International Journal of Control, Automation, and Systems 23(1) (2025) 1-40
http://dx.doi.org/10.1007/s12555-024-0990-1
ISSN:1598-6446 eISSN:2005-4092
http://www.springer.com/12555

Reinforcement Learning for Process Control: Review and Benchmark Problems

Joonsoo Park , Hyein Jung , Jong Woo Kim* , and Jong Min Lee*

Abstract: The success of reinforcement learning (RL) combined with deep neural networks has led to the development of numerous RL algorithms that have demonstrated remarkable performance across various domains. However, despite its potential in process control, relatively limited research has explored RL in this field. This paper aims to bridge the gap between RL and process control, providing potential applications and insights for process control engineers. In this review, we first summarize previous efforts to apply RL to process control. Next, we provide an overview of RL concepts and categorize recent RL algorithms, analyzing the strengths and weaknesses of each category. We implement fourteen RL algorithms and apply them to six relevant benchmark environments, conducting quantitative analyses to identify the most suitable approaches for specific process control problems. Finally, we draw conclusions and outline future research directions to advance RL's application in process control.

Keywords: Approximate dynamic programming, optimal control, process control, reinforcement learning.

1. INTRODUCTION

In recent years, the integration of reinforcement learning (RL) into process control has captured significant attention from both academia and industry. Process control, a critical function in sectors such as chemical engineering and biomanufacturing, involves the regulation of dynamic systems to achieve desired outputs. While traditional control methods are robust and well-established, they often face challenges when dealing with complex, nonlinear, and uncertain environments. RL, a branch of machine learning focused on optimizing decision-making through trial-and-error interactions with the environment [1], offers a promising alternative. RL has demonstrated notable success across various domains, including board games [2,3], computer games [4,5], robotics [6,7], autonomous vehicles [8,9], healthcare [10], and drug design [11]. By enabling systems to learn and adapt in real time, RL has the potential to revolutionize process control, enhancing efficiency, flexibility, and performance beyond capabilities of conventional methods. This review explores the growing body of research on the application of RL to process control, highlighting key developments, challenges, and future directions.

RL distinguishes itself as a powerful and versatile approach for process control due to its ability to learn optimal control policies through interactions with dynamic environments. Unlike traditional control methods that often rely on predefined models, RL adapts to the complexities and uncertainties inherent in process control systems, enabling it to handle nonlinearities, time delays, and disturbances in real time. Additionally, RL's model-free nature allows it to operate effectively even when an accurate system model is unavailable, making it particularly advantageous in industrial applications where process dynamics are difficult to characterize. Through continuous performance improvements via trial-and-error, RL has the potential to outperform classical controllers, offering more robust solutions to the demands of modern process industries. Consequently, RL holds significant promise to advance process control.

Several review papers have examined the application of RL within the domain of process systems engineering (PSE), providing valuable insights into its potential and associated challenges. Shin *et al*. [12] conducted a comprehensive analysis comparing RL with model predictive control (MPC) for process control and mathematical programming (MP) for multi-scale decision-making problems, highlighting RL's unique capabilities in managing complex system dynamics. Spielberg *et al*. [13] introduced foundational RL concepts and proposed a model-free deep RL controller, emphasizing its applicability in

Manuscript received October 31, 2024; accepted November 11, 2024. Recommended by Editor-in-Chief Hyo-Sun Ahn. The Institute of Engineering Research at Seoul National University provided research facilities for this work. The source code of this paper is provided in http://github.com/skjw1224/epelRL.

Joonsoo Park, Hyein Jung, and Jong Min Lee are with the School of Chemical and Biological Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea (e-mails: {jsphero, 976hannah, jongmin}@snu.ac.kr). Jong Woo Kim is with the Department of Energy and Chemical Engineering, Innovation Center for Chemical Engineering, Incheon National University, 119, Academy-ro, Yeonsu-gu, Incheon 22012, Korea (e-mail: jong.w.kim@inu.ac.kr).
* Corresponding authors.

Table 1. List of the papers that applied RL to process control.

Description	References
Early work before DNNs	[18], [19], [20], [21], [22], [23], [24], [25], [26], [27]
Direct application	[28], [29], [30], [31], [32], [33]
Experimental work	[34], [35], [36], [37], [38], [39]
Integration of conventional control methods	[40], [41], [42], [43], [44], [45]
Safety constraints	[46], [47], [48], [49], [50], [43], [51], [52], [53], [54]
Offline pre-training	[55], [56], [57], [58], [59], [60]
Batch processes	[61], [62], [63]

the design of adaptive control strategies. Nian *et al*. [14] provided a broad overview of various RL applications, demonstrating RL's versatility in addressing a wide range of challenges within the PSE domain. Yoo *et al*. [15] focused on batch process control, illustrating RL's advantages over traditional model-based control methods while also discussing the practical issues encountered during implementation. Oh [16] explored the comparison between RL and data-driven MPC, highlighting their applications in chemical and biological processes. Additionally, Lawrence *et al*. [17] reviewed the integration of machine learning techniques, including RL, into industrial sensing and control, highlighting their growing importance in enhancing system performance and decision-making processes. Collectively, these reviews emphasize the potential of RL in the PSE domain while also identifying areas that require further research and development.

