Expert Systems With Applications 200 (2022) 116966

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Probabilistic deep learning model as a tool for supporting the fast simulation of a thermal–hydraulic code

Seunghyoung Ryu [a], Hyeonmin Kim [b], Seung Geun Kim [a], Kyungho Jin [b], Jaehyun Cho [b], Jinkyun Park [b,*]

[a] Artificial Intelligence Application & Strategy Team, Korea Atomic Energy Research Institute, 111, Daedeok-daero 989 beon-gil, Daejeon, 34057, South Korea
[b] Risk Assessment Research Team, Korea Atomic Energy Research Institute, 111, Daedeok-daero 989 beon-gil, Daejeon 34057, South Korea

ARTICLE INFO

Keywords:
Deep learning
LSTM
Surrogate model
Thermal-hydraulic code
Recurrent neural networks
Quantile regression
Positional encoding

ABSTRACT

Following the Fukushima Daiichi accident, enhancing the safety of nuclear power plants has become the priority mission for the future of nuclear energy. Probabilistic safety assessment (PSA) is a well-known technique to quantify the anticipated risk of nuclear power plants according to the accident scenario. One approach to strengthening nuclear safety is reducing the uncertainty in PSA by analyzing a wide spectrum of accident scenarios. In doing this, massive simulations of thermal–hydraulic (TH) dynamics are required by running TH code. As the computation time for such large-scale simulation is a heavy burden, it is necessary to develop a fast simulation model, but related research has recently begun. For doing this, conditional autoencoder was firstly introduced, however, prediction accuracy can be further improved by exploiting temporal characteristics of simulation data. In this paper, we formalize deep learning-based fast simulation model of TH code, and propose a novel deep learning model, namely ensemble quantile recurrent neural network (eQRNN). By leveraging bi-directional long short-term memory, concatenative positional encoding, quantile regression, and model ensemble, the proposed eQRNN can provide a more accurate prediction on the simulation results and uncertain boundaries of its prediction. Compared to the base RNN model, the proposed eQRNN shows 39% and 28% lower error overall in terms of mean absolute percentage error (MAPE) and mean squared error (MSE). Finally, eQRNN achieved 75%, 79% lower MAPE and MSE than the existing conditional autoencoder-based model.

1. Introduction

After the Fukushima Daiichi accident in 2011, safety enhancement in utilizing nuclear energy became the number one priority in the nuclear industry. Since then, various research to strengthen the safety of nuclear power plants (NPPs) has been actively conducted worldwide. During the same period, great technological advancements led to the *fourth industrial revolution* involving artificial intelligence, big data, and the internet of things. In particular, the deep learning technique is the most rapidly emerging (and still developing) technology in the last decade. Through diverse deep learning techniques, machines have been able to catch up with or outperform human performance in various applications including but not limited to natural language processing (Brown et al., 2020), computer vision (Brock, De, Smith, & Simonyan, 2021), and the game Go (Silver et al., 2017), areas in which it had been almost impossible to surpass human performance in the past.

Based on these big successes, the use of deep learning techniques is being actively adopted in various industrial sectors such as for smart factories (Wang, Ma, Zhang, Gao, & Wu, 2018), smart grids (Lai, Zhong, Pan, Ng, & Lai, 2021; Ryu, Choi, Lee, & Kim, 2019), smart cities (Chen et al., 2019; Chen, Wang, Huang, De, & Coenen, 2021), and health care (da Silva, Schmidt, da Costa, da Rosa Righi, & Eskofier, 2021; Faust, Hagiwara, Hong, Lih, & Acharya, 2018). In this respect, it is natural to expect that these deep learning techniques can also be employed in the nuclear industry to achieve the next level of nuclear safety. Although deep learning techniques in the nuclear industry are still in their infancy at this moment, huge efforts are being made for related applications. As part of this effort, typical topics in which deep learning techniques are involved for enhancing the operational safety of NPPs are event diagnosis (dos Santos, Pinheiro, do Desterro, & de Avellar, 2019; Lin, Wang, & Wu, 2021; Radaideh, Pigg, Kozlowski, Deng, & Qu, 2020), reactor anomaly detection (Caliva, de Ribeiro, & Mylonakis, 2018), sabotage

* Corresponding author.
E-mail addresses: ashryu@kaeri.re.kr (S. Ryu), hyeonmin@kaeri.re.kr (H. Kim), sgkim92@kaeri.re.kr (S.G. Kim), jin716@kaeri.re.kr (K. Jin), chojh@kaeri.re.kr (J. Cho), kshpjk@kaeri.re.kr (J. Park).

https://doi.org/10.1016/j.eswa.2022.116966
Received 10 May 2021; Received in revised form 10 November 2021; Accepted 20 March 2022
Available online 24 March 2022

detection (Chen & Demachi, 2019), signal reconstruction (Kim, Chae, & Seong, 2020), and probabilistic safety assessment (PSA) (Kim, Cho, & Park, 2020). A large portion of these studies have focused on the detection and diagnosis of an event based on general machine learning tasks (i.e., classification), while there are relatively few studies on PSA applications.

A PSA is a methodology to evaluate the safety of an NPP by quantifying the amount of anticipated risk from diverse accident scenarios that can result in an undesired consequence (e.g., core damage or the release of radioactive materials to the environment). To achieve a higher level of safety in NPPs, it is crucial to create a catalog that identifies accident scenarios as realistically as possible. Theoretically, this can be guaranteed by analyzing a broad spectrum of potential scenarios that can be generated with respect to the combination of numerous factors (e.g., the status of key components and the characteristics of human responses) and their sequences. Accordingly, if we are able to identify the consequence of each potential scenario by simulating its progression in detail, it is possible to obtain the catalog of accident scenarios. This simulation can be done by running a high precision thermal–hydraulic (TH) code, where each simulation usually takes a couple of hours depending on the nature of the combined factors.

Unfortunately, this approach is less advantageous because available resources are limited in reality. Specifically, in the case of a highly complicated system like an NPP, enormous calculation time of TH code in simulating a huge number of possible scenarios is a technical burden under finite computational resources. Therefore, the number of scenarios to be simulated by TH code is limited to a certain range in traditional PSA, and out-of-bound accident scenarios remain as the uncertainty in PSA results. In this regard, a fast and accurate surrogate model of TH code is a potential solution that minimize the uncertainty in PSA. The fast simulation model has advantages in the following points. First, it expands the spectrum of accident scenarios within a limited computational time at the expense of a small fraction of accuracy. Therefore, the uncertainty in PSA can be mitigated as the number of scenario is increased. Second, the fast simulation model allows us to simulate a specific scenario in real-time, thus it can be further embedded in the field operations.

Various methodologies can be used for building fast surrogate of TH code, for example, reduced order modelling (Mandelli et al., 2016), optimization algorithms (Abualigah, Diabat, Mirjalili, Abd Elaziz, & Gandomi, 2021; Abualigah, Yousri, et al., 2021; Abualigah & Diabat, 2021), and deep learning (Kim et al., 2020). In this work, we build surrogate in a data-driven way by leveraging deep learning. Deep learning based fast simulation model of TH code was firstly introduced by (Kim et al., 2020). With fully-connected neural network (FNN) as a back-bone network structure, two-staged conditional autoencoder was utilized for modeling latent feature space of TH code data effectively. However, prediction accuracy can be further improved by leveraging recurrent neural networks (RNN) that could effectively exploit temporal features of time-series data. In doing this, we propose an ensemble quantile recurrent neural network (eQRNN)-based in-situ model that adopts bi-directional long short-term memory (LSTM), concatenative positional encoding (PE), quantile regression, and model ensemble. By combining these key elements, the proposed model shows improved accuracy and provides the uncertain boundaries of prediction results. Our contributions can be summarized as follows.

· We formalize the framework of deep learning-based surrogate model of TH code, and propose a novel deep neural network model for TH code, i.e., eQRNN. By training the trajectory of key reactor variables, the proposed eQRNN model shows the highest prediction accuracy compared to the other baseline and existing models.

· To improve the model accuracy, we adopt bi-directional LSTM and propose concatenative PE. With concatenative PE, the RNN model could achieve 16.31% lower MAPE on average compared to the model without PE.

· By applying multiple quantile regression with pinball loss, the QRNN model gives a prediction of median as well as target quantiles to provide the information of uncertain boundary of model results. Moreover, experimental result shows that converting to a quantile model improves the performance by leveraging the advantage of multi-task learning.

· Since the deep learning model manipulates the data in batch manner, the result of multiple scenarios can be obtained at once through trained eQRNN model, thus required computational time is drastically reduced in seconds.

The rest of this paper is organized as follows. In Section 2, as the background information of this study, a brief introduction about the limitations of the traditional PSA approach is given along with the necessity for a fast simulation technique. Section 3 then describes the overall framework and proposed methodologies. After that, in Section 4, the results of case studies are compared based on six different models. Finally, the conclusions of this study are drawn in Section 5 with associated discussion and future work.

2. Background information

As mentioned at the end of Section 1, there are several limitations in the traditional PSA approach, and one of them is the uncertainty of PSA results (Aldemir, 2013). In order to soundly resolve this limitation, it is necessary to incorporate a breakthrough technology such as a deep learning technique. In this section, the limitation pertaining to the identification of accident scenarios is explained with underlying idea for developing a surrogate model of TH code.

2.1. Limitation in identifying accident scenarios

In general, it is anticipated that the more accident scenarios we distinguish, the more realistic PSA results we have. To clarify this, let us consider a hypothetical system as depicted in Fig. 1, which consists of four components (*Tank 1*, *Valves B* and *C*, and *Safety Disk*) with an annunciator warning of a high water level in Tank 1.

As shown in Fig. 1, a human operator can manually open or close two valves (*Valve B* and *Valve C*) in order to maintain the water level of *Tank 1* within a specific range. In addition, when the water level of *Tank 1* exceeds a predefined set-point, an automatic control signal to close *Valve B* is generated with a warning alarm to inform the human operator. In this case, it is possible to develop diverse potential scenarios by combining the status of the components with the responses of the human operator (hereafter the term *Event Heading* will be used for representing the status of each component). Then, for example, if an accident of this hypothetical system is defined as a rupture of the *Safety Disk*, a catalog of accident scenarios can be picked out from the inventory of potential scenarios. Fig. 2 illustrates seven potential scenarios and three accident scenarios (shaded in gray) developed from four *Event Headings*.

