Neighboring State-Aware Policy for Deep Reinforcement Learning

Meng Xu, Xinhong Chen, Guanyi Zhao, Zihao Wen, Weiwei Fu, and Jianping Wang, *Fellow, IEEE*

***Abstract*—Deep reinforcement learning (DRL) methods, which train a policy to obtain the sequence of actions required to complete a task, have achieved remarkable success across diverse applications. It is a long-standing open issue in the DRL community to make the trained policy gradually approach the theoretically globally optimal policy, and existing research has also explored several challenges, such as exploration-exploitation, to improve the quality of the obtained policy. However, most DRL methods rely solely on the current state for decision-making, leading to short-sightedness and suboptimal learning. To overcome this, we propose a neighboring state-aware policy that enhances existing DRL methods by incorporating a neighboring state sequence in the decision-making process. Specifically, our approach saves multiple past and future states and concatenates them as the neighboring state sequence, along with the current state, and inputs them to the actor to generate an action during the training process. This global perspective, provided by neighboring states, is similar to human decision-making and helps the agent better understand state evolution, leading to improved policy learning. We present two specific implementations of our approach and demonstrate through extensive experiments that it effectively enhances ten representative DRL methods across nine tasks, based on three metrics, including return.**

***Index Terms*—Deep reinforcement learning (DRL), global perspective, neighboring state sequence, policy learning.**

Nomenclature

θ^{μ}	Actor.
θ	Critic.
$Q(s, \cdot)$	Q-value.
$\hat{Q}(s, \cdot)$	Target Q-value.
$\pi(\cdot; \theta^{\mu})$	Action policy.
$\mathcal{L}(\theta)$	Loss function.
r	Reward.
s	State.
a	Action.
$\mathcal{D}_0$	Mini-batch.
$\mathcal{D}$	Replay buffer.
N	Size of the mini-batch.
SA_t	Primitive sample.
Macro_t	Macro-sample.
neig_t	Neighboring state sequence.
h	Representation vector.

Received 9 October 2024; revised 20 February 2025, 5 May 2025, and 28 May 2025; accepted 11 June 2025. Date of publication 26 June 2025; date of current version 9 October 2025. This work was supported in part by Hong Kong Research Grant Council through the General Research Fund (GRF) under Grant 11216323 and in part by the Research Impact Fund (RIF) Project under Grant R5060-19. *(Corresponding authors: Xinhong Chen; Jianping Wang.)*

The authors are with the Department of Computer Science, City University of Hong Kong, Hong Kong, SAR, China (e-mail: mxu247-c@my.cityu.edu.hk; xinhong.chen@cityu.edu.hk; guanyzhao3-c@my.cityu.edu.hk; zihaowen2@cityu.edu.hk; weiweifu2-c@my.cityu.edu.hk; jianwang@cityu.edu.hk).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNNLS.2025.3581217, provided by the authors.

Digital Object Identifier 10.1109/TNNLS.2025.3581217

I. Introduction

DRL has seen widespread adoption in various applications, such as recommendation systems [1], [2], [3], autonomous driving [4], and robot control [5], [6], due to its robust learning capabilities. Currently, the actor-critic architecture is the most commonly used DRL structure, which employs an actor to generate an action based on the current state, and a critic to evaluate the value of that action. Notably, learning a better policy to achieve higher returns has remained a long-standing challenge in DRL.

Although the current DRL community has made great progress in addressing well-known technical dilemmas such as the exploration-exploitation dilemma [7], [8], [9], enabling DRL to learn better policies, existing DRL methods still harbor a technical limitation. Specifically, current DRL methods typically utilize only the current state as input when generating actions with policies. In fact, research in psychology shows that human decision-making often involves considering past, present, and future states, leading to more informed decisions by analyzing the evolution of events [10], [11]. Likewise, DRL research is frequently inspired by human decision-making psychology [12]. Motivated by these observations, integrating not only the current state but also past and future states—the neighboring state sequence—into DRL agents' decision-making would enable them to better understand which states led to the current state and what subsequent states may arise from it. This would facilitate the analysis of state evolution, referred to as global awareness, thereby leading to the development of improved policies. The lack of global awareness of existing DRL methods results in suboptimal policies that hinder the effective execution of many real-world tasks. For example, a wheeled robot that makes short-sighted movement decisions based solely on the current environmental state may develop a suboptimal policy, leading to difficulties in navigating through an obstacle-laden environment without collisions, which can result in inefficient task performance and potential damage to the robot.

Thus, this work aims to propose a novel policy that can be seamlessly integrated into existing DRL methods, leveraging the neighboring state sequence to determine better actions and learn improved policies. To this end, two significant challenges must be addressed to achieve this goal. **First**, collecting multiple past and future states for model training presents a challenge. In particular, future states are more difficult to collect directly than past states. This is because, during the training process, the model gathers samples in real-time and updates itself accordingly. Consequently, the model can only utilize states that have already occurred for updates, while future states, being unobserved, cannot be obtained. **Second**, there are significant differences in the structure and information content of the neighboring state sequence and the current state. Specifically, the neighboring state sequence contains multiple past and future states along with the current state, resulting in a large volume of information and a long temporal dimension. In contrast, the current state consists of immediate information and has a relatively simple structure. Therefore, these two inputs cannot be simply concatenated for the actor, making the effective processing of these two inputs another challenge.

To address the first challenge, we develop an n-step delayed model update approach. Specifically, at each training step, existing DRL methods typically incorporate only the current state into the samples for model updates, resulting in the update process relying solely on this single state. In contrast, at each training step, we utilize a queue to store multiple consecutive past states, forming a state sequence that is added to the samples. We then use these samples, which include the state sequence, to update the model from n steps ago. Consequently, for the model from n steps ago, the states collected in the subsequent n steps are considered future states. Thus, this model's delayed update method categorizes the states in the sequence into past and future states based on the moments when the model is supposed to be updated but has not yet been updated.

To address the second challenge, we employ distinct methods for processing the neighboring state sequence and the current state. Specifically, for the neighboring state sequence, we utilize a model capable of handling long data sequences and extracting key features and relationships between states, such as a 2-D convolutional neural network (CNN) or a Transformer, to convert it into a representation vector. In contrast, for the current state, we adopt a fully connected layer (FCL), which is well-suited for processing short input information, to represent it as a vector. The two representation vectors are then concatenated and passed through two hidden layers to determine an action. Finally, we present a theoretical analysis of our method's capacity to enhance existing DRL approaches, bringing them closer to the optimal policy. In addition, we conduct a complexity analysis to demonstrate that our approach does not introduce significant additional complexity, thereby ensuring its effectiveness.

In summary, our contributions are as follows.

1) We propose a novel perspective to implement a policy, referred to as the neighboring state-aware policy, which can be compatible with any specific model (e.g., 2-D CNN) and can be seamlessly integrated into existing DRL methods to help them learn better policies. In particular, our method is easy to implement and has achieved remarkable performance.
2) Our method saves the past and future states during the training process and concatenates them as the neighboring state sequence, which is then combined with the current state as the input to the actor to generate an action. We present two specific implementations of our approach, using 2-D CNNs and Transformer architectures, respectively. Furthermore, we provide a theoretical analysis and complexity analysis of our method.
3) Extensive experimental results show that our method effectively improves ten representative DRL methods across nine tasks, based on three metrics, including return. Eight of these are the latest state-of-the-art (SOTA) DRL methods, while the other two are commonly used approaches.

This article is organized as follows. Section II reviews related work, Section III covers background and problem definition, Section IV presents the proposed approach, Section V reports the experimental results, and Section VI concludes the study.

II. Related Work

A. DRL

The main objective of DRL is to determine an optimal policy that maximizes the cumulative discounted rewards over time. DRL's formal emergence was in 2015, when Mnih et al. [13] introduced deep neural networks to handle high-dimensional observations for policy learning. This advancement paved the way for DRL, which has now been successfully deployed to numerous application tasks, and has even become the mainstream solution for various applications. The actor-critic architecture is currently the most prevalent in DRL. In recent years, numerous SOTA actor-critic methods have been proposed to address various challenges inherent to DRL, such as exploration and exploitation [7], [9], [14], overestimation of Q-values [15], [16], [17], reward decomposition [18], [19], lifelong learning [20], visual representation [21], and the difference in data distribution between off-policy samples and current policies [20], [22].

However, existing DRL methods typically only input the current state to the actor to generate actions. This approach provides the actor with information about the current state alone, which can lead to short-sighted action decisions and suboptimal policies. In contrast to existing methods, our approach utilizes both the neighboring state sequence and the current state as input to the actor. This allows the agent to better analyze the state transition process during training, enabling it to make more informed and prudent decisions.

B. DRL Policy Using Additional Input

Providing more input to the DRL policy is an effective way to facilitate learning a better policy. In the DRL community, there have been efforts to leverage contextual task information to improve the generalization capability of DRL in multitask scenarios [23], [24], [25], [26], [27]. Recent research has also explored integrating contextual tasks into Transformer

[28], [29], meta-learning [30], [31], [32], [33], offline DRL [34], and imitation learning [28], [35] to achieve cross-task online or offline learning. However, these existing contextual DRL approaches typically use contextual information referring to multiple related tasks (e.g., environment parameters) and aim to achieve generalization across different scenarios by effectively integrating this task-level information. In contrast, our approach of incorporating past, present, and future states is focused on improving action decision-making within a single task, rather than achieving cross-task generalization.

A related type of DRL is n-step learning, which utilizes future multiple-step returns to compute a Q-value, effectively addressing the challenge of inaccurate Q-value estimation [36], [37], [38]. However, our approach differs from these studies, as we aim to provide the actor with neighboring state information, including past and future states, to generate actions, rather than to improve the Q-value estimation. Another type of DRL uses past state memories to build an environment model and infer the true state of the environment to solve policy learning in partially observable environments [39], [40], [41], [42], [43]. While our method aims to propose a novel and general perspective of implementing policies, rather than building an environment model. Our method also differs from approaches that use multiple image frames as state inputs for decision-making in DRL models, such as [21], because even though these methods input multiple consecutive image frames, these frames are concatenated and treated as a single current state.

Recent studies have investigated adding noise to the current state to create similar noisy states and using these noisy state sequences along with the current state to train noise-robust DRL agents [44]. Other methods have used a few past states as constraints to address DRL policy learning in constrained environments [45], while also leveraging recurrent neural networks to integrate several past states for improving DRL learning [46]. Beyond single-agent DRL, in multiagent DRL, existing research also incorporates neighboring agents' state information to improve individual decisions and team collaboration, such as [47] and [48]. These methods differ from ours, which involves directly incorporating a sequence of multiple neighboring states into the policy. Overall, while our method differs from existing DRL approaches, it is not in conflict with them and can be integrated by introducing neighboring state sequences into the policy input.

III. Preliminaries

This section presents the problem formulation and overviews two commonly used DRL algorithms, twin delayed deep deterministic (TD3) policy gradient and soft actor-critic (SAC), which underpin advanced techniques like lifelong DRL [20] and sparse DRL [49], and are used in our experiments. Key notations are listed above.

A. Problem Formulation

The DRL framework employs a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, \mathbb{P}, \gamma \rangle$, which defines the state space $\mathcal{S}$, action space $\mathcal{A}$, reward function r, transition matrix $\mathbb{P}$, and discount factor γ. Within this MDP, the agent's decision-making is guided by a policy $\pi{:}\mathcal{S} \rightarrow \mathcal{A}$. At each time step t, the DRL agent selects an action a_t from $\mathcal{A}$ based on the current state $s_t \in \mathcal{S}$, and receives a reward r_t from the environment. The agent's objective is to learn an optimal policy $\pi(s; \theta^\mu)$ that maximizes the cumulative long-term reward

$$J(\pi(\theta^\mu)) = \max_{\theta^\mu} \mathbb{E}_{\pi(\theta^\mu)} \left[\sum_{i=t}^{\mathrm{T}} \gamma^{i-t} r_i | s_0, a_0 \right]. \tag{1}$$

In the current DRL community, a key architectural approach is the actor-critic method. This method comprises two components: an actor, denoted as θ^μ, which represents the policy $\pi(s; \theta^\mu)$, and a critic, represented by θ, which parameterizes the Q function $Q(s, a; \theta)$. The Q function is designed to estimate the expected cumulative reward of following the policy π when starting from state s and taking action a, as follows: $Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{\mathrm{T}} \gamma^{i-t} r_i | s_t = s, a_t = a \right]$.

