Explainable Parallel RCNN with Novel Feature Representation for Time Series Forecasting

Jimeng Shi ✉[1], Rukmangadh Myana[1], Vitalii Stebliankin[1], Azam Shirali[1], and Giri Narasimhan[1]

Knight Foundation School of Computing and Information Sciences,
Florida International University
{jshi008, rmyan001, vsteb002, ashir018, giri}@fiu.edu

Abstract. Accurate time series forecasting is a fundamental challenge in data science, as it is often affected by external covariates such as weather or human intervention, which in many applications, may be predicted with reasonable accuracy. We refer to them as *predicted future covariates*. However, existing methods that attempt to predict time series in an iterative manner with auto-regressive models end up with exponential error accumulations. Other strategies that consider the past and future in the encoder and decoder respectively limit themselves by dealing with the past and future data separately. To address these limitations, a novel feature representation strategy - *shifting* - is proposed to fuse the past data and future covariates such that their interactions can be considered. To extract complex dynamics in time series, we develop a parallel deep learning framework composed of RNN and CNN, both of which are used in a hierarchical fashion. We also utilize the *skip connection* technique to improve the model's performance. Extensive experiments on three datasets reveal the effectiveness of our method. Finally, we demonstrate the model *interpretability* using the Grad-CAM algorithm.

1 Introduction

Time series forecasting plays an essential role in many scenarios in real life. Accurate forecasting allows people to do better resource management [21] and optimization decisions [5] for critical processes. Applications include demand forecasting in retail [2], dynamic assignments of beds to patients [35], monthly inflation forecasting [1], and much more. Because of its popularity and significance, many time series forecasting methods have been explored. Traditional statistical forecasting methods, such as autoregression [8], exponential smoothing [13], and ARIMA [3], are widely utilized for univariate time series. These methods learn the temporal features (e.g., trends and seasonality) from past data and achieve good performance for univariate time series prediction. But they are ineffective to learn the complex dynamics among multivariate time series, partly because of their inability to take advantage of *covariates* - independent variables that can influence the target variable, although perhaps not directly.

Good time series forecasting requires substantial amounts of historical data of the target variable(s) to learn temporal patterns. They also require the exogenous covariates to learn the dependent relationships. More importantly, in many applications, some of the covariates can be predicted with reasonable accuracy for the immediate future. We refer to such covariates from the immediate future as *predicted future covariates*. For example, in terms of the task predicting water levels in a river or canal system, a covariate of interest could be *precipitation*. And it is possible to use historical data as well as reasonably accurate predictions for the near future, which may be obtained from the weather services.

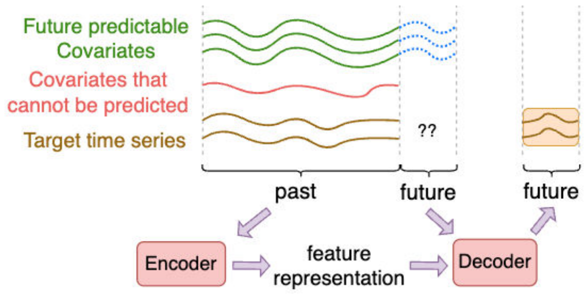


Fig. 1. Direct method using Seq2Seq models with encoder and decoder.

Existing methods employing both past and future data for time series forecasting problems are mainly divided into two categories: (1) **iterative methods** [23,25] that iteratively predict one step at a time, and (2) **direct methods** [19,31] that are trained to explicitly forecast the pre-defined horizons with sequence-to-sequence models (which originated from the speech translation domain [20]). However, they have several limitations. The iterative methods consider the prediction output from the previous time step as the input for the next time step during the model training process. Such methods suffer from error accumulation caused by the multiplication of errors.

In another direct strategy (see Fig. 1), the Seq2Seq framework – encoder and decoder [22] absorbs the historical data in the encoder and includes the predicted future covariates in the decoder. Such a strategy considers historical data and the predicted future covariates separately, probably causing the model to miss the past-future connections. Some researchers have added an attention layer [7,34] in the Seq2Seq framework to capture more local or global information, but the prediction performance improves only slightly and fails to handle the inherent constraints of the Seq2Seq model.

In this work, we aim to address the existing limitations, and our five-fold contributions are listed below:

– To avoid separately considering the past and future data, we propose a novel feature representation strategy called *shifting*, which can contextually link

the past with the predicted future covariate as an integrated input. *Shifting* also makes it possible to use a single compact model to effectively combine both past and future data simultaneously.
– To improve the efficiency of the model, we introduce a parallel framework composed of RNN and CNN (ParaRCNN) to capture complex time series dynamics. Note that ParaRCNN is a single and compact model compared to the Seq2Seq architecture.
– Our model can make multi-step predictions in a one-shot manner, which can avoid error accumulation in contrast to auto-regressive models.
– We adapt the *skip connection* to facilitate improved learning since such a technique can maximize the usability of input features.
– We provide the model interpretability with the Grad-CAM algorithm to identify how each time step and feature contributes to the final predictions.

2 Problem Formulation

Let $\mathbf{Z}_{1:t}^{N} = (z_1^n, z_2^n, ..., z_t^n)_{n=1}^{N} \in R^{t\times N}$ be N univariate time series of target variables, where $z_t^n \in R$ denotes the value of the n-th target variable at time t. Let $\mathbf{X}_{1:t}^{M} = (x_1^m, x_2^m, \ldots, x_t^m)_{m=1}^{M} \in R^{t\times M}$ be M the observed time-varying covariates that are measured until time t and that cannot be predicted for the future. Finally, let $\mathbf{Y}_{1:t}^{Q} = (y_1^q, y_2^q, \ldots, y_t^q)_{q=1}^{Q} \in R^{t\times Q}$ be the Q time series for covariates measured until time t, but which can be reliably estimated for the near future; we let $\mathbf{Y}_{t+1:t+k}^{Q} \in R^{t\times Q}$ denote those predicted covariates k time steps into the future. We refer to these estimable variables as *future predictable covariates.* The goal of forecasting models is to compute the predicted trajectories of the target time series. We will refer to these as $\hat{\mathbf{Z}}_{t+1:t+k}^{N}$ (k is the forecasting length) to distinguish it from the measured target time series. The computations assumes that the input data is heterogeneous and includes the historical data (target variables $\mathbf{Z}_{1:t}^{N}$, observed covariates $\mathbf{X}_{1:t}^{M}$, historical future predictable covariates $\mathbf{Y}_{1:t}^{Q}$), and predicted future covariates $\mathbf{Y}_{t+1:t+k}^{Q}$. Note that the term "covariate" in this paper refers to those exogenous time-varying covariates rather than time itself.

