and atmospheric pressure, which are complexly related and not easily modelled using rigorous statistical or numerical methods [1]. Traditional methods have proven useful for producing baseline estimates, but these do very poorly in capturing long-term temporal relationships, especially short- to long-range nonlinear dependencies and interactively innovative features, particularly for the shorter to medium forecasting horizons. The need to capture these relationships is critical given the importance of rainfall prediction across an array of high-impact domains which include precision agriculture, flood mitigation, water resource planning, infrastructure resilience, and disaster risk preparedness and mitigation [3], [4], [5]. Reliable precipitation forecasting is critical for mitigation and disaster response management [6]. The accurate and timely prediction of heavy rainfall events is

Received on 22.08.2025
Accepted on 26.09.2025
Published on 03.10.2025

critical for flood risk management and disaster preparedness. Heavy rainfall prediction begins with the ability to observe, examine and assess the rainfall data at any given point. Only then can effective measures be taken to offset any magnitude of climate variations. This provides knowledge to assist in surface and subterranean hydrological resources planning [7].

Of late, deep learning has emerged as a promising alternative for chronicle data modeling. Long Short-Term Memory (LSTM) networks are a form of recurrent neural network and have an ability to learn temporal dynamics from multiple input streams, or multivariate inputs [8]. Their network architecture involves gating mechanisms and internal memory states, which allow the model to learn that the temporal order of past instances can affect the future when applying the LSTM architecture using historical data (for example, as when predicting rainfall). LSTM and other deep models remain attractive where more research is needed, but the gap between the on-going research and actual application of deep learning systems in practice is large [1], [9], particularly for users who are not machine learning or Python developers. Most state-of-the-art forecasting models use Python solidifying or frameworks like PyTorch [10] or TensorFlow, and while these are flexible frameworks, they create problems when it comes to dropping these models into cross-platform, maintainable software stacks. Non-expert users, the users who typically put forecasting models into a service like meteorologists, environmental researchers or public-sector analysts, usually do not have the tools or knowledge to directly deploy and interface these models for real-time applications. Moreover, production-grade applications need an effective means of not only collecting and using accurate models, but a sensible and robust pipeline of integrating incoming data, pre-processing incoming data, executing the models to produce a result, visualizing the result, and integrating this has resulted into a larger system.

This study addresses those limitations through a new focus on developing, implementing, and distributing a modular, user-friendly, extensible forecasting application entirely written in Java. Also, the application is a bridge between high-performance machine learning models and the ability to actually use them in practice, as it unites a PyTorch-trained LSTM model exported in Open Neural Network Exchange format (.onnx) with the Deep Java Library (DJL) for deploying ONNX model in a Java context, providing platform-independent, Python-dependent inference at a relatively efficient speed; all without the typical, traditional Python dependencies and calls. Additionally, the application is designed with both expert and novice end-users in mind by providing a graphical user interface (GUI) built upon JavaFX that allows for easily uploading data in CSV format, real time and batch model inference and visualization (including confidence intervals), and exporting results for offline exploration.

This work does not place emphasis on improving predictive accuracy for the current state and is focused on building a reusable and extensible software framework for the use of machine-learning models inside operational, cross-platform forecasting systems. Rainfall prediction is used as an example, but it is important to note that the system and design are flexible and can be used within any domain, or for any time-series forecasting problem, which could include energy consumption, traffic flow, or financial queries. In regard to the secondary goals of this paper, the contribution, in the form of the layered toolkit, the context, scalability, and operational examples for rainfall forecasting, include: (i) a modular Java package toolkit for the inference of ONNX-based deep learning models based on DJL runtime, (ii) a custom *Translator* pipeline for Java native data structures and DJL objects (tensors) as intermediate data storage for projected pre-batched inference and real-time inference, (iii) an integrated JavaFX-based GUI that allows non-programmers to work with complex time-series models without writing code, and (iv) a demonstration of the toolkit architecture, extension, and usage with rainfall forecasting as an example use case. The address of this gap in tooling and deployment serves to promote advanced time-series modeling methods and make operationally relevant in a wider scientific, industrial, and decision support capacity.

2.0 RELATED WORKS

Conventional time-series forecasting models such as ARIMA, SVR, and MLR often do not perform well when predicting atmospheric data, which frequently contain multiple input variables that vary in time and space, because they are poor at capturing complex inter-variable relationships and long-term dependencies [5], [11], [12], [13]. While LSTM networks and Gated Recurrent Units (GRUs) represent a generational leap forward for accurate prediction problems, only over the past five years have advanced sequence modeling architectures become widely available, so that prediction problems now have “Fast Reasoning” architectures capable of automated feature development from the input data based on temporal dependencies [5], [8]. With LSTMs somewhat uniquely positioned, models are shown to consistently outperform conventional strategies for short- and medium-term rainfall forecasting by being able to model inter-variable interactions and long-term temporal variability. Figure 1 represents the general summation of deep learning applications in collaboration with LSTM features, particularly in this case the application with weather forecasting.

Figure 1. Network diagram illustrating the correlation between deep learning applications and their usability in LSTM models for intelligent decision-making features, focusing on their integration in weather predictive capabilities

Anticipating rain can be difficult but it can utilize data-mining methods to demonstrate how previous weather data can identify the patterns. Hybrid models used historical rainfall observations and nearby raw data streams, from remote monitoring devices to abstract the features. In terms of rainfall prediction investment, machine learning methods can be used regardless of climate or time scale which includes any time interval or CAD for LSTM architectures [8]. Using a Convolutional Neural Network (CNN), a model was created to predict such things as monthly rainfall totals for a location [3]. To assess the performance of the system, the mean square error, mean absolute error, and root mean square error were used. Another research study introduced an LSTM-based model for predicting rainfall in Jimma, Ethiopia, aimed at enhancing sustainable agricultural methods [14]. The proposed model's performance was assessed using RMSE, MAPE, NSE, and R^2 metrics. The LSTM model successfully predicted daily rainfall with greater accuracy compared to ML models such as MLP, KNN, SVM, and DT. This LSTM model may serve as a dependable and beneficial resource for forecasting daily rainfall totals in Jimma, aiding agricultural practices and management.

