water

Article

Multi-Step Ahead Water Level Forecasting Using Deep Neural Networks

Fahimeh Sharafkhani [1], Steven Corns [1,*] and Robert Holmes [2]

[1] Engineering Management and Systems Engineering Department, Missouri University of Science & Technology, Rolla, MO 65409, USA; fsgg7@mst.edu
[2] Civil, Architectural, and Environmental Engineering Department, Missouri University of Science & Technology, Rolla, MO 65409, USA; holmesrr@mst.edu
* Correspondence: cornss@mst.edu

Abstract: Stream gauge height (water level) is a significant indicator for forecasting future floods. Flooding occurs when the water level exceeds the flood stage. Predicting imminent floods can save lives, protect infrastructure, and improve road traffic management and transportation. Deep neural networks have been increasingly used in this domain due to their predictive capabilities in capturing complex features and interdependencies. This study employs four distinct models—Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), transformer, and LSTNet—with MLP serving as the baseline model to forecast water levels. The models are trained using data from 20 distinct river gages across the state of Missouri to ensure consistent performance. Random search optimization is employed for hyperparameter tuning. The prediction intervals are set at 4, 6, 8, and 10 (each interval equivalent to 30 min) to ensure that performance results are robust and not due to random weight initialization or suboptimal hyperparameters and are consistent throughout different prediction intervals. The findings of this study indicate that the LSTNet model leads to a better performance than the other models, with a median RMSE of 0.00724, 0.00959, 0.01204, and 0.01230 for the 4, 6, 8, and 10 intervals, respectively. As climate change leads to localized storms driven by atmospheric shifts, water level fluctuations are becoming increasingly extreme, further exacerbating data drift in real-world datasets. The LSTNet model demonstrates superior performance in terms of RMSE, MAE, and the correlation coefficient across all prediction intervals when forecasting water levels under data drift conditions.

Keywords: gage height prediction; deep neural networks; LSTNet; transformers; water level forecasting

check for updates

Citation: Sharafkhani, F.; Corns, S.; Holmes, R. Multi-Step Ahead Water Level Forecasting Using Deep Neural Networks. *Water* **2024**, *16*, 3153. https://doi.org/10.3390/w16213153

Academic Editor: Enrico Creaco

Received: 10 September 2024
Revised: 23 October 2024
Accepted: 28 October 2024
Published: 4 November 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Flooding occurs when the water level exceeds the stream's flood stage. This threshold varies for each location and gage, and the severity of the flooding depends on how much water has exceeded this level, making the water level (also known as gage height and stage) the most prominent indicator of imminent flooding. In Missouri, water level oscillations are often caused by extreme and localized rainfall brought on by storms. With climate change leading to more frequent storms, this will result in more abrupt water level fluctuations. Predicting water levels enables early warning, allowing for timely evacuation, rerouting transportation, managing road traffic, and deploying flood protection measures [1].

Studies in flood forecasting and water level prediction usually make use of statistical, heuristic, and machine learning models to forecast flood and water levels. Traditional statistical models such as autoregressive (AR) [2,3], moving average (MA), autoregressive moving average (ARMA), and autoregressive integrated moving average (ARIMA) are among the popular models for flood forecasting and water level prediction [4–6]. The autoregressive models use past values to predict future values. Like other time series data, water level changes can be attributable to both short-term water level fluctuations

(e.g., hourly) and long-term water level fluctuations (e.g., annual). Thus, water level data have long-term and short-term dependencies. Although autoregressive methods have been popular in research, they fall short in distinguishing the long-term and short-term dependencies and how to leverage them. These traditional models are a subset of linear regression models [7]. Hence, they are of limited flexibility when forecasting non-linear time series. Additionally, these methods were reported not to lead to satisfactory results for short-term predictions, and at least a decade of historical data is required for them to produce a meaningful forecast [8]. To overcome these shortcomings when analyzing more complex time series problems, machine learning models have risen in popularity. According to a review study by [9], artificial neural networks (ANNs) are the most widely used machine learning methods due to their accuracy, high fault tolerance, and parallel processing, hence they are good at dealing with complex dependencies associated with flood prediction. This paper investigates how ANN performance can be further improved when combined with other methods, such as statistical methods and hybrid models where they are used in conjunction with other machine learning models [9]. Ghorpade et al. [10] reviewed the use of different machine learning models and concluded that among different models (including linear regression, support vector machines, decision trees, etc.) the long short-term memory networks (LSTMs) have a superior performance for flood prediction. LSTMs are a form of a recurrent neural network that has proven beneficial in overcoming the problem of vanishing gradients [11]. LSTMs have been used in many studies for water level prediction. In a study by Gude et al. [12], the performance of a statistical model (autoregressive moving average) is compared with the performance of an LSTM, and it was concluded that LSTMs can provide more accurate water level predictions and uncertainty estimates. Li et al. [13] examined LSTMs and their variants for water level prediction. This study used an LSTM, bi-LSTM, CNN-LSTM, and CNN-attention-LSTM, with the latter outperforming other models in all metrics. This suggests that an attention layer can improve the performance of LSTM.

Although the use of attention mechanisms in tandem with other base models can be beneficial, recent research suggests that using pure attention can even lead to better results. Transformer architecture uses pure attention while having lower computational intensity. This architecture has achieved state-of-the-art results in sequential modeling for natural language processing [14]. Transformers have gained popularity since their introduction and have provided superior performance while reducing the computational complexity of natural language processing applications. The architecture has been slightly tailored to suit other applications, including time series prediction [15,16], which usually takes place via adding a positional encoding component. Before the existence of deep neural transformers, the attention mechanism was applied on top of another architecture to increase the model's overall performance capability [17–19]. However, the transformer architecture relies solely on the power of self-attention layers for capturing long-term and short-term dependencies [14]. Although transformers have recently been favored, there is some hesitation when it comes to the use of this architecture in time series forecasting. Zeng et al. [20] compared a transformer architecture with a one-layer linear model for nine benchmark datasets. Surprisingly, the linear model outperformed the transformer model in all but one case. The authors believe that this stems from the fact that using the self-attention mechanism will cause the loss of ordering information. This is not a major concern for semantic-rich applications such as natural language processing but has a crucial role in modeling temporal changes necessary for time series modeling [14–20].

Hybrid architectures for predicting time series using LSTMs in conjunction with convolutional neural networks (CNNs) are promising. These models leverage the capabilities of CNNs for spatial feature extraction and Recurrent Neural Networks (RNNs) for temporal feature extraction. The long short-term time series network (LSTNet) is specially tailored for time series [21]. This model uses a layer of RNN in conjunction with a skip-RNN layer; the latter can go back through time to extract information from previous time steps. Additionally, the model integrates an autoregressive layer to mitigate scalability challenges

commonly encountered by artificial neural networks (ANNs). Some studies derive advantages from employing the LSTNet architecture [22–26]. For instance, Lee et al. [23] used an LSTNet to forecast short-term load in power systems. While LSTNets are designed for time series data, its specific architecture has not been thoroughly explored or validated in the context of water level forecasting, leaving a gap in its applications within this domain. In this study, we have chosen LSTNets, transformers, and LSTMs to compare against the MLP as the baseline model. The performance of LSTNet is compared to these popular deep learning models commonly utilized in water research studies to evaluate which of the models consistently displays better performance that can be extended to all the river gages in the state. In addition, an evaluation is conducted to determine which of these models can perform better under the challenges accompanied by time series forecasting, such as data drift. Data drift is a common challenge that occurs when the distribution of data changes over time, making it more difficult for the model to provide accurate predictions. A model that can overcome data drift can handle an unseen or holdout dataset with different distributions with higher accuracy. Section 2 describes the dataset and study area, data preprocessing, methodology, deep learning models, hyperparameter tuning, experiments and implementation, and evaluation metrics. Model performance comparison and result interpretation are conducted in Section 3. Discussion and insights for future work are presented in Section 4. Finally, Section 5 reviews and concludes the results of this study.

2. Materials and Methods

2.1. Dataset and Study Area

The gage height dataset used in this research is extracted from the United States Geological Survey (USGS). USGS has an established network of active river gages collecting stage/gage height data throughout the state of Missouri. These gages provide stage/streamflow readings, enabling a comprehensive understanding of river conditions, facilitating the development of predictive models, and offering real-time updates on water conditions. The historical stage/gage height information is extracted from the USGS website in time series format with 30-min intervals for 20 selected gages. For training a deep learning model, the last 10,000 values almost equal to six months, ranging from 1 June 2021 to 30 December 2021, have been chosen. Each catchment in the dataset has been assigned a unique index value ranging from 1 to 20, corresponding to the river gage location in the state of Missouri.

