MACHINE
LEARNING
Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
21 October 2024

REVISED
30 January 2025

ACCEPTED FOR PUBLICATION
28 February 2025

PUBLISHED
19 March 2025

PAPER

Bayesian optimization of hybrid quantum LSTM in a mixed model for precipitation forecasting

Yumin Dong* and Huanxin Ding[1]

College of Computer and Information Science, Chongqing Normal University, Chongqing 401331, People's Republic of China
[1] This author contributed equally to this work.
* Author to whom any correspondence should be addressed.

E-mail: dym@cqnu.edu.cn and 1300267644@qq.com

Keywords: random forest regression, temporal convolution, quantum long short-term memory network, rainfall prediction

Supplementary material for this article is available online

Abstract

Precipitation forecasting has important applications in meteorological research. Accurate forecasting is of great significance for reducing the impact of floods, optimizing crop planting plans, rationally allocating water resources, and ensuring traffic safety. However, the factors affecting precipitation are complex and nonlinear, and have spatiotemporal variability, making rainfall forecasting extremely challenging. In response to these challenges, this paper proposes a hybrid model based on temporal convolutional network, quantum long short-term memory network (QLSTM), and random forest regression (RFR) to achieve more accurate rainfall forecasting. The hyperparameters of the model are optimized using the Bayesian optimization algorithm to obtain the best performance. Experiments are conducted on meteorological datasets from Seattle and Ukraine, and the results are verified using mean absolute error (MAE), root mean square error (RMSE), and bias evaluation indicators. The results show that the proposed hybrid model outperforms traditional models such as RFR, support vector machine, K-nearest neighbor, LSTM, and QLSTM in terms of MAE, RMSE, and bias. The proposed model achieves improvements of 1.89% MAE, 2.65% RMSE, and 31% Bias, respectively. These results highlight the improved forecast accuracy and robustness of the proposed hybrid model. This research provides a new approach to weather forecasting and demonstrates the potential of combining quantum computing with traditional machine learning techniques.

List of Abbreviations

LSTM	Long short-term memory
QLSTM	Quantum long short-term memory network
BO	Bayesian optimization
BOA	Bayesian optimization algorithm
RMSE	Root mean square error
MAE	Mean absolute error
HN	Hidden neuron
RFR	Random forest regression
SVM	Support vector machine
KNNs	K-nearest neighbors
VQC	Variational quantum circuit
QNN	Quantum neural network
GP	Gaussian process
MSE	Mean square error

1. Introduction

With the intensification of global climate change and the frequent occurrence of extreme weather events, accurate rainfall prediction is essential for disaster prevention and mitigation, agricultural production, water management, and transportation security [1, 2]. Very often, floods caused by rainfall can damage roads and crops and severely affect people's quality of life, while droughts caused by less rainfall can directly affect soil moisture and water availability, leading to a reduction in crop yields and, in severe cases, to an imbalance in the ecosystem. Both little and too much rainfall can cause us great suffering and in severe cases lead to massive destruction and death, so accurate rainfall forecasts are particularly important to us [3, 4]. If we can have accurate rainfall forecasts, we will reduce unnecessary suffering and casualties [5]. Many methods have been proposed to accurately predict rainfall and promote urban areas [6, 7]. However, rainfall is a complex and unpredictable process because it is affected by many uncertainties. Rainfall sequences are characterized by significant stochasticity, uncertainty, and nonlinearity [8, 9], and the timing of rainfall is difficult to grasp, making accurate prediction extremely challenging.

Traditional prediction methods mainly rely on physical and statistical models, but with the increase in the diversity and complexity of meteorological data, it brings serious challenges to the traditional prediction models [7, 10]. With the rise of artificial intelligence and machine learning, which has attracted the attention of many researchers, artificial neural networks, as a commonly used forecasting tool, have attempted to predict rainfall in a novel way through a unique time series analysis [11, 12]. Mandal and Jothiprakash used a multilayer perceptron, radial basis function neural network (RBFNN), and time-delay recurrent neural networks (RNNs) to model and predict rainfall [13]. While on the side of deep learning algorithms [14], researchers have found LSTM networks to be dominant and widely recognized in the prediction of time series data [6, 15, 16]. Sawale and Gupta [17] utilized a hybrid architecture consisting of backpropagation networks and HNs based on datasets of temperature, humidity, and wind speed parameters for the prediction of atmospheric conditions, revealing a nonlinear relationship between historical data. atmospheric conditions prediction, revealing nonlinear relationships between historical data. Kang *et al* predicted precipitation in Jingdezhen by using LSTM network model and compared its performance with other classical statistical methods and machine learning algorithms [18]. Empirical studies have shown that the LSTM method is suitable for precipitation forecasting [19]. Several studies have proposed techniques combining reinforced LSTM networks with deep learning methods for predicting rainfall in specific areas [20]. For example, one study utilized LSTM and deep learning methods to predict rainfall data in Hyderabad region [11]. In addition, many innovative works have been presented, including a rainfall prediction method using a hybrid RBFNN using particle swarm optimization [21] and genetic algorithms [22], as well as a comparative analysis of LSTM-based networks with parametric prediction models [23], and hybrid models combining convolutional neural networks (CNNs) and LSTMs [24, 25], such as temporal convolutional networks (TCNs), to enhance the feature extraction capability. LSTM networks with memory units perform well in multi-step-ahead prediction compared to networks without memory units [20]. There are still some limitations of existing deep learning models, many models are difficult to capture the sudden and nonlinear characteristics of extreme precipitation events [26], resulting in insufficient prediction accuracy, and LSTM also has its own shortcomings, such as its performance is greatly affected by model parameters and structure, and how to optimize its hyperparameters to make it perform better in prediction is a key problem. At this time BO stands out as an effective hyper-parameter optimization method, which performs better in many challenging optimization algorithms, and with its ability to efficiently find the optimal solution in a high-dimensional complex space, it has been successfully applied in many fields, and shows a unique advantage in the optimization of deep learning models [27]. The hyperparameters optimized by BO make our prediction performance go to the next level. On the other hand, with the meteoric rise of quantum computing showing advantages in various fields, quantum computing is able to show exponential growth in mimeographed problems, especially when dealing with large-scale and complex data can provide great computational power [20, 28]. Many researchers have combined quantum computing with neural networks or machine learning models to produce more efficient results [29], and in the case of weather prediction, QLSTM with the properties of quantum computing, stand out in predicting time series data. In order to enhance the performance of QLSTM and the ability to describe the spatial correlation, a hybrid model is proposed that mixes TCNs, QLSTMs, RFR, and incorporates BO hyper-parameters to perform with prediction of rainfall.The purpose of this study is to improve the prediction accuracy, reduce the error, and enhance the robustness of the model. The innovations of this paper are mainly reflected in the following aspects:

1. Quantum-enhanced temporal memory fusion: In this study, we propose a multi-model fusion method of QLSTM, TCN and RFR, which realizes joint feature extraction and collaborative prediction, and enhances the modeling ability of LSTM network through quantum computing. The modeling capabilities of quantum computing improve the prediction performance of complex time series data.
2. BO for quantum hyperparameter tuning: A BO approach is used to fine-tune the hyperparameters of QLSTM and TCN models to maximize the performance of the models and to optimize the prediction accuracy and generalization ability of the entire hybrid model.

This study improves the accuracy and precision of precipitation prediction by combining deep and integrated learning methods such as TCN, QLSTM and RFR, especially in capturing the performance of cyclic, trending and sudden events of precipitation, and sudden precipitation events, resolving uncertainties in precipitation prediction, tuning the model hyperparameters by BO, which improves the model performance and reduces the waste of computational resources, and exploring the multi-model fusion strategy to improve the generalization ability and stability of the model. It provides a new solution for the field of weather prediction and demonstrates the great potential of combining quantum computing and deep learning.

The structure of the rest of the paper is organized as follows: the second section presents related work; the third section describes the components of the model used in this paper, the fourth section describes the data processing and evaluates the relevant metrics of the hybrid model using the Seattle and Ukrainian datasets as the objects of the study before comparing it with other algorithmic models and deriving the results, and the fifth section draws conclusions and looks to the future.

2. Related work

Rainfall is one of the most significant and important elements of nature and is vital to urban economies. However, rainfall-induced flooding can damage roads and crops, greatly affecting people's quality of life [3, 6]. Many studies have focused on rainfall forecasting, and a variety of rainfall forecasting models have been developed, among which RNNs and deep learning methods perform well in rainfall prediction, but there are still some limitations, such as insufficient prediction accuracy for extreme precipitation events. Liu *et al* found an improved RNN model became DSTP, which employs multiple attentional layers to jointly improve the input features and capture long time dependencies [13], but RNNs also have some disadvantages, e.g. the serial calculation, in which the end of the previous time step must be waited for before the next one can be started. Compared to traditional RNNs, TCNs support parallel computation and each weight of each of their layers can be updated synchronously, which captures long time dependencies more effectively and avoids the problems of gradient descent and exploding [30]. Some researchers have also utilized machine learning algorithms for time series forecasting, Kokilavani *et al* (used machine learning algorithms (e.g. decision tree regression, gradient boosting, AdaBoost, and random forest regression) to forecast monthly rainfall in Coimbatore district of Tamil Nadu. They found that random forest regression algorithm performed the best in terms of prediction accuracy with 89.2% accuracy [31]. The basic principle of RFR is to reduce the variance of a single tree model by integrating multiple decision trees to improve the accuracy and robustness of the prediction. The predictions of each decision tree are averaged to obtain the final prediction. RFR reduces the risk of overfitting of a single model by integrating multiple decision trees, thus improving the accuracy and robustness of the prediction [28]. However, decision tree models are prone to overfitting and difficult to capture long-distance dependencies.

Later, with the advancement of quantum computing [32], many researchers and scientists combined quantum computing and neural networks to process some large-scale time series data. Jia *et al* [33] gave a brief overview of many key aspects of classical neural networks and QNNs, and discussed how neural networks can be used to represent quantum states and density operators. Cong *et al* [34] proposed and analyzed a quantum CNN model, demonstrated its potential, and discussed the possibility of its experimental implementation. Cocos *et al* [35] demonstrated the effectiveness of QLSTM in learning and predicting social media text language, paving the way for future deep learning applications in quantum dynamics. Yu *et al* (20-23) proposed a QLSTM-based model for predicting solar irradiance one hour in advance [36]. In this study, the weight parameters of the four gating controls were optimized by embedding a VQC into the LSTM, which significantly improved the prediction accuracy. The experimental results show that the QLSTM model outperforms the traditional model in all performance metrics, demonstrating the potential of quantum computing in time series prediction [29, 36]. It is worth our attention that the performance of a model is affected by its hyperparameters, but the hyperparameters of a model are interdependent, and finding the optimal hyperparameters through an iterative trial-and-error method usually takes a lot of time and effort. Grid search, stochastic search, and BOA [28] have been proposed as hyperparameter optimization methods for machine learning models. While grid search is able to find the optimal hyperparameters by trying all possible combinations of parameters [37], it is very time-consuming as the number of parameters increases [31]. In contrast, stochastic search can find suitable hyperparameters faster by randomly choosing a range of parameters [38], but it is not reliable enough for training complex models. However, a common problem of these methods is that their search process is not systematic and cannot fully utilize the previous evaluation results. The BOA, on the other hand, can optimize the hyperparameters more efficiently and quickly by intelligently searching for the optimal value of any objective function. BO is a black-box optimization method that can efficiently find the optimal hyperparameter combination of the model with limited computational resources [28]. Therefore, an innovative hybrid model is proposed, which combines TCN, QLSTM and RFR, and optimizes the hyperparameters of QLSTM and TCN models through BOA to select the optimal parameters and achieve better prediction results.

3. TCN-QLSTM-RFR model based on BO

3.1. BO hyperparameters

In order to further enhance the performance of the model, this paper introduces the BOA to automatically tune multiple hyperparameters of the mixed model. The goal of BO is to find a set of hyperparameters through multiple experiments that improves the model's predictive performance. BO continuously updates the probability distribution of the hyperparameters and intelligently selects based on historical results, avoiding the high computational cost of exhaustive search.

