Original article

Attention-based hybrid convolutional-long short-term memory network for bridge pier hysteresis and backbone curves prediction

Integrated Computer-Aided Engineering
1–20

DOI: 10.1177/10692509241301240
journals.sagepub.com/home/ico

Omid Yazdanpanah[1], Minwoo Chang[2] and Ensieh Ali Bakhshi[2]

Abstract

This paper proposes a solution to the problem of automatically predicting hysteresis and backbone curves of bridge piers under seismic loads. The proposed solution utilizes a stacked hybrid Convolutional Neural Network-bidirectional Cuda Deep Neural Network Long Short Term Memory layer benefiting from the skip connections technique and incorporates a custom task-specific attention layer to enhance its performance. The proposed framework borrows the functional API provided by the Keras library in Python to construct a model taking into account horizontal and vertical ground accelerations, actuator loads in both horizontal and vertical directions, the effective pier height, the second moment of area, and the superstructure mass as input features. The deep learning model demands a substantial amount of data for effective training, validation, and testing. An error-sensitive analysis suggests that a comprehensive dataset should consist of a minimum of 12 sets of pier data for real-time hybrid simulations and 17 sets for cyclic experiments (10 for high-speed and seven for low-speed scenarios). This extensive dataset is deemed essential for the optimal performance of the model. The same deep learning framework and optimization of hyperparameters apply when training real-time hybrid simulations and conducting cyclic tests. After 5000 epochs, the proposed hybrid loss function, combining mean square and mean absolute errors, exhibits a steady and gradual decrease toward near-zero values within the datasets used for training and validation. Additionally, over 93% correlation exists between the predicted unseen time series responses and those derived from empirical measurements. Overall, the proposed deep learning model offers significant advantages, notably in terms of time and cost savings associated with experimental endeavors for new tests. By providing a rapid and accurate understanding of the hysteretic behavior of bridge piers, this model contributes to more efficient bridge design processes. Ultimately, it facilitates precision in design decisions, leading to enhanced accuracy and effectiveness in bridge engineering.

Keywords

Bridge piers, CNN, bidirectional CuDNNLSTM network, task-specific attention, skip connections, hybrid loss, predicted hysteresis, and backbone curves

Received: 4 July 2023; accepted: 10 September 2024

1 Introduction

Hysteresis and backbone curves are two important concepts in the analysis and design of bridge piers. These curves can be used to predict the behavior of the pier under different loading conditions, such as those caused by earthquakes, to identify potential failure modes, maintenance or retrofitting strategies, and to optimize the design of the pier to ensure that it is able to withstand the expected loading conditions over its design life.[1–5]

Today's deep learning architectures encompass a variety of advanced models and techniques that were developed to tackle complex tasks in various domains.[6–14] The versatile progress in this field has the potential to revolutionize the understanding of bridge piers' behavior and performance.[15] By leveraging deep learning, engineers and researchers can automate and enhance the analysis of bridge piers through their corresponding historical response prediction, leading to deeper insights and improved outcomes. However,

[1]Hybrid Structural Testing Center (Core Research Center for Smart Infrastructure), Myongji University, Yongin-si, Republic of Korea
[2]Department of Civil and Environmental Engineering, Myongji University, Yongin-si, Republic of Korea

Corresponding authors:
Omid Yazdanpanah, Hybrid Structural Testing Center (Core Research Center for Smart Infrastructure), Myongji University, Yongin-si, Republic of Korea.
Email: omidyazdanpanah@mju.ac.kr

Minwoo Chang, Department of Civil and Environmental Engineering, Myongji University, Yongin-si, Republic of Korea.
Email: cmw321@mju.ac.kr

there is indeed a significant interest in a vast amount of training data for deep learning techniques. Training data plays a crucial role in training deep neural networks, enabling them to thrive and learn patterns, extract features, make accurate predictions, and better generalize to unseen data.[16–23]

Efforts can be made to collect data from various sources relevant to bridge piers. This can include sensor data such as accelerometers, strain gauges, or displacement sensors, from monitoring simulated models or laboratory test setups. Various testing methods can be employed to record bridge pier responses under different loading conditions, including fast and slow cyclic tests with different loading rates, as well as more advanced and accurate techniques such as real-time hybrid simulations (RTHS) or shaking table tests.[16,17]

Cyclic tests are valuable in investigating the seismic response of structures by utilizing servo-hydraulic actuators to impose a predetermined displacement history onto a test specimen.[24,25] However, slow cyclic tests have limitations in capturing the loading rate-dependent behavior and structural failure mechanism under dynamic loads at fast speeds.[26,27] To conquer the challenges associated with the fast cyclic test under axial load, Chae et al.[28] proposed a novel method that combines a displacement-adaptive time series (D-ATS) compensator using a flexible loading beam (FLB) and an adaptive time series (ATS) compensator.[29] This approach enables the successful application of a constant axial force to reinforced concrete (RC) bridge piers during fast cyclic tests. By employing a setup where the two actuators in vertical directions are linked to the FLB, the method resolves the issue of determining force boundary conditions for axially stiff members like piers and walls.

Shaking table tests can be highly accurate in capturing the structural responses of bridge piers under seismic excitations. However, they often require the full-scale structure to be constructed on the shaking table, which can be costly and impractical for large structures. Instead, RTHS offers a cost-effective alternative for assessing the seismic response of large structures, including bridge piers. In RTHS, only the critical substructure or component of the bridge, i.e. pier, is physically tested, while the rest of the structure is simulated numerically using computational models. The experimental results are integrated with real-time computer simulations to capture the comprehensive seismic behavior of the complex original structure at a global level.[28,30,31]

Various methods were explored to predict the dynamic behavior exhibited by civil infrastructures when subjected to ground motions, including multifractal dimensions,[32] symbolic and Bayesian regressions,[33–37] ARIMA,[38,39] SVM,[40] MLP networks,[41] CNNs,[42,43] and long short-term memory (LSTM) networks.[24–45] Nevertheless, each method has its limitations. Multifractal dimensions and symbolic regression can only predict the peak drift ratio and not the entire time history response. ARIMA is limited by linearity and stationarity. MLP and CNN models struggle with high nonlinearity and plastic deformations.[46] SVM treats each row as a distinct training sample and predicts outcomes independently of prior patterns. Its performance tends to excel when working with smaller datasets.[47] Although LSTM networks offer improvements over traditional recurrent neural networks (RNNs),[48–51] they can be computationally expensive and prone to overfitting. Compared to traditional LSTM, Bidirectional LSTM (Bi-LSTM) layer has shown improved performance in predicting nonlinear time series data. By considering the past and future context simultaneously, the Bi-LSTM can better capture long-term dependencies and make more accurate predictions.[16,17,50,52] However, using a Bi-LSTM layer comes with some computational and memory costs.[17] CuDNN (CUDA Deep Neural Network) is a library provided by NVIDIA that optimizes deep neural network computations for GPUs.[53] A Bidirectional CuDNNLSTM (Bi-CuDNNLSTM) network would utilize a CuDNN-accelerated implementation of the LSTM layer to process the input sequence bidirectionally. This combination can enhance the model's ability to capture complex temporal patterns, dependencies, and relationships in time series data.[17]

The primary objective of this study is to develop deep learning networks that are not only more accurate but also more robust in predicting the time history responses of bridge pier under RTHS and cyclic tests and their corresponding hysteresis and backbone curves, with the ability to generalize well to unseen data. The aim is to overcome the limitations of existing methods and enhance the predictive capabilities of the models. This study aims to provide reliable and efficient predictions for various applications and scenarios by improving the accuracy and robustness of deep learning networks. The first major contribution of this paper involves the utilization of a stacked convolutional neural network (CNN)-Bi-CuDNNLSTM network, which takes advantage of the skip connections technique, also known as residual connections[54] (the second contribution). The rationale behind employing CNN followed by Bi-CuDNNLSTM layer lies in the fact that CNN aids in extracting intricate features, while the Bi-CuDNNLSTM network identifies relationships among these extracted features. The utilization of CuDNNLSTM serves the purpose of expediting the learning process. This network architecture is further enhanced by incorporating a custom task-specific attention layer (the third contribution) to selectively concentrate on the most pertinent segments of the input sequence when generating predictions.[55] Another significant contribution of this paper is the introduction of a hybrid loss function, which combines the mean square error (MSE) and mean absolute error (MAE). The benefit of using a hybrid loss function is that it can leverage the strengths of both MSE and MAE. The other major highlights of this paper are using the exponential learning rate scheduler instead of a constant learning rate, the weight decay regularization technique, and causal CNN. Furthermore, an

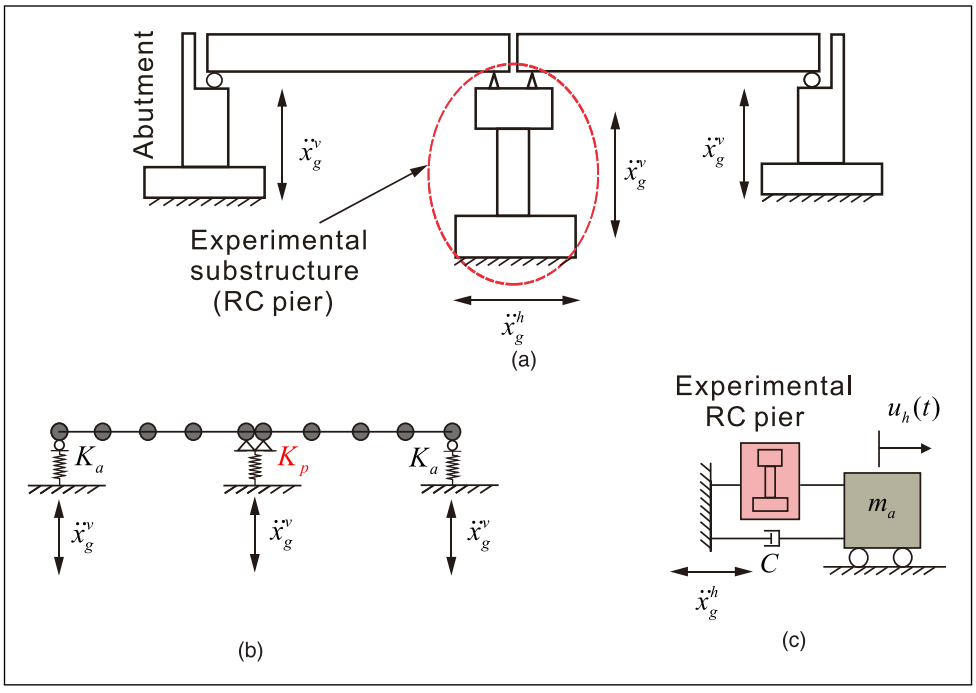

Figure 1. Selected bridge structure for RTHS, fast and slow cyclic tests. (a) A bridge structure with two spans designed for RTHS. (b) Analytical modeling for simulating vertical vibrations in the bridge structure. (c) Horizontal RTHS model.

error-sensitive analysis is made for selecting the minimum number of samples for training datasets helping to optimize resource utilization, reduce the overall cost associated with data acquisition, and avoid overfitting. The key takeaway from the paper is that these additions aim to improve the overall performance of the deep learning network, enhancing its ability to learn and make accurate predictions of unseen datasets.