In existing DRL methods, the actor usually generates an action a_t based only on the current state s_t, which is $a_t = \pi(s_t; \theta^\mu)$. This leads to short-sighted decision-making, resulting in a suboptimal policy. In contrast, we propose to use both the neighboring state neigh and the current state together for the actor to make action decisions, as in $a_t = \pi(s_t, \mathrm{neigh}; \theta^\mu)$, which can help existing DRL methods learn a better policy.

B. Commonly Used DRL Algorithms

1) TD3: TD3 modifies deep deterministic policy gradient (DDPG) with two key changes: twin critics to reduce overestimation by using separate critics for Q-value estimation, and delayed updating to decrease correlation between consecutive updates by updating the actor and critics less frequently. The objective function of TD3 can be expressed as follows: $\mathcal{L}(\theta^\mu, \theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\sum_{i=1}^{2} (Q_i(s, a) - y)^2 \right]$. Here, $\mathcal{L}(\theta^\mu, \theta)$ is the objective function to minimize, (s, a, r, s') is a transition from the replay buffer $\mathcal{D}$, $Q_1(s, a)$ and $Q_2(s, a)$ are the Q-values from twin critics, and $y = r + \gamma \min_{i=1,2} Q_i^{\mathrm{target}}(s', \pi_{\mathrm{target}}(s'))$ is the target Q-value. TD3 uses an actor to learn the policy $\pi_{\theta^\mu}(a|s)$ and two critics to estimate $Q_1(s, a)$ and $Q_2(s, a)$. The actor is updated by maximizing the first critic's Q-value, while the critics are updated by minimizing the objective function.

2) SAC: SAC combines actor-critic methods with entropy regularization to optimize policy for maximizing cumulative reward and enhancing exploration. The objective function of SAC can be expressed as follows: $\mathcal{L}(\theta^\mu, \theta) = \mathbb{E}_\pi[Q_1(s, a) - \alpha' \log(\pi_{\theta^\mu}(a|s))]$. Here, $\mathcal{L}(\theta^\mu, \theta)$ is the objective function to maximize, $\pi_{\theta^\mu}(a|s)$ is the policy, and α' is the temperature parameter controlling entropy regularization. SAC includes an actor for learning the policy and two critics for estimating the Q-values $Q_1(s, a)$ and $Q_2(s, a)$. The actor is updated with stochastic gradient ascent to maximize the objective, while the critics are updated by minimizing the temporal difference (TD) loss.

IV. Methodology

This section introduces our framework and training process, along with the corresponding pseudocode. We then present implementations based on 2-D CNN and Transformer architectures, commonly used for processing long data sequences, and conclude with a theoretical and complexity analysis.

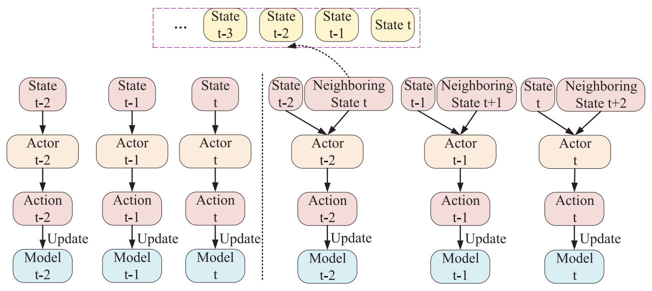


Fig. 1. Training process for our method. This figure compares DRL utilizing neighboring states (right) with conventional DRL (left).

A. Overview of Our Perspective

An overview of our perspective is illustrated in Fig. 1, which shows the training process of our method, and Fig. 2, which depicts the framework of our approach.

In Fig. 1, we illustrate the training process using an MDP with three-time steps (e.g., $t-2$, $t-1$, and t). In this process, conventional DRL methods typically utilize only the current state to generate actions for the actor during training. In contrast, our approach employs both the neighboring state sequence and the current state, using an n-step delayed update method (e.g., with $n = 2$ as shown in Fig. 1) to train the model. Specifically, at step t, we utilize a stack to store multiple consecutive past states, forming a state sequence that, along with the state at $t - n$, serves as input for the actor. This enables the actor to generate actions used to update the model at $t - n$. For the model at $t - n$, the states from $t - n + 1$ to t in the state sequence represent future states, while the states prior to $t - n$ denote past states; therefore, we refer to this state sequence as the neighboring state sequence for state $t - n$.

In Fig. 2, we input the current state s along with its corresponding neighboring state sequence neig into the actor, which subsequently generates an action a. Given that the neighboring state sequence is a lengthy data sequence, we employ a model adept at handling such sequences, such as a 2-D CNN or Transformer, to process it and derive a feature vector. This feature vector is then input into an FCL to obtain a representation vector h_1. Simultaneously, the current state is also fed into a separate FCL to produce a representation vector h_2. The two representation vectors, h_1 and h_2, are concatenated to form the final representation vector $h = h_1 \oplus h_2$. This vector h is then passed through two hidden layers to generate the final action. The critic structure follows the baseline DRL method, using only the current state s and action a to estimate the Q-value. Finally, the TD error is computed using the current Q-values from two critics and the next Q-values from two target critics to update both critics and the actor.

B. Training Procedure of Our Perspective

In this section, we use the TD3 and SAC algorithms as an example to demonstrate the training process of our perspective. Our perspective can be integrated into any DRL algorithm by making corresponding modifications to the training rules of the actor and critic. Let $\text{neig}_{t+n} = (s_{t-m}, \dots, s_{t-1}, s_t, s_{t+1}, \dots, s_{t+n})$ represent the neighboring state sequence at time $t+n$, where m is the number of past states and n is the number of future states in the sequence. The length of neig_{t+n} is fixed at $m + n + 1$, and it is implemented as a stack data structure, meaning that each step in the training process introduces a new state and removes an old state using a first-in, first-out (FIFO) approach. We implement our perspective as follows.

1) Collecting Samples: During the training phase, at each step, the agent utilizes the actor to process the neighboring state sequence neig_t and the current state s_t, generating an action a_t. The agent then executes the action a_t within the environment, resulting in a reward r_t and a transition to the next state s_{t+1}. Subsequently, s_{t+1} is added to the neighboring state sequence while removing the oldest state, thereby updating it to neig_{t+1}. This sample, denoted as $\text{SA}_t = (\text{neig}_t, s_t, a_t, r_t, s_{t+1}, \text{neig}_{t+1})$, is then acquired and stored in the replay buffer $\mathcal{D}$. Here,

$$\text{neig}_t = (s_{t-m-n}, \dots, s_{t-1-n}, s_{t-n}, s_{t+1-n}, \dots, s_t)$$

and

$$\text{neig}_{t+1} = \left(s_{t-m-n+1}, \dots, s_{t-n}, s_{t-n+1}, s_{t+2-n}, \dots, s_{t+1}\right).$$

At this stage, the neighboring state sequence neig_t provided to the actor comprises solely past states, as future states are unavailable during the sample collection phase. However, in the subsequent model update process, we employ n-step delayed updates to generate future states for the neighboring state sequence.

2) Updating DRL Models: During the training phase, at each step, we sample a mini-batch and update the model using n-step delayed updates. Specifically, when the current training step $t < n$, there are insufficient future states available. In this case, we utilize a neighboring state sequence that includes only past states and the current state to update the model. When $t \geq n$, we start including n future states in the neighboring state sequence to update the model, that is, at the training step t, we update the model at the $t - n$ step. To achieve n-step delayed model training, we first randomly sample samples from the replay buffer.

a) Random sampling: Let SA_t be a primitive sample, and let $\text{Macro}_t = (\text{SA}_{t-n}, \text{SA}_{t+1-n}, \dots, \text{SA}_t)$ be a Macro-sample. Here,

$$\text{SA}_{t-n} = \left(\text{neig}_{t-n}, s_{t-n}, a_{t-n}, r_{t-n}, s_{t+1-n}, \text{neig}_{t+1-n}\right).$$

For each update, we randomly sample N Macro-samples from the replay buffer $\mathcal{D}$ to form a mini-batch $\mathcal{D}_0$, which is then used to update the model. During the subsequent model update process, for each macro-sample Macro_t, we will only use the elements $s_{t-n}, a_{t-n}, r_{t-n}, s_{t+1-n}$ from SA_{t-n} and the elements $\text{neig}_t, \text{neig}_{t+1}$ from SA_t.

b) Updating critics: We first update the critic. The target Q-value $\hat{Q}(s_{t-n}, a_{t-n})$ is computed as

$$\hat{Q}\left(s_{t-n}, a_{t-n}\right) \leftarrow r_{t-n} + \gamma \min_{i=1,2} Q_{\theta^{Q_i'}}\left(s_{t+1-n}, \tilde{a}\right) \tag{2}$$

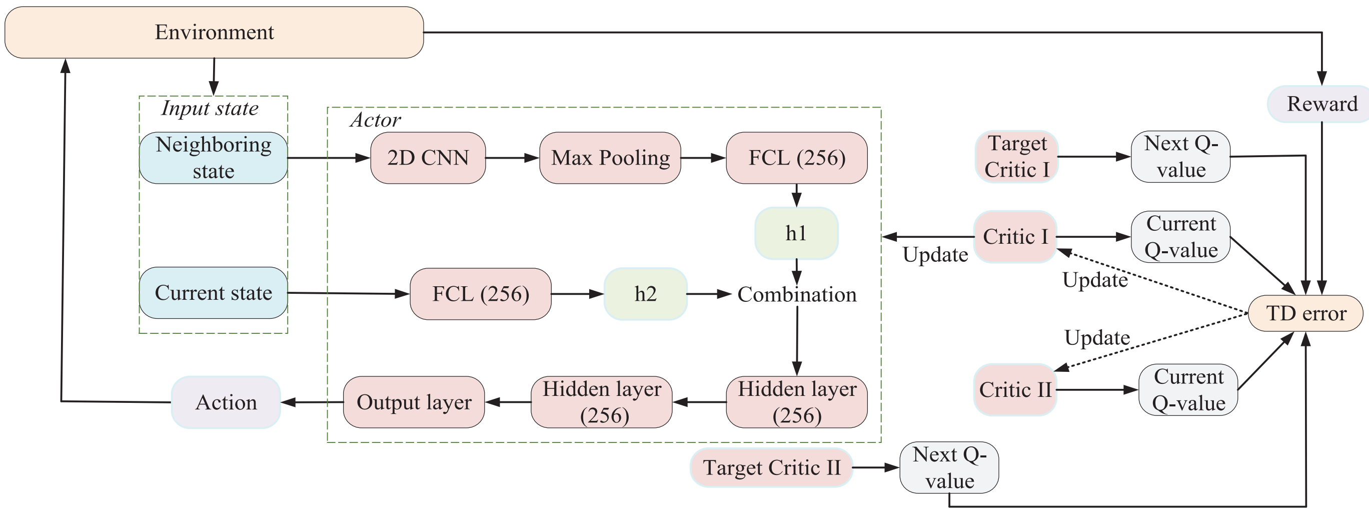


Fig. 2. Framework of our perspective, exemplified through an implementation based on 2-D CNN and integrated into TD3. In this context, the 2-D CNN extracts essential local features from the neighboring state sequence by sliding the convolutional kernel. Max pooling, on the other hand, reduces the size of the feature map by selecting the maximum value from local features, thereby preserving important information and decreasing computational complexity.

where $\tilde{a} \leftarrow \pi_{\theta^{\mu'}}(s_{t+1-n}, \text{neig}_{t+1}) + \epsilon, \epsilon \sim \mathcal{N}(0, 0.2)$. Here, $\theta^{\mu'}$ is the target actor and $\theta^{Q_i'}$ is the target critic. Then, the critic θ^{Q_i} is updated with the target Q-value, as follows:

$$\theta^{Q_i} \leftarrow \theta^{Q_i} + \alpha \nabla_{\theta^{Q_i}} N^{-1} \times \sum_{(s,a)\in\mathcal{D}_0} \left(\hat{Q}(s_{t-n}, a_{t-n}) - Q_{\theta^{Q_i}}(s_{t-n}, a_{t-n})\right)^2 \tag{3}$$

where α represents the learning rate.

c) Updating the actor: Here, we use TD3 and SAC as examples to present the actor update process. Specifically, In the case of TD3, the actor update utilizes two key components: $Q_{\theta^{Q_1}}(s_{t-n}, \pi_{\theta^{\mu}(s_{t-n}, \text{neig}_t)})$ and $\pi_{\theta^{\mu}}(s_{t-n}, \text{neig}_t)$. The gradient $J(\theta^{\mu})$ for the actor is illustrated as follows:

$$J(\theta^{\mu}) = N^{-1} \sum_{(s,a)\in\mathcal{D}_0} \nabla_{\hat{a}} Q_{\theta^{Q_1}}(s_{t-n}, \hat{a}) \nabla_{\theta^{\mu}} \hat{a} \tag{4}$$

where $\hat{a} = \pi_{\theta^{\mu}}(s_{t-n}, \text{neig}_t)$. In contrast, the actor update method differs in the case of the SAC algorithm.