As shown in figure 1, the process of BO is to first define the objective function, which is the core of BO, receive hyperparameters as input, and return the performance of the model on the test set. The input parameters of the objective function are the hidden layer size, the number of quantum bits in the quantum layer, the number of quantum layers, the learning rate, the number of channels of the convolution layer in the TCN network, the convolution kernel size, and the Dropout rate. Then set the range of hyperparameters, which is the key part of BO. BO first defines a search space, that is, the possible value range of hyperparameters. For each hyperparameter that needs to be optimized, we set an interval. The table shows the hyperparameter range of BO and the results of the hyperparameters obtained by BO. BO then randomly selects some initial hyperparameters to be evaluated, which are combined into init_points points. Then BO uses a probability model to select the next set of hyperparameters to be evaluated based on the known hyperparameter evaluation results, that is, the value of the objective function. In addition, one of the key steps in BO is to build a surrogate model. Since it is usually expensive to directly evaluate the objective function, we use a surrogate model (usually a GP) to approximate the objective function. The GP is a probability-based model that can predict the value of the objective function at other points based on existing data points. The advantage of the GP is that it not only provides predictions of the objective function value, but also provides uncertainty estimates, which helps to make more informed choices between unevaluated points. In BO, the GP fits the objective function based on the existing hyperparameters and data points of the objective function value. With the surrogate model, BO needs to select the next evaluation point. This process is done through the acquisition function, which is used to select the points that are most likely to bring performance improvements based on the surrogate model. At this stage, BO will measure the potential of different hyperparameters through expected improvements to select the optimal hyperparameters for my model performance. For the selected hyperparameter combination, the objective function bayesian_opt_fun runs and returns the evaluation results. This evaluation result will help Bayes update its probability model to make a better choice next time. After each evaluation, BO will update its built-in probability model. Finally, Bayes will repeat the above process until the preset maximum number of iterations is reached, at which time the search space will gradually converge to an optimal hyperparameter combination. Once the stopping condition is reached, BO will output the optimal hyperparameter combination, that is, the hyperparameter configuration that can minimize the objective function found by the optimization process. These optimal hyperparameter configurations will be used for the final model training. Table 1 below shows the hyperparameters after BO on two data sets.

3.2. TCN model

TCN is a deep learning model specifically designed for time series data. TCN is mainly used to process time series data, especially effective for data with temporal dependencies. TCN is a variant of CNNs that can capture long-term dependencies more effectively than traditional RNNs, while avoiding the problems of gradient vanishing and explosion.

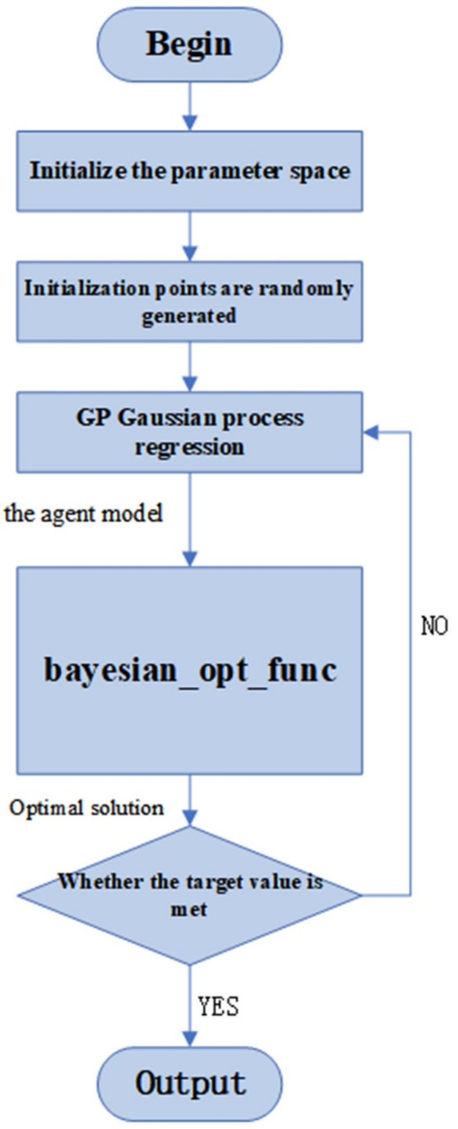
Begin
Initialize the parameter space
Initialization points are randomly generated
GP Gaussian process regression
Update the agent model
bayesian_opt_func
Optimal solution
Whether the target value is met
NO
YES
Output

Figure 1. Bayesian optimization flowchart.

Table 1. Bayesian optimization parameter table.

Parameter	Range of values	Optimal value 1	Optimal value 2
Hidden-size	(64, 256)	182	246
n-qubits	(2, 8)	5	2
n-qlayers	(1, 3)	1	1
Num-channels	(32, 128)	[59, 43, 43]	[37, 115, 115]
Kernel-size	(2, 5)	4	4
Dropout	(0.1, 0.5)	0.404 397 423 487 099 56	0.249 816 047 538 945
Learning-rate	$(1 \times 10^{-4}, 1 \times 10^{-2})$	0.003 588	0.006 027

Figure 2 illustrates the TemporalConvNet structure for processing time series data in the model of this paper, which efficiently captures long-range dependencies in the data through multilayer dilated convolution (Dilated Conv1D). The input to the network first passes through a dilated convolutional layer, which expands the sensory field by inserting holes between the convolutional kernels, allowing the convolutional operation to cover a wider range of timesteps, and thus extracting more information from the time series. The output of each convolutional layer is sequentially normalized by a batch normalization (BatchNorm1d) layer to ensure that the data has a stable distribution before being fed into the activation function, which helps to speed up the convergence of the model and improve the stability of the training. Next, after the ReLU activation function, the model introduces nonlinear properties to enhance its ability to represent complex patterns. Subsequently, the Dropout layer randomly discards some of the neurons to further prevent

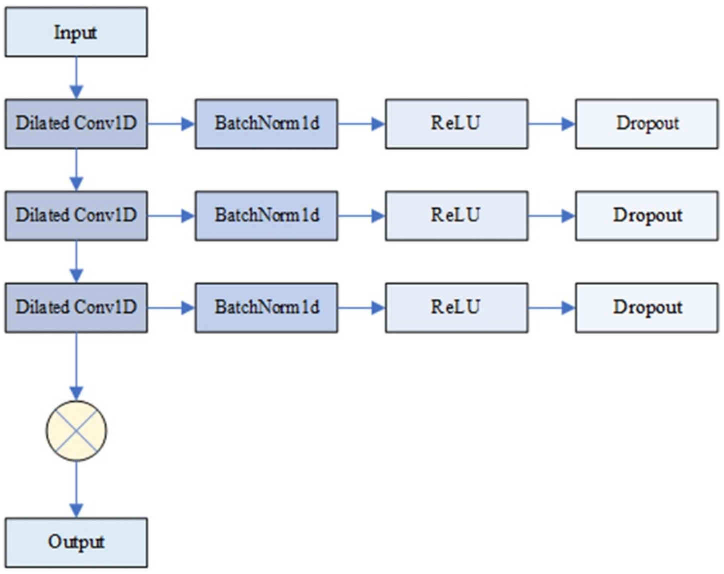

Figure 2. Structural diagram of TCN work.

Algorithm 1. TemporalConvNet model pseudocode.

```
1: **Define** TemporalConvNet(input_data):
2:    Initialize convolution layers (Conv1D), activation function (ReLU), batch normalization (BatchNorm), and
      Dropout
3:    **Define** convolutional layer sequence:
4: **for** each layer **do**
5:        Apply convolution (Conv1D)
6:        Apply BatchNorm
7:        Apply ReLU activation
8:        Apply Dropout
9: **end for**
10:    **Return** convolutional output data
```

Table 2. Key formulas of temporal convolutional network (TCN).

Formula type	Formula
Temporal convolution	$y_t = W_{\text{conv}} * x_t + b_{\text{conv}}$ where W_{conv} is the convolution kernel and $*$ denotes the convolution operation.
Dilated convolution	$y_t = W_{\text{conv}} * x_{t-d} + b_{\text{conv}}$ d is the dilation factor, which expands the receptive field of the convolution.
Activation function	$y_t = \max(0, y_t)$ ReLU ensures that the output is non-negative.

the overfitting problem of the model. This process is repeated at multiple layers of the network, gradually extracting increasingly complex features. Eventually, the outputs of each layer are summarized to generate the final prediction of the model.

The following algorithm 1 shows the pseudo code of TCN, which is used as a feature extractor for the input data in the model of this paper, and is mainly responsible for extracting the local patterns from the weather data. Unlike traditional convolutional layers, dilation convolution introduces voids between the convolutional kernels, allowing the convolution to cover a wider range of time scales, and dilation convolution in multiple convolutional layers on the TCN is able to capture dependencies on different time scales. The TemporalConvNet class is used to define the TCN, which uses multiple convolutional layers and sets the dilation factor on each layer, allowing features to be extracted at different time steps for feature extraction, and further enhances feature learning with batch normalization and ReLUctant activation functions. Table 2 below illustrates the formulas used in the TCN model structure.

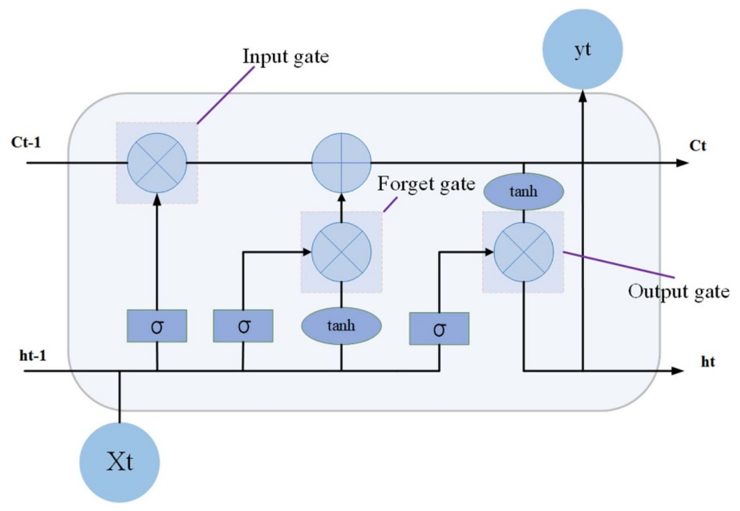

Figure 3. Structure diagram of LSTM.

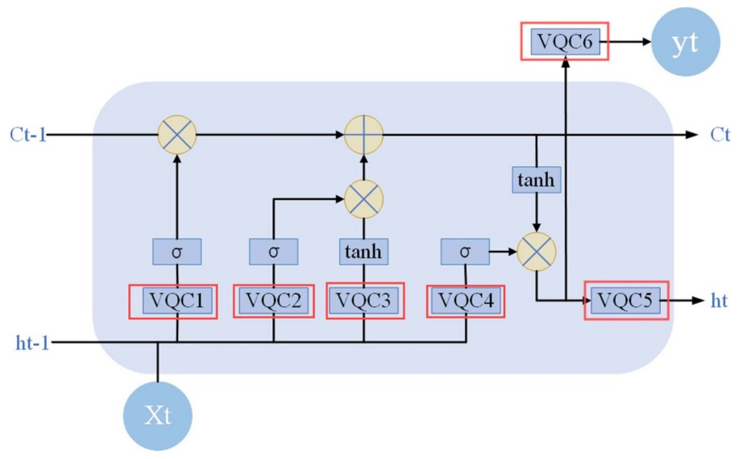

Figure 4. Structure diagram of QLSTM.

3.3. QLSTM model

QLSTM is a combination of classical LSTM and quantum computing. The classical LSTM controls the flow of information through gating mechanisms (such as input gate, forget gate, and output gate) to achieve the effect of 'long short-term memory.' However, LSTM may encounter limitations when dealing with highly nonlinear problems. To enhance the model's nonlinear expression capability, this paper introduces a quantum computing module, combining quantum circuits with LSTM to form a hybrid QLSTM network model. Its main purpose is to leverage the parallelism of quantum computing and the complexity of quantum states to improve the modeling capability of time series data. By adding quantum circuits to the hidden layers of classical LSTM, it is hoped to enhance the ability to learn complex relationships and patterns.

Figure 3 shows the structure of LSTM unit,The LSTM structure includes input-output gates, update gates, forgetting gates.The input gate in LSTM is responsible for controlling how much of the input data X_t should be added to the cell state. The activation function used here is a sigmoid function , which outputs values between 0 and 1, determining the flow of the input data; the forget gate decides how much of the previous cell state C_{t-1} should be discarded. This also uses a sigmoid function, which outputs values between 0 and 1; the tanh activation function is used to calculate the new memory content, and then this memory is modulated by the input and forget gates to update the cell state C_t; the output gate decides the next hidden state h_t, which is used for the output of the LSTM at time step t, it also uses a sigmoid function to regulate this flow.

Figure 4 shows the overall structure of QLSTM, which combines elements of quantum computing and classical LSTM. In this structure, the key gating mechanisms of the traditional LSTM—forget gate, input gate, update gate, and output gate—are enhanced by VQCs to capture more complex patterns and features.