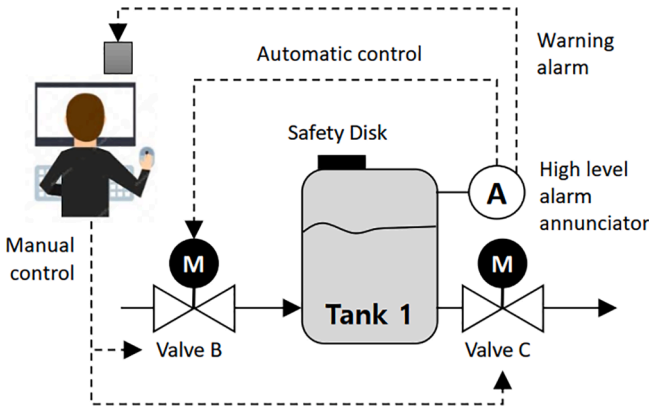


Fig. 1. Configuration of hypothetical system; modified from (Park, 2018).

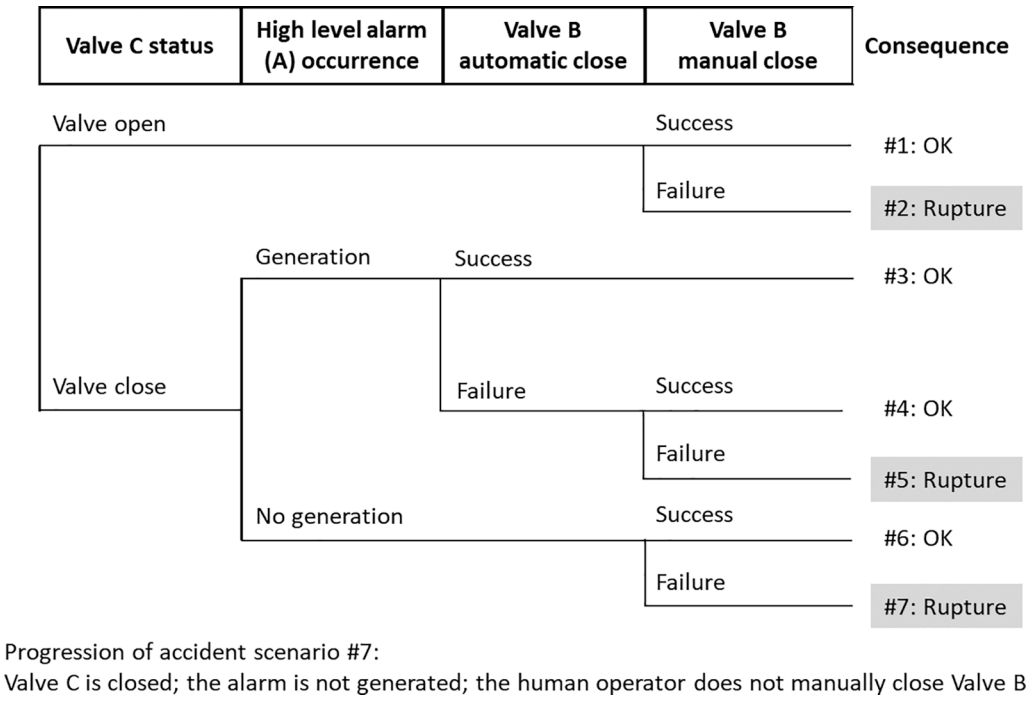


Fig. 2. Example of potential scenarios and accident scenarios for the hypothetical system; modified from (Park, 2018).

That is, of seven potential scenarios, it is expected that three scenarios (#2, #5, and #7 in Fig. 2) result in the rupture of the *Safety Disk*. It should be noted that running a TH code is necessary to determine the consequence of each potential scenario. If we are able to estimate the occurrence frequency of each accident scenario, the risk of this hypothetical system corresponds to the sum of the occurrence frequencies of all accident scenarios. Actually, although the abovementioned explanation is too simple and exaggerated to fully exhibit the nature of the PSA approach including its significant benefits, it is possible to say that the very first step to visualize the risk of any system is to identify what can go wrong (e.g., accident scenarios).

Accordingly, the identification of accident scenarios is one of the crucial factors affecting the uncertainty of PSA results. It is natural then to expect that the more accident scenarios we are able to identify for a target system, the more the uncertainty of PSA results decreases. Unfortunately, in reality, this idea is unrealistic for complicated systems such as NPPs because the number of components to be considered for the development of potential scenarios drastically goes up (i.e., an increase of *Event Headings* in Fig. 2). In addition, the binary branch of each *Event Heading* seems to be insufficient for representing the actual progression of each potential scenario.

In order to clarify the latter aspect, let us consider the response of the human operator pertaining to the *Event Heading* 'Valve B manual close'. In Fig. 2, only two conditions (*Success* and *Failure*) were considered to describe the progression of the potential scenarios. However, it is more relevant to assume that this response should be conducted when the human operator properly recognizes the existence of the high-water level alarm. This implies that, in terms of reducing the uncertainty of PSA results, it is necessary to incorporate a spectrum of response times into the progression of the potential scenarios (refer to Fig. 3). However, computation time of simulating thermal–hydraulic dynamics in NPP to obtain the consequence of each scenario is critical; it takes a couple of hours depending on the nature of the combined factors. Considering the number of potential scenarios to be analyzed, it is indispensable to secure a breakthrough technology that allows us to extremely shorten the run time of the TH code. For this reason, the use of a deep learning

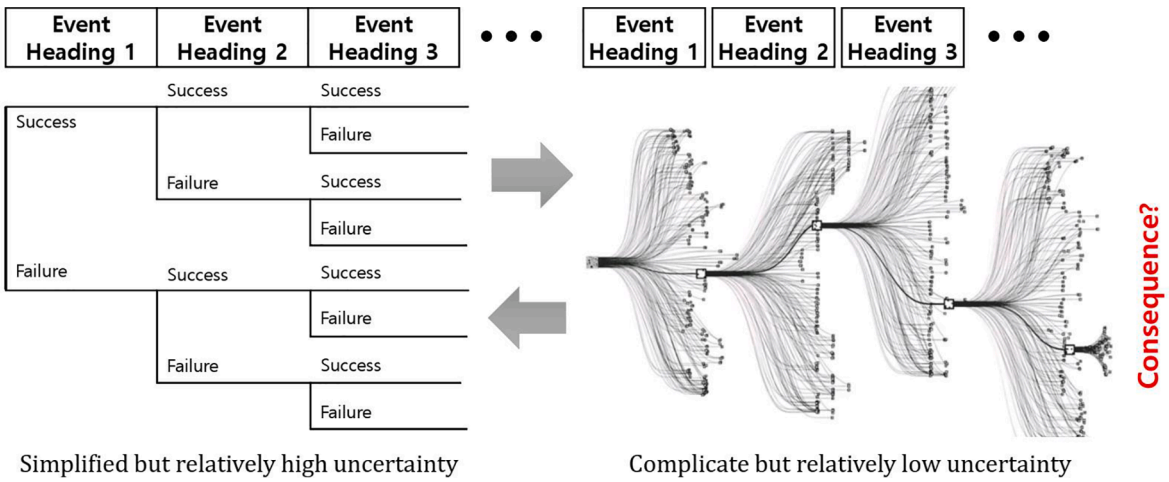


Fig. 3. Underlying idea to reduce the uncertainty of PSA results; modified from (Park & Yoon, 2018).

technique is considered in this study.

2.2. Development of a surrogate model from a deep learning perspective

In this section, we formulate the operation of a TH code in the following function form to build a deep learning surrogate model. Let $\mathbf{x} = [x_1, x_2, \cdots, x_d]^T \in \mathbb{R}^d$ be an input column vector of the TH code, where x_i contains information of the events or actions that correspond to *Event Headings* in Fig. 2. Then the TH code computes its output $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_T]^T \in \mathbb{R}^{T\times N}$ from $\mathbf{x}$, where $\mathbf{y}_t$ is observation of N output process parameters at time t. In short, the TH code is a function of $\mathbf{x}$ with an output of multivariate time-series data representing the trajectories of diverse process parameters. In this paper, we refer to this function $\mathbf{Y} = f(\mathbf{x})$ as the external form of the TH code, whereas the internal form calculates the results $\mathbf{y}_t$ for each time-step autoregressively.

The original input vector $\mathbf{x}$ consists of different state variables. Let a indicate a specific action that affects the progression of a potential scenario. Then, the information of a can be represented with two state variables $x^a = x_k$ and $t^a = x_{k+1}$, where x^a represents the degree of the operator's manual control or component status (e.g., degree of valve opening), and t^a corresponds to the time point when x^a occurred (e.g., timing of valve opening). This pair of degree–timing state variables (x^a, t^a) can be reformulated as a sequence $z^a = [z_1^a, z_2^a, \cdots, z_T^a]$ by setting $z_t^a = x^a$ when $t = t^a$; otherwise $z_t^a = 0$. Then the available information at current time-step t can be denoted as $z_t = \left[z_t^1, \cdots z_t^{d'}\right]^T$. After converting input vector $\mathbf{x}$ into time-series $\boldsymbol{Z} = [z_1, \cdots, z_T]^T \in \mathbb{R}^{T\times d'}$, the TH code calculates the current output $\mathbf{y}_t$ based on the past inputs $z_1, \cdots, z_t$, i.e., $\mathbf{y}_t = g(z_1, \cdots, z_t) = g(z_t, \mathbf{h}_{t-1})$, where $\mathbf{h}$ is a hidden representation of the past sequence.

Following the above formulation, we classify the TH code surrogate into two classes: i) in-situ model and ii) stepwise model. The in-situ model is a surrogate of external function $\mathbf{Y} = f(\mathbf{x})$, while the stepwise model is a surrogate of internal function $\mathbf{y}_t = g(z_t, \mathbf{h}_{t-1})$. The main difference between the two models is whether the action information is available in advance or not. The advantage of the in-situ model is its capability to leverage action information in advance. Hence it is easier to train and shows more accurate results within a fixed time window. On the other hand, the stepwise model works more similarly to the internal operation of TH code. Generally, TH code calculates the expected behavior (process parameter) of a target system (e.g., an NPP) at the next step based on its physics model and the associated states. Thus, stepwise model reflects the effect of the degree and the time of an action in real-time. Also, the length of the output is not limited to the specific time window because it predicts the output of the next time step recursively. However, the major drawbacks of the stepwise model are training difficulty due to sparsity of $\boldsymbol{Z}$ in temporal domain, and the accumulation of prediction errors as the prediction is performed recursively.