3 Related Work

Traditional statistical methods learn the temporal patterns only based on historical data [3,28] of target variables themselves (see Eq. (1)). However, many approaches also aim to learn the dependent relationship between target variables and covariates, especially for the predicted future covariate [4,9,19,23,25,29,31]. Related research can be mainly categorized into iterative methods using auto-regressive models and direct strategies that use sequence-to-sequence models. We have:

$$\hat{\mathbf{Z}}_{t+i}^{N} = \mathbf{F}_{\theta}(\mathbf{Z}_{1:t}^{N}, \hat{\mathbf{Z}}_{t+1:t+i-1}^{N}), \tag{1}$$

where $\mathbf{F}_{\theta}(\cdot)$ is a prediction model with a set of learnable parameters θ; $\hat{\mathbf{Z}}_{t+i}^{N}$ is the N target variables i time step into the future for $i = 1, 2, \ldots, k$.

Iterative methods. The iterative strategy recursively uses a one-step-ahead forecasting model [6,32] multiple times where the predicted value for the previous time step is used as the input to forecast the next time step. A typical iterative framework is the DeepAR model [25] from Amazon Research. During the training process, to predict target values $\hat{\mathbf{Z}}_t^N$ at time step t, the inputs to the network are the covariates $\mathbf{Y}_t^Q$, the target values at the previous time step $\mathbf{Z}_{t-1}^N$, and the previous network output $\mathbf{h}_{t-1}$. Note that the previous target values are known during training. During inference, measured target values $\mathbf{Z}_{t-1}^N$ are replaced by predicted target values $\hat{\mathbf{Z}}_{t-1}^N$ and then fed back to predict the next time step of $\hat{\mathbf{Z}}_{t+1}^N$ until the end of the prediction range. A mathematical formulation of such forecasting methods is given in Eq. (2) using the notation in Section 2. Similar approaches were adopted in [14,16,23] using different backbones. However, an inherent shortcoming of this method is that errors accumulate multiplicatively since later predictions depend on earlier predictions.

$$\hat{\mathbf{Z}}_{t+i}^N = \mathbf{F}_\theta(\mathbf{Z}_{1:t}^N, \hat{\mathbf{Z}}_{t+1:t+i-1}^N, \mathbf{X}_{1:t}^M, \mathbf{Y}_{1:t+i}^Q). \tag{2}$$

Direct methods. The typical Seq2Seq framework for direct methods is shown in Fig 1. It deals with past and future data separately in the encoder and decoder components, respectively. The encoder model learns the feature representation of past data, which is saved as context vectors in a hidden state. The decoder model takes as input the encoder output and the additional future covariates to predict the future target values. Examples of this approach include the MQRNN model [31] that used an LSTM as the encoder to generate context vectors, which are then combined with future covariates and fed into a multi-layer perceptron (MLP) to predict the future horizon. Some efforts [7,9] have utilized a temporal attention mechanism between the encoder and the decoder. This architecture can learn the relevance of different parts of the feature representations from historical data by computing "attentional" weights. The weighted feature representations are then passed into the decoder to predict future time steps. Temporal Fusion Transformer [19] combined gated residual networks (GRNs) and an attention mechanism [30] as an additional decoder on top of the traditional encoder-decoder model. They used GRNs to filter unnecessary information and employed the additional decoder with an attention mechanism to capture long-term dependencies. Generally, the direct methods can be modeled as follows:

$$\begin{aligned}\mathbf{H}_t &= \mathbf{F}_{encoder}(\mathbf{Z}_{1:t}^N, \mathbf{X}_{1:t}^M, \mathbf{Y}_{1:t}^Q),\\ \hat{\mathbf{Z}}_{t+i}^N &= \mathbf{F}_{decoder}(\mathbf{H}_t, \mathbf{Y}_{t+1:t+i}^Q).\end{aligned} \tag{3}$$

Direct methods that use the Seq2Seq framework with the encoder and decoder in series might be prone to miss some interactions between the past and future due to separate processing styles. Moreover, the Seq2Seq framework is complicated and computationally time-consuming because of the use of two models – the encoder and the decoder. This provided the motivation for us to explore a compact model that simultaneously analyzes the measured past and the predictable future.

4 Methodology

In this section, we first illustrate the *shifting* strategy that fuses the past and future data in a structured way for an integrated feature representation. Then we present the details of the proposed model architecture and discuss how it learns from the fused data and the skip connection technique. In this paper, we define a sliding window [11] (also called rolling window [17] or look-back window [27]) of a certain length, w, as the input from the recent past, and to predict future time steps of length k.

4.1 Data Fusion with Shifting

To avoid dealing with the past and future data separately, we shift the covariates for the future period of interest (blue dotted trajectory in Fig. 2) back by s time steps, such that they are aligned and fused with all historical time series to produce distinct feature vectors. Then both the past and future data are fed into a single model together. Now the inputs are composed of all the past time series (target and covariates) aligned from time steps $t-w+1$ to t with future predictable covariates from time steps $t-w+1+s$ to $t+s$. Specifically, at each time step, we obtain a 4-tuple (z_j, x_j, y_j, y_{j+s}), which is input to a state cell in the RNN (Fig. 4) or a filter kernel in the CNN (Fig. 5), thus fusing the information from the historical data (z_j, x_j, y_j) at time j and future predictable covariates y_{j+s} at time $j+s$. The above design allows both the past and future to be considered in one single component of the model at the same time. The set of target variables, $\mathbf{Z}^N_{t+1:t+k}$ are predicted in the forecasting horizon from $t+1$ to $t+k$. The *shifting* strategy is illustrated in Fig. 2 and modelled as Eq. (4) below:

$$\hat{\mathbf{Z}}^N_{t+1:t+k} = \mathbf{G}_\theta(\mathbf{Z}^N_{t-w+1:t}, \mathbf{X}^M_{t-w+1:t}, \mathbf{Y}^Q_{t-w+1:t}, \mathbf{Y}^Q_{t-w+1+s:t+s}), \qquad (4)$$

where $\mathbf{G}_\theta(\cdot)$ is a function with learnable parameters θ; and $\mathbf{Y}^Q_{t-w+1+s:t+s}$ is the future predictable covariates along with predictions from s time steps into the future and then shifted back by s time steps (green trajectories merged with dotted blue trajectories in Fig. 2). Note that the shifted future predictable covariates $\mathbf{Y}^Q_{t-w+1+s:t+s}$ and the single unified model given by $\mathbf{G}_\theta$ in Eq. (4) differentiate our method from the previous methods discussed in Section 3.

4.2 Network Architectures

With the input data transformed and fused (Fig. 2, right) by the *shifting* strategy, we develop a parallel framework composed of RNN and CNN, both of which are in a hierarchical structure. As shown in Fig. 3, both the number of filters for CNN and the number of units for RNN decrease over the layers to extract high-level time series dynamics. More specifically, since RNN and CNN learn the temporal dependency and dynamics in different mechanisms, we construct

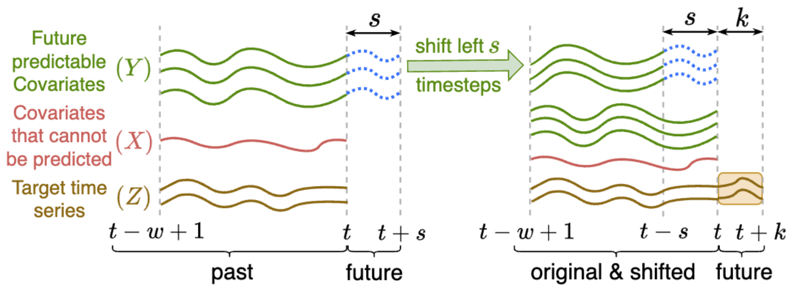


Fig. 2. Input data transformed by the *shifting* strategy. Left: Trajectories of all variables before transformation. Right: Original trajectories along with shifted future predictable covariates. Predicted output is the future k time steps of the target variables.