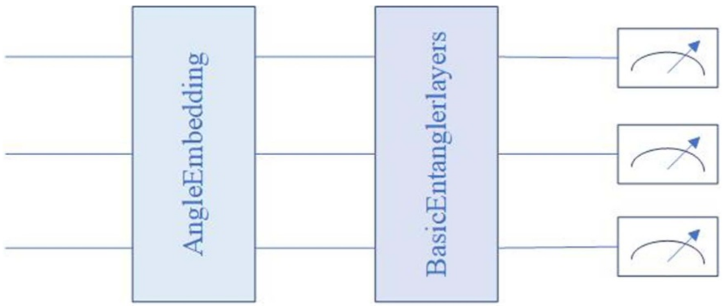

Figure 5. Circuit diagram used for QLSTM.

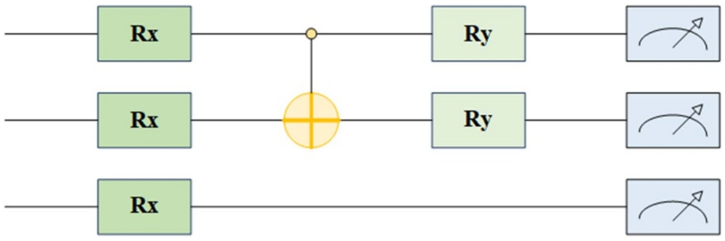

Figure 6. Rotation and entanglement of quantum bits in VQC.

Each quantum convolution layer processes the input data and the previous hidden state into quantum states, and then uses the sigmoid or tanh function to determine the information flow and information transformation. Ultimately, this quantum-processed information is used to update the cell state C_t and the hidden state h_t, providing a richer feature representation for subsequent time steps. The quantum computing part of QLSTM significantly enhances the model's representation ability and complex pattern capture capabilities by introducing VQC combined with the computational process of the traditional LSTM. In this part, each VQC module (VQC1–VQC6 as shown in the figure) is responsible for converting information in the classical LSTM model into quantum information. Specifically, these VQC modules convert the input data X_t and the hidden state h_{t-1} of the previous time step into quantum states in order to take advantage of quantum computing for deep information processing. Quantum circuits process data through a series of quantum operations such as quantum gates, rotation operations, and entanglement operations. Quantum gates (such as Pauli gates, Hadamard gates, etc) are used to change the state of quantum bits so that they can capture different information in the data; rotation operations optimize the representation of information by adjusting the phase of quantum bits, thereby improving the expressiveness of data; quantum entanglement operations can simultaneously process multiple information streams by establishing complex quantum correlations between quantum bits, and increase the efficiency and complexity of information transmission. These quantum operations can not only process linear relationships in data, but also explore potential nonlinear relationships, enabling QLSTM to better process complex time series data. The output of the quantum circuit will be further processed by classical activation functions (such as sigmoid or tanh), which are responsible for determining how the results of quantum computing affect the state update of the model. In QLSTM, the activation function is used to update the memory cell state C_t and hidden state h_t of the LSTM, ensuring that these states accurately reflect the patterns and features in the input data at each time step. In order to improve the adaptability of the quantum computing part in training, the quantum circuit in the VQC module contains adjustable parameters. These parameters can be continuously optimized during the training process to adjust the behavior of the quantum circuit so that the quantum computing part can better work with the classical part of the traditional LSTM. Through this hybrid quantum and classical architecture, QLSTM is able to surpass the traditional LSTM model in capturing complex time series relationships and high-dimensional data features, providing stronger modeling capabilities and prediction accuracy.

The VQC in QLSTM is shown in figures 5 and 6 is based on figure 5 shows a simple VQC internal gate operation and entanglement manipulation to facilitate better understanding by the reader. The VQC consists of two main modules: the AngleEmbedding and the BasicEntanglerLayers. The AngleEmbedding Module: this module converts classical data into quantum information through a quantum revolving door. In

Algorithm 2. QLSTM model pseudocode.

```
1: Define QLSTM(input_data):
2:     Use TCN to extract features:
3:         Pass input data through TCN network
4:     Initialize hidden state ($h_t$) and cell state ($c_t$)
5: for each time step $t$ do
6:         Extract current time step data ($x_t$) from TCN output
7:         Pass $x_t$ through quantum layer (QLayer):
8:             Apply quantum circuit (quantum gates)
9:         Update hidden state ($h_t$) and cell state ($c_t$):
10:            $h_t = \tanh(\text{quantum output})$
11:            $c_t = \tanh(\text{quantum output})$
12: end for
13:     Apply fully connected layer (fc) to compute final prediction output
14:     Return prediction result
```

quantum machine learning, classical data (e.g. numbers or vectors) must be transformed into quantum information before they can be processed by quantum circuits. AngleEmbedding uses rotating gates (e.g. R_x, R_y gates) to map the input data to the state space of quantum bits. The angle of each rotating gate is adjusted according to the characteristics of the input data, allowing the data to be represented in quantum space. Specifically, AngleEmbedding maps each classical input data to the angle value of a quantum bit via a rotation gate (e.g. R_x, R_y, R_z). For example, if we have a set of classical input data X_t these inputs will each be mapped to the angle of rotation of a quantum bit through a rotating gate operation. Example: the classical data $x = [x_1, x_2, \ldots, x_n]$ will be transformed into the angle of rotation of the quantum bit for the operation: $R_x(\theta)$ or $R_y(\theta)$ or $R_z(\theta)$ each rotating gate rotates the state of a quantum bit along a specific axis. The angle of rotation is determined by the input data so that classical data can be mapped into the quantum state. BasicEntanglerLayers Module:BasicEntanglerLayers implements entanglement between quantum bits through gate operations such as CNOT gates, CZ gates, and so on. Entanglement gates serve to correlate information between quantum bits to form a quantum entangled state. This enhances the representation capability of the quantum circuit. Quantum entanglement allows information to be shared between quantum bits, enabling the circuit to handle more complex patterns and relationships. Quantum revolving gates further regulate the states of quantum bits, increasing the flexibility of the circuit. With these two modules, the VQC is able to create strong couplings between quantum bits, increasing the circuit's ability to process data and making it particularly suitable for capturing nonlinear relationships. Example: CNOT gate: When a CNOT gate is applied between qubits Q_1 and Q_2, Q_1 is the control qubit and Q_2 is the target qubit. When Q_1 is in the $|1\rangle$state, the state of Q_2 is flipped (from $|0\rangle$ to $|1\rangle$, or from $|1\rangle$ to $|0\rangle$). CZ gate: creates Z-axis entanglement between quantum bits, where a phase flip occurs only when both quantum bits are in the $|1\rangle$ state.

At the end of a quantum circuit, a quantum measurement is usually performed to obtain information about the state of the quantum bits. The PauliZ measurement is used in this code.The PauliZ measurement operation is one of the fundamental measurements in quantum computing and is typically used to measure the Z-axis state of a quantum bit. For each quantum bit, whose state can be $|0\rangle$ or $|1\rangle$, the PauliZ measurement will determine the probability that a quantum bit has $|0\rangle$ or $|1\rangle$.quantum bits are in a superposition before the measurement (e.g:($\alpha|0\rangle + \beta|1\rangle$) and collapses to the state of $|0\rangle$ or $|1\rangle$ after the measurement, which determines the output of the quantum circuit. For figure 6, a quantum circuit is shown, which contains rotation gates (R_x and R_y) and a CNOT entanglement gate. The qubit is rotated through the R_x and R_y gates, which rotate the state of the qubit along the X-axis and Y-axis respectively. Subsequently, the qubit is entangled through the CNOT gate to form mutual dependence between the qubits. The overall structure shows the rotation and entanglement process of the qubit, which is often used for information processing and qubit interaction in quantum computing.

Algorithm 2 below shows the pseudo-code for QLSTM. In the model of this paper, QLSTM first passes the time series features extracted from the TCN to the quantum layer. At the quantum layer, we use quantum circuits to process the input data. The output of the quantum circuit is the expected values of the quantum bits, which are further processed through the classical layer, ultimately producing a hidden state. Specifically, as shown in the quantum circuit diagram, the quantum part uses angle embedding to embed the data into quantum space, and then processes it through basic entanglement layers using quantum gate operations.

Table 3. Formulas for QLSTM model components.

Component	Formula and description
Quantum embedding layer	$v_t = W_{\text{in}} \cdot x_t + b_{\text{in}}$ Input transformation for quantum embedding, where: x_t : Input data, W_{in}: Weight matrix, b_{in}: Bias vector, v_t: Transformed input.
Quantum circuit output	$\text{Quantum Output} = \text{VQC}\,(v_t)$ The variational quantum circuit (VQC) processes v_t, where: v_t: Input to VQC, VQC: Quantum circuit output.
Angle embedding	$\theta_i = v_{t,i}$ Encodes input into rotation angles, where:$v_{t,i}$: ith input, θ_i: Rotation angle.
Basic entangling layers	Introduces entanglement via weights W_{entangle}, where: W_{entangle}: Entanglement weights.
Forget gate (LSTM)	$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$ Controls forgetting, where: f_t: Forget gate output, σ: Sigmoid function, W_f: Weight matrix, h_{t-1}: Previous hidden state, x_t: Input, b_f: Bias.
Input gate (LSTM)	$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$ Controls input, where: i_t: Input gate output, W_i: Weight matrix, b_i: Bias, others as above.
Output gate (LSTM)	$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$ Determines cell output, where: o_t: Output gate value, W_o: Weight matrix, b_o: Bias.
Cell state update	$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$ Updates cell state, where: c_t: Cell state, c_{t-1}: Previous state, W_c: Weight matrix, b_c: Bias.
Hidden state update	$h_t = o_t \cdot \tanh\left(c_t\right)$ Updates hidden state, where:h_t: Hidden state, c_t: Cell state, o_t: Output gate.

Algorithm 3. Random forest regression pseudocode.

```
1: **Define** RandomForestRegression(training_data, target_data):
2:     Initialize number of decision trees (*tree_nums*)
3: **for** each decision tree **do**
4:         Randomly select training samples and feature subsets
5:         Train decision tree:
6:             Select optimal split feature
7:             Split data based on chosen feature
8:             Repeat until max depth or leaf node conditions are met
9: **end for**
10:     **Return** trained random forest model
```

Finally, the output of the QLSTM is the hidden state and cell state processed by the quantum circuit, which are then passed to the fully connected layer for the final output.The table shows the mathematical formulas used in QLSTM. Table 3 below illustrates the formulas used in the QLSTM model structure.

3.4. RFR model

RFR is a classic ensemble learning method that is often used for regression tasks. In code, the RandomForestRegression class in RFR contains a set of decision trees. Each tree is trained to make predictions by randomly selecting features and samples. During the training process, RFR performs regression fitting on the features extracted from TCN and QLSTM, learning the mapping relationships in the data. After training is completed, RFR makes predictions on the test set and calculates evaluation metrics. The algorithm 3 below shows the pseudocode for RFR. The input to RFR consists of features extracted by QLSTM ($qlstm_features_train$ and $qlstm_features_test$), which are time series data features extracted through quantum circuits and TCN. The output of RFR is the regression predictions made by the random forest model based on the extracted features, resulting in the final expected values (such as precipitation and other continuous variables). The table 4 includes the formulas used in RFR.

3.5. Proposed the TCN-QLSTM-RFR model optimized by BO

The hybrid model proposed in this paper combines TCN, QLSTM, and RFR, and uses BO to optimize the parameters of the QLSTM and TCN models for precipitation prediction. As shown in figure 7, this is the main process flow of the hybrid model, and table 5 clearly lists the parameter settings of the model on the two datasets in this paper.

The construction of TCN-QLSTM-RFR combines the three models of TCN, QLSTM, and RFR to make full use of their respective advantages, and after obtaining the optimal hyper-parameters after BO, the

Table 4. Formulas for random forest regression components.

Component	Formula and description
Leaf node value	$\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ The predicted value at a leaf node is the mean of the target values y_i at that node.
Loss function	$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$ The mean square error (MSE) is used to optimize the model.
Tree splitting criterion	$\text{Best Split} = \arg\min_{\text{split}}\left(\text{Var}\left(y_{\text{left}}\right) + \text{Var}\left(y_{\text{right}}\right)\right)$ The best split minimizes the sum of variances of the left and right child nodes.