A research study of hourly rainfall forecasting in western part of Ghana was conducted [1] utilizing data sourced from the European Centre for Medium-Range Weather Forecasts (ECMWF). The study also developed Deep Learning models based on the LSTM algorithm. The researchers performed a correlation analysis to find the various relationships and important parameters/features affecting rainfall. The correlation analysis indicated

that temperature, pressure, and humidity are the most significant contributors to rainfall in the region. The proposed LSTM architecture performed the best with a lower MSE and RMSE than the other configuration models. In the future, the plan is to explore more alternatives of model architecture by increasing the number of training epochs past 200. A study proposed a CNN-LSTM framework for a multidimensional precipitation index forecasting model [11]. The model utilized monthly means from Pune, Maharashtra, which are sourced from world meteorological archival data between the 1970's and 2002. This model efficiently estimated precipitation in line with expectations using local features while utilizing long-term features well. The model achieved a RMSE of 6.752 or 204% better than several traditional time series methods, but the higher computational costs with large datasets limited the model's possibilities. This highlights some areas for future research and improves the possibility for applications in precipitation forecasting.

Forecasting rainfall can be difficult because of the complex, dynamic, and changing nature of it and the effects of climate change. A study compared three modelling approaches for rainfall forecasting: statistical models; machine learning algorithms; and hybrids of models [15]. The neural networks model for deep learning is to develop a model to predict regional precipitation using satellite, radar, and ground data. The deep learning model uses the data mining (DM) and machine learning (ML) techniques [8]. Hybrid models that incorporate combination remote sensing data and machine learning techniques need more research because of the small number of applications which are able to model problem instances accurately. Only a small handful of researchers have been able to predict rainfall accurately [8]. A comparison study has evaluated the Artificial Neural Network (ANN), Simple Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM) techniques [9]. LSTM reported the best performance in accuracy using Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE). The research team concentrated on cold region river systems by using LSTM modelling to assist with the future challenges associated with modelling flood forecasting in changing climate conditions. For the modelling system, hourly USGS gage water level data were used to inform the LSTM model, which resulted in encouraging findings and potential improvements in flood forecasting short-term prediction capabilities and warning systems. However, to move forward, considerable effort must be made in developing longer-term prediction of flood forecasting methods and techniques and determining the accuracy.

The variability and skewedness of rainfall distribution means that modeling rainfall brings its own level of difficulty, since numerical weather prediction (NWP), which requires discretely gridded simulations, adds complication [16]. Furthermore, the resolution of the grid also influences the bias in rainfall estimations. For example, a deep neural network (DNN) implementation had loss function optimization that tends to represent heavy rainfall events effectively [16]. The U-Net based DNN led to improvements in heavy rainfall event prediction that ranged from two times to over six times in water level forecast skill through loss function tuning to learn the tail distributions better. In terms of in-depth analysis of rainfall distribution prediction through LSTM, a research synthesis was conducted where the analysis of 94 articles that used LSTM since 2001 was performed to review short-term forecasting of flood events [17]. The research synthesis suggested that hybrid model use is superior to standalone models and that the combination of model and input is crucial for good operational forecasting accuracy.

Climate models assess climate change's effect on flood risk and extreme precipitation, but they tend to have a lower spatial resolution, leading to less precision [18]. One study employed a Deep Learning (DL) approach in which a reanalysis product was used as input to achieve greater accuracy for precipitation predictions. This particular research demonstrated a special TRU-NET model, employing an encoder-decoder structure, which equally uses 2D cross-attention [18]. The TRU-NET results were shown to have less RMSE and MAE than typical DL models and state-of-the-art dynamical models, while confidence metrics were produced for all seasons and locations. Another study indicated a new and original Conv3D-GRU model for short-term rainfall intensity predictions based on radar, called a Conv3D-GRU, which was able to successfully analyse features from radar echo map image features while enhancing accuracy in their forecasting [5]. A proposal for a low-cost IoT system capable of automatic rainfall recording and monitoring also produced model precision with rainfall predictions, and a new graph neural network methodology was also introduced that had a much higher degree of precision in prediction of regional heavy rainfall by modeling spatial dependencies [7]. Testing recent models on a 72-month dataset, validation indicated that the synthetically produced approach could estimate heavy rainfall with regional applicability in areas of limited resources or where weather observations are sparse.

3.0 RESEARCH FRAMEWORK

The use of LSTM models in the operational production environment is typically limited. While a lot of LSTM models are being implemented in Python, there is usually the need for specialized systems that cannot leverage the entirety of the programming language library. Advances in the formats for exchanging modeling formats, specifically the Open Neural Network Exchange (ONNX) format and flexible runtimes like the Deep Java Library (DJL), have provided new deployment of a pre-trained machine learning or deep learning model almost anywhere it can be connected [19]. The work in this paper takes that gap in the research and the behaviour of implementing deep learning models by implementing a cross-platform implementation of a LSTM-based rainfall forecasting system that leverages the predictive capabilities of deep learning, but is fully deployable across several systems and ecosystems, made possible with Java and ONNX.

3.1 System architecture

The architecture of the cloud-based forecasting toolkit that has been proposed is predicated on modularity, usability, and the cross-platform deployment of deep learning models for time-series forecasting. While LSTM networks have been shown to work well with sequential data like weather variables, the focus of the proposed toolkit lies in the software engineering aspects of fielding models in desktop environments, utilizing a layered abstraction that can promote better maintainability and extensibility. Figures 2 and 3 are both representations of the proposed development workflow methodology for input and output process in relation with data inferencing. The proposed system comprises of four primary layers:

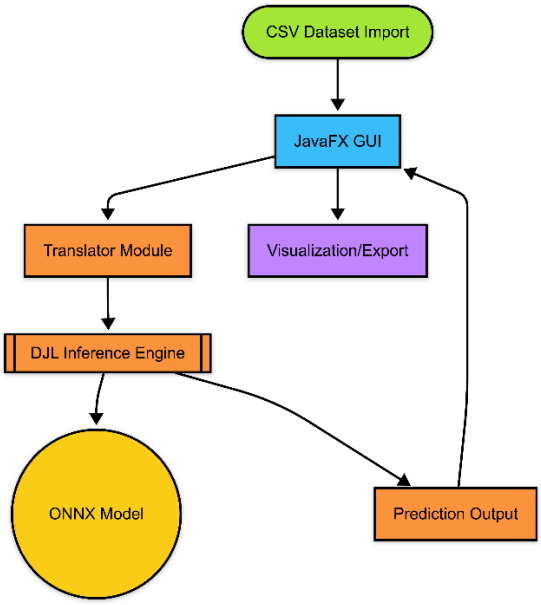

Figure 2. System architecture of the proposed development prototype

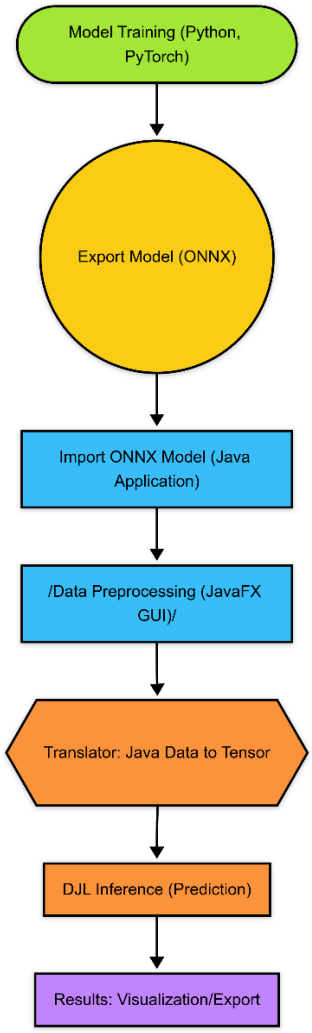

Figure 3. Workflow diagram from training phase to prediction and data instance export for further analysis

i. User interface layer

The user interface was developed using JavaFX, allowing the user to visually interact with the system without needing to parse code. The user can submit comma-separated value (.csv) file which contains time-series data along with a meteorological feature (e.g. humidity threat and temperature). Once uploaded, and depending on the invocation of the models to make forecasts using the model controls that are embedded, the user will receive output and immediate feedback. The user can generate plots of varying dependencies, including line plots or scatter plots, with optional confidence intervals for improved interpretability of forecasted values. As an example, the user may upload rainfall recorded in Malaysia in July 2025 to predict July for the same year using the LSTM model, ultimately producing plots of predicted trends. The potentiated results can be saved to an image file and/or an Excel or CSV spreadsheet system for reporting and/or analysis.

ii. Data handling layer

The data-processing element is the tool that transforms input values into the structure used by the model. The process involves a variety of processing activities:

a) *Validation*: It ensures that the uploaded documents contain all the required fields and the appropriate formatting of the input. If the features are missing or wrongly formatted, the values would either be highlighted for the user to correct or, in the case of with missing data, would be imputed [20].

b) *Normalization*: It normalizes input features either min-max normalization (rescale feature from [0,1]) or z-score normalization (which creates a feature mean of zero and variance of one) depending on which normalization the users have chosen [21], [22]. Normalization is needed so that the different inputs to the neural network have common numerical behaviour.

c) *Creation of Sliding Windows*: The application will take raw sequential data and partition into overlapping windows of the user-specified size [23]. For example, with a fixed window size of 10 and 4 input features (temperature, humidity, wind-speed, and pressure), each sliding window would produce tensors of shape [batch_size, 10 (number of historical time-steps), 4 (number of files)]. Each window can be thought of as capturing a sequence of ten continuous time-steps that the model will use for forecasting a future value (in this case, rainfall at the next time point).

These changes are managed internally with Java data structures, and the result is forwarded to the model integration layer in a format suitable for tensor operations.

iii. Model Integration layer

The model integration layer is the engine behind the system which contains the execution of the ONNX-encoded LSTM model using the Deep Java Library (DJL) inference engine. The model integration layer hides the challenges of deep-learning inference and allows pre-trained models to be used in a Java application. The ONNX model is developed in a contained environment with PyTorch and then incorporated into the application using the model loading API provided within the DJL. The *RainfallBatchTranslator* is a custom Translator class responsible for input and output translations.

- Input translation: Translates a 3D Java float array (1,10,4) into DJL's *NDArray*, the shape of the input must match the time and feature dimensions expected by the model.
- Output translation: Extracts and organizes the models predicted output (typically one float value for estimated rainfall in mm) from the output tensor.

The *RainfallBatchTranslator* allows concurrent translation of multiple sequences that allows batch inference predicting rainfall over different sites or moments in time. A batch of 50 samples, each with 10-time steps and 4 features, are structured as a tensor of shape [50, 10, 4] and this accommodates the processing of samples. This layer can be treated the same as any other model which allows for any ONNX-exported time-series regression model on the market to replace the LSTM, while also enabling other forecasting tasks such as energy consumption or equipment breakdown predictions.

iv. Visualization layer

The last layer deals with output visualization and interpretation. When inference is complete, the results are fed into a visualization module. The visualization module displays the predictions in chart form using JavaFX components (for example, *LineChart*, or *AreaChart*) and includes options for confidence intervals and/or historical trends. Users can also toggle various visual components on or off, export figures as PNGs, or as Excel files, or explore data interactively. For example, if a user wants to compare predicted rainfall to historical averages, they could overlay predicted output with raw data points in CSV, within the application interface. The visualization layer adds interpretability to communicate results, especially important when a decision can be made but stakeholders are not aware of the mechanics of the model. Figure 4 highlights the recent main contribution of LSTM applications in automated intelligent decision-making processes; in this scope emphasizing on the utilization of LSTM for the purpose of predicting advanced weather outcomes.