In this paper, we focus on the development of the in-situ model, i.e., a surrogate for $\mathbf{Y} = f(\mathbf{x})$. To this end, with the massive input and output data obtained from TH code $\{(\mathbf{x}_i, \mathbf{Y}_i)\}$, a deep learning model should be trained so that it can generate the trajectories of the process parameters even from untrained scenarios. Indeed, the feasibility of the concept of a deep learning-based surrogate model was successfully demonstrated recently (Kim et al., 2020); in this study, a prototype of the in-situ model based on the two-staged conditional autoencoder (cAE) was introduced. The unique feature of the cAE model is that the latent space of the in-situ model is pre-trained via an autoencoder. In the first stage, the network learns to extract the important features by unsupervised learning of the trajectories. Then in the second stage, the in-situ model is fine-tuned from the pre-trained network where the latent vector from autoencoder is added to the input scenario vector. This two-stage approach helps to increase training efficiency and shows more accurate prediction results compared to fully connected neural network (FNN) and recurrent neural network (RNN) architectures. However, it was observed that the model has lower accuracy when the trajectory fluctuates. Therefore, to increase model accuracy with a simplified training process, a novel architecture, namely eQRNN, is proposed in this study.

3. Framework and methodologies

To provide a more accurate and reliable surrogate model for TH code, we propose a novel in-situ model by leveraging four key elements: a) bi-directional LSTM, b) concatenative PE, c) quantile regression, and d) model ensemble. In this section, we illustrate the overall framework of the proposed model, and describe the operation of the key elements.

3.1. Overall model framework

The overall framework of the proposed model is illustrated in Fig. 4. First, we build a TH code database by running Multi-Dimensional Analysis of Reactor Safety (MARS) code. Then TH code input $\mathbf{x}$ (state variables) and output $\mathbf{Y}$ (process parameter) are each divided into training, validation, and test sets. Next, channel-wise Z-score normalization is performed for both $\mathbf{x}$ and $\mathbf{Y}$ in the data preprocessing stage. Hence, x_k becomes $(x_k - \mu_k)/\sigma_k$, where μ_k and σ_k are the mean and standard deviation of x_k. The dotted box in Fig. 4 illustrates the structure of a single QRNN model. It takes normalized $\mathbf{x}$ as an input and predicts $\mathbf{Y}_\tau$, where τ is the predetermined target quantile. After the neural network receives the input vector, it expands the dimension of $\mathbf{x}$ along the time axis. Then, PE vectors $\mathbf{p}_t$ are concatenated to $\mathbf{x}_t$. After applying PE, the converted input vectors (i.e., $\left[\mathbf{x}_t^T, \mathbf{p}_t^T\right]^T$) are fed to the bi-directional LSTM layers and subsequently to the dense layers, where the last dense layer generates a value corresponding to the target quantile. In our experiment, we select the target quantile $\tau = \{0.1, 0.3, 0.5, 0.7, 0.9\}$, which covers predictions with a confidence interval between 10% and 90%. Finally, multiple QRNN models can be trained by setting different random seed and weight initializations for model ensemble.

In following subsections, we describe the operation of the a) bi-directional LSTM, b) concatenative PE, c) quantile regression, and d) model ensemble in detail.

3.2. Neural networks and bi-directional LSTM

In this section, brief explanations about existing deep learning models are given, including FNN, RNN, and bi-directional LSTM that were considered for the construction of the proposed framework.

First of all, an FNN is a basic artificial neural network structure that is composed of a number of dense layers (referred to as fully connected layers) with many nodes. A dense layer has weights, biases, and nonlinear activation functions including sigmoid, tanh, and rectified linear unit (ReLU) that perform a nonlinear transformation on the weighted summation of the layer input. Therefore, each layer works as a vector-to-vector nonlinear function that takes the output of a previous layer as the input. By stacking multiple dense layers, the FNN achieves powerful abstraction capability. As its name states, each node in the upper layer is connected to every node in the lower layer; every nodes are treated equally. Therefore, it is structurally difficult for FNN to consider the concepts such as order, location, or context between nodes. Thus, the FNN does not efficiently utilize the spatial or temporal relationships that are supposed to be implicitly involved in input vectors. This led to the development of the convolutional neural network (CNN) and the RNN, which are good at dealing with image and time-series data by incorporating spatial and temporal relationships, respectively.

An RNN is a class of neural network architecture that receives output or hidden states of the previous time-step as an input of the current time-step. Various RNN architectures have been developed so far, but long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) is the most common (and still powerful) among them. The base RNN

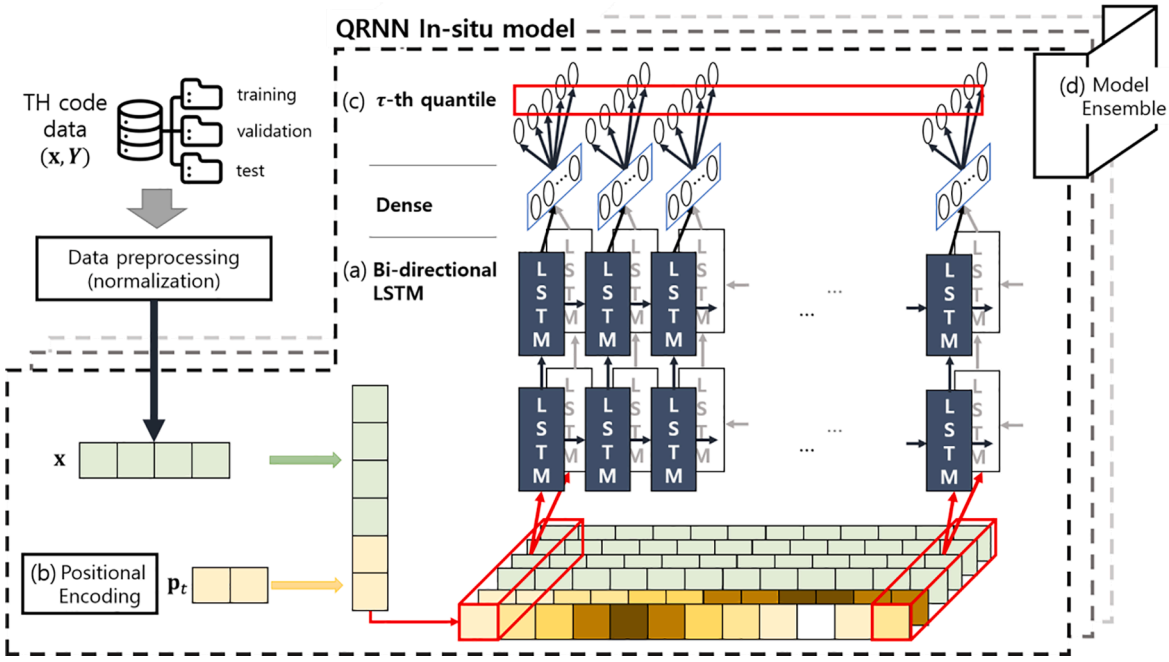


Fig. 4. Overall framework of the eQRNN in-situ model.

architecture suffers from capturing the long-term dependency of long sequential data due to several technical issues such as the vanishing gradient problem. In LSTM, there are gates that control the flow of input data, and cells that work as a memory of hidden states. The gated connection in LSTM establishes a kind of highway that conveys gradients to the distant past (Karpathy, Johnson, & Fei-Fei, 2015). This makes it possible to overcome the vanishing gradient problem so the network can learn long-term dependency. In considering problem, the output is long-term time-series data from multiple sensors, thus LSTM is suitable solution rather than FNN (general network structure) or convolutional neural network (which is specialized to image data). The structure of the original LSTM is depicted in Fig. 5. Compared to the node of an FNN, an LSTM consists of many cells with relevant weights, biases, and activation functions.

In dealing with sequential data (e.g., time-series data), tasks are typically classified into many-to-many or many-to-one problems according to the shape of the input–output pair. For example, the stepwise model in our case solves the many-to-many problem since it needs to calculate $\mathbf{y}_t$ for all time-step t with z_t. The in-situ model, however, is basically for the one-to-many problem, which is atypical, for which a simple approach is to convert it to a many-to-many problem by tiling the input vector along the time axis: $\mathbf{x}_t = \mathbf{x}, \forall t \in [0, T]$, where T is the total simulation time. Therefore, the appearance of the in-situ model is $\mathbf{Y} = f(\mathbf{x})$, and the RNN-based in-situ model learns $\mathbf{y}_t = f(\mathbf{x}_t, \boldsymbol{h}_{t-1})$, as distinct from the stepwise model $\mathbf{y}_t = g(z_t, \mathbf{h}_{t-1})$.

It should be noted that one of the key differences between the in-situ model and the stepwise model is whether action data is available in advance or not. The input vector of the in-situ model already contains the time of action; hence, it can be regarded as knowing or planning action in advance. It is therefore possible to utilize bi-directional LSTM

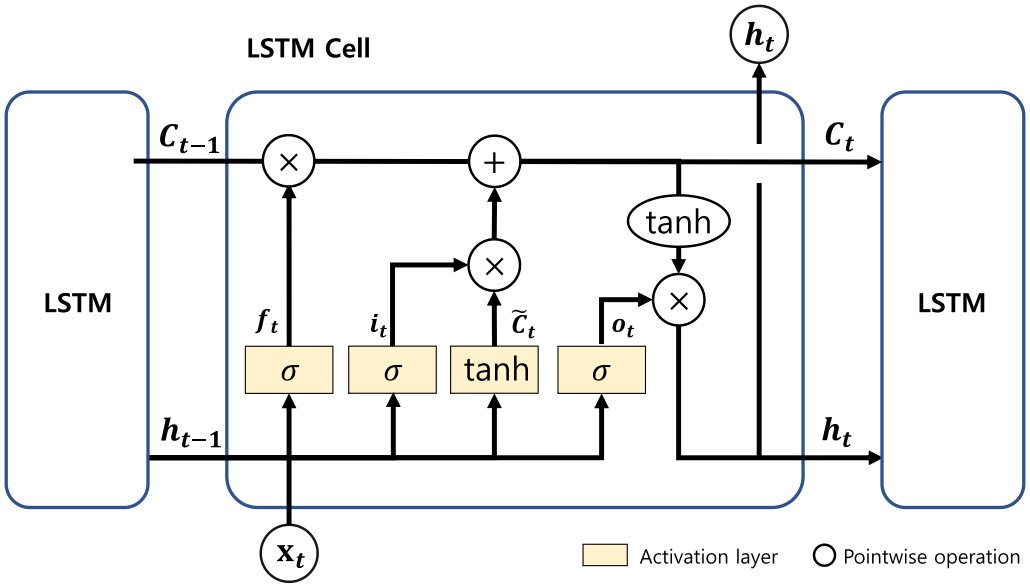


Fig. 5. The structure of LSTM; modified from (Olah, 2015).