2 Overview of experimental databases

Figure 1 depicts the structure of a 2-span bridge for the tests, showing ground accelerations in both the horizontal and vertical directions. The RC pier is scaled and manufactured in accordance with the guidelines set by the Korea Road and Transportation Association.[56] The bridge pier holds greater significance than the superstructures (deck and girders), due to the behavior influenced by the loading rate and the nonlinearity demonstrated in its deformations primarily concentrated at its base. Equation (1) is employed to compute the bridge response in the horizontal direction. To address the discrepancy in axial force caused by a minor deviation in axial displacement, Chae et al.[28,29] resolved the issue by utilizing the stiffness of the pier under axial loads during its initial phase and transforming the control issue from displacement-based to force-based control (Figure 1(c)). The abutment and RC pier stiffness coefficients, as well as the damping coefficient of roller supports, are denoted by K_a, K_p, and C, respectively, refer to Figure 1(b) and (c).

$$m_a\ddot{u}_h + c\dot{u}_h + R_p = -m\ddot{x}_g^h, \quad m_a = m - m_e. \tag{1}$$

where m_a, u_h, and m represent the analytical mass, horizontal displacement, and total mass of the bridge superstructures, respectively. R_p is the experimental restoring force from the RC pier. m_e, c, $\dot{u}_h$ and $\ddot{u}_h$ stand for experimental mass associated with the test setup, the system damping coefficient, velocity, and acceleration of the bridge superstructures, respectively. $\ddot{x}_g^h$ signifies the horizontal ground acceleration.

Figure 2 displays experimental setups for RTHS, fast, and slow cyclic tests at Hystec. The setups for the datasets in 2017 and 2018 are derived from the work of Chae et al.,[26,57] including pier section characteristics, nominal and effective heights, and test configurations. Additionally, 13 new tests were conducted in 2022 to examine the changes in various factors such as vertical ground acceleration, cross-sections, axial loads, superstructure masses, and heights. A compliance spring known as FLB is positioned at the upper part of the bridge piers to facilitate the application of axial force, further information can be found in.[26,28,57,58] The details of these tests are listed in Tables 1 and 2. It should be noted that in order to

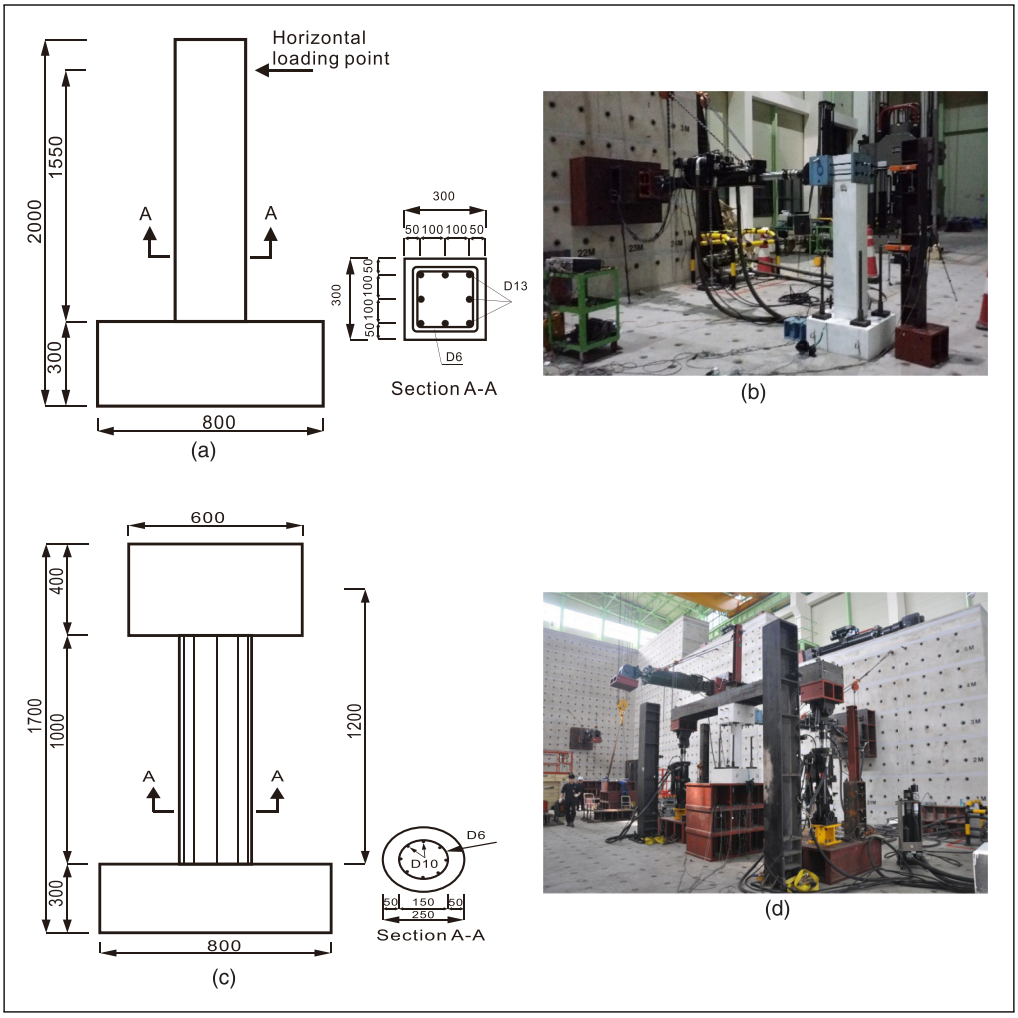

Figure 2. Experimental configurations established at Hystec. (a) Square section RC pier specimen (unit: mm). (b) Experimental test setup (2017). (c) Circular section RC piers specimen (unit: mm). (d) Experimental test setup (2017). (e) Circular section RC piers specimen (unit: mm). (f) Experimental test setup (2018). (g) Square section with vertical load (unit: mm). (h) Experimental test setup (2022). *(continued)*

minimize the disturbance caused by the oil column resonance, a low-pass Butterworth filter with a cutoff frequency of 5 Hz is utilized on the horizontal and vertical actuator forces. In addition, to achieve consistent lengths for all time series data available in the database, a data augmentation technique called padding with linear interpolation is employed for certain sequences with shorter lengths. This approach involves introducing variations within the sequence while still preserving the overall trend and characteristics of the original data. Implementing this technique ensures consistent lengths for all time series data, increases training data diversity, and removes potential outliers.

3 The proposed deep learning network

Experimentation and fine-tuning may be necessary to find the optimal combination of architecture, loss function, training algorithm, hyperparameter tuning, and data preprocessing techniques to create a neural network that is well-suited to the task at hand and more likely to achieve good performance for unseen data. A detailed description of the algorithms used in the proposed attention-based stacked CNN-bidirectional CuDNNLSTM network is provided in the following subsections:

3.1 Task-specific attention

Task-specific attention refers to a mechanism in machine learning models, particularly in sequence data such as time series or natural language processing (NLP), enabling the model to concentrate its attention on specific segments of the input data that are pertinent to the given task.[59,60] Attention layers are a form of attentional interface that allow neural networks to attend to different parts of the input sequence during processing,[61] and task-specific attention is

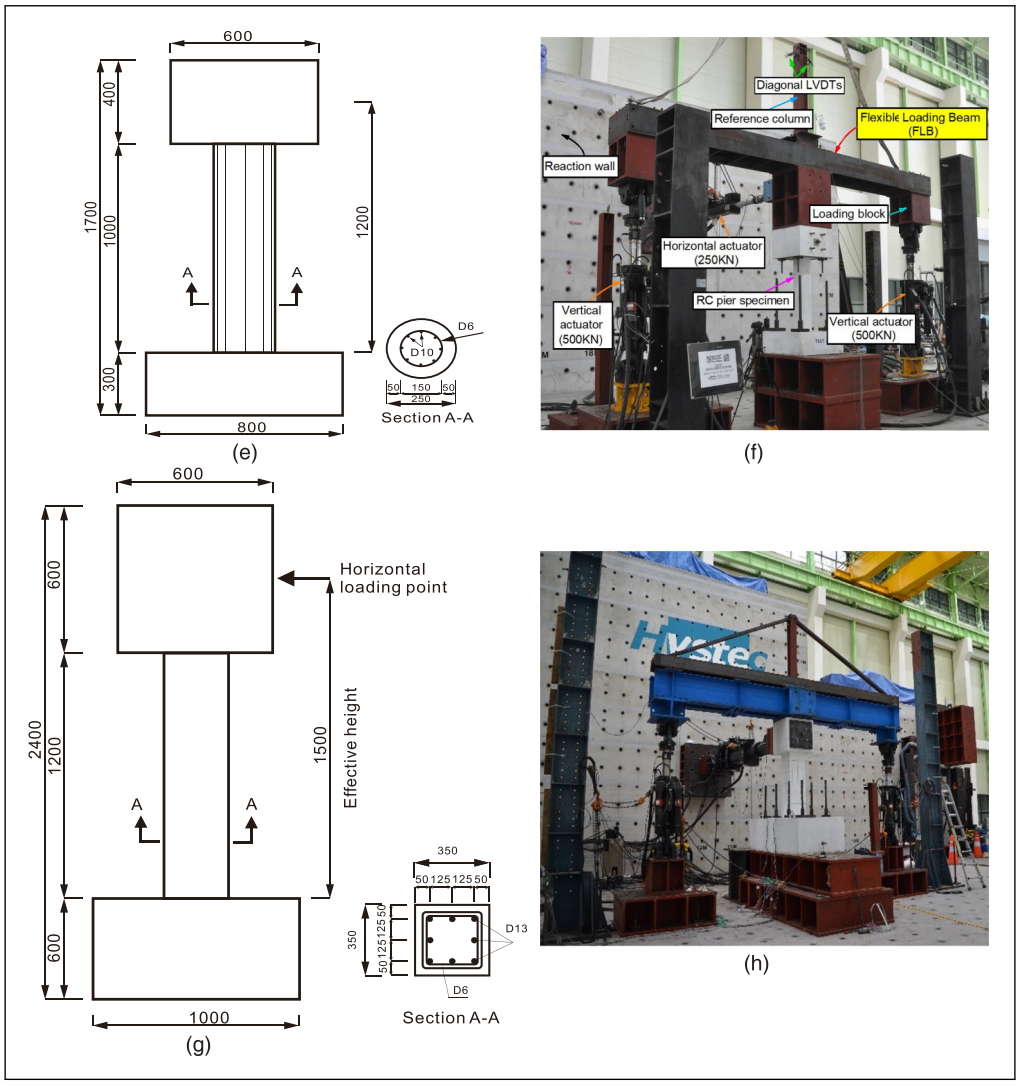

Figure 2. (Continued)

Table 1. The attributes of the experimental configurations for RTHS.