This review paper makes several contributions to process control through the application of RL methodologies. We broadly survey RL methods, offering a detailed examination of recent advancements in both model-free and model-based approaches. We compare the strengths and limitations of fourteen RL methods to process control and optimization applications. Furthermore, we feature six benchmark problems for PSE domain using open-source tools. These benchmarks offer a reproducible framework for evaluating RL algorithms, thereby supporting ongoing research and development for process control engineers.

The remainder of this paper is structured as follows: Section 2 provides an overview of the fundamental concepts of RL. Section 3 categorizes existing algorithms based on key criteria. Section 4 details the specific RL algorithms employed in our experiments, while Section 5 describes the benchmark environments used. In Section 6, we present and analyze the experimental results. Section 7 discusses potential future research directions for the application of RL in process control. Finally, Section 8 offers concluding remarks.

1.1. Applications of RL to process control

In this section, we present a review of the existing literature that applies RL techniques to process control, categorizing key contributions according to the specific challenges and types of processes involved. These works illustrate how RL can address challenges such as system uncertainty and nonlinearity, providing alternatives to traditional control methods. The relevant studies are summarized in Table 1.

Several studies have applied RL methods to process control, particularly in the early stages before deep neural networks became prevalent. Hoskins and Himmelblau [18] implemented RL in a continuous stirred tank reactor (CSTR) using an adaptive heuristic critic algorithm, approximating the performance of a PID controller but requiring extensive experimentation. Wilson and Martinez [19] used a combination of fuzzy modeling and RL for batch process automation, while Anderson *et al*. [20] applied RL to tune a PID controller in a simulated heating coil. Martinez [21] and Ahamed *et al*. [22] tackled batch process optimization and power system control, respectively by applying RL methods. However, early RL methods were constrained by the curse of dimensionality, struggling to scale in high-dimensional spaces. To overcome this, researchers began exploring more computationally feasible alternatives, such as approximate dynamic programming (ADP). Several studies utilized ADP for process control [23-25], and Lee and Lee [64] extended ADP to dual adaptive control. Nosair *et al*. [26] and Yang and Lee [27] later demonstrated robustness guarantees in dynamic optimization when using RL methods.

Recent advancements in RL have shown that direct application of various RL algorithms can significantly improve control performance. Spielberg *et al*. [28] applied the basic actor-critic algorithm to both single-input single-output (SISO) and multiple-input multiple-output (MIMO) linear systems, demonstrating its versatility across diverse control scenarios. Oh *et al*. [29] employed the double deep Q-Network (double DQN) in a simulated moving bed (SMB) system, achieving strong performance. Bao *et al*. [30] introduced a variant of deep deterministic policy gradient (DDPG), known as the deep deterministic actor-critic predictor (DDACP), which enhances learning by decomposing the value function into immediate and future rewards. Joshi *et al*. [31,32] proposed the twin actor TD3 (TATD3) and twin actor SAC (TASAC) algorithms, which integrate multiple actors into the twin delayed deep deterministic policy gradient (TD3) and soft actor critic (SAC) frameworks, respectively, uti-

lizing an ensemble learning strategy. Deng *et al*. [33] extended the proximal policy optimization (PPO) algorithm by incorporating multiple actors, effectively applying it to strip rolling control in the steel industry without relying on domain-specific knowledge.

Most research on applying RL in process control has been simulation-based, with only a few studies conducting experimental validation to confirm the performance of RL algorithms in real-world settings. For instance, Pandian and Noel [34] and Dogru *et al*. [35] demonstrated the efficacy of RL-based controllers in water level control experiments involving a quadruple-tank system and a hybrid three-tank system, respectively. Degrave *et al*. [36] provided further experimental evidence by showing that a trained RL controller could effectively shape and maintain high-temperature plasma within a tokamak vessel. Lawrence *et al*. [37] and Dogru *et al*. [38] investigated RL-based PID controller tuning, with experimental results confirming their effectiveness. Although Zhu *et al*. [39] did not conduct physical experiments, they validated their proposed RL algorithm through extensive simulations of the vinyl acetate monomer (VAM) process, demonstrating its capability to control a complex chemical plant with multiple units.

