Towards Online Continuous Reinforcement Learning on Industrial Internet of Things

Cheng Qian*, Wei Yu*, Xing Liu*, David Griffith†, and Nada Golmie†
*Towson University, USA
Emails: {cqian1,xliu10}@students.towson.edu, wyu@towson.edu
†National Institute of Standards and Technology (NIST), USA
Emails:{david.griffith, nada.golmie}@nist.gov

***Abstract*—In Industrial Internet of Things (IIoT), the information and communication technologies powered by IoT can greatly improve the efficiency and timeliness of information exchange between different industrial components. Likewise, machine learning techniques such as reinforcement learning can benefit from the massive amounts of data collected via IoT to successfully automate industrial monitoring and control processes. Nonetheless, training machine learning models, such as reinforcement learning models, require a significant investment of time, and a trained model can only work on a specific system in a specific environment. When the application scenario of reinforcement learning changes, or the application environment changes, the reinforcement learning model needs to be retrained. Thus, it is critical to design techniques that can reduce the overhead of retraining reinforcement learning models, enabling them adapt to constantly changing environments. In this paper, toward improving the performance of learning models in dynamic IIoT, we propose an online continuous reinforcement learning strategy. In our process, when the retraining condition is triggered, our online continuous learning strategy will re-engage the training process and update the well-trained model. To evaluate the performance of our proposed approach, we categorize the entire application space for applying reinforcement learning to IIoT systems into four scenarios, namely, non-continuous learning without learning model sharing, non-continuous learning with learning model sharing, continuous learning without learning model sharing, and continuous learning without learning model sharing. For each scenario, we design a Q-learning based reinforcement learning algorithm. Via extensive evaluation, our results show that the online continuous reinforcement learning approach that we propose can significantly reduce the overhead of retraining the learning model, enabling the learning algorithm to quickly adapt to a changing environment.**

***Keywords*-Industrial Internet of Things, Online continuous learning, Reinforcement learning**

I. Introduction

The Industrial Internet of Things (IIoT), also known as Industry 4.0, is considered to be one of the most important technological advancements in the industrial field [1]. The IIoT integrates IoT technologies into industrial manufacturing and production processes to improve industrial monitoring, control, and automation. As a typical cyber-physical system (CPS), the IIoT system consists of cyber subsystem and physical subsystem. The cyber subsystem consists of control, networking, and computing components, which cooperate to collect, transfer, and analyze the data collected from the system to realize the control loops of manufacturing and production. The physical subsystem consists of the control objects in physical factories and plants, such as the continuous stirred tank reactor for chemical reaction management [2], [3]. To achieve high connectivity, reliability, efficiency, and intelligence in factories and manufacturing plants, advanced data analysis techniques such as machine learning are used to process the massive amounts of data collected by sensors.

Machine learning techniques have been widely adopted to assist in data analysis for a variety of applications [4]. These algorithms can learn and improve from experience without explicit programming. Generally speaking, machine learning techniques can be divided into three types: supervised machine learning, unsupervised machine learning, and reinforcement learning. Supervised machine learning leverages labeled data for training, and uses the trained model to label unlabeled data. Unsupervised machine learning can be used to cluster and correlate unlabeled data, discovering hidden patterns associated with the data. Reinforcement machine learning can make decisions based on interactions with the environment over time by choosing actions based on the system's state and the rewards that are associated with the actions that are available to the system. In this study, we use reinforcement learning as an example to demonstrate its ability for controlling a typical industrial manufacturing process. Our design can be generally applied to other machine learning techniques and IIoT scenarios.

Applying machine learning to the IIoT system has enabled a variety of benefits and improvements in industrial manufacturing processes, such as enabling IIoT systems to adapt and learn from new environments, detecting and correcting system errors based on historical data, and predicting future events based on pre-trained models. Nonetheless, applying machine learning to IIoT environments also raises some challenges. The IIoT system is dynamic. It is difficult to keep the model up-to-date based on the huge volume of dynamic data collected from the system. To achieve frequent model updates, traditional machine learning needs more computation power and longer training times. Thus, how to reduce the training overhead without affecting the accuracy of machine learning remains an unsolved issue.

To address this problem, in this paper we design an online continuous machine learning technique to make the reinforcement learning algorithm adapt to the dynamically changing IIoT environment. By sharing some learned models from

similar or identical systems, the convergence time of the new system can be largely reduced. Online continuous learning is a specific machine learning strategy that is capable of handling newly arriving sequential data. It is intended for learning process based on the latest arrived data and updating the model rapidly so that the learned model can be adapted to dynamic IIoT systems. Compared to offline continuous learning, online continuous learning is efficient in both time and space costs. Meanwhile, online continuous learning is also suitable for real-time monitoring and control in IIoT systems.

To summarize, we make the following contributions:

- **Framework:** We propose a general framework to implement online continuous reinforcement learning in a typical IIoT environment [2]. In detail, we categorize the entire problem space into four scenarios, namely: scenario A (non-continuous learning without learning model sharing), scenario B (non-continuous learning with learning model sharing), scenario C (continuous learning without learning model sharing), and scenario D (continuous learning without learning model sharing).
- **Online Reinforcement Learning:** We use scenario A as the traditional offline discontinuous benchmark scenario, and design an online continuous learning approach for scenarios B, C, and D, respectively. For scenario A, we design the benchmark Q-learning algorithm to make it suitable for our IIoT environment. For scenario B, we leverage the learning model (i.e., Q-table) shared from another physical system to reduce the convergence time. For scenario C, we design a re-trigger mechanism to reengage the learning process so that the system can adapt to the dynamic IIoT environment. For scenario D, we use both the re-trigger strategy and learning model sharing to make the system converge faster in a dynamic IIoT environment.
- **Extensive Validation:** We conduct extensive performance evaluations to validate the effectiveness of our online continuous learning approach on a representative IIoT system. Our experimental results confirm the effectiveness of the online continuous reinforcement learning algorithms between the same, similar, and different IIoT systems.

