

# Salinity Prediction Of Raw Water Using Temporal Kolmogorov-Arnold Network - TKAN

Nguyen Gia Bao, Pham Nguyen Anh  
22520109, 22520069  
University of Information Technology,  
Ho Chi Minh City, Viet Nam

Lecturer: Duong Viet Hang  
hangdv@uit.edu.vn  
University of Information Technology,  
Ho Chi Minh City, Viet Nam

## Abstract

*Accurate prediction of raw water salinity is vital for managing water supply and environmental health. This study focuses on forecasting salinity levels in the Saigon River, where salinity fluctuates due to tides, rainfall, and upstream discharge.*

*We apply the Temporal Kolmogorov-Arnold Network (TKAN) to capture complex temporal dependencies in historical salinity data. The model was trained on real measurements from the Saigon River and evaluated against LSTM and TCN baselines.*

*Experimental results show that TKAN provides better in terms of MAE and MAPE. These findings highlight TKAN's potential for reliable salinity forecasting in riverine systems.*

## 1 Introduction

Salinity in raw water is a key parameter in water quality management, especially in regions affected by tidal intrusion and seasonal changes. In southern Vietnam, the Saigon River is a major water source for domestic and agricultural use. However, salinity levels in the river fluctuate due to a combination of natural and human factors, including tidal forces, rainfall patterns, and upstream dam operations. These variations can impact freshwater availability, irrigation systems, and treatment plant operations.

Accurate salinity prediction helps water authorities plan effectively, avoid saline intrusion during critical periods, and ensure safe water supply. Traditional statistical models often fail to capture the complex, nonlinear, and dynamic nature of salinity patterns, especially under climate and hydrological variability.

In this study, we explore the use of the Temporal Kolmogorov-Arnold Network (TKAN)

— a recent deep learning architecture tailored for time series forecasting — to predict salinity levels in the Saigon River.

We trained and tested the model using real-world salinity measurements from the Saigon River and compared it to conventional models such as Long Short-Term Memory (LSTM) networks and Temporal Convolutional Networks (TCN),... the results show that TKAN consistently outperforms these baselines, offering improved prediction accuracy and better generalization over time.

This work demonstrates the potential of modern deep learning techniques for environmental monitoring and supports more effective water resource planning in urban river systems like the Saigon River.

## 2 Related Work

Time series forecasting for water quality, particularly salinity prediction, has been explored using both statistical and deep learning models. Traditional approaches such as ARIMA are limited by their assumption of stationarity and linear dependencies. To address these limitations, recent studies have adopted machine learning models including Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRU), and Temporal Convolutional Networks (TCN).

A recent study by Huynh and Do (2024) applied these models to a 6-year dataset from the Saigon River, evaluating model performance across different window sizes and input variable combinations. Data preprocessing involved outlier removal and linear interpolation.

Their work demonstrated the effectiveness of deep learning models, particularly CNN and LSTM, in short-term salinity prediction.

**Long Short-Term Memory (LSTM)** networks are one of the most widely used models for sequential data. They are a type of recurrent neural network (RNN) that address the vanishing gradient problem through a gating mechanism, allowing them to maintain long-term dependencies. LSTM models have been applied successfully in various forecasting domains, including hydrology and water quality prediction. However, their performance tends to degrade in longer forecast horizons due to limited capacity in modeling complex temporal patterns over extended sequences.

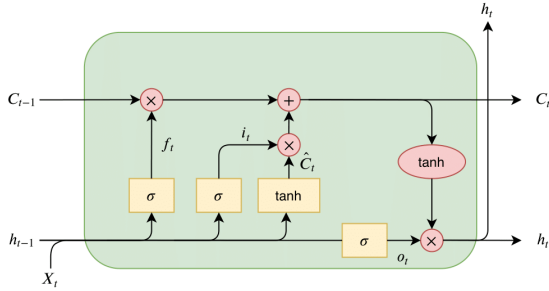


Figure 1: LSTM architecture

To address some of LSTM’s limitations, recent research has explored architectures with stronger representational capabilities. One such model is the **Kolmogorov-Arnold Network (KAN)**, which is inspired by the Kolmogorov-Arnold representation theorem. Unlike traditional MLPs and RNNs that use fixed activation functions on nodes, KANs place *learnable* activation functions on edges, implemented via splines. This design allows KANs to learn both global compositional structures and local nonlinearities, leading to superior accuracy and interpretability, especially in scientific domains. Empirical results have shown that KANs can outperform MLPs on function approximation tasks and achieve favorable scaling laws.

Building upon KANs, the **Temporal Kolmogorov-Arnold Network (TKAN)** extends the architecture for sequence modeling by incorporating recurrent mechanisms tai-

lored to time series data. TKAN integrates temporal recurrence and activation structures, making it suitable for capturing both short-term dynamics and long-term patterns. Unlike standard RNNs, TKAN achieves better generalization with fewer parameters and has shown promising results in domains such as air quality prediction and now salinity forecasting.

Our approach builds on the work of Huynh and Do (2024). However, we enhance their pipeline by incorporating **forward/backward filling** for missing data, testing various input configurations (window size, forecast horizon, and number of variables), and ultimately applying TKAN as a more powerful alternative to LSTM. Our results demonstrate that TKAN consistently outperforms both LSTM and baseline models across all evaluated configurations. Moreover, while previous works mainly used linear interpolation for missing data, we enhanced preprocessing using forward and backward fill, which better suits real-time systems and avoids interpolation biases.

Our approach is distinguished by its combination of advanced data imputation and the use of a more powerful forecasting model (TKAN), which has shown superior performance over LSTM, GRU, ... in multi-step forecasting tasks. This positions TKAN as a promising model for long-horizon, high-variability environmental time series such as salinity in the Saigon River.

### 3 Data

The dataset used in this project originates from the **Hoa Phu pumping station** on the **Saigon River**, a vital raw water intake point for the **Tan Hiep Water Treatment Plant** in Ho Chi Minh City, Vietnam. This location plays a key role in the city’s water supply system, and the data reflects real-world conditions in a tropical, urban river that is influenced by tidal patterns, rainfall, and upstream reservoir operations.

The dataset spans a period of **six years**, from **January 1, 2017 to December 31, 2022**, and consists of **2,191 daily records**. Each record includes **50 variables** representing various physical, chemical, and operational water quality indicators. These variables include

measurements such as *pH*, *turbidity*, *chlorine concentration*, *iron*, *ammonia*, *aluminum*, and *salinity*, collected both at the river intake and at multiple points throughout the treatment process.

For this study, we focused on a subset of features that are most relevant to the task of **salinity prediction**, especially the salinity level of the Saigon River (“*Man\_song\_saigon*”), along with *pH* and *electrical conductivity*, due to their known correlation with salinity variations. These variables were selected based on domain knowledge and practical significance for modeling and monitoring purposes.

The dataset presents typical real-world challenges such as seasonal trends, sensor noise, and missing values. However, its richness and temporal continuity offer a strong basis for applying and evaluating deep learning models. The long observation period and daily frequency enable the modeling of both short-term fluctuations and long-term seasonal trends.

Overall, the dataset provides a realistic and comprehensive foundation for developing predictive models using advanced time series techniques, particularly the **Temporal Kolmogorov-Arnold Network (TKAN)**, to forecast salinity levels under complex and dynamic environmental conditions.

## 4 Methods

### 4.1 Kolmogorov-Arnold Theorem

The Kolmogorov–Arnold representation theorem is a profound result in the theory of multivariate continuous functions. It was first proven by Andrey Kolmogorov in 1957 and later refined by his student Vladimir Arnold in 1958. The theorem states that any continuous function of multiple variables can be represented exactly using only a finite number of continuous functions of a single variable and the operation of addition.

#### 4.1.1 Mathematical Statement

Let  $f : [0, 1]^n \rightarrow \mathbb{R}$  be any continuous function defined on the  $n$ -dimensional unit cube. Then, there exist  $2n + 1$  continuous functions  $\Phi_q : \mathbb{R} \rightarrow \mathbb{R}$  and  $n$  families of continuous

functions  $\varphi_{q,p} : [0, 1] \rightarrow \mathbb{R}$  such that:

$$f(x_1, x_2, \dots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \varphi_{q,p}(x_p) \right). \quad (1)$$

Here, the inner functions  $\varphi_{q,p}$  depend only on the individual variables  $x_p$ , and the outer functions  $\Phi_q$  are applied to the resulting sums. The functions  $\Phi_q$  and  $\varphi_{q,p}$  are independent of the function  $f$  itself — they are universal in a certain sense — and the representation depends only on  $f$  through the specific choice of these functions.

#### 4.1.2 Interpretation

This representation shows that every multivariate continuous function can be expressed as a superposition of univariate functions and addition. In other words, multivariate complexity can be reduced to a finite structure involving only one-dimensional operations. The only multivariate operation used is the summation, which is associative and commutative.