In process control, conventional model-based methods, such as MPC, remain prevalent; however, recent research has increasingly focused on integrating these techniques with RL to enhance control performance. Kim *et al*. [40,41] proposed an algorithm to approximate the solution of the Hamilton-Jacobi-Bellman (HJB) equation for optimal control, offering a novel approach that blends RL with traditional control theory. Furthermore, Kim *et al*. [42,43] developed algorithms utilizing differential dynamic programming (DDP) to address two-stage optimal control and constrained dynamic optimization problems. Alhazmi *et al*. [44] introduced an RL-based economic MPC (EMPC) algorithm that effectively combines EMPC and RL, while Oh *et al*. [45] proposed a Q-learning-based MPC (Q-MPC), where the terminal cost is approximated using Double DQN, creating a synergistic integration of RL and MPC.

In practical applications of RL for process control, it is essential to consider safety constraints to ensure reliable and secure operations. One common approach to handling constraints in RL involves incorporating a penalty term into the reward function for constraint violations. Yoo *et al*. [46] advanced this approach by introducing a dynamic penalty strategy to address issues such as slow convergence and entrapment in local minima. Tang *et al*. [47] later validated the effectiveness of dynamic penalty-based RL in the temperature control of a zinc oxide rotary volatile kiln. Beyond penalty-based methods, Pan *et al*. [48] and Petsagkourakis *et al*. [49] applied constraint-tightening techniques to develop the oracle-assisted constrained Q-learning and chance-constrained policy optimization (CCPO) algorithms, respectively, both of which ensure that operational constraints are satisfied with high probability. Mowbray *et al*. [50] further contributed to this field by proposing an RL algorithm that employs a Gaussian process to account for model-plant mismatches, thereby satisfying constraints with high probability. Kim *et al*. [43] and Jallet *et al*. [51] considered an exact method to handle constraints in the RL framework by introducing primal-dual DDP. Additionally, Kim and Lee [52] introduced an RL algorithm based on policy iteration, which guarantees asymptotic stability by constraining the value function to a control Lyapunov function and updating the policy using a variant of Sontag's formula. Subsequent developments in safe RL have included algorithms that consider state and input constraints through the use of barrier functions [53], as well as those that address model-plant mismatches and stochastic disturbances by leveraging Gaussian processes [54].

To mitigate the limitations of RL, particularly the extensive exploration required—which can be both risky and costly in the process control domain—offline pre-training has been explored as a means to enhance data efficiency and reduce the risks associated with real-world exploration. Offline training can be conducted using either existing datasets or data generated through various techniques, followed by online updates to refine the models. Hwangbo and Sin [55] proposed a method that employs Monte Carlo sampling to generate historical data, which is then used to train DQN agents. Mowbray *et al*. [56] and McClement *et al*. [57] utilized apprenticeship learning and meta-RL, respectively, performing offline pre-training to increase data efficiency. Another approach involves generating historical data through inverse model control [58] and offset-free MPC [59] to pre-train RL agents offline, followed by online training to further improve data efficiency and reduce exploration risks. Additionally, Alhazmi and Sarathy [60] developed a technique that applies offline RL algorithms by leveraging operation logs generated from EMPC, further enhancing the safety and efficiency of RL in process control applications.

Batch processes present unique challenges due to their unsteady-state operation, significant stochasticity, and highly nonlinear dynamics. These complexities render conventional model-based control methods difficult to apply, prompting the exploration of RL approaches for process control in batch processes. Petsagkourakis *et al*. [61] and Byun *et al*. [62] introduced the policy gradient method to tackle batch-to-batch optimization problems, offering a novel approach to improving performance across successive batches. Additionally, Yoo *et al*. [63] demonstrated the effectiveness of combining DDPG with Monte Carlo learning method, further enhancing control by segmenting the phases of the batch process to address its inherent complexities.

2. BASIC STRUCTURE OF RL

2.1. Definition

RL is a type of machine learning focused on sequential decision-making, where an agent learns to make decisions by interacting with an environment to achieve specific goals [1]. The agent receives feedback through a reward signal which evaluates the benefit of its actions. This reward signal can be delayed and incomplete, meaning the agent might not receive immediate feedback for its actions, and the feedback it does receive may not fully capture the long-term consequences of those actions. The objective, therefore, is to maximize the expected cumulative reward over time, rather than just immediate rewards. To achieve this, a key challenge in RL is balancing exploration and exploitation: the agent must explore different actions to discover their effects and potential rewards while also exploiting known actions that yield high rewards to optimize its performance. This balance ensures the agent continuously improves its decision-making strategy by learning from both its successes and failures.