The location of river gages used in this study is shown on the map in Figure 1. detailed information for each gage, including site number, site name, latitude (decimal degrees), and longitude (decimal degrees), can be found in Table 1.

Table 1. River gages of interest and their site number, site name, longitude and latitude.

Site Number	Site Name	Latitude	Longitude
07065495	Jacks Fork at Alley Spring, MO	37.1481667	−91.4430833
07014500	Meramec River near Sullivan, MO	38.1585278	−91.1084722
07061270	East Fork Black River near Lesterville, MO	37.5525556	−90.8424444
07066000	Jacks Fork at Eminence, MO	37.1540833	−91.3581667
07019000	Meramec River near Eureka, MO	38.5051389	−90.5901667
06930000	Big Piney River near Big Piney, MO	37.6656389	−92.0499167
06926080	Osage River at Tuscumbia, MO	38.2325389	−92.4586111
06923950	Niangua River at Tunnel Dam near Macks Creek, MO	37.9369444	−92.8513611
06921350	Pomme de Terre River near Hermitage, MO	37.9060556	−93.3289167
05508000	Salt River near New London, MO	39.6125000	−91.4073611
06926000	Osage River near Bagnell, MO	38.1907222	−92.6071389
06921070	Pomme de Terre River near Polk, MO	37.6826667	−93.3703333
05500000	South Fabius River near Taylor, MO	39.8974722	−91.5802778

Table 1. *Cont.*

Site Number	Site Name	Latitude	Longitude
06923940	Niangua River ab Lake Niangua nr Macks Creek, MO	37.8633333	−92.8980556
06813000	Tarkio River at Fairfax, MO	40.3394167	−95.4068333
05495000	Fox River at Wayland, MO	40.3924167	−91.5978889
06933500	Gasconade River at Jerome, MO	37.9299167	−91.9773333
07013000	Meramec River near Steelville, MO	37.9984722	−91.3609444
07067000	Current River at Van Buren, MO	36.9913889	−91.0135000
06928300	Roubidoux Creek above Fort Leonard Wood, MO	37.6010833	−92.2338056

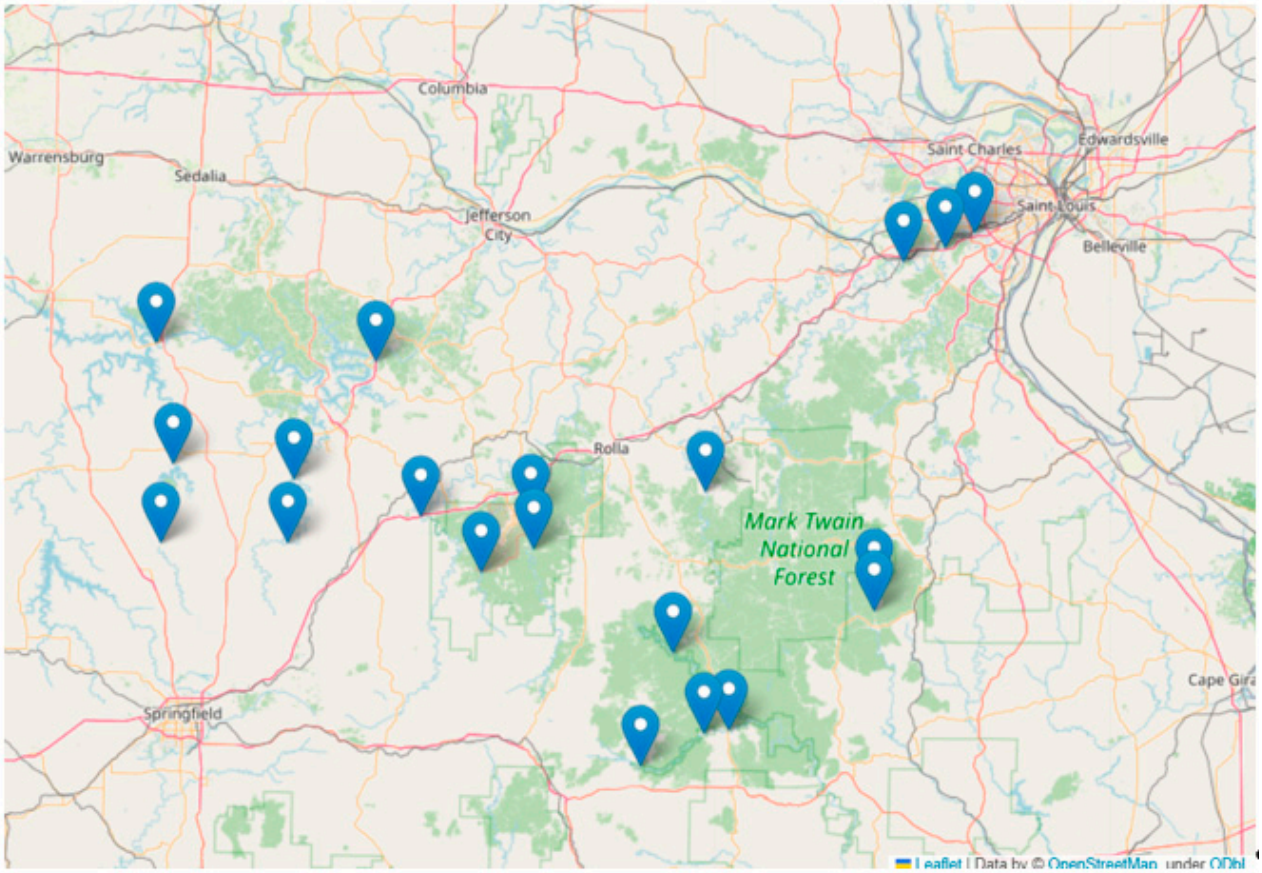


Figure 1. USGS river gages where gage height is monitored across the state of Missouri.

2.2. Dataset Preprocessing

Prior to dividing the data into training and validation sets, preprocessing steps such as addressing missing data and normalizing the data are carried out. Given that the water level data exhibits continuous and uniform fluctuations, we used a uniform distribution for data imputation, with the minimum and maximum values of the distribution determined by the preceding and succeeding available data points, respectively. The normalization and scaling are done separately for the training, validation, and test data to prevent data leakage. Additionally, the dataset is transformed into windows of time steps. For each prediction made for n steps into the future, the model utilizes the preceding m sequence of water level values observed, such that if the value at time step t is named X_t, then input is $\{x_{t-m}, \ldots, x_t\}$ and the predicted output = x_{t+n}. The value of n is set to 4-, 6-, 8-, and 10-time steps, which corresponds to a prediction interval from 2 to 5 h. On the other hand, the selection of m is determined through the process of hyperparameter tuning. After completing these preprocessing procedures, the data is divided into training and validation sets using the conventional train-validation-test split method, with 20% of the data reserved

as holdout for testing, and the remaining 80% used for training and validation with a 9:1 ratio. In addition, we employed walk-forward cross-validation. This method is particularly suited for time series data, as it preserves the sequential order of information, which is crucial for capturing trends and patterns over time and updating the model weights in case of drift. By using this method, we updated the models' trainable parameters using each fold of validation, thereby utilizing all portions of the training data without risking data leakage.

2.3. Methodology

Twenty datasets comprising water level values from selected USGS river gages of interest were chosen for analysis with our deep learning models. The model selection process was driven by validation performance, with the top-performing model on the validation dataset chosen as the best model for testing. Notably, the optimal hyperparameter combination and considerations, such as time series window size, may vary among the best models for each gage. In addition, an extensive hyperparameter search employing the Keras random search algorithm was conducted. Parameter analysis was also performed for selecting configurations like time series window size (timesteps) and batch generation before feeding the preprocessed data into the model. Ultimately, the performance results are juxtaposed to determine the preferred model for future water level predictions. The code has been written using the Python programming language Version 3.10 and the TensorFlow library Version 2.11 [27].

2.4. Deep Learning Models

The four different deep learning models analyzed for predicting water levels in this study were MLPs, LSTM, LSTNet, and transformers. Among these models, LSTNet and LSTM possess recurrent gates that facilitate the retention and omission of information, distinguishing them from MLP and transformer, which lack this capacity. The positional encoding added to the transformer architecture is a compensatory mechanism to mitigate this impact.