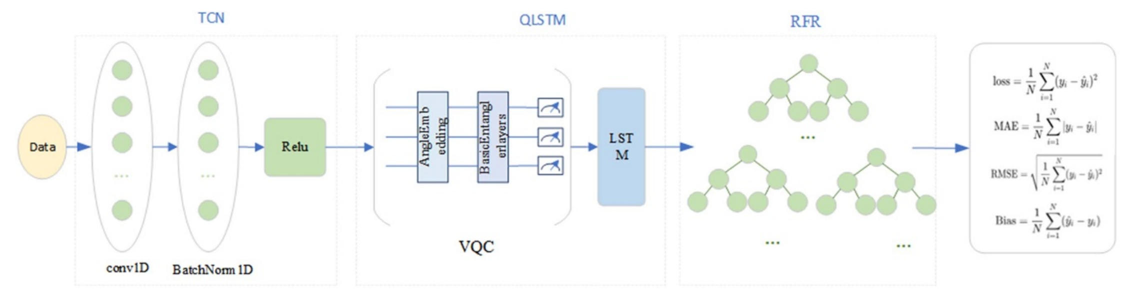
TCN
QLSTM
RFR
Data
conv1D
BatchNorm 1D
Relu
AngleEmbedding
BasicEntanglerlayers
VQC
LSTM
$\text{loss} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$
$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$
$\text{Bias} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)$

Figure 7. Total model flow chart.

Table 5. Parameter setting table for each model.

TCN Parameter	Dataset 1	Dataset 2
Num-channels	[59, 43, 43]	[37, 115, 115]
Kernel-size	4	4
Dropout	0.404 397 423 487 099 56	0.249 816 047 538 945
QlSTM Parameter	data 1	data 2
Hidden-size	182	246
n-qubits	5	2
n-qlayers	1	1
Dropout	0.404 397 423 487 099 56	0.249 816 047 538 945
Learning-rate	0.003 588	0.006 027
RFR Parameter	data 1	data 2
Tree-nums	100	100
Max-depth	10	15
Min-sample-split	4	4
Min-sample-leaf	2	2

hyper-parameter results are applied to the model for experiments. As shown in the figure, the data in the model are first processed by TCN, which captures the long-distance dependencies in the time series through multiple convolutional layers and expansion convolution. TCN transforms the input data (such as temperature, humidity, wind speed and other temporal features) into high-dimensional temporal feature representations, and extracts the important temporal patterns in the data. Next, the TCN output features are passed to (QLSTM). QLSTM combines quantum computing and classical LSTM using Quantum Layer (QLSTM) to map the features extracted by TCN to quantum states. The introduction of quantum computing allows the model to handle complex temporal patterns and enhances the ability to model potentially complex relationships in the data. The LSTM part further captures the long and short term dependencies in the time-series data, and ultimately generates a more informative representation of the temporal features. Finally, the quantum LSTM-processed features are passed to the RFR model. RFR is a powerful integrated learning method that performs regression prediction on data by training multiple decision trees. Each decision tree is trained on a different subset of features and a subset of samples, and finally synthesized into a stable prediction through a voting mechanism. RFR is effective in reducing overfitting and provides good prediction performance on complex data patterns.

Table 6. Meteorological records for the entire month of February in seattle.

Date	Precipitation	Topamax	Temmink	Wind	Weather
1 February 2012	13.5	8.9	3.3	2.7	Rain
2 February 2012	0	8.3	1.7	2.6	Sun
3 February 2012	0	14.4	2.2	5.3	Sun
4 February 2012	0	15.6	5	4.3	Sun
5 February 2012	0	13.9	1.7	2.9	Sun
6 February 2012	0	16.1	1.7	5	Sun
7 February 2012	0.3	15.6	7.8	5.3	Rain
8 February 2012	2.8	10	5	2.7	Rain
9 February 2012	2.5	11.1	7.8	2.4	Rain
10 February 2012	2.5	12.8	6.7	3	Rain
11 February 2012	0.8	8.9	5.6	3.4	Rain
12 February 2012	1	8.3	5	1.3	Rain
13 February 2012	11.4	7.2	4.4	1.4	Rain
14 February 2012	2.5	6.7	1.1	3.1	Drizzle
15 February 2012	0	7.2	0.6	1.8	Rain
16 February 2012	1.8	7.2	3.3	2.1	Rain
17 February 2012	17.3	10	4.4	3.4	Rain
18 February 2012	6.4	6.7	3.9	8.1	Sun
19 February 2012	0	6.7	2.2	4.7	Rain
20 February 2012	3	7.8	1.7	2.9	Rain
21 February 2012	0.8	10	7.8	7.5	Rain
22 February 2012	8.6	10	2.8	5.9	Sun
23 February 2012	0	8.3	2.8	3.9	Rain
24 February 2012	11.4	6.7	4.4	3.5	Rain
25 February 2012	0	7.2	2.8	6.4	Rain
26 February 2012	1.3	5	−1.1	3.4	Snow
27 February 2012	0	6.7	−2.2	3	Sun
28 February 2012	3.6	6.7	−0.6	4.2	Snow
29 February 2012	0.8	5	1.1	7	Snow

Table 7. Monthly weather data for Kryvyi Rih in 2022.

City	Year	Month	Avg temp	Max temp	Min temp	Wind speed	Pre	Snow depth
Kryvyi Rih	2022	1	−0.1	8.8	−14	3.8	52.2	4
Kryvyi Rih	2022	2	2.4	10	−5	3.2	30.6	3
Kryvyi Rih	2022	3	2.5	18.4	−9	2.3	13.3	0.7
Kryvyi Rih	2022	4	6.6	20.2	−1.2	2.7	65.8	4
Kryvyi Rih	2022	5	14.2	28.6	1.4	2.4	50.6	0
Kryvyi Rih	2022	6	20	33.5	6.8	2	24.3	0
Kryvyi Rih	2022	7	19.9	32.2	8.2	2.3	102	0
Kryvyi Rih	2022	8	20.2	30.4	11.2	1.8	83.2	0
Kryvyi Rih	2022	9	11.7	21.2	0.4	1.9	135.2	0
Kryvyi Rih	2022	10	10.8	21	1.2	1.9	27.3	0
Kryvyi Rih	2022	11	−0.2	1.8	−1.8	1.4	0	0
Kryvyi Rih	2022	12	−0.2	10	−20.2	2.4	0	0

4. Simulation results and analysis

4.1. Weather dataset

This study uses two sets of data sets. One set is a meteorological record from Seattle, totaling 1461 data. The dataset covers daily meteorological information from 1 January 2012 to 31 December 2015. Each data record includes six features: date, rainfall (mm), maximum temperature (Celsius), minimum temperature (Celsius), wind speed (m s^{-1}), and weather. The other set of data sets is a dataset of meteorological indicators at various locations in Ukraine from 2010 to 2023. Each data record includes nine features: city, year, month, average temperature (Celsius), maximum temperature (Celsius), minimum temperature (Celsius), average wind speed (m s^{-1}), total precipitation (mm), and maximum snow depth (cm). Since the focus of this study is rainfall prediction, rainfall is selected as the prediction label, and other variables are used as input features. As shown in table 6, it is the weather record for the whole month of February in Seattle, and as shown in table 7, it is the weather record for one year in a certain region of Ukraine.

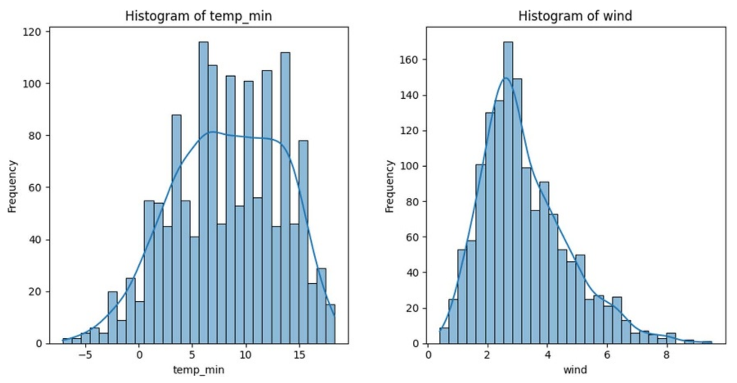

Figure 8. Distribution of minimum temperatures and wind speeds types in dataset 1.

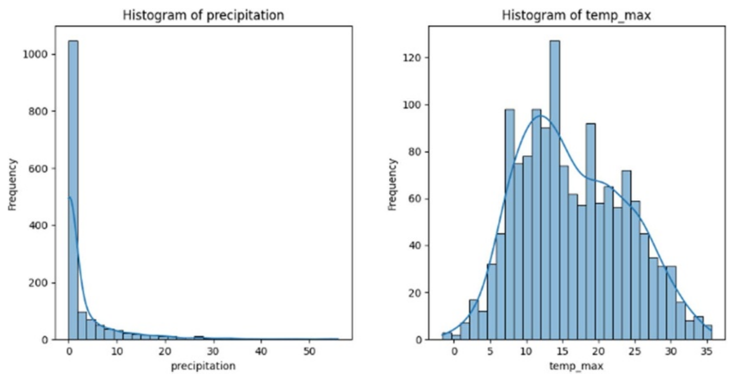

Figure 9. Distribution of precipitation and maximum temperature in dataset 1.

Figures 8–10 illustrate various statistical characteristics of meteorological data in the study area. Each figure contains a histogram and a fitted kernel density estimation curve to show the distribution of each meteorological variable.

Figures 8 and 9 are the statistical characteristics of meteorological data in Dataset 1. Figure 8 illustrates the distribution of minimum temperature and wind speed. The minimum temperature shows a bimodal distribution, with most of the data concentrated between 5 °C and 10 °C. On the other hand, the wind speed data shows a right-skewed distribution, indicating that the wind speed is low on most days. Figure 9 illustrates the distribution of precipitation and maximum temperature. The precipitation data shows a high degree of skewness, with most of the precipitation concentrated between 0 and 10 mm, indicating that precipitation is small most of the time. The distribution of maximum temperature is relatively smooth, generally concentrated between 15 °C and 25 °C. Figure 10 is the statistical characteristics of meteorological data in Dataset 2, giving the distribution of average temperature, maximum temperature, minimum temperature, average wind speed, total precipitation, and maximum snow depth. Overall, the temperature data show typical seasonal fluctuations, while the precipitation has a large skewness, indicating that precipitation events are infrequent, but once they occur, they often bring a lot of precipitation.

These statistical data help to gain a deeper understanding of the seasonal changes and distribution characteristics of meteorological variables, and provide a reliable data basis for subsequent analysis.

4.2. Data set preprocessing

In this research, a thorough data preprocessing phase was conducted on both datasets to enhance data quality and boost model stability and prediction accuracy. The preprocessing involved addressing missing values, encoding categorical variables, standardizing numerical features, and formatting the data for compatibility with PyTorch models.

For the Seattle dataset, missing values are managed using df.dropna(), which removes any rows with absent precipitation values or other missing attributes, ensuring the model is trained on complete and accurate data. The date field, initially in datetime format, is transformed into a Unix timestamp (in seconds), allowing the model to analyze time-related patterns and trends effectively. By representing time as Unix

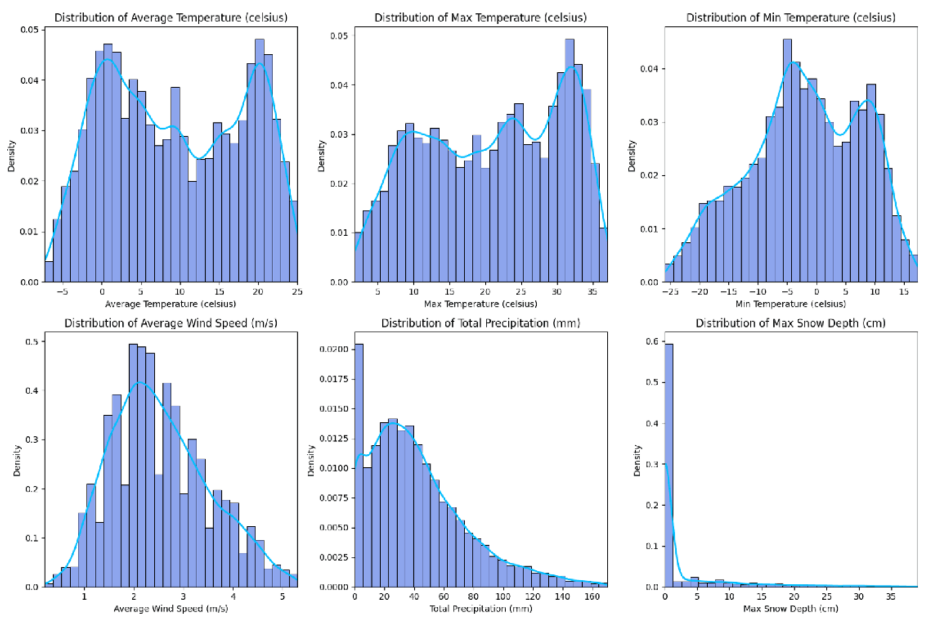

Figure 10. Distribution of data for each feature in dataset 2.

timestamps, the model can better manage the temporal elements of weather data. Weather columns indicating various conditions (such as sunny, rainy, and cloudy) are converted using LabelEncoder, which translates these categorical weather conditions into numeric values suitable for machine learning models. The Seattle dataset specifically targets daily precipitation forecasting, with the training and testing datasets reflecting this daily timeframe. It includes precipitation values for each day and aims to predict the precipitation for a future day. Each model's output is based on meteorological data from the current and previous days, predicting the amount of precipitation for a specific future day.