The remainder of this paper is organized as follows: In Section II, we briefly review the three types of machine learning techniques, online continuous learning, and an IIoT simulation environment. In Section III, we provide literature reviews relevant to our study. In Section IV, we present our approach in detail. In Section V, we present our experimental results to validate the efficacy of our approach. Finally, we conclude the paper in Section VI.

II. Preliminary

In this section, we first provide an overview of the categories of machine learning. Then, we present online continuous learning. Finally, we introduce an IIoT environment that we use to conduct experiments.

A. Machine Learning Techniques

Generally speaking, machine learning techniques can be categorized into three types: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning tends to build a function that maps inputs to outputs based on labeled input/output training pairs. It can be used for a variety of purposes, such as classification, regression, and others. Unsupervised learning is a technique that is used to group similar components and identifying patterns inside the data. It is used to extract the structure from the datasets, which are not classified or not labeled. Unsupervised learning can be used to address problems such as clustering and association.

Reinforcement learning is a machine learning technique based on the Markov decision process (MDP). Reinforcement learning can be a model-less machine learning method, which can be considered as the computation approach of automation, and it is a goal-directed learning and decision process. The difference between reinforcement learning and the other machine learning techniques is that it can interact with the environment without the need to conduct the labeling or to build a model like the traditional supervised machine learning [5]. Reinforcement learning defines a framework, which can enable interaction between the learning agent and its environment. The system state, action, and reward are key components in reinforcement learning. The system state represents the observation of the environment. Based on the current state, the agent may take an action based on learned interactions with the environment. Based on the action, a reward will be given by the environment to the agent to determine whether the action it has taken was good or bad.

B. Online Continuous Machine Learning

For model training and updates, a given machine learning scheme can be operated either offline or online. Taking offline supervised learning as an example, all datasets should be available to the model during the training process. The model is not usable unless the training process is completed. However, in online continuous learning, the full data is not available to the model initially. The algorithm will process the data in sequential order. The model will continue updating while the data is continuously arriving, with snapshots of trained models being applied for use. One benefit of online continuous learning is that it can maximize accuracy and keep the predicted model constantly up to date. As the data is always updated and the data size can be small, model retraining can be highly efficient. Thus, online continuous machine learning is useful for handling the learning process in large-scale machine learning tasks on dynamic systems such as IIoT systems.

C. Wireless Cyber-Physical Simulator (WCPS)

We use WCPS as our IIoT simulation environment to evaluate the efficacy of our online reinforcement learning ap-

proach for IIoT systems[1]. WCPS is an open-source simulation environment for wireless control systems [6], which consists of Simulink and TinyOS Simulator (TOSSIM) wireless sensor simulator. The WCPS can simulate the dynamic changes of the physical system and the network, as well as their interactions.

Fig. 1 illustrates the architecture of the WCPS system [6]. From the figure, we can see that the plant model generates sensor data and uses a Simulink cross-platform function to call TOSSIM [7]. The sensing data are then passed into the sensors in TOSSIM through the Simulink function call. After that, the TOSSIM simulates the network environment based on the network conditions (packet loss rate, etc.). The results of the simulation (e.g., whether a packet is lost or not) will be transmitted back to Simulink through the Python interface. The packet collector extracts the packet loss information from the message pool. The packet delivery information, such as packet loss and latency, and the sensing data will be sent to the data block first and then ultimately transmitted to the controller. As we can see in Fig. 1, the reference signal is regarded as a control signal input by the user to the controller. The network scheduling table is calculated by the scheduling module in the network manager and deployed to the time-division multiple access (TDMA) Medium Access Control (MAC) layer. The received signal strength indicator (RSSI) and noise are provided to TOSSIM to assist in the simulation of realistic networks.

As shown in Fig. 2, the Interfacing Block extracts delay and loss information from TOSSIM messages, and the Data Block determines what data will be used for discrete control during each sampling period. It is worth noting that the architecture of WCPS provides the flexibility to incorporate different structural models and implement alternative scheduling-control approaches.

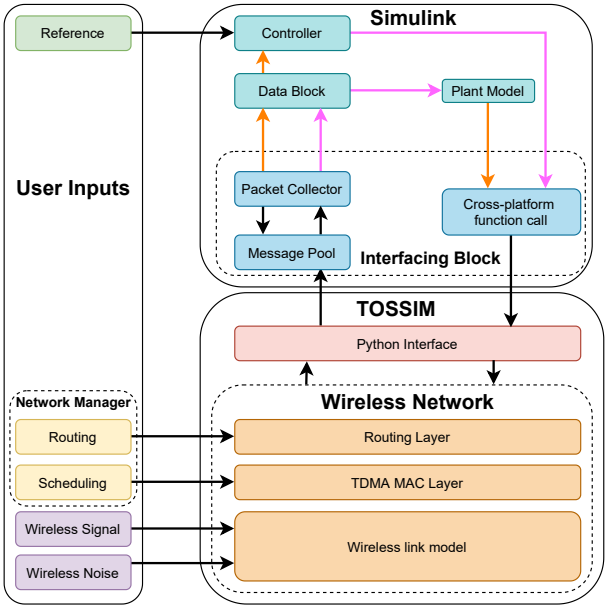

Fig. 1. Architecture of WCPS [6]

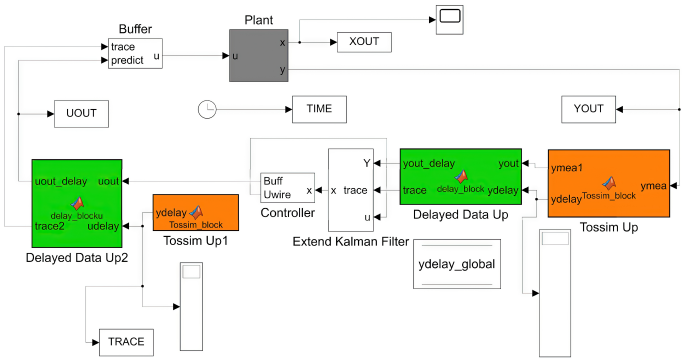

Fig. 2. Simulink Module

III. Related Work

In this section, we review some relevant research efforts related to applying online continuous learning to IIoT systems. First, we review some existing works on IIoT. Then, we introduce some existing machine learning efforts applied to IIoT. Finally, we discuss some works which focused on online continuous learning techniques and their applications in IIoT and other CPS.