For SAC, the gradient $J(\theta^{\mu})$ is as follows:

$$J(\theta^{\mu}) = \nabla_{\theta^{\mu}} \left(-\frac{1}{N} \sum_{(s,a)\in\mathcal{D}_0} \left(\frac{\sum_{i=1}^{2} Q_{\theta^{Q_i}}(s_{t-n}, \hat{a})}{2} + \alpha' \mathcal{H} \right) \right). \tag{5}$$

Here, $\mathcal{H} = \pi_{\theta^{\mu}}(\hat{a}|s_{t-n}) \log \pi_{\theta^{\mu}}(\hat{a}|s_{t-n})$. The temperature parameter α' determines the degree of entropy $\mathcal{H}$.

3) Updating the Target Networks: After updating the actor and critic, the next step is to update the target networks. For TD3, this includes an actor and two critics, whereas for SAC, only the critics are updated. These updates employ a soft method with a delay, as illustrated below: $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}; \theta^{Q_i'} \leftarrow \tau\theta^{Q_i} + (1-\tau)\theta^{Q_i'}, i = 1, 2$. Here, $\tau = 0.005$ is the coefficient. The above process demonstrates the implementation of the training stage for our method. During the inference stage, we utilize only past states as the sequence of neighboring states, combined with the current state, to generate actions at each time step. This is necessary because, during inference, only past states are available, and future states cannot be accessed. This is formally expressed as $a_t = \mu(s_t, \text{neig}_t|\theta^{\mu})$, where neig_t represents the sequence of past states. Since our method does not alter the Q-value update or model training, it integrates seamlessly into existing DRL approaches without affecting convergence. We present the pseudocode for our method's training process, using the TD3 algorithm as an example (see Algorithm 1).

Remark 1: Our method differs from existing DRL approaches in the following three key aspects. 1) *Architectural Difference:* First, there is a distinction in the actor architecture. Existing DRL methods typically input and process only the current state, lacking a mechanism for handling neighboring state sequences, which limits their ability to learn from these neighboring states. In contrast, our architecture is designed to handle both the current state and neighboring state sequences. 2) *Model Training Difference:* Second, there is a difference in model training. Existing DRL methods update the current model at the current time step, whereas our approach employs n-step delayed updates. This means that at the current time step t, we update the model corresponding to the time step $t-n$. This method ensures that the neighboring state sequence includes n future states. 3) *Composition of Off-Policy Samples:* Finally, there is a difference in the composition of off-policy samples. Existing DRL methods typically utilize quadruplets, represented as (s_t, a_t, r_t, s_{t+1}). In contrast, our method leverages neighboring state sequences $(\text{neig}_t, \text{neig}_{t+1})$, which are also stored in the replay buffer. As a result, our off-policy samples are structured as sextuplets: $(\text{neig}_t, s_t, a_t, r_t, s_{t+1}, \text{neig}_{t+1})$.

C. 2-D CNN-Based Approach for Neighboring State

2-D CNNs are well-established and straightforward models for processing sequential data. They effectively extract latent features and local dependencies between states in a neighboring state sequence, thereby enhancing policy learning in DRL. This section presents the use of 2-D CNNs to process the neighboring state sequence in order to implement our method, as shown in Fig. 3. We first concatenate each

Algorithm 1 Training Procedure of Our Method

```
1: Initialize actor $\theta^{\mu}$ and critics $\theta^{Q_1}$, $\theta^{Q_2}$ with random weights
2: Initialize target networks $\theta^{\mu'}, \theta^{Q'_1}, \theta^{Q'_2}$ with weights $\theta^{Q'_1} \leftarrow \theta^{Q_1}$, $\theta^{Q'_2} \leftarrow \theta^{Q_2}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
3: Initialize replay buffer $\mathcal{D}$
4: Initialize a random process for action exploration
5: Initialize the maximum number of training steps $M$
6: $t = 1$
7: repeat
8:     Receive the current state $s_t$
9:     Select action $a_t = \mathrm{clip}(\mu(s_t, neig_t|\theta^{\mu}) + \epsilon)$
10:    Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
11:    Obtain neighboring state $neig_t$ and $neig_{t+1}$
12:    Store primitive sample $SA_t = (neig_t, s_t, a_t, r_t, s_{t+1}, neig_{t+1})$ in $\mathcal{D}$
13:    Sample a random mini-batch $\mathcal{D}_0$ of $N$ Macro-sample $Macro_t$ from $\mathcal{D}$ and each Macro-sample is $Macro_t = (SA_{t-n}, SA_{t+1-n}, \dots, SA_t)$
14:    Compute the target $Q$-value with each Macro-sample $Macro_t$ in $\mathcal{D}_0$: $\hat{Q}(s_{t-n}, a_{t-n}) = r_{t-n} + \gamma \min_{j=1,2} Q'_j(s_{t+1-n}, \mu'(s_{t+1-n}, neig_{t+1}|\theta^{\mu'}))$
15:    Update critics by minimizing the loss: $\mathcal{L} = N^{-1}\sum(\hat{Q}(s_{t-n}, a_{t-n}) - Q_j(s_{t-n}, a_{t-n}|\theta^{Q_j}))^2$, for $j = 1, 2$
16:    Compute $\hat{a} = \pi_{\theta^{\mu}}(s_{t-n}, neig_t)$
17:    Update actor with each Macro-sample $Macro_t$ in $\mathcal{D}_0$: $N^{-1}\sum \nabla_{\hat{a}} Q_{\theta^{Q_1}}(s_{t-n}, \hat{a}) \nabla_{\theta^{\mu}} \hat{a}$
18:    Every $d$ steps, update target networks: $\theta^{Q'_j} \leftarrow \tau\theta^{Q_j} + (1-\tau)\theta^{Q'_j}$, $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$, for $j = 1, 2$
19:    $t{+}{+}$
20: until $M - t$
```

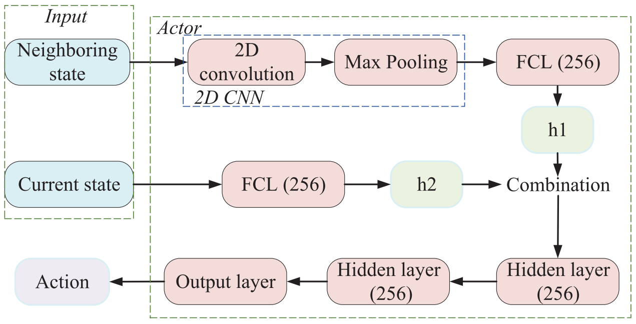


Fig. 3. 2-D CNN-based our policy (actor). Here, 2-D convolution is employed to extract essential local features, while max pooling is utilized to reduce the size of the feature map and decrease computational complexity.

state in the neighboring state sequence. For example, let $\mathrm{neig}_{t+n} = (s_{t-m}, \dots, s_{t-1}, s_t, s_{t+1}, \dots, s_{t+n})$, then we obtain $\mathrm{neig}_{t-m:t+n} = s_{t-m} \oplus s_{t-1} \oplus s_t \oplus \cdots \oplus s_{t+n}$.

We then apply a 2-D convolution operation using a filter $w \in \mathbb{R}^{a\times b}$, where each window consists of a states of length b, on $\mathrm{neig}_{t-m:t+n}$, to obtain a feature vector $fv_i = f(w * s_{i:i+a-1} + b_i)$. Here, $*$ denotes the 2-D convolution between two matrices, b_i is a bias, and f is an activation function. Specifically, the filter w is applied to each window of the states $s_{1:a}; s_{2:a+1}; \dots; s_{m+n-a:m+n-1}$ to produce a feature map $\boldsymbol{fv} = fv_1, fv_2, \dots, fv_{m+n-a-1}$. Finally, we apply a max pooling operation to each feature map $fv_1, fv_2, \dots, fv_{m+n-a-1}$ to capture the feature with the highest value, obtaining $\hat{\boldsymbol{fv}} = \hat{fv}_1, \hat{fv}_2, \dots, \hat{fv}_{m+n-a-1}$, and then calculate the maximum value $h_1 = \max(\hat{fv}_1, \hat{fv}_2, \dots, \hat{fv}_{m+n-a-1})$ as the final feature vector.

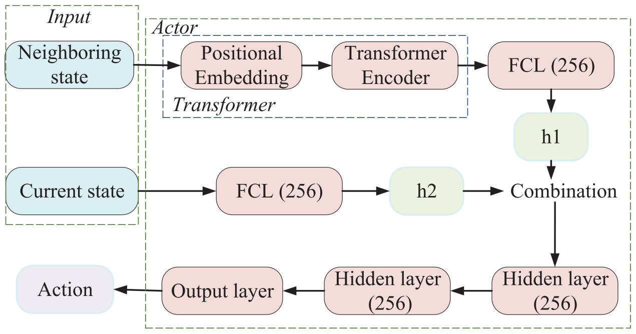


Fig. 4. Transformer-based policy (actor). Here, positional encoding is used to represent the relative positions of each state, while the Transformer encoder extracts essential features.

D. Transformer-Based Approach for Neighboring State

This section introduces an alternative architecture for processing neighboring state sequences: the Transformer architecture. Transformers excel at modeling temporal dependencies and relationships between states within a neighboring sequence. In addition to the two well-established and commonly used models, 2-D CNNs and Transformers, other models designed for processing long data sequences, such as the Mamba model [50] and long short-term memory (LSTM) models, can also be integrated into our method. We now present our approach based on the Transformer, as illustrated in Fig. 4.

The Transformer first processes the neighboring state sequence neig_t using positional encoding, where the dimensionality of the positional encoding is defined as $d_{\mathrm{neig}_t} = (m + n + 1)s$, with s representing the dimensionality of the original state input to the DRL actor. The positional embedding is encoded as follows:

$$\mathrm{PE}_{(\mathrm{pos}\ ,2\ i)} = \sin\left(\mathrm{pos}\ /10000^{2i/d_{\mathrm{neig}_t}}\right)$$
$$\mathrm{PE}_{(\mathrm{pos}\ ,2\ i+1)} = \cos\left(\mathrm{pos}\ /10000^{2i/d_{\mathrm{neig}_t}}\right) \quad (6)$$

where pos is the position and i is the dimension. Positional encoding adds positional information to each state in the neighboring state sequence, enabling the model to understand the relative positions of states within the sequence. After applying positional encoding, we utilize a Transformer encoder layer on the encoded representation $\mathrm{PE}(\mathrm{neig}_t)$, as follows:

$$h_1 = \mathrm{Transformer\_Encoder}\left(\mathrm{PE}\left(\mathrm{neig}_t\right)\right). \quad (7)$$

In this Transformer encoder layer, the multihead attention mechanism allows the model to focus on different states within the neighboring state sequence simultaneously, capturing the complex relationships between states. Each attention head computes a weighted sum based on the relationships among queries, keys, and values, thereby extracting important features. Finally, the outputs of all attention heads are concatenated and subjected to a linear transformation, resulting in a comprehensive feature vector. This vector is further processed through an FCL of size 256 to produce the final feature vector $h_1 \in \mathbb{R}^{\text{hidden\_size}}$ for the neighboring state sequence, where hidden_size denotes the dimensionality of the final output from the Transformer encoder.