Multi-Layer Perceptron (MLP) is a fundamental type of feedforward neural network, consisting of multiple layers of interconnected neurons [28–30]. A non-linear activation function is applied to the weighted sum of each neuron's inputs in the hidden layer. The outputs of the hidden layers are then passed through another activation function to generate the final output. MLPs are commonly used for a wide variety of tasks; however, there is a downside to using MLPs in that they may not be able to extract long-range dependencies due to the lack of a gated unit, and they are prone to vanishing and exploding gradients. The number of hidden layers and neurons for each layer were tuned using hyperparameter tuning [31]. The model's architecture can be viewed in Figure 2a.

Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture that has gained widespread popularity due to its ability to effectively process and model sequential data. The main advantage of LSTM lies in its capability to overcome the vanishing and exploding gradient problem typically associated with traditional RNNs [11]. LSTM cells are designed with a sophisticated gating mechanism that selectively retains and forgets information, allowing the network to capture long-range dependencies and effectively handle sequences with varying time lags. The memory cell in LSTM contains three main gates: the input gate, the forget gate, and the output gate. Together with the cell state, these gates enable the LSTM to remember important information over extended intervals, making it suitable for tasks like time series analysis, natural language processing, and speech recognition. However, despite its strengths, LSTM also has certain limitations. First, the architecture is computationally memory-intensive, which can hinder its application in resource-constrained environments. The complexity of LSTM networks also makes them more challenging to train and tune, often requiring a larger amount of data and more extensive hyperparameter optimization. Additionally, while LSTMs can handle long-range dependencies, they might still struggle with capturing extremely long-term dependencies

in sequences [32,33]. This limitation may result in less effective modeling of extremely complex patterns or dependencies that span across vast time intervals [34]. The number of stacked LSTM layers, the number of hidden units for each layer, and the dropout rate of this model are adjusted using the Keras random search tuner. The architecture of this model can be observed in Figure 2b [35].

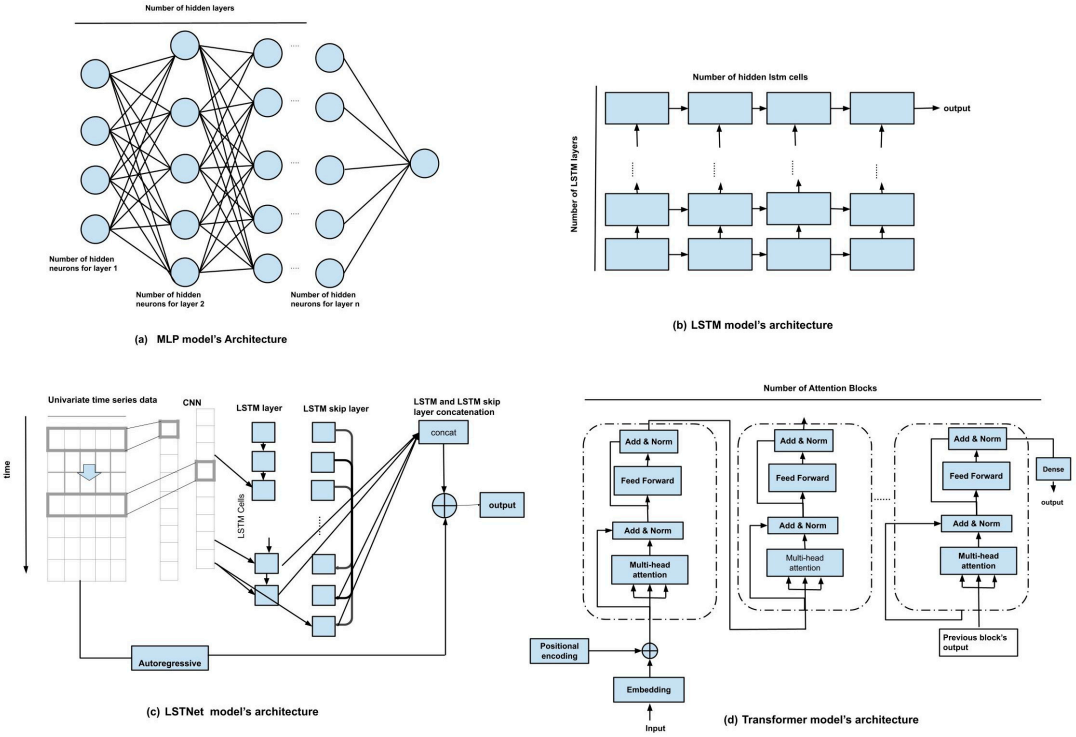


Figure 2. Architecture of deep learning models.

LSTNet is a neural network architecture designed for time series forecasting, as proposed in the paper [21]. LSTNet combines the strengths of both convolutional neural networks (CNNs) and long-short-term memory (LSTM) networks to capture short-term and long-term patterns in time series data. The key idea behind LSTNet is to use CNNs to model short-term dependencies efficiently and LSTMs to capture long-term dependencies in the time series. The model consists of two main components: the CNN-based first layer, which encodes the recent history of the time series, and the LSTM-based layers, which process the encoded features to generate forecasts. LSTNet includes a skip-LSTM layer getting the encoded information from the CNN layer, allowing the model to go back through time and extract important information from previous steps. Finally, LSTNet tries to overcome the output scalability issues by adding the output of an autoregressive model to the output of the concatenated LSTM and skip-LSTM layers. In this model, the number of CNN filters and kernels, number of LSTM hidden units, number of skip-LSTM hidden units, number of layers skipped, and dropout rates are adjusted with hyperparameter tuning. The architectural representation of the LSTNet model is showcased in Figure 2c. However, the original paper uses GRU as the recurrent neural network; we found that LSTM is better suited to our application, as LSTM has the ability to handle long-term dependencies more effectively than GRU. Additionally, we set a dynamic number of stacked LSTM layers, with the number of layers and LSTM units for each layer determined through hyperparameter tuning. The look-back window size was also determined through hyperparameter tuning.

The transformer architecture is a deep learning model introduced by Vaswani et al. [14,34]. It represents a significant departure from traditional sequential models like LSTMs by relying solely on self-attention mechanisms for handling long-range dependencies in sequential data. The core idea of the transformer is the attention

mechanism, which allows the model to focus on relevant parts of the input sequence while processing each token. The transformer is parallelizable, leading to lower training times. A transformer encoder tailored for time series prediction is used in this study. The head size, number of multi-head attention layers, number of transformer blocks, number of feedforward layers, number of hidden units for each feedforward layer, and different dropout rates are optimized using hyperparameter tuning. The architecture of this model can be observed in Figure 2d.

2.5. Hyperparameter Tuning

Given that achieving the best performance of these models depends on suitable hyperparameter tuning and the specific characteristics of the dataset, a hyperparameter tuning process was conducted. This tuning procedure was executed utilizing the Keras random search tuner. Random hyperparameter tuning has been shown to outperform grid search, requiring shorter computation time and reduced processing costs [35]. Although none of the methods used for hyperparameter tuning guarantee optimal hyperparameters and may lead to suboptimal hyperparameter selection, these methods will help us to avoid poor hyperparameter configurations and can boost the performance of deep learning models by preventing both underfitting and overfitting by a large margin [36]. To determine the optimal hyperparameter values, 20 distinct trials were executed for each of the deep learning architectures. Hyperparameters unique to a particular model, such as the number of transformer blocks or CNN filters, were fine-tuned to maximize performance. Moreover, certain hyperparameters were common across all models, such as the time series window/step size, learning rate, batch size, maximum training epochs, and early stopping patience, were assigned consistent lower and upper bounds. This approach was adopted to ensure equitable experimental conditions for all models. Moreover, early stopping was implemented to prevent overfitting.

2.6. Experiments and Implemenation Setting

All experiments were conducted using Python programming language, the TensorFlow library [27], and the same general hyperparameter tunings, with an NVIDIA GeForce RTX 3090 GPU. We looked back at "window size" time steps, where window size has resulted from the general hyperparameter tuning and the water level for 4, 6, 8, and 10 future prediction intervals are predicted. Adam was used as the optimization algorithm. To ensure that performance superiority was not the result of random weight initialization, the model's random weights were set to a constant value. The hyperparameters common across all models are presented in Table 2.

Table 2. Hyperparameters common across all the models.