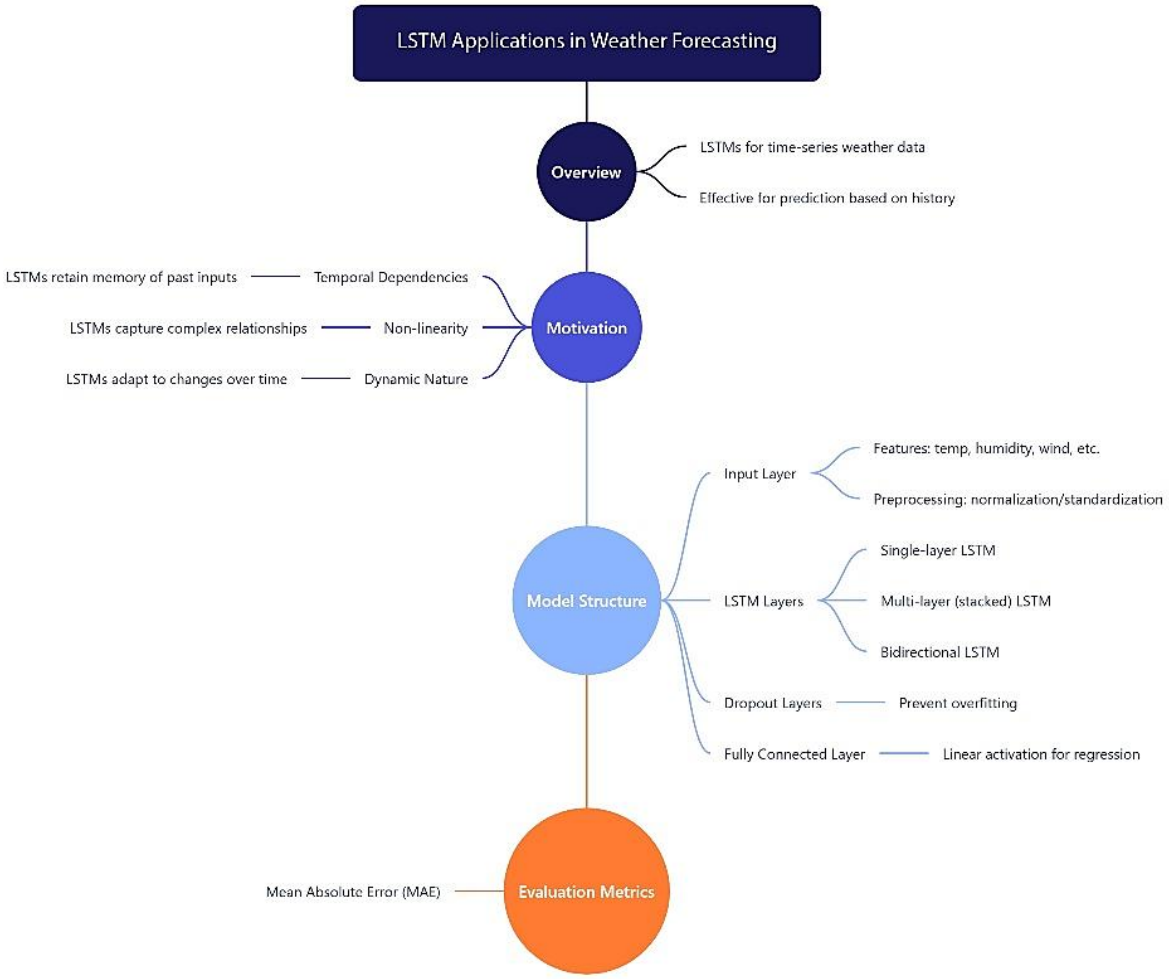

Figure 4. Visualization on the concept flow of LSTM application for weather forecasting

3.2 Data Handling and Preprocessing

The system has two components, and both require input datasets in CSV format where these datasets have rows of observational records representing meteorological events with timestamps. Suggested features tested in this study include *height*, *minMeanTemp*, *maxMeanTemp*, *meanRelHum*, and *rainfall*. Attention to preprocessing is important for valid inference, as neural networks can be affected by the ranges of input values, missing values, and boundary conditions. The data handling and preprocessing modules are critical for transforming raw weather data into systematically organized model input. Table 1 presents the five main parameters for calculating rain precipitation percentages.

Since there was not any tool made available to convert ONNX modelling features from comma-separated value datasets, this study developed a Python script specialized for rainfall prediction using tabular meteorological data. This script functions to read in a CSV file, clean its target and numeric columns, manipulate the non-valid or missing data, performed one-hot encoding for the categorical variable "state," and standardized features using a *StandardScaler* for reproducibility. Subsequently, the dataset is trained by utilizing an *MLPRegressor* neural network where the model can be saved at any point in the pipeline. The saved model and *StandardScaler* are exportable or translated into ONNX format. The script is also able to plot regression model prediction for any record with a provided sample pulled from the pipeline, with a debugging section to plot the transformed features and also the prediction, utilizing the data from the instance selected. The data used for this project were obtained from the Department of Statistics Malaysia, which included mean temperature, rainfall and mean relative humidity data from 2000 to 2021 (https://archive.data.gov.my/data/dataset/mean-temperature-rainfall-and-mean-relative-humidity-malaysia/resource/15b3c8a2-ef0d-4044-8fc1-f261fa9cd7b0).

The framework for the research adopts an automated, modular pipeline for rain prediction with the reference of *Actual* (ground truth precipitation measurement from official dataset) and *Prediction* (values derived from LSTM predictive modelling) columns from CSV files. The data gleaning is the next step which includes removing all spaces and removing empty columns or columns with the wrong format. Continuous features are standardized with a typical z-score method yielding a mean of 0 and variance of 1 removing bias from each individual feature to attempt to stabilize the modeling. Exploratory data analysis is conducted to check summary statistics and find anomalies to be removed. Verified datasets are split into independent training and test sets to maintain rigour when modeling the predictive function for the model development. Clean datasets from training and test sets will produce visualizable results by means of how data are displayed using time series plots while preserving the values for reproducibility, transparency and explainability.

Table 1: The analysis on feature variables utilized for capturing rain precipitation frequency referenced from Malaysia's 2000-2021 annual rain dataset retrieved from Department of Statistics

Feature	Importance	Explanation	Role in Prediction
state	Important	The "state" variable captures geographic variations in rainfall patterns	Influencing model predictions based on regional climate and topography differences.
height	Important	Elevation from sea level that affects climate and rainfall	Determines relative temperature, pressure, humidity changes
minMeanTemp	Very Important	Minimum mean temperature of the lowest average temperature representing the timespan range	Affects moisture, dew point, and rainfall
maxMeanTemp	Very Important	Maximum mean temperature of the highest average temperature representing the timespan range	Influences evaporation and rainfall
meanRelHum	Important	Mean relative humidity representing maximal amount of air holding at given temperature	Measures air's moisture content affecting precipitation
rainfall	Very Important	Target variable that is the baseline of precipitation amount recorded over time	Model predicts precipitation amount over specific time

The preprocessing module reads the labelled CSV files into an array of numbers, normalizes the data with sliding window normalization for overlapping sequences and marks any rows that were malformed. The values that were normalized are then reshaped into a 3D tensor for the inputs to the ONNX model to assist in more accurate predictions. After the normalized values have been transformed, a tensor is created, and the data are pre-structured for the ONNX model. This adds to rigour used when constructing, evaluating, and predicting from the testing and the training sets.