This result is striking because, at first glance, it seems impossible that arbitrary multivariate behavior can be encoded through such a restricted structure. However, the power of the theorem lies in the fact that the number of terms,  $2n + 1$ , is independent of the form of  $f$ , and the dimensionality is fully handled through the structure of the inner univariate functions  $\varphi_{q,p}$ .

### 4.2 Formal Properties

Some important mathematical properties of the Kolmogorov–Arnold representation include:

- **Universality:** The representation is valid for all continuous functions on compact domains such as  $[0, 1]^n$ .
- **Constructiveness:** While the theorem guarantees existence, the explicit construction of the functions  $\Phi_q$  and  $\varphi_{q,p}$  is highly nontrivial and often not practical.
- **Non-smoothness:** The functions  $\varphi_{q,p}$  and  $\Phi_q$  constructed in the original proofs are typically non-smooth, possibly even

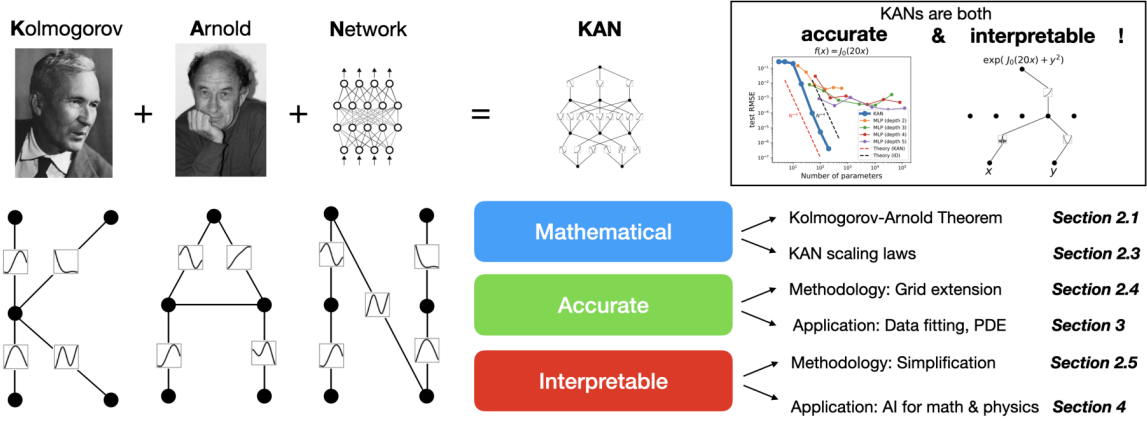


Figure 2: KAN's overview

nowhere differentiable. This limits the usefulness of the representation in applications requiring smooth approximations.

- **Dimensional reduction:** The theorem effectively reduces an  $n$ -dimensional problem to a set of one-dimensional problems, hence bypassing the so-called "curse of dimensionality" in a purely theoretical sense.

#### 4.2.1 Limitations and Challenges

Despite its elegance, the Kolmogorov–Arnold theorem has practical limitations:

- The resulting univariate functions may be extremely irregular or fractal-like, making them difficult to approximate numerically.
- There is no general method to compute or approximate the functions  $\Phi_q$  and  $\varphi_{q,p}$  for a given target function  $f$ .
- The representation is not unique; many different sets of functions can satisfy the theorem for the same  $f$ .

For these reasons, while the theorem is of major theoretical importance in functional analysis and approximation theory, it has historically had limited application in numerical mathematics and machine learning.

#### 4.2.2 Conclusion

The Kolmogorov–Arnold representation theorem is a foundational result in mathematics that establishes the sufficiency of univariate continuous functions and addition in representing arbitrary continuous multivariate functions. Its elegance lies in its reductionist nature: despite the infinite complexity of multivariate functions, they can all be captured by simple univariate components. While practical limitations exist, the theorem remains a central theoretical insight into the structure of continuous functions.

#### 4.3 Kolmogorov–Arnold Networks (KANs)

Kolmogorov–Arnold Networks (KANs) are a novel class of neural network architectures directly inspired by the Kolmogorov–Arnold representation theorem. Unlike traditional feed-forward neural networks such as Multi-Layer Perceptrons (MLPs), KANs replace linear weights and fixed nonlinear activations with learnable univariate functions placed on the edges of the computational graph. This structural innovation enables more interpretable and efficient function approximators, especially for problems exhibiting compositional structure.

##### 4.3.1 Architecture of KANs

Suppose we have a supervised learning task consisting of input-output pairs  $\{x_i, y_i\}$ , where we want to find a function  $f$  such that  $y_i \approx$

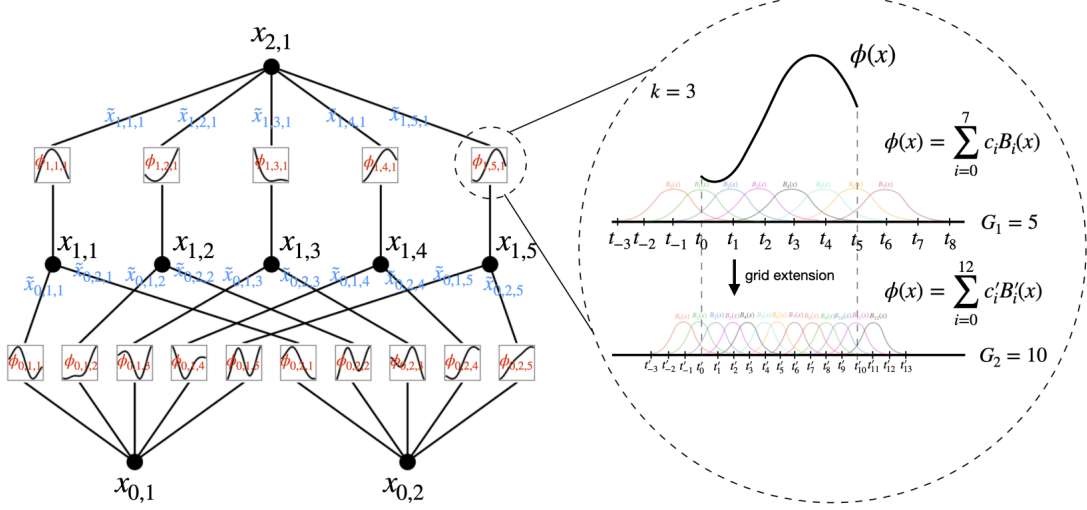


Figure 3: Left: Notations of activations that flow through the network. Right: an activation function is parameterized as a B-spline, which allows switching between coarse-grained and fine-grained grids.

$f(x_i)$  for all data points. Equation (2.1) implies that we are done if we can find appropriate univariate functions  $\phi_{q,p}$  and  $\Phi_q$ . This inspires us to design a neural network which explicitly parametrizes Equation (2.1). Since all functions to be learned are univariate functions, we can parametrize each 1D function as a B-spline curve, with learnable coefficients of local B-spline basis functions (see 3, right).

Now we have a prototype of KAN (Kolmogorov–Arnold Network), whose computation graph is exactly specified by Equation (2.1) and illustrated in Figure 0.1(b) (with the input dimension  $n = 2$ ). It appears as a two-layer neural network with activation functions placed on edges instead of nodes (simple summation is performed on nodes), and with width  $2n + 1$  in the middle layer.

As mentioned, such a network is known to be too simple to approximate any function arbitrarily well in practice with smooth splines! We therefore generalize our KAN to be wider and deeper.

It is not immediately clear how to make KANs deeper, since Kolmogorov–Arnold representations correspond to two-layer KANs. To the best of our knowledge, there is not yet a “generalized” version of the theorem that corresponds to deeper KANs.

The breakthrough occurs when we notice

the analogy between MLPs and KANs. In MLPs, once we define a layer (which is composed of a linear transformation and nonlinearities), we can stack more layers to make the network deeper. To build deep KANs, we should first answer: “what is a KAN layer?”

It turns out that a KAN layer with  $n_{\text{in}}$ -dimensional inputs and  $n_{\text{out}}$ -dimensional outputs can be defined as a matrix of 1D functions:

$$\Phi = \{\phi_{q,p}\}, \quad p = 1, 2, \dots, n_{\text{in}}, \quad q = 1, 2, \dots, n_{\text{out}}. \quad (2)$$

In the Kolmogorov–Arnold theorem, the inner functions form a KAN layer with  $n_{\text{in}} = n$  and  $n_{\text{out}} = 2n + 1$ , and the outer functions form a KAN layer with  $n_{\text{in}} = 2n + 1$  and  $n_{\text{out}} = 1$ . So the Kolmogorov–Arnold representations in Equation (2.1) are simply compositions of two KAN layers. Now it becomes clear what it means to have deeper Kolmogorov–Arnold representations: simply stack more KAN layers!