(Schuster & Paliwal, 1997), where the information also flows backward in time (future to past). Bi-directional LSTM is composed of two LSTM layers with different directions. The forward direction layer (i.e., dark colored LSTMs in Fig. 4) is the same as the above RNN model $\mathbf{y}_t^f = f^f(\mathbf{x}_t, \mathbf{h}_{t-1})$. On the contrary, the backward direction layer (white LSTMs in Fig. 4, behind the forward direction layer) learns $\mathbf{y}_t^b = f^b(\mathbf{x}_t, \mathbf{h}_{t+1})$. Finally, $\mathbf{y}_t$ can be derived from $\mathbf{y}_t^f$ and $\mathbf{y}_t^b$.

3.3. Positional encoding

In a seminal paper of deep learning, "Attention is all you need" (Vaswani et al., 2017), PE was introduced with a transformer architecture to capture an order-related feature in text data. Although we do not utilize transformer architecture here, we apply PE with LSTM to leverage temporal information explicitly during the training and inference stages, whereas the base RNN in-situ model takes the same input vector for all time-steps. With PE, the time-step t is encoded to a vector, $\mathbf{p}_t = \left[p_{t,1}, \cdots, p_{t,d}\right]^T \in \mathbb{R}^d$. This vector can be formulated in several ways (e.g., absolute PE, relative PE (Ke, He, & Liu, 2020)), and we apply sinusoidal PE, which is generally used in many applications (Choi, Kim, & Choo, 2020; Mildenhall et al., 2020). Sinusoidal PE generates $\mathbf{p}_t$ by the following equation.

$$p_{i,j} = \begin{cases} \sin\left(t/10000^{(i/d)}\right), & \text{if } i \text{ is even,} \\ \cos\left(t/10000^{(i/d)}\right), & \text{if } i \text{ is odd.} \end{cases} \tag{1}$$

In the original work of (Vaswani et al., 2017), $\mathbf{p}_t$ has the same dimension as input vector $\mathbf{x}_t$, and the network is trained based on their summation $\hat{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{p}_t$, i.e., *additive* PE. The additive PE shares the axis between $\mathbf{x}_t$ and $\mathbf{p}_t$, and thus the order information is embedded to $\hat{\mathbf{x}}_t$. This $\hat{\mathbf{x}}_t$ can be considered as an engineered feature, but also may cause semantic distortion in the proposed framework. For example, if $\mathbf{x}_k$ denotes the degree of valve opening, then adding $p_{(\cdot),k}$ to $\mathbf{x}_k$ directly changes the meaning of the value. This can be mitigated if the dimension is high enough to prevent information loss after the summation of $\mathbf{x}_t$ and $\mathbf{p}_t$; for example, the dimension of $\mathbf{x}_t$ was 512 in (Vaswani et al., 2017). However, in case of the proposed in-situ model, the dimensions of the state (16 dimensions in our case) are relatively small to cover both state and positional information. In this regard, we propose *concatenative* PE where the input becomes $\hat{\mathbf{x}}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{p}_t \end{bmatrix}$. Thus, concatenative PE explicitly separates the scenario-space and time–space. To support this, we show that the additive PE is regularized version of concatenative PE in Proposition 1.

Proposition 1. For input vector $\mathbf{x}_t$ and PE vector $\mathbf{p}_t$ in same dimension, corresponding weight matrices W_x, W_p, additive PE is constrained on its weights so that $W_x = W_p$, whereas concatenative PE is not.

Proof. Suppose transformed input $\hat{\mathbf{x}}_t$ goes through typical fully connected layer which can be denoted as $z = \sigma(W\hat{\mathbf{x}}_t)$ where nonlinear activation function σ and weights W with inherent bias term. Let z be a d_z-dimensional vector and $\mathbf{x}_t, \mathbf{p}_t$ be d-dimensional input and PE vectors, respectively. In additive PE, $\hat{\mathbf{x}}_t$ is $[\mathbf{x}_t + \mathbf{p}_t]$ and W is $d_z \times d$ matrix. Then, $W\hat{\mathbf{x}}_t$ is $W\mathbf{x}_t + W\mathbf{p}_t$. On the other hand $\hat{\mathbf{x}}_t$ in concatenative PE is $\begin{bmatrix} \mathbf{x}_t \\ \mathbf{p}_t \end{bmatrix} \in R^{2d}$ and W is $d_z \times 2d$ matrix. By separating W into two $d_z \times d$ sub-weight matrices ($W = [W_x, W_p]$), calculation of $W\hat{\mathbf{x}}_t$ becomes $W\hat{\mathbf{x}}_t = [W_x, W_p]\begin{bmatrix} \mathbf{x}_t \\ \mathbf{p}_t \end{bmatrix} = W_x\mathbf{x}_t + W_p\mathbf{p}_t$. Comparing the two equations, additive PE is constrained on its weights W to be $W = W_x = W_p$.

Because weights in concatenative PE are not constrained on the condition of $W_x = W_p$, the accuracy may increase by finding optimal weights for $\mathbf{x}_t$ and $\mathbf{p}_t$, independently. Note that for concatenative PE, the dimension of PE vector d_{PE} can differ from $\mathbf{x}_t$ unlike additive PE, and we set d_{PE} to 8, which is half of the original input dimension.

3.4. Quantile regression with pinball loss

For some TH code results, oscillating parts are observed. They might be actual physical phenomena, or artifacts generated by the code itself. Whichever case is true, the surrogate model should be able to accurately approximate the TH code result. In addition, if probabilistic information is able to be provided to analysts, operators, and decision-makers, it is helpful to know which results of the surrogate model are accepted. For this, instead of the typical mean squared error (MSE) loss, we utilize quantile regression by training the neural network with pinball loss (Steinwart & Christmann, 2011). The quantile model is targeted to learn the median and other quantiles, whereas the base in-situ model is trained with MSE loss to predict the mean value from output distribution. For a random variable X and its cumulative distribution function $F_X(x)$, the τ-th quantile is $F_X^{-1}(\tau)$, where $\tau \in (0, 1)$. The quantile model outputs the τ-th quantile, and hence the quantile in-situ model becomes $[Y_{\tau_1}, Y_{\tau_2}, \cdots] = f(\mathbf{x})$. Since there is no exact label for target quantiles, loss needs to be calculated based on a single sample from the underlyingunknown distribution. Quantile regression can be achieved by minimizing pinball loss (Biau & Patra, 2011; Koenker & Hallock, 2001). For the τ-th quantile, the pinball loss is.

$$L_\tau(y, \hat{y}) = f(x) = \begin{cases} \tau(y - \hat{y}), & \text{if } y \geq \hat{y} \\ (\tau - 1)(y - \hat{y}), & \text{if } y < \hat{y} \end{cases} \tag{2}$$

where y is the real value and $\hat{y}$ is a prediction of the τ-th quantile. Intuitively, pinball loss is a tilted absolute error with respect to the target quantile. For low quantiles $\tau < 0.5$, pinball loss incurs more penalty when the model over-predicts (i.e., $\hat{y} \geq y$), and consequently, the network is trained to under-predict. Conversely, the network over-predicts for high quantiles $\tau > 0.5$.

3.5. Model ensemble

We also apply model ensemble to improve the generalization performance. As illustrated in Fig. 4, we build five models that have the same structure but are trained with different random seed and weight initializations. Thus, we could obtain five results for the same input vector, and the result of the ensemble model is the average of those five results. Since neural networks are powerful nonlinear function estimators, the model can be overfitted to the training set, which leads to bad performance in the inference of the test set. This is called overfitting, a problem that model ensemble can mitigate.

4. Case studies

In this section, we introduce preliminary results with simple FNN and RNN models, and then describe the data configuration, model structure, and experimental results from four target process variables.

4.1. Preliminary results with simple FNN and LSTM

Figure 6 shows a part of the preliminary results obtained from simple FNN and LSTM based in-situ models. The target process parameter is the pressure of a pressurizer (PZR). The simple FNN model has one hidden layer with 100 nodes, and the simple RNN has one LSTM layer with 128 cells and one output dense layer with a single node. Note that the FNN directly expands the input vector into sequential output data, while the LSTM model first expands the input vector along the time axis and then extracts the features, producing scalar values as univariate time-series output. As can be seen in Fig. 6, the simple FNN and RNN models predicted the trajectory of the PZR pressure with relatively low accuracy. Although the results seem to generally follow the trend of the real trajectory, there exists plenty of room for accuracy improvement.

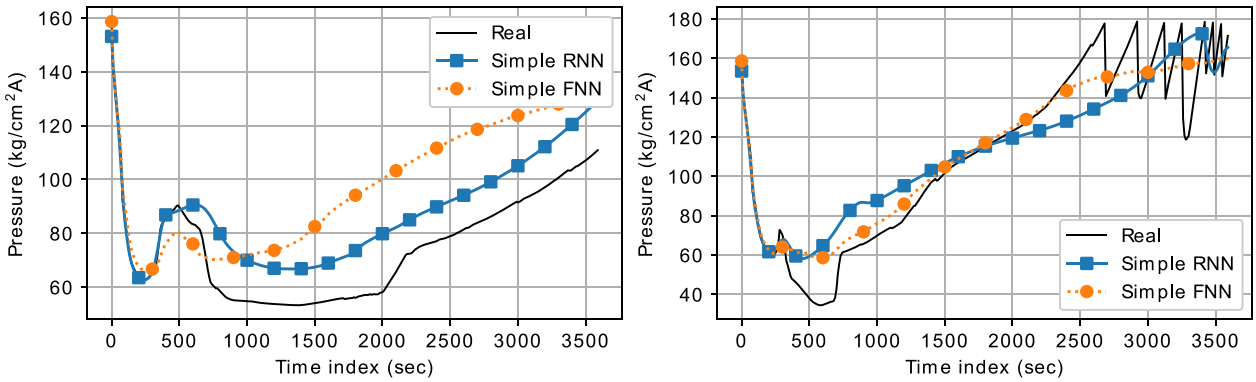


Fig. 6. Prediction examples of simple neural network models.