Hyperparameter Name	Lower Bound	Upper Bound	Sampling
Learning rate	e−4	e−2	log
Window size	48	336	default
Epochs	10	60	default
Batch number	64	256	log

2.7. Evaluation Metrics

This study uses Root Mean Squared Error (RMSE) as the loss function. RMSE is the square root of the Mean Squared Error (MSE), and it has the same units as the target variable, making it easier to interpret, compare across different scales, and understand the magnitude of error relative to the flood stage. RMSE emphasizes larger variances in error, which is vital for our use case, as we require a higher penalty for significant errors. Flooding typically results from abrupt changes in water level values, and when such changes occur, models often tend to underpredict them due to the regression toward the mean effect. By using RMSE, which squares the error, we increase the penalty for larger discrepancies between predicted and actual values, thereby focusing on penalizing these significant errors more effectively. We also

employ two other performance metrics to assess the prediction model's effectiveness: Mean Absolute Error (MAE) and Pearson's Correlation Coefficient. RMSE and MAE measure the disparity between actual values and predictions generated from the samples, with smaller values indicating improved performance. On the other hand, Pearson's correlation coefficient captures the strength of the linear relationship between actual values and predictions. The closer the correlation coefficient is to 1, the stronger the correlation between the two variables. The correlation coefficient may not accurately reflect the strength of the relationship in case the relationship between variables is not linear; therefore, it is better to use the correlation coefficient in tandem with other metrics.

Assuming that n is the total number of input features for the test dataset, y_i represents the actual values of labels, and p_i represents the predicted value for the test data, The performance metrics can be formally represented as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2} \tag{1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - p_i| \tag{2}$$

$$\text{Pearson}'\text{s correlation coefficient} = \frac{\sum(y_i - \overline{y})(p_i - \overline{p})}{\sqrt{\sum(y_i - \overline{y})^2 \sum(p_i - \overline{p})^2}} \tag{3}$$

3. Results

The four models were trained across 20 individual datasets with a diverse array of hyperparameters. These hyperparameters encompass the number of blocks of each architecture, hidden layers, dropout values, batch size, learning rate, training epochs, and early stopping patience. An iterative process, employing the Keras random search tuner, is employed to efficiently optimize these hyperparameters, thereby selecting the model configuration that yields the most favorable evaluation performance. Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Pearson's correlation coefficient are employed to assess the effect of the loss function on model performance.

In addition, the results are evaluated using different prediction intervals to ensure that the performance results of the models are consistent.

Figure 3 Shows the error frequency histogram for each model averaged over 20 datasets for a prediction interval of four steps ahead. The red plots show the normal distribution curve plotted with the mean and standard deviation of the average errors pertinent to each model. Given the histogram in Figure 3, it can be observed that the histogram is right-skewed, the average error is close to zero, and the mode is smaller than the average error, indicating that overpredicting is more frequent. This right-skewedness is likely to occur in real-world datasets, as the frequency diagram of water levels for each individual gage is right-skewed. Moreover, it can be observed that the 2σ for the LSTNet's normal curve is smaller compared to the other models. This deviation from normal distribution can be indicative that the average values for each model may not be ideal for final comparisons; therefore, given the difficulty of comparing the performance results of 20 datasets, we use a boxplot to inspect the median and percentiles of performance metrics for each individual dataset. Figure 4 shows the performance results of the models across different datasets. This visualization offers a summary of both the central tendencies and the spread of performance metrics across 20 datasets for different prediction intervals. Moreover, the boxplot is a good visualization technique that enables the elimination of outliers, which may be the result of random weight initialization or hyperparameter tuning. Each box contains the minimum, Q1, median, Q3, and maximum for the designated performance metric across 20 datasets. The median value is the middle value for the selected performance metric across all datasets when the values are ordered from the smallest to the largest. It represents the value that separates the higher half of the RMSE from the lower half of the RMSE. The first quartile, denoted as Q1, is the value that divides the lower 25% of the RMSEs when ordered. It is the data point at the 25th

percentile. In the boxplot, it marks the bottom of the box. The third quartile, denoted as Q3, is the value that separates the lower 75% from the upper 25% of the selected performance metric's values when ordered. It represents the data point at the 75th percentile. In a boxplot, it marks the top of the box. The maximum and minimum values are the largest and smallest RMSE values, excluding outliers (the river gages with errors above the maximum or below the minimum defined in Equations (5) and (6)). The interquartile range (IR) is calculated as follows:

$$IR = Q3 - Q1 \tag{4}$$

Given (4), we have:

$$\text{Maximum} = \min (Q3 + 1.5 \times IR, \text{max_value}) \tag{5}$$

$$\text{Minimum} = \max (Q1 - 1.5 \times IR, \text{min_value}) \tag{6}$$

In Figure 4, if we suppose the current time step is t, t + n indicates the prediction interval where n is the number of steps ahead in the future. Each step is equivalent to a 30-min interval. Figure 4a–c shows the RMSE, MAE, and correlation coefficient of the test dataset for four steps ahead, respectively. The median RMSE for MLP, LSTM, LSTNet, and transformer is 0.00985, 0.01300, 0.00724, and 0.00862, with LSTNet having the lowest median RMSE. LSTNet also has the smallest Q1, Q3, and maximum RMSE. The median MAE for MLP, LSTM, LSTNet, and transformer is 0.00784, 0.01115, 0.00555, and 0.00700, respectively, with LSTNet having the lowest median MAE along with the lowest Q1, Q3, and maximum. Regarding the correlation coefficient (Figure 4c), where values fall between 0 and 1, with larger values indicating better performance, LSTNet has the highest minimum, Q1, median, Q3, and maximum, consistently showing the best performance among all models in terms of the correlation coefficient.

Figure 4d–l exhibit performance metrics for prediction intervals of 6 (d–f), 8 (g–i), and 10 (j–l). Based on Figure 4, the following conclusions can be drawn:

- With the increment of prediction interval, the performance of all the models degrades, which was expected theoretically.
- The LSTNet purportedly has a better performance compared to other models, and this performance advantage remains consistent as the prediction interval increases, which highlights the robustness of LSTNet in predicting multiple datasets and multiple prediction intervals.

Figure 5 illustrates the point-by-point averaged normalized errors across 20 datasets, with the errors scaled from −1 to 1 to facilitate comparison. These errors delineate the performance of LSTNet across distinct datasets. If the error for dataset1 at timestamp t is denoted as e1, and similarly for dataset2 at timestamp t as e2, and so forth, the vector E(t) = {e1, . . ., e20} is formed. Subsequently, the median, q1, and q3 point-by-point error is computed as the median, q1, and q3 of E(t) = {e1, . . ., e20}.

In Figure 6, the plot for water level values from various river gages is presented. The dataset consists of 10,000 timesteps. Utilizing the initial 8000 values for training and evaluation, the remaining 2000 values are reserved for testing, where data drift occurs. The error in Figure 5 remains negligible for the initial 1000 test timesteps, followed by a discernible spike in error observed from timestep 1000 to 1250. The median, mean, q1, and q3 point-by-point errors exhibit larger magnitudes compared to the preceding and subsequent timesteps. Notably, timestep 1000 aligns with the corresponding point on the dotted red line in Figure 6. As can be seen, the water level values of several river gages face sudden fluctuations simultaneously (red and orange dashed lines, where the red lines show a water level value that is larger than previously seen values). This indicates a change in the distribution of data due to an external influence beyond the input feature itself, and the intensity of these water level fluctuations is unprecedented for some of the river gages. Since the water level prediction is accomplished using a univariate dataset, it cannot reflect the effects of data drift under external influences.