Let us introduce some notation. The shape of a KAN is represented by an integer array:

$$[n_0, n_1, \dots, n_L], \quad (3)$$

where  $n_i$  is the number of nodes in the  $i$ -th layer of the computational graph. We denote the  $i$ -th neuron in the  $l$ -th layer by  $(l, i)$ , and

its activation value by  $x_{l,i}$ . Between layer  $l$  and  $l + 1$ , there are  $n_l n_{l+1}$  activation functions. The activation function that connects  $(l, i)$  and  $(l + 1, j)$  is denoted by

$$\begin{aligned} \phi_{l,j,i}, \\ l = 0, \dots, L - 1; \quad i = 1, \dots, n_l; \\ j = 1, \dots, n_{l+1}. \end{aligned} \quad (4)$$

The pre-activation of  $\phi_{l,j,i}$  is simply  $x_{l,i}$ , and its post-activation is

$$\tilde{x}_{l,j,i} \equiv \phi_{l,j,i}(x_{l,i}).$$

The activation value of the  $(l + 1, j)$ -th neuron is the sum of all incoming post-activations:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \quad j = 1, \dots, n_{l+1}. \quad (5)$$

In matrix form, this becomes:

$$x_{l+1} = \Phi_l x_l, \quad (6)$$

where  $\Phi_l$  is the function matrix corresponding to the  $l$ -th KAN layer:

$$\Phi_l = \begin{pmatrix} \phi_{l,1,1}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \ddots & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}.$$

A general KAN network is a composition of  $L$  layers. Given an input vector  $x_0 \in \mathbb{R}^{n_0}$ , the output of KAN is:

$$\text{KAN}(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_0)(x_0). \quad (7)$$

Assuming output dimension  $n_L = 1$ , and defining  $f(x) \equiv \text{KAN}(x)$ , we can express:

$$\begin{aligned} f(x) = \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{L-1,1,i_{L-1}} \left( \sum_{i_{L-2}=1}^{n_{L-2}} \cdots \right. \\ \left. \sum_{i_1=1}^{n_1} \phi_{1,i_2,i_1} \left( \sum_{i_0=1}^{n_0} \phi_{0,i_1,i_0}(x_{i_0}) \right) \cdots \right). \end{aligned} \quad (8)$$

This expression is cumbersome, so our abstraction of KAN layers and their visualizations are cleaner and intuitive. The original Kolmogorov–Arnold representation corresponds to a 2-layer KAN with shape  $[n, 2n + 1, 1]$ .

All operations are differentiable, so KANs can be trained via backpropagation.

For comparison, an MLP can be written as interleaving of affine transformations  $W$  and nonlinearities  $\sigma$ :

$$\text{MLP}(x) = (W_{L-1} \circ \sigma \circ W_{L-2} \circ \sigma \circ \cdots \circ \sigma \circ W_0)(x). \quad (9)$$

In MLPs, linear and nonlinear components are treated separately (as  $W$  and  $\sigma$ ), whereas KANs treat both together via  $\Phi$ .

### Implementation Details.

Although a KAN layer looks simple, optimizing it requires some techniques:

- **Residual activation functions:**

$$\phi(x) = w_b b(x) + w_s \cdot \text{spline}(x), \quad (10)$$

where

$$b(x) = \text{silu}(x) = \frac{x}{1 + e^{-x}}, \quad (11)$$

and the spline function is a linear combination of B-spline basis:

$$\text{spline}(x) = \sum_i c_i B_i(x), \quad (12)$$

with trainable coefficients  $c_i$ .

- **Initialization scales:** Initialize  $w_s = 1$  and  $\text{spline}(x) \approx 0$ .  $w_b$  is initialized via Xavier initialization.

- **Adaptive spline grids:** Update grid points dynamically during training to cope with values evolving out of the fixed region.

**Parameter Count.** Assume:

- Depth  $L$ ,
- Equal width  $n_0 = n_1 = \cdots = n_L = N$ ,
- Each spline of order  $k$  on  $G$  intervals ( $G + 1$  grid points).

Then the parameter count is approximately:

$$\mathcal{O}(N^2 L (G + k)) \sim \mathcal{O}(N^2 L G),$$

compared to an MLP:

$$\mathcal{O}(N^2 L).$$

Though KAN appears less efficient, it often needs a smaller  $N$  than MLPs and hence better generalization and interpretability.

### 4.3.2 Comparison with MLPs

- **Activation placement:** In MLPs, activations are applied to nodes; in KANs, they are applied to edges.
- **Linear vs nonlinear units:** MLPs use fixed nonlinear activations and learn linear weights; KANs use no linear weights but learn the nonlinear functions.
- **Interpretability:** KANs allow visualization and symbolic interpretation of the learned univariate functions, making them more transparent.
- **Compositional efficiency:** KANs can better exploit function compositionality, yielding improved generalization and efficiency in scientific and symbolic tasks.

### 4.3.3 Expressive Power and Scaling Laws

KANs generalize the original Kolmogorov–Arnold decomposition by extending it to arbitrary depth and width. It has been theoretically shown that under suitable smoothness assumptions, KANs with spline-based functions can achieve approximation errors that scale favorably with grid resolution  $G$ :

$$\|f - f_{\text{KAN}}\|_{\infty} = \mathcal{O}(G^{-k-1}),$$

where  $k$  is the spline order. This implies faster convergence and better scaling than standard MLPs, particularly for structured functions in low- to moderate-dimensional settings.

### 4.3.4 Conclusion

Kolmogorov–Arnold Networks represent a new direction in neural network architecture design, inspired by deep mathematical insights. By reversing the traditional roles of weights and activations, KANs offer a flexible, interpretable, and theoretically grounded framework for function approximation. Their ability to model compositional and symbolic structures makes them particularly appealing for applications in scientific machine learning, symbolic regression, and interpretable AI.

### 4.4 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a special type of Recurrent Neural Network (RNN) architecture that is capable of learning long-range dependencies in sequential data. It was first introduced by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing and exploding gradient problems that often occur in training traditional RNNs. Since then, LSTMs have become a fundamental building block in many applications involving time-dependent or sequence data, including natural language processing, time series forecasting, speech recognition, and video analysis.

Unlike standard RNNs, which tend to struggle when learning dependencies across longer sequences, LSTMs are designed with an internal memory mechanism that enables them to preserve information over extended periods. This is achieved through the use of a gated architecture, which controls the flow of information into and out of the memory cell.

An LSTM cell consists of three primary gates:

- **Forget gate ( $f_t$ ):** Decides what information from the previous cell state should be discarded.
- **Input gate ( $i_t$ ):** Determines which values will be updated and added to the cell state.
- **Output gate ( $o_t$ ):** Decides what the next hidden state should be, based on the cell state.

The mathematical formulation of the LSTM cell at time step  $t$  is given by:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (13)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (14)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (15)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (16)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (17)$$

$$h_t = o_t \odot \tanh(C_t) \quad (18)$$

Where:



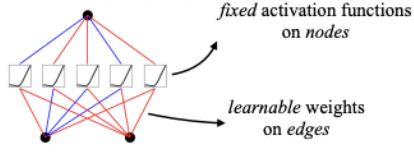
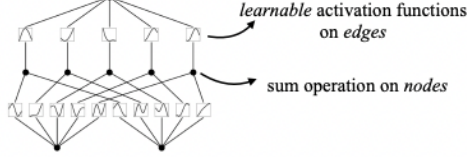
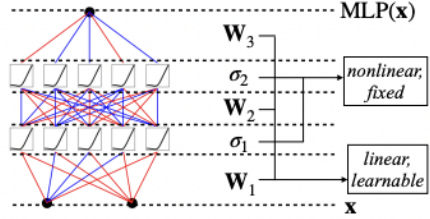
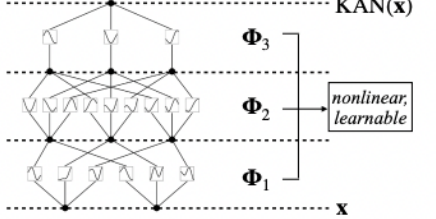
Model	<b>Multi-Layer Perceptron (MLP)</b>	<b>Kolmogorov-Arnold Network (KAN)</b>
Theorem	<b>Universal Approximation Theorem</b>	<b>Kolmogorov-Arnold Representation Theorem</b>
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a)  fixed activation functions on nodes learnable weights on edges	(b)  learnable activation functions on edges sum operation on nodes
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	(c)  MLP(x) $\mathbf{W}_3$ $\sigma_2$ $\mathbf{W}_2$ $\sigma_1$ $\mathbf{W}_1$ $\mathbf{x}$ nonlinear, fixed linear, learnable	(d)  KAN(x) $\Phi_3$ $\Phi_2$ $\Phi_1$ $\mathbf{x}$ nonlinear, learnable

Figure 0.1: Multi-Layer Perceptrons (MLPs) vs. Kolmogorov-Arnold Networks (KANs)

Figure 4: KAN and MLP comparison

- $x_t$  is the input at time step  $t$
- $h_{t-1}$  is the hidden state from the previous time step
- $C_{t-1}$  is the previous cell state
- $f_t, i_t, o_t$  are the forget, input, and output gates, respectively
- $\tilde{C}_t$  is the candidate cell state
- $C_t$  is the updated cell state
- $h_t$  is the new hidden state
- $\sigma(\cdot)$  denotes the sigmoid activation function
- $\tanh(\cdot)$  denotes the hyperbolic tangent activation function
- $\odot$  denotes element-wise (Hadamard) product

The *forget gate*  $f_t$  allows the LSTM to selectively remove information from the memory cell, while the *input gate*  $i_t$  and the candidate

value  $\tilde{C}_t$  enable the model to incorporate new information. The *output gate*  $o_t$  determines how much of the internal cell state contributes to the output hidden state  $h_t$ .