Specimen #	Horizontal Ground Acc. (m/s^2)	Vertical Load (N)	Vertical Ground Acc. (m/s^2)	Effective Height (m)	Moment of Inertia (m^4)	Mass (kg)	Year	Data Splitting
1	GA_{h1} (0.1)	14.3×10^4	GA_v (0.1)	2	1.917×10^{-4}	1019.7	2018	Train
2	GA_{h1} (0.35)	14.3×10^4	GA_v (0.35)	2	1.917×10^{-4}	1019.7	2018	
3	GA_{h1} (0.5)	14.3×10^4	GA_v (0.5)	2	1.917×10^{-4}	1019.7	2018	
4	GA_{h1} (0.1)	0	0	2	1.917×10^{-4}	1019.7	2018	
5	GA_{h1} (0.35)	0	0	2	1.917×10^{-4}	1019.7	2018	
6	GA_{h1} (0.5)	0	0	2	1.917×10^{-4}	1019.7	2018	
7	GA_{h2} (0.1)	0	0	1.55	6.75×10^{-4}	6614	2017	
8	GA_{h1} (0.30)	39.2×10^4	0	1.50	1.25×10^{-3}	8×10^4	2022	Validation
9	GA_{h1} (0.30)	39.2×10^4	GA_v (1.0)	1.50	1.25×10^{-3}	8×10^4	2022	
10	GA_{h1} (0.30)	0	0	1.50	1.25×10^{-3}	8×10^4	2022	
11	GA_{h1} (0.30)	39.2×10^4	GA_v (0.3)	1.50	1.25×10^{-3}	8×10^4	2022	Test (unseen)
12	GA_{h1} (0.30)	39.2×10^4	0	1.50	1.25×10^{-3}	8×10^4	2022	

Table 2. The properties of the cyclic tests with fast and slow loading rates.

Pier #	Test Method	Maximum Cyclic Displacement (m)	Vertical Load (m·kg·s-2)	Effective Height (m)	Moment of Inertia (m4)	Mass (kg)	Year	Data Splitting
1	Slow cyclic	100×10^{-3}	14.5×10^{4}	1.2	1.9175×10^{-4}	6614	2017	Train
2	Slow cyclic	82.5×10^{-3}	2×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
3	Slow cyclic	82.5×10^{-3}	4×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
4	Fast cyclic	166.7×10^{-3}	18×10^{4}	2	1.917×10^{-4}	1019.7	2018	
5	Fast cyclic	166.7×10^{-3}	10.6×10^{4}	2	1.917×10^{-4}	1019.7	2018	
6	Fast cyclic	20×10^{-3}	15×10^{4}	1.2	1.9175×10^{-4}	6614	2017	
7	Fast cyclic	82.5×10^{-3}	2×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
8	Fast cyclic	100×10^{-3}	14.5×10^{4}	1.2	1.9175×10^{-4}	6614	2017	
9	Fast cyclic	82.5×10^{-3}	4×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
10	Slow cyclic	166.7×10^{-3}	10.4×10^{4}	2	1.917×10^{-4}	1019.7	2018	Validation
11	Slow cyclic	82.5×10^{-3}	6×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
12	Fast cyclic	100×10^{-3}	14.5×10^{4}	1.2	1.9175×10^{-4}	6614	2017	
13	Fast cyclic	82.5×10^{-3}	6×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
14	Slow cyclic	166.7×10^{-3}	3.2×10^{4}	2	1.917×10^{-4}	1019.7	2018	Test (unseen)
15	Slow cyclic	82.5×10^{-3}	8×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
16	Fast cyclic	82.5×10^{-3}	8×10^{5}	1.5	1.25×10^{-3}	8×10^{4}	2022	
17	Fast cyclic	166.7×10^{-3}	3.2×10^{4}	2	1.917×10^{-4}	1019.7	2018	

a type of attention layer that is designed to help the model perform a specific task,[62–67] here time series response prediction. This approach avoids attempting to fit the entirety of the data, thereby enhancing the model's generalization capabilities and mitigating the risk of overfitting. Additionally, the attention layer can help to identify important features of the data, which can lead to more effective feature selection and dimensionality reduction.

The custom TaskSpecificAttention layer, presented in Appendix A, calculates "soft" weights for each time step in the context window, which can be computed sequentially. Soft weights can change during each runtime, in contrast to "hard" weights, which are pre-trained and fine-tuned and remain frozen afterward. The layer has three trainable weights: W, b, and u. The W weight is used to transform the input tensor into a tensor of shape (batch_size, time_steps, units). The b weight is used to add a bias term to the transformed tensor. The u weight is used to compute the attention scores for each time step in the input tensor. The attention scores are computed by taking the dot product of the transformed input tensor and the u weight. The attention scores are then passed through a softmax activation function to obtain the attention weights. The attention weights are then used to compute a weighted sum of the input tensor. If return_sequences is True, the layer returns a tensor of shape (batch_size, time_steps, features). If return_sequences is False, the layer returns a tensor of shape (batch_size, features). The activation parameter specifies the activation function to use, set to ELU, the Exponential Linear Unit activation function. The TaskSpecificAttention layer can be used with CNN and bidirectional CuDNNLSTM layers to improve the performance of deep learning models.

The equations defining the TaskSpecificAttention layer are presented in Equations (2)–(5):

$$u_{it} = \sigma(Wx_{it} + b) \tag{2}$$

$$a_{it} = u_{it}^{T} u \tag{3}$$

$$\alpha_{it} = \frac{e^{a_{it}}}{\sum_{j=1}^{N} e^{a_{ij}}} \tag{4}$$

$$c_i = \sum_{t=1}^{N} \alpha_{it} x_{it} \tag{5}$$

in which x_{it}, W, b, u, σ, u_{it}, a_{it}, α_{it}, c_i, and N stand for the input tensor at time step t and feature i, the weight matrix, the bias vector, the attention vector, the activation function, the transformed input tensor at time step t and feature i, the attention score for time step t and feature i, the attention weight for time step t and feature i, the context vector for feature i, and the total number of time steps in the input tensor, respectively.

3.2 Convolutional neural network (CNN) with causal padding

Causal padding is a technique used in one-dimensional CNNs (Conv1D layer) to preserve the temporal order of the input sequence. It is commonly used in tasks such as NLP or time series analysis, where the output at each time step should only depend on the previous time steps.[68,69] In a standard CNN, the padding is applied symmetrically to the input, which means that the same amount of padding is added before and after the input sequence. This is suitable for tasks where the output at each time step can

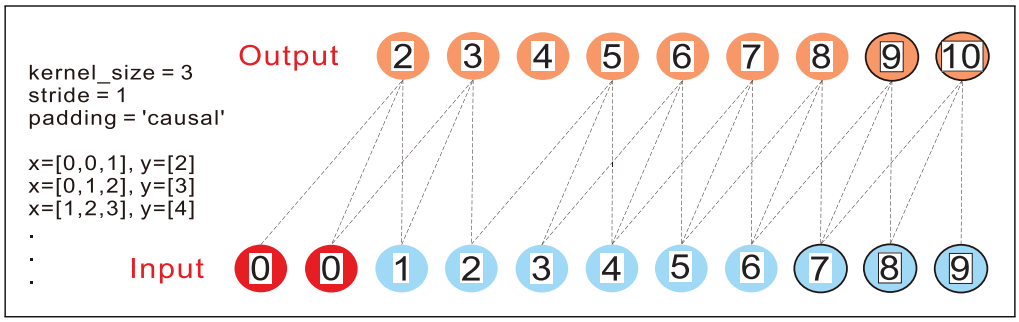

Figure 3. Causal convolution layer.

depend on both past and future time steps. However, in tasks where only the past time steps are relevant, such as sequence prediction, causal padding is necessary. Causal padding ensures that during the convolution operation, the filters only have access to the past values and cannot see the future values. This is achieved by adding padding only to the left side of the input sequence while leaving the right side unpadded. As a result, the size of the output sequence is the same as the input sequence, but the information flow is restricted to the past, find Figure 3 in detail. In this study, the parameters of filters, kernel_size, activation, and padding, are set as 100, 3, Exponential Linear Unit activation function (ELU), and causal, respectively. The causal Conv1D layer is complemented by Max Pooling[70,71] and UpSampling[72] layers, employed to respectively reduce the input representation size through downsampling and enhance spatial dimensions by incorporating available information. In this study, a pooling size of two is chosen for the investigation.

3.3 Bidirectional CuDNNLSTM network

A Bidirectional CuDNNLSTM (Bi-CuDNNLSTM) network is a type of RNN architecture that incorporates the use of bidirectional processing and the CuDNN implementation developed by NVIDIA for faster, more efficient, and optimized GPU-accelerated training.[17] The bidirectional aspect of the network means that it processes the input sequences in both forward and backward directions simultaneously, capturing a more comprehensive understanding of the input sequences, and then concatenates them to produce the final output sequence.[50] The default parameters for Bi-CuDNNLSTM remain unchanged, except for the "units" parameter which is set to 100. The optimization of hyperparameters for the CuDNNLSTM network relies on the trial and error processes through the GridSearchCV[73] technique implemented in the Python package. Initial tests in the parametric study revealed that adhering to the default parameters in the CuDNNLSTM network yields optimal predictions, with the exception of "units" and "return_sequences". As a result, it is recommended to apply the following configurations to the default parameters in the Python Keras CuDNNLSTM library[74]: units = 100 and return_sequences = True.

In this paper, connecting the output of two bidirectional CuDNNLSTM layers using skip (residual) connections,[54,75–77] by implementing element-wise addition using the Add() layer, can mitigate the degradation problem that arises when training very deep neural networks. This approach enhances the network's capability to capture intricate dependencies and patterns in the data, refer to Figure 4 for more detailed information.

3.4 Hybrid loss function and evaluation metrics

This paper benefits from a hybrid loss function that amalgamates the Mean Squared Error (*MSE*) and Mean Absolute Error (*MAE*) to leverage their respective strengths. *MSE* captures the overall trend but is sensitive to outliers, while *MAE* handles outliers better and captures local fluctuations. The hybrid loss function presented in equation (6) balances these aspects, leading to a more accurate and robust model for time series prediction.

$$\text{Hybrid loss} = w \times MSE + (1 - w) \times MAE. \tag{6}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{7}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{8}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}|y_i - \bar{y}_i|^2} \tag{9}$$

where w is a weight that controls the balance between the *MSE* and *MAE*. A value of $w = 0.5$ gives equal weight to both losses and is set for training the dataset. y_i, $\hat{y}_i$, $\bar{y}_i$, and n stand for the actual value of the ith observation, the predicted value of the ith observation, the mean of the actual values, and the number of observations, respectively.

During training and validation, the model will calculate and display *MSE*, *MAE*, and R^2, refer to equations (7)–(9), as evaluation metrics in addition to the loss value. This can help to monitor the performance of the model and gain insights into its ability to capture the patterns and trends in the time series data.