E. Theoretical Analysis

This section presents a theoretical upper bound on the difference between the expected cumulative rewards of our policy and the theoretically optimal policy, as well as the difference between the Q-values corresponding to our policy and the optimal policy. We will introduce an assumption, a definition, and two lemmas, with Lemma 2 being derived from the definition. Following this, we will derive Theorem 1, in which we discuss why our approach improves upon existing DRL methods, Theorem 2, which is based on Lemma 1, and Theorem 3, which is based on Lemma 2.

Assumption 1: In DRL settings, the magnitude of the reward signal $|r(s)|$ is constrained by a finite upper bound $R_{\max}$, such that $|r(s)| \leq R_{\max}$.

Definition 1: Let K be a positive constant, and x and y denote arbitrary inputs. For a function $f{:}\mathbb{R}^n \to \mathbb{R}^m$, if there exists a constant $K \geq 0$ such that the following inequality holds:

$$\|f(x) - f(y)\| \leq K\|x - y\|. \tag{8}$$

In this case, the rate of change of the function is globally constrained, and the function is called Lipschitz continuous.

Lemma 1: Based on the findings from the study by [51], for any two actions $\pi_1(s)$ and $\pi_2(s)$ belonging to the action space $\mathcal{A}$, there exists a positive constant C_k such that the following inequality holds:

$$\int_{\mathcal{S}} \left|\text{dis}^{\pi_1(s)} - \text{dis}^{\pi_2(s)}\right| ds \leq C_k \max_{s\in\mathcal{S}} \|\pi_1(s) - \pi_2(s)\| \tag{9}$$

where $\text{dis}^{\pi}{:}\mathcal{S} \to \mathbb{R}$ denotes the occupancy measure [52] of policy π. According to the literature [52], the occupancy measure $\text{dis}^{\pi(s')}$ can be defined as

$$\text{dis}^{\pi(s')} = \int_{\mathcal{S}} \sum_{t=0}^{\infty} (1-\gamma)\gamma^t p_0(s) p\left(s \to s', t, \pi\right) ds \tag{10}$$

where $p(s \to s', t, \pi)$ represents the probability density of transitioning to state s' after t steps, starting from state s and following policy π.

Lemma 2: In DRL, a neural network is frequently used to implement the critic, which calculates the Q-values. Existing studies in DRL usually make the assumption that the critic adheres to the Lipschitz continuity condition. Let K_Q represent a positive constant. This leads to the following bound between the Q-values for $\pi^1(s)$ and $\pi^2(s)$:

$$\left\|Q\left(s, \pi^1(s)\right) - Q\left(s, \pi^2(s)\right)\right\| \leq K_Q \left\|\pi^1(s) - \pi^2(s)\right\|. \tag{11}$$

Theorem 1: Let $\mathbb{E}_{s\sim \text{dis}^{\pi^*(s)}}[r(s)]$ be the expected cumulative reward of the optimal policy π^*, and $\mathbb{E}_{s\sim \text{dis}^{\pi(s,s_{\text{neighbor}})}}[r(s)]$ be the expected cumulative reward of our policy $\pi(s, s_{\text{neighbor}})$, where s_{neighbor} represents the neighboring state sequence. Let $\mathbb{E}s \sim \text{dis}^{\pi(s)}[r(s)]$ be the expected cumulative reward of the existing DRL policy π. Here, according to the literature [52], $\mathbb{E}_{s\sim d^{\pi(s)}}[r(s)]$ is defined as follows:

$$\mathbb{E}_{s\sim \text{dis}^{\pi(s)}}[r(s)] = \int_{\mathcal{S}} r(s)\text{dis}^{\pi(s)} ds. \tag{12}$$

Thus, the difference between the expected cumulative rewards of the optimal policy π^* and our policy $\pi(s, s_{\text{neighbor}})$ is

$$\begin{aligned}
&\left|\mathbb{E}_{s\sim \text{dis}^{\pi^*(s)}}[r(s)] - \mathbb{E}_{s\sim \text{dis}^{\pi\left(s,s_{\text{neighbor}}\right)}}[r(s)]\right| \\
&= \left|\int_{\mathcal{S}} r(s)\left(\text{dis}^{\pi^*(s)} - \text{dis}^{\pi\left(s,s_{\text{neighbor}}\right)}\right) ds\right| \\
&\leq \int_{\mathcal{S}} |r(s)| \left|\left(\text{dis}^{\pi^*(s)} - \text{dis}^{\pi\left(s,s_{\text{neighbor}}\right)}\right)\right| ds \\
&\overset{\text{(Assumption 1)}}{\leq} R_{\max} \int_{\mathcal{S}} \left(\left|\text{dis}^{\pi^*(s)} - \text{dis}^{\pi\left(s,s_{\text{neighbor}}\right)}\right|\right) ds \\
&\overset{\text{(Lemma 1)}}{\leq} C_k R_{\max} \max_{s\in\mathcal{S}} \left\|\pi^*(s) - \pi\left(s, s_{\text{neighbor}}\right)\right\|.
\end{aligned} \tag{13}$$

Similarly, let $\mathbb{E}_{s\sim \text{dis}^{\pi(s)}}[r(s)]$ represent the policy $\pi(s)$ used by existing DRL methods. An upper bound for $|\mathbb{E}_{s\sim \text{dis}^{\pi^*(s)}}[r(s)] - \mathbb{E}_{s\sim \text{dis}^{\pi(s)}}[r(s)]|$ is given by

$$\begin{aligned}
&\left|\mathbb{E}_{s\sim \text{dis}^{\pi^*(s)}}[r(s)] - \mathbb{E}_{s\sim \text{dis}^{\pi(s)}}[r(s)]\right| \\
&\qquad \leq C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s)\|.
\end{aligned} \tag{14}$$

Discussion: From the derivation above, it is evident that a smaller upper bound, such as $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s, s_{\text{neighbor}})\|$, has the potential to bring $\pi(s, s_{\text{neighbor}})$ closer to the theoretically optimal policy, thus enhancing the performance of our method. In the following, we will analyze the relationship between $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s)\|$ and $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s, s_{\text{neighbor}})\|$.

Since $C_k R_{\max}$ is a fixed positive constant and the theoretically optimal policy $\pi^*(s)$ is also fixed, the terms $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s)\|$ and $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s, s_{\text{neighbor}})\|$ are dependent on $\pi(s)$ and $\pi(s, s_{\text{neighbor}})$, respectively. In other words, the relationship between $\pi(s)$ and $\pi(s, s_{\text{neighbor}})$ will dictate the relationship between these two terms. We now proceed to analyze the relationship between $\pi(s)$ and $\pi(s, s_{\text{neighbor}})$.

As noted in previous studies [53], [54], the DRL policy $\pi(s) = \phi^\top(s)w$ can be expressed as a combination of a representation module $\phi(s) \in \mathbb{R}^{N\times 1}$, which is a feature representation of dimension N, and a decision module $w \in \mathbb{R}^{N\times 1}$ (implemented by the final layer of the policy network, typically an FCL). Therefore, the policy $\pi(s)$ in existing DRL methods is expressed as $\pi(s) = \phi^\top(s)w$, while our policy $\pi(s, s_{\text{neighbor}})$ is expressed as $\pi(s, s_{\text{neighbor}}) = \phi^\top(s, s_{\text{neighbor}})w$, where the decision module for $\pi(s, s_{\text{neighbor}})$ is identical to that for $\pi(s)$, both being implemented by an FCL. For our policy, $\phi(s, s_{\text{neighbor}})$ consists of two components: one is the representation $\phi(s)$ of the current state s under the existing DRL policy, and the other is the representation $\phi(s_{\text{neighbor}})$ of the neighboring state. These two representations are concatenated into a single vector, so $\phi(s, s_{\text{neighbor}}) = [\phi(s); \phi(s_{\text{neighbor}})] = [\phi(s); \phi(s_m \oplus s \oplus s_n)]$, where s_m represents the sequence of past states and s_n represents the sequence of future states. Based on this, we can derive the following expression:

$$\begin{aligned}\pi(s, s_{\text{neighbor}}) - \pi(s) &= \phi^\top\left(s, s_{\text{neighbor}}\right) w - \phi^\top(s)w\\ &= \left[\phi^\top(s); \phi^\top\left(s_{\text{neighbor}}\right)\right] w - \phi^\top(s)w\\ &= \left[\phi^\top(s); \phi^\top\left(s_{\text{neighbor}}\right)\right] w\\ &\quad - \left[\phi^\top(s); \mathbf{0}^\top\right] w\\ &= \left[\mathbf{0}^\top; \phi^\top\left(s_{\text{neighbor}}\right)\right] w\\ &= \left[\mathbf{0}^\top; \phi^\top\left(s_m \oplus s \oplus s_n\right)\right] w. \end{aligned} \tag{15}$$

From this derivation, it is clear that $\pi(s, s_{\text{neighbor}}) - \pi(s)$ is directly influenced by $[\mathbf{0}; \phi^\top(s_m \oplus s \oplus s_n)]w$, which, in turn, depends on s_n and s_m. Therefore, the choice of s_n and s_m plays a critical role in determining the relationship between $\pi(s, s_{\text{neighbor}})$ and $\pi(s)$, such as ensuring that $\pi(s, s_{\text{neighbor}}) - \pi(s) > 0$, which will ultimately impact the relationship between $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s)\|$ and $C_k R_{\max} \max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s, s_{\text{neighbor}})\|$. In conclusion, appropriate choices of s_n and s_m will lead to improved performance of our method compared to existing DRL approaches.

Theorem 2: In DRL, the learning process progressively converges toward the optimal policy. As a result, the difference between our policy and the optimal policy is generally bounded, denoted by ρ, which can be formally written as $\max_{s\in\mathcal{S}} \|\pi^*(s) - \pi(s, s_{\text{neighbor}})\| \le \rho$. Consequently, an upper bound for $|\mathbb{E}_{s\sim \text{dis}^{\pi^*(s)}}[r(s)] - \mathbb{E}_{s\sim \text{dis}^{\pi(s,s_{\text{neighbor}})}}[r(s)]|$ is provided by

$$\left|\mathbb{E}_{s\sim \text{dis}^{\pi^*(s)}}[r(s)] - \mathbb{E}_{s\sim \text{dis}^{\pi\left(s,s_{\text{neighbor}}\right)}}[r(s)]\right| \le C_k R_{\max}\rho. \tag{16}$$

Theorem 3: Let $Q(s, \pi^*(s))$ and $Q(s, \pi(s, s_{\text{neighbor}}))$ denote the Q-values for the optimal policy $\pi^*(s)$ and our policy $\pi(s, s_{\text{neighbor}})$, respectively. The difference between these Q-values can be bounded as follows:

$$\begin{aligned}&\left\| Q\left(s, \pi^*(s)\right) - Q\left(s, \pi\left(s, s_{\text{neighbor}}\right)\right)\right\|\\ &\overset{\text{(Lemma 2)}}{\le} K_Q\left(\left\|\pi^*(s) - \pi(s, s_{\text{neighbor}}(s)\right\|\right) \le K_Q\rho. \end{aligned}\tag{17}$$

Remark 2: Our approach adheres to the foundational principles of the DRL process, specifically the MDP, making it theoretically sound. The key rationale is as follows: our method adheres to the Markov property, where the future state depends solely on the current state and action, while past states and actions do not influence future states. Although we incorporate additional neighboring state information from both past and future states as inputs to the DRL policy, these neighboring states influence only the current action decision and do not affect state transitions after the action is executed. Consequently, the future state remains dependent exclusively on the current state and action. Moreover, our method does not modify the essential components of the MDP, including the state space, action space, state transition probabilities, and reward function.