Thanks to this architecture, LSTMs are capable of capturing both short-term and long-term dependencies within sequences. This is especially beneficial in tasks such as machine translation, where a word's meaning might depend on previous words in a sentence, or in stock price prediction, where past prices influence future trends.

Despite their strengths, LSTMs are computationally more expensive than traditional RNNs due to the complex gating mechanism. However, their performance gains often justify the additional computational cost. In practice, various extensions such as Bidirectional LSTM (BiLSTM) and stacked LSTM networks are used to further enhance model performance.

#### 4.5 Temporal Kolmogorov–Arnold Networks (TKANs)

Temporal Kolmogorov–Arnold Networks (TKANs) extend the Kolmogorov–Arnold



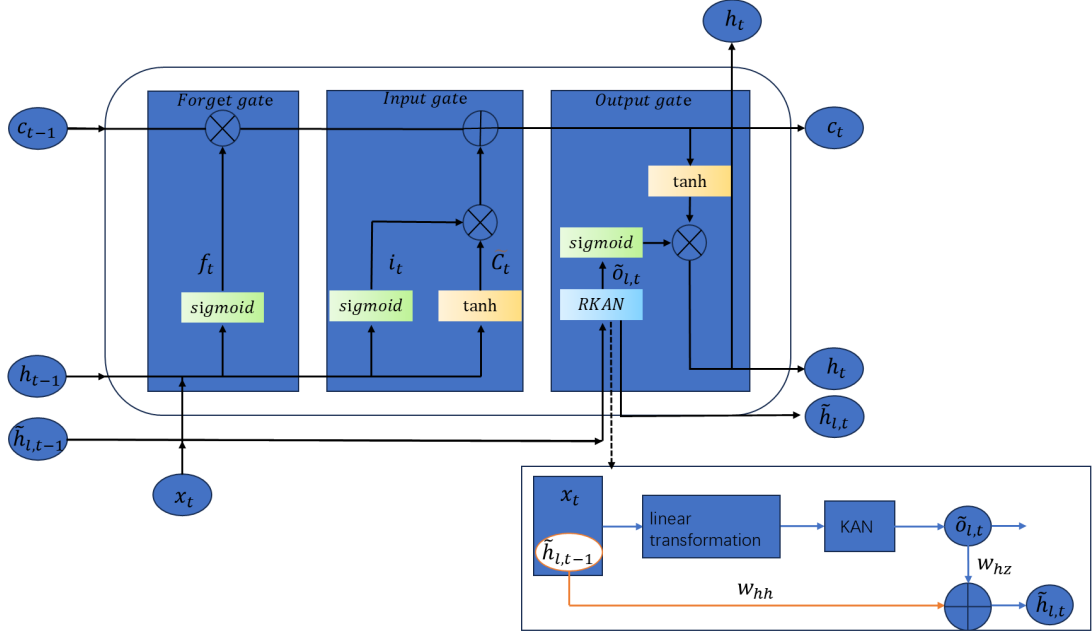


Figure 5: TKAN and RKAN architecture

Network (KAN) framework to handle sequential data and temporal dependencies. Inspired by the success of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) architectures in modeling time series, TKANs integrate the compositional structure of KANs with recurrence and memory mechanisms. This hybrid design allows TKANs to model long-term dependencies and nonlinear dynamics in time series with enhanced accuracy and interpretability.

#### 4.5.1 Architecture Overview

TKANs are built from layers called **Recurring Kolmogorov–Arnold Networks (RKANs)**, which are KAN layers extended with temporal recurrence. These RKANs are coupled with modified LSTM-style memory management components, enabling the network to retain and control access to past information.

##### Key components of TKAN:

- **RKAN Layers:** KAN layers that incorporate previous hidden states into their computation.
- **Gating Mechanisms:** Adapted from LSTM, including forget, input, and output

gates, which regulate the flow of information through time.

#### 4.5.2 Recurring Kolmogorov–Arnold Networks (RKANs)

To introduce temporal recurrence into KANs, each RKAN layer maintains a hidden memory state  $\tilde{h}_{\ell,t}$  at time  $t$  for each layer  $\ell$ . The input to the RKAN at time  $t$  is a combination of the current input  $x_t$  and the previous hidden state:

$$s_{\ell,t} = W_{\ell,x}x_t + W_{\ell,h}\tilde{h}_{\ell,t-1}, \quad (19)$$

where  $W_{\ell,x}$  and  $W_{\ell,h}$  are learnable weight matrices. The output of the RKAN layer is given by:

$$\tilde{o}_t = \varphi_\ell(s_{\ell,t}), \quad (20)$$

where  $\varphi_\ell$  is a standard KAN layer consisting of learnable univariate spline functions. The updated memory state is computed as:

$$\tilde{h}_{\ell,t} = W_{hh}\tilde{h}_{\ell,t-1} + W_{hz}\tilde{o}_t. \quad (21)$$

This design allows the RKAN layer to retain short-term memory across time steps and learn temporal dependencies directly.

### 4.5.3 LSTM-Inspired Gating in TKAN

To enhance memory management, TKAN integrates LSTM-like gating mechanisms. For each time step  $t$ , the gates are computed as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad (22)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad (23)$$

$$o_t = \sigma(W_o r_t + b_o), \quad (24)$$

where  $r_t$  is the concatenated output from multiple RKAN sublayers:

$$r_t = \text{Concat}[\varphi_1(s_{1,t}), \dots, \varphi_L(s_{L,t})]. \quad (25)$$

The memory cell and final hidden state update are:

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \quad (26)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (27)$$

$$h_t = o_t \odot \tanh(c_t). \quad (28)$$

This architecture ensures that TKAN can remember relevant long-term information and selectively update or forget past states, allowing for robust modeling of complex temporal patterns.

### 4.5.4 Advantages of TKAN

- **Expressiveness:** TKANs combine the universal approximation power of KANs with sequence modeling capabilities of RNNs.
- **Interpretability:** Like KANs, the univariate spline functions in TKAN can be visualized and interpreted.
- **Temporal Efficiency:** The RKAN memory design captures sequential dynamics more effectively than standard KANs or MLPs.
- **Stability:** Experiments show that TKANs provide more stable learning curves and lower prediction variance across training runs, especially in multi-step forecasting tasks.

## 5 Experiments

### 5.1 Data Preprocessing

#### 5.1.1 Handling Columns with High Missing Rates

The raw dataset contains 50 features, many of which are collected infrequently (e.g., every few days). As a result, several columns exhibit a high proportion of missing values, making them unsuitable for effective time series modeling. Figure 7 shows the top columns with the highest missing data percentages.

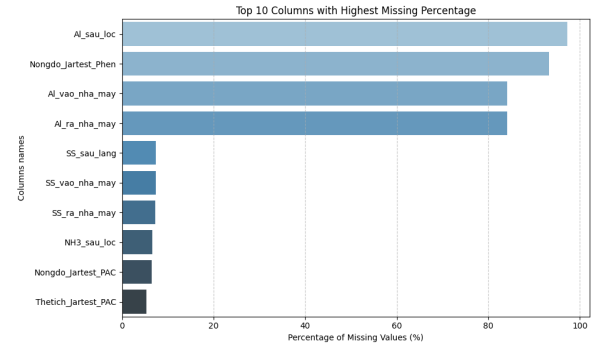


Figure 7: Columns with the highest missing data percentages

After reviewing the missing data distribution, we decided to remove columns with excessive missing values. Specifically, we dropped 4 columns and retained the remaining 46 features for further analysis and modeling.

#### 5.1.2 Handling Outliers

To assess data quality, we visualized the distribution of selected features using box plots. As illustrated in Figure 8, many features contain extreme values or outliers.