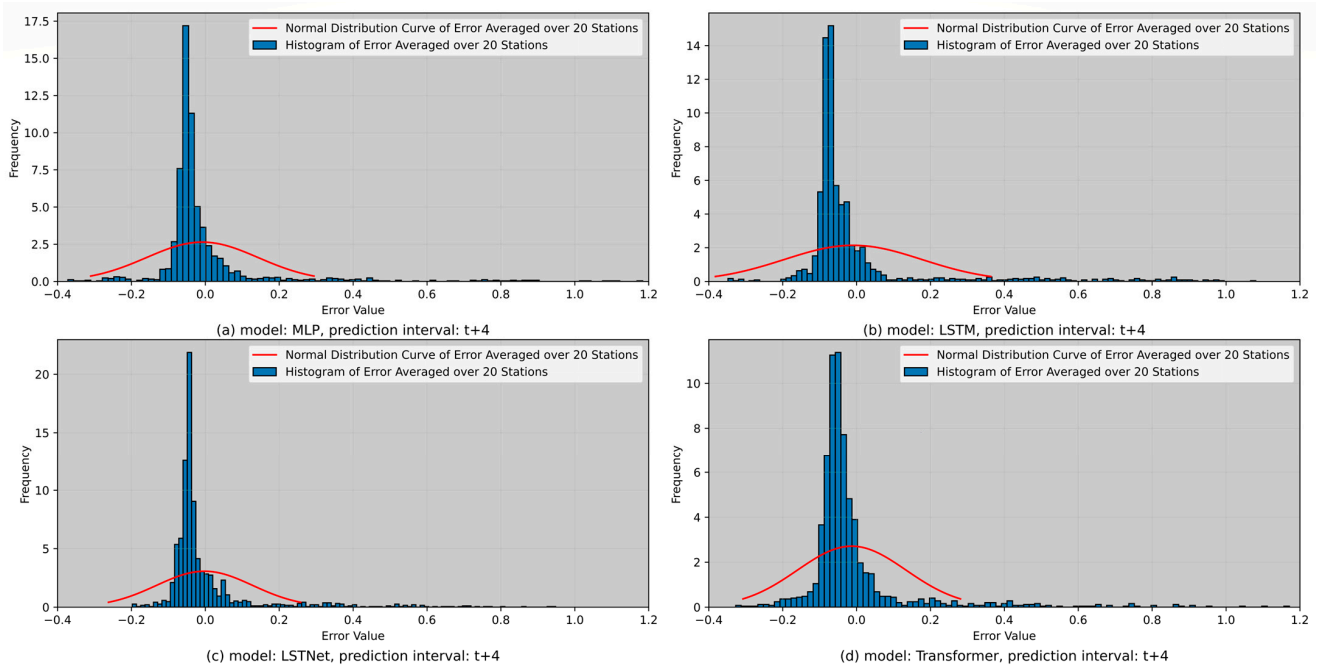


Figure 3. Error frequency histogram averaged over 20 datasets for each model for prediction interval t + 4.

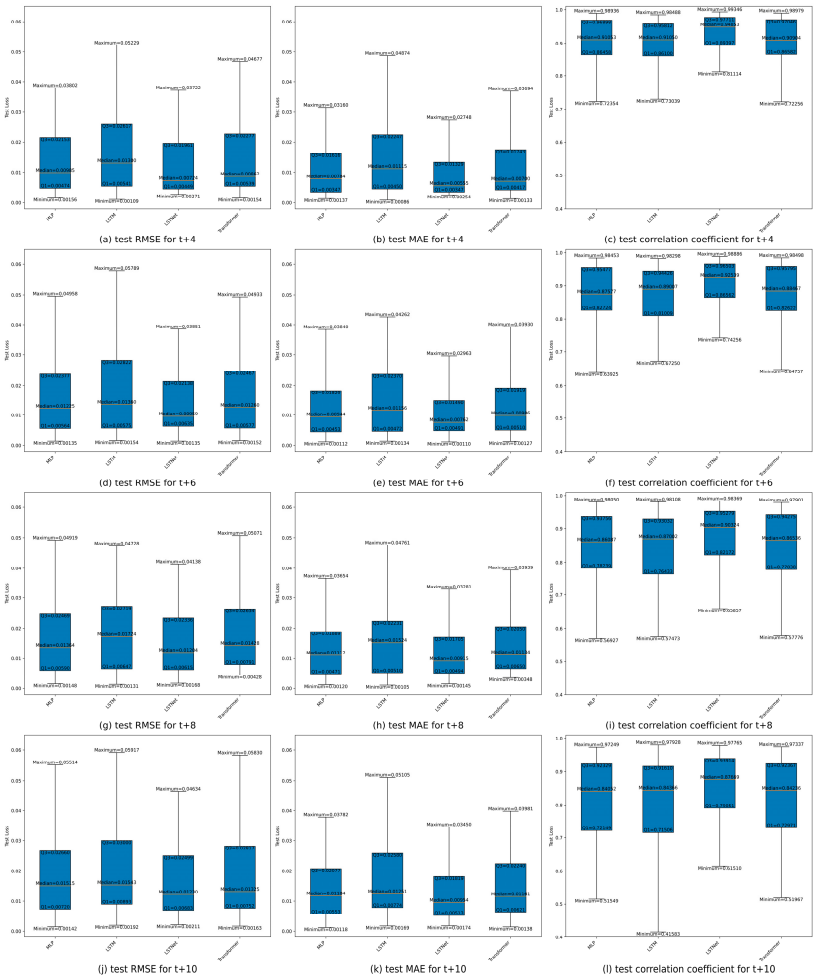


Figure 4. Boxplot of performance metrics for all models across 20 datasets/river gages for different prediction intervals.

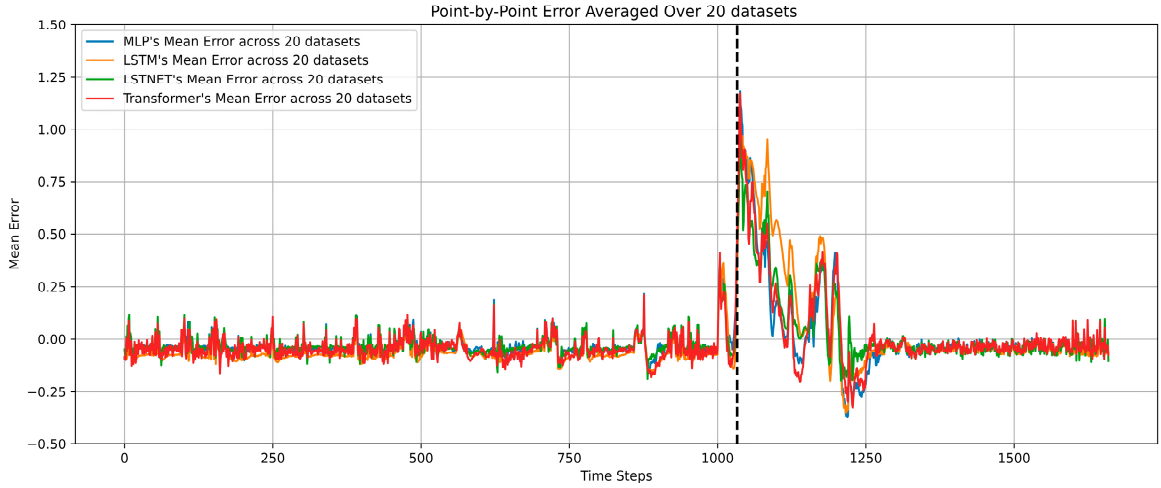


Figure 5. Point-by-point averaged normalized error from 19 November 2021 to 25 December 2021 across 20 datasets.