RNN and CNN in parallel, which benefits the model by capturing heterogeneous feature representations from input time series. Meanwhile, the *skip connection* technique is utilized to enhance learning since it maximizes the usability of the input features. Lastly, the fused input, the CNN output, and the RNN output are concatenated together and fed into a fully-connected layer to make the final predictions.

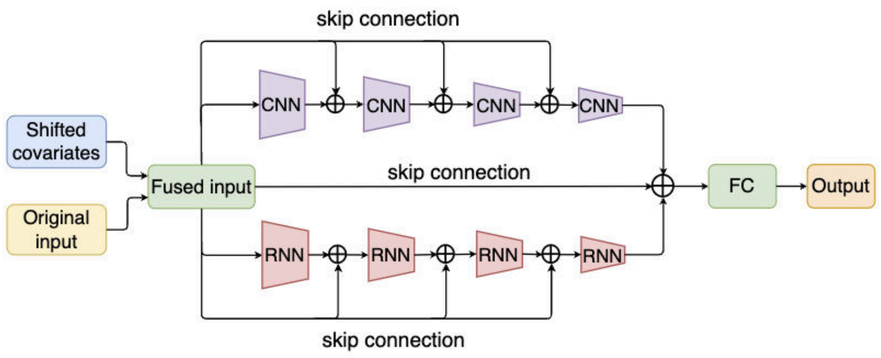


Fig. 3. Architecture of the proposed ParaRCNN model. There are 256, 128, 64, and 32 filters for CNN modules (Conv1D) and 128, 64, 32, and 16 units for RNN modules (SimpeRNN), respectively.

RNN with Shifting Recurrent Neural Networks (RNNs) learn the temporal dependency from input features in the recent past to future one or more target variables by recurrently training and updating the transitions of an internal (hidden) state from the last time step to the current time step. To predict the future k time steps, the standard RNNs were further modified to remove the hidden states $\mathbf{h}_{t+1}, \mathbf{h}_{t+2}, .., \mathbf{h}_{t+k}$ to enable a one-shot prediction while avoiding the accumulation of prediction errors. As shown in Fig. 4, we implement the

RNNs with only w hidden states in our paper. The predicted future covariates are shifted to the past by s time steps and aligned with past data by the *shifting* strategy such that the input for each hidden state $\mathbf{h}_j$ at time $t = j$ is a 4-tuple $(\mathbf{z}_j, \mathbf{x}_j, \mathbf{y}_j, \mathbf{y}_{j+s})$. Hierarchical RNNs queued in series (Fig. 3) are expected to distill the high-level features from the input time series. At last, the RNNs generate the prediction for target variables $(\mathbf{z}_{t+1}, \mathbf{z}_{t+2}, \ldots, \mathbf{z}_{t+k})$ in a one-shot manner. The hidden states are recursively computed by:

$$\begin{aligned} h_j &= f(\mathbf{h}_{j-1}, \mathbf{z}_j, \mathbf{x}_j, \mathbf{y}_j, \mathbf{y}_{j+s}), \\ &= tanh(\mathbf{b} + \mathbf{U}^T h_{j-1} + \mathbf{W}^T(\mathbf{z}_j, \mathbf{x}_j, \mathbf{y}_j, \mathbf{y}_{j+s})), \end{aligned} \tag{5}$$

where f is an activation function (hyperbolic tangent function); $\mathbf{h}_j$ and $\mathbf{h}_{j-1}$ refer to the current and previous hidden states; $\mathbf{z}_j$, $\mathbf{x}_j$, and $\mathbf{y}_j$ represent the target time series, observed covariates, and predictable future covariates from the past w time steps; $\mathbf{y}_{j+s}$ denotes the predicted future covariates from k steps into the future; $\mathbf{U}, \mathbf{W}$ are weight matrices and $\mathbf{b}$ is the bias vector.

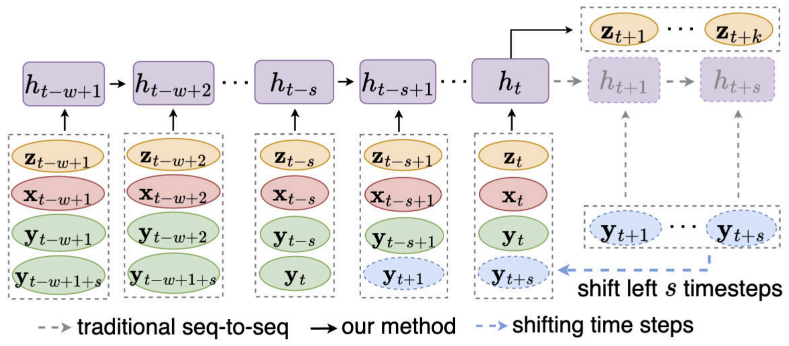

h_{t-w+1}
h_{t-w+2}
h_{t-s}
h_{t-s+1}
h_t
h_{t+1}
h_{t+s}
$\mathbf{z}_{t+1}$ ⋯ $\mathbf{z}_{t+k}$
$\mathbf{z}_{t-w+1}$, $\mathbf{x}_{t-w+1}$, $\mathbf{y}_{t-w+1}$, $\mathbf{y}_{t-w+1+s}$
$\mathbf{z}_{t-w+2}$, $\mathbf{x}_{t-w+2}$, $\mathbf{y}_{t-w+2}$, $\mathbf{y}_{t-w+1+s}$
$\mathbf{z}_{t-s}$, $\mathbf{x}_{t-s}$, $\mathbf{y}_{t-s}$, $\mathbf{y}_t$
$\mathbf{z}_{t-s+1}$, $\mathbf{x}_{t-s+1}$, $\mathbf{y}_{t-s+1}$, $\mathbf{y}_{t+1}$
$\mathbf{z}_t$, $\mathbf{x}_t$, $\mathbf{y}_t$, $\mathbf{y}_{t+s}$
$\mathbf{y}_{t+1}$ ⋯ $\mathbf{y}_{t+s}$
shift left s timesteps
traditional seq-to-seq
our method
shifting time steps

Fig. 4. The RNN architecture with the *shifting* strategy. Dashed blue ovals represent predicted future covariates. Solid ovals are historical target variables and covariates. The last row has the shifted covariates.