3.3 Model overview

At the predictive core of this system is a Long Short-Term Memory (LSTM) neural network, selected because its architecture is particularly suited for learning long-term dependencies within time-series data. The LSTM model specifies a regression architecture to predict rainfall by estimating the total rainfall in the future, based on past consecutive samples of meteorological observations. The variable nature of atmospheric conditions could be studied as a spatiotemporal phenomenon and therefore through interrelated input variables, temporal behaviours could be learned and incorporated into good predictive models which provide reliable information within such densely variable environments. Figure 5 lists down the included functionality classes encased with the proposed LSTM application system development. The toolkit utilizes a deep learning model based on the LSTM architecture. The LSTM architecture can model sequential data apart from functioning to model long-range dependencies in time-series forecasting. LSTMs include memory cells and gating mechanisms, which allow them to retain historical memory while ignoring noise in time. LSTMs work arbitrarily well to model the statistical relationships of meteorological variables and delayed precipitation impacts. For the research scope, the developed system modelling implements a 10-step sliding window to evaluate multivariate weather data, based on temperature, humidity, wind speed and atmospheric pressure (independent variables), with precipitation represented as the dependent variable. The 3D structure of the model allows dynamic modelling of temporal trend change, particularly with respect to precipitation. The model is constructed with one LSTM layer containing 16 hidden units and dropout probability set to 0.2 to avoid overfitting. The second layer is a single-density layer that predicts the input as rainfall in millimetres. The model is designed and trained using PyTorch [10] due to its usability, speed, and predictive ability. After training is done, the model is exported for use in an ONNX format, allowing the model to be imported across programming environments. Within a Java application, the Deep Java Library (DJL) can execute ONNX code using ONNX runtime, using an instance of a *Translator* class, which converts Java-native types into DJL's *NDArray* tensors for ONNX to process, resulting in predictions.

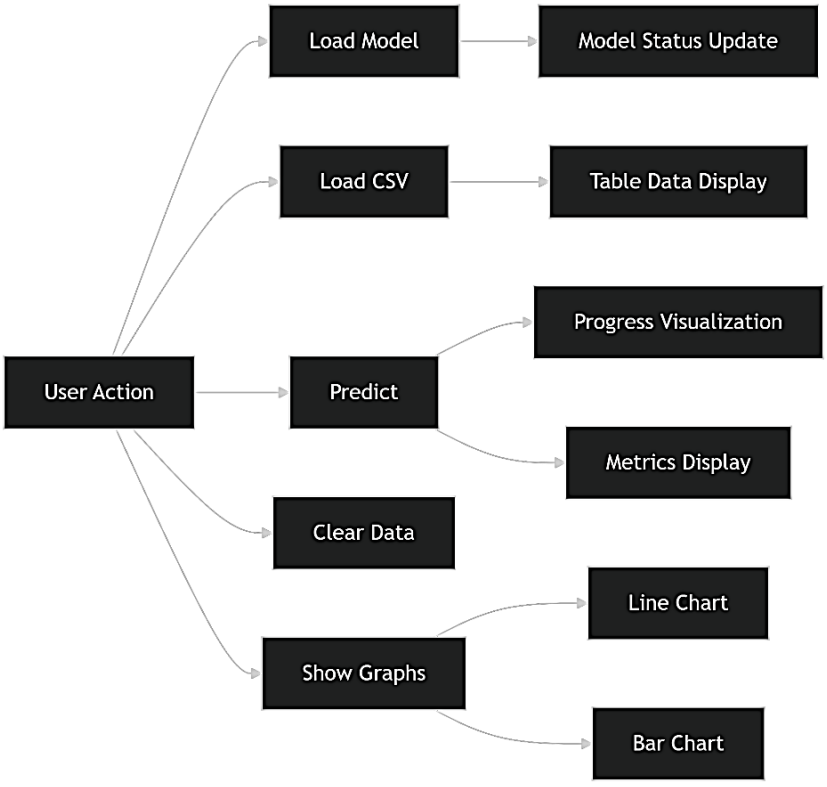

Figure 5. Diagram for the User Interface (UI) flow of the proposed research method

The system offers single-instance and batch predictions with the use of *RainfallBatchTranslator*, which can take different multi-sample data types (i.e., [batch_size, 10, 4]) and provide predicted outputs. This allows datasets that have been assembled in real-time (historical time series) to be processed at the same time. The program is designed to be extensible with support for several model types (e.g., GRUs, 1D CNN, or Transformers) as it requires the ONNX runtime layer. Besides that, the developed program has an agnostic architecture that provides the ability to deploy the model to different training layers (or frameworks) such as TensorFlow, Keras, and Scikit-Learn through an ONNX conversion, which is not tied to any local Python dependencies. The DJL runtime is optimized for inference on CPU or GPU, suitable for any desktop or server use case. Its use of modular pieces allows the use of pre-trained models and adjusts input or models easily without impacting the training process significantly.

3.4 DJL integration and translator implementation with batch inference support

Figure 6 illustrates the workflow of the developed LSTM system for automated decision-making, in this case weather prediction capabilities. The Deep Java Library (DJL) acts as the runtime engine that allows the user to import and run LSTM models in ONNX format. When developing with a DJL, the user begins with a *Criteria* object, which defines the model type, the assumptions for input/output, the target engine (OnnxRuntime) and a path to the ONNX model. A custom *Translator* class is required to take raw Java inputs to *NDArray* tensors and reconstitute them back to Java primitives to manipulate the outputs. Specifically, the input *Translator* translates a 3D float array with shape [1,10,4] into an *NDList* which is the input type needed by the method to make inference, while the output *Translator* takes the *NDArray* tokens and interpret rainfall predictions. In addition, DJL supports a modular approach for model development, allowing the user to swap out models and use the same input processing code. The DJL can manage memory use and ensure thread safe operations for several instances of the same model, with *RainfallBatchTranslator* used for batch predictions. DJL builds flexibility into the model development process with the ability to be cross-compatible, with respect to CPU and GPU, as well as providing input validation and diagnostic for reliable deployment.