Generally speaking, IIoT is a way of leveraging IoT techniques in industrial manufacturing operations. Specifically, applying IoT into industrial systems can improve the productivity, efficiency, safety, and intelligence of IIoT system operations [1], [8], [9]. Most existing research efforts related to IIoT have focused on the application of IIoT, the design of IIoT architectures, and the optimization of IIoT performance. For the application of IIoT, a number of research efforts have leveraged the data generated by sensors to assist in the operation of the industrial manufacturing processes. For example, Zhang *et al.* [10] conducted experiments to validate that the implementation of IIoT could improve the performance of wind farms. Likewise, Zhou *et al.* [11] studied an IIoT system to monitor an underground coal mine. Sklyar and Kharchenko [12] developed a framework to use assurance case methodology for IIoT.

Regarding the design of IIoT architectures, there have been several recent efforts. For example, Xiong *et al.* [13] defined a security framework for the IIoT environment with a self-designed information upload strategy and encryption method. Likewise, Hatzivasilis *et al.* [14] designed a novel IIoT protocol to improve the efficiency and availability of the whole system. Wang *et al.* [15] designed an IIoT framework to save on energy consumption. For the optimization of IIoT system, Dou *et al.* [16] leveraged edge computing to optimize video transmission in IIoT. Liu *et al.* [17] designed an ITCFN based on Simulated Annealing Particle Swarm Optimization with Load Balancing (SAPSO-LB) algorithm to reduce the data processing latency in the IIoT system. Likewise, Li *et al.* [18] leveraged edge computing in an Software Defined Network (SDN)-Based IIoT system to improve data transmission.

Growing interest has also been demonstrated in applying

[1]Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

machine learning in IIoT systems. Most existing research efforts have focused on using machine learning as an advanced data analysis tool to assist in the operation of IIoT systems. For instance, some efforts have been made to use machine learning to perform data pre-processing for advanced industrial operations. Specifically, Lele [19] leveraged unsupervised machine learning for obtaining principle components of data generated by an oil well. Kanawaday and Sane [20] used supervised machine learning to predict the necessary maintenance of the slitting machine. Shah and Tiwari [21] leveraged machine learning to detect anomalous behavior of machines in IIoT environments. Yang *et al.* [22] designed a machine learning framework by using mobile edge computing to distribute machine learning tasks in IIoT. Arachchige *et al.* [23] leveraged a PriModChain framework to enforce privacy and trustworthiness on IIoT data by combining differential privacy, federated machine learning, Ethereum blockchain, and smart contracts.

In addition to the offline learning in IIoT, online continuous learning techniques have been used to assist IIoT systems and other CPS. Most of efforts have leveraged distributed machine learning and online continuous learning techniques to improve system performance. For example, Liu *et al.* [24] leveraged edge computing and deep reinforcement learning to improve the performance of the investigated system. Chen *et al.* [25] investigated online learning assisted edge computing to reduce training time.

Compared with these prior works, in this paper, we do not focus on using edge computing to assist online continuous learning to improve the performance of the system. Instead, we focus on designing the algorithm to re-engage the learning process of reinforcement learning when the environment changes to handle dynamic events. Further, we investigate how to share the reinforcement learning models in our online continuous learning approach so that the learning convergence speed can be improved.

IV. System Approach

In this section, we present our approach in detail.

A. Overview

In this paper, we propose our online continuous learning approach to enable reinforcement learning to adapt to the dynamic environments of IIoT systems. Generally speaking, IoT systems are highly dynamic, constantly changing over time. Nonetheless, a well-trained reinforcement learning model can only be applied to a specific environment. Once the environment changes, we generally need to retrain the reinforcement learning model from scratch. The training process of reinforcement learning not only consumes a significant amount of time, but also requires high computing capabilities. Therefore, in order to enable the reinforcement learning model to adapt to the constantly changing IIoT environment, we propose an online continuous learning approach to reduce the cost of retraining reinforcement learning in IIoT. We take the Q-learning algorithm as an example to illustrate how our proposed approach continuously updates the learning model and causes the learning model converge quickly.

To illustrate how to apply online continuous learning in IIoT, we first design a framework that categorizes the IIoT problem space into four scenarios. Our framework considers two orthogonal dimensions. The first is whether the learning model will be shared or not. The second is whether the learning model will be updated or not. Based on these two dimensions, we present the problem space in Fig. 3, which consists of four scenarios:

- *Scenario A: Non-Continuous learning without Learning Model Sharing.* In this scenario, we consider that the learning model (e.g., Q-table) will not be shared, and no continuous learning will be implemented.
- *Scenario B: Non-Continuous Learning with Learning Model Sharing.* In this scenario, we consider that the learning model (e.g., Q-table) will be shared between systems, but no continuous learning will be implemented.
- *Scenario C: Continuous Learning without Learning Model Sharing.* In this scenario, we consider that the learning model (e.g., Q-table) will not be shared between systems, but continuous learning will be implemented.
- *Scenario D: Continuous Learning with Learning Model Sharing.* In this scenario, we consider that the learning model (e.g., Q-table) will be shared between systems, and continuous learning will be implemented.

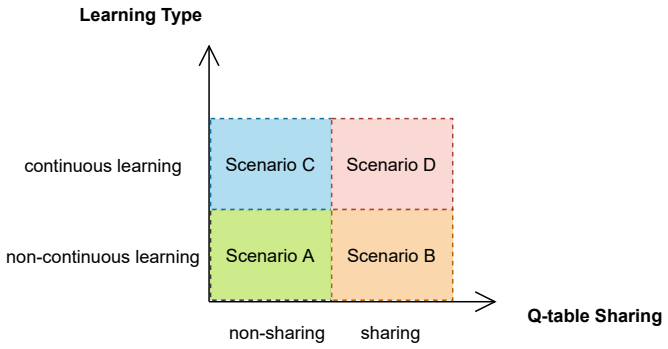

Fig. 3. Problem Space

In the following, we will describe the four scenarios and our proposed online continuous learning algorithms for each scenario in detail.