For the Ukrainian dataset, the method df.dropna() was employed for 'missing data handling' to ensure that only complete rows were utilized for both training and testing. The 'City' column, which contains the names of categorized cities, is encoded with a tag code that transforms the city names into different integers, allowing the model to interpret them as numeric features. The focus of precipitation forecasting in this dataset is on monthly precipitation, and the data reflects this time frame, containing monthly precipitation values for various regions in Ukraine and predicting future precipitation. The model's output is based on meteorological data from the current and previous months, with predictions made for the upcoming month.

The two datasets are created by excluding the target variable from the feature set. Numeric features in both datasets are normalized using StandardScaler to ensure uniform scaling across all numeric variables. This normalization is crucial as it prevents features with a wide range of values from disproportionately influencing the model's learning. After normalization, the data is converted into PyTorch tensors for efficient training with neural networks, with features and target variables transformed into appropriate tensor formats.

To construct the training and testing matrices, time series data processing methods are applied. The predictors include meteorological features such as temperature, humidity, and wind speed, while the target variable is precipitation. These features and target variables are extracted and combined from the original dataset, with meteorological data serving as input features and precipitation as the target variable. The feature matrix (X) comprises multiple meteorological data columns, while the target matrix (y) represents precipitation. The dataset is split into training and test sets in an 80%–20% ratio, with random partitioning to ensure that the samples in both sets have similar distributions, thereby reducing the risk of model overfitting.

4.3. Model evaluation metrics

Three main evaluation metrics are used in the models used to forecast precipitation in this paper: MAE, RMSE, and the Bias. RMSE, MAE, and Bias evaluate predictive performance from three perspectives: error magnitude, average error, and directionality, which can comprehensively reflect the strengths and weaknesses of the model. A single metric may obscure some potential issues, while using multiple metrics together can provide more detailed information. For example, RMSE may reveal that the model performs poorly in certain extreme weather conditions, while Bias can help identify the model's overall tendency (such as consistently overestimating precipitation). This aids in further targeted optimization of the model. By combining these three metrics, this study allows for a comprehensive assessment of the performance of the precipitation prediction model, not only to measure the overall accuracy of the model's predictions, but also to determine how well the model performs in dealing with the extremes and their systematic errors. These metrics measure the model's prediction performance from the following different perspectives:

1. MAE
   MAE is used to measure the average absolute difference between predicted values and actual values [21]. MAE reflects the average gap between the model's predicted values and the actual values, providing an overall level of error. This 'average error' is particularly meaningful in weather forecasting. A smaller MAE value indicates a smaller overall prediction error of the model,

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

   In equation (1), y_i represents the predicted value of the ith sample, $\hat{y}_i$ represents the predicted value of the ith sample and n is the total number of samples.
2. RMSE
   RMSE is another commonly used error measurement method that calculates the square root of the average of the squared prediction errors. Unlike MAE, RMSE is more sensitive to larger errors because the errors are squared during the calculation, making it more suitable for scenarios where there is a high demand for large error handling. RMSE is consistent with the units of the original data, allowing for a direct measurement of the absolute size of the error, which is convenient for practical applications,

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

   In equation (2), y_i and $\hat{y}_i$ represent the actual and predicted values of the ith sample, respectively, and n is the total number of samples.
3. Bias
   Bias measures the systematic deviation of the model's predictions from the actual values. It calculates the average difference between the predicted and actual values, reflecting whether the model tends to consistently overestimate or underestimate.The formula for calculating Bias is:

$$\mathrm{Bias} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right). \tag{3}$$

   In equation (3), y_i and $\hat{y}_i$ represent the actual and predicted values of the ith sample, respectively, and n is the total number of samples. A positive Bias value indicates that the model tends to overestimate, while a negative value indicates that the model tends to underestimate [21]. A value of zero suggests that the model has no significant systematic bias. In this paper, in order to make the Bias metric clearer in the comparison experiments, we use the absolute value of Bias in the experimental visualization plots. The closer the absolute value is to zero, the smaller the system bias is.

4.4. Model training

The model was built using the Python deep learning framework PyTorch and the quantum computing framework PennyLane. The loss function was set to MSE, and the optimizer was Adam. Only one experiment was conducted, with the experimental training rounds set to 100. The experiment was conducted in a CPU environment, with batch size dynamically adjusted based on memory usage. The experimental platform was configured as follows: the operating system was Microsoft Windows 11, the processor was Intel64 Family 6 Model 140 Stepping 1 GenuineIntel @ 2.42 GHz, with 16GB of RAM, and a maximum virtual memory of 21.56GB, without GPU acceleration. The training process demonstrated the model's convergence and the variation of the loss function, making it suitable for handling small- to medium-scale deep learning and quantum computing tasks on a CPU.

To evaluate the models, this experiment randomly splits the dataset, where 20% is used as the test set and 80% as the training set. The training set is used to train the hybrid model and the test set is used to evaluate the hybrid model. The loss function during training is shown in the figure, illustrating the variation of training loss and testing loss with epoch number after BO. The loss function for the hybrid model on dataset 1 and dataset 2 with MSE is shown in figure It can be seen from the figure that both the training loss and the

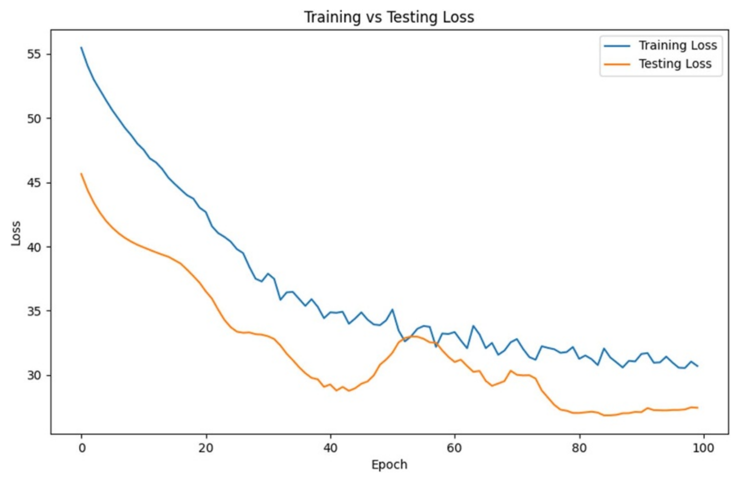

Figure 11. Loss function variation for test and training sets on dataset 1.

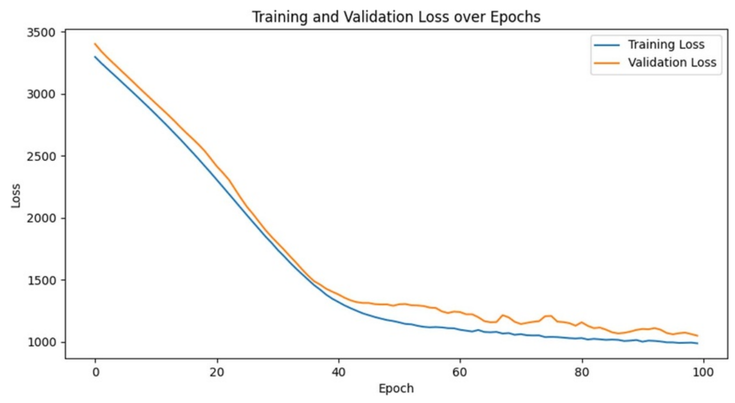

Figure 12. Loss function variation for test and training sets on dataset 2.

test loss decrease with training, indicating that the model's ability to fit the data is increasing. From figure 11, it can be seen that both the training loss and the validation loss show a rapid decrease in the early stages of training, indicating that the model learns the data distribution better in the early stages. Starting from the 20th Epoch, the decrease of the validation loss gradually slows down, but still maintains a more stable decreasing trend with the increase of training. This indicates that the model's performance on the validation set continues to be optimized without significant overfitting, and eventually the loss on the training set begins to show signs of convergence after 100 training sessions, while the loss on the test set stabilizes after the 77th training session. As can be seen in figure 12, the difference between the training loss and the test loss is obvious in the initial phase, with the training loss decreasing rapidly from around 55, while the test loss gradually stabilizes from close to 45. When the training proceeds to about 30 epochs, the decreasing trend of the test loss gradually slows down and begins to converge after 40 epochs, and finally the experiment begins to converge at about 85 epochs, these results show that the model can effectively capture the patterns in the data, and at the same time, through a reasonable training strategy, the model successfully avoids overfitting while maintaining the prediction ability, thus achieving a good prediction performance. Through this figure, we can clearly see the performance of the model during the whole training process, especially the loss of the test set tends to stabilize, which indicates that the model has good generalization ability.

In order to more intuitively evaluate the prediction effect of the model proposed in this paper, figures 13 and 14 show the comparison between the precipitation predicted by the model and the actual precipitation. As can be seen from the figure, the blue curve represents the actual precipitation and the orange curve represents the predicted value of the model. Overall, the predicted value and the actual value have a high degree of overlap, indicating that the model can better capture the trend and change of precipitation. The experimental results show that the model performs well in the precipitation prediction task. As shown in figure 13, in the first 300 samples, the comparison between the actual precipitation value and the predicted value shows a high correlation. The model can better capture the trend of medium and low precipitation.

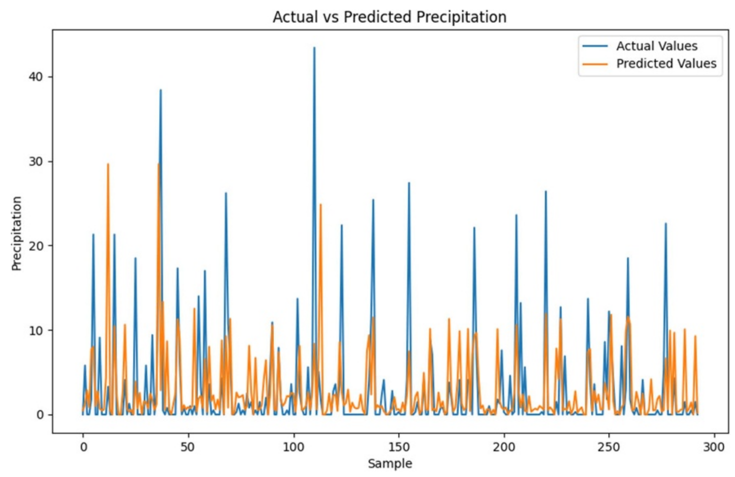

Figure 13. Visual comparison between model predicted precipitation and actual precipitation on dataset 1.

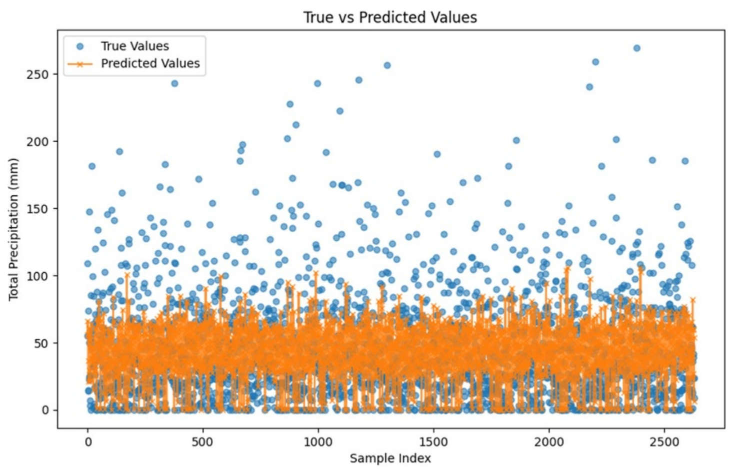

Figure 14. Visual comparison between model predicted precipitation and actual precipitation on dataset 2.