Specifically, these two models struggled with predicting (1) the up and down pattern around 500 s in Fig. 6a, and (2) the fluctuating pattern after 2,500 s in Fig. 6b.

4.2. Description of TH code simulation data

In order to systematically and effectively respond to an accident in an NPP, human operators working in the main control room must take a series of actions based on the instructions of the emergency operating procedures (EOPs). Accordingly, each potential scenario should include a sequence of such human operator actions (for convenience, the term *required tasks* will be used hereafter). This implies that, in order to train a deep learning model, it is essential to build a large database that contains TH code results with respect to diverse potential scenarios. For this reason, we developed a total of 104,625 potential scenarios that include the required tasks, as summarized in Table 1.

The catalog of required tasks considered in the development of the potential scenarios was carefully selected with the support of a domain expert with sufficient experience in NPP operations. To this end, two kinds of representative initiating events (IEs) and the contents of the EOPs to cope with them were meticulously reviewed. Ultimately, we decided to focus on the main steam line break (MSLB) and steam generator tube rupture (SGTR) IEs with varying break sizes: 1A, 2A, and 4A for SGTR, and 1A and 2A for MSLB. Then we set up the degree and/or timing of six required tasks that should be conducted for coping with the early stages of the IEs. These six required tasks are: (1) opening the charging valve, (2) turning the PZR heater on, (3) checking the safety injection actuation signal (SIAS), (4) manually stopping the reactor coolant pump (RCP), (5) opening the turbine bypass valve (TBV), and (6) opening the atmospheric dump valve (ADV). The degree of these required tasks is set to $[0, 100]$ for binary control cases (e.g., Start/Stop or Open/Close), and $[0, 50, 100]$ or $[0, 5, 10, 25, 50, 75, 100]$ for continuous control cases (e.g., 50% opened valve). In addition, the timing of each required task is set to the mean, 5th, 95th, and 99th percentile values with an immediate response (i.e., timing is equal to 0), which were determined from an existing human performance database (Kim et al., 2019). Hence, the input vector $\mathbf{x}$ can be constructed based on the combination of degrees and timings described in Table 1. Note that the number of potential scenarios does not exactly match the number of full combinations based on the degrees and timings, because we did not consider inappropriate combinations resulting from unrealistic sequences of timings.

Table 1
Scenario configuration.

Index (k)	Degree (x_k)	Time* (x_k)	Description
1	1 (1A), 2 (2A),4 (4A)	—	Break size (A: area) SGTR: 1A, 2A, 4A, / MSLB: 1A, 2A
2, 3	0, 50, 100	0, 1.77, 3.1, 5.425, 9.3	PZR level below the set point; Charging valve open
4, 5	0, 50, 100	0, 2.06, 3.6, 6.3, 10.8	PZR pressure below the set point; PZR heater on
6, 7	0, 1	0, 2.06, 3.6, 6.3, 10.8	PZR pressure below the set point; SIAS actuate
8, 9	0 (No trip),1 (Trip)	0, 2.06, 3.6, 6.3, 10.8	PZR pressure below the set point; RCP 1A & 2A trip
10, 11	0 (No trip),1 (Trip)	0, 2.06, 3.6 6.3, 10.8	PCS margin below the set point; RCP 1A & 1B & 2A & 2B trip
12, 13	0, 5, 10, 25, 50, 75, 100	0, 2.629, 4.6, 8.05, 13.8	RCS temp above the set point; TBV open
14, 15	0, 5, 10, 25, 50, 75, 100	0, 2.629, 4.6, 8.05, 13.8	RCS temp below the set point; ADV open
16	0 (SGTR),1 (MSLB)	—	Class of initiating events

*Minute.
SGTR: Steam Generator Tube Rupture; MSLB: Main Steam Line Break; SIAS: Safety Injection Actuation Signal; RCP: Reactor Coolant Pump; TBV: Turbine Bypass Valve; ADV: Atmospheric Dump Valve.

With respect to each potential scenario, the trajectories of four key process variables that can represent the safety status of NPPs were collected up to 3600 s at 10 s intervals (i.e., each trajectory consists of 360 data points). These key process variables are: (1) PZR pressure (PZRP), (2) PZR level (PZRL), (3) reactor coolant system (RCS) subcooling margin (RCM), and (4) steam generator #1 pressure (SG1P) of which one or more tubes were ruptured.

After finishing the collection of trajectories D (the term D will represent the whole dataset), we classified datasets into training, validation, and test sets. We further divided test set into two subsets, namely Test_{in} and Test_{out} sets, to check the prediction performance of untrained scenarios having different prediction difficulty. First, Test_{out} is composed of trajectory data with $x_5 = 3.6$ (i.e., any trajectory data pertaining to the start of PRZ heater at 3.6 min when PRZP is lower than a set point). The rest of the trajectory data ($D - \text{Test}_{\text{out}}$) is randomly divided into training, validation, and Test_{in} according to the data split ratio of 7:1:2. By doing this, more neighboring scenarios of Test_{in} are used for training, thus Test_{out} becomes more difficult to predict. Finally, these Test_{in} and Test_{out} sets will be used to investigate the prediction accuracy of eQRNN (refer to Section 4.5) and each dataset is summarized in Table 2.

4.3. Description of model structure

In the experiment, we compared the proposed QRNN and its ensemble (eQRNN) to four different deep learning models. Table 3 summarizes the network structure of each model with the layer type and corresponding network parameters.

In Table 3, the values in parentheses describe different layer

Table 2
Dataset configuration.

Dataset (D)	Size	Description
Training	60,637 (58%)	Training set for weight update from $D - \text{Test}_{\text{out}}$
Validation	7,988 (8%)	Validation set for hyper-parameter selection from $(D - \text{Test}_{\text{out}})$
Test_{in}	18,000 (17%)	Randomly selected scenarios from $D - \text{Test}_{\text{out}}$
Test_{out}	18,000 (17%)	Specific case with $x_5 = 3.6$

Table 3
Model structure.

Model	Structure	Description
FNN	Input(16) - Dense(64) - Dense (128) - Dense(256) - Dropout (0.2) - Dense(360)	FNN in-situ model
RNN	Input(16) - Tile(16,360) - BiLSTM(256) - BiLSTM(256) - Dense(128) - Dense(64)s-Dropout(0.2) - Dense (1)	Bi-directional LSTM model
RNN_{aPE}	Input(16) - Tile(16,360) - aPE (16,360) - BiLSTM(256) - BiLSTM(256) - Dense(128)-Dense(64) - Dropout(0.2) - Dense (1)	Bi-directional LSTM model with additive PE, $d_{PE} = 16$, weighted by 0.5
RNN_{cPE}	Input(16) - Tile(16,360) - cPE (24,360) - BiLSTM(256) - BiLSTM(256) - Dense(128)-Dense(64) - Dropout(0.2) - Dense (1)	Bi-directional LSTM model with concatenative PE, $d_{PE} = 8$.
QRNN_{cPE}	Input(16) - Tile(16,360) - cPE (24,360) - BiLSTM(256) - BiLSTM(256) - Dense(128)-Dense(64) - Dropout(0.2) - Dense (5)	Bi-directional LSTM model with concatenative PE and quantile regression $\tau = (0.1, 0.3, 0.5, 0.7, 0.9)$.
$\text{eQRNN}_{\text{cPE}}$	Same as QRNN	Ensemble of five QRNN models

properties: i) the output shape for *Input* and *Tile*, ii) the number of neurons/cells for *Dense* and *Bi-LSTM*, and iii) the dropout rates for *Dropout*. The abbreviations *aPE* and *cPE* stand for additive and concatenative PE. All models share the same hyper-parameters, as listed in Table 4. MSE is used as the loss function except for with the QRNN, which utilizes pinball loss. Weights are trained with the Adam optimizer with a recommended initial learning rate of 0.001 (Kingma & Ba, 2014), and we set the epochs to 150. During the training, learning rate decay and early stopping are applied by monitoring the validation loss. All models are built using *tensorflow* (Abadi et al., 2016), and hyper-parameters are modified from the API documentation. In experiments, we build a model for each variable (i.e., univariate output), and an effective way to perform multivariate quantile regression with a single model remains for future work. Note that the FNN and RNN model is not the same model used in Fig. 6, and have increased model complexity.

To examine the advantages of the proposed model (utilizing bi-LSTM, concatenative PE, quantile regression, and model ensemble), we first fixed the base RNN structure and hyper-parameters by trial and error, and then compared the results by incrementally adding each modules. The base FNN and RNN structures (which we note provided a reasonable level of accuracy) were determined by trial and error, and thus their performance may increase through further optimization and tuning of the model structure and hyper-parameters (e.g., grid search, Bayesian optimization), which is beyond our scope of research.

Table 4
Hyper-parameters.

Hyper-parameter	Configuration
Loss	MSE, Pinball loss (for QRNN)
Optimizer	Adam
Learning rate	1e-3 (with reduced LR on plateau)
Batch size	512
Epochs	150 (with early stopping)

4.4. Results

In this section, we compare the results of four target process variables in terms of mean absolute percentage error (MAPE) and MSE. Subscripts *in* and *out* indicate the result of Test_{in} and Test_{out}, respectively. The MAPE and MSE of QRNN and eQRNN are calculated based on the prediction of the median in order to compare with the results of the point forecast models: FNN, RNN, RNN_{aPE}, and RNN_{cPE}. Note that we alter the prediction error for atmospheric pressure to zero for the result of SG1P, because the error for a small denominator (which is less important in terms of the overall system) dominates the error for a high-pressure region of interest.

Table 5 shows the mean and standard deviation of MAPE and MSE between the real values from the TH code run and the predictions from the diverse deep learning models. Except for eQRNN, we present the result having the smallest MSE among five trials per each model (FNN, RNNs, and QRNN). The bold and underlined values indicate the column-wise lowest and the second-lowest value, respectively. First, the FNN model shows the worst error rates among the six models due to its structural limitation (refer to Section 3.2). Compared to the FNN, the error rate of the baseline RNN model (bi-directional LSTM without PE) is significantly reduced. The average MAPE (average over the entire test set) is decreased by 62.9%, 71.4%, 61.8%, and 67.3% in the order of PZRP, PZRL, RCM, and SG1P, respectively; from now on, all comparison results are given in the same order. Although the base RNN exploits the same input for all time-steps, it could achieve improved accuracy thanks to the increased model complexity and structural advantage of LSTM.