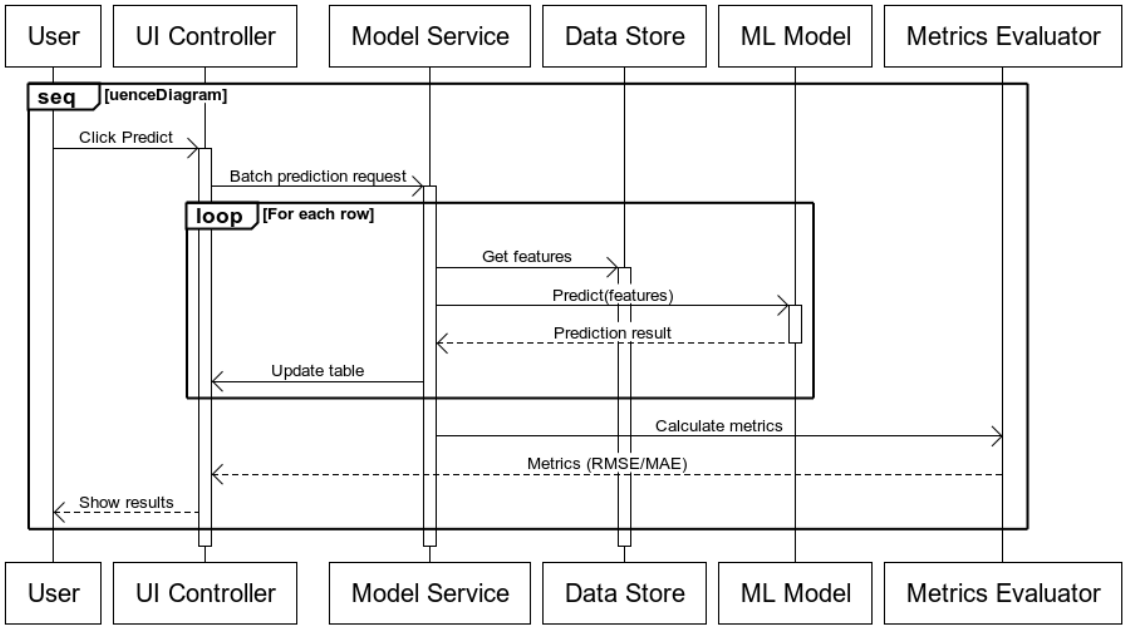

Figure 6. Prediction workflow of the intended deep learning mechanism for aggregating rainfall precipitation from feed data

3.5 Model Integration and Functional Validation

This study performed limited functional testing to confirm that the deployed model works properly within the nested system. The functional testing confirmed that the data ingestion, model inference, and output features worked as intended, in the real world. A LSTM-based prediction model for rainfall prediction was used as the basis for the model that made use of multivariate weather data (temperature, humidity, pressure, and wind speed). The model was developed in PyTorch and exported into ONNX format, which was executed in the Offsetting Risk Java application using DJL, which triggers the ONNX-Runtime backend. The inference was tested in both single-instance format and batch-mode format, and the *Translator* and *RainfallBatchTranslator* code was stable, and produced an output prediction for each input instance with an average latency of well under 250 milliseconds per sample while running on a standard CPU, meeting real-time requirements for predictive analytics. The predictive values of the LSTM model have a weak correlation to realistic rain, as the dataset underfitted the model parameters. Even so, the framework allows for testing of different models and parameters which could improve predictive performance in the long run. The frameworks are also compatible with ONNX models, enabling a multitude of abilities for time series forecasting.

4.0 RESULTS AND DISCUSSIONS

This section evaluates the system-related aspects of a forecasting toolkit, including model integration, inference capabilities, interface needs, and scalability. Unlike typical studies concentrating on predictive accuracy, this proposed a study that explores real-world application and the implementation of modular design for LSTM modelling in Java. For testing, the study analyses Malaysia's daily rainfall data from 2001 to 2021, including weather features and utilizes a synthetic ONNX model for validation.

4.1 System integration and inference execution

The LSTM-based rainfall forecasting model was exported in PyTorch using the open neural network exchange (ONNX) format so that it could then be loaded and executed inside the Java runtime using Deep Java Library (DJL) with the ONNX Runtime backend. The end-to-end pipeline for the rainfall forecasting from ingesting data, executing the model, and displaying the result is working reliably with no runtime exceptions or type mismatches, indicating that the model integration layer was working as expected. Inference was validated with both a single instance as well as a batch-mode input. With the batch testing, the system was able to efficiently complete processing of 100 samples with 10-time steps for 4 input features, with total inference processing time under 1 second when running on a standard 16-core CPU (i7-13700HX, 24 thread). These results show that the toolkit can facilitate low-latency inference in real-time or near-real-time use cases. Figure 7 and 8 both illustrates the prototype LSTM-based application developed in this study with each constituting supporting features, and the output generated from the tested Malaysia's rainfall precipitation data instance (actual versus predicted rainfall). One thing to note in retrospect of this study is that the premise of analysing inferencing dataset is not limited to weather prediction purposes; however, it is open to other LSTM-based dataset nomenclature as well, mentioned in Section 3.1.