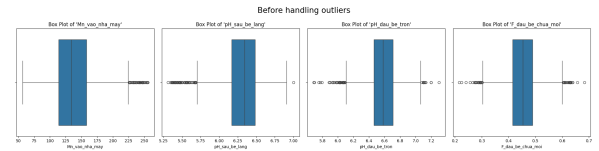


Figure 8: Box plots of selected features showing presence of outliers

We applied the **three standard deviations rule ( $3\sigma$  rule)** to detect and remove outliers. This statistical approach assumes the data is approximately normally distributed, and it

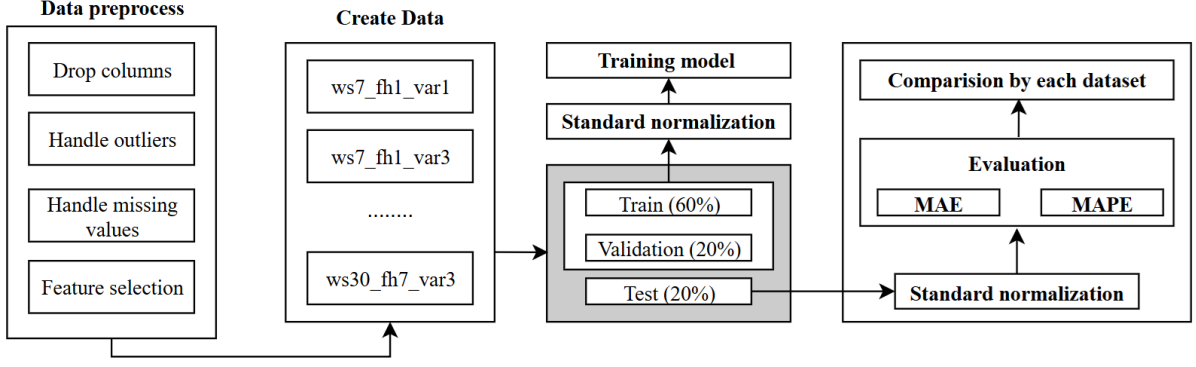


Figure 6: Pipeline overview.

classifies any data point outside the interval  $[\mu - 3\sigma, \mu + 3\sigma]$  as an outlier, where  $\mu$  is the mean and  $\sigma$  is the standard deviation of the feature.

Detected outliers were replaced with NaN values, to be handled later during the imputation process.

### 5.1.3 Filling Missing Values

Despite removing high-missing-value columns, the dataset still contained missing entries in the retained features. Given the sequential nature of the data, we used **linear interpolation** to estimate missing values between known points.

$$\hat{y}(x) = y_i + \frac{(y_{i+1} - y_i)(x - x_i)}{(x_{i+1} - x_i)}$$

where:

- $\hat{y}(x)$  is the interpolated value at  $x$ ,
- $y_i$  and  $y_{i+1}$  are the known values at  $x_i$  and  $x_{i+1}$ ,
- $x$  is the point at which we want to interpolate.

However, this method is ineffective at the boundaries of the dataset (i.e., the beginning or end of the series), where surrounding data points are unavailable. To handle these edge cases, we applied a combination of **backward fill** and **forward fill** techniques.

This hybrid imputation approach ensures that all missing values are filled while preserving the temporal consistency required for time series modeling.

### 5.1.4 Feature Selection

After cleaning the dataset, we retained a total of 46 features across 2,191 time steps. However, using all features can lead to increased model complexity and longer training time, without necessarily improving performance. To address this, we performed feature selection to retain only the most relevant predictors for salinity forecasting. We utilized the Pearson correlation coefficient  $r_{XY}$ , defined as:

$$r_{XY} = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

where:

- $X_i$  and  $Y_i$  are the values of features  $X$  and  $Y$  for the  $i$ -th observation,
- $\bar{X}$  and  $\bar{Y}$  are the means of features  $X$  and  $Y$ , respectively.

Since our objective is to predict the **salinity of raw water before any treatment**, we only considered features measured *before the water enters the treatment process*. Based on domain knowledge and correlation analysis, we selected the following features:

- **Man\_song\_saigon** – Salinity of the Saigon River (target variable).
- **Dodan\_vao\_nha\_may** – Electrical conductivity of incoming water.
- **pH\_Song\_SG** – pH value of the river water.

These features are sensor-based, readily available in real time, and have strong predictive relationships with water salinity. The final cleaned and feature-selected dataset was used as input for model training and evaluation.

## 5.2 Create Dataset

To investigate the impact of key parameters on the performance of salinity forecasting, we constructed multiple datasets by systematically varying three important factors: **window size**, **forecast horizon**, and **number of input variables**. The goal of this process is to identify which configuration provides the most accurate and robust predictions using time series models.

- **Window size (ws):** This parameter defines the number of past time steps used as input features to predict future values. We explored three different values: **7 days**, **15 days**, and **30 days**. A smaller window captures short-term dependencies, while a larger window allows the model to incorporate longer historical patterns.
- **Forecast horizon (fh):** This defines how many days ahead the model is expected to make predictions. We considered **1-day**, **3-day**, and **7-day** forecast horizons to assess both short-term and longer-term forecasting capabilities.
- **Number of input variables (var):** We used two configurations: **1 variable** (only the salinity value of the Saigon River) and **3 variables** (salinity, pH, and electrical conductivity). The multivariate setup provides richer contextual information, which can potentially improve prediction accuracy, but may also increase model complexity and training time.

By combining all values of these three parameters, we generated a total of **18 unique datasets**:

$$3 \text{ (window sizes)} \times 3 \text{ (forecast horizons)} \\ \times 2 \text{ (variable settings)} = 18 \text{ datasets}$$

Each dataset consists of input-output pairs, where the input is a sequence of measurements (based on the selected variables and window size), and the output is the predicted salinity value(s) after the forecast horizon.

From this point onward, for clarity and consistency, we adopt a standardized naming convention for each dataset using the following format:

$$ws\{window\ size\}_fh\{forecast\ horizon\}_var\{number\ of\ variables\}$$

For example, the dataset named `ws15_fh3_var1` refers to a dataset that uses a 15-day input window, a 3-day forecast horizon, and 1 input variable (salinity only). This convention simplifies reference and comparison throughout our experimental results.

The datasets were split into three groups: **60%** for the training set, **20%** for the validation set, and **20%** for the test set. After the data splitting process, the dataset will be normalized using the following formula prior to training and evaluation:

$$z = \frac{x - \mu}{\sigma}$$

where  $z$  denotes the normalized value,  $x$  is the original data,  $\mu$  represents the mean of the dataset, and  $\sigma$  is the standard deviation of the dataset. This normalization process ensures that the data is scaled to a standard range, facilitating improved model performance and convergence during training.

This structured approach allows us to rigorously evaluate how different time series configurations influence model performance, helping us determine the optimal setup for salinity prediction on the Saigon River dataset.

## 5.3 Experiment Settings

In this study, we focused on the **Temporal Kolmogorov-Arnold Network (TKAN)** as the primary architecture for time series forecasting. For comparison purposes, we also implemented the **Long Short-Term Memory**

(LSTM) model. To address different forecasting requirements, both TKAN and LSTM were configured for three distinct prediction horizons: **1 day**, **3 days**, and **7 days** ahead. Each model was tuned with its own architecture and training settings, as detailed in the following sections. TKAN remained the main focus due to its promising capability to capture complex temporal dependencies.

### 5.3.1 TKAN

**TKAN for 1-day Forecast Horizon** For short-term forecasting, we used a relatively compact TKAN model with 8 hidden units. The sub-KAN layers were set with an input and output dimensionality of 15.

- **Units:** 8
- **Sub-KAN Input/Output Dim:** 15
- **Activation:** tanh
- **Recurrent Activation:** sigmoid
- **Return Sequences:** False
- **Batch Size:** 128
- **Max Epochs:** 50

**TKAN for 3-day Forecast Horizon** To handle medium-range predictions, we increased the model complexity by expanding the TKAN to 32 units and using sub-KAN layers of dimension 30.

- **Units:** 32
- **Sub-KAN Input/Output Dim:** 30
- **Activation:** tanh
- **Recurrent Activation:** sigmoid
- **Return Sequences:** False
- **Batch Size:** 128
- **Max Epochs:** 50

**TKAN for 7-day Forecast Horizon** For long-term forecasting, the model was further scaled up to 64 units with sub-KAN layers of dimension 40 to better capture extended temporal dependencies.

- **Units:** 64
- **Sub-KAN Input/Output Dim:** 40
- **Activation:** tanh
- **Recurrent Activation:** sigmoid
- **Return Sequences:** False
- **Batch Size:** 128
- **Max Epochs:** 50

### 5.3.2 LSTM

For comparative purposes, we also implemented the Long Short-Term Memory (LSTM) model using the same forecasting horizons: **1 day**, **3 days**, and **7 days**. The configurations were aligned with TKAN settings to ensure a fair comparison, but without the sub-KAN dimensions.