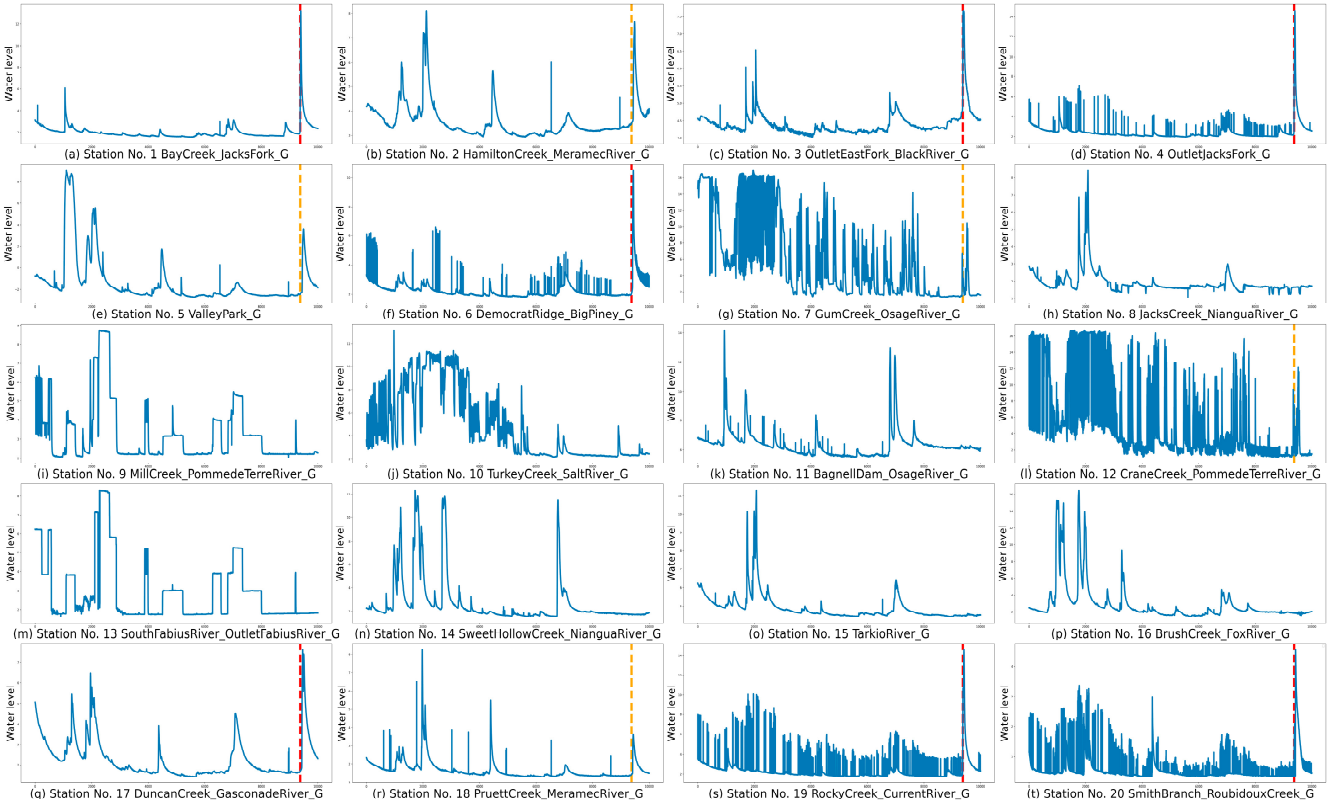


Figure 6. Water level values from 5 June 2021 to 30 December 2021 for 20 selected river gages in the state of Missouri.

While Figure 6 shows that some gages underwent unprecedented fluctuations not observed in the training dataset, a Kolmogorov-Smirnov test was also used to examine this statistically. The Kolmogorov-Smirnov test is a non-parametric statistical test used to compare two distributions. It assesses whether two samples are drawn from the same distribution or if one sample is significantly different from a reference distribution. In the context of drift detection, it can help identify whether the distribution of incoming data has changed compared to a baseline dataset. The null hypothesis (H0) for the Kolmogorov-Smirnov test states that the two samples come from the same distribution. A *p*-value of

0.05 or smaller indicates strong evidence against the null hypothesis, leading to its rejection and suggesting that distribution drift is present.

The Kolmogorov-Smirnov test was applied to the training and test data (Table 3). Based on these results, distribution drift is detected for all gages.

Table 3. Application of the Kolmogorov-Smirnov test to detect distribution drift.

Site Number	Kolmogorov_Smirnov	*p*_Value	Distribution Drift Detection
7065200	0.255865373	2.90459×10^{-94}	TRUE
7017020	0.44059629	3.791×10^{-286}	TRUE
7061270	0.456577816	3.4889×10^{-308}	TRUE
7065495	0.179218605	3.53902×10^{-46}	TRUE
7019000	0.313355426	3.9732×10^{-142}	TRUE
6930000	0.290131339	1.6044×10^{-121}	TRUE
6926000	0.66990631	0	TRUE
6923940	0.785534377	0	TRUE
6921350	0.585454751	0	TRUE
5507800	0.601415161	0	TRUE
6922500	0.327369298	2.3638×10^{-155}	TRUE
6921070	0.307456086	9.85×10^{-137}	TRUE
5498700	0.472200127	0	TRUE
6923250	0.382433963	1.7801×10^{-213}	TRUE
6811760	0.801746248	0	TRUE
5494300	0.649429179	0	TRUE
6928000	0.310265606	3.2275×10^{-139}	TRUE
7010350	0.284247686	1.2574×10^{-116}	TRUE
7064533	0.658398369	0	TRUE
6928300	0.369372002	8.0818×10^{-199}	TRUE

The results indicated that LSTNet demonstrates overall better performance across our test datasets for all prediction intervals. Since all of the datasets have undergone distribution drift, we can conclude that LSTNet performs better under drift conditions.

A sensitivity test was performed to determine whether there is a relationship between specific ranges of hyperparameters and lower RMSE. First, an Ordinary Least Squares (OLS) regression was performed. In this case, the dependent variable was the error, and the independent variables were the hyperparameters. The OLS test found no significant relationship between the hyperparameters and the error, suggesting that any existing relationships are likely non-linear or more intricate (Table 4).

An OLS regression test was performed on the LSTNet model's hyperparameters and error to determine whether the error is dependent on the hyperparameters. The results for other models were similar to those of LSTNet.

However, because the OLS test assumes a linear relationship, it cannot detect more complex interactions. Due to the limitations of OLS in identifying only linear dependencies, we also conducted a Variance Inflation Factor (VIF) test to detect multicollinearity. Multicollinearity occurs when independent variables (hyperparameters) are highly correlated, making it difficult to isolate their individual effects.

The results of the VIF test indicate moderate to high multicollinearity between different hyperparameters, which complicates the use of statistical tests that only detect linear dependencies. While some combinations of hyperparameters led to better model performance, the presence of multicollinearity and complex interrelationships made it difficult to identify those patterns using standard statistical methods.

In general, VIF values between 1 and 5 suggest moderate multicollinearity, while values above 5 indicate high multicollinearity. This high multicollinearity makes it difficult to isolate the individual impact of each hyperparameter on the error, as the variables may interact with each other in complex, non-linear ways. Table 5 shows the results of the VIF test for LSTNet's hyperparameters. from the results in Table 5, we can conclude that, with

the exception of two hyperparameters that showed moderate multicollinearity, all other hyperparameters exhibited high multicollinearity.

Table 4. OLS Regression Results.

OLS Regression Results						
Dep. Variable:		test loss		R-squared		0.025
Model		OLS		Adj. R-squared		−0.029
Method		Least Squares		F-statistic		0.4617
No. Observations		20		Prob (F-statistic)		0.505
Df Residuals		18		Log-likelihood		54.035
Covariance Type		non-robust		AIC		−104.1
				BIC		−102.1
	coef	std err	t	P > |t|	[0.025	0.975]
steps	3.629×10^{-5}	5.34×10^{-5}	0.679	0.505	-7.59×10^{-5}	0.000
hidRNN	4.871×10^{-5}	6.34×10^{-5}	0.768	0.453	-8.46×10^{-5}	0.000
hidCNN	4.871×10^{-5}	6.34×10^{-5}	0.768	0.453	-8.46×10^{-5}	0.000
hidskip	4.871×10^{-6}	6.34×10^{-6}	0.768	0.453	-8.46×10^{-6}	1.82×10^{-5}
CNNKernel	2.922×10^{-6}	3.81×10^{-6}	0.768	0.453	-5.07×10^{-6}	1.09×10^{-5}
skip	2.338×10^{-5}	3.04×10^{-5}	0.768	0.453	-4.06×10^{-5}	8.73×10^{-5}
Hwwindow	1.461×10^{-6}	1.9×10^{-6}	0.768	0.453	-2.54×10^{-6}	5.46×10^{-6}
dropout	4.871×10^{-8}	6.34×10^{-8}	0.768	0.453	-8.46×10^{-8}	1.82×10^{-7}
Omnibus		3.963		Durbin-Watson	1.651	
Prob (Omnibus)		0.138		Jarque-Bera (JB)	3.116	
Skew		0.953		Prob (JB)	0.211	
Kurtosis		2.674		Cond. No.	3.89×10^{34}	

Table 5. Results of VIF test for LSTNet hyperparameters.

Feature	**VIF**
train_split	72.13951946
epochs	4.908380834
batch_no	4.393483781
learning_rate	1.724006837
HP_STEPS	4.821154053
PATIENCE_1	9.679036063
STEPS_AHEAD	28.57620268
hidRNN	7.300817294
hidCNN	7.212433206
hidskip	4.047502256
CNNkernel	5.334312709
skip	6.486225671
Hwwindow	5.289707626
dropout	6.906498173
val_split	2.645945636

The analysis revealed no specific combination or pattern of hyperparameters that consistently led to lower RMSE values. However, the improvement in RMSE resulting from hyperparameter tuning is evident. It can be reasoned that due to multicollinearity among hyperparameters, they are not independent; rather, they influence each other. This interdependence makes it challenging to identify which ranges of hyperparameters or specific relationships contribute to better accuracy. Nevertheless, it can be concluded that certain combinations of hyperparameters have resulted in improved model performance. The results were the same for all models.