2.2. Markov decision process

RL problems are typically formulated using a Markov decision process (MDP) [65], which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ represents a set of states, $\mathcal{A}$ represents a set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ represents the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor. The state space represents all possible situations the agent can encounter, while the action space includes all possible actions the agent can take. The reward function provides feedback on the immediate benefit of actions given states, and the transition probability describes the likelihood of moving from one state to another given a particular action. The discount factor determines the importance of future rewards. MDPs rely on the Markov property, which states that the future is independent of the past given the present state, meaning the next state depends only on the current state and action. In contrast, a partially observable MDP (POMDP) accounts for situations where the agent cannot directly observe the entire state of the environment, receiving only limited information instead.

RL algorithms often involve one or more of these components: policy, value function, and model. A policy $\pi(a|s)$ is the behavior function of an agent, mapping states to actions. It can be stochastic or deterministic. A value function is a long-term prediction of future rewards and can be used to evaluate whether given states are good or bad. Therefore, it often provides criteria for the agent on what actions to take. Specifically, there is a state-value function $V(s)$ that represents the value of a given state s and an action-value function $Q(s,a)$ that represents the value of taking a certain action a in a given state s. Lastly, the model predicts the next state and reward given the current state and action. However, model-free algorithms do not require a model, while in model-based algorithms, even when a model is provided, it may be imperfect.

To maximize cumulative reward, the primary objective of RL, it is essential to learn a policy that optimizes the expected return from each state. The expected return can be expressed through the state-value function and action-value function of a MDP, as defined in (1) and (2).

$$V^{\pi}(s) = \mathbb{E}_{\pi(\cdot|s)}\left[G_t | s_t = s\right], \tag{1}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi(\cdot|s)}\left[G_t | s_t = s, a_t = a\right]. \tag{2}$$

Here $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ represents the discounted return. Equations (1) and (2) can be decomposed and rewritten as (3) and (4) respectively, known as the Bellman expectation equation [66].

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[V^{\pi}(s')\right]\right], \tag{3}$$

$$Q^{\pi}(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a), a'\sim\pi(\cdot|s')}\left[Q^{\pi}(s',a')\right]. \tag{4}$$

The state-value function and action-value function that satisfy $V^*(s) = \max_{\pi} V^{\pi}(s)$ and $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$ are defined as the optimal value functions. Finding the optimal value function is crucial for solving the MDPs. For the optimal value function, the Bellman equation can be formulated as (5) and (6), known as the Bellman optimality equation [66].

$$V^*(s) = \max_{a}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[V^*(s')\right]\right], \tag{5}$$

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[\max_{a'} Q^{\pi}(s',a')\right]. \tag{6}$$

Solving the Bellman optimality equation is essential to determine the optimal policy. The Bellman optimality equation, however, is nonlinear, so in general, there is no closed-form solution. Therefore, it is mainly solved using iterative methods such as dynamic programming, Monte carlo method, and temporal difference method.

2.3. Dynamic programming

Dynamic programming (DP) is a method used to solve complex problems by breaking them down into simpler subproblems, and it is effective when a perfect model of the MDP is available, as it utilizes the full knowledge of the system's dynamics [1,67]. Policy iteration (PI) and value iteration (VI) are two fundamental algorithms in the field of DP used to solve MDPs. Given a model that represents the environment, these algorithms try to find an optimal policy that maximizes the expected cumulative reward in an iterative manner.

The PI algorithm alternates between policy evaluation and policy improvement. Given a policy π_k and a value function $V_k^{\pi}(s)$, policy evaluation step calculates the value function $V_{k+1}^{\pi}(s)$ for all states s using the Bellman expectation equation. With the value function $V_{k+1}^{\pi}(s)$ computed

in the policy evaluation step, the policy improvement step updates the policy π_k to π_{k+1} by choosing the greedy action which maximizes the value function. The PI algorithm can be represented formally as follows:

$$V^{\pi}_{k+1}(s) = \mathbb{E}_{a\sim\pi_k(\cdot|s)}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[V^{\pi}_{k}(s')\right]\right], \tag{7}$$

$$\pi_{k+1}(a|s) = \arg\max_{a}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[V^{\pi}_{k+1}(s')\right]\right]. \tag{8}$$

These steps are repeated iteratively: evaluate the current policy, then improve the policy based on the value function, until the policy converges to the optimal policy π^*.