F. Complexity Analysis

Let us analyze the complexity of our method using the example of a 2-D CNN. Specifically, our method builds upon the baseline DRL approach by adding a 2-D CNN layer, a max pooling operator, and an FCL to process the neighboring state sequence. Since the computational and storage costs of the max pooling operator can be ignored, the complexity of our method is determined by the 2-D CNN and FCL components. The **time complexity** is $O(n^2 * k^2 * c_{\text{in}} * c_{\text{out}} + c_{\text{out}} * c_{\text{fc}})$ where n represents the length of the feature map from the neighboring state sequence, k is the size of the convolutional kernel, c_{in} is the number of input channels, c_{out} is the number of output channels in the 2-D CNN layer, and c_{fc} is the output dimension of the FCL. The **space complexity** is $O(k^2 * c_{\text{in}} * c_{\text{out}} + n^2 * c_{\text{out}} + c_{\text{out}} * c_{\text{fc}} + 2M|\mathcal{D}|)$ where n represents the length of the input vector, c_{out} is the number of output channels in the CNN layer, M is the length of the neighboring state sequence, $|\mathcal{D}|$ is the size of the replay buffer, and c_{fc} is the output dimension of the FCL.

Based on the above analysis, it is evident that the 2-D CNN-based implementation of our method does not introduce significant additional complexity compared to the original DRL method, ensuring that our approach can be widely adopted. In fact, the complexity of our method may vary depending on the selected model, allowing users to choose the most suitable model based on their specific requirements. However, regardless of the model chosen, improvements in policy performance come at the cost of increased complexity, a notion widely accepted within the DRL community.

V. Experiments

This section presents the experimental results and analysis, along with the limitations of the current work. The software implementation utilized Torch 1.2.0, Gym 0.16.0, and MuJoCo-py 1.50.0.1, running on an Intel Core i7-9700 processor with 64-GB RAM and an NVIDIA RTX 2080 GPU. Each method is evaluated with five different random seeds.

A. Experiment Preparation

1) Baselines: We select ten representative DRL methods, two of which are commonly used DRL algorithms: the **TD3** and **SAC** algorithms. The other eight are the most recent SOTA methods used as baselines: one step Q-learning-SAC (OSQ-SAC) [22], policy correction-SAC (PC-SAC) [20], stochastic weighted TD3 (SW-TD3) [16], generalized-activated TD3 (GA-TD3) [15], N-step method (NSTEP) [55], dynamic sparse training (DST) with sparsity 0.9 [49], recurrent off-policy DRL (RODRL) [46], and multicritic TD3 (MC-TD3) [17]. Among these, OSQ-SAC and PC-SAC are implemented on top of SAC, while SW-TD3, MC-TD3, DST, NSTEP, and GA-TD3 are implemented on top of TD3. RODRL is implemented for both TD3 and SAC algorithms. In the experiments, we will evaluate the performance of the original version of each baseline as well as their performance after integrating our method, in order to demonstrate the effectiveness of our method in improving existing DRL methods.

2) Benchmarks: We evaluate the performance of our method on five standard MuJoCo tasks (HalfCheetah-v2, Hopper-v2, Walker2d-v2, Humanoid-v2, and Ant-v2), consistent with a wide range of prior DRL literature. Meanwhile, we include four navigation tasks, which are commonly used in DRL research and are more constrained than the standard

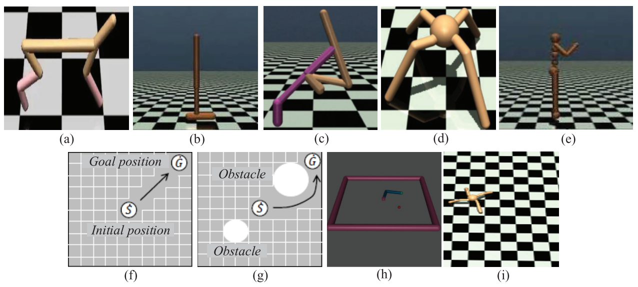


Fig. 5. Experimental scenes. (a) HalfCheetah. (b) Hopper. (c) Walker2d. (d) Ant. (e) Humanoid-v2. (f) 2DNav-v1. (g) 2DNav-v2. In this scenario, the white circle represents an obstacle, with the starting point labeled as S and the target position as G. (h) ReacherNav. (i) AntNav.

MuJoCo tasks. These tasks specify goal positions or obstacles (inaccessible areas) as movement restrictions, serving as additional validation benchmarks: 2DNav-v1 [56], 2DNav-v2 [56], Reacher navigation (ReacherNav) [56], and ant navigation (AntNav) [56]. Here are brief descriptions of the tasks. ReacherNav and AntNav are more challenging variations of the standard MuJoCo tasks (Ant-v2 and Reacher-v2), where the goal is to navigate a two-joint, torque-controlled robotic arm or an ant robot from a starting location to a target location. The tasks 2DNav-v1 and 2DNav-v2 involve guiding a point agent to a goal in a 2-D environment. 2DNav-v2 introduces obstacles, varying both the target position and the arrangement of obstacles as different initial conditions to enhance complexity. Fig. 5 illustrates nine learning tasks.

3) Metrics: To evaluate our method, we use three performance metrics: **the final return** (presented in the main text), **the episode return** (shown in the supplementary material), and **training stability** (presented in the main text). The first two are the most commonly used and key DRL performance metrics, as the DRL community focuses more on the quality of policy performance. Specifically, **the episode return** refers to the average reward accumulated over a set number of episodes and is visualized as a curve over time. In contrast, **the final return** represents the average reward obtained during the last 100 000 steps of the 1 million-step training process, presented in a tabular format. Higher values for both the episode return and final return indicate better performance. **Training stability** refers to the variation in model performance across different random seeds (which correspond to different initial states); smaller variations indicate higher training stability. In DRL, model performance is typically assessed based on the cumulative reward achieved during training.

4) Implementation Details: Here, we present the implementation details of our method as well as the baseline methods, primarily focused on the implementation of the DRL models. The additional implementation specifics of the baseline methods can be found in the corresponding literature and will not be repeated here. All methods under evaluation were implemented with the same set of hyperparameters.

DRL Model: The DRL model employs a multilayer perceptron architecture consisting of three layers: two hidden layers and an output layer. Within this architecture, each hidden layer, present in both the actor and critic modules, is composed of 256 neurons and employs the ReLU activation function. The actor's output layer, on the other hand, utilizes the Tanh activation function. During the training process, a fixed learning rate of 0.001 is applied for both the actor and critic components, with the optimization performed using the Adam optimizer. The mini-batches are formed by selecting 256 samples, and the replay buffer has a capacity of 1 000 000 samples. Furthermore, the models employ a discount factor of 0.99 and undergo a soft update of the target network every 2 steps at a rate of 0.005. Regarding the SAC algorithm, the temperature parameter is set to a constant value of 1, without any automated adjustment.

Our Method: In our work, the neighboring state sequence consists of 21 elements: 16 past states, 4 future states, and 1 current state. The hyperparameter configurations for the 2-D CNN and Transformer are outlined below. *Parameter Configuration for 2-D CNN:* For the 2-D CNN-based implementation, as illustrated in Fig. 3, the first layer of the actor is a 2-D convolutional layer with 1 input channel, 512 output channels, a kernel size of (8, the dimension of the neighboring state sequence), a stride of 2, and no padding. This convolutional layer is followed by a ReLU activation function. Subsequently, a max pooling layer with a size of (1, 1) is used to reduce the spatial dimensions of the convolutional layer's output, resulting in a feature vector of length 512. This feature vector, along with the current state, is then separately input into individual FCLs with 256 neurons each. Finally, the outputs of these FCLs are concatenated and passed through two additional hidden layers, each with 256 neurons. *Parameter Configuration for Transformer:* The parameter configuration for the Transformer consists of two components. The first is Positional Embedding, which includes two hyperparameters: a maximum sequence length of 5000 and a dropout rate of 0.1. The second component is the Transformer encoder, characterized by one layer, a dropout rate of 0.1, four heads in the multihead attention mechanism, and an output dimension of hidden_size = 256.

B. Comparison With SOTA DRL Methods

For our evaluation, we allocate 1 000 000 training steps $T_{\max}$ for each training run, with the initial 10 000 steps dedicated to random exploration. Within each training run, we execute ten test episodes every 5000 training steps. The average return from these test episodes is then calculated, resulting in 200 data points that are used to plot the episode return curve, which is presented in the supplementary material. Consequently, in Algorithm 1, the value of M is set to 990 000 steps. As for the model architecture in this section, we choose to implement our method using a 2-D CNN because they are a classic model for processing long data sequences.

1) Experimental Results: The experimental results are summarized in nomenclature, which focuses on the final return. The table presents the average return over the last 100 000 training steps from five random seed runs, labeled as "Mean" (a key focus in the DRL community), along with the standard deviation of these returns, denoted as "Standard Deviation." This format is consistently applied across all other tables in

TABLE I

PERFORMANCE COMPARISON IN TERMS OF FINAL RETURN (MEAN ± STANDARD DEVIATION) WHEN IMPLEMENTING OUR METHOD USING 2-D CNN. HERE, OUR SW-TD3 IS THE SW-TD3 METHOD THAT INTEGRATES OUR METHOD. THE BOLDED RESULTS HIGHLIGHT THAT OUR METHOD OUTPERFORMS THE BASELINE IN TERMS OF AVERAGE RETURN (MEAN). THIS PRESENTATION FORMAT IS ALSO USED IN THE OTHER TABLES

	Implemented on TD3			
Methods	HalfCheetah	Hopper	Walker2d	Ant
SW-TD3	11157.19 ± 579.82	2530.25 ± 903.42	3469.81 ± 843.43	1382.55 ± 557.53
Our SW-TD3	**11748.12** ± 504.7	2204.44 ± 598.85	2870.47 ± 746.46	**2772.64** ± 1016.62
GA-TD3	10779.69 ± 598.9	2931.36 ± 912.54	4048.12 ± 331.05	1131.86 ± 1192.03
Our GA-TD3	**12118.54** ± 764.3	**3470.03** ± 278.52	**4594.46** ± 418.4	**4214.59** ± 603.65
NSTEP	9182.8 ± 1296.59	3343.63 ± 462.4	4300.66 ± 670.78	3249.67 ± 1281.98
Our NSTEP	**11448.32** ± 1051.59	3240.65 ± 605.11	**4696.2** ± 631.14	**4566.96** ± 468.15
DST	8631.05 ± 1863.75	3367.79 ± 290.23	3860.37 ± 445.52	2994.99 ± 703.24
Our DST	**10980.16** ± 596.0	**3392.54** ± 440.41	**4559.56** ± 436.45	**4480.47** ± 904.59
MC-TD3	10099.82 ± 967.35	3198.9 ± 386.35	4379.88 ± 389.05	3202.43 ± 954.75
Our MC-TD3	**11238.54** ± 698.48	**3298.42** ± 702.85	4303.53 ± 915.62	**4208.08** ± 898.02
TD3	10241.35 ± 1144.21	3334.95 ± 472.08	3893.26 ± 352.39	4104.4 ± 1034.78
Our TD3	**11617.99** ± 505.9	**3346.09** ± 478.49	**4593.03** ± 1127.52	**4932.62** ± 603.29
	Implemented on SAC			
Methods	HalfCheetah	Hopper	Walker2d	Ant
OSQ-SAC	9885.59 ± 2213.52	2656.96 ± 899.24	4455.35 ± 576.53	3839.8 ± 1008.42
Our OSQ-SAC	**11704.34** ± 961.98	**3257.96** ± 543.22	**4872.61** ± 466.67	**3966.72** ± 1463.44
PC-SAC	11492.99 ± 220.38	2868.69 ± 718.15	4521.81 ± 577.62	4045.86 ± 1286.63
Our PC-SAC	**12283.84** ± 784.24	**3105.74** ± 664.68	**4634.76** ± 792.86	**4575.61** ± 1429.86
SAC	11664.49 ± 448.22	2427.12 ± 738.07	4783.88 ± 545.97	4246.29 ± 710.91
Our SAC	**11932.49** ± 608.3	**3198.42** ± 671.82	**5043.01** ± 602.98	**5291.13** ± 352.6