Most of the predicted values are close to the actual values, especially in smaller precipitation samples. The model prediction accuracy is high and the error is small. Figure 14 also shows the distribution of actual values and predicted values on the entire data set in the form of a scatter plot. It can be seen that most of the scatter points are distributed near the diagonal, indicating that the model has a good fitting effect on the overall precipitation distribution, which shows that the model has effective learning and prediction capabilities for the precipitation of most samples. From these two figures, we can see that under low precipitation and normal weather conditions, the model's prediction performance is relatively stable and accurate, and the difference between the actual value in blue and the predicted value in orange is small. This shows that the model has strong prediction ability under normal weather conditions (such as light rainfall or no precipitation), which provides solid support for the basic performance of the model. For low precipitation conditions, the model can track the changing trend of precipitation well, so under these more common weather conditions, the model's prediction results have high reliability. However, heavy rain and extreme weather events (such as typhoons) usually lead to extreme fluctuations in weather data. Heavy rain is usually accompanied by a large amount of precipitation in a very short period of time. This sudden weather phenomenon often has high uncertainty, especially when precipitation grows exponentially. The model may not fully capture this rapidly changing trend, resulting in a difference between the predicted value and the true value; and extreme weather events such as typhoons and hurricanes usually lead to a large amount of precipitation in local areas, and there may be large errors due to the lack of sufficient extreme weather cases in the model training data. As can be seen in the figure, the rainfall in figure 13 exceeds 20, and the rainfall in figure 14 exceeds 100. These data may be difficult for the model to handle, so a large prediction error will be generated. Therefore, in the table, it can be seen that in these abnormal weather conditions, the real value and the predicted value are very different, which also means that the model needs to be further

Table 8. The average value of the evaluation index after 15 repeated experiments on dataset 1 for each model.

Metric	TCN-QLSTM-RFR	QLSTM	LSTM	SVM	RFR	KNN
MAE-Datasetl	2.8435	2.8775	3.0288	3.0623	3.0366	3.0295
RMSE-Datasetl	5.3108	5.3241	6.2967	6.2	5.3837	6.1098
Bias-Datasetl	0.1054	0.1711	0.3393	1.87	0.2824	0.2874

Table 9. The average value of the evaluation index after 15 repeated experiments on dataset 2 for each model.

Metric	TCN-QLSTM-RFR	QLSTM	LSTM	SVM	RFR	KNN
MAE-Dataset2	25.5	28.3585	28.156	26.1791	28.1112	28.9303
RMSE-Dataset2	35.1482	37.0741	37.9946	36.9204	37.7047	38.8142
Bias-Dataset2	0.8857	3.0921	2.6146	8.12	1.1606	8.6846

optimized, especially the adaptability to extreme weather conditions. However, under certain extreme weather conditions, the model can still capture the overall trend changes. For example, although the predicted value (orange) and the actual value (blue) have a large deviation in the case of extreme precipitation, the fluctuation pattern of the two still shows a certain correlation. This shows that the model can maintain the basic prediction of the trend when encountering extreme weather. Although the error is large, it can still identify the drastic changes in precipitation and try to reflect the fluctuations in the actual situation. For extreme weather such as heavy rain or typhoons, although the predicted value sometimes deviates from the true value, the model can still better reflect the characteristics of overall climate change when identifying these extreme changes. For example, the sharp increase in precipitation is clearly visible on the chart, and the manifestation of this trend is still very important for subsequent model improvements or meteorological early warning systems. In general, the model showed good prediction ability, especially in the small and medium ranges of precipitation. Its prediction results were in good agreement with the actual values, providing reliable support for the accurate forecast of precipitation.

The computational complexity of this model is mainly reflected in two parts: first, the convolution operation of the TCN layer, whose complexity depends on the dimension of the input data, the size of the convolution kernel and the number of layers; second, the QLSTM part, which is calculated through quantum circuits, in which the complexity of quantum gate operations increases with the number of quantum bits and quantum layers. Specifically, each convolution operation of the TCN part requires a time complexity of $\mathcal{O}(\mathbb{N} \star \mathcal{M} \star \mathbb{K})$, where N is the input sequence length, M is the number of channels, and K is the convolution kernel size. The quantum circuit computational complexity of QLSTM is even higher, especially when the number of quantum bits $n -$ qlayers increases, the depth and computational time of the quantum circuit increase exponentially, so it is necessary to weigh the computational resources of quantum computation and the effect of the model. In addition, the running performance of the model depends on the data scale, the complexity of the model and the hardware environment. Due to the introduction of the quantum computing part, the computational time of the training process is longer than that of traditional neural networks, especially when multiple runs (such as 15 training iterations) are performed. Each training involves TCN feature extraction and quantum computing processes. As the number of training rounds increases, the CPU load will gradually increase. Although the quantum computing part may lead to slower training speed, by combining it with Random Forest, the model relies on the classical machine learning model for final prediction after feature extraction, which enables the model to achieve relatively stable and better performance results after multiple training cycles.

4.5. Comparison of different models

In order to verify the superiority and robustness of the proposed BO hybrid model, this paper will compare this hybrid model with RFR, SVM, KNN, LSTM, and QLSTM respectively. As shown in the figure, each model sets the same learning rate, epoch, number of hidden layers and other related parameter values, and in order to avoid the randomness of the results, the experimental results are more convincing. Each algorithm model is repeated 15 times, and the experimental batch (Epoch) is set to 100 for each repeated experiment. Tables 8 and 9 show the average values of each evaluation index of each model in 15 repeated experiments on two data sets.

It can be clearly seen from the table that after BO, the model proposed in this paper has achieved the best performance in terms of MAE, RMSE and Bias, which is significantly lower than other models. Specifically, the model proposed in this paper is better than QLSTM, RFR, KNN, SVM, and LSTM in terms of MAE, RMSE, and Bias. In terms of MAE and RMSE, the optimized hybrid model has a strong nonlinear feature

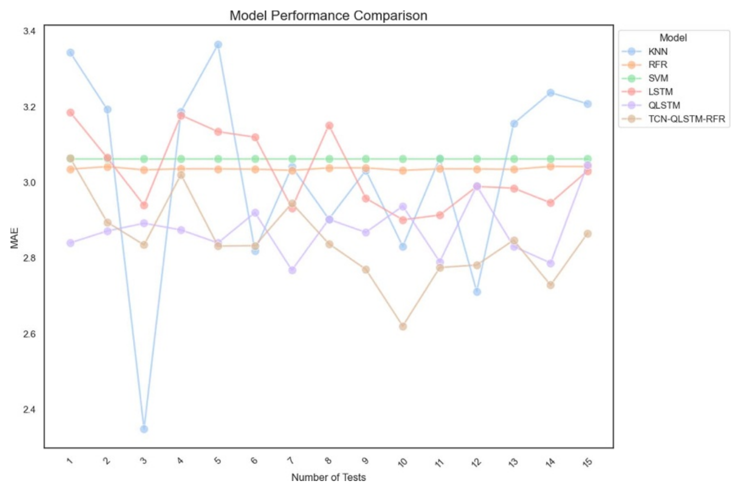

Figure 15. The change of MAE index of each algorithm model with 15 replicate experiments on dataset 1.

extraction capability, which effectively reduces the prediction error. In addition, the Bias value is also low, indicating that the prediction of the model is more accurate and stable. On the first dataset, the research model improved 1.2% MAE, 0.5% RMSE, and 38.5%Bias; on the second dataset, the research model improved 2.59% MAE, 4.8% RMSE, and 23.68% Bias. As a result, the hybrid model significantly improves the overall prediction accuracy and robustness, further verifying its advantages and applicability in precipitation prediction tasks.

Figures 15–20 show the results of the comparison between this paper's model and other models for three key evaluation metrics (MAE, RMSE, and Absolute Value of Bias) in the rainfall prediction task, respectively. The horizontal axis represents the number of rounds of training (Epoch), and the vertical axis represents the values of MAE, RMSE, and Bias, respectively. With these graphs, we are able to comprehensively evaluate the performance of different models in rainfall prediction.

Figures 15 and 16 show the MAE values of each model after 15 repeated experiments on the dataset. Specifically, in dataset 1, KNN (blue) showed large fluctuations in most tests, especially in the 1st, 2nd, 4th, 5th, and 13th tests, the MAE value was high (close to 3.5). The MAE performance of RFR (orange) was relatively stable, and it remained around 3.0 most of the time, with little fluctuation, but in the 2nd and 10th tests, the MAE value increased slightly. The MAE value of SVM (green) was relatively stable, almost similar to RFR, usually between 2.9 and 3.1, showing a certain stability. The MAE of LSTM (pink) had some fluctuations, especially when the number of tests was small (1st, 4th, and 8th), the MAE value was high, but in some tests, it performed well and the MAE value was low. The MAE of QLSTM (purple) fluctuated less, and the MAE of most tests was maintained between 2.8 and 3.0, and the performance was relatively stable. TCN-QLSTM-RFR (light brown) performed best in all tests, with its MAE value always maintained between 2.6 and 2.9, and the fluctuation was very small. This model always maintained a low error under different test times and performed the most stably, indicating that it has high prediction accuracy and consistency in this task. In Dataset 2, the MAE value of the KNN model fluctuated greatly, especially in the 1st, 5th, 9th, and 13th tests, when the MAE value was close to or exceeded 30, showing great instability. The MAE value of the RFR model was relatively stable, fluctuating around 28. Although there were slight fluctuations in some tests (such as the 12th), the overall trend remained relatively smooth. The performance of SVM was also relatively stable, with the MAE value maintained between 26 and 27 most of the time. Overall, SVM showed good stability with small fluctuations. The LSTM model showed large fluctuations in some tests (such as the 5th, 8th, and 11th), with the MAE value varying around 29, showing a certain degree of instability. The QLSTM model performed relatively stably, with MAE values between 27 and 30, without significant fluctuations, showing good reliability and consistency. The TCN-QLSTM-RFR model performed best in all tests, with its MAE value always remaining below 26, showing very low error values, and almost no fluctuations. It was the most stable of all the models, showing the highest accuracy and consistency.

Figures 17 and 18 show the RMSE values of each model after 15 repeated experiments on the data set. Specifically, in Dataset 1, KNN (blue) has large fluctuations in most data points. At the 2nd, 7th, 10th and 12th points, the RMSE values are significantly higher, and the RMSE value at the 15th data point is also high, showing unstable performance. The RMSE of the RFR (orange) model remains relatively stable at most data points, ranging from about 5 to 5.5. It does not have as large fluctuations as the KNN model, and the overall

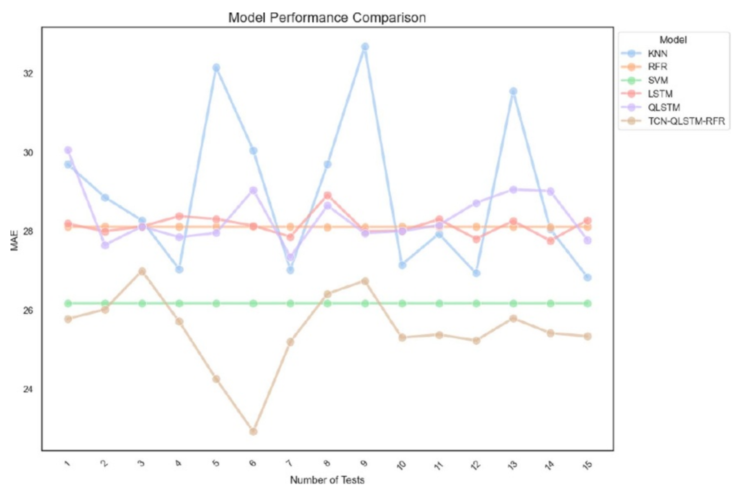

Figure 16. The change of MAE index of each algorithm model with 15 replicate experiments on dataset 2.

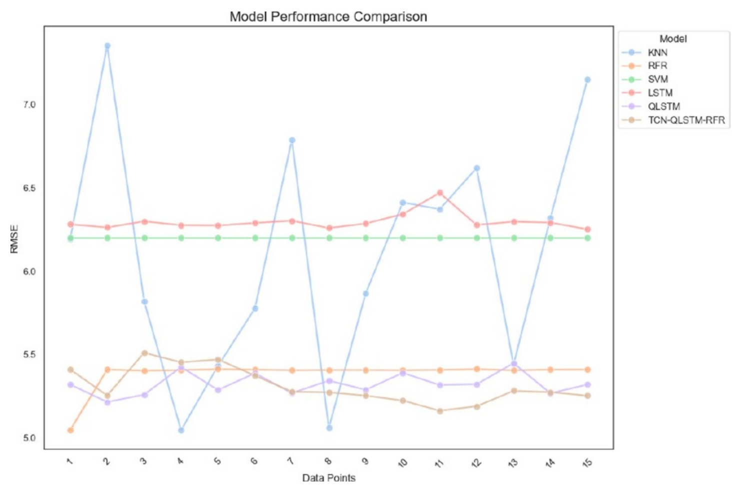

Figure 17. The change of RMSE index of each algorithm model with 15 replicate experiments on dataset 1.

trend is relatively smooth, indicating that it has better stability when processing different data points. SVM (green) Although the performance of the SVM model is relatively stable, the RMSE remains around 6 at most test points, and only slightly fluctuates when there are fewer data points, but the data value is larger. The RMSE value of LSTM (pink) has higher fluctuations when there are fewer data points, showing better performance overall, but the overall RMSE is high and basically exceeds 6. The RMSE value of QLSTM (purple) is stable, remaining around 5.5, with small fluctuations, showing its good stability and reliability. TCN-QLSTM-RFR (light brown) has the best performance among all models, with RMSE values between 5.0 and 5.5 for all data points. Its RMSE value is much lower than that of other models and is very stable with almost no fluctuation, indicating that it has the strongest accuracy and consistency when processing different data points. In Dataset 2, the KNN model performed moderately in most tests, with RMSE values fluctuating around 38, but it exceeded 40 in the 8th and 9th tests, which is a poor performance. The RFR model performed relatively stably, with RMSE roughly maintained at around 37.5, but there were some fluctuations, especially in the first few experiments. The RMSE performance of SVM was relatively stable, maintained between 36 and 37, and the overall trend was relatively stable. The RMSE value of LSTM fluctuated greatly, performing poorly in some tests, but showed a low RMSE in the 4th test. The RMSE of QLSTM fluctuated around 37, which was close to LSTM overall, but with less fluctuation. The TCN-QLSTM-RFR combined model performed best in the test, with a significantly lower RMSE value than other models. Especially when the number of tests was small, its RMSE value was the lowest, mostly maintained between 34 and 36, with little fluctuation.