The VI algorithm is more direct method that focuses on updating the value function to converge to the optimal value function $V^*(s)$. Instead of evaluating a specific policy π as in the PI algorithm, the VI algorithm updates the value function using the Bellman optimality equation. For each state s, the value function is updated as follows:

$$V_{k+1}(s) = \max_{a}\left[r(s,a) + \gamma \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\left[V_{k}(s')\right]\right]. \tag{9}$$

This step is repeated until the value function converges, and once the value function has converged to the optimal value function $V^*(s)$, the optimal policy π^* can be extracted as shown in (10).

$$\pi^*(a|s) = \arg\max_{a}\left[r(s,a) + \gamma V^*(s')\right]. \tag{10}$$

A critical distinction between PI and VI lies in their convergence properties and computational efficiency. PI involves relatively fewer iterations because it updates the policy only when the value function for the current policy has been evaluated, so it converges faster than VI. On the other hand, VI typically involves more iterations than PI, but each iteration is computationally cheaper since it only updates the value function.

While DP methods yield exact solutions, they have a few limitations when applied in practice. First, DP requires an accurate model representing the transition dynamics and reward function of the environment, which is often unavailable in real-world applications. Furthermore, DP suffers from high computational costs, as entire state and action spaces have to be explored. This computational cost increases exponentially with the dimensionality of the state and action spaces, a phenomenon known as the curse of dimensionality. To address these issues, model-free and approximate methods should be employed.

2.4. Approximate dynamic programming

Approximate dynamic programming (ADP) is a set of methods that addresses the limitations of conventional DP [68,69]. ADP leverages sampled experiences to estimate the value function, thereby eliminating the need for an explicit system model. When the dimensions of the state and action spaces are large or continuous, making an exact solution of the Bellman equations intractable, ADP iteratively approximates a value function and a policy instead of calculating the exact value for every state and action as in DP.

2.4.1 Monte Carlo methods

Monte Carlo (MC) methods estimate the expected rewards by the average of the sampled returns from multiple episodes. The MC-based policy evaluation process can be represented as follows:

$$V^{\pi}_{k+1}(s_t) \leftarrow V^{\pi}_{k}(s_t) + \alpha\left[G_t - V^{\pi}_{k}(s_t)\right], \tag{11}$$

where α represents the learning rate. Unlike DP, MC methods do not utilize bootstrapping, making them usable even in non-Markovian environments. However, MC methods can only be applied to episodic MDPs because they update the value function using the return value as the target after the completion of a full episode. Consequently, they cannot be directly applied to infinite horizon problems. Additionally, the value function estimate is unbiased but has a high variance.

2.4.2 Temporal difference methods

Temporal difference (TD) methods learn directly from experience samples like MC methods, but employ bootstrapping for the value function estimate. Here, bootstrapping refers to the method that updates the current value estimate with the previous value estimate. Therefore, there is no need to wait for the end of an episode, enabling online learning. The simplest one-step TD update is represented as follows:

$$V^{\pi}_{k+1}(s) \leftarrow V^{\pi}_{k}(s) + \alpha\left[r(s,a) + \gamma V^{\pi}_{k}(s') - V^{\pi}_{k}(s)\right]. \tag{12}$$

Here, the term in brackets denotes the temporal difference between the target and current values, referred to as the TD error or Bellman error.

SARSA and Q-learning are two classic TD methods that differ in how they update the value function during policy evaluation. SARSA [70] is an on-policy TD method where the agent learns the value function using state-action-reward-next state-next action tuples (s,a,r,s',a'). It updates Q-values based on the actions a and a' both chosen from the current policy. Thus, the policy evaluation step of SARSA can be represented as

$$Q^{\pi}_{k+1}(s,a) \leftarrow Q^{\pi}_{k}(s,a) + \alpha\left[r(s,a) + \gamma Q^{\pi}_{k}(s',a') - Q^{\pi}_{k}(s,a)\right]. \tag{13}$$

In contrast, Q-learning [71,72] is an off-policy TD method where the agent learns the optimal policy with behavior and target policies that can differ. Q-learning updates Q-values by selecting the action with the maximum

Q-value in the next state, regardless of the action taken by the current policy. The policy evaluation step of Q-learning can be represented as

$$Q^{\pi}_{k+1}(s,a) \leftarrow Q^{\pi}_{k}(s,a) + \alpha\big[r(s,a) + \gamma \max_{a'} Q^{\pi}_{k}(s',a') - Q^{\pi}_{k}(s,a)\big]. \quad (14)$$

DP methods are characterized by low variance but high bias, whereas MC methods exhibit low bias at the expense of high variance. Therefore, TD-based RL algorithms aim to leverage the favorable trade-off between bias and variance by adjusting the prediction horizon or eligibility parameters. However, like other methods, TD approaches encounter challenges in large state spaces, necessitating the implementation of function approximation techniques.