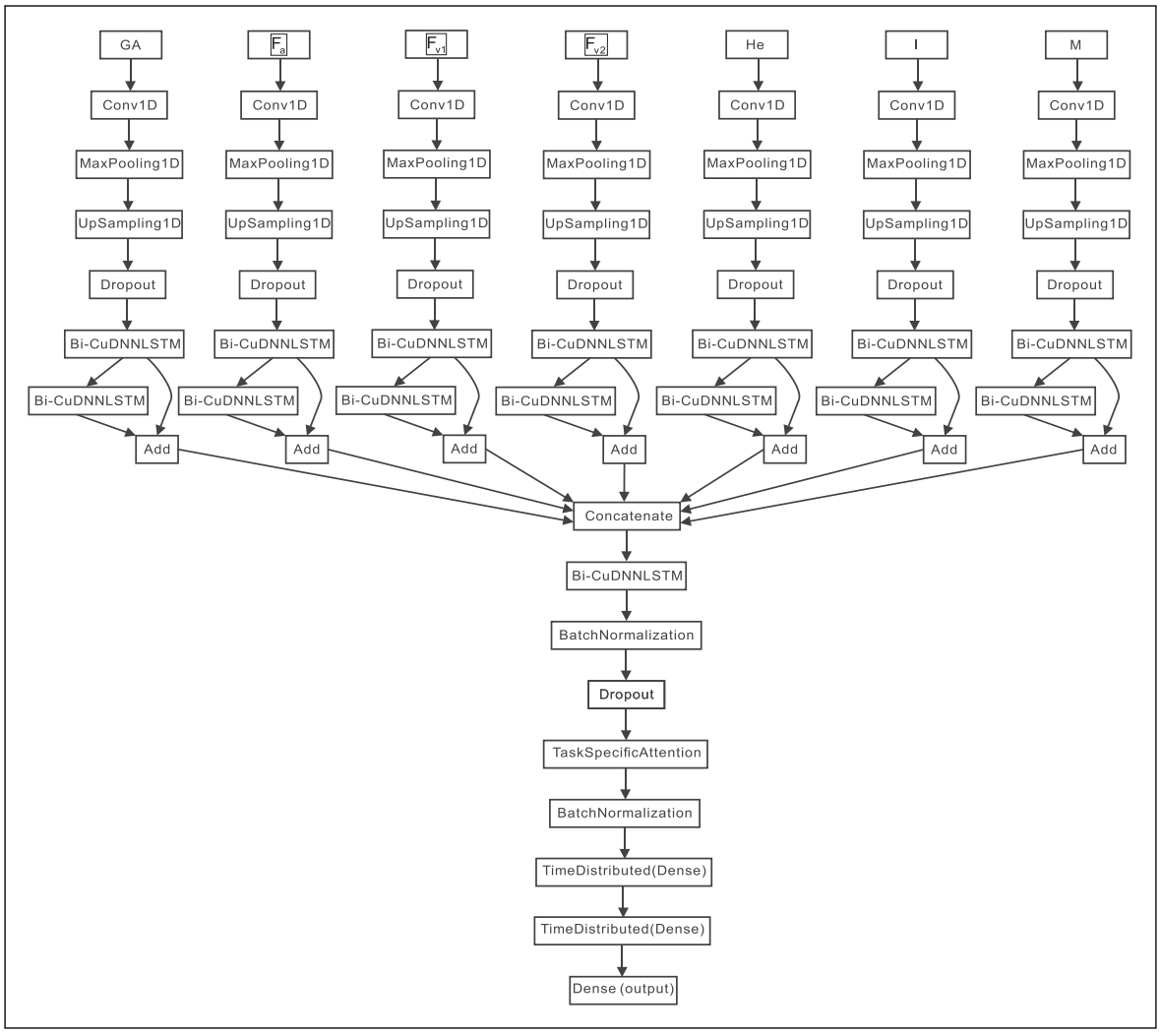

Figure 4. A plot of the proposed attention-based stacked CNN-bidirectional CuDNNLSTM model involving multiple inputs for RTHS tests.

3.5 Exponential linear unit (ELU) activation function

ELU avoids the dead neuron problem of Rectified Linear Unit (ReLU) activation function by defining

$$ELU(x) = \begin{cases} x, & \text{if} \quad x \geq 0 \\ \alpha \times (exp(x) - 1), & \text{if} \quad x < 0 \end{cases} \tag{10}$$

In equation (10) x refers to the input to the function and α is a hyperparameter that controls the slope of negative values.[78] This allows the flexibility to adjust the slope based on the problem at hand. In this study, α is set to 0.001. *ELU*, with its exponential nature for negative inputs, allows the units to capture both positive and negative information, potentially leading to a better representation of learning than *tanh* in certain situations. It also has a non-zero and non-contractive gradient for both positive and negative inputs that hinders vanishing gradients.[78] By alleviating the issues of dead neurons and oversensitivity to initial weights, ELU activation can lead to better generalization and reduced overfitting.

Batch normalization[79] and dropout[80] are commonly used regularization techniques in deep learning models to improve generalization and prevent overfitting. These two regularization techniques, when used together, can complement each other and enhance the model's ability to generalize well to unseen data, prevent overfitting, and improve the model's overall performance. The use of dropout and batch normalization for activations can vary depending on the specific task and the characteristics of the dataset. It is crucial to experiment with different configurations and evaluate their impact on the performance of the model to determine the most suitable approach for a specific task. In this study, a dropout rate of 0.3 is utilized.

The TimeDistributed Dense layer[81] is a wrapper layer that applies a fully connected (Dense) layer to each element of a sequence independently. It captures local patterns and dependencies within each time step, allowing for the modeling of sequential data. This layer is commonly used with recurrent layers to add flexibility and learn temporal patterns in the data.[16,17] This research utilizes two TimeDistributed(Dense) layers followed by a Dense layer as the output layer. The first TimeDistributed(Dense) layer has 100 units with the ELU activation function, and the second TimeDistributed(Dense) layer has 50 units with the ELU activation function as well.

The exponential decay learning rate scheduler ensures faster convergence, stable training, and improved generalization, resulting in superior model performance. It starts with a higher learning rate for rapid progress, gradually decreasing it to fine-tune the parameters and converge toward a better solution. This approach prevents instability, large oscillations, and overshooting. Additionally, it enhances the model's ability to generalize to unseen data by reducing sensitivity to minor fluctuations and capturing broader patterns.[82] The initial learning rate and decay constant are set to 0.001 and 0.005, respectively, find Appendix C for more details. Moreover, a weight decay of 0.0001 is utilized as a regularization technique during model training to mitigate overfitting, see Appendix D.

Finally, the proposed deep learning model to estimate the hysteresis and backbone curves of bridge piers under RTHS tests is presented in Figure 4. The GA, F_a, F_{v1}, F_{v2}, H_e, I, and M values are used as input features in this study. These parameters encompass the ground acceleration experienced in the horizontal direction, the force exerted by the actuator in the horizontal plane, the forces generated by the vertical actuators on both the right and left sides, pier effective height, the second moment of area, and the superstructure mass for the RTHS setup. It is important to note that the influence of the ground acceleration experienced in the vertical direction is taken into account for the vertical load. To satisfy the requirements of the LSTM network, the input uncertain variables need to be formatted as a 3D array with the shape of (samples or batch size, time steps, features).[44] Unscaled input variables in deep learning models, such as LSTM, can cause unstable learning and large errors. To mitigate this, the data is scaled using sklearn.preprocessing.MinMaxScaler, see equation (11), to range between $[-1, 1]$.[83] Furthermore, a stacked scheme proposed by Zhang et al.[46] is employed for the Bi-CuDNNLSTM network, enhancing the speed and precision of predictions of the time history responses for RTHS. GridSearchCV class is utilized in Python, along with the Keras library,[73] to optimize the window size, denoting the size of each stack, for deep learning models. The grid search process results in determining a window size of 40 as the optimal choice for achieving the best predictions. This is necessary due to the presence of a long sequence of time-series data, consisting of 43,520 data points for each variable. In this case, each input feature has a shape of (None, 1088, 40). A batch size of 64 is utilized in the training and validation processes of the RTHS tests. The step-by-step workflow of the suggested deep model is visualized in Figure 5, providing a detailed representation of the sequential process. In this figure, "s", "w", and "n" stand for the number of stacks, window size (the size of each stack), and the total number of time steps within each sequence.

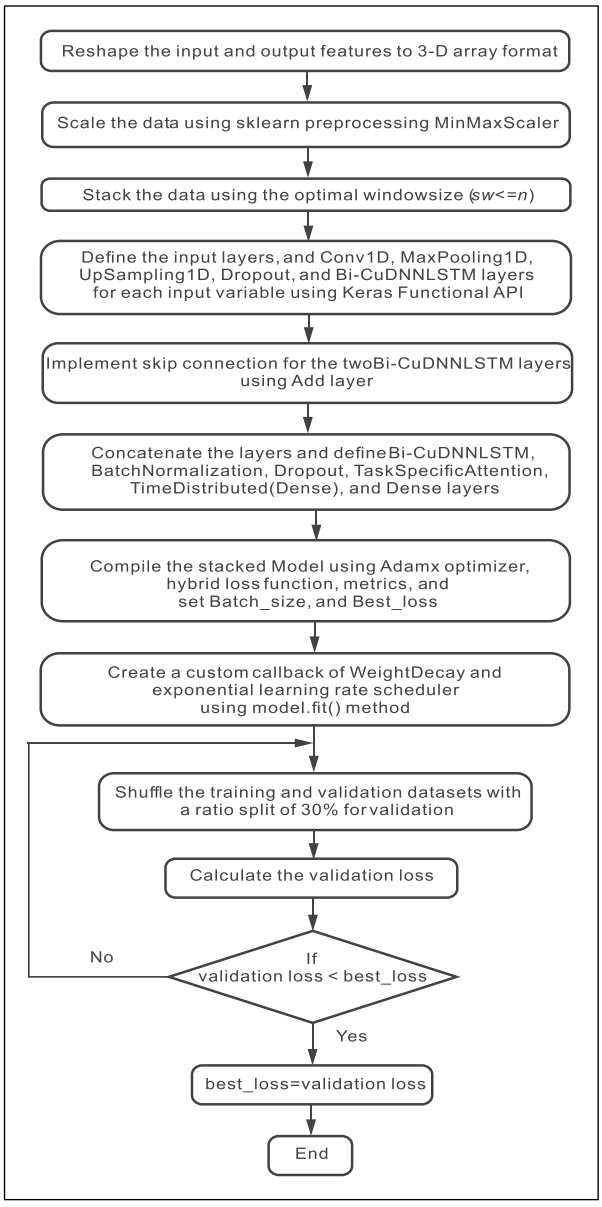

Figure 5. Flowchart depicting the proposed approach utilized for the research.

$$X_{scaled} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \times (\max - \min) + \min \qquad (11)$$

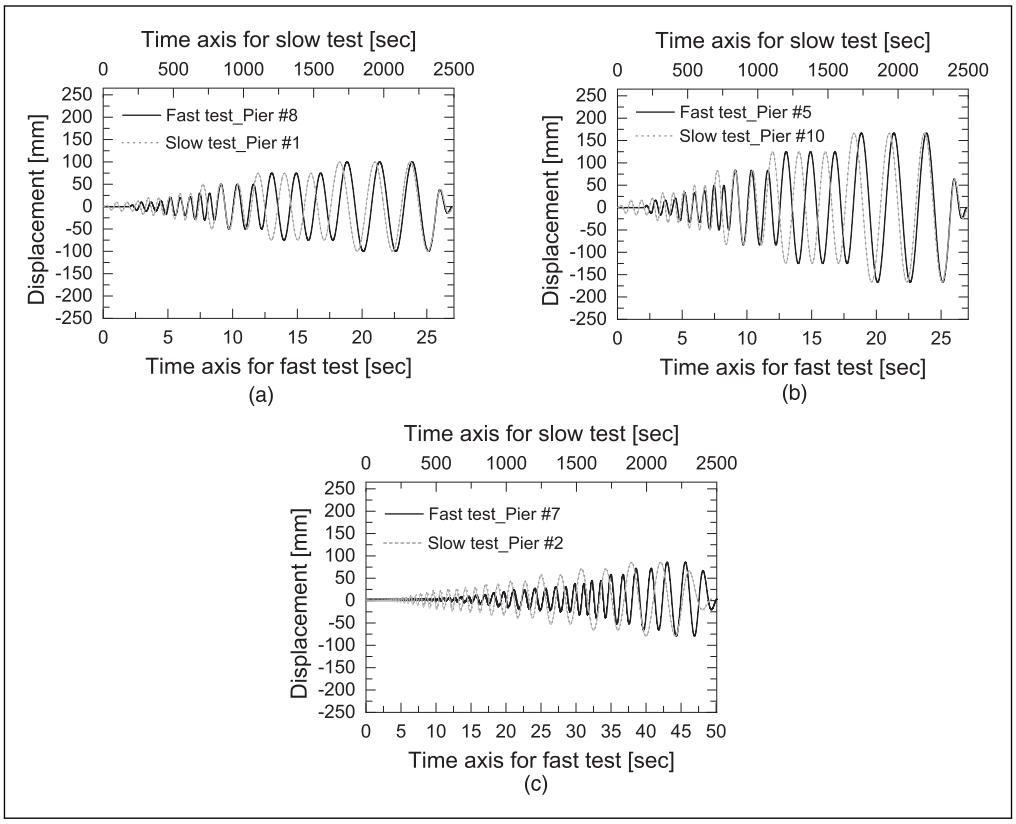

Figure 6. Displacement time histories for slow and fast tests. (a) Displacement time histories (2017). (b) Displacement time histories (2018). (c) Displacement time histories (2022).

where X, $X_{\min}$, and $X_{\max}$ are the original, minimum, and maximum values of the original data, respectively. min and max are the minimum and maximum values of the desired range, which are set here to -1 and 1, respectively.