CNN with Shifting CNN is a popular model in the image processing field because of the powerful learning ability of *convolutional kernels* embedded inside. 2D-CNN is widely adopted to deal with images [33] by moving 2-D convolutional kernels along the height and width dimension of each image. For multivariate time series, it consisted of multiple univariate time series that fundamentally are sequential in nature. Therefore, 1-D convolutional kernels (also called filters) are used in our paper to learn the temporal and cross-feature dependency [15]. We consider the multivariate time series as a matrix with the shape of (rows, columns) [36] where the rows represent the time steps, and the columns represent the features that generally equal the number of time series dimensions. We also

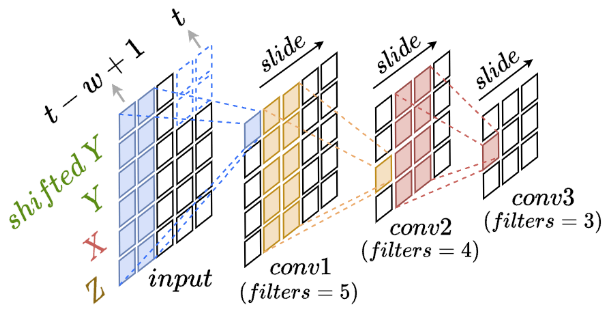


Fig. 5. CNN with 3 convolutional layers. Input includes all original variables and the shifted predicted future covariates. Each filter includes past and future information. Each row represents the convolution results with one filter.

tried 2D-CNN, and the performance was not much different from 1D-CNN, but it needs more computation resources.

As Fig. 5 shows, the shifted future predictable covariates and the original observed data are simultaneously considered by the sliding 1-D convolutional kernels. In other words, each 1-D convolutional kernel learns from the historical data (past) and the predicted data s time steps ahead (future). Such convolutional operations on both the history and predicted future input could be described by Eq. (6). Formally, a convolution operation between two convolutional layers is given by Eq. (7).

$$V_j = \sigma(K_j \circledast (\mathbf{Z}^N_{j:j+\Delta t}, \mathbf{X}^M_{j:j+\Delta t}, \mathbf{Y}^Q_{j:j+\Delta t}, \mathbf{Y}^Q_{j+s:j+\Delta t+s})), \tag{6}$$

where $\circledast$ refers to the convolution operator; K_j is a filter at the time j; Δt is the length of segmented time series for the convolutioncomputations; σ represents the activation function; V_j is the output value at time j.

$$a^{l+1}_j = \sigma(b^l_j + \sum_{f=1}^{F^l} K^l_{jf} \circledast a^l_f), \tag{7}$$

where $\circledast$ represents the convolution operator; l indexes the layer, f indexes the filter; K_j is a filter at the time j; F^l is the number of filters used in the l^{th} layer; σ denotes the activation function.

Skip Connection It is used to train a deep neural network by copying and bypassing the input from the former layers to the deeper layers by matrix addition. ResNets add a skip-connection that bypasses the non-linear transformations with an identity function. For example, given a single image x_0 that is passed through subsequent convolutional layers, each layer implements a non-linear transformation $H(\cdot)$. The output of l^{th} layer with skip connection looks as:

$$x_l = H_l(x_{l-1}) + x_{l-1}. \tag{8}$$

DenseNets [12] achieves skip connections by concatenation. In their work, for each layer, the feature maps of all preceding layers and their own feature maps are used as inputs into all subsequent layers by simple concatenation as shown in Eq. (9). There are $L(L+1)/2$ skip connections for the networks with L layers.

$$x_l = H_l(x_0, x_1, \ldots, x_{l-1}), \tag{9}$$

where $x_0, x_1, \ldots x_{l-1}$ denotes the concatenation of the feature maps produced in previous layers. It shows how the l^{th} layer considers the feature maps of all former layers as input.

$$x_l = H_l(x_{l-1}) + x_0. \tag{10}$$

However, challenges persist with both strategies. ResNets hinder the skip connection because of the matrix addition, which needs the same dimension for both preceding and subsequent matrices. DenseNets have a more complex structure with $L(L+1)/2$ connections as it conveys all former outputs to the latter layers. U-Net models simply pass the original input once to the latter layers. In our model, we adopt L skip connections by bypassing the original input to every latter layer with concatenation (see Eq. (10) and Fig. 6d). Such a structure can facilitate the model by reusing the original input many times and learning it directly while avoiding the *vanishing gradient* issue of deeper layers [18].

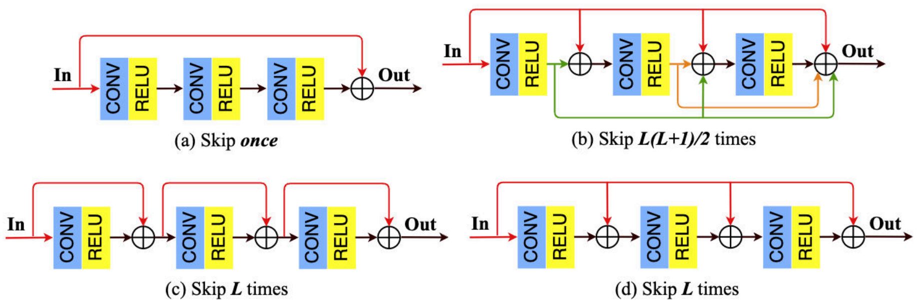


Fig. 6. Various strategies for skip connection. We adopt the strategy of (d) in our paper and compare it with (a) Skip *once* used in U-net [24], (b) Skip $L(L+1)/2$ times used in DenseNet [12], and (c) the benchmark strategy skipping L times.

5 Experiments

5.1 Datasets

Three real-world datasets were used for time series forecasting tasks. *Beijing PM2.5* and *Electricity price* datasets are publicly available from the Machine Learning Repository of the University of California, Irvine, and Kaggle repositories, respectively. The third one is the *Water Stage* dataset downloaded from the South Florida Water Management District (SFWMD) website.

Beijing PM2.5 It includes hourly observed data from January 1, 2010, to December 31, 2014. We consider PM2.5 as the target variable to predict, other variables such as dew, temperature, pressure, wind speed, wind direction, snow, and rain are covariates that can be predicted and can influence PM2.5 values. $\mathit{PM2.5} \in [0, 671]\ \mu g/m^3$.

Electricity price It has two hourly datasets from January 1, 2015, to December 31, 2018. `Energy_dataset.csv` includes energy demand, generation, and prices, while `weather_features.csv` gives the weather features temperature, humidity, etc. Electricity price is the target variable to predict, while the prior known predictable covariates are energy demand, generation, and weather features. *Electricity price* $\in [\$9.33, \$116.8]$ in this dataset.

Water stage This is an hourly dataset from January 1, 2010, to December 31, 2020, and includes information on water levels, the height of gate opening, water flow values through the gate, water volumes pumped at gates, and rainfall measures. The water stage is the target variable while other variables are covariates. Rainfall, gate position, and pump control are future covariates that can be predicted. *Water stage* $\in [-1.25, 4.05]$ feet in the dataset.