#### LSTM for 1-day Forecast Horizon

- **Units:** 8
- **Activation:** tanh
- **Recurrent Activation:** sigmoid
- **Return Sequences:** False
- **Batch Size:** 128
- **Max Epochs:** 50

#### LSTM for 3-day Forecast Horizon

- **Units:** 32
- **Activation:** tanh
- **Recurrent Activation:** sigmoid
- **Return Sequences:** False
- **Batch Size:** 128
- **Max Epochs:** 50

## LSTM for 7-day Forecast Horizon

- **Units:** 64
- **Activation:** tanh
- **Recurrent Activation:** sigmoid
- **Return Sequences:** False
- **Batch Size:** 128
- **Max Epochs:** 50

### 5.3.3 Baseline Configuration

To evaluate the performance of our proposed model, we established a baseline using results from a related study that employed the same dataset. The baseline consists of several traditional and deep learning models widely used in time series forecasting tasks. These models include:

- **ARIMA (AutoRegressive Integrated Moving Average):** ARIMA is a classical statistical model commonly used for analyzing and forecasting univariate time series data. It combines three components: autoregression (AR), differencing (I), and moving average (MA). The AR component captures the relationship between an observation and a number of lagged observations, the I component deals with making the time series stationary by differencing, and the MA component models the error of the prediction as a linear combination of past error terms. ARIMA is particularly effective for datasets with strong linear trends or seasonal effects, but it struggles with nonlinear relationships and multivariate data.
- **Artificial Neural Network (ANN):** ANNs are a class of feedforward neural networks inspired by the structure of the human brain. They consist of an input layer, one or more hidden layers, and an output layer. Each layer contains nodes (neurons) connected by weighted edges, and each node applies an activation function to its inputs. ANNs are capable of

modeling nonlinear relationships and are suitable for both univariate and multivariate forecasting. However, traditional ANNs lack memory and therefore do not capture temporal dependencies directly, which can be a limitation in time series tasks.

- **Convolutional Neural Network (CNN):** CNNs, though originally designed for image processing, have been effectively adapted for time series analysis using one-dimensional (1D) convolutions. A CNN captures local patterns in the data by applying convolutional filters over temporal windows. This allows the network to learn short-term dependencies such as local trends and periodic patterns. CNNs are computationally efficient and can extract high-level features from raw sequences, but they may be limited in modeling long-term temporal dependencies unless combined with recurrent or attention mechanisms.
- **Gated Recurrent Unit (GRU):** GRUs are a type of recurrent neural network that offer a simplified architecture compared to LSTM by combining the forget and input gates into a single update gate. This reduction in complexity often leads to faster training and fewer parameters, while still maintaining the ability to capture temporal dynamics in sequence data. GRUs are particularly effective for problems where short to medium-range dependencies are sufficient, and they are commonly used in real-time prediction systems due to their efficiency.
- **Temporal Convolutional Network (TCN):** TCNs are a modern alternative to RNN-based models for sequence modeling. They employ dilated causal convolutions, which allow the receptive field to grow exponentially with the number of layers, making it possible to model long-range dependencies efficiently. TCNs also incorporate residual



Table 1: Number of parameters for TKAN and LSTM across datasets

Forecast Horizon	Dataset	TKAN	LSTM
fh = 1	ws7_fh1_var1	2,417	329
	ws7_fh1_var3	2,495	393
	ws15_fh1_var1	2,417	329
	ws15_fh1_var3	2,495	393
	ws30_fh1_var1	2,417	329
	ws30_fh1_var3	2,495	393
fh = 3	ws7_fh3_var1	12,035	4,451
	ws7_fh3_var3	12,287	4,707
	ws15_fh3_var1	12,035	4,451
	ws15_fh3_var3	12,287	4,707
	ws30_fh3_var1	12,035	4,451
	ws30_fh3_var3	12,287	4,707
fh = 7	ws7_fh7_var1	29,191	17,351
	ws7_fh7_var3	29,655	17,863
	ws15_fh7_var1	29,191	17,351
	ws15_fh7_var3	29,655	17,863
	ws30_fh7_var1	29,191	17,351
	ws30_fh7_var3	29,655	17,863

connections, which facilitate training deep networks. Unlike RNNs, TCNs process sequences in parallel and avoid the vanishing gradient problem. They are particularly well-suited for long sequence inputs and are known for their stability and interpretability.

The best-performing result from this set of models, for each dataset configuration, is selected as the **baseline** for comparison. This diverse set of models ensures a comprehensive evaluation, ranging from classical statistical techniques to modern deep learning architectures.

**Baseline Results** Table 2 summarizes the baseline performance for various dataset configurations, including combinations of different window sizes (ws), forecast horizons (fh), and input variable counts (var). Each entry shows the best-performing model for that specific configuration along with its Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE).

Table 2: Performance of baseline models on different dataset configurations.

Dataset	Model	MAE	MAPE (%)
ws7_fh1_var1	CNN_1Var	1.40	4.31
ws7_fh1_var3	GRU_3Var	1.50	4.67
ws7_fh3_var1	GRU_1Var	3.20	10.00
ws7_fh3_var3	ANN_3Var	3.33	10.30
ws7_fh7_var1	ANN_1Var	5.38	16.70
ws7_fh7_var3	ANN_3Var	5.61	16.30
ws15_fh1_var1	CNN_1Var	1.43	4.40
ws15_fh1_var3	GRU_3Var	1.62	5.15
ws15_fh3_var1	CNN_1Var	3.24	10.00
ws15_fh3_var3	GRU_3Var	3.47	10.40
ws15_fh7_var1	CNN_1Var	5.77	17.30
ws15_fh7_var3	LSTM_3Var	5.61	15.90
ws30_fh1_var1	GRU_1Var	1.52	4.57
ws30_fh1_var3	GRU_3Var	1.56	4.90
ws30_fh3_var1	GRU_1Var	3.42	10.70
ws30_fh3_var3	TCN_3Var	3.58	10.80
ws30_fh7_var1	CNN_1Var	5.50	17.10
ws30_fh7_var3	ANN_3Var	5.79	18.80

### 5.3.4 Optimization and Training Strategy

All models were trained using the **Adam optimizer** with a fixed learning rate of 0.001, which is well-suited for time series forecasting tasks due to its adaptive learning rate and efficient handling of sparse gradients.

To improve training efficiency, reduce over-

fitting, and ensure optimal model performance, we employed **callback function** provided by the Keras framework. Callbacks are utility functions executed during training to monitor and modify the training process dynamically.

**EarlyStopping:** This callback stops the training process early if the model’s performance on a validation metric does not improve after a certain number of epochs. It is particularly useful for preventing overfitting and reducing training time.

```
EarlyStopping(monitor='val_loss',
              patience=20,
              restore_best_weights=True)
```

- **monitor:** Metric to monitor (e.g., val\_loss, val\_accuracy).
- **patience:** Number of epochs with no improvement before stopping.
- **restore\_best\_weights:** If True, the model will revert to the weights with the best monitored score.

In our experiments, we primarily used EarlyStopping and optionally to ensure efficient training and retain the best model. These strategies help avoid overfitting, accelerate convergence, and improve the reproducibility and robustness of the results.

## 5.4 Results

### 5.4.1 Evaluation Metrics

To evaluate the accuracy and robustness of forecasting models, we employed two widely used performance metrics: **Mean Absolute Error (MAE)** and **Mean Absolute Percentage Error (MAPE)**.

- **Mean Absolute Error (MAE)** quantifies the average magnitude of the errors between predicted and actual values without considering their direction. It is computed as:

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

where  $y_i$  is the actual value and  $\hat{y}_i$  is the predicted value at time  $i$ .

- **Mean Absolute Percentage Error (MAPE)** provides a relative error measure expressed as a percentage, making it easier to interpret in real-world applications. It is defined as:

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

### 5.4.2 Experimental Setup

To ensure statistical reliability, we performed **50 independent runs** for both the **Temporal Kolmogorov-Arnold Network (TKAN)** and the **Long Short-Term Memory (LSTM)** model across all dataset configurations. These results were then compared with baseline models, which predicts the mean of past values or uses naive forecasts.

The choices of model architecture, training hyperparameters, and optimization strategy in this study were made based on a combination of empirical evidence, literature best practices, and characteristics of the dataset.

**Model architecture:** We adapted the TKAN model to different forecasting horizons by adjusting the number of hidden units and the dimensions of sub-KAN layers. For short-term forecasts (1 day ahead), a smaller model with 8 units and 15-dimensional sub-KANs was sufficient to capture local temporal dependencies while remaining computationally efficient. For longer horizons (3 and 7 days), we increased the model complexity to 32 and 64 units, respectively, with wider sub-KAN layers (30 and 40 dimensions) to capture more abstract and long-range temporal patterns. This scaling strategy aligns with common practices in time series modeling, where longer forecasts typically require models with greater capacity.