B. Scenario A: Non-Continuous learning without Learning Model Sharing

Before introducing the algorithm designed for this scenario, we briefly introduce the basic concept of Q-learning [26]. In general, Q-learning is an off-policy reinforcement learning technique that is designed to select the action to take based on the current system state and the reward function. The reason why it is called "off-policy" reinforcement learning is that it can take actions outside the current policy. For example, the agent can take random actions to receive rewards.

In the Q-learning algorithm, there are two important concepts: exploration rate and exploitation rate. Exploration is a

strategy based on the assumption that the agent has chosen a non-optimal action under the current state and gained more knowledge about the environment. This knowledge makes it possible to ignore the local optimal policy and instead reach the global optimal policy. Exploitation is a strategy that selects the best action based only upon the current Q-value of the state-action pair. During the training phase, the exploration rate decreases as the number of training steps increases, which means that the model is more likely to not update the Q-value in its Q-table after a certain number of Q-table updates.

The purpose of the baseline Q-learning algorithm is to demonstrate that the reinforcement learning algorithm is effective under IIoT environments. The detailed procedure is described in Algorithm 1. In this algorithm, we first initialize the value of all entries in the Q-table to 0. The variable α represents the learning rate in the Q-learning algorithm. When performing the initialization, we set its value to 0.5, which represents a balance between exploration and exploitation rates. The exploration rate defines the possibility that the model will update the Q-value, while the exploiting rate defines the possibility that the model will use the existing Q-value. After the initialization, we set the variable k, which will increase during the training period. When k increases, the learning rate α decreases. If α approaches 0, the Q-table will no longer be updated. Also, *system_working_time* determines the total time that the system remains operational. The variable *diff* represents the disparity between the current controlled variable (e.g., temperature) and control objective/target (e.g., temperature) in the IIoT system. The variable *diffmean* represents the mean disparity between the current controlled variable (e.g., temperature) and control objective/target (e.g., temperature) in the IIoT system within one iteration. The variable *sum* represents the summation of the disparity between the current controlled variable (e.g., temperature) and control objective/target (e.g., temperature). The variable *windowsize* represents how many controlled variables can be received during a single iteration of the training process. Based on the disparity between the current controlled variable and control objective/target, we manually divide the range of control objectives into several states. The last variable to initialize is the control objective/target, which the IIoT system needs to reach. When the training completes, the algorithm will save the Q-table, which can be shared with other IIoT systems. The total time complexity of this algorithm is $O(n)$, where n is the number of training epochs.

Algorithm 1: Algorithm for Scenario A

Result: Q(s,a)
Initialization: Q(s,a) $\leftarrow$ 0, α=$\frac{1}{2(k+1)}$, k=0, diff=0, states, target, sum=0, diffmean=0, windowsize, i=0, system_working_time
while *iteration* $<$ *system_working_time* **do**
  k++
  iteration++
  while $i <$ *windowsize* **do**
    diff = abs(controlled_variable$_i$ - target)
    sum += diff
    i++
  end
  diffmean = sum / windowsize
  sum = 0
  i = 0
  update: Q(s,a) $\leftarrow$ (1-α)*Q(s,a)+α*(1-diffmean)
  Save updated Q-table
end

Algorithm 2: Algorithm for Scenario B

Result: Q(s,a)
Initialization: Q(s,a) $\leftarrow$ another system, α=$\frac{1}{2(k+1)}$, k=0, diff=0, states, target, sum=0, diffmean=0, windowsize, i=0, system_working_time
while *iteration* $<$ *system_working_time* **do**
  k++
  iteration++
  while $i <$ *windowsize* **do**
    diff = abs(controlled_variable$_i$ - target)
    sum += diff
    i++
  end
  diffmean = sum / windowsize
  sum = 0
  i = 0
  update: Q(s,a) $\leftarrow$ (1-α)*Q(s,a)+α*(1-diffmean)
  Save updated Q-table
end

C. Scenario B: Non-Continuous Learning with Learning Model Sharing

In this case, we use a trained model from the same type of system as the starting point in order to improve the convergence speed of learning process. The basic policy for sharing the learning model (e.g., Q-table) is that only the learning model shared from the same or similar system can improve the learning convergence of the new system. It is worth noting that if the learning model (e.g., Q-table) is from a different physical system, convergence of the system will actually be degraded. The detailed results can be found in Section V.

The procedure for scenario B is detailed in Algorithm 2. In Algorithm 2, as the preliminary version of Q-table is predetermined from another IIoT system, it will be loaded in the beginning. Similar to Algorithm 1, the overall complexity is $O(n)$, where n is the number of training epochs.

D. Scenario C: Continuous Learning without Learning Model Sharing

In scenario C, we consider that the learning model can be dynamically updated. Consider that the environment (e.g., control objective, network conditions) is changed, and we need to re-engage the learning progress by resetting the exploration rate. The online Q-learning is targeted to re-trigger the learning process after the training environment has significantly changed [27].

The detailed procedure of the learning process for scenario C is described in Algorithm 3. In this algorithm, we continuously compute the difference between the controlled variable and objective/target. The variable *flag* represents whether the system has reached the convergence state or not. The threshold d represents the re-trigger point determined by the difference between *controlled_variable* and *target*. The threshold α represents the checkpoint for whether the system is at convergence or not. If the IIoT environment changes, there

will be a significant change between the controlled variable and objective/target (i.e., temperature). When this occurs and the system has already reached the convergence state, we will re-trigger the learning process by setting k to 0. Meanwhile, the learning rate of α will return to 0.5. The overall complexity is of this algorithm is $O(n)$, where n is the number of training epochs.