TABLE II

PERFORMANCE CHANGES AS THE NUMBER OF SEEDS INCREASES (MEAN ± STANDARD DEVIATION). HERE, NOS REFERS TO THE NUMBER OF SEEDS. SPECIFICALLY, NOS (5) INDICATES THAT THE NUMBER OF SEEDS IS 5

Methods	NoS (5)	NoS (10)
Hopper (TD3)	3334.95 ± 472.08	3261.53 ± 476.82
Hopper (Our TD3)	**3346.09** ± 478.49	**3420.85** ± 453.0
Hopper (MC)	3198.9 ± 386.35	3035.78 ± 824.1
Hopper (Our MC)	**3298.42** ± 702.85	**3108.37** ± 770.83
Hopper (DST)	3367.79 ± 290.23	3141.87 ± 804.66
Hopper (Our DST)	**3392.54** ± 440.41	**3262.81** ± 431.6
Ant (OSQ)	3839.8 ± 1008.42	3946.21 ± 1244.1
Ant (Our OSQ)	**3966.72** ± 1463.44	**4116.31** ± 1063.83
Walker2d (PC)	4521.81 ± 577.62	4537.14 ± 845.84
Walker2d (Our PC)	**4634.76** ± 792.86	**4743.5** ± 826.59

Section V. From the results, it is evident that, for both the TD3 and SAC-based baseline DRL methods, integrating our approach led to higher average returns in most cases, as indicated by the bolded results. This demonstrates the effectiveness and generality of our method in enhancing existing DRL approaches and its broad applicability to various underlying DRL algorithms.

In addition to the main experimental results presented in nomenclature, we conduct further experiments, as shown in Table II, to explore how the performance of different methods varies with the number of random seeds. This is done to demonstrate that the improvements our method achieves over existing DRL techniques are statistically significant. Specifically, we select instances from nomenclature where our method shows relatively smaller improvements compared to baseline DRL methods and examine how the final return changes as the number of random seeds increases from 5 to 10. When the random seed count is 10, we calculate the average reward and standard deviation from the last 10 000 steps over ten runs. The results clearly indicate that, regardless of whether the random seed count is 5 or 10, our method consistently outperforms the existing DRL methods. This confirms the statistical significance of our performance improvements, ensuring that they are not due to random chance.

2) Analysis of Results: Existing DRL methods typically only use the current state when making action decisions during training and inference. However, by solely relying on the current state, the information available for their action decision-making is severely limited, which results in them only being able to obtain suboptimal policies. In contrast, our approach saves the neighboring state sequence and provides it together with the current state as input to the actor when making action decisions. These additional neighboring states provide the agent with more information about the evolution of the state and a global awareness when making decisions. Furthermore, utilizing CNNs to extract latent features and capture local dependencies between states within the data sequence allows the DRL method to effectively leverage the neighboring state sequence, ultimately resulting in enhanced policies.

C. Validation of Our Method Implemented Using a Transformer

This section evaluates an alternative implementation of our method that utilizes a Transformer model to process neighboring state sequences. Specifically, we assess its impact on the performance of two widely used DRL algorithms, TD3 and SAC, as well as three SOTA DRL algorithms: MC-TD3, NSTEP, and PC-SAC, across four MuJoCo environments. The

TABLE III

Performance Comparison of Our Method Utilizing 2-D CNN and Transformer in Terms of Final Return (Mean ± Standard Deviation). Here, Our Method Is Implemented Using a Transformer Architecture. The Bolded Results Indicate That Our Implementation Outperforms the Alternative Implementation as Well as the Corresponding Baseline DRL Method

Methods	HalfCheetah	Hopper	Walker2d	Ant
TD3	10241.35 $\pm$ 1144.21	3334.95 $\pm$ 472.08	3893.26 $\pm$ 352.39	4104.4 $\pm$ 1034.78
Our TD3 (2D CNN)	**11617.99** $\pm$ 505.9	3346.09 $\pm$ 478.49	4593.03 $\pm$ 1127.52	4932.62 $\pm$ 603.29
Our TD3 (transformer)	11410.14 $\pm$ 620.31	**3408.75** $\pm$ 401.73	**4808.62** $\pm$ 354.18	**5013.64** $\pm$ 475.99
SAC	11664.49 $\pm$ 448.22	2427.12 $\pm$ 738.07	4783.88 $\pm$ 545.97	4246.29 $\pm$ 710.91
Our SAC (2D CNN)	**11932.49** $\pm$ 608.3	3198.42 $\pm$ 671.82	5043.01 $\pm$ 602.98	5291.13 $\pm$ 352.6
Our SAC (transformer)	10412.87 $\pm$ 1191.37	**3203.12** $\pm$ 641.94	**5466.72** $\pm$ 679.76	**5450.14** $\pm$ 422.55
MC-TD3	10099.82 $\pm$ 967.35	3198.9 $\pm$ 386.35	4379.88 $\pm$ 389.05	3202.43 $\pm$ 954.75
Our MC-TD3 (2D CNN)	11238.54 $\pm$ 698.48	**3298.42** $\pm$ 702.85	4303.53 $\pm$ 915.62	4208.08 $\pm$ 898.02
Our MC-TD3 (transformer)	**11607.62** $\pm$ 1374.0	3024.05 $\pm$ 656.46	4358.52 $\pm$ 869.32	**4410.34** $\pm$ 650.73
PC-SAC	11492.99 $\pm$ 220.38	2868.69 $\pm$ 718.15	4521.81 $\pm$ 577.62	4045.86 $\pm$ 1286.63
Our PC-SAC (2D CNN)	12283.84 $\pm$ 784.24	3105.74 $\pm$ 664.68	4634.76 $\pm$ 792.86	4575.61 $\pm$ 1429.86
Our PC-SAC (transformer)	**13058.94** $\pm$ 868.74	**3576.96** $\pm$ 216.15	**4972.54** $\pm$ 769.97	**4771.83** $\pm$ 909.51
NSTEP	9182.8 $\pm$ 1296.59	3343.63 $\pm$ 462.4	4300.66 $\pm$ 670.78	3249.67 $\pm$ 1281.98
Our NSTEP (2D CNN)	**11448.32** $\pm$ 1051.59	3240.65 $\pm$ 605.11	**4696.2** $\pm$ 631.14	4566.96 $\pm$ 468.15
Our NSTEP (transformer)	11208.51 $\pm$ 742.9	**3413.04** $\pm$ 480.1	4632.39 $\pm$ 799.59	**4800.67** $\pm$ 578.36

evaluation focuses on the final return metric, with episode return curves presented in the supplementary material.

1) Experimental Results: Table III presents the experimental results in terms of final return. These results highlight two key findings. **First**, our Transformer-based approach significantly enhances existing DRL algorithms in most cases, as validated across four different DRL algorithms, in terms of both return metrics. This highlights the substantial performance benefits of incorporating neighboring state sequences into existing DRL methods and underscores the generality of our approach. **Second**, the two different implementations of our method—Transformer-based and CNN-based—exhibit distinct performance characteristics. The Transformer-based implementation slightly outperforms the CNN-based implementation in most cases, suggesting that employing more advanced models to process neighboring state sequences is likely to yield better performance. This improvement can be attributed to the advanced models' ability to more effectively capture relevant features from neighboring state sequences for action decision.

2) Analysis of Results: Similarly, existing DRL methods typically provide only the current state to the policy when making action decisions. This limitation impedes their ability to learn state evolution and results in a lack of global awareness, leading to suboptimal policies. Neighboring state sequences contain additional information about state progression and global awareness. Transformers, renowned for their effectiveness in modeling temporal dependencies and relationships between states, excel in managing neighboring state sequences and thus enhance the performance of existing DRL methods. Moreover, the implementation of different models can yield varying performance outcomes. Since Transformers can more effectively capture long-range dependencies and global relationships between states compared to CNNs, they often achieve superior performance in most cases. In addition to the CNN and Transformer evaluated in our experiments, other advanced models can also be seamlessly integrated into our approach. Therefore, when applying our method, users can test different models based on their specific task requirements to identify the optimal implementation of our approach.

D. Verification Under Navigation Tasks

This section evaluates the performance of our method on four challenging navigation tasks, which are more complex than standard MuJoCo tasks. First, we use the TD3 algorithm and the SOTA NSTEP method as baseline DRL algorithms to compare our approach with a SOTA recurrent DRL method, RODRL. RODRL incorporates recurrent neural networks into the policy, allowing it to leverage past state information to enhance learning. Both our method and RODRL are integrated into TD3 and NSTEP. Then, we use DST, OSQ-SAC, PC-SAC, and MC-TD3 as underlying algorithms to further validate that our method can improve existing DRL approaches on these four tasks. Our method is implemented using a 2-D CNN.

1) Experimental Results: The experimental results are shown in Table IV, where "Mean" represents the average return, defined as the mean reward obtained during the learning process. A higher value indicates better performance. The standard deviation represents the variability of the average return across five different random seeds, reflecting the training stability of DRL under different initial conditions. A smaller standard deviation indicates less variation and higher stability. From the results in Table IV, we draw three conclusions. **First, and most importantly**, when TD3 and NSTEP are integrated with our method (Our TD3 and Our NSTEP), they achieve the highest average return, outperforming the baseline and RODRL methods, suggesting that our method learns a better policy. **Second**, when our method is integrated into DST, OSQ-SAC, PC-SAC, and MC-TD3, it results in better returns, improving existing DRL methods. Combined with the results obtained from standard MuJoCo tasks, our method improves existing DRL approaches in the vast majority of cases. For instance, for the TD3 algorithm, our method demonstrates better performance across all eight tasks, with significant improvements observed in seven of them. **Third**, although performance varies across different random seeds, our method exhibits the lowest standard deviation in most cases, indicating smaller performance variation and showcasing superior training stability in complex navigation tasks.

2) Analysis of Results: Compared to the original baseline methods (such as TD3 and NSTEP), which rely solely on

TABLE IV

PERFORMANCE COMPARISON UNDER FOUR NAVIGATION TASKS IN TERMS OF AVERAGE RETURN (MEAN $\pm$ STANDARD DEVIATION). HERE, TD3-RODRL REFERS TO INTEGRATING RODRL INTO THE TD3 METHOD, WITH OTHER METHODS HAVING ANALOGOUS REPRESENTATIONS

Methods	2DNav-v1	2DNav-v2	ReacherPos	AntNavi
NSTEP	-52.987 ± 3.103	-92.926 ± 8.472	-20.463 ± 0.385	-25.732 ± 0.356
NSTEP-RODRL	-147.289 ± 4.662	-337.012 ± 11.923	-30.621 ± 1.585	-23.316 ± 0.292
Our NSTEP	$\mathbf{-17.667} \pm 0.952$	$\mathbf{-21.570} \pm 1.136$	$\mathbf{-16.862} \pm 0.286$	$\mathbf{-18.179} \pm 0.365$
TD3	-31.780 ± 1.606	-82.169 ± 9.264	-19.140 ± 0.216	-27.690 ± 0.408
TD3-RODRL	-105.607 ± 4.569	-235.627 ± 12.940	-19.700 ± 0.147	-23.654 ± 0.249
Our TD3	$\mathbf{-16.276} \pm 0.835$	$\mathbf{-22.088} \pm 1.050$	$\mathbf{-17.349} \pm 0.270$	$\mathbf{-20.423} \pm 0.383$
MC-TD3	-91.471 ± 4.687	-37.929 ± 1.741	-18.808 ± 0.235	-27.542 ± 0.446
Our MC-TD3	$\mathbf{-13.947} \pm 0.971$	$\mathbf{-19.369} \pm 0.601$	$\mathbf{-17.242} \pm 0.314$	$\mathbf{-17.752} \pm 0.338$
DST	-90.576 ± 4.353	-285.399 ± 16.133	-50.755 ± 1.722	20.359 ± 0.225
Our DST	$\mathbf{-65.427} \pm 2.874$	$\mathbf{-146.594} \pm 8.035$	$\mathbf{-17.155} \pm 0.282$	$\mathbf{-17.314} \pm 0.233$
OSQ-SAC	-75.918 ± 3.743	-47.143 ± 4.082	-18.499 ± 0.247	-24.006 ± 0.368
Our OSQ-SAC	$\mathbf{-24.344} \pm 1.566$	$\mathbf{-11.528} \pm 0.561$	$\mathbf{-13.709} \pm 0.230$	$\mathbf{-17.429} \pm 0.267$
PC-SAC	-72.785 ± 4.033	-157.214 ± 8.314	-50.664 ± 1.353	-20.340 ± 0.175
Our PC-SAC	$\mathbf{-53.258} \pm 2.639$	$\mathbf{-54.161} \pm 2.494$	$\mathbf{-19.691} \pm 0.580$	$\mathbf{-16.126} \pm 0.259$