Applying PE to the RNN provides further performance improvements. Both additive and concatenative PE show lower error rates compared to the base RNN without PE. The MAPE of RNN_{aPE} is decreased by 11.2%, 9.6%, and 7.7% on average for PZRP, RCM, and SG1P; for the case of RCM, the error increased by 0.1%. As we expected, the accuracy of the concatenative PE shows better performance than the additive PE. The MAPE of RNN_{cPE} is decreased by 27.8%, 11.4%, 20.0%, and 6.1% compared to the base RNN model. This result implies that utilizing temporal information explicitly via concatenation is helpful in the case of low-dimensional data.

Next, we compare the results of QRNN and eQRNN. These results show that the change to a quantile model yields further improvements in model accuracy. Moreover, both models provide the uncertain boundaries of the prediction result. As can be seen in Table 5, eQRNN shows the best performance in most cases. Compared to RNN_{cPE}, MAPE is reduced by 14.6%, 42.6%, 19.8%, and 8.8% for QRNN and by 20.7%, 42.6%, 26.3%, and 18.4% for eQRNN, respectively. Also, eQRNN and QRNN show lower standard deviations in most cases.

The improved performance of eQRNN is also recognized by the empirical cumulative distribution function (ECDF) of MAPE and MSE. Figs. 7 and 8 plot the ECDF of MAPE and MSE distributions over the entire test set. The ECDF in the upper left corner implies better accuracy because it has a lower mean and variance. For all cases, eQRNN outperforms the other models, followed by QRNN, RNN_{cPE}, RNN_{aPE}, RNN, and FNN in order.

Specifically, utilization of quantile regression leads to notable performance improvement, which can be observed in the form of gaps between the two graphs. For example, there exists a gap between the graph of RNN_{cPE} and eQRNN in Fig. 7b, c, d, and Fig. 8d. This performance improvement can be interpreted as follows. By changing to a quantile model and training with pinball loss, the QRNN model obtains the

Table 5
Prediction error comparison.

Model	$MAPE_{in}$		$MAPE_{out}$		MSE_{in}		MSE_{out}	
	mean	std	mean	std	mean	std	mean	std
Target	PZRP							
FNN	2.04	1.49	2.21	1.73	11.7	25.2	15	45
RNN	0.7	0.62	0.88	0.85	3.2	9.1	4.96	17.3
RNN_{aPE}	0.58	0.56	0.82	1.07	2.67	10.2	6.11	42.3
RNN_{cPE}	0.47	**0.46**	0.67	0.91	1.92	**7.39**	5.08	47.3
$QRNN_{cPE}$	0.42	0.48	0.55	0.93	2.05	8.35	3.65	33.6
$eQRNN_{cPE}$	**0.37**	0.47	**0.53**	**0.77**	**1.77**	7.53	**3.45**	**18.8**
Target	PZRL							
FNN	2.07	1.76	2.55	3.02	1.3	6.58	3.33	17.8
RNN	0.52	0.65	0.8	1.64	0.2	4.18	1.35	18
RNN_{aPE}	0.49	0.94	0.83	2.25	0.23	6.38	1.49	15.5
RNN_{cPE}	0.44	0.5	0.73	1.96	0.15	**2.78**	1.44	20.3
$QRNN_{cPE}$	0.27	0.53	**0.4**	**0.96**	0.15	3.57	**0.55**	10.1
$eQRNN_{cPE}$	**0.24**	**0.47**	0.43	1.29	**0.14**	3.6	0.65	**9.34**
Target	RCM							
FNN	11.9	47.79	12.4	12.4	9.7	32.6	14	76.3
RNN	4.3	39.2	4.99	6.4	2.64	**24**	13.1	173.4
RNN_{aPE}	3.82	36.2	4.58	6.19	2.77	29.3	12.1	157.1
RNN_{cPE}	3.29	**28**	4.14	5.76	2.37	26.2	7.12	105.9
$QRNN_{cPE}$	2.71	39.4	3.25	4.83	2.5	28.9	6.13	88.8
$eQRNN_{cPE}$	**2.44**	32.3	**3.04**	**4.58**	**2.09**	26.5	**6.05**	**67.2**
Target	SG1P							
FNN	0.78	1.71	0.78	0.75	1.33	3.06	1.45	3.53
RNN	0.25	**0.25**	0.26	**0.34**	**0.17**	**1.86**	0.28	2.57
RNN_{aPE}	0.22	0.29	0.24	0.36	0.17	2.31	0.29	2.62
RNN_{cPE}	0.23	0.3	0.25	0.35	0.18	2.3	**0.28**	**2.31**
$QRNN_{cPE}$	0.21	0.37	0.23	0.45	0.23	3.51	0.37	4.37
$eQRNN_{cPE}$	**0.18**	0.37	**0.21**	0.44	0.2	3.38	0.35	4.18

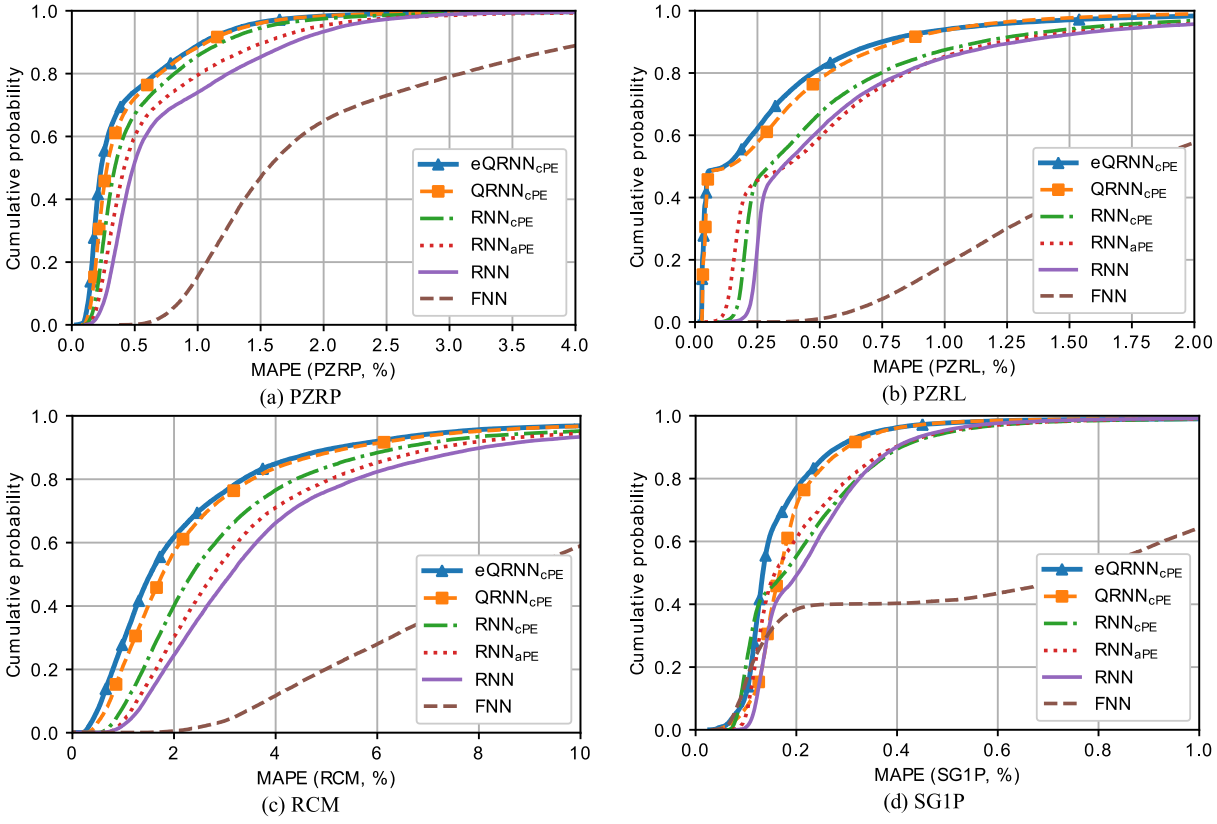


Fig. 7. Empirical CDF of error distribution (MAPE).

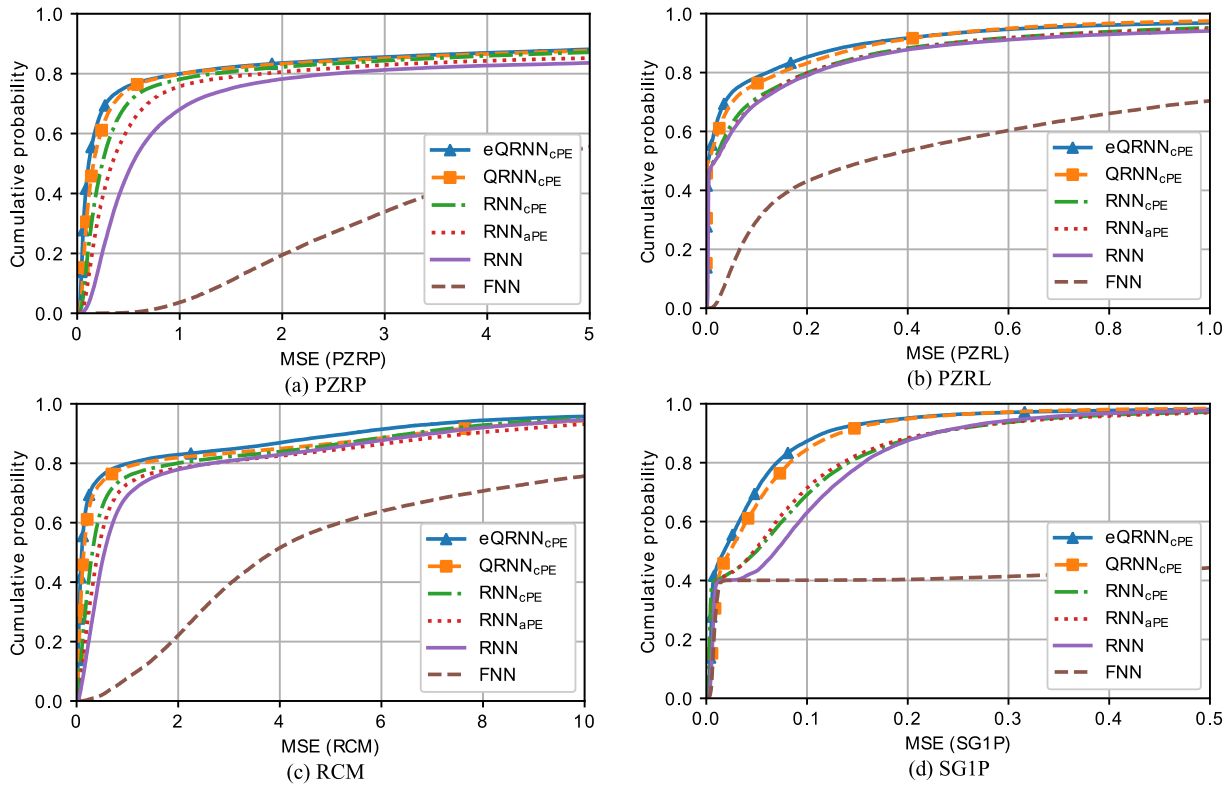


Fig. 8. Empirical CDF of error distribution (MSE).

advantage of multi-task learning (Rodrigues & Pereira, 2020). In multi-task learning, the model can learn key domain information that can be shared in performing multiple tasks, and thus generalization performance is improved. In this perspective, a quantile model performs multiple tasks, i.e., prediction of different quantiles. There exists a nonlinear relationship between two quantiles $\mathbf{Y}_{\tau_1}, \mathbf{Y}_{\tau_2}$, and they share key base information (input vector). Thus, training them together is a simplified multi-task learning problem, where the tasks are scored from a single label by pinball loss.