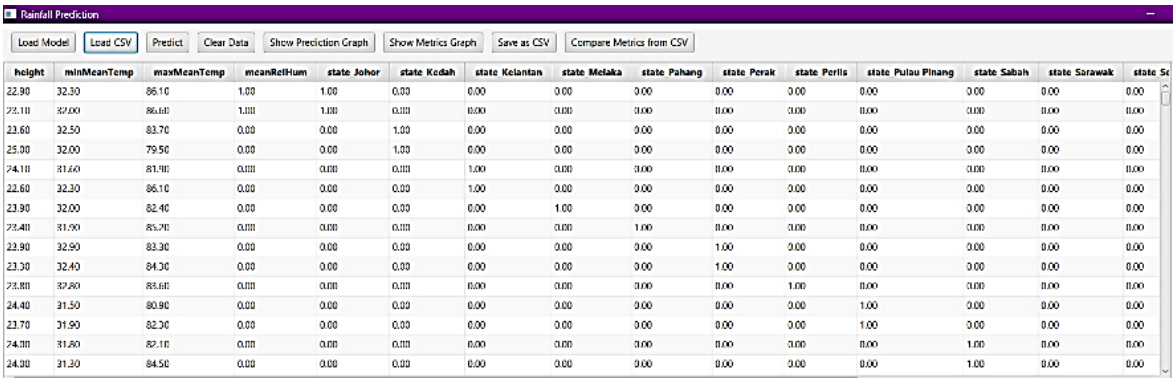

Figure 7. Interface snippet of the Java program integrated with Python mechanics for predicting rain precipitation volumes within the span of assigned timeframe

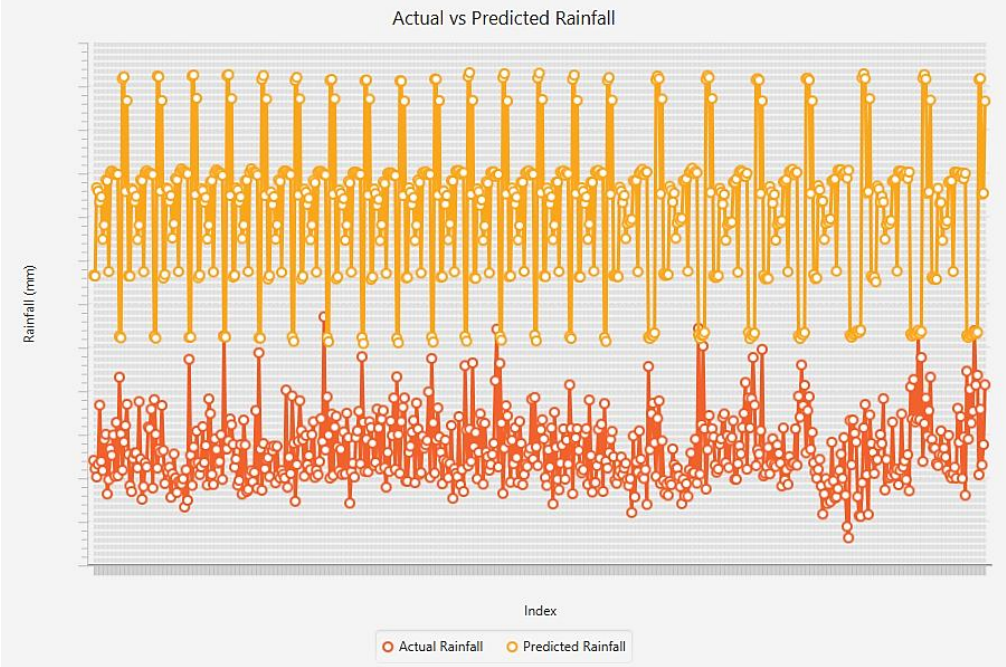

Figure 8. Output of the predicted raindrop capacity across the stipulated timespan for Malaysia region (2001-2021)

4.2 Interface responsiveness and user interaction

The proposed toolkit emphasizes usability and interactive responsiveness; thus, it has the potential for non-technical users who do not have proficiency in programming or in deep learning. The user interface utilizes JavaFX, and thus the experience is for a modern desktop application. Users can load tabular weather data in CSV format through a simple file selection user interface and subsequently, the system implements structural validation and ensures necessary fields exist and that data can be in permitted formats. Users can simply initiate an inference for single predictions or in batch inference, which is entirely abstract behind a single click method of operation. Visual outputs include various types of outputs visualized as line and area charts that are rendered and updated dynamically using the JavaFX charting components, which provide near to instantaneous feedback on model predictions. Users are also able to include uncertainty through the option of confidence intervals for better interpretable and analytical value. The system provides multiple export options in formats like CSV, Excel, and PNG for easy usability into downstream analytical workflows or to decision-makers. The system was tested for responsiveness, as there was no perceived detrimental degradation of responsiveness which supports the presumption that the toolkit is suitable for use in operational environments daily.

4.3 Modularity and extensibility

The developed experimental toolkit is established with modularity as a first-order principle enabling extensibility and long-term maintainability without major engineering effort. The fundamental modular abstraction is founded on decoupling model logic from the interface and data preprocessing layers through a *Criteria* object and user-defined *Translator* classes. The versatility of the toolkit is best exemplified by the fact that the ONNX model can be swapped for a model using a different deep learning architecture (for example, a Gated Recurrent Unit or a Multilayer Perceptron) with no changes to the other layers of the application. This demonstrates the fundamental principle that the framework is model agnostic allowing for any ONNX model, trained in PyTorch, TensorFlow or other compliant frameworks, to be incorporated into the pipeline without changing any GUI code or preprocessing logic. The toolkit can also run on completely different hardware configurations and can be used on either CPU or GPU systems and is therefore deployable on both resource-constrained edge devices and high-performance CPUs. Moreover, the custom *RainfallBatchTranslator* implementation allows batched inference operations performed over very large datasets, which again, enhance the adaptability of the framework. In summary, the modular approach permits the toolkit to potentially evolve with changing model standards, along with forecasting requirements.

4.4 Observed limitations and future directions

The model produced relatively acceptable results but was also underfitting due to a limited diversity in the training dataset. Apart from this, the system was useful in structuring the operational prediction process. This was demonstrated during a simulated rain forecasting exercise using Malaysia's historical weather data, which linked machine learning models with domain-specific decision-making tools, so non-technical users could execute complex forecasting processes through this desktop application. The goal for the future is to improve predictive accuracy and reliability through the integration of probabilistic results, multi-step predictions and live data streams from active sensors. The second desirable objective is to deploy in cloud or edge situations which will help achieve broader scalability. Lastly, the modular architecture allows for direct comparisons and benchmarking in research or industry by enabling the change out of otherwise unrelated models with minimal effort.