The findings indicate that LSTNet performs more effectively for univariate water level forecasting under drift conditions. the combination of CNN for extracting local dependencies, LSTM for longer-term dependencies, a skip-RNN layer to enable the model

to traverse through time, and autoregressive layers for minimizing scaling issues have contributed to LSTNet achieving better performance.

4. Discussion

The findings of this study underscore the potential of the LSTNet Model in improving water level predictions across multiple datasets and prediction intervals, highlighting its robustness and adaptability to various hydrological conditions. The LSTNet model demonstrated lower median RMSE and MAE values, as well as higher correlation coefficients, compared to MLP, LSTM, and transformer models across 20 river gages and all prediction intervals (4, 6, 8, and 10 steps ahead). This performance advantage persisted as the prediction interval increased, suggesting that LSTNet is better equipped to capture both short-term and long-term dependencies in water level data. As expected, the prediction error increases as the prediction interval extends.

In the field of hydrology, overcoming data drift becomes crucial since hydrological phenomena are not solely dependent on a univariate input but are also influenced by various factors (e.g., climate change, frequent precipitation brought about by storms). Although including the other factors as input features might help in increasing the prediction accuracy, there might be some limitations such as the lack of historical datasets for such features or lack of computational resources to train models with more input features, as more features will subsequently result in more trainable weights and added computational complexity.

Data and concept drift are prevalent challenges encountered in real-world datasets. In our case, data drift is characterized by changes in water level distribution, and concept drift is characterized by changes in the dynamics between future water levels (output) and historical values (input). Flooding occurs when water level values exceed a certain threshold known as the flood stage; this threshold varies for each location and gage, and the severity of the flood depends on how much water has surpassed this level. However, larger water level values are considered outliers; they do not occur with the same frequency as other values and deviate from the mean of the distribution, making them harder to predict accurately. This effect is called regression toward the mean. outliers have frequency and variability that are not consistently captured, so when the model predicts based on patterns in the data, it pulls these extreme values toward the average. Moreover, our models are optimized to minimize the overall error; since extreme values contribute disproportionately to the error, the model may downweigh them to fit the majority of the data. all this contributes to the observed underprediction of larger errors.

The problem of underpredicting extreme values is not limited to water level forecasting but is also present in any time-series forecasting problem regardless of the model used. It is challenging to accurately predict outliers. When dealing with this problem in other domains that work with non-sequential data, solutions like balancing an imbalanced dataset can be effective; however, in the time-series forecasting field, under-sampling is not viable, as the order of sequential information is lost. However, there are several remedies to diminish the effect. First, time series cross-validation can be used to incorporate all portions of validation data in optimization to prevent missing outlier data without data leakage. Second, a suitable loss function can be chosen to emphasize larger errors. In addition to these techniques, there are drift detection mechanisms such as ADWIN (Adaptive Windowing) [37–39] and DDM (Drift Detection Method) [40,41]. The former detects changes in data distribution by calculating statistical properties within a specified window and comparing them to a reference distribution, while the latter focuses on the performance metrics of the predictive model to identify concept drift. These drift detection mechanisms are typically employed in real-world applications in production environments where models are not continuously optimized with real-time data, and predictions rely on pretrained models. when a drift or performance decrement is detected, the model is optimized with the new distribution to accommodate emerging patterns. future studies can use these methods to implement adaptive sampling in the optimization process to ensure extreme values are captured, while avoiding oversampling of frequent non-extreme data points. This approach may

further improve prediction accuracy for applications like flood detection, where accurate predictions for extreme values are vital.

The results of this study showed that although LSTNet has not been widely favored by researchers for water level forecasting, it is a capable model when modified and tuned. it is important to note that flooding in the state has usually resulted from high -intensity rainfall brought about by localized storms, and an abrupt rise in water level is generally due to precipitation, whether local or from an upstream area. This highlights the importance of considering hydraulic connectivity and graph dynamics when predicting water levels. One of the limitations on this study was the analysis of gages as independent locations, however using graph neural networks can be beneficial for taking this study further. LSTNet is a capable model for predicting time series data, and the strength of this model can be used within a graph model to enhance the performance of future models.

In conclusion, this study presents LSTNet as a promising tool for water level prediction, offering improved accuracy and robustness compared to other deep learning models examined in this paper. As climate change continues to impact hydrological systems, the development of such adaptive and accurate models becomes increasingly crucial for effective water resource management and flood risk mitigation.

5. Conclusions

Reliable predictions of water levels are critical, as they play a vital role in evaluating flood-related risks, enabling timely warning dissemination, and facilitating the implementation of effective flood control measures. Our objective revolves around examining the impact of Deep Neural Network models in predicting water levels for 20 selected gage stations across the state of Missouri. Since each dataset exhibits different trends and patterns that make them distinct, we scrutinize the cumulative performance of each model across these 20 distinct datasets to develop a robust tool. The performance of LSTNet was compared to three other neural networks: MLP, LSTM, and transformers. All models underwent hyperparameter tuning to enhance their effectiveness and identify the top-performing configuration among various hyperparameter combinations. Our findings suggest that LSTNet, which has not previously been used for water level prediction, exhibits better performance in forecasting water levels compared to the baseline and other models under consideration. Additionally, our results show that the LSTNet was better able to handle data drift than the other DNNs examined. This advantage is due to the LSTNet model's ability to simultaneously focus on both short-term and long-term dependencies and patterns by employing a combination of CNN and RNN. The incorporation of the RNN-skip component enhances the retention of longer-term historical information that a standard RNN fails to capture and, notably, the autoregressive components play a crucial role by addressing scaling issues. Further investigation will examine methods for enhancing the model's performance in drift scenarios, including drift detection mechanisms to optimize the model only with less frequent extreme values.

Author Contributions: Conceptualization, F.S. and S.C.; methodology, F.S.; software, F.S.; validation, F.S. and S.C.; formal analysis, F.S.; investigation, F.S., S.C. and R.H.; resources, F.S., S.C. and R.H.; data curation, F.S.; writing—original draft preparation, F.S.; writing—review and editing, F.S., S.C. and R.H.; visualization, F.S.; supervision, S.C.; project administration, S.C.; funding acquisition, S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Missouri Department of Transportation (MoDOT) and Mid-America Transportation Center (TR202023).