We further compared the performance of QRNN to other models (including cAE model (Kim et al., 2020), and deeper FNN) in terms of model parameter, inference time, and error metrics. Experimental results on PZRP data is given in Table 6. Model parameter is the number of trainable weights in neural network, and inference time is time spent on the prediction of all scenarios in Test$_{in}$ set with mini batch size of 64. As can be seen, RNN based models have higher model complexity, and requires longer inference time due to recurrent loop in LSTM network. But considering the time taken in the original TH code simulation (more than hours for 1 accident scenario), inference time of RNN in second order for 18,000 scenarios is negligible. Next, in order to support that the performance improvement in QRNN model is not simply due to increase in model parameters, we compared the results of deeper FNN (DFNN) which has similar number of model parameters as RNN by increasing the depth and width of base FNN. According to the results, DFNN shows much lower error rates compared to FNN, but the errors are still higher than that of RNN. Furthermore, the error of QRNN is reduced by almost half of DFNN, and these results imply that the multiple quantile regression and concatenative positional encoding are the main drivers of performance improvement without significant change in model complexity. We further performed experiments on recently developed cAE. cAE achieved higher accuracy than FNN, but lower than RNN due to structural limitation of FNN based approach. Finally, the proposed QRNN shows 73% and 77% lower MAPE and MSE than cAE (75% and 79% compared to eQRNN).

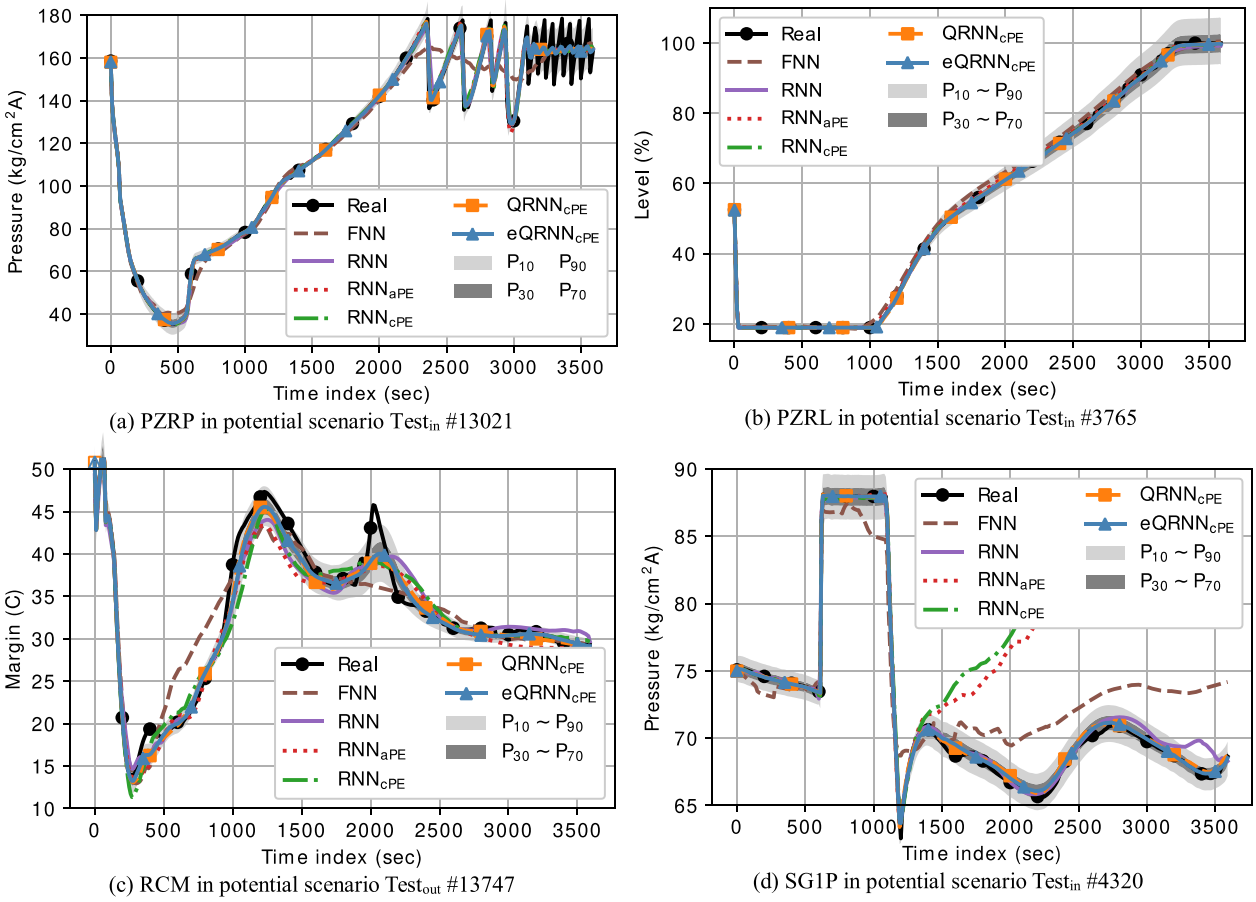
Table 6
Model Comparison on PZRP dataset.

Model	Model parameter	Inference time	MAPE$_{in}$	MAPE$_{out}$	MSE$_{in}$	MSE$_{out}$
FNN	135 k	0.2 s	2.04	2.21	11.7	15
cAE	220 k(418 k)*	0.2 s	1.69	1.96	9.49	15.19
DFNN	2,420 k	0.2 s	0.81	1.12	3.95	7.24
RNN	2,208 k	9.1 s	0.7	0.88	3.2	4.96
RNN$_{aPE}$	2,208 k	9.2 s	0.58	0.82	2.67	6.11
RNN$_{cPE}$	2,224 k	9.3 s	0.47	0.67	1.92	5.08
QRNN$_{cPE}$	2,225 k	9.4 s	0.42	0.55	2.05	3.65

*parameter of autoencoder.

4.5. Visualization of prediction

In order to perform a qualitative analysis on the results visually, the predicted trajectories of randomly selected scenarios are given in Fig. 9. Here, Fig. 9a is an example of a PZRP prediction. Note that the MAPE of eQRNN is 1.02%, which is a lot higher than the average MAPE of Test$_{in}$ (0.37%). The predictions of the RNN-based models follow the original trajectory as well. However, the oscillating pattern that occurs after 3,000 s is flattened. In these areas, QRNN and eQRNN have advantages since they provide quantile predictions that cover the oscillations. The second example, in Fig. 9b, is the result of PZRL, where all models show similar results. Fig. 9c is an example RCM prediction with scenario number 13,747 in Test$_{out}$. Unlike the previous examples (Fig. 9a & b), each model generates a different trajectory. Specifically, the trajectories are different between 1,000–3,000 s, where the shape of the graph changes dynamically. Compared to the other process parameters, the results of RCM show roughly 10 times higher MAPE; the MAPEs of QRNN and eQRNN for this example are 3% and 2.66%. However, the uncertain area given by eQRNN covers the original trajectory. The final example of SG1P is given in Fig. 9d. In this example, only RNN, QRNN, and eQRNN models have a similar shape to the original graph, with



(a) PZRP in potential scenario $Test_{in}$ #13021

(b) PZRL in potential scenario $Test_{in}$ #3765

(c) RCM in potential scenario $Test_{out}$ #13747

(d) SG1P in potential scenario $Test_{in}$ #4320

Fig. 9. Examples of prediction results.

MAPEs of these models of 0.62%, 0.34%, and 0.29%, respectively.

5. Discussions and conclusion

As briefly discussed in Section 2.1, the identification of accident scenarios is crucial for reducing the uncertainty of PSA results. Unfortunately, each accident scenario should be determined by using a precise TH code, for which the run time takes several hours on average. Accordingly, a breakthrough technology that allows us to predict the consequence of a potential scenario within a very short time can resolve this issue. For this reason, in this paper, we investigated a surrogate model of TH code and proposed a novel framework of an in-situ model based on deep learning. Notably, the eQRNN, the proposed model, provides the uncertainty boundaries of the prediction results. By leveraging bi-directional LSTM, concatenative PE, quantile regression, and model ensemble, the proposed eQRNN outperforms other deep learning models. To verify the suggested model, the results from four process parameters (i.e., PZR pressure, PZR level, RCS margins, and SG#1 pressure) were compared between models in terms of MAPE and MSE. Each increasingly complex model improves the prediction accuracy; specifically, the concatenative PE and quantile model shows notable error reduction. Compared to the base RNN, the proposed eQRNN model shows 43%, 49%, 41%, and 23% lower MAPE for PZRP, PZRL, RCM, and SG1P, respectively: overall, 39% lower MAPE and 28% lower MSE. This strongly implies that a fast running in-situ model can be secured with an affordable range of prediction error.

Currently, a model that generates only a univariate output has been developed. For the actual application of such a model, it is necessary to make several models according to the variables that affect the scenario (i.e., to monitor the variables that make diverse scenarios). This would affect calculation time and volume, as the number of models increases according to the number of variables to be monitored. Therefore, a multivariate model is needed to solve this problem.