5.2 Training & Evaluation

Our models. We predict $k = 24$ hours in the future with input windows of size $w = 72$ hours and predicted future covariates in the same future horizons. We consider the entire target series $\mathbf{Z}^N_{t+1:t+k}$ as the ground-truth labels in supervised learning, which can allow one-shot forecasting to avoid the error accumulation of the traditional iterative prediction. For each dataset, we selected the first 80% as the training set to train the models, and the remaining 20% was chosen as the test set to evaluate the performance. *Max-Min normalization* shown as Eq. (11) was used to squeeze the input data into $[0, 1]$ to avoid possible data bias due to the different scales. We also used *early stopping* and *L1L2 regularization* to alleviate overfitting. There are several hyperparameters in our model. We set $\{16, 32, 64, 128, 256\}$ as the candidate numbers of internal units of RNNs and filters in CNNs. {1e-3, 5e-4, 1e-4, 5e-5, 1e-5} was tested as the learning rate and regularization factor. The shifting length s was validated with the range of $[1, w + k]$ (see Fig. 7). Open-source code can be accessed via the link [1].

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{11}$$

Baseline models. DeepAR [25] iteratively predicting future time steps was viewed as one of the baseline models. Seq2Seq approaches include MQRNN [31] and Temporal Fusion Transformer (TFT) [19], which consider the past data and future covariates separately in the encoder-decoder framework. To validate the functionality of *shifting*, we also adapted baselines as a single branch in Fig. 3 (RNN or CNN) as backbones with the encoder-decoder framework (no *shifting*). We refer to them as RNN-RNN and CNN-CNN in Table 1.

[1] https://github.com/JimengShi/ParaRCNN-Time-Series-Forecasting.

Algorithm 1 Model Training

```
Input: covariate time series: $\mathbf{X}_{1,T}, \mathbf{Y}_{1,T}$;
       target time series: $\mathbf{Z}_{1,T}$, where $T$ is the total length of data set..
Parameter: $w$: sliding window length, $k$: forecasting length, $s$: shifted length.
Output: well-trained model
 1: // construct training instance pairs
 2: $D \leftarrow \emptyset$
 3: for each available time point $w \leq t \leq T - s$ do
 4:     $S_{past} \leftarrow \{\mathbf{X}_{t-w+1,t}, \mathbf{Y}_{t-w+1,t}, \mathbf{Z}_{t-w+1,t}\}$
 5:     $S_{shifted} \leftarrow \{\mathbf{Y}_{t-w+1+s,t+s}\}$
 6:     $S_{target} \leftarrow \mathbf{Z}_{t+1,t+k}$
 7:     put a instance pair $(\{S_{past}, S_{shifted}\}, S_{target})$ into $D$
 8: end for
 9: // train the model
10: initialize all learnable parameters $\theta$ for the model
11: repeat
12:     randomly selects a batch of instance pairs $D_b$ from $D$
13:     model outputs $\hat{\mathbf{Z}}_{t+1,t+k}$ for each batch
14:     finds $\theta$ by minimizing the loss function in Eq. (12)
15: until stopping criteria is satisfied
16: return well-trained model with the best parameters $\theta$
```

All models were trained by minimizing the loss function in Eq. (12), which describes the mean square error between predicted and ground-truth values. The training process is given as Algorithm 1. The testing process is achieved by the trained model with the same data processing as the first 8 rows. Mean Absolute Errors (MAEs) and Root Mean Square Errors (RMSEs) are the metrics to evaluate the trained models. Each experiment was repeated 5 times with 5 random seeds. Table 1 reports the average results with an error bound.

$$L(\mathbf{Z}, \hat{\mathbf{Z}}) = \frac{1}{\Phi} \sum_{\phi=1}^{\Phi} [(\mathbf{Z}^N_{t+1,t+k})^\phi - (\hat{\mathbf{Z}}^N_{t+1,t+k})^\phi]^2. \tag{12}$$

5.3 Hyperparameter Study

Shift length. Fig. 7 shows the MAEs and RMSEs using ParaRCNN with different shifting lengths on the `Water-stage` dataset, which can help us to delineate the relationship between the shifting lengths and the model performance. We observed that $k \leq s \leq w$ generates better performance.

Model layers. After ensuring the shift length $s = k$, we try to analyze the best number of layers for the ParaRCNN model. We found 3 or 4 layers (see Fig. 8) perform the best for the datasets in our paper (3 layers for `Electricity` dataset, 4 layers for `Water-stage` and `PM2.5` dataset).

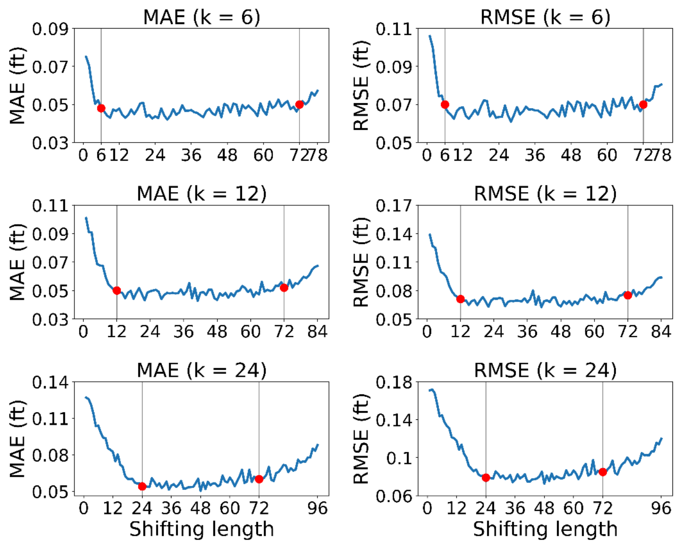


Fig. 7. MAE & RMSE for different forecasting lengths (k) and shift lengths (s). The left red point of each subplot represents the errors when $s = k$ while the right red point denotes the errors when $s = w$. ($w = 72$ hours, $k = 6, 12, 24$ hours.)

5.4 Prediction Results

The first 5 rows in Table 1 show the performance of the baseline models. Our models are listed in the last 5 rows. We use a single RNN architecture of RNN-RNN and apply *shifting* to it as RNN-Shift. To test the effectiveness of *skip connection*, we add it to RNN-Shift and call it RNN-Shift-SC. A similar process is applied to CNN-Shift and CNN-Shift-SC. At last, we propose ParaRCNN (see Fig. 3) by combining RNN and CNN in parallel with both *shifting* and *skip connection* techniques. Compared with baseline models in Table 1, the performance of models with *shifting* is comparable or slightly better than some baselines, while ParaRCNN achieves the best with the help of *shifting* and *skip connection.*

5.5 Skip Connection Study

We apply *skip connection* with different strategies (see Fig. 6) to the ParaRNN model. Taking an example of L layers, there are five situations considered: (a) **One** skip in U-net [24]; (b) $\mathbf{L(L+1)/2}$ skips in DenseNet [12]; (c) $\mathbf{L}$ skips; (d) $\mathbf{L}$ skips; and (e) **No** skip connection. Fig. 9 shows the performance of (e) without a skip connection is clearly much poor than others and (a-d) is roughly the same. The possible reason is that our network is a shallow one with only 4 layers. $\mathbf{L}$ and $\mathbf{L(L+1)/2}$ skips do not exist a big difference. However, the number of skip connections is indeed reduced from $\mathbf{L(L+1)/2}$ to $\mathbf{L}$.