Even so, there were significant challenges during the preprocessing tasks. NaN, or not-a-number values in the features and target variable displayed otherwise deterministic behaviour, requiring a cleaning pipeline to remove incomplete records. While this is important from an integrity perspective, it maintained bias and distorted the shape of the dataset, if the absence of records was related to important predictive features. Standardization methods, such as z-score scaling, were employed to mitigate the influence of skewed distributions on feature values. Addressing missing values during the preprocessing phase is essential to minimize redundancy and reduce the introduction of noise into the dataset. Furthermore, standard scalers require no NaNs and can only be fitted on the training data alone to avoid leak. Properly handling missing values and the step for normalizing the features are typically fundamental to providing a stable and accurate rainfall prediction model. Another alternative for improvement of data inferencing is to universalize the capability of reproducing feed data regardless of domain typing such as energy load forecasting and traffic flow that contains variance in data availability diaspora. However, it could be made compatible for further processing with LSTM learning models so that the proposed system usage could be further diversified according to scalability of the intended purpose. In the research argument, the variability of functionality across related execution environment regardless of computing resources and variability in particular related with the light but broad machine learning of the proposed application use are highlighted. Additionally, another opportunity for further integration is cloud deployment where the generic functionalities of the application could be augmented with cloud computing to further support the capability of the developed modular system architecture.

5.0 CONCLUSION

The presented framework has provided a well-articulated, modular, and extensible tool for runtime deployment of deep learning models in platform-independent desktop applications. Although the experimental validation of predictable accuracy in the case study for rainfall forecasting is underdeveloped, it is still sufficient as an initial demonstration of the framework's capabilities. The rationale for rain prediction as a case study is good based on the publicly available datasets and environmental justification. For it to be more convincing as a demonstration of the framework's generalizability, it would have been useful to demonstrate it on a second domain such as energy load forecasting or traffic flow prediction. The graphical user interface (GUI) is intended to be easy-to-use and accessible to non-programmers, but there is no formal usability testing completed for this study with non-technical users. It is anticipated to be further helpful to include usability testing, providing better evidence for the accessibility and user experience with the expansion possibility of the LSTM application to be imposed on other automation domain as well. Because the framework is designed for scalable deployment, including at the edge and in the cloud, future work would consider the same landscape in security, efficient streaming of data, and optimization of resources as well. Overall, this work lays a simple foundation to operationalize deep learning time series models and points to potential areas of future validation and advancements in various real-world applications.

ACKNOWLEDGEMENT

The author gratefully acknowledges Universiti Pendidikan Sultan Idris, which provided support for this research and subsequent publications. Furthermore, thanks are extended to the peers who reviewed the article and provided positive feedback. This research received no financial support from any funding agency (including public, private, or not-for-profit agencies). The source code for the proposed forecasting toolkit (which was written in Java) will be publicly available on the author's GitHub repository (https://github.com/blackcontractor90) following formal article publication to promote reproducibility and future work.

AUTHORS CONTRIBUTION

Farid Morsidi (Conceptualisation; Methodology; Validation; Formal analysis; Data curation; Formal analysis; Investigation; Resources; Software; Visualisation; Writing - original draft; Writing - review & editing)

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. The formulation of the program codes and composition of the research study is done in the discretion of the first author.

REFERENCES

[1] S. Chen *et al.*, "Rainfall Forecasting in Sub-Sahara Africa-Ghana using LSTM Deep Learning Approach," *Int. J. Eng. Res. Technol.*, vol. 10, no. 3, pp. 464–470, 2021, [Online]. Available: www.ijert.org

[2] 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.

[3] A. S. M and S. M. J. Amali, "RAINFALL DETECTION USING DEEP LEARNING TECHNIQUE," *J. Sci. Technol. Res.*, vol. 1, no. 5, pp. 37–42, 2024, [Online]. Available: https://philpapers.org/rec/ARURDU

[4] F. Morsidi, "Multi-Depot Dispatch Deployment Analysis on Classifying Preparedness Phase for Flood-Prone Coastal Demography in Sarawak," *J. ICT Educ.*, vol. 9, no. 2, pp. 175–190, Dec. 2022, doi: 10.37134/jictie.vol9.2.13.2022.

[5] 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.

[6] F. Morsidi and I. Y. Panessai, "Overview of the Integral Impact of MDVRP Routing Variables on Routing Heuristics," *Appl. Inf. Technol. Comput. Sci.*, vol. 4(1), no. 1, pp. 1723–1738, 2023, doi: 10.30880/aitcs.2023.04.01.105.

[7] E. Salcedo, "Graph Learning-based Regional Heavy Rainfall Prediction Using Low-Cost Rain Gauges," 2024, doi: 10.1109/LA-CCI62337.2024.10814868.

[8] 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.

[9] B. M. Preethi, R. Gowtham, S. Aishvarya, S. Karthick, and D. G. Sabareesh, "Rainfall Prediction using Machine Learning and Deep Learning Algorithms," *Int. J. Recent Technol. Eng.*, vol. 10, no. 4, pp. 251–254, 2021, doi: 10.35940/ijrte.d6611.1110421.

[10] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019.

[11] 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.

[12] 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.

[13] P. Lahti, H. Põldoja, J. Lehtonen, S. Ventelä, I. Tuomola, and M. Väyrynen, "An Enterprise Time Series Forecasting System for Cloud Applications Using Transfer Learning," *Sensors*, vol. 21, no. 5, p. 1590, 2021, doi: 10.3390/s21051590.

[14] 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.

[15] A.-C. Akazan, V. R. Mbingui, G. L. R. N'guessan, and I. Karambal, "Localized Weather Prediction Using Kolmogorov-Arnold Network-Based Models and Deep RNNs," pp. 1–17, 2025, [Online]. Available: http://arxiv.org/abs/2505.22686

[16] 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.

[17] 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.

[18] 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.

[19] J. Zhang and K. Feng, "Forecast the future in a timeseries data with Deep Java Library (DJL)," 2025. [Online]. Available: https://docs.djl.ai/master/extensions/timeseries/docs/forecast_with_M5_data.html

[20] 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

[21] A. Rácz, D. Bajusz, and K. Héberger, "Effect of dataset size and train/test split ratios in qsar/qspr

multiclass classification," *Molecules*, vol. 26, no. 4, pp. 1–16, 2021, doi: 10.3390/molecules26041111.

[22] 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.

[23] 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