2.5. Value function approximation

The curse of dimensionality makes it computationally prohibitive to store and update the value functions for every possible state-action pair in large-scale problems. To address this challenge, value function approximation techniques are employed. The objective of value function approximation is to minimize the mean squared value error between the true value and the approximate value functions. However, the true value is not known a priori, so a target value, such as the return G_t in MC methods or the TD target $r(s,a)+\gamma V(s')$ in TD learning, is used instead.

2.5.1 Linear approximation

The simplest function approximation approach is a linear method, where a feature vector $\phi \in \mathbb{R}^f$ is expressed as a linear function of a weight vector $w \in \mathbb{R}^f$, with ϕ serving as the basis function, as shown in (15). The basis function ϕ can take various forms, including polynomials, radial basis functions (RBFs), etc.

$$V_w(s) = w^T \phi(s) = \sum_{i=1}^{f} w_i \phi_i(s). \quad (15)$$

When using linear approximators, the least squares method can be employed [73,74]. This method minimizes the sum of squared differences between predicted and actual returns by directly computing the TD fixed point, as given by (16)-(18) [1].

$$A = \mathbb{E}\big[\phi(s)(\phi(s) - \gamma\phi(s'))^T\big], \quad (16)$$

$$b = \mathbb{E}\big[r(s,a)\phi(s)\big], \quad (17)$$

$$w = A^{-1}b. \quad (18)$$

This method uses the entire data and updates it in one step, allowing for efficient data usage and low variance. Additionally, it performs batch updates, which is advantageous when training with fixed data offline. However, the linear approximations may not be suitable for more complex, non-linear problems. Representative RL algorithms using the least squares method include Least Squares Temporal Difference and Least Squares Policy Iteration.

Least squares temporal difference (LSTD) [75,76] computes the weight vector of the linearly approximated value function as follows:

$$\hat{A}_t = \frac{1}{t}\sum_{k=0}^{t-1} \phi(s_k)(\phi(s_k) - \gamma\phi(s_{k+1}))^T + \epsilon I, \quad (19)$$

$$\hat{b}_t = \frac{1}{t}\sum_{k=0}^{t-1} r_k \phi(s_k), \quad (20)$$

$$w_t = \hat{A}_t^{-1}\hat{b}_t. \quad (21)$$

Here, ϵI is the identity matrix with small scalar $\epsilon > 0$ ensuring that $\hat{A}_t$ is invertible. Compared to traditional TD methods, LSTD has the advantage of providing more accurate and stable value function estimates by solving a least squares problem. However, LSTD can be computationally intensive, particularly in high-dimensional state spaces, due to the necessity of inverting matrices.

Least squares policy iteration (LSPI) [77] combines LSTD with PI to achieve stable and rapid policy improvement. It alternates between policy evaluation and policy improvement, following the PI method. During the policy evaluation step, it uses the LSTD method to estimate the action-value function for the current policy. In the policy improvement step, it updates the policy by selecting actions that maximize the estimated action-value function. By combining LSTD and PI, LSPI provides a data-efficient and computationally feasible approach to finding optimal policies.

2.5.2 Nonlinear approximation

Nonlinear function approximators, neural networks (NNs) or artificial neural networks (ANNs) can be utilized. An NN takes a hidden state z, computed with a linear weight and bias matrix as shown in (22), as input to an activation function g, which is a nonlinear function.

$$V_w(s) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(s),$$
$$f^{(l)}(z) = g(W^{(l)}z + b^{(l)}). \quad (22)$$

These networks can be layered to simulate nonlinear characteristics. Representative activation functions include the logistic sigmoid, hyperbolic tangent (Tanh), and rectified linear unit (ReLU). Various other activation functions are detailed in Apicella *et al*. [78] and Dubey *et al*. [79].

Recently, RL algorithms that utilize deep neural networks (DNNs) as function approximators have demonstrated strong performance on complex and high-dimensional systems [4,80]. This approach, known as deep reinforcement learning (DRL), has been successfully applied to various fields such as robotics [81], gaming [5], and drug design [11]. Beyond the basic form of neural networks (NNs), RL can be extended to more complex

systems by employing modified forms of NNs, such as recurrent neural networks (RNNs), which are specialized for time series data, and convolutional neural networks (CNNs), which excel in pattern recognition in the field of vision. Additionally, various RL algorithms leveraging attention-based transformer architectures have been introduced recently [82,83].

Stochastic gradient descent (SGD) is a key optimization technique in RL used to update the parameters of the value function approximator based on the observed data [84]. SGD updates the approximator's parameters incrementally using gradients calculated from a single sample or a small batch of sampled experiences, rather than the entire dataset, making it computationally efficient. This approach helps in navigating the vast and often noisy state-action spaces, enabling the agent to iteratively improve its value function by minimizing the prediction error, which measures the discrepancy between the predicted and actual values. Examples of SGD methods include AdaGrad [85], AdaDelta [86], and Adam [87].