Table 1. MAEs & RMSEs with $k = 24$ hours on the test sets.

Methods	Beijing PM2.5		Electricity Price		Water Stage	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
MQRNN	33.94±1.14	53.13±1.22	3.48±0.14	4.69±0.19	0.121±1e-2	0.156±4e-2
DeepAR	36.57±0.72	57.75±0.98	5.23±0.12	6.59±0.18	0.196±9e-3	0.231±1e-2
TFT	36.32±0.82	60.13±1.37	3.76±0.16	5.52±0.24	0.119±7e-3	0.158±9e-3
RNN-RNN	33.43±0.79	52.43±1.15	4.27±0.15	5.72±0.26	0.142±4e-3	0.177±8e-3
CNN-CNN	33.90±0.57	53.15±1.22	3.78±0.14	5.08±0.21	0.110±8e-3	0.177±9e-3
RNN-Shift	33.37±0.59	52.96±1.27	3.96±0.13	5.23±0.24	0.109±1e-2	0.151±9e-3
RNN-Shift-SC	31.90±0.55	50.89±1.09	3.49±0.12	4.65±0.18	0.071±7e-3	0.096±7e-3
CNN-Shift	33.55±0.46	52.94±1.11	3.85±0.14	5.09±0.20	0.131±8e-3	0.158±9e-3
CNN-Shift-SC	31.76±0.43	50.61±1.08	3.48±0.12	4.69±0.17	0.059±5e-3	0.081±6e-3
ParaRCNN	31.48±0.36	49.97±0.89	3.39±0.10	4.60±0.13	0.054±4e-3	0.075±9e-3

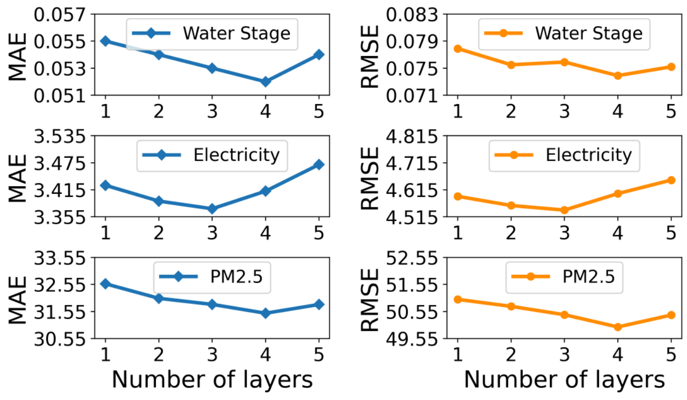


Fig. 8. Model performance vs. Number of layers ($w = 72$ hours, $k = 24$ hours).

5.6 Model Explainability

After the model was trained, we analyzed how much each time step and feature contribute to the final outputs. With the Grad-CAM algorithm [26], we first compute the gradient of the target values with respect to the feature map activations of the concatenated layer. These gradients flow back over the input of shape (time steps × features) to obtain the neuron importance weights (see Fig. 10 in Appendix A). The water stage at S1, S25A, S25B, and S26 are target values to predict. The first 19 rows are original past input and the last 9 rows shifted covariates (shifted future covariates are from 48 to 72). It shows our model pays more attention to these future covariates since target variables in the future horizon have a dependent relationship with them. This is as described in Section 4.1. The water stages at different stations are more correlated because these stations are adjacent to the ocean and water stages are changing with the trend of the tide (WS_S4). We visualize each time series in Appendix B (see Figs. 11 and 12) providing better observations for readers.

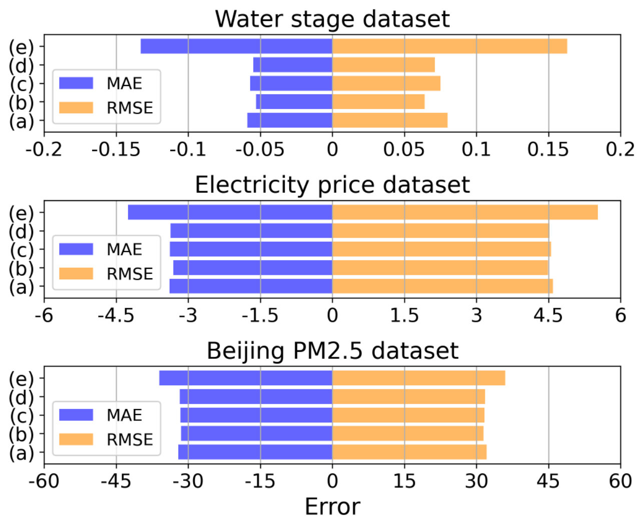


Fig. 9. MAE & RMSE for various strategies to implement *skip connection*. (a) **One** skip in U-net [24]; (b) $\mathbf{L(L+1)/2}$ skips in DenseNet [12]; (c) **L** skips as one benchmark; (d) **L** skips we are using in our paper; and (e) **No** skip connection.

6 Discussion and Conclusions

We have demonstrated with experiments that the utilization of future covariates can enhance performance. The model explainability shows their importance from another point of view. To take the advantage of future covariates, the proposed data fusion method, *shifting*, can generate comparable or slightly better performance with a single compact model. Besides, our experiments delineate an appropriate range of the *shift* length (see Fig. 7). When $s < k$ or $s > w$, considerably lower performances occur since the models only get to utilize some of the predicted covariates from the future k time steps. However, when $s > k$, either the performance is flat or deteriorates as s is increased. We observe that $k \leq s \leq w$ generates better performance since all predicted future covariates in the forecasting horizon are included. The variations for $k \leq s \leq w$ are too small to be significant.

Skip connection can further improve the model performance. Our implementation strategy that presents the original input to each subsequent layer generated roughly the same or better performance when compared with other strategies. Our *ParaRCNN* model equipped with *shifting* and *skip connection* techniques consistently outperformed all other models in our paper.