In the case of the in-situ model, the output of the model is made for full-length simulation. Although there is an advantage in that quick branching is possible in consideration of the status of components with the response of human operators at all times, the model always calculates the full-length simulation time when either a new status of the components or a new response of the human operator is given. Therefore, it is necessary to slice and merge the data in order to make a new potential scenario. For example, when using a model based on eQRNN where branching should be performed considering quantiles, the number of inferences will increase the same amount as the number of quantiles to be considered for each branching point.

In order to overcome this limitation, a stepwise model was considered. Even though the stepwise model is sensitive to error and harder to train than the in-situ model, it calculates the output using previous information. With this virtue, it is able to reflect actions in real-time more easily than the in-situ model can. Moreover, the stepwise model can be developed for on-site trajectory generation, which can also be applied to the development of reinforcement-learning-based autonomous operation models. Therefore, as a parallel work, a feasibility study of the stepwise model is ongoing.

Nevertheless, it is evident that the in-situ model proposed in this study plays a significant role in reducing the uncertainty of PSA results by discovering unknown accident scenarios. In order to clarify this claim, let us consider a series of *Event Headings* exemplified below, which consist of specific conditions and the required action to be taken by human operators when these conditions are met.

- Event Heading 1: Adjust *Valve A* when the pressure of *Tank B* is lower than 50 kgf/cm^2.
- Event Heading 2: Operate *Heater C* unless the pressure of *Tank D* is greater than 75 kgf/cm^2.
- Event Heading 3: Start *Pump E* when the water level of *Tank B* is increasing.
- Event Heading *n*: […].

In traditional PSA, due to the limitation of available resources in TH code simulations, the variability of potential scenarios is reduced by introducing a series of conservative assumptions. It should be emphasized that these conservative assumptions are themselves generated based on the results of intensive TH code runs for a large number of potential scenarios. Therefore, although it seems that the sequence of *Event Headings* is simple (i.e., static sequence), it is strongly expected that most accident scenarios can be covered by traditional PSA because numerous potential scenarios of which the consequence is trivial are already excluded. However, it is more realistic to assume that the sequence of *Event Headings* is variable depending on the characteristics of the responses previously done by human operators (i.e., dynamic sequence). That is, in accordance with the timing and/or the degree of opening of *Valve A* in the above *Event Heading* 1, it is natural to assume that the next *Event Heading* could either be *Event Heading* 2 or *Event Heading* 3. In other words, the result of the human response included in *Event Heading 1* can affect which *Event Heading* should appear next.

If this dynamic sequence is indeed the more realistic assumption, then the number of simulations using a precise TH code will drastically increase because of the existence of hidden potential scenarios (Heo, Baek, Kwon, Kim, & Park, 2021)(). In this case, the use of the in-situ model proposed in this study should be considered for its contributions in revealing the inventory of additional accident scenarios from these additional potential scenarios, which have not been considered in traditional PSA before. This study is a good starting point to accomplish this goal.

CRediT authorship contribution statement

Seunghyoung Ryu: Investigation, Methodology, Software, Writing – original draft. **Hyeonmin Kim:** Conceptualization, Data curation, Writing – original draft. **Seung Geun Kim:** Validation, Writing – review & editing. **Kyungho Jin:** Validation, Writing – review & editing. **Jaehyun Cho:** Funding acquisition, Project administration. **Jinkyun Park:** Conceptualization, Funding acquisition, Supervision, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by the Nuclear Research & Development Program grant from the National Research Foundation of Korea (NRF), funded by the Ministry of Science and ICT (NRF 2019M2C9A1055906 and NRF 2020M2C9A1061638).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., & &. (2016). *Tensorflow: A system for large-scale machine learning, 16* pp. 265–283). *OSDI*.

Abualigah, L., & Diabat, A. (2021). Advances in sine cosine algorithm: A comprehensive survey. *Artificial Intelligence Review*, 1–42.

Abualigah, L., Diabat, A., Mirjalili, S., Abd Elaziz, M., & Gandomi, A. H. (2021). The arithmetic optimization algorithm. *Computer Methods in Applied Mechanics and Engineering, 376*, Article 113609.

Abualigah, L., Yousri, D., Abd Elaziz, M., Ewees, A. A., Al-qaness, M. A. A., & Gandomi, A. H. (2021). Aquila optimizer: A novel meta-heuristic optimization Algorithm. *Computers & Industrial Engineering, 157*, Article 107250.

Aldemir, T. (2013). A survey of dynamic methodologies for probabilistic safety assessment of nuclear power plants. *Annals of Nuclear Energy, 52*, 113–124.

Biau, G., & Patra, B. (2011). Sequential quantile prediction of time series. *IEEE Transactions on Information Theory, 57*(3), 1664–1674.

Brock, A., De, S., Smith, S. L., & Simonyan, K. (2021). High-performance large-scale image recognition without normalization. *International Conference on Machine Learning (PMLR)*, 1059–1071.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems, 33*, 1877–1901.

Caliva, F., de Ribeiro, F. S., Mylonakis, A., Demazi'ere, C., Vinai, P., Leontidis, G., & Kollias, S. (2018). A deep learning approach to anomaly detection in nuclear reactors. *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Chen, Q., Wang, W., Huang, K., De, S., & Coenen, F. (2021). Multi-modal generative adversarial networks for traffic event detection in smart cities. *Expert Systems with Applications, 177*, Article 114939.

Chen, Q., Wang, W., Wu, F., De, S., Wang, R., Zhang, B., & Huang, X. (2019). A survey on an emerging area: Deep learning for smart city data. *IEEE Transactions on Emerging Topics in Computational Intelligence, 3*(5), 392–410.

Chen, S., & Demachi, K. (2019). Proposal of an insider sabotage detection method for nuclear security using deep learning. *Journal of Nuclear Science and Technology, 56*(7), 599–607.

Choi, S., Kim, J. T., & Choo, J. (2020). Cars can't fly up in the sky: Improving urban-scene segmentation via height-driven attention networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 9373–9383).

da Silva, D. B., Schmidt, D., da Costa, C. A., da Rosa Righi, R., & Eskofier, B. (2021). DeepSigns: A predictive model based on Deep Learning for the early detection of patient health deterioration. *Expert Systems with Applications, 165*, Article 113905.

dos Santos, M. C., Pinheiro, V. H. C., do Desterro, F. S. M., de Avellar, R. K., et al. (2019). Deep rectifier neural network applied to the accident identification problem in a PWR nuclear power plant. *Annals of Nuclear Energy, 133*, 400–408.

Faust, O., Hagiwara, Y., Hong, T. J., Lih, O. S., & Acharya, U. R. (2018). Deep learning for healthcare applications based on physiological signals: A review. *Computer Methods and Programs in Biomedicine, 161*, 1–13.

Heo, G., Baek, S., Kwon, D., Kim, H., & Park, J. (2021). Recent reserch towards integrated deterministic-probabilistic safety assessment in Korea. *Nuclear Engineering and Technology, 53*(11), 3465–3473.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.

Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *ArXiv Preprint*. ArXiv:1506.02078.

Ke, G., He, D., & Liu, T.-Y. (2020). Rethinking positional encoding in language pre-training. *International Conference on Learning Representations*.

Kim, Y., Kim, J., Park, J., Choi, S. Y., Kim, S., Jung, W., … & Shin, S. K. (2019). *An HRA Method for Digital Main Control Rooms–Part I: Estimating the Failure Probability of Timely Performance*. KAERI/TR-7607/2019.

Kim, H., Cho, J., & Park, J. (2020). Application of a deep learning technique to the development of a fast accident scenario identifier. *IEEE Access, 8*, 177363–177373.

Kim, S. G., Chae, Y. H., & Seong, P. H. (2020). Development of a generative-adversarial-network-based signal reconstruction method for nuclear power plants. *Annals of Nuclear Energy, 142*, Article 107410.

Koenker, R., & Hallock, K. F. (2001). Quantile regression. *Journal of Economic Perspectives, 15*(4), 143–156.

Lai, C. S., Zhong, C., Pan, K., Ng, W. W. Y., & Lai, L. L. (2021). A deep learning based hybrid method for hourly solar radiation forecasting. *Expert Systems with Applications, 177*, Article 114941.

Lin, T.-H., Wang, T.-C., & Wu, S.-C. (2021). Deep learning schemes for event identification and signal reconstruction in nuclear power plants with sensor faults. *Annals of Nuclear Energy, 154*, Article 108113.

Park, Jinkyun, & Yoon, Jae Young (2018). Toward the use of deep learning techniques to enhance PSA quality: A digital twin. Proceedings of the Korea Nuclear Society 2018 Fall Meeting.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*.

Mandelli, D., Alfonsi, A., Talbot, P., Wang, C., Maljovec, D., Smith, C., Rabiti, C., & Cogliati, J. (2016). *An overview of reduced order modeling techniques for safety applications*.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. *European Conference on Computer Vision*, 405–421.

Olah, C. (2015). Understanding lstm networks. *URL Http://Colah. Github. Io/Posts/2015-08-Understanding-LSTMs*.

Park, J. K. (2018). *Feasibility study on the digital twin of a PSA scenario development based on a deep-learning technique*. Korea Atomic Energy Research Institute, Article KAERI/RR-4379/2018.

Radaideh, M. I., Pigg, C., Kozlowski, T., Deng, Y., & Qu, A. (2020). Neural-based time series forecasting of loss of coolant accidents in nuclear power plants. *Expert Systems with Applications, 160*, Article 113699.

Rodrigues, F., & Pereira, F. C. (2020). Beyond expectation: Deep joint mean and quantile regression for spatiotemporal problems. *IEEE Transactions on Neural Networks and Learning Systems, 31*(12), 5377–5389.

Ryu, S., Choi, H., Lee, H., & Kim, H. (2019). Convolutional autoencoder based feature extraction and clustering for customer load analysis. *IEEE Transactions on Power Systems, 35*(2), 1048–1060.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing, 45*(11), 2673–2681.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., , … Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature, 550*(7676), 354–359.

Steinwart, I., & Christmann, A. (2011). Estimating conditional quantiles with the help of the pinball loss. *Bernoulli, 17*(1), 211–225.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008.

Wang, J., Ma, Y., Zhang, L., Gao, R. X., & Wu, D. (2018). Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems, 48*, 144–156.