TABLE V

PERFORMANCE COMPARISON UNDER A NOISY ENVIRONMENT (MEAN $\pm$ STANDARD DEVIATION)

Methods	HalfCheetah	Walker2d	Ant	Humanoid
TD3	10208.08 ± 441.28	2636.74 ± 414.08	100.08 ± 64.42	3630.47 ± 469.11
Our TD3	$\mathbf{10890.77} \pm 688.07$	$\mathbf{4580.7} \pm 630.39$	$\mathbf{2936.14} \pm 2175.28$	$\mathbf{5078.38} \pm 408.61$
SAC	11061.16 ± 247.24	3521.61 ± 1287.89	4620.54 ± 514.2	4282.03 ± 496.16
Our SAC	$\mathbf{11292.55} \pm 469.87$	$\mathbf{5128.78} \pm 383.99$	$\mathbf{5597.79} \pm 526.27$	$\mathbf{4698.32} \pm 75.16$

the current state, and the RODRL method, which depends on a few past states, our approach improves policy learning by incorporating sequences of neighboring states. This provides both past and future state information, offering deeper insights into state evolution for better decision-making and improved performance. Specifically, for navigation tasks, relying solely on the current state can only assess the agent's relative distance to obstacles or targets at a given moment. In contrast, using neighboring states allows the agent to understand its relative distance to obstacles or targets over a duration, facilitating better decision-making in more complex tasks.

E. Verification Under Noisy Environment

This section validates our method under noisy tasks. Specifically, we conduct experiments using the TD3 and SAC algorithms as baseline methods in four noisy MuJoCo environments [57]: HalfCheetah-v2, Ant-v2, Humanoid-v2, and Walker2d-v2. Among these, the Humanoid-v2 task has the highest dimensional state and action space, making it the most challenging. Our method is implemented using a 2-D CNN.

The experimental results, shown in Table V, report the final return. As observed, integrating our method (Our TD3 and Our SAC) into baseline approaches results in higher final returns, demonstrating its effectiveness in improving existing DRL methods, even in noisy environments. Similar to noise-free MuJoCo settings, DRL learning in noisy environments benefits from incorporating neighboring state sequences into the policy. This approach helps the agent understand state evolution, ultimately improving policy learning and achieving higher returns.

F. Sensitivity Analysis of the Hyperparameter

This section presents the sensitivity analysis of the hyperparameter, which is the proportion n of future states in the neighboring state sequence. We conduct experiments using the TD3 algorithm on three challenging MuJoCo continuous control tasks: the HalfCheetah-v2, Ant-v2, and Walker2d-v2 environments. Our method is implemented using a 2-D CNN.

1) Experimental Results: The length of the neighboring state sequence is 21, and we set n to 0:21, 2:21, 4:21, 10:21, and 20:21, respectively. When n is 0:21, our method's neighboring state sequence only uses the past states and the current state for action decision-making, and when n is 20:21, our method's neighboring state sequence only uses the future states and the current state. The experimental results are shown in Table VI in terms of the final return. Based on the experimental results, we can draw the following conclusions: **First**, different values of n have varying impacts on the performance of our method, in terms of the final return, indicating that the number of future states in the neighboring state affects the performance of our method. However, the impact of different n on the performance of our method is not very large. **Second**, when n is set to 4, our method achieved the best performance, surpassing other hyperparameter configurations, including those that rely solely on past states (n is 0:21) or future states (n is 20:21). This finding suggests that the neighboring state sequence, rather than exclusively utilizing past or future states, is essential for effective action decision-making. Consequently, we adopted the hyperparameter configuration of n = 4:21 for this work.

2) Analysis of Results: The future state in the neighboring state sequence provides information about potential outcomes from the current state, while the past state offers historical context. If there is insufficient past information, the agent may struggle to learn adequately from past experiences. Similarly, a lack of future states can result in inadequate future experiences. Therefore, the proportion of future states in the neighboring state sequence impacts the performance of our

TABLE VI

Hyperparameter Sensitivity Analysis (Mean ± Standard Deviation)

Methods	HalfCheetah (TD3)	Walker2d (TD3)	Ant (TD3)
$n = 0:21$	11414.5 ± 540.85	4040.12 ± 737.75	3057.554 ± 281.191
$n = 2:21$	10296.06 ± 1900.47	4562.3 ± 601.46	3215.126 ± 439.253
$\mathbf{n = 4:21}$	$\mathbf{11617.99} \pm 505.9$	$\mathbf{4593.03} \pm 1127.52$	$\mathbf{4932.62} \pm 603.29$
$n = 10:21$	11505.38 ± 535.6	4458.04 ± 690.39	3659.696 ± 1038.834
$n = 20:21$	11503.06 ± 595.43	4567.16 ± 732.57	3833.389 ± 367.833

TABLE VII

Comparing the Length of Different Neighboring State Sequences Under MuJoCo Tasks (Mean ± Standard Deviation)

Methods	HalfCheetah (TD3)	Walker2d (TD3)	Ant (TD3)
$n = 2:10$	11414.5 ± 540.85	4040.12 ± 737.75	4472.61525 ± 645.589
$n = 8:42$	10296.06 ± 1900.47	4562.3 ± 601.46	3634.360 ± 1984.482
$\mathbf{n = 4:21}$	$\mathbf{11617.99} \pm 505.9$	$\mathbf{4593.03} \pm 1127.52$	$\mathbf{4932.62} \pm 603.29$

TABLE VIII

Comparison of the Proportions of Future States Across Three Navigation Tasks, Using a Fixed-Length Neighboring State Sequence (Mean ± Standard Deviation)

Methods	2DNav-v1 (TD3)	2DNav-v2 (TD3)	AntNavi (TD3)
TD3	-31.780 ± 1.606	-82.169 ± 9.264	-27.690 ± 0.408
$n = 2:42$	-26.386 ± 1.511	-79.523 ± 5.636	-29.367 ± 0.579
$n = 8:42$	-17.287 ± 0.996	-32.199 ± 3.645	-25.091 ± 0.398
$n = 10:42$	-19.711 ± 1.056	-38.062 ± 4.494	-26.391 ± 0.401
$\mathbf{n = 4:21}$	$\mathbf{-16.276} \pm 0.835$	$\mathbf{-22.088} \pm 1.050$	$\mathbf{-20.423} \pm 0.383$

method, with n = 4:21 identified as the optimal configuration based on our experiments.

G. Comparing the Different Lengths of Neighboring State Sequences

This section analyzes the impact of varying lengths of neighboring state sequences on the performance of our method. We continue to conduct experiments using the TD3 algorithm across three standard MuJoCo tasks: HalfCheetah-v2, Ant-v2, and Walker2d-v2, as well as three navigation tasks: 2DNav-v1, 2DNav-v2, and AntNavi. Our method is implemented using a 2-D CNN. First, we fix the proportion n = 4:21 of future states in the neighboring state sequence and compare the impact of different neighboring state sequence lengths on performance. We use two baseline methods: one with n = 2:10, which employs neighboring state sequences of length 10, and another with n = 8:42, which uses neighboring state sequences of length 42. Next, we fix the neighboring sequence length at 42 and compare the proportions of future states in the neighboring state sequence. This involves three baselines: n = 2:42, n = 8:42, and n = 10:42.

1) Experimental Results: The experimental results, presented in Tables VII and VIII, are evaluated in terms of the final return. Based on the results, we draw two conclusions. **First**, our method achieves the best performance when using n = 4:21, outperforming both n = 2:10 and n = 8:42. This indicates that the length of the neighboring state sequence significantly affects the performance of our method, with both excessively short and long neighboring state sequences failing to deliver optimal results. **Second**, the n = 8:42 method outperforms both the n = 2:42 and n = 10:42 methods, suggesting that the n = 4:21 ratio is the optimal proportion of future states with varying neighboring state sequence lengths. In addition, n = 4:21 outperforms n = 8:42, achieving the best performance. This demonstrates that, at the optimal future state ratio, the length of the neighboring state sequence significantly influences the performance of our method, with n = 4:21 showing the best results.

2) Analysis of Results: When the neighboring state sequence is relatively short, it provides limited past and future state information, which may not be sufficient for making more effective decisions. However, this also results in lower computational complexity when processing the sequence. In contrast, when the neighboring state sequence is longer, it contains a substantial amount of past and future state information. However, an excessive number of states is likely to include many worthless pieces of information that negatively impact the model's learning, a phenomenon also encountered in traditional deep learning [58], [59]. Moreover, processing longer neighboring state sequences requires greater computational resources. Therefore, selecting an optimal neighboring state sequence length is crucial, as it ensures the best model performance while minimizing computational complexity.

H. Limitations and Future Directions

Two limitations warrant exploration in future work. **First**, the number of future states used in this study is empirically determined and fixed. Exploring an adaptive mechanism to automatically identify the optimal number of future states for different scenarios could be a valuable direction for future research. In addition, more advanced techniques for processing neighboring state sequences could be further explored. **Second**, our method is validated on simulated motion-restricted tasks (e.g., four navigation tasks) and control tasks in noisy environments, demonstrating its potential for real-world robotic applications. However, real-world scenarios present additional challenges, including partially observable states, delayed actions, and increased sensor noise. To address these, we plan to integrate techniques from the DRL community, such as domain adaptation, to enable effective model transfer from simulation to reality under real-world noise conditions. Furthermore, to compensate for missing real-time information regarding states and actions due to partially observable states and delayed actions, we incorporate multiple past and future states, along with their corresponding actions, into a sequence of neighboring states, adaptively adjusting the sequence length and the number of future states to accommodate varying environmental conditions.

VI. Conclusion and Limitation

This work proposes a novel perspective for implementing a policy in DRL, referred to as the neighboring state-aware policy, for action decision-making, which can be seamlessly integrated into existing DRL methods. This policy uses the neighboring state sequence, along with the current state, as input to the actor to decide an action. This approach enables DRL's decision-making to utilize not only the information provided by the current state but also the potentially learned global awareness from the neighboring states. Through extensive experiments, our method significantly enhances ten representative DRL methods in terms of return and other metrics. In the future, we plan to explore adaptive tuning methods for hyperparameters, better techniques for processing neighboring state sequences, and apply our approach to real-world robotic tasks.

References

[1] Y. Li, Y. Xing, X. Lan, X. Li, H. Chen, and D. Jiang, "AlignMamba: Enhancing multimodal mamba with local and global cross-modal alignment," 2024, *arXiv:2412.00833*.

[2] Y. Li, X. Lan, H. Chen, K. Lu, and D. Jiang, "Multimodal PEAR chain-of-thought reasoning for multimodal sentiment analysis," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 20, no. 9, pp. 1–23, Sep. 2024.

[3] Y. Li, W. Gan, K. Lu, D. Jiang, and R. Jain, "AVES: An audio-visual emotion stream dataset for temporal emotion detection," *IEEE Trans. Affect. Comput.*, vol. 16, no. 1, pp. 438–450, Jan. 2025.