Algorithm 3: Algorithm for Scenario C

```
Result: Q(s,a)
Initialization: Q(s,a) ← 0, α=1/(2(k+1)), k=0, diff=0, states, target,
  sum=0, diffmean=0, windowsize, i=0, flag=0, system_working_time
while iteration < system_working_time do
    k++
    iteration++
    while i < windowsize do
        diff = abs(controlled_variable_i - target)
        sum += diff
        i++
    end
    diffmean = sum / windowsize
    sum = 0
    i = 0
    if diffmean <= threshold_d && α <= threshold_α then
        flag = 1
    end
    if diffmean > threshold_d && flag == 1 then
        k = 0
        flag = 0
    end
    update: Q(s,a) ← (1-α)*Q(s,a)+α*(1-diffmean)
    Save updated Q-table
end
```

E. Scenario D: Continuous Learning with Learning Model Sharing

In scenario D, we consider the learning model will be shared between systems and also be updated when the system state changes. To this end, we design Algorithm 4. Compared to Algorithm 3, we consider the additional factor of learning models being shared from another existing system, which will be loaded directly in the beginning. By doing this, the learning time can be largely reduced. Similar to other algorithms, the overall complexity of Algorithm 4 is $O(n)$, where n is the number of training epochs.

V. Performance Evaluation

In this section, we provide the performance evaluation results, which demonstrate the efficacy of four algorithms designed for the four scenarios defined in Section IV. In the following, we first present the methodology and then introduce and discuss the evaluation results.

A. Methodology

In all four scenarios, we use a representative process control system called the Continuous Stirred Tank Reactor (CSTR) system [28], which is a fluid temperature control system. In the CSTR system, the objective is to control the temperature of the liquid in the tank by modifying the steam flow rate. As it is a representative system used in the industrial field,

Algorithm 4: Algorithm for Scenario D

```
Result: Q(s,a)
Initialization: Q(s,a) ← another system, α=1/(2(k+1)), k=0, diff=0,
  states, target, sum=0, diffmean=0, windowsize, i=0, flag=0,
  system_working_time
while iteration < system_working_time do
    k++
    iteration++
    while i < windowsize do
        diff = abs(controlled_variable_i - target)
        sum += diff
        i++
    end
    diffmean = sum / windowsize
    sum = 0
    i = 0
    if diffmean <= threshold_d && α <= threshold_α then
        flag = 1
    end
    if diffmean > threshold_d && flag == 1 then
        k = 0
    end
    update: Q(s,a) ← (1-α)*Q(s,a)+α*(1-diffmean)
    Save updated Q-table
end
```

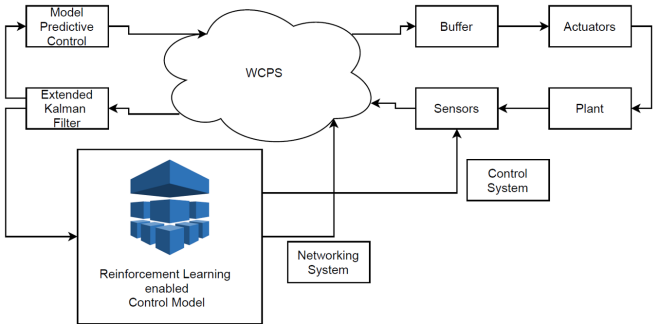

Fig. 4. Architecture of the CSTR System

we leverage this system as our implementation in WCPS to validate our approach.

To illustrate how the system works, Fig. 4 shows the architecture of the system. The sensors in the plant (CSTR) first send data through the WCPS module and transmit them to the Kalman filter [29]. With the Kalman filter, we can observe the condition of the network and estimate the state based on the system input, which is the sensing data. The state observer of the Kalman filter predicts the next system state based on the previous one. It performs the update by comparing its predicted state value with the actual state value. If the network has any problem, such as packet loss, the observer only outputs the predicted value. The model prediction controller (MPC) manages the commands and modifies them based on the system estimation, and transmits them to the plant. The reinforcement learning model takes the estimated system input state and adjusts the control system and networking system constantly.

In our simulation, the controllable objectives of the CSTR system are the target temperature, steam flow rate, and sampling rate. The target temperature refers to the temperature of the liquid in the water tank that should be reached. Steam flow represents the speed of hot steam flowing in the heating tube.

Since our CSTR system increases the temperature of the liquid in the reaction container via the heat exchange between the heating tube and the liquid in the reaction container, the steam flow will affect the heat exchange rate and will determine the change in the temperature of the reaction container. The sampling rate is the sampling frequency of the sensor. A higher sampling rate can provide more accurate temperature information but more data needs to be transmitted. With the increase of training time, the temperature of the liquid in the water tank gradually approaches the target temperature. If the difference between the target temperature and the current temperature is within 0.5 °C, we consider the control system has achieved the objective and remains stable.

Recall that in scenario B, we consider the policy to determine what kind of Q-table can be shared. We will evaluate the effectiveness of such a policy. To this end, we consider another IIoT system, called the DC Motor system [30], as the source from which to share the Q-table with the CSTR system. Note that the control objective of the DC motor system is the voltage and the target speed measured by rotations per minute (rpm). The system needs to control the voltage to keep the speed of the motor close to the target speed.

The software that we used in the experiment is MATLAB r2020a and Docker. The software ran on a PC with Windows 10 version 2004. The hardware in this PC is an Intel i7-8700 CPU and 16 GB DDR4 RAM.

B. Results

In the following, we show the performance evaluation results of four scenarios, which correspond to the four algorithms detailed above.