References

1. Baybuza, I.: Inflation forecasting using machine learning methods. Russian Journal of Money and Finance **77**(4), 42–59 (2018)
2. Böse, J., Valentin F., Jan G., Tim J., Dustin L., David S., Sebastian S., Matthias S., Wang Y.: Probabilistic demand forecasting at scale. In: Proceedings of the VLDB Endowment 10, pp. 1694–1705 (2017)
3. Chen, P., Pedersen, T., Bak-Jensen, B., Chen, Z.: ARIMA-based time series model of stochastic wind power generation. IEEE transactions on power systems **25**(2), 667–676 (2009)
4. Chen, Y., Kang, Y., Chen, Y., Wang, Z.: Probabilistic forecasting with temporal convolutional neural network. Neurocomputing **399**, 491–501 (2020)
5. Cinar, Y., Hamid, M., Parantapa G., Eric G., Ali A., Vadim S.: Position-based content attention for time series forecasting with sequence-to-sequence RNNs. In: NeurIPS, vol. 24, pp. 533–544 (2017)
6. Dong, G., Fataliyev, K., Wang, L.: One-step and multi-step ahead stock prediction using backpropagation neural networks. In: 9th International Conference on Information, Communications & Signal Processing, pp. 1–5 (2013)
7. Du, S., Li, T., Yang, Y., Horng, S. J.: Multivariate time series forecasting via attention-based encoder–decoder framework. Neurocomputing **388**, 269-279 (2020)
8. Efendi, R., Arbaiy, N., Deris, M.: A new procedure in stock market forecasting based on fuzzy random auto-regression time series model. Information Sciences **441**, 113–132 (2018)
9. Fan, C., Zhang, Y., Pan, Y., Li, X., Zhang, C., Yuan, R., Wu, D., Wang, W., Pei, J., Huang, H.: Multi-horizon time series forecasting with temporal attention learning. In: SIGKDD. pp. 2527–2535 (2019)
10. Fauvel, K., Lin, T., Masson, V., Fromont, É., Termier, A.: Xcm: An explainable convolutional neural network for multivariate time series classification. Mathematics. Mathematics **9**(23), 3137 (2021)
11. Gidea, M., Katz, Y.: Topological data analysis of financial time series: Landscapes of crashes. Physica A: Statistical Mechanics and its Applications, **491**, 820–834 (2018)
12. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K. Q.: Densely connected convolutional networks. In: CVPR. pp. 4700–4708 (2017)
13. Hyndman, R., Koehler, A., Ord, J., Snyder, R.: Forecasting with exponential smoothing: the state space approach. Springer Science & Business Media (2008)
14. Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A., Bengio, Y.: Professor forcing: A new algorithm for training recurrent networks. In: NeurIPS (2016)
15. Tang, W., Long, G., Liu, L., Zhou, T., Jiang, J., Blumenstein, M.: Rethinking 1d-cnn for time series classification: A stronger baseline. arXiv preprint arXiv:2002.10061 (2020)
16. Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y. X., Yan, X.: Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In: NeurIPS (2019)
17. Li, L., Noorian, F., Moss, D. J., Leong, P.: Rolling window time series prediction using MapReduce. In: Proceedings of the 2014 IEEE 15th international conference on information reuse and integration, pp. 757-764. IEEE (2014)
18. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: NeurIPS (2018)

19. Lim, B., Arık, S. Ö., Loeff, N., Pfister, T.: Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting **37**(4), 1748-1764 (2021)
20. Luong, M. T., Le, Q. V., Sutskever, I., Vinyals, O., Kaiser, L.: Multi-task sequence to sequence learning. arXiv preprint arXiv:1511.06114 (2015)
21. Niu, W., Feng, Z.: Evaluating the performances of several artificial intelligence methods in forecasting daily streamflow time series for sustainable water resources management. Sustainable Cities and Society **64**, 102562 (2021)
22. Qing, X., Jin, J., Niu, Y., Zhao, S.: Time–space coupled learning method for model reduction of distributed parameter systems with encoder-decoder and RNN. AIChE Journal **66**(8), e16251 (2020)
23. Rangapuram, S., Seeger, M., Gasthaus, J., Stella, L., Wang, Y., Januschowski, T.: Deep state space models for time series forecasting. In: NeurIPS (2018)
24. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference. pp. 234–241. Springer International Publishing (2015)
25. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**(3), 1181-1191 (2020)
26. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: ICCV. pp. 618–626 (2017)
27. Shi, J., Jain, M., Narasimhan, G.: Time series forecasting (tsf) using various deep learning models. arXiv preprint arXiv:2204.11115 (2022)
28. Siami-Namini, S., Tavakoli, N., Namin, A.: A comparison of ARIMA and LSTM in forecasting time series. In: 17th IEEE international conference on machine learning and applications (ICMLA), pp. 1394–1401. IEEE (2018)
29. Therneau, T., Crowson, C., Atkinson, E.: Using time dependent covariates and time dependent coefficients in the cox model. Survival Vignettes **2**(3), 1–25 (2017)
30. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Polosukhin, I.: Attention is all you need. In: NeurIPS (2017)
31. Wen, R., Torkkola, K., Narayanaswamy, B., Madeka, D.: A multi-horizon quantile recurrent forecaster. arXiv preprint arXiv:1711.11053 (2017)
32. Yang, B., Oh, M., Tan, A.: Machine condition prognosis based on regression trees and one-step-ahead prediction. Mechanical Systems and Signal Processing **22**(5), 1179–1193 (2008)
33. Yang, X., Ye, Y., Li, X., Lau, R. Y., Zhang, X., Huang, X.: Hyperspectral image classification with deep learning models. IEEE Transactions on Geoscience and Remote Sensing **56**(9), 5408-5423. (2018)
34. Yuan, Y., Lin, L., Huo, L. Z., Kong, Y. L., Zhou, Z. G., Wu, B., Jia, Y.: Using an attention-based LSTM encoder–decoder network for near real-time disturbance detection. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **13**, 1819-1832 (2020)
35. Zhang, J., Nawata, K.: Multi-step prediction for influenza outbreak by an adjusted long short-term memory. Epidemiology & Infection **146**(7), 809–816 (2018)
36. Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Chawla, N. V.: A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In: AAAI (2019)

7 Appendix

7.1 Model Explainability

We provide the explainability of the trained model using the Water Stage dataset. The following figure shows how important each feature and each time step is for the final predictions.

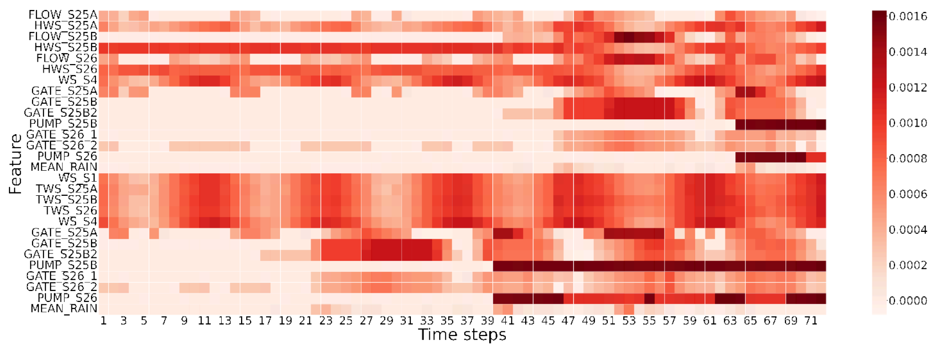
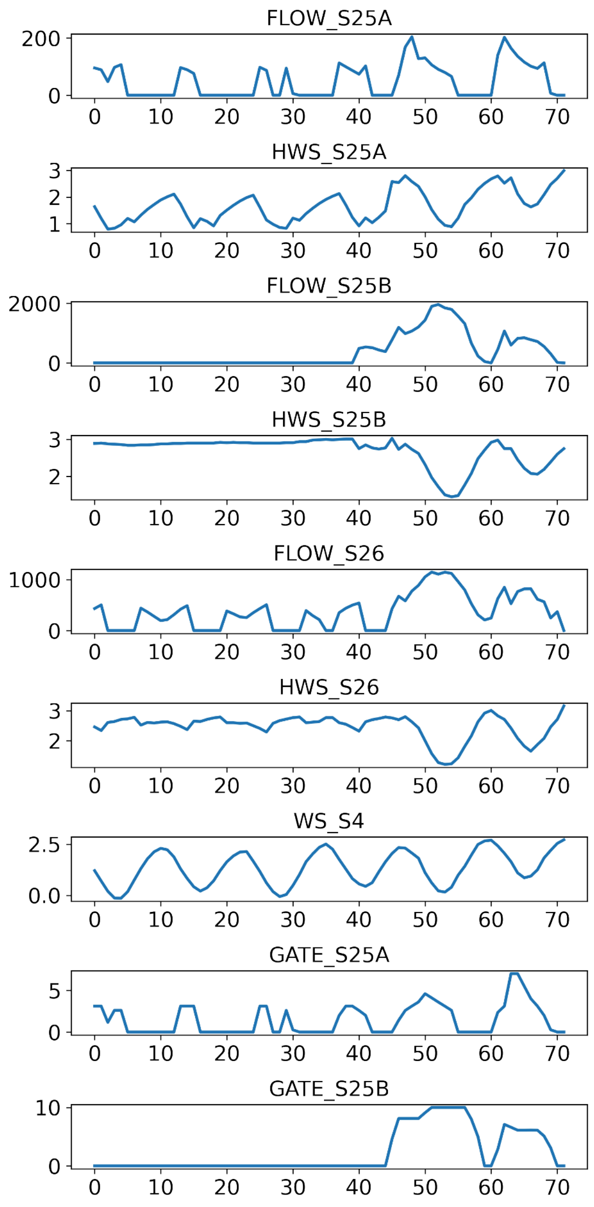