A similar deep learning architecture is maintained with a minor modification for the fast and slow cyclic tests. The input feature *GA* is excluded, and an additional Task-Specific Attention layer is introduced right after the input layer for each variable. Moreover, a window size of 39 is chosen for data stacking, resulting in the shape of each input variable as (None, 1228, 39). Figure 6 provides additional details on predefined sinusoidal displacement time histories applied to the horizontal actuator. This includes variations in loading rates observed during fast and slow cyclic tests. Specifically, the slow tests conducted in 2017 are 91 times slower than their fast counterparts, while in 2018 and 2022, this discrepancy is 89 and 48 times, respectively.

4 Predicting hysteresis and backbone curves

It is evident from Figure 7 that the training and validation losses exhibit a decreasing trend, eventually reaching nearly zero after 5000 epochs. The batch size is set to 64. The following subsections discuss the prediction results of RTHS and cyclic tests.

A thorough analysis focusing on error sensitivity is conducted to determine the minimum number of samples required for the training datasets. As outlined in Tables 3 and 4 for RTHS and cyclic tests, respectively, four error measures are utilized to investigate how the minimum number of training samples should be. For RTHS, the error analysis starts from 4 to 7 training samples for the two unseen datasets. By increasing the number of training samples from 4 to 7, it can be inferred that the ratio between the maximum displacement in predictions and observations approximately leans towards one. In addition, there is an increase in R^2 and a decrease in the standard deviation (SD) measures of the normalized error, while the absolute mean of the normalized error does not follow a normal trend. The normalized estimation error (ε) is straightforwardly computed by equation (12) that becomes:

$$\varepsilon = \frac{Y_i^o - Y_i^p}{\max(|Y_i^o|)}, \quad i = 1 \dots N. \tag{12}$$

in which Y^o, Y^p, and i, stand for observation (experimental) response, prediction response, and the number of time steps, respectively.

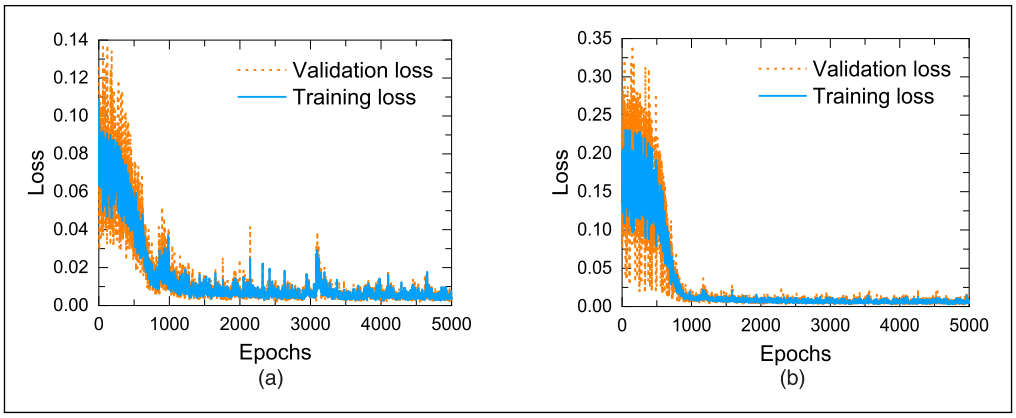

Figure 7. Variations observed in training and validation losses over time. (a) RTHS tests. (b) Fast and slow cyclic tests

Table 3. Error sensitivity analysis of unseen datasets to find the minimum number of training samples for RTHS tests.

Training samples #	| Mean |*	SD**	R^2	Maximum displacement ratio (prediction/observation)	Unseen data #
4	0.0679	0.0479	0.89	0.8015	11
4	0.0891	0.0533	0.8772	0.8087	12
5	0.0078	0.0722	0.7566	0.9397	11
5	0.0279	0.0811	0.7165	0.9509	12
6	0.0143	0.0382	0.9393	0.9962	11
6	0.0016	0.0406	0.929	1.01	12
7	0.0299	0.0287	0.961	0.9934	11
7	0.0104	0.0302	0.9595	1.008	12

*Absolute mean of the normalized error, ** Standard deviation.

Table 4. Error sensitivity analysis of unseen datasets to find the minimum number of training samples for cyclic tests.

Training samples #	| Mean |*	SD**	R^2	Maximum displacement ratio (prediction/observation)	Unseen data #
5	0.0138	0.0963	0.9626	0.8964	14
5	0.0252	0.1112	0.8835	0.6955	15
5	0.0236	0.1351	0.9423	1.227	16
5	0.0089	0.1298	0.9389	0.9083	17
6	0.0043	0.0954	0.9667	0.9362	14
6	0.0962	0.1222	0.864	0.8556	15
6	0.023	0.1593	0.9507	1.3306	16
6	0.0253	0.1037	0.9612	0.888	17
7	0.0019	0.087	0.9721	0.9087	14
7	0.0768	0.1051	0.8968	0.7582	15
7	0.0608	0.1253	0.9442	1.1124	16
7	7.38E-5	0.1136	0.9521	0.8933	17
8	0.0214	0.0889	0.9679	0.9395	14
8	0.0718	0.1011	0.9127	0.8753	15
8	0.012	0.1368	0.9421	1.3328	16
8	0.0411	0.1149	0.9588	0.9874	17
9	0.0031	0.1041	0.964	0.8824	14
9	0.0965	0.086	0.9349	0.9448	15
9	0.0081	0.1287	0.9452	1.211	16
9	0.0013	0.1174	0.9485	0.9263	17

*Absolute mean of the normalized error, ** Standard deviation.

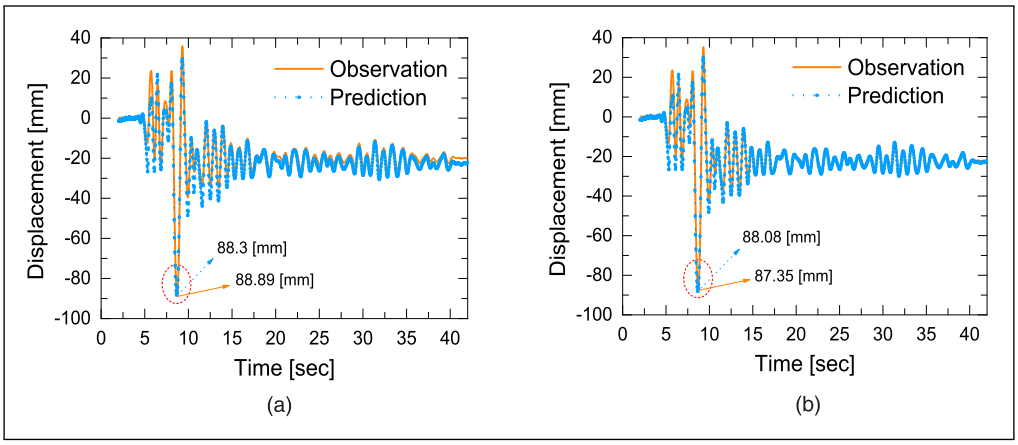

Figure 8. Comparative evaluation between the estimated and empirical displacement time series associated with the unseen dataset under RTHS tests. (a) Displacement time histories (pier 11). (b) Displacement time histories (pier 12).

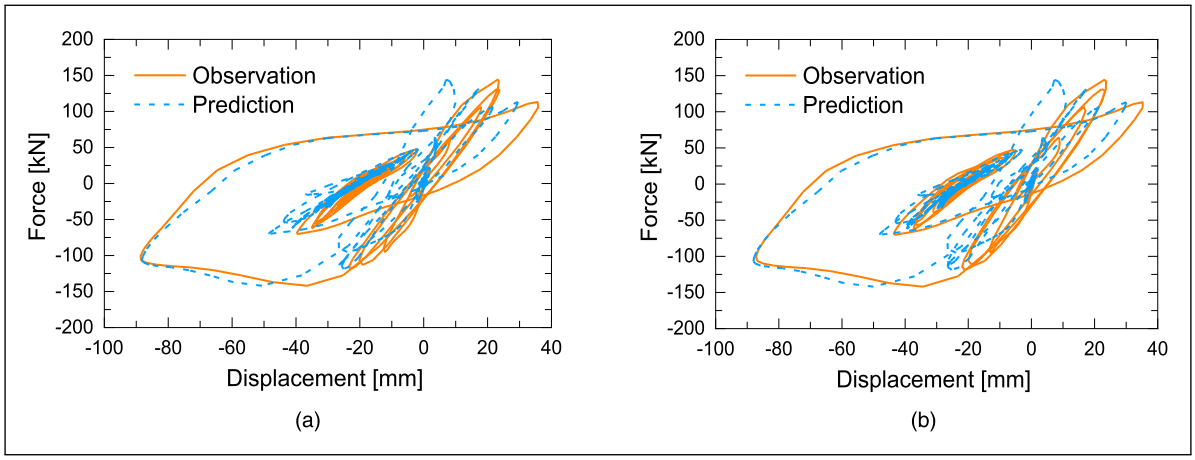

Figure 9. Comparing the performance of the estimated and empirical hysteresis curves concerning the unseen dataset under RTHS tests. (a) Hysteresis curves (pier 11). (b) Hysteresis curves (pier 12).

The same achievements remain valid for Table 4; increasing the number of training samples helps improve the unseen date predictions. It is important to mention that, in order to maintain a concise paper length, the discussion is focused solely on the results obtained from the unseen data. As outlined in Tables 3 and 4, the optimized training samples for both RTHS with seven samples and cyclic tests with nine samples yield a correlation surpassing 93% between the predicted unseen time series responses and those acquired from empirical measurements.