1) Scenario A: Non-Continuous learning without Learning Model Sharing: To evaluate the efficacy of Algorithm 1, we set the sampling rate to ten samples per second, meaning that the sampling rate is relatively high and the network conditions in this scenario are good. The state that we define in this scenario is based on the the range from 0 °C to 0.1 °C as state 1, 0.1 °C to 0.3 °C as state 2, 0.3 °C to 0.5 °C as state 3, 0.5 °C to 2 °C as state 4, 2 °C to 4 °C as state 5, 4 °C to 6 °C as state 6, 6 °C to 8 °C as state 7, and greater than 8 °C as state 8. Based on the different states, the CSTR system needs to adjust its steam flow rate to achieve the target temperature. As shown in Fig. 5, the model updates every 4 s. We use different colors in the same curve to represent each update time period. The green dash in this figure is the target temperature, representing the target that the CSTR system needs to achieve. The multi-colored curve shown in the figure represents the current temperature in the CSTR system. Also, there is some fluctuation in the curve, meaning that the system is still learning from the environment. We can see that the curve achieves convergence in 68 s, confirming that Algorithm 1 is effective for scenario A. Note that in all cases, the convergence means that the difference between the target temperature and the current temperature reaches 0.1 °C.

2) Scenario B: Non-Continuous Learning with Learning Model Sharing: In scenario B, we investigate how well offline non-continuous learning performs with different types of model sharing: same system, similar system, and different physical systems. This scenario evaluates Algorithm 2.

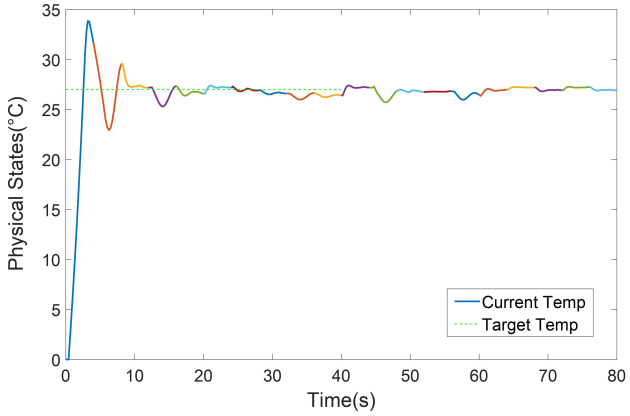

Fig. 5. Non-continuous Learning without Learning Model Sharing

- **Learning model shared between same physical systems:** In this case, we evaluate the efficacy of Algorithm 2 to deal with non-continuous learning with learning model sharing. To do so, we use two 8-state CSTR systems to conduct learning model (i.e., Q-table) sharing. The sampling rate is set to 10 samples per second, meaning that the network condition is good. As shown in Fig. 6, the model is updated every 4 s, each of which corresponds to a different color in the curve. The green dash represents the target temperature, while the curve represents the current temperature in the CSTR system. From the figure, we can see that the curve achieves convergence in 16 s, confirming that sharing the learning model (i.e., Q-table) can improve performance significantly by reducing the time to reach convergence.
- **Learning model shared between similar physical systems:** In this case, to build a similar physical system, we modify the state definition in the 8-state CSTR system and change it to a 16 state CSTR system. For example, in the 8-state CSTR system, we define the temperature range from 0 °C to 0.1 °C as state 1. In the 16-state CSTR system, we define the temperature range from 0 °C to 0.05 °C as state 1. In this scenario, we first ensure that Algorithm 2 in the 16 state CSTR system can work effectively, achieving baseline performance to measure improvement. From Fig. 7, the system is under a good network condition, where the green dash represents the target temperature. The model update period is set to 4 s, each of which corresponds to a different color in the curve. The curve represents the real temperature in the CSTR system. We can see from the figure that the curve archives convergence in 140 s, confirming that Algorithm 2 is effective in the 16 state CSTR system. Next, we load the Q-table from the 8-state CSTR system to the 16-state CSTR system with some modifications. For example, state 1 and state 2 of the Q-table in the 16-state CSTR system remains the same as state 1 of the Q-table in the 8-state CSTR system. The result is shown in Fig. 8. We can observe that the curve achieves convergence in 16 s, confirming that Algorithm 2 can

improve performance by reducing the time to reach convergence, even when the shared model is of a different scale.

- **Learning model shared between different physical systems:** In this case, we use a DC Motor system to generate an 8-state Q-table and share it with an 8-state CSTR system to validate its performance. By comparing Fig. 5 to Fig. 9, we can see that under the same network conditions, the Q-table sharing between different physical systems has a negative impact on the system performance. We can see in Fig. 5 that the system achieves convergence in 68 s. In Fig. 9, the system takes 158 s to achieve convergence. The results from both figures confirm that the learning model shared between different physical systems can deteriorate the learning effectiveness.

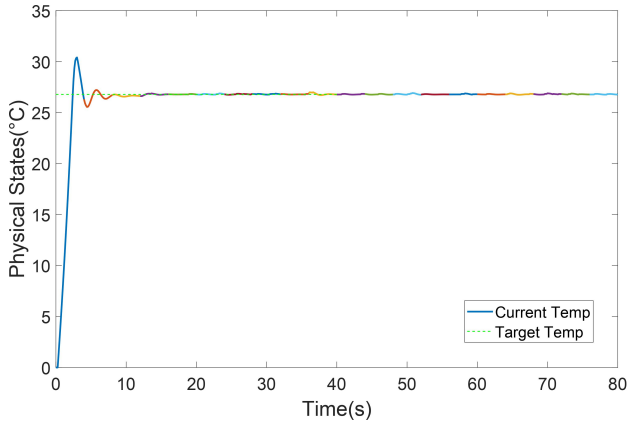

Fig. 6. Non-continuous Learning with Learning Model Sharing for the Same CSTR Systems

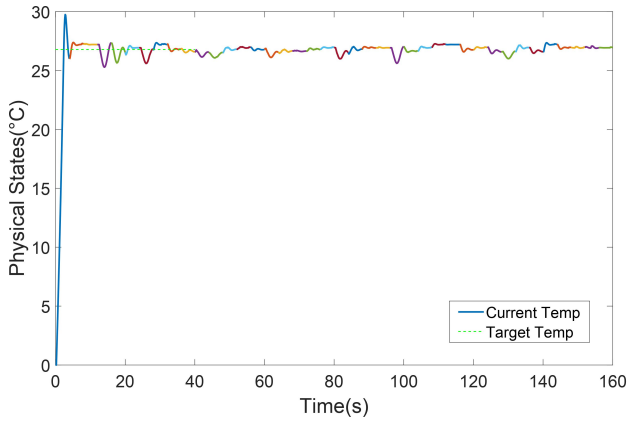

Fig. 7. Non-continuous Learning without Learning Model Sharing for 16-state (Similar) CSTR system

3) Scenario C: Continuous Learning without Learning Model Sharing: Scenarios A and B do not consider updates to the learning model (i.e., Q-table) over time, otherwise denoted as continuous learning. When the environment (e.g., control objectives, network conditions) changes, we need to re-engage the learning process by resetting the learning rate to rapidly return the system performance to a desirable level. To simulate environment dynamics, we change the network conditions from good to bad.