**Batch size and epochs:** Large batch sizes (**128**) helped stabilize gradient updates in deeper models. The number of training epochs was limited (**150**) to avoid overfitting. These values were determined through trial-and-error and confirmed by monitoring validation loss during early runs.

**Learning rate and optimizer:** We used the Adam optimizer with a fixed learning rate of

**0.001**, a widely adopted configuration in deep learning due to its adaptive moment estimation and robust convergence behavior. This setting provides a good balance between training stability and convergence speed across diverse architectures.

**Early stopping callback:** To prevent overfitting and unnecessary computation, we employed the `EarlyStopping` callback with a patience of **20 epochs**. This means training was stopped if no improvement in validation loss was observed for 20 consecutive epochs. Additionally, the model automatically restored the best weights, ensuring optimal performance without manual intervention.

Overall, these choices were informed by the nature of the salinity time series data — which exhibits nonlinear trends, occasional noise, and temporal variability — as well as by the need for generalization and training efficiency across 18 dataset configurations.

### 5.4.3 Performance Comparison

The following figures illustrate the average MAE and MAPE scores across 18 dataset variations (combinations of window size, forecast horizon, and number of input variables) for all three types of models.

#### 5.4.4 Detailed Observations

**TKAN vs LSTM** Experimental results indicate that the Temporal Kolmogorov-Arnold Network (TKAN) consistently outperforms the Long Short-Term Memory (LSTM) model in terms of both Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) across all dataset configurations. Notably, TKAN not only achieves lower forecasting errors, but also demonstrates more robust behavior, as evidenced by its stable performance over 50 runs. This suggests that TKAN is less sensitive to random initialization and training dynamics, making it a more reliable choice for time series forecasting tasks.

However, one key trade-off is observed in training efficiency. As shown in Figure 11, TKAN requires substantially more training time than LSTM. While the LSTM model maintains relatively consistent training dura-

tions across datasets, TKAN exhibits increased training time, particularly when the input dimensionality is scaled up. This can be attributed to the architecture of TKAN, where both the sub layers input and output dimensions are adjusted to accommodate higher-dimensional input features, leading to greater computational cost.

Despite this overhead, TKAN’s superior accuracy and robustness make it a compelling choice for applications that demand high prediction quality. In scenarios where forecast precision is critical—such as energy demand prediction, financial modeling, or medical time series—this trade-off in training time may be acceptable or even preferable.

**TKAN vs Baseline Models** In comparison with baseline models, TKAN continues to demonstrate strong performance. It achieves lower MAE on the majority of evaluated datasets, particularly excelling in scenarios with longer forecast horizons (e.g.,  $fh = 3$  and  $fh = 7$ ). This highlights the model’s capacity to capture complex temporal dependencies over extended future windows, a common challenge in time series prediction.

Interestingly, for short-term forecasting tasks (i.e.,  $fh = 1$ ), TKAN’s performance is generally on par with or slightly worse than the best baseline models. This outcome suggests that the advantages of the TKAN architecture become more pronounced as the forecasting horizon increases. Nevertheless, even in these cases, TKAN maintains a high degree of stability, which is not always observed in baseline methods.

Overall, TKAN strikes a strong balance between accuracy and generalization. While it does not outperform baseline models on every single dataset, it consistently delivers top-tier performance across a wide range of configurations. These results suggest that TKAN is a versatile and effective forecasting model, particularly suitable for high-stakes and long-horizon prediction tasks where both accuracy and robustness are critical.