2.5.3 Nonparametric approximation

The performance of previous function approximators is highly dependent on hyperparameters such as the number of parameters and network structure. Therefore, it could be considered to use nonparametric approximators instead [88]. Common nonparametric function approximators include methods such as Gaussian processes [89] and kernel regression [90]. Although these methods perform well for certain systems, they encounter a significant issue: their computational cost escalates rapidly as the complexity and dimensionality of the system increase.

2.6. Policy gradient

The methods introduced so far are value-based algorithms that learn an action-value or Q-function and select actions based on the highest value from this learned Q-function. Alternatively, we can consider a parameterized policy π_ϕ to learn the optimal policy directly, thereby deciding the action without relying on the value function. A policy-based algorithm utilizing a parameterized policy offers several advantages. First, it allows for efficient exploration by learning stochastic policies. Second, while value-based methods become computationally expensive as the dimensionality of the action space increases, policy-based methods can be applied to high-dimensional or continuous action spaces. Finally, policy-based methods could be preferable in environments where prior knowledge can be directly incorporated into the policy, and in some cases, it is easier to learn the policy directly than to learn the value functions.

To determine the optimal policy, it is essential to identify a parameter ϕ that maximizes the objective function, as illustrated in (23).

$$\begin{aligned} J(\phi) &= \sum_{s\in\mathcal{S}} d^\pi(s) V^\pi(s) \\ &= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\phi(a|s) Q^\pi(s,a), \end{aligned} \tag{23}$$

where $d^\pi(s)$ represents the stationary distribution of states for policy π. The gradient of the objective function $J(\phi)$ with respect to ϕ can be obtained by policy gradient theorem [91], as demonstrated in (24).

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\phi}\left[Q^\pi(s,a)\nabla_\phi \ln \pi_\phi(a|s)\right]. \tag{24}$$

Several methods for expressing the return in (24) have been proposed, allowing for the management of bias and variance trade-offs depending on the form of return employed [92]. The most representative and classic policy gradient algorithms are REINFORCE and the Actor-Critic algorithm.

The REINFORCE algorithm [93], a foundational approach among the policy gradient methods, leverages MC estimates of episodic returns to update the policy. The process begins by executing the policy to generate a trajectory, followed by calculating the return G_t by summing the discounted reward for each time step. Policy parameters are updated based on (24) but with the unbiased return. However, a significant issue with the naive REINFORCE algorithm is its high variance in gradient estimates, which can result in unstable training and slow convergence. To mitigate this issue, a baseline b_t can be introduced to reduce variance, though this introduces some bias. This baseline, often represented by the value function, subtracts an estimate of the expected return from the actual return, thereby stabilizing updates and enhancing learning efficiency.

The actor-critic algorithm [94,95] synergistically combines the advantages of value-based and policy-based methods. This algorithm comprises two primary components: the actor and the critic. The actor, representing the policy $\pi_\phi(a|s)$, selects actions based on the current state, thereby directly optimizing the policy by adjusting its parameters ϕ using policy gradient methods. Conversely, the critic evaluates the action by estimating the value function, either as the state-value $V_w^\pi(s)$ or the action-value $Q_\theta^\pi(s,a)$, and updates its parameters to minimize the temporal difference error. The actor-critic framework effectively addresses the limitations of purely value-based methods, which struggle with policy optimization, and purely policy-based methods, which can suffer from high variance in gradient estimates. This dual architecture enables efficient learning by leveraging the critic's value function to provide a lower-variance estimate of the policy gradient for the actor, thereby enhancing the stability and convergence properties of the learning process. This method is particularly advantageous for problems with

continuous action spaces and has demonstrated significant efficacy in various complex RL tasks.

3. OVERVIEW OF RL ALGORITHMS

RL algorithms can be broadly categorized into two types: model-free and model-based methods. Model-free RL focuses on directly learning a policy $\pi(a_t|s_t)$ which maps states to actions without relying on an explicit model of the environment. This approach simplifies the learning process but typically requires a larger amount of data. In contrast, model-based RL involves constructing a model $p(s_{t+1}|s_t,a_t)$ to predict the next state given the current state and action. This model is subsequently used to derive a policy. When the probability of a trajectory τ is defined by (25), model-free methods update policy using samples without employing a model, while model-based methods leverage the model for planning.

$$p(\tau) = p(s_0)\prod_{t=0}^{T}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t). \tag{25}$$

Model-free methods tend to perform better in complex environments where an accurate model is difficult to obtain but can be less sample-efficient. Conversely, model-based methods are generally more sample-efficient but are sensitive to model inaccuracies. In this chapter, we will explore both model-free and model-based methods, examining their underlying principles, advantages, and limitations. The categories and corresponding algorithms for each method are given in Table 2.