Figures 19 and 20 show the absolute value of Bias after 15 repeated experiments for each model. Specifically, the bias performance of KNN (blue) in Dataset 1 fluctuates greatly, especially in the 2nd, 6th,

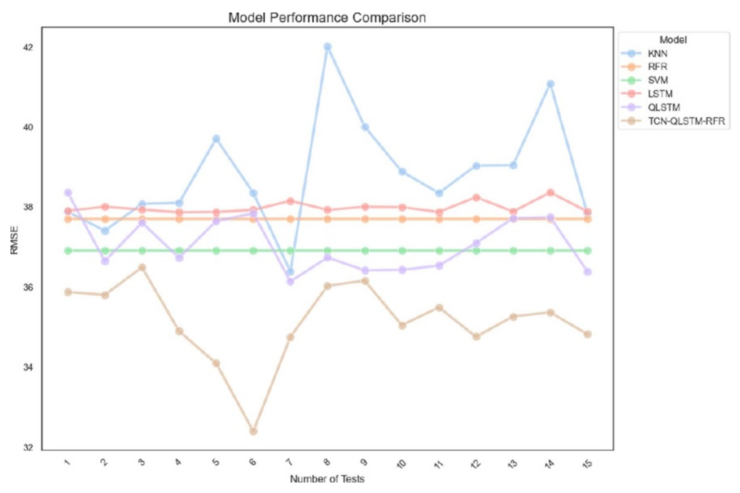

Figure 18. The change of RMSE index of each algorithm model with 15 replicate experiments on dataset 2.

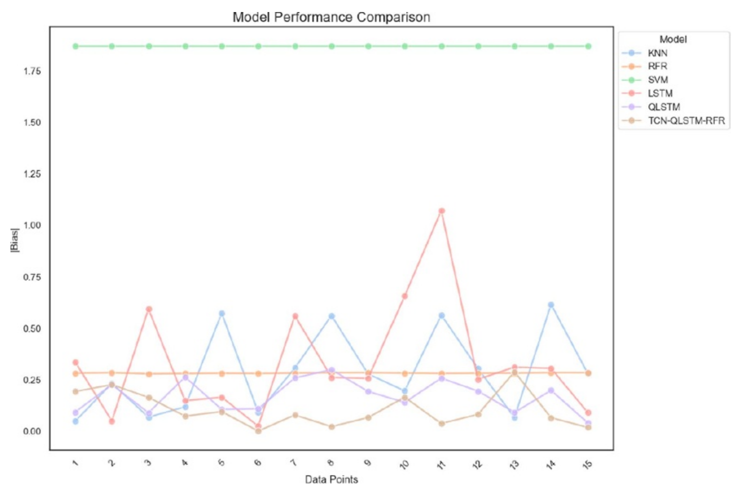

Figure 19. The index change of the absolute value of Bias of each algorithm model with 15 replicates of experiments on dataset 1.

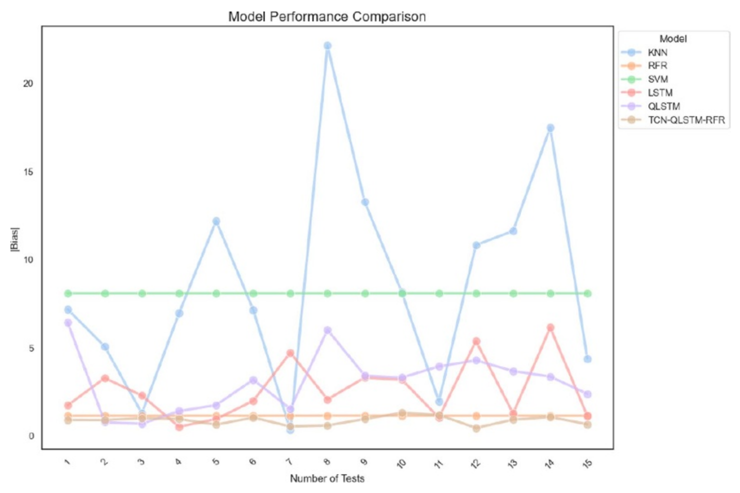

Figure 20. The index change of the absolute value of Bias of each algorithm model with 15 replicates of experiments on dataset 2.

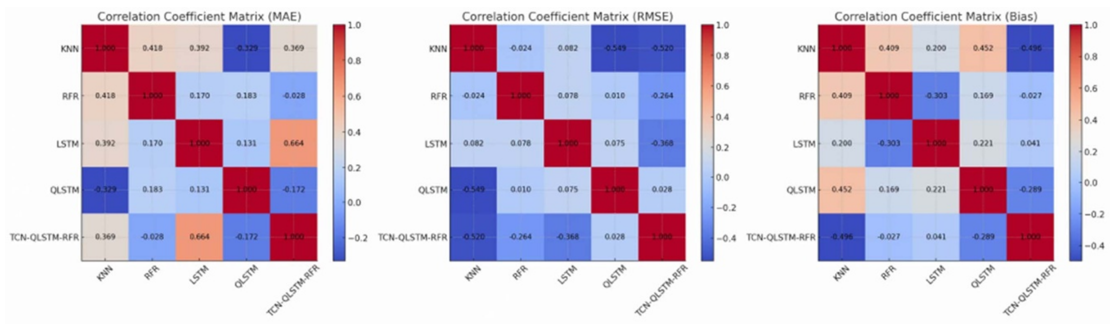

Figure 21. Correlation coefficients of prediction results of each model on dataset 1.

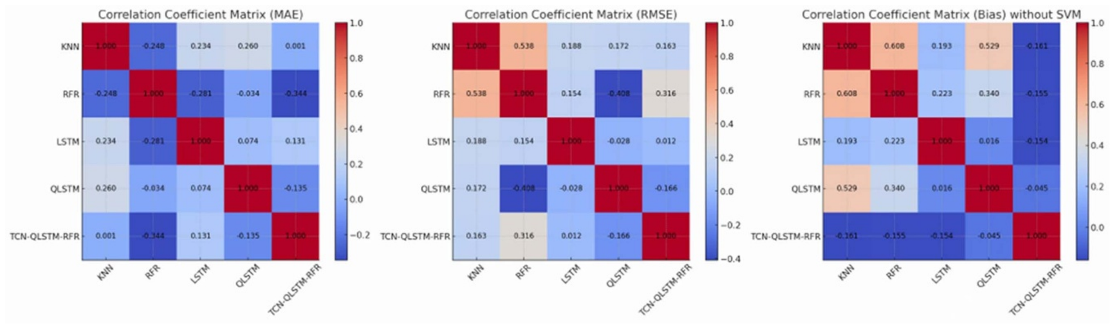

Figure 22. Correlation coefficients of prediction results of each model on dataset 2.

8th, 13th, and 14th tests, where the bias is large, showing a high systematic error. At some data points, KNN has a small bias, but overall it is unstable and has a large error. RFR (orange) has a relatively low and stable bias value, usually between 0.25 and 0.53, and performs relatively smoothly with a small error. At some data points (such as the 7th test), its bias increases slightly, but overall it performs well. SVM (green) has an almost completely stable bias, always maintaining at 1.87, without significant fluctuations, but the bias value is large. LSTM (pink): The bias fluctuates significantly when there are fewer data points, especially in the 2nd, 7th, and 10th tests, where the bias value increases significantly. Afterwards, as the number of data points increases, the bias of LSTM gradually stabilizes, but occasionally there are large fluctuations. QLSTM (purple) performs well in bias control overall, but there are still some fluctuations. TCN-QLSTM-RFR (light brown) showed the best bias control in all tests, with the bias value always maintained between 0 and 0.25 and very stable. There was almost no fluctuation, showing very low systematic error and high stability. In Dataset 2, the bias of the KNN model fluctuated significantly in the 1st, 5th, 8th, 13th, and 14th tests, especially in the 8th test, the bias value increased significantly. The bias value of RFR was relatively stable, usually maintained between 0 and 1.5, with small fluctuations, showing some systematic error but overall relatively stable. The bias value of SVM was very stable, but the data was high despite the stability of the bias value. The bias of LSTM fluctuated greatly in the 2nd, 7th, 12th, and 14th tests, especially in the 12th and 14th tests, the bias value increased significantly, close to 6. The bias value of QLSTM remained low in multiple tests, usually maintained between 0 and 4, and was relatively stable. It performed relatively well, showing a small systematic error and small fluctuations. The TCN-QLSTM-RFR model maintained a very stable bias value between 0 and 1 in all tests. It performed best among all the models, with almost no fluctuation in bias control, always maintaining a low system error, and showing very high accuracy and stability.

This study conducted prediction experiments on multiple models on the Seattle dataset and the Ukraine dataset, and evaluated the linear relationship between the prediction results of these models by calculating the correlation coefficient. The visualization of the correlation coefficient reveals the consistency and difference of different models on the two datasets. As shown in figures 21 and 22. Because the value of the SVM evaluation index is the same every time, its variance is 0, and it is not included when calculating the correlation coefficient. These correlation coefficients provide a quantitative basis for us to have a deeper understanding of the similarities and differences between the prediction results of different models. These analyses help to better evaluate the prediction ability of each model, and also show that the prediction performance of the model in this study is good.

The results of these comparative experiments clearly show that the model proposed in this paper, after BO, is used for feature extraction in combination with TCN, and the extracted features are fed into a random forest regression model for final prediction. In 15 experiments, the model significantly outperforms QLSTM, LSTM and a single random forest model in three key metrics including MAE, RMSE and Bias. In all the tests, the performance of TCN-QLSTM-RFR is always stable with minimum fluctuation range, especially in extreme data distribution scenarios. The comprehensive experimental results show that TCN-QLSTM-RFR is the best performing model in this study. In the three core metrics (Bias, MAE, and RMSE), its accuracy, stability, and robustness are significantly better than those of the other compared models, providing a powerful solution for complex weather prediction tasks.The model in this paper is able to capture the complex nonlinear relationships in time series data more effectively and has higher prediction accuracy and stability in rainfall prediction tasks. In addition, the hybrid model combines the excellent performance of TCN in feature extraction and the deep feature learning capability of quantum computing to further enhance the performance of the model. By comparing the performance of traditional machine learning models, the structure combining quantum computing and deep learning proposed in this paper significantly outperforms these traditional models in terms of prediction performance, demonstrating stronger generalization ability and prediction accuracy. This also indicates that the hybrid model proposed in this paper has significant application potential and practical value in the processing and prediction of complex meteorological data.