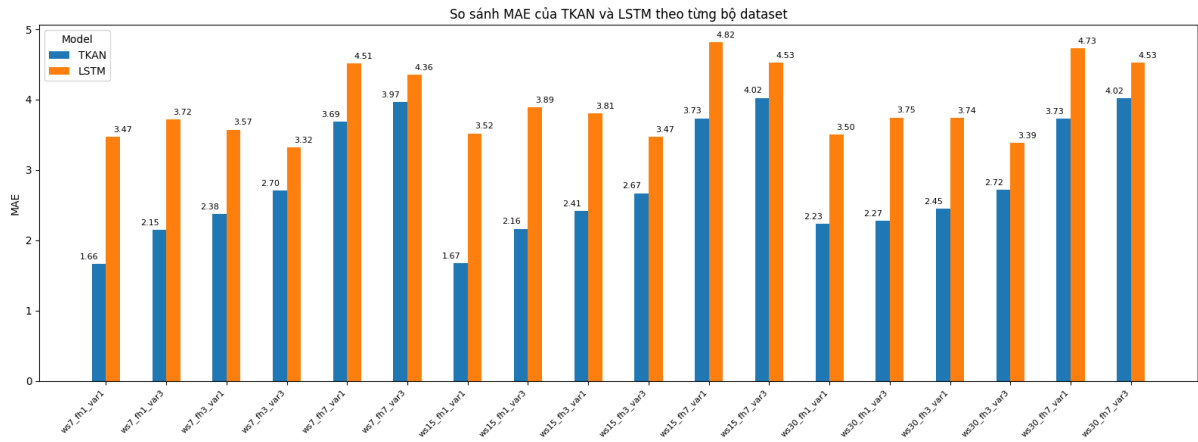


Figure 9: MAE : TKAN and LSTM model

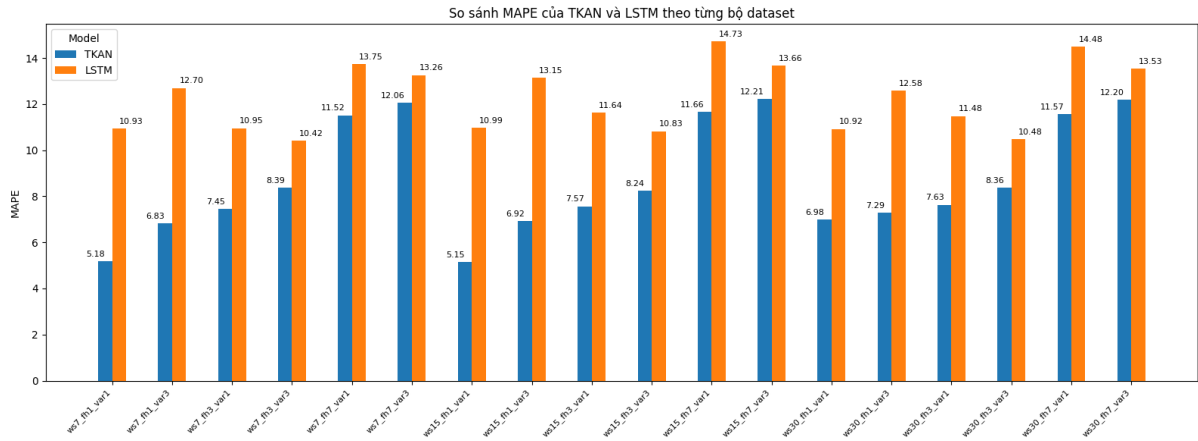


Figure 10: MAE : TKAN and LSTM model

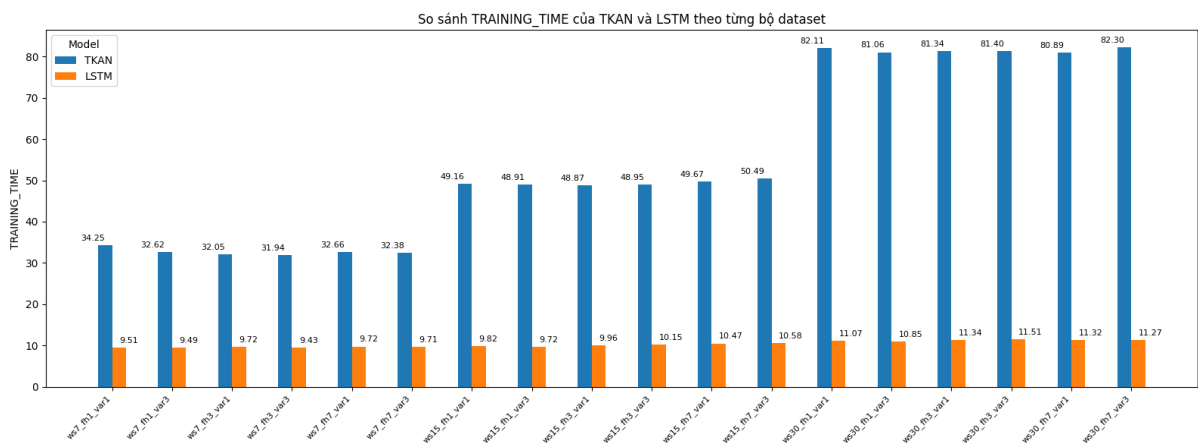


Figure 11: MAE : TKAN and LSTM model

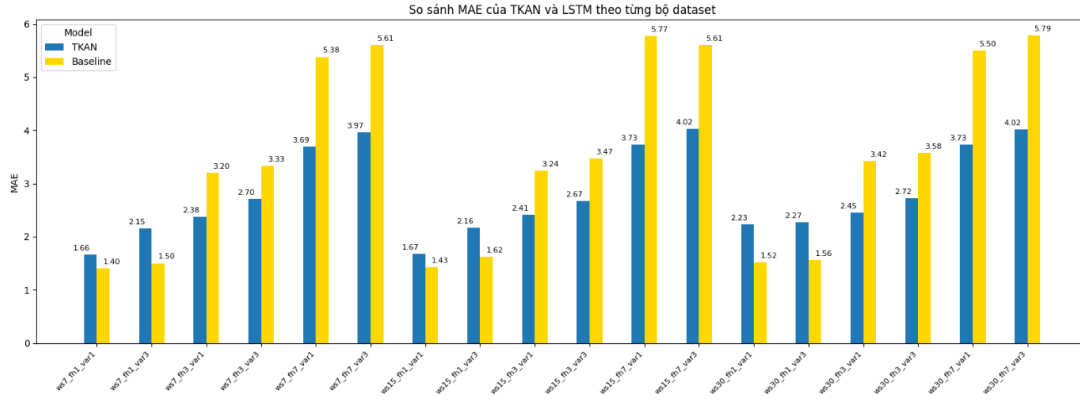


Figure 12: MAE : TKAN and Baseline models

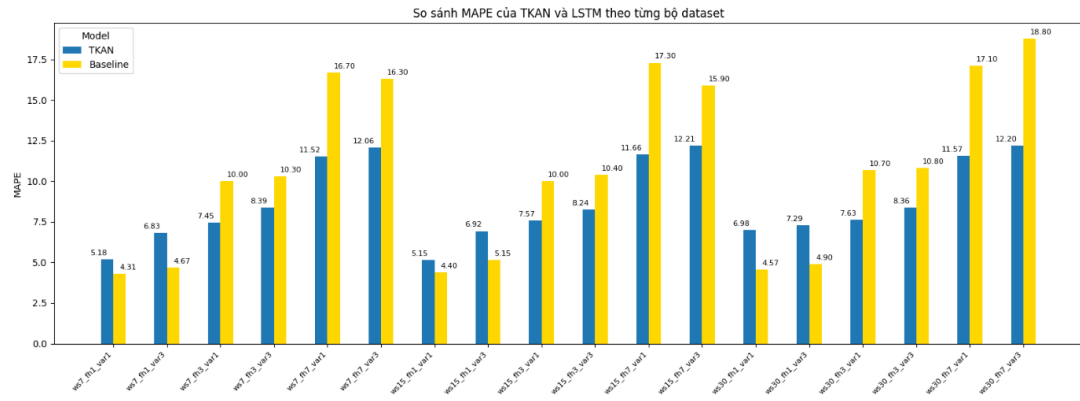


Figure 13: MAE : TKAN and Baseline models

## 6 Conclusion

In this study, we investigated the application of the **Temporal Kolmogorov-Arnold Network (TKAN)** for forecasting salinity levels in raw water from the Saigon River. Accurate salinity prediction plays a crucial role in water resource management, particularly for urban water treatment systems affected by tidal and seasonal variations.

We systematically evaluated TKAN across 18 datasets with varying window sizes, forecast horizons, and input variable combinations. The model's performance was compared to both a baseline predictor and the Long Short-Term Memory (LSTM) network. Experiments were repeated 50 times per configuration to ensure statistical robustness.

The results demonstrate that TKAN consistently outperforms baseline models in terms of both MAE and MAPE, especially for medium- and long-term forecasts. TKAN's temporal representation capabilities allow it to effec-

tively capture complex temporal dependencies and nonlinear patterns in water salinity data. It also shows improved stability across various configurations, making it suitable for practical deployment in real-time environmental monitoring systems.

**Future Work** While this study demonstrates the strong potential of the Temporal Kolmogorov-Arnold Network (TKAN) for salinity prediction in the Saigon River, several directions remain open for future exploration. One important area is the incorporation of external variables such as rainfall, tidal levels, upstream discharge, and weather conditions, which may significantly influence salinity patterns and improve forecasting accuracy, particularly for longer horizons.

Additionally, extending the model to support multi-step and probabilistic forecasting would allow it to better handle uncertainty and provide more informative predictions for decision-making in water treatment operations.

Improving interpretability is another key direction; visualizing attention weights or applying feature attribution techniques could offer insights into how the model makes its predictions.

Moreover, integrating TKAN into a real-time monitoring and prediction system would be valuable for operational use, requiring considerations for system latency, data stream processing, and model retraining strategies. Future studies may also compare TKAN against other emerging deep learning architectures, such as Transformer-based models or Neural ODEs, to better understand its strengths and limitations in diverse forecasting contexts.

Another potential avenue involves the use of **Bayesian Optimization** for automated hyperparameter tuning. In this study, we attempted to apply Bayesian Optimization to search for optimal values of TKAN’s key parameters, such as the number of hidden units, sub-KAN dimensions, and learning rate. While the method is theoretically powerful, our experiments did not show a clear improvement over manually tuned configurations. This may be due to the limited data size, complex search space, or sensitivity of the model to initialization. Future work may consider enhancing the effectiveness of this approach by exploring larger datasets, alternative optimization strategies such as Tree-structured Parzen Estimators (TPE), or combining domain knowledge with automated search. These strategies could make tuning more efficient and ultimately lead to better model performance in practice.

Collectively, these research directions aim to enhance the accuracy, robustness, and deployability of deep learning-based salinity forecasting systems in dynamic and data-scarce environments.

Overall, this work highlights TKAN as a powerful and reliable deep learning model for time series forecasting in the context of environmental data. Future research may explore its integration with external factors such as rainfall, tidal data, or upstream discharge rates to further enhance prediction accuracy in dynamic river systems like the Saigon River.

## References

- [1] R. Genet and G. Inzirillo, *Temporal Kolmogorov-Arnold Networks: Temporally Recurrent Kolmogorov-Arnold Networks*, arXiv preprint arXiv:2405.07344, 2024.
- [2] T. Huynh and K. Do, *Water Quality Forecasting Using Deep Learning Models: A Case Study on Salinity in the Saigon River*, Ho Chi Minh City University of Technology, 2024.
- [3] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
- [4] S. Bai, J. Z. Kolter, and V. Koltun, *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*, arXiv preprint arXiv:1803.01271, 2018.
- [5] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd edition, OTexts, 2021. Available at: <https://otexts.com/fpp3/>
- [6] W. J. Conover, *Practical Nonparametric Statistics*, 3rd ed., Wiley, 1999.
- [7] Z. Liu, Z. Li, M. Elhoseiny, and B. Zhou, *Kolmogorov-Arnold Networks*, arXiv preprint arXiv:2404.19756, 2024.
- [8] A. Mitropolitsky, D. Krotov, *Kolmogorov-Arnold Networks for Time Series: Bridging Predictive Power and Interpretability*, arXiv preprint arXiv:2406.02496, 2024.
- [9] L. Pan, Z. Liu, Z. Li, et al., *Are KANs Effective for Multivariate Time Series Forecasting?*, arXiv preprint arXiv:2408.11306, 2024.
- [10] L. Pan, Z. Liu, et al., *Kolmogorov-Arnold Networks (KANs) for Time Series Analysis*, arXiv preprint arXiv:2405.08790, 2024.
- [11] J. Ren, D. Krotov, *A Temporal Kolmogorov-Arnold Transformer for Time Series Forecasting*, arXiv preprint arXiv:2406.02486, 2024.
- [12] T. H. Nguyen, T. V. Bui, et al., *Application of artificial intelligence for forecasting surface quality index of irrigation systems in the Red River Delta, Vietnam*, Environmental Systems Research, vol. 13, no. 15, 2024.
- [13] J. Liang, Y. Zhao, et al., *Water Pollution Prediction Based on Deep Belief Network in Big Data of Water Environment Monitoring*, Scientific Programming, vol. 2021, Article ID 6689844, 2021.
- [14] Y. Gong, et al., *The Utility of Machine Learning Models for Predicting Chemical Contaminants in Drinking Water: Promise, Challenges, and Opportunities*, Current Environmental Health Reports, vol. 10, 2023.



- [15] S. A. Bak, et al., *Prediction of long-term water quality using machine learning enhanced by Bayesian optimisation*, Environmental Modelling Software, vol. 168, 2024.
- [16] Y. Bengio, P. Simard, and P. Frasconi, *The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions*, IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 1994.
- [17] D. Kohli, A. Goyal, *Optimal Ratio for Data Splitting*, arXiv preprint arXiv:2202.03326, 2022.
- [18] Z. Zhao, W. Chen, et al., *Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Forecasting Network-wide Traffic State with Missing Values*, arXiv preprint arXiv:2005.11627, 2020.