Data Availability Statement: All the datasets used in this paper are publicly available at United States Geological Survey (USGS) website at https://waterdata.usgs.gov/ (accessed on 21 July 2023).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Mohammed, S.J.; Zubaidi, S.L.; Ortega-Martorell, S.; Al-Ansari, N.; Ethaib, S.; Hashim, K. Application of Hybrid Machine Learning Models and Data Pre-Processing to Predict Water Level of Watersheds: Recent Trends and Future Perspective. *Cogent Eng.* **2022**, *9*, 2143051. [CrossRef]
2. Chen, Y.; Gan, M.; Pan, S.; Pan, H.; Zhu, X.; Tao, Z. Application of Auto-Regressive (AR) Analysis to Improve Short-Term Prediction of Water Levels in the Yangtze Estuary. *J. Hydrol.* **2020**, *590*, 125386. [CrossRef]
3. Chao, Z.; Hua-sheng, H.; Wei-min, B.; Luo-ping, Z. Robust Recursive Estimation of Auto-Regressive Updating Model Parameters for Real-Time Flood Forecasting. *J. Hydrol.* **2008**, *349*, 376–382. [CrossRef]
4. Ab Razak, N.H.; Aris, A.Z.; Ramli, M.F.; Looi, L.J.; Juahir, H. Temporal Flood Incidence Forecasting for Segamat River (Malaysia) Using Autoregressive Integrated Moving Average Modelling. *J. Flood Risk Manag.* **2018**, *11*, S794–S804. [CrossRef]
5. Bidwell, V.J.; Griffiths, G.A. Adaptive Flood Forecasting: An Application to the Waimakariri River. *J. Hydrol.* **1994**, *32*, 1–15.
6. Wang, F.; Liang, X. Water Level Prediction Based on Improved WMCP-ARIMA Model. In Proceedings of the International Conference on Statistics, Applied Mathematics, and Computing Science (CSAMCS 2021), Nanjing, China, 27 November 2021; Volume 12163, pp. 488–497.
7. Yan, Q.; Ma, C. Application of Integrated ARIMA and RBF Network for Groundwater Level Forecasting. *Environ. Earth Sci.* **2016**, *75*, 396. [CrossRef]
8. Thompson, S.A. *Hydrology for Water Management*; CRC Press: London, UK, 2017; ISBN 0203751434.
9. Mosavi, A.; Ozturk, P.; Chau, K. Flood Prediction Using Machine Learning Models: Literature Review. *Water* **2018**, *10*, 1536. [CrossRef]
10. Ghorpade, P.; Gadge, A.; Lende, A.; Chordiya, H.; Gosavi, G.; Mishra, A.; Hooli, B.; Ingle, Y.S.; Shaikh, N. Flood Forecasting Using Machine Learning: A Review. In Proceedings of the 2021 8th International Conference on Smart Computing and Communications (ICSCC), Kerala, India, 1–3 July 2021; pp. 32–36.
11. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
12. Gude, V.; Corns, S.; Long, S. Flood Prediction and Uncertainty Estimation Using Deep Learning. *Water* **2020**, *12*, 884. [CrossRef]
13. Li, H.; Zhang, L.; Zhang, Y.; Yao, Y.; Wang, R.; Dai, Y. Water-Level Prediction Analysis for the Three Gorges Reservoir Area Based on a Hybrid Model of LSTM and Its Variants. *Water* **2024**, *16*, 1227. [CrossRef]
14. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 389–399.
15. Mohammadi Farsani, R.; Pazouki, E. A Transformer Self-Attention Model for Time Series Forecasting. *J. Electr. Comput. Eng. Innov.* **2020**, *9*, 1–10.
16. Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; Zhang, W. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, virtually, 2–9 February 2021; Volume 35, pp. 11106–11115.
17. Wang, Y.; Huang, Y.; Xiao, M.; Zhou, S.; Xiong, B.; Jin, Z. Medium-Long-Term Prediction of Water Level Based on an Improved Spatio-Temporal Attention Mechanism for Long Short-Term Memory Networks. *J. Hydrol.* **2023**, *618*, 129163. [CrossRef]
18. Noor, F.; Haq, S.; Rakib, M.; Ahmed, T.; Jamal, Z.; Siam, Z.S.; Hasan, R.T.; Adnan, M.S.G.; Dewan, A.; Rahman, R.M. Water Level Forecasting Using Spatiotemporal Attention-Based Long Short-Term Memory Network. *Water* **2022**, *14*, 612. [CrossRef]
19. Dai, Z.; Zhang, M.; Nedjah, N.; Xu, D.; Ye, F. A Hydrological Data Prediction Model Based on Lstm with Attention Mechanism. *Water* **2023**, *15*, 670. [CrossRef]
20. Zeng, A.; Chen, M.; Zhang, L.; Xu, Q. Are Transformers Effective for Time Series Forecasting? In Proceedings of the AAAI Conference on Artificial Intelligence, Washington, DC, USA, 7–14 February 2023; Volume 37, pp. 11121–11128. [CrossRef]
21. Lai, G.; Chang, W.-C.; Yang, Y.; Liu, H. Modeling Long-and Short-Term Temporal Patterns with Deep Neural Networks. In Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, Ann Arbor, MI, USA, 8–12 July 2018; pp. 95–104.
22. Sharafkhani, F.; Corns, S. Evaluating Sensor Placement via Machine Learning Models. In Proceedings of the Proceedings of the International Annual Conference of the American Society for Engineering Management, American Society for Engineering Management (ASEM), Denver, CO, USA, 25–28 October 2023; pp. 1–8.
23. Lee, C.-H.; Yang, S.; Fan, Y.; Lin, X.; Tao, H.; Wu, C. Application of Temperature Prediction Model Based on LSTNet in Telecommunication Room. In Proceedings of the 2020 5th International Conference on Mathematics and Artificial Intelligence, Chengdu China, 10–13 April 2020; pp. 114–118.
24. Hong, W.; Chen, M.; Gao, P.; Hong, D. Medium-and Long-Term Orbit Prediction of Satellite Based on LSTNet. In Proceedings of the 2023 15th International Conference on Computer and Automation Engineering (ICCAE), Sydney, Australia, 3–5 March 2023; pp. 534–540.
25. Liu, R.; Chen, L.; Hu, W.; Huang, Q. Short-term Load Forecasting Based on LSTNet in Power System. *Int. Trans. Electr. Energy Syst.* **2021**, *31*, e13164. [CrossRef]
26. Ghobadi, F.; Kang, D. Improving Long-Term Streamflow Prediction in a Poorly Gauged Basin Using Geo-Spatiotemporal Mesoscale Data and Attention-Based Deep Learning: A Comparative Study. *J. Hydrol.* **2022**, *615*, 128608. [CrossRef]
27. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv* **2016**, arXiv:1603.04467.

28. Gardner, M.W.; Dorling, S.R. Artificial Neural Networks (the Multilayer Perceptron)—A Review of Applications in the Atmospheric Sciences. *Atmos. Environ.* **1998**, *32*, 2627–2636. [CrossRef]
29. Popescu, M.-C.; Balas, V.E.; Perescu-Popescu, L.; Mastorakis, N. Multilayer Perceptron and Neural Networks. *WSEAS Trans. Circuits Syst.* **2009**, *8*, 579–588.
30. Ramchoun, H.; Ghanou, Y.; Ettaouil, M.; Janati Idrissi, M.A. Multilayer Perceptron: Architecture Optimization and Training. In Proceedings of the 2nd International Conference on Big Data, Cloud and Applications, New York, NY, USA, 29–30 March 2016.
31. Kolbusz, J.; Rozycki, P.; Wilamowski, B.M. The Study of Architecture MLP with Linear Neurons in Order to Eliminate the "Vanishing Gradient" Problem. In *International Conference on Artificial Intelligence and Soft Computing*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 97–106.
32. Greff, K.; Srivastava, R.K.; Koutník, J.; Steunebrink, B.R.; Schmidhuber, J. LSTM: A Search Space Odyssey. *IEEE Trans. Neural Netw. Learn Syst.* **2016**, *28*, 2222–2232. [CrossRef] [PubMed]
33. Reimers, N.; Gurevych, I. Optimal Hyperparameters for Deep Lstm-Networks for Sequence Labeling Tasks. *arXiv* **2017**, arXiv:1707.06799.
34. Karpathy, A.; Johnson, J.; Fei-Fei, L. Visualizing and Understanding Recurrent Networks. *arXiv* **2015**, arXiv:1506.02078.
35. O'Malley, T.; Bursztein, E.; Long, J.; Chollet, F.; Jin, H.; Invernizzi, L.; de Marmiesse, G.; Fu, Y.; Podivìn, J.; Schäfer, F. Keras Tuner. 2019. Available online: https://github.com/keras-team/keras-tuner (accessed on 2 April 2022).
36. Yang, E.-S.; Kim, J.D.; Park, C.-Y.; Song, H.-J.; Kim, Y.-S. Hyperparameter Tuning for Hidden Unit Conditional Random Fields. *Eng. Comput.* **2017**, *34*, 2054–2062. [CrossRef]
37. Bifet, A.; Gavalda, R. Learning from Time-Changing Data with Adaptive Windowing. In Proceedings of the 2007 SIAM International Conference on Data Mining, Minneapolis, MN, USA, 26–28 April 2007; pp. 443–448.
38. Moharram, H.; Awad, A.; El-Kafrawy, P.M. Optimizing ADWIN for Steady Streams. In Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, New York, NY, USA, 25–29 April 2022; pp. 450–459.
39. Chaudhari, A.; AA, H.S.; Raut, R.; Sarlan, A. A Novel Approach of Adpative Window 2 Technique and Kalman Filter-"KalADWIN2" for Detection of Concept Drift. In *International Visual Informatics Conference*; Springer: Singapore, 2023; pp. 453–467.
40. Pinagé, F.; dos Santos, E.M.; Gama, J. A Drift Detection Method Based on Dynamic Classifier Selection. *Data Min. Knowl. Discov.* **2020**, *34*, 50–74. [CrossRef]
41. Costa, A.F.J.; Albuquerque, R.A.S.; Dos Santos, E.M. A Drift Detection Method Based on Active Learning. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 14 October 2018; pp. 1–8.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.