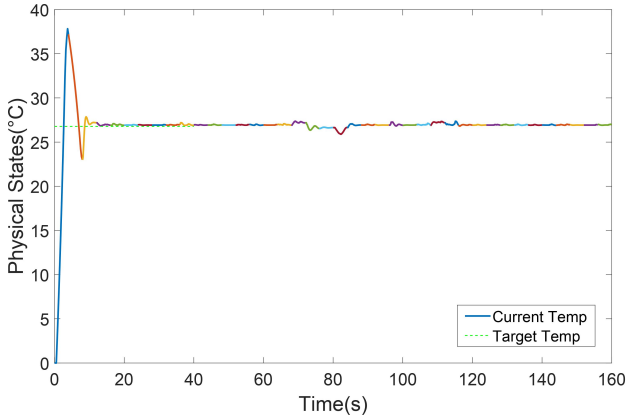

Fig. 8. Non-continuous Learning with Learning Model Sharing for the 16-state Similar CSTR system

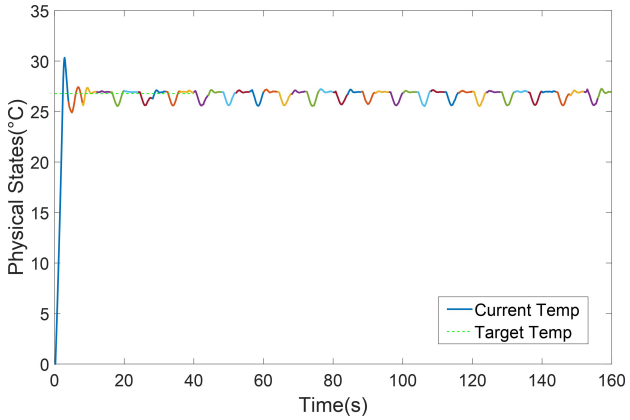

Fig. 9. Non-continuous Learning with Learning Model Sharing between Different Systems

As shown in Fig. 10, the model updates every 4 s and represents in the curve in different colors. The sampling rate is reduced from 10 samples per second to 1 sample per second at time 80 s, which means the network condition changes from good to bad at time 80 s. In addition, the target temperature will change from 27 °C to 35 °C at the same time (80 s). Based on Algorithm 3, when the difference between the target temperature and the current temperature exceeds the threshold of 0.5 °C, the learning process is re-triggered. From the figure, we can observe that Algorithm 3 can successfully re-trigger the learning process when the environment changes. When the network is under good condition, the convergence time is at 48 s. When the network condition is bad, the convergence time is at 76 s.

4) Scenario D: Continuous Learning with Learning Model Sharing: In scenario D, we investigate the performance of online continuous learning in combination with model (i.e., Q-table) sharing, otherwise denoted as Algorithm 4, when sharing occurs between the same IIoT systems. We use two 8-state CSTR system to conduct the Q-table sharing. The sampling rate is set from 10 samples per second to 1 sample per second, meaning that the network condition is getting worse. The target temperature is also changing, starting at 27 °C and changing to 35 °C, then 17 °C, and finally 23 °C. The environment changes at 40 s, 80 s, and 120 s. As shown in Fig. 11, the model is updated every 4 s, each of which

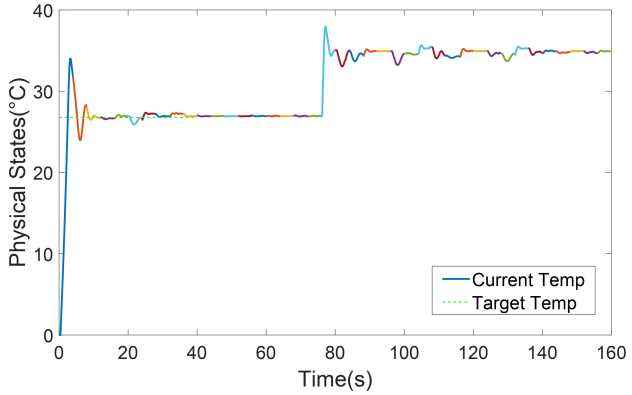

Fig. 10. Continuous Learning without Learning Model Sharing

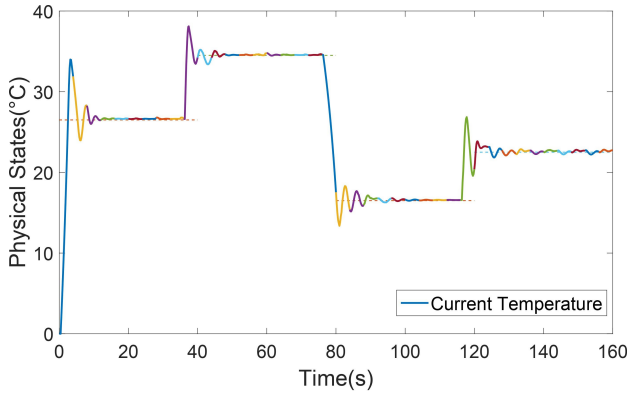

Fig. 11. Continuous Learning with Learning Model Sharing between the Same CSTR systems

corresponds to a different color in the curve. The green dash represents the target temperature. The curve represents the current temperature in the CSTR system. We can see that, from time period 0 s to 80 s, the network condition is good, and the learning model (i.e., Q-table) sharing is effective during this time period. From period (0 to 40 s) and (40 to 80 s) the system achieves convergence in 12 s. From time period (80 to 120 s), the network condition is getting worse, the convergence is 28 s. During the interval (120 to 160 s), the network condition is bad and the convergence time is at 36 s. Even under this situation, the convergence time is still better than Algorithm 1 in Fig. 5, which needs 68 s to achieve convergence. Also, at times 40 s, 80 s, and 120 s, the learning process is successfully re-triggered. Therefore, we confirm that Algorithm 4 is effective.