Fig. 10. Importance weights (feature vs. time step) with Grad-CAM algorithm.

7.2 Visualization of Time Series

We visualize the time series in Fig. 10 below. The unit of each feature is ignored. We refer readers to see Fig. 2 for better understanding.



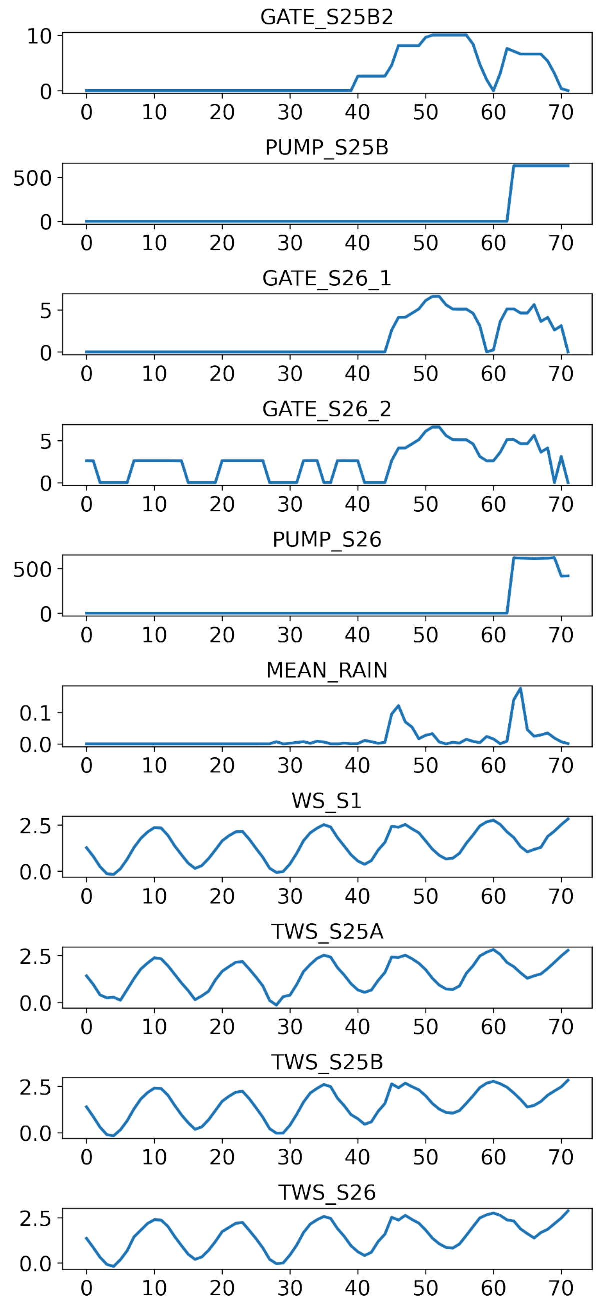


Fig. 11. Visualization of target variables and covariates from the past.

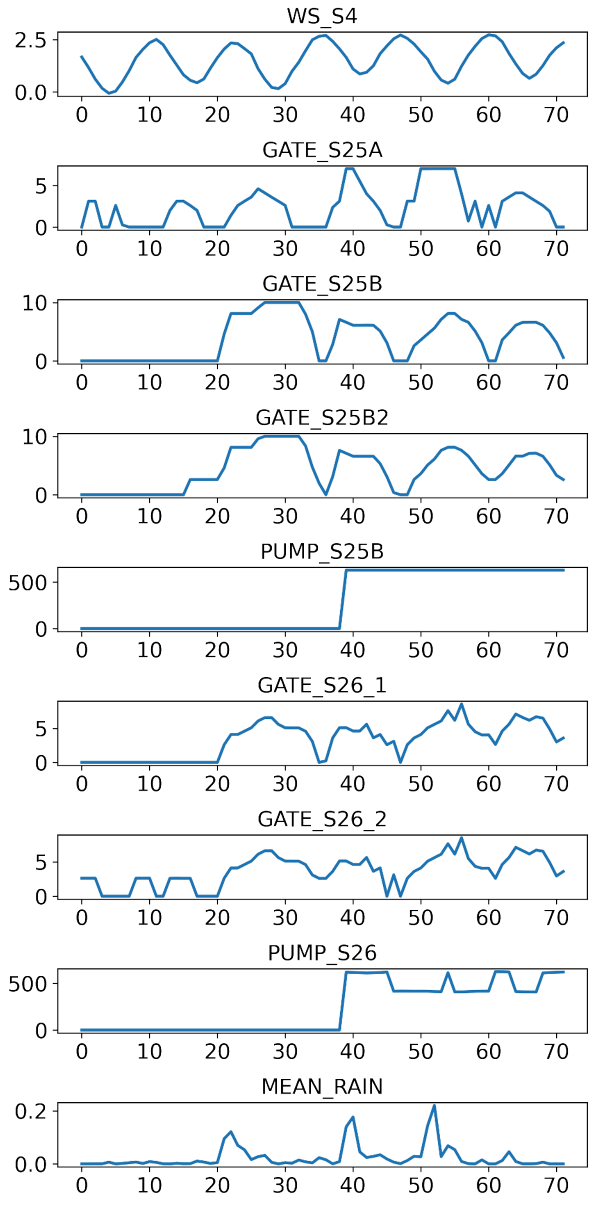


Fig. 12. Visualization of shifted future predictable covariates.