5. Conclusion and future prospects

In this paper, we propose a hybrid model for rainfall prediction combining TCN, QLSTM, and RFR. This study provides a new approach to the field of weather prediction and demonstrates the potential of combining quantum computing and classical machine learning models that are designed to take full advantage of all three: TCN has a powerful time-series feature extraction capability to effectively capture long-range dependencies in the data; QLSTM combines the unique advantages of quantum computing, which is further enhanced by quantum convolutional layers LSTM's ability to model complex nonlinear patterns; and RFR, as a classical integrated learning algorithm, not only can effectively handle high-dimensional data, but also has strong overfitting resistance, which makes the model perform well in dealing with complex weather data. BO, on the other hand, provides key support for the hyperparameter tuning of the model, automatically finding the optimal parameter combinations, which significantly improves the prediction performance of the model.The experimental results clearly show that the hybrid model combining quantum computing and deep learning exhibits excellent performance in the rainfall prediction task. By introducing BO, the hyperparameters of the TCN and QLSTM models are effectively tuned, which, combined with the powerful integrated learning capability of RFR, enables my model to better adapt to different rainfall patterns, thus significantly improving the accuracy and robustness of the prediction. In the comparison experiments with a variety of traditional machine learning models (e.g. Random Forest, SVM, KNN) and deep learning models (e.g. LSTM, QLSTM), the hybrid model proposed in this paper has the lowest error values in the three key evaluation metrics of Bias, MAE, and RMSE, which demonstrates its advantages in dealing with complex time series data. Specifically, the model is able to capture rainfall volatility well, accurately predict rainfall peaks and trends, and exhibit good stability and convergence during the training process. These results validate the potential and advantages of combining quantum computing with deep learning and BO in complex meteorological data processing, and the model performs well in handling complex time series data, better adapts to different rainfall patterns, and improves the accuracy and robustness of prediction.

The innovation of this paper is to combine quantum computing with deep learning and ensemble learning, and automatically tune the model through BO, which significantly improves the performance of the model in rainfall prediction tasks. TCN can efficiently capture long-term dependencies in data, so as to better extract time series features. Its unique dilated convolution structure expands the perception domain, enabling the model to learn a wider range of time information, so as to better understand the dynamic changes of rainfall data. QLSTM combines the advantages of quantum computing and classical LSTM, and can better learn complex nonlinear relationships and enhance the prediction ability of the model. The introduction of the quantum computing module enables the model to utilize the parallelism and complexity of quantum states to more effectively process the nonlinear features in rainfall data. As a classic ensemble learning algorithm, RFR can effectively process high-dimensional data and has strong anti-overfitting ability. Through the combination of multiple decision trees, it can effectively reduce the sensitivity of the model to noise data and improve the generalization ability of the model; the BOA can automatically find the best parameter combination for the model, thereby improving the prediction performance of the model; its surrogate model based on GP can effectively balance exploration and utilization and find the best parameter configuration of the model. However, there are also some limitations. For example, the QLSTM model relies on quantum computing resources, and quantum computing technology is still in the development stage. The available quantum computing resources are limited, which may limit the scale and application scope of the model. With the continuous development of quantum computing technology, the application of the QLSTM model will be broader. The QLSTM model integrates many different models, and the model structure is more complex, which may lead to an increase in the training time and computing cost of the model. The model structure is relatively complex, which may lead to an increase in the model training time and computing cost. The model is not sensitive enough to abnormal weather data, and the prediction of extreme weather is not accurate enough. It is necessary to increase data training to improve the model performance. Future research can consider optimizing the model structure, reducing the complexity of the model, and improving the efficiency of model training. Future research can continue to explore the application of quantum computing in the prediction of other meteorological variables, further optimize the quantum circuit structure, and improve its adaptability and performance in more complex tasks. In addition, in extreme precipitation prediction, when comparing the true value and the predicted value, although it has advantages over other models, there are also some shortcomings, mainly underestimating the actual precipitation in the future, which needs further improvement. Further improvement is needed in the future. In summary, this paper demonstrates the great potential of combining quantum computing with deep learning, providing an innovative and efficient solution for the field of weather forecasting. This model not only provides new ideas for the processing of complex time series data in theory, but also lays a solid foundation for the development of future weather forecasting applications.

Data availability statement

The data cannot be made publicly available upon publication because they are not available in a format that is sufficiently accessible or reusable by other researchers. The data that support the findings of this study are available upon reasonable request from the authors.

Ethical approval and consent to participate

Not applicable. This study does not involve human participants or animals.

Consent for publication

All authors consent to the publication of this manuscript.

Author contributions statement

Huanxin Ding conducted an experiment and wrote a manuscript.Yumin Dong designed the experiment and provided the necessary financial support. All authors reviewed the manuscript.

Conflict of interest

The author(s) declare no competing interests.

Funding

This work was supported by National Natural Science Foundation of China (No.61772295), Key Projects of Chongqing Natural Science Foundation Innovation Development Joint Fund (CSTB2023NSCQ-LZX0139), Open Fund of Advanced Cryptography and System Security Key Laboratory of Sichuan Province (Grant No. SKLACSS–202208) and Technology Research Program of Chongqing Municipal Education Commission (Grant no. KJZD-M202000501).

ORCID iD

Yumin Dong https://orcid.org/0000-0002-7890-4427

References

[1] Watson A, Miller J, Fleischer M and De Clercq W 2018 Estimation of groundwater recharge via percolation outputs from a rainfall/runoff model for the Verlorenvlei estuarine system, west coast, South africa *J. Hydrol.* **558** 238–54

[2] Li W, Zhang P, Dong H, Jia Y and Cao W 2022 RSDF-AM-LSTM: regional scale division rainfall forecasting using attention and LSTM *ACM/IMS Trans. on Data Science (TDS)* **2** 1–27

[3] Salehin I, Talha I M, Hasan M M, Dip S T and Nessa N 2021 An artificial intelligence based rainfall prediction using LSTM and neural network *2020 IEEE Int. Women in Engineering (WIE) Conf. on Electrical and Computer Engineering (WIECON-ECE)* (https://doi.org/10.1109/WIECON-ECE52138.2020.9398022)

[4] Roudier P, Andersson J, Donnelly C, Feyen L, Greuell W and Ludwig F 2016 Projections of future floods and hydrological droughts in europe under a +2 °C global warming *Clim. Change* **135** 341–55

[5] Le V M, Pham B T, Le T T, Ly H B and Le L M 2020 Daily rainfall prediction using nonlinear autoregressive neural network *Micro-Electronics and Telecommunication Engineering: Proc. 3rd ICMETE 2019* (Springer) pp 213–21

[6] Zhou Y, Dong Z and Bao X 2024 A ship trajectory prediction method based on an optuna–BILSTM model *Appl. Sci.* **14** 3719

[7] Beheshti Z, Firouzi M, Shamsuddin S M, Zibarzani M and Yusop Z 2016 A new rainfall forecasting model using the CAPSO algorithm and an artificial neural network *Neural Comput. Appl.* **27** 2551–65

[8] Peng Y, Xu W, Wang P and You F 2015 Flood forecasting coupled with tigge ensemble precipitation forecasts *J. Tianjin Univ. (Sci. Technol.)* **48** 177–84

[9] Li Q, Du X, Ni P, Han Q, Xu K and Yuan Z 2024 Efficient bayesian inference for finite element model updating with surrogate modeling techniques *J. Civ. Struct. Health Monit.* **14** 997–1015

[10] Zhang P, Jia Y, Zhang L, Gao J and Leung H 2018 A deep belief network based precipitation forecast approach using multiple environmental factors *Intell. Data Anal.* **22** 843–66

[11] Poornima S and Pushpalatha M 2019 Prediction of rainfall using intensified LSTM based recurrent neural network with weighted linear units *Atmosphere* **10** 668 2019

[12] Mandal T and Jothiprakash V 2012 Short-term rainfall prediction using ann and mt techniques *ISH J. Hydraul. Eng.* **18** 20–6

[13] Liu Y, Gong C, Yang L and Chen Y 2020 DSTP-RNN: a dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction *Expert Syst. Appl.* **143** 113082

[14] Gomes J and Velho L 2013 *Image Processing for Computer Graphics* (Springer)

[15] Fan P, Wang D, Wang W, Zhang X and Sun Y 2024 A novel multi-energy load forecasting method based on building flexibility feature recognition technology and multi-task learning model integrating LSTM *Energy* **308** 132976

[16] Samad A *et al* 2020 An approach for rainfall prediction using long short term memory neural network *2020 IEEE 5th Int. Conf. on Computing Communication and Automation (ICCCA)* (IEEE) pp 190–5

[17] Sawale G J and Gupta S R 2013 Use of artificial neural network in data mining for weather forecasting *Int. J. Comput. Sci. Appl.* **6** 383–7 (available at: www.researchpublications.org/IJCSA/NCAICN-13/244.pdf)

[18] Kang J, Wang H, Yuan F, Wang Z, Huang J and Qiu T 2020 Prediction of precipitation based on recurrent neural networks in Jingdezhen, Jiangxi province, China *Atmosphere* **11** 246

[19] Zhang X, Wu X, He S and Zhao D 2021 Precipitation forecast based on CEEMD–LSTM coupled model *Water Supply* **21** 4641–57

[20] Kratzert F, Klotz D, Brenner C, Schulz K and Herrnegger M 2018 Rainfall–runoff modelling using long short-term memory (LSTM) networks *Hydrol. Earth Syst. Sci.* **22** 6005–22

[21] Dai L 2024 Performance analysis of deep learning-based electric load forecasting model with particle swarm optimization *Heliyon* **10** e35273

[22] Wu J, Long J and Liu M 2015 Evolving rbf neural networks for rainfall prediction using hybrid particle swarm optimization and genetic algorithm *Neurocomputing* **148** 136–42

[23] Swapna M and Sudhakar N 2018 A hybrid model for rainfall prediction using both parametrized and time series models *Int. J. Pure Appl. Math.* **119** 1549–56 (available at: www.researchgate.net/profile/Swapna-Medikonda-2/publication/362278778_A_HYBRID_MODEL_FOR_RAINFALL_PREDICTION_USING_BOTH_PARAMETRIZED_AND_TIME_SERIES_MODELS/links/62e0db3a3c0ea878875fff6f/A-HYBRID-MODEL-FOR-RAINFALL-PREDICTION-USING-BOTH-PARAMETRIZED-AND-TIME-SERIES-MODELS.pdf)

[24] Shi X, Chen Z, Wang H, Yeung D Y, Wong W K and Woo W C 2015 Convolutional lstm network: a machine learning approach for precipitation nowcasting *Advances in Neural Information Processing Systems* vol 28 (available at: https://proceedings.neurips.cc/paper/2015/hash/07563a3fe3bbe7e3ba84431ad9d055af-Abstract.html)

[25] Wang Y V, Kim S H, Lyu G, Lee C L, Ryu S, Lee G, Min K H and Kafatos M C 2024 Nowcasting heavy rainfall with convolutional long short-term memory networks: a pixelwise modeling approach *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **17** 8424–33

[26] LeCun Y, Bengio Y and Hinton G 2015 Deep learning *Nature* **521** 436–44

[27] Salehin I, Talha I M, Hasan M M, Dip S T, Saifuzzaman M and Moon N N 2020 An artificial intelligence based rainfall prediction using lstm and neural network *2020 IEEE Int. Women in Engineering (WIE) Conf. on Electrical and Computer Engineering (WIECON-ECE)* (IEEE) pp 5–8

[28] Snoek J, Larochelle H and Adams R P 2012 Practical Bayesian optimization of machine learning algorithms *Advances in Neural Information Processing Systems* vol 25 (available at: https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html)

[29] Gong L H, Pei J J, Zhang T F and Zhou N R 2024 Quantum convolutional neural network based on variational quantum circuits *Opt. Commun.* **550** 129993

[30] Fan J, Zhang K, Huang Y, Zhu Y and Chen B 2021 Parallel spatio-temporal attention-based tcn for multivariate time series prediction *Neural Comput. Appl.* **35** 13109–18

[31] Wang X, Wang X and Zhang S 2022 Adverse drug reaction detection from social media based on quantum Bi-LSTM with attention *IEEE Access* **11** 16194–202

[32] Sordoni A, Nie J Y and Bengio Y 2013 Modeling term dependencies with quantum language models for ir *Proc. 36th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval* pp 653–62

[33] Jia Z A, Yi B, Zhai R, Wu Y C, Guo G C and Guo G P 2019 Quantum neural network states: a brief review of methods and applications *Adv. Quantum Technol.* **2** 1800077

[34] Cong I, Choi S and Lukin M D 2019 Quantum convolutional neural networks *Nat. Phys.* **15** 1273–8
[35] Cocos A, Fiks A G and Masino A J 2017 Deep learning for pharmacovigilance: recurrent neural network architectures for labeling adverse drug reactions in twitter posts *J. Am. Med. Inf. Assoc.* **24** 813–21
[36] Yu Y, Hu G, Liu C, Xiong J and Wu Z 2023 Prediction of solar irradiance one hour ahead based on quantum long short-term memory network *IEEE Trans. Quantum Eng.* **4** 1–15
[37] Liashchynskyi P and Liashchynskyi P 2019 Grid search, random search, genetic algorithm: a big comparison for nas (arXiv:1912.06059)
[38] Wu J, Chen X Y, Zhang H, Xiong L D, Lei H and Deng S H 2019 Hyperparameter optimization for machine learning models based on Bayesian optimization *J. Electron. Sci. Technol.* **17** 26–40