3.1. Model-free methods

Interacting with the environment, particularly when the transition dynamics are unknown, model-free methods often employ TD learning to estimate value functions, which guide the policy toward optimality. Despite their simplicity and ability to generalize in unknown or highly stochastic environments, these methods typically suffer from low sample efficiency, requiring large amounts of data and frequent interactions with the environment. Moreover, their performance depends heavily on well-designed exploration strategies. However, advances such as the use of DNNs, experience replay, and advanced optimization techniques, alongside hardware accelerators like GPUs, have significantly improved their data efficiency and exploration capabilities [4,104]. These developments have sparked increased research, leading to notable improvements in the performance of model-free algorithms.

In this section, we categorize model-free RL methods into three main groups: value-based methods, policy-based methods, and inference-based methods. Value-based methods aim to find the optimal policy by learning a value function without directly parameterizing the policy. Policy-based methods, in contrast, directly optimize policy parameters through the use of policy gradients. Finally, inference-based methods reformulate RL as a probabilistic inference problem. Each category represents a distinct approach, and the principles and characteristics of the algorithms in each will be thoroughly examined.

3.1.1 Value-based methods

Value-based methods focus on estimating the value of each state-action pair to derive a policy that maximizes the expected cumulative reward without direct parameterization. The most classic algorithms in this category include least-squares temporal difference (LSTD) [75,76] and least-squares policy iteration (LSPI) [77], both of which utilize the linear approximator introduced earlier. However, algorithms employing linear approximators often suffer from poor generalization performance when handling high-dimensional input.

Deep Q-networks (DQN) [4] utilize deep neural networks to estimate the Q-function and have demonstrated strong performance in environments with high-dimensional state spaces. Subsequent variants of DQN, such as Double DQN [96] and Dueling DQN [97], have shown improved performance over the original algorithm.

In the C51 algorithm [100], distributional RL was introduced to approximate the value distribution instead of the expected value used in conventional RL algorithms. Since then, various distributional RL algorithms, such as quantile regression DQN (QR-DQN) [101], implicit quantile networks (IQN) [102], and fully-parameterized quantile function (FQF) [103], have been proposed to address the limitations of C51.

Moreover, DQN has evolved into various forms, such as recurrent replay distributed DQN (R2D2) [98], which utilizes RNNs as an approximator, and Rainbow [99], which achieved state-of-the-art performance on the benchmark suite of 57 Atari 2600 games [160] by integrating multiple RL methods such as double Q-learning, dueling networks, distributional RL, noisy nets, n-step learning [1] and prioritized experience replay (PER) [161].

Value-based methods have the advantage of simplicity and a strong theoretical foundation, as they estimate only the Q-function without learning the policy separately. However, they are limited to discrete action spaces and require a separate exploration strategy, such as epsilon-greedy.

3.1.2 Policy-based methods

The policy gradient theorem [91] can be used to directly optimize and train a parameterized policy for continuous action spaces. The most representative methods are REINFORCE [93], which updates the policy episodically based on MC returns, and actor-critic [94,95], which updates the policy and value function, respectively.

In addition to the classic policy gradient methods, several advanced algorithms have been developed to further

Table 2. Categories of RL algorithms introduced in Section 3, with implemented algorithms written in **bold**.

Model	Method	Sub-category	Algorithms
Model-free	Value-based methods	Linear approximator	LSTD [75,76], LSPI [77]
		DNN approximator	**DQN** [4], Double-DQN [96], Dueling-DQN [97], R2D2 [98], Rainbow [99]
		Distributional RL	C51 [100], **QR-DQN** [101], IQN [102], FQF [103]
	Policy-based methods	Naive policy gradient	REINFORCE [93], **A2C** [104], A3C [104], ACER [105], IMPALA [106]
		Natural policy gradient	NPG [107], NAC [108], NES [109], **TRPO** [110], **PPO** [111], ACKTR [112]
		Deterministic policy gradient	DPG [113], **DDPG** [80], **TD3** [114], D4PG [115], MADDPG [116]
	Inference-based methods	EM for policy search	RWR [117], **PoWER** [118,119], VIP [120], MPO [121], **REPS** [122], **PI**^{2} [123], GPS [124-126]
		Max Ent RL	Z-learning [127], AICO [128], G-learning [129], PCL [130], PGQL [131], Soft Q-learning [132], SVPG [133], **SAC** [134,135], DIAYN [136]
Model-based	Analytic gradient computation	Optimal control	LQR [137