[4] B. R. Kiran et al., "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.

[5] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: A comprehensive survey," *Artif. Intell. Rev.*, vol. 55, no. 2, pp. 945–990, Feb. 2022.

[6] M. Xu and J. Wang, "Deep reinforcement learning for parameter tuning of robot visual servoing," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 2, pp. 1–27, Apr. 2023.

[7] K. Lin, Y. Li, Q. Liu, D. Li, X. Shi, and S. Chen, "Almost surely safe exploration and exploitation for deep reinforcement learning with state safety estimation," *Inf. Sci.*, vol. 662, Mar. 2024, Art. no. 120261.

[8] M. Xu, X. Chen, Z. Wen, W. Fu, and J. Wang, "A two-stage selective experience replay for double-actor deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 22, 2025, doi: 10.1109/TNNLS.2025.3557930.

[9] M. Simchowitz and A. Slivkins, "Exploration and incentives in reinforcement learning," *Oper. Res.*, vol. 72, no. 3, pp. 983–998, May 2024.

[10] J.-C. Pomerol, "Artificial intelligence and human decision making," *Eur. J. Oper. Res.*, vol. 99, no. 1, pp. 3–25, Jan. 1997.

[11] D. Hardman, *Judgment and Decision Making: Psychological Perspectives*, vol. 11. Hoboken, NJ, USA: Wiley, 2009.

[12] H. Shteingart and Y. Loewenstein, "Reinforcement learning and human behavior," *Current Opinion Neurobiol.*, vol. 25, pp. 93–98, Apr. 2014.

[13] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[14] J. Cheng, K. Li, J. Lin, and P. Pachuca, "Neighboring state-based RL exploration," 2022, *arXiv:2212.10712*.

[15] J. Lyu, Y. Yang, J. Yan, and X. Li, "Value activation for bias alleviation: Generalized-activated deep double deterministic policy gradients," *Neurocomputing*, vol. 518, pp. 70–81, Jan. 2023.

[16] B. Saglam, F. B. Mutlu, D. C. Cicek, and S. S. Kozat, "Parameter-free reduction of the estimation bias in deep reinforcement learning for deterministic policy gradients," *Neural Process. Lett.*, vol. 56, no. 2, pp. 1–25, Mar. 2024.

[17] H. Yin, C. Wang, C. Yan, X. Xiang, B. Cai, and C. Wei, "Deep reinforcement learning with multicritic TD3 for decentralized multirobot path planning," *IEEE Trans. Cogn. Develop. Syst.*, vol. 16, no. 4, pp. 1233–1247, Aug. 2024.

[18] M. Xu, X. Chen, Y. She, Y. Jin, and J. Wang, "Time-varying weights in multi-reward architecture for deep reinforcement learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 8, no. 2, pp. 1865–1881, Apr. 2024.

[19] M. Xu, Y. She, Y. Jin, and J. Wang, "Dynamic weights and prior reward in policy fusion for compound agent learning," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, pp. 1–28, Dec. 2023.

[20] M. Xu, X. Chen, and J. Wang, "Policy correction and state-conditioned action evaluation for few-shot lifelong deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 36, no. 4, pp. 6843–6857, Apr. 2025.

[21] F. Munguia-Galeano, A.-H. Tan, and Z. Ji, "Deep reinforcement learning with explicit context representation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 36, no. 1, pp. 419–432, Jan. 2025.

[22] B. Saglam, D. C. Çiçek, F. B. Mutlu, and S. Kozat, "Mitigating off-policy bias in actor-critic methods with one-step Q-learning: A novel correction approach," *Trans. Mach. Learn. Res.*, vol. 1, pp. 1–27, Aug. 2024.

[23] H. Eghbal-zadeh, F. Henkel, and G. Widmer, "Learning to infer unseen contexts in causal contextual reinforcement learning," in *Proc. Self-Supervision Reinforcement Learn. (SSL-RL) Workshop*, Mar. 2021, pp. 1–11.

[24] C. Benjamins et al., "Contextualize me—The case for context in reinforcement learning," 2022, *arXiv:2202.04500*.

[25] S. Sodhani, A. Zhang, and J. Pineau, "Multi-task reinforcement learning with context-based representations," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 9767–9779.

[26] T. Eimer, A. Biedenkapp, F. Hutter, and M. Lindauer, "Self-paced context evaluation for contextual reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, Jan. 2021, pp. 2948–2958.

[27] C. Benjamins et al., "CARL: A benchmark for contextual and adaptive reinforcement learning," 2021, *arXiv:2110.02102*.

[28] J. Liu, L. He, Y. Kang, Z. Zhuang, D. Wang, and H. Xu, "CEIL: Generalized contextual imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2023, pp. 75491–75516.

[29] J. Lee et al., "Supervised pretraining can learn in-context reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2023, pp. 43057–43083.

[30] C. X. Lu et al., "Structured state space models for in-context reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2023, pp. 1–16.

[31] B. Chen et al., "Context-aware safe reinforcement learning for non-stationary environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 10689–10695.

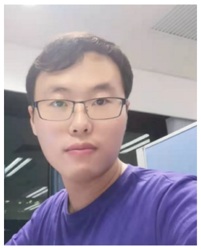
[32] S. Rezaei-Shoshtari, C. Morissette, F. R. Hogan, G. Dudek, and D. Meger, "Hypernetworks for zero-shot transfer in reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, Jan. 2022, pp. 9579–9587.

[33] J. Jiang, L. Yan, X. Yu, and Y. Guan, "Contextual policy transfer in meta-reinforcement learning via active learning," in *Proc. Int. Conf. Web Inf. Syst. Appl.*, Jan. 2022, pp. 354–365.

[34] R. Lin et al., "Contextual transformer for offline meta reinforcement learning," 2022, *arXiv:2211.08016*.

[35] X. Guo, S. Chang, M. Yu, G. Tesauro, and M. Campbell, "Hybrid reinforcement learning with expert state sequences," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 3739–3746.

[36] J. Zhong, R. Wu, and J. Si, "A long N-step surrogate stage reward for deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–13.
[37] Y. Tan, P. Hu, L. Pan, J. Huang, and L. Huang, "RLx2: Training a sparse deep reinforcement learning model from scratch," 2022, *arXiv:2205.15043*.
[38] X. Li, D. Wang, M. Zhao, and J. Qiao, "Reinforcement learning control with N-step information for wastewater treatment systems," *Eng. Appl. Artif. Intell.*, vol. 133, Jul. 2024, Art. no. 108033.
[39] P. Malekzadeh and K. N. Plataniotis, "Combining information-seeking exploration and reward maximization: Unified inference on continuous state and action spaces under partial observability," 2022, *arXiv:2212.07946*.
[40] E. Muškardin, M. Tappler, B. K. Aichernig, and I. Pill, "Reinforcement learning under partial observability guided by learned environment models," in *Proc. Int. Conf. Integr. Formal Methods*, Jan. 2022, pp. 257–276.
[41] S. Xie, Z. Zhang, H. Yu, and X. Luo, "Recurrent prediction model for partially observable MDPs," *Inf. Sci.*, vol. 620, pp. 125–141, Jan. 2023.
[42] X. Zhao, D. Zhang, L. Han, T. Zhang, and B. Xu, "ODE-based recurrent model-free reinforcement learning for POMDPs," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2023, pp. 1–17.
[43] L. Meng, R. Gorbet, and D. Kulic, "Memory-based deep reinforcement learning for POMDPs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 5619–5626.
[44] Q. Shen, Y. Li, H. Jiang, Z. Wang, and T. Zhao, "Deep reinforcement learning with robust and smooth policy," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 8707–8718.
[45] A. Wachi and Y. Sui, "Safe reinforcement learning in constrained Markov decision processes," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 9797–9806.
[46] Z. Yang and H. Nguyen, "Recurrent off-policy baselines for memory-based continuous control," 2021, *arXiv:2110.12628*.
[47] S. Nayak et al., "Scalable multi-agent reinforcement learning through intelligent information aggregation," in *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 25817–25833.
[48] H. Mao et al., "Neighborhood cognition consistent multi-agent reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 5, pp. 7219–7226.
[49] M. Xu, X. Chen, and J. Wang, "A novel topology adaptation strategy for dynamic sparse training in deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 11, 2024, doi: 10.1109/TNNLS.2024.3446996.
[50] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," 2023, *arXiv:2312.00752*.
[51] H. Xiong, T. Xu, L. Zhao, Y. Liang, and W. Zhang, "Deterministic policy gradient: Convergence analysis," in *Proc. Uncertainty Artif. Intell.*, 2022, pp. 2159–2169.
[52] Y. Ran, Y.-C. Li, F. Zhang, Z. Zhang, and Y. Yu, "Policy regularization with dataset constraint for offline reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, Jan. 2023, pp. 28701–28717.
[53] Q. He, T. Zhou, M. Fang, and S. Maghsudi, "Adaptive regularization of representation rank as an implicit constraint of Bellman equation," 2024, *arXiv:2404.12754*.
[54] W. Zhou, Y. Li, Y. Yang, H. Wang, and T. M. Hospedales, "Online meta-critic learning for off-policy actor-critic methods," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2020, pp. 17662–17673.
[55] M. Zhang et al., "Efficient reinforcement learning with the novel N-step method and V-network," *IEEE Trans. Cybern.*, vol. 54, no. 10, pp. 6048–6057, Oct. 2024.
[56] Z. Wang, C. Chen, and D. Dong, "A Dirichlet process mixture of robust task models for scalable lifelong reinforcement learning," *IEEE Trans. Cybern.*, vol. 53, no. 12, pp. 7509–7520, Dec. 2023.
[57] B. Grooten et al., "Automatic noise filtering with dynamic sparse training in deep reinforcement learning," 2023, *arXiv:2302.06548*.
[58] G. Sokar, Z. Atashgahi, M. Pechenizkiy, and D. C. Mocanu, "Where to pay attention in sparse training for feature selection?," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2022, pp. 1627–1642.
[59] K. Liu, Z. Atashgahi, G. Sokar, M. Pechenizkiy, and D. C. Mocanu, "Supervised feature selection via ensemble gradient information from sparse neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2024, pp. 3952–3960.



Meng Xu received the B.Sc. and M.S. degrees in software engineering and computer science from Northwestern Polytechnical University, Xi'an, China, in 2017 and 2020, respectively, and the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, Hong Kong, in 2024.

He is currently a Post-Doctoral Researcher with the City University of Hong Kong. His research interests include reinforcement learning and autonomous driving.

Xinhong Chen received the B.Eng. degree in software engineering from Sun Yat-sen University, Guangdong, China, in 2018, and the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2022.

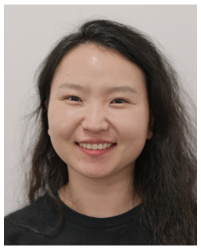
He is currently a Post-Doctoral with the Department of Computer Science, City University of Hong Kong. His research interests include natural language processing, sentiment analysis, autonomous driving, and causality mining.

Guanyi Zhao received the B.Sc. degree in nuclear physics from Jilin University, Changchun, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong, Hong Kong.

His research interests include autonomous driving and deep learning.

Zihao Wen received the B.Eng. degree in computer science from Zhejiang University, Hangzhou, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong, Hong Kong, China.

His research interests include trajectory prediction and motion planning of autonomous driving.



Weiwei Fu received the B.Sc. degree in communication engineering from Zhengzhou University, Zhengzhou, China, in 2018, and the M.Sc. degree in computer science from Chinese University of Hong Kong, Hong Kong, in 2019. She is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong, Hong Kong.

Her research interests include autonomous driving, software testing, and sensor attack and defense.

Jianping Wang (Fellow, IEEE) received the B.S. and M.S. degrees in computer science from Nankai University, Tianjin, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from The University of Texas at Dallas, Richardson, TX, USA, in 2003.

She is currently a Chair Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. Her research interests include security, autonomous driving, cloud computing, edge computing, and machine learning.