During RTHS tests, the performance of the estimated displacement time histories is evaluated by comparing them to the experimental displacement time histories of the unseen dataset. This allows for an evaluation of the accuracy and effectiveness of the predictions. Figure 8 showcases strong concurrence between the experimental and predicted unseen data, listed in Table 1, particularly with regard to the maximum displacement, which is of utmost significance in assessing the performance of bridge piers subjected to seismic excitations from an engineering perspective. The proposed deep learning model also shows a notable level of robustness in accurately predicting displacement time series for nonlinear regions. The findings become more interesting with the corresponding hysteresis curves. As revealed in Figure 9, the trend of hysteresis curves is satisfactorily predicted. There is much more improvement in the results presented in Figures 8 and 9 compared with those obtained in.[16,84] To assess the reliability of the proposed deep model in predicting time series responses of bridge piers, an additional dataset is allocated for cyclic tests, taking into account loading rate effects. The performance discrepancy between predicted and experimentally measured force-time histories for the four unseen datasets, summarized in Table 2 for slow and fast cyclic tests, is presented in Figure 10. Subsequently, the corresponding hysteresis and backbone curves are displayed in Figure 11, exhibiting significant improvements

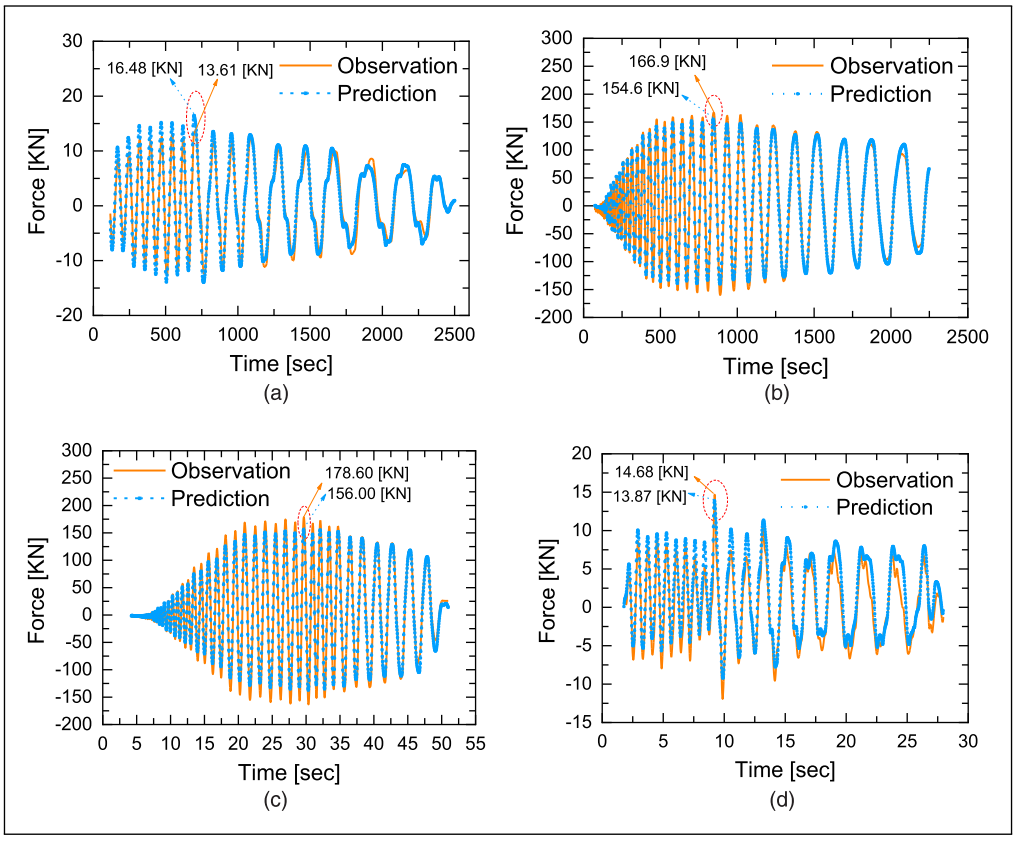

Figure 10. Assessing the performance discrepancy between predicted and experimentally measured force-time histories in an unseen dataset under slow and fast cyclic tests. (a) Force time histories (pier 14). (b) Force time histories (pier 15). (c) Force time histories (pier 16). (d) Force time histories (pier 17).

Table 5. Improvement in performance achieved by the proposed deep learning model for the maximum response.

	Proposed model	Ref[16]	Ref[84]	Ref[17]	Experimental maximum response
RTHS test #11	88.30 mm	78.84 mm	83.64 mm	—	88.89 mm
RTHS test #12	88.08 mm	78.48 mm	83.62 mm	—	87.35 mm
Slow cyclic test #14	16.48 KN	—	—	18.16 KN	13.61 KN
Slow cyclic test #15	154.60 KN	—	—	149.73 KN	166.90 KN
Fast cyclic test #16	156.00 KN	—	—	144.00 KN	178.60 KN
Fast cyclic test #17	13.87 KN	—	—	11.81 KN	14.68 KN

compared to the findings obtained in the previous study referenced as.[17] The detailed results are outlined in Table 5. As presented in the table, the performance of the proposed deep learning model in predicting the maximum displacement of unseen RTHS tests is compared with the results from Refs.[16,84] and the actual values obtained during the experimental endeavors. The findings demonstrate that the proposed model successfully pulls the verification tests off, with predictions showing strong agreement with the precise experimental values. The errors amount to only 0.66% and 0.84% for RTHS tests #11 and #12, respectively. Furthermore, there is a notable improvement of 10.64% and 9.31% for RTHS tests #11 and #12, respectively, compared to Ref.[16] In comparison to Ref.,[84] the improvement stands at 5.24% and 3.43% for the respective tests.

In the next trial, Table 5 provides the predicted maximum force of unseen cyclic tests #14, #15, #16, and #17 compared with those of Ref.[17] and the exact maximum force obtained from experiments. The enhancements observed for the individual cyclic tests are 12.34%, 2.92%, 6.72%, and 14.03%, respectively.

Furthermore, an additional examination is carried out to ascertain the training and validation runtime. As listed in Table 6, the runtime of RTHS tests for the proposed model, which incorporates Bi-CuDNNLSTM, is roughly half that of the same model utilizing Bi-LSTM. In the case of cyclic

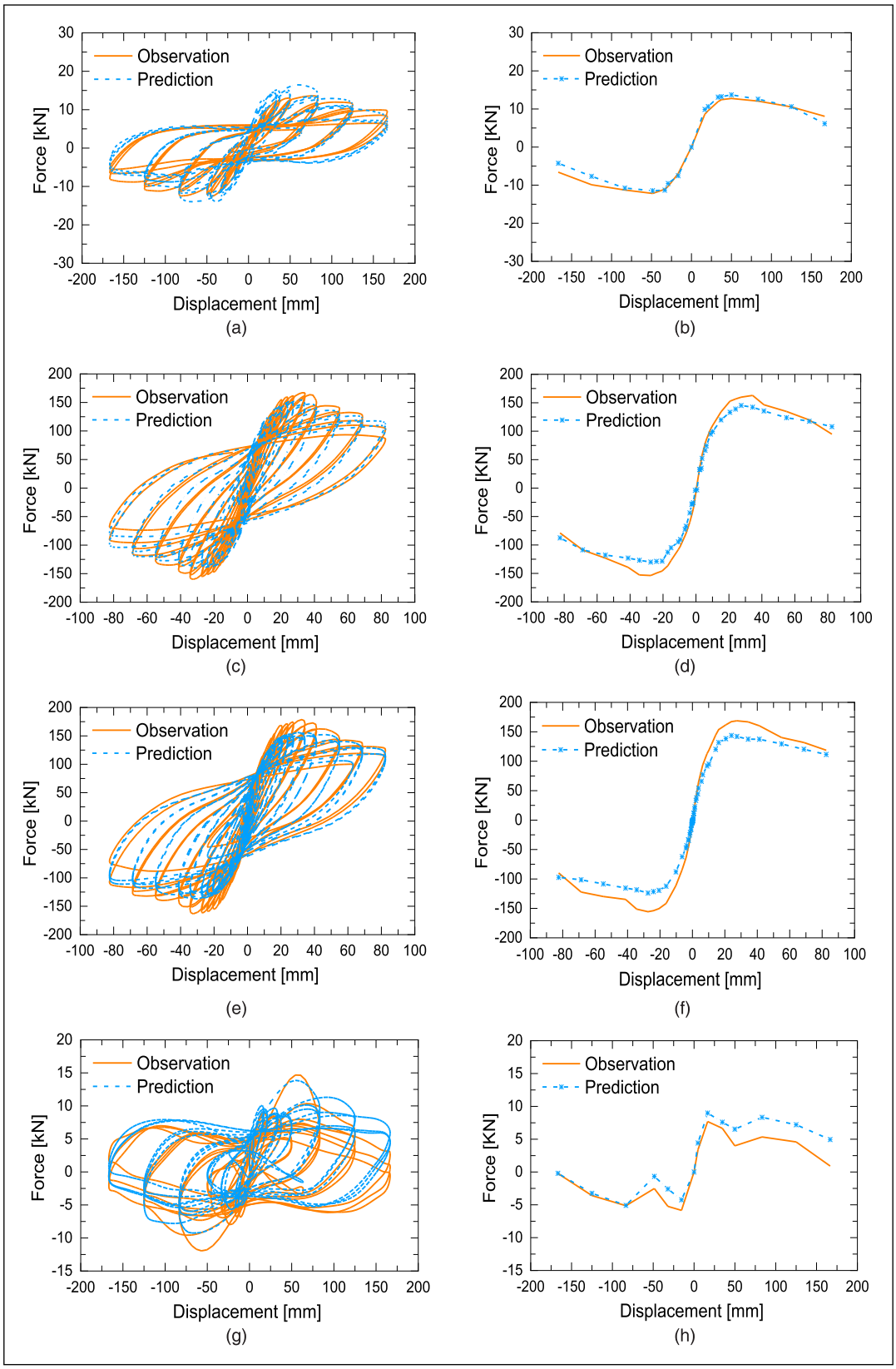

Figure 11. Comparative analysis of the estimated and empirical hysteresis and backbone curves relying on the unseen dataset for slow and fast cyclic tests. (a) Hysteresis curves (pier 14). (b) Backbone curves (pier 14). (c) Hysteresis curves (pier 15). (d) Backbone curves (pier 15). (e) Hysteresis curves (pier 16). (f) Backbone curves (pier 16). (g) Hysteresis curves (pier 17). (h) Backbone curves (pier 17).

Table 6. Network training runtime analysis (hour).

	The proposed model with Bi-CuDNNLSTM	The proposed model with Bi-LSTM
RTHS test	2.2352	4.6625
Slow and fast cyclic tests	2.5959	4.1973

tests, this translates to being 1.60 h faster compared to the corresponding model with Bi-LSTM.

5 Conclusions

Although deep learning has shown great promise in time series prediction, it does not completely negate the need for domain expertise. Understanding the underlying principles of time series analysis and having domain knowledge can still be valuable in tasks such as data preprocessing, feature selection, network selection, hyperparameter tuning, and interpreting the results. This is particularly beneficial in scenarios where data scarcity is a challenge. Combining deep learning techniques with domain expertise makes it possible to achieve more accurate and meaningful predictions in time series analysis.

Due to the considerable costs and time constraints associated with preparing new experimental test setups to generate a large dataset, this paper concentrates on enhancing the accuracy of previous time series response predictions for bridge piers subjected to RTHS and cyclic tests, specifically considering the effects of loading rate behavior. To accomplish this, novel deep learning techniques are employed, offering improved predictive capabilities of the behavior of bridge piers in a more cost-effective and time-efficient manner.