VI. Final Remarks

In this paper, we proposed a general framework to apply online continuous learning in IIoT. We categorized the problem of applying reinforcement learning in IIoT systems into four scenarios: scenario A (non-continuous learning without learning model sharing), scenario B (non-continuous learning with learning model sharing), scenario C (continuous learning without learning model sharing), and scenario D (continuous learning with learning model sharing). We used scenario A as the traditional discontinuous benchmark scenario, and designed an online continuous learning approach for scenarios B, and D. To validate the effectiveness of our approach, we applied our online continuous learning approach on a representative IIoT system. Our experimental results demonstrated that, compared to the existing discontinuous approaches, our proposed online continuous learning approach can improve the convergence of reinforcement learning in dynamic IIoT environments, and can achieve better industrial control performance. Future work is necessary to improve this research. First, we shall explore the effectiveness of our proposed approach in other CPS and IIoT scenarios, and extend our framework to other machine learning algorithms such as convolutional neural networks (CNNs), long short-term memory (LSTM), and others. Second, we shall design a theoretical model to quantify the similarity of different CPS and IIoT systems so that learning models of appropriately similar systems can be leveraged to improve the learning convergence of others.

References

[1] H. Xu, W. Yu, D. Griffith, and N. Golmie, "A survey on industrial internet of things: A cyber-physical systems perspective," *IEEE Access*, vol. 6, pp. 78 238–78 259, 2018.

[2] H. Xu, X. Liu, W. Yu, D. Griffith, and N. Golmie, "Reinforcement learning-based control and networking co-design for industrial internet of things," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 885–898, 2020.

[3] N. Kamala, "Studies in modeling and design of controllers for a nonideal continuous stirred tank reactor," 2013.

[4] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24 411–24 432, 2018.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] B. Li, Y. Ma, T. Westenbroek, C. Wu, H. Gonzalez, and C. Lu, "Wireless routing and control: a cyber-physical case study," in *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2016, pp. 1–10.

[7] mathworks, "Simulink documentation," https://www.mathworks.com/products/simulink.html.

[8] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[9] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of industrial information integration*, vol. 6, pp. 1–10, 2017.

[10] P. Zhang, Y. Wu, and H. Zhu, "Open ecosystem for future industrial internet of things (iiot): Architecture and application," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 1–11, 2020.

[11] C. Zhou, N. Damiano, B. Whisner, and M. Reyes, "Industrial internet of things:(iiot) applications in underground coal mines," *Mining engineering*, vol. 69, no. 12, p. 50, 2017.

[12] V. Sklyar and V. Kharchenko, "Challenges in assurance case application for industrial iot," in *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 2, 2017, pp. 736–739.

[13] J. Xiong, R. Ma, L. Chen, Y. Tian, Q. Li, X. Liu, and Z. Yao, "A personalized privacy protection framework for mobile crowdsensing in iiot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4231–4241, 2019.

[14] G. Hatzivasilis, K. Fysarakis, O. Soultatos, I. Askoxylakis, I. Papaefstathiou, and G. Demetriou, "The industrial internet of things as an enabler for a circular economy hy-lp: a novel iiot protocol, evaluated on a wind parks sdn/nfv-enabled 5g industrial network," *Computer Communications*, vol. 119, pp. 127–137, 2018.

[15] K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu, "Green industrial internet of things architecture: An energy-efficient perspective," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 48–54, 2016.

[16] W. Dou, X. Zhao, X. Yin, H. Wang, Y. Luo, and L. Qi, "Edge computing-enabled deep learning for real-time video optimization in iiot," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2020.

[17] W. Liu, G. Huang, A. Zheng, and J. Liu, "Research on the optimization of iiot data processing latency," *Computer Communications*, vol. 151, pp. 290–298, 2020.

[18] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in sdn-based industrial internet of things with edge computing," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1351–1360, 2018.

[19] S. Lele, "Analysis of well head pressure sensor data for anomaly detection in oil well using iiot and unsupervised learning technique," 2019.

[20] A. Kanawaday and A. Sane, "Machine learning for predictive maintenance of industrial machines using iot sensor data," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, pp. 87–90.

[21] G. Shah and A. Tiwari, "Anomaly detection in iiot: A case study using machine learning," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2018, pp. 295–300.

[22] B. Yang, X. Cao, X. Li, Q. Zhang, and L. Qian, "Mobile-edge-computing-based hierarchical machine learning tasks distribution for iiot," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2169–2180, 2020.

[23] P. C. M. Arachchige, P. Bertok, I. Khalil, D. Liu, S. Camtepe, and M. Atiquzzaman, "A trustworthy privacy preserving framework for machine learning in industrial iot systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6092–6102, 2020.

[24] M. Liu, F. R. Yu, Y. Teng, V. C. Leung, and M. Song, "Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: A deep reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3559–3570, 2019.

[25] B. Chen, J. Wan, Y. Lan, M. Imran, D. Li, and N. Guizani, "Improving cognitive ability of edge intelligent iiot through machine learning," *IEEE Network*, vol. 33, no. 5, pp. 61–67, 2019.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] M. O. Duff, "Q-learning for bandit problems," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 209–217.

[28] S. Ge, C. Hang, and T. Zhang, "Nonlinear adaptive control using neural networks and its application to cstr systems," *Journal of process control*, vol. 9, no. 4, pp. 313–323, 1999.

[29] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[30] Mathworks, "Dc motor," https://www.mathworks.com/help/control/getstart/linear-lti-models.html.