This study introduces a highly accurate and robust deep learning model that exhibits excellent generalization capabilities to unseen data. The proposed model surpasses previous predictions and offers reliable and efficient predictions for a range of applications and scenarios by enhancing the accuracy and robustness of deep learning networks. This paper introduces a hybrid loss function, combining MSE and MAE, offering improved training performance. Other noteworthy features include the adoption of a low-pass Butterworth filter, data augmentation technique, the exponential learning rate scheduler, weight decay regularization technique, Adamax optimizer, ELU activation function, BatchNormalization, dropout, Time Distributed Dense, Max Pooling, and causal CNN. The model incorporates a stacked CNN-Bi-CuDNNLSTM architecture, leveraging skip connections to enhance performance. It utilizes cuDNN, a GPU-accelerated library, to significantly accelerate the training process. The network architecture is further improved by integrating a custom task-specific attention layer, allowing the model to focus on the most relevant parts of the input sequence when making predictions. To optimize resource utilization and reduce costs associated with data acquisition, an error-sensitive analysis is conducted to determine the minimum number of samples required for training datasets, preventing overfitting. The proposed framework utilizes the functional API provided by the Keras Python library, incorporating input features such as the ground acceleration experienced in both the horizontal and vertical directions, the force exerted by the actuators in the horizontal and vertical planes, effective pier height, the second moment of area, and the superstructure mass. The deep learning model is trained, validated, and tested using extensive experimental databases consisting of 12 RTHS, 10 fast, and seven slow cyclic tests. The proposed hybrid loss function demonstrates a decent decrease to near-zero values within the training/validation dataset following 5000 epochs. Notably, the predicted time series responses exhibit over 93% correlation with the experimentally measured values. The proposed deep learning model effectively captures the pier behavior in terms of linearity and nonlinearity, in addition, accurately predicts the peak of displacement and force observed in different bridge piers during RTHS and cyclic tests. The predicted time history responses adeptly capture the corresponding hysteresis and backbone curves, closely mirroring the patterns observed in the experimental data. This precision contributes to diminished labor and experimental costs.

Acknowledgments

The authors gratefully acknowledge the financial support provided by the Brain Pool program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (2022H1D3A2A01096273). Moreover, the authors appreciate all the support provided by Yunbyeong Chae, an associate professor at Seoul National University, for the technical comments, and the engineers and laborers at the Hybrid Structural Testing Center (Hystec), Myongji University, Yongin-si, South Korea to provide experimental setups.

Statements and declarations

Funding

The author(s) received no financial support for the research, authorship and/or publication of this article.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

1. Qin H, Bi K, Dong H, et al. Shake table tests on RC double-column bridge piers with self-centering energy dissipation braces. *J Bridge Eng* 2023; 28: 04023049.
2. Shen Y, Freddi F, Li Y, et al. Parametric experimental investigation of unbonded post-tensioned reinforced concrete bridge piers under cyclic loading. *Earthquake Eng Struct Dynam* 2022; 51: 3479–3504.
3. Han Q, Jia Z, Xu K, et al. Hysteretic behavior investigation of self-centering double-column rocking piers for seismic resilience. *Eng Struct* 2019; 188: 218–232.
4. Huang C, Li Y, Gu Q, et al. Machine learning–based hysteretic lateral force-displacement models of reinforced concrete columns. *J Struct Eng* 2022; 148: 04021291.
5. Ni X, Xiong Q, Kong Q, et al. Deep HystereticNet to predict hysteretic performance of RC columns against cyclic loading. *Eng Struct* 2022; 273: 115103.
6. Urdiales J, Mart´ın D and Armingol JM. An improved deep learning architecture for multi-object tracking systems. *Integr Comput-Aided Eng* 2023; 30: 121–134.
7. Wang M and Cheng JC. A unified convolutional neural network integrated with conditional random field for pipe defect segmentation. *Comput-Aided Civil Infrastruct Eng* 2020; 35: 162–177.
8. Zhang S, Zhou L, Chen X, et al. Network-wide traffic speed forecasting: 3D convolutional neural network with ensemble empirical mode decomposition. *Comput-Aided Civil Infrastruct Eng* 2020; 35: 1132–1147.
9. Demertzis K, Iliadis L and Pimenidis E. Geo-AI to aid disaster response by memory-augmented deep reservoir computing. *Integr Comput-Aided Eng* 2021; 28: 383–398.
10. Chen S, Leng Y and Labi S. A deep learning algorithm for simulating autonomous driving considering prior knowledge and temporal information. *Comput-Aided Civil Infrastruct Eng* 2020; 35: 305–321.
11. Shajihan SAV, Hoang T, Mechitov K, et al. Wireless SmartVision system for synchronized displacement monitoring of railroad bridges. *Comput-Aided Civil Infrastruct Eng* 2022; 37: 1070–1088.
12. Rafiei MH and Adeli H. A novel unsupervised deep learning model for global and local health condition assessment of structures. *Eng Struct* 2018; 156: 598–607.
13. Rafiei MH and Adeli H. NEEWS: A novel earthquake early warning system using neural dynamic classification and neural dynamic optimization model. *Soil Dynam Earthquake Eng* 2017; 100: 417–427.
14. Matinfar M, Khaji N and Ahmadi G. Deep convolutional generative adversarial networks for generation of numerous artificial spectrum-compatible earthquake accelerograms using a limited ground motion records. *Comput-Aided Civil Infrastruct Eng* 2023; 38: 225–240.
15. Li T and Wu T. Modeling nonlinear flutter behavior of long-span bridges using knowledge-enhanced long short-term memory network. *Comput-Aided Civil Infrastruct Eng* 2023; 38: 1504–1519.
16. Yazdanpanah O, Chang M, Park M, et al. Seismic response prediction of RC bridge piers through stacked long short-term memory network. *Structures* 2022; 45: 1990–2006.
17. Yazdanpanah O, Chang M, Park M, et al. Force-deformation relationship prediction of bridge piers through stacked LSTM network using fast and slow cyclic tests. *Struct Eng Mech* 2023; 85: 469–484.
18. Rafiei MH and Adeli H. A novel machine learning based algorithm to detect damage in highrise building structures. *Struct Design Tall Special Build* 2017; 26: e1400.
19. Rafiei MH, Khushefati WH, Demirboga R, et al. Supervised deep restricted boltzmann machine for estimation of concrete compressive strength. *ACI Mater J* 2017; 114: 237–244.
20. Hamidia M, Mansourdehghan S, Asjodi AH, et al. Machine learning-aided scenario-based seismic drift measurement for RC moment frames using visual features of surface damage. *Measurement* 2022; 205: 112195.
21. Javadinasab Hormozabad S, Gutierrez Soto M and Adeli H. Integrating structural control, health monitoring, and energy harvesting for smart cities. *Expert Syst* 2021; 38: e12845.
22. Pezeshki H, Adeli H, Pavlou D, et al. State of the art in structural health monitoring of offshore and marine structures. *Proc Institut Civil Eng-Maritime Eng* 2023; 176: 89–108.
23. Yamane T, Chun P-J, Dang J, et al. Recording of bridge damage areas by 3D integration of multiple images and reduction of the variability in detected results. *Comput-Aided Civil Infrastruct Eng* 2023; 38: 2391–2407.
24. AL-Hawarneh M and Alam MS. Lateral cyclic response of RC bridge piers made of recycled concrete: Experimental study. *J Bridge Eng* 2021; 26: 04021018.
25. Salehi M, Valigura J, Sideris P, et al. Experimental assessment of second-generation hybrid sliding-rocking bridge columns under reversed lateral loading for free and fixed end rotation conditions. *J Bridge Eng* 2021; 26: 04021071.
26. Chae Y, Lee J, Park M, et al. Fast and slow cyclic tests for reinforced concrete columns with an improved axial force control. *J Struct Eng* 2019; 145: 04019044.
27. Lamarche CP and Tremblay R. Seismically induced cyclic buckling of steel columns including residual-stress and strain-rate effects. *J Construct Steel Res* 2011; 67: 1401–1410.
28. Chae Y, Rabiee R, Dursun A, et al. Real-time force control for servo-hydraulic actuator systems using adaptive time series compensator and compliance springs. *Earthquake Eng Struct Dynam* 2018; 47: 854–871.
29. Chae Y, Kazemibidokhti K and Ricles JM. Adaptive time series compensator for delay compensation of servo-hydraulic actuator systems for real-time hybrid simulation. *Earthquake Eng Struct Dynam* 2013; 42: 1697–1715.
30. Karavasilis TL, Ricles JM, Sause R, et al. Experimental evaluation of the seismic performance of steel MRFs with compressed elastomer dampers using large-scale real-time hybrid simulation. *Eng Struct* 2011; 33: 1859–1869.
31. Al-Subaihawi S, Ricles JM and Quiel SE. Online explicit model updating of nonlinear viscous dampers for real time

hybrid simulation. *Soil Dynam Earthquake Eng* 2022; 154: 107108.
32. Hamidia M, Shokrollahi N and Ardakani RR. The collapse margin ratio of steel frames considering the vertical component of earthquake ground motions. *J Construct Steel Res* 2022; 188: 107054.
33. Hamidia M, Afzali M, Jamshidian S, et al. Post-earthquake stiffness loss estimation for reinforced concrete columns using fractal analysis of crack patterns. *Struct Concrete* 2023; 24: 3933–3951.
34. Hamidia M. Simplified seismic collapse capacity-based evaluation and design of frame buildings with and without supplemental damping systems. State University of New York at Buffalo, 2013.
35. Jeon JS, Shafieezadeh A and DesRoches R. Statistical models for shear strength of RC beam-column joints using machine-learning techniques. *Earthquake Eng Struct Dynam* 2014; 43: 2075–2095.
36. Yazdanpanah O, Dolatshahi KM and Moammer O. Earthquake-induced economic loss estimation of eccentrically braced frames through roof acceleration-based non-model approach. *J Construct Steel Res* 2021; 187: 106888.
37. Yazdanpanah O, Dolatshahi KM and Moammer O. Rapid seismic fragility curves assessment of eccentrically braced frames through an output-only nonmodel-based procedure and machine learning techniques. *Eng Struct* 2023; 278: 115290.
38. Chang MK, Kwiatkowski JW, Nau RF, et al. ARMA models for earthquake ground motions. *Earthquake Eng Struct Dynam* 1982; 10: 651–662.
39. Alevizakou EG and Pantazis G. A comparative evaluation of various models for prediction of displacements. *Appl Geomat* 2017; 9: 93–103.
40. Luo H and Paal SG. A data-free, support vector machine-based physics-driven estimator for dynamic response computation. *Comput-Aided Civil Infrastruct Eng* 2023; 38: 26–48.
41. Yang F, Moayedi H and Mosavi A. Predicting the degree of dissolved oxygen using three types of multi-layer perceptron-based artificial neural networks. *Sustainability* 2021; 13: 9898.
42. Huang P and Chen Z. Deep learning for nonlinear seismic responses prediction of subway station. *Eng Struct* 2021; 244: 112735.
43. Oh BK, Park Y and Park HS. Seismic response prediction method for building structures using convolutional neural network. *Struct Control Health Monitor* 2020; 27: e2519.
44. Hochreiter S and Schmidhuber J. Long short-term memory. *Neural Comput* 1997; 9: 1735–1780.
45. Bui KTT, Torres JF, Gutiérrez-Avilés D, et al. Deformation forecasting of a hydropower dam by hybridizing a long short-term memory deep learning network with the coronavirus optimization algorithm. *Comput-Aided Civil Infrastruct Eng* 2022; 37: 1368–1386.
46. Zhang R, Chen Z, Chen S, et al. Deep long short-term memory networks for nonlinear structural seismic response prediction. *Comput Struct* 2019; 220: 55–68.
47. Núñez H, Gonzalez-Abril L and Angulo C. Improving SVM classification on imbalanced datasets by introducing a new bias. *J Classif* 2017; 34: 427–443.
48. Goller C and Kuchler A. Learning task-dependent distributed representations by backpropagation through structure. In: Proceedings of International Conference on Neural Networks (ICNN'96), 1996, vol. 1, pp.347–352. IEEE.
49. Jordan MI. Serial order: A parallel distributed processing approach. *Adv Psychol* 1997; 121: 471–495.
50. Schuster M and Paliwal KK. Bidirectional recurrent neural networks. *IEEE Trans Signal Process* 1997; 45: 2673–2681.
51. Perez-Ramirez CA, Amezquita-Sanchez JP, Valtierra-Rodriguez M, et al. Recurrent neural network model with Bayesian training and mutual information for response prediction of large buildings. *Eng Struct* 2019; 178: 603–615.
52. Mousavi SM and Beroza GC. A machine-learning approach for earthquake magnitude estimation. *Geophys Res Lett* 2020; 47: e2019GL085976.
53. The NVIDIA CUDA® Deep Neural Network library (cuDNN-11.2). 2021. https://developer.nvidia.com/cudnn
54. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp.770–778.
55. Chun PJ, Yamane T and Maemura Y. A deep learning based image captioning method to automatically generate comprehensive explanations of bridge damage. *Comput-Aided Civil Infrastruct Eng* 2022; 37: 1387–1401.
56. Korea Road and Transportation Association. *Bridge design specifications (limit state design method)*. Gunsulbook; [in Korean], 2012.
57. Chae Y, Park M, Kim CY, et al. Experimental study on the rate-dependency of reinforced concrete structures using slow and real-time hybrid simulations. *Eng Struct* 2017; 132: 648–658.
58. Chae Y, Lee J, Park M, et al. Real-time hybrid simulation for an RC bridge pier subjected to both horizontal and vertical ground motions. *Earthquake Eng Struct Dynam* 2018; 47: 1673–1679.
59. Hu D. An introductory survey on attention mechanisms in NLP problems. In: Intelligent Systems and Applications: Proceedings of the 2019 Intelligent Systems Conference (IntelliSys), 2020, vol. 2, pp.432–448. Springer International Publishing.
60. Gu Y, Tinn R, Cheng H, et al. Domain-specific language model pretraining for biomedical natural language processing. *ACM Trans Comput Healthcare* 2021; 3: 1–23.
61. Bahdanau D, Cho K and Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint. 2014; 1409.0473.

62. Lee SY. Task Specific Attention is one more thing you need for object detection. arXiv preprint arXiv. 2202; 09048.
63. Liu S, Johns E and Davison AJ. End-to-end multi-task learning with attention. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp.1871–1880.
64. Blackwood G, Ballesteros M and Ward T. Multilingual neural machine translation with task-specific attention. arXiv preprint arXiv. 2018; 1806.03280.
65. Yan C, Hao Y, Li L, et al. Task-adaptive attention for image captioning. *IEEE Trans Circuits Syst Video Technol* 2021; 32: 43–51.
66. Ishihara K, Kanervisto A, Miura J, et al. Multi-task learning with attention for end-to-end autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp.2902–2911.
67. Sood E, Tannert S, Müller P, et al. Improving natural language processing tasks with human gaze-guided neural attention. *Adv Neural Inform Process Syst* 2020; 33: 6327–6341.
68. Pandey A and Wang D. TCNN: Temporal convolutional neural network for real-time speech enhancement in the time domain. In: ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp.6875–6879. IEEE.
69. Mariani S, Rendu Q, Urbani M, et al. Causal dilated convolutional neural networks for automatic inspection of ultrasonic signals in non-destructive evaluation and structural health monitoring. *Mech Syst Signal Process* 2021; 157: 107748.
70. Giusti A, Cireşan DC, Masci J, et al. Fast image scanning with deep max-pooling convolutional neural networks. In: 2013 IEEE International Conference on Image Processing, 2013, pp.4034–4038.
71. Nagi J, Ducatelle F, Di Caro GA, et al. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In: 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), 2011, pp.342–347.
72. Yuan Z, Zhu S, Chang C, et al. An unsupervised method based on convolutional variational auto-encoder and anomaly detection algorithms for light rail squat localization. *Construct Build Mater* 2021; 313: 125563.
73. https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
74. https://github.com/tensorflow/docs/blob/r1.12/site/en/api_docs/python/tf/keras/layers/CuDNNLSTM.md
75. Duan Y, Li H, He M, et al. A BiGRU autoencoder remaining useful life prediction scheme with attention mechanism and skip connection. *IEEE Sens J* 2021; 21: 10905–10914.
76. Wang TC, Liu MY, Zhu JY, et al. High-resolution image synthesis and semantic manipulation with conditional gans. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp.8798–8807.
77. Karras T, Aila T, Laine S, et al. Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv. 2017; 1710.10196.
78. Clevert DA, Unterthiner T and Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv. 2015; 1511.07289.
79. Ioffe S and Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, 2015, pp.448–456. Pmlr.
80. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 2014; 15: 1929–1958.
81. https://keras.io/api/layers/recurrent_layers/time_distributed/
82. Ng A. Deep Learning Specialization, Course 2, Week 2, Optimization Methods; Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization. An online non-credit course authorized by DeepLearning. AI and offered through Coursera, Stanford University. 2023. https://www.coursera.org/specializations/deep-learning
83. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
84. Yazdanpanah O, Chang M and Ali Bakhshi E. Attention-based stacked long-short term memory network for displacement time history prediction of reinforced concrete bridge piers using experimental real-time hybrid simulation. In: 4th International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI' 2022), Corfu, Greece, 19–21 October 2022.

Appendix A: The custom layer of Task-Specific Attention

```
import tensorflow as tf
from tensorflow.keras.layers import Layer
    class TaskSpecificAttention(tf.keras.layers.Layer):
        def __init__(self, units = 100, return_sequences =
True, activation='ELU', name = None, **kwargs):
            super(TaskSpecificAttention,
self).__init__(name = name)
            self.units = units
            self.return_sequences = return_sequences
            self.activation = tf.keras.activations.get(activation)
            super(TaskSpecificAttention,
self).__init__(**kwargs)
        def build(self, input_shape):
            self.time_steps = input_shape[1]
            self.features = input_shape[2]
            self.W = self.add_weight(name='W',
shape = (self.features, self.units), initializer='glorot_
uniform')
            self.b = self.add_weight(name='b',
shape = (self.units,), initializer='zeros')
            self.u = self.add_weight(name='u',
shape = (self.units,), initializer='glorot_uniform')
            super(TaskSpecificAttention,
self).build(input_shape)
        def call(self, inputs):
```

```
        u_it = self.activation(tf.tensordot(inputs, self.W,
axes = 1)
+ self.b)
        attention_scores = tf.tensordot(u_it, self.u,
axes = 1)
        attention_weights = tf.expand_dims(tf.nn.softmax
(attention_scores), axis = −1)
        weighted_inputs = tf.multiply(inputs,
attention_weights)
        if self.return_sequences:
          return weighted_inputs
        else:
          return tf.reduce_sum(weighted_inputs, axis = 1)
    def get_config(self):
        config = super(TaskSpecificAttention,
self).get_config()
        config.update({
          'units': self.units,
          'return_sequences': self.return_sequences,
          'activation': tf.keras.activations.serialize
(self.activation),
    })
        return config
```

It works in the following manner:

- The class is initialized with parameters such as the number of units, whether to return sequences and the activation function.
- The build method initializes W, b, and u based on the input shape.
- in the call method, the attention mechanism is applied.
- The attention mechanism is applied to the output of the preceding CNN and bidirectional CuDNNLSTM layers.

It refines the representation by assigning different weights to different parts of the input sequence.

The parameters W, b, and u are learned during the model training using the add_weight function. These parameters are task-specific because they are initialized randomly and then updated through the learning process to capture task-specific information. The attention scores are calculated by taking the dot product between the transformed inputs (u_it) and the learned weight u. This operation captures the relevance of each element in the input sequence to the task. The attention scores are passed through a softmax activation function (tf.nn.softmax) to obtain attention weights. The softmax function normalizes the attention scores, ensuring that they represent a valid probability distribution over the input sequence. The input sequence is multiplied element-wise with the attention weights (weighted_inputs). This step applies the attention weights to the input sequence, emphasizing the relevant elements and downplaying the less relevant ones.

A detailed examination of the key components of the code is outlined here as follows:

- "__init__": The constructor initializes the layer with various parameters such as the number of units, whether to return sequences, and the activation function.
- " build": This method is called when the layer is built. It initializes the weights (W, b, and u) used by the attention mechanism based on the input shape.
- "call": This method defines the forward pass of the layer. It computes the attention scores, applies softmax to obtain attention weights, and then multiplies these weights with the input to get the weighted inputs.
- "get_config": This method returns a dictionary containing the configuration of the layer, which includes parameters like the number of units, whether to return sequences and the activation function.
- Computes intermediate representation u_it using a trainable weight matrix (W) and a bias term (b).
- Computes attention scores using another trainable weight vector (u).
- Applies softmax to obtain attention weights.
- Multiplies the input by the attention weights to get the weighted inputs.
- The code assumes that the input shape is (batch_size, time_steps, features).

Appendix B: Exponential Linear Unit activation function (ELU)import

```
from tensorflow.keras.layers import Layer
from tensorflow.keras import backend as K
  class ELU(Layer):
    def __init__(self, alpha = 0.001, **kwargs):
      super(ELU, self).__init__(**kwargs)
      self.alpha = alpha
    def call(self, x):
      return K.elu(x, alpha = self.alpha)
    def get_config(self):
      config = super().get_config().copy()
      config.update({'alpha': self.alpha})
        return config
```

Appendix C: Learning rate scheduler callback

```
from tensorflow.keras.callbacks import Learn-
ingRateScheduler
  def exp_decay(epoch, lr):
    initial_lr = 0.001
    k = 0.005
    lr = initial_lr * np.exp(-k*epoch)
    return lr
```

```
lr_scheduler = LearningRateScheduler(exp_decay, verbose = 1)
```

The lr_scheduler is passed to the callbacks argument when fitting the model using the model.fit() method. The learning rate is adjusted according to the exponential decay formula during training. initial_lr and k sand for the initial learning rate and the decay constant, it determines the rate at which the learning rate decreases over epochs.

Appendix D: A custom callback of Weight decay in TensorFlow Keras

```
from tensorflow.keras.callbacks import Callback
  class WeightDecay(Callback):
    def __init__(self, weight_decay):
      super().__init__()
      self.weight_decay = weight_decay
    def on_batch_end(self, batch, logs = None):
      for weight in self.model.trainable_weights:
        weight.assign(weight * (1 - self.weight_decay))
  wd_callback = WeightDecay(weight_decay = 0.0001)
```

By using the WeightDecay callback during training, the weights of the model will be decayed at the end of each batch, helping to prevent overfitting.

To use the WeightDecay callback and lr_scheduler, they are passed to the callbacks argument as callbacks = [lr_scheduler, wd_callback] when fitting the model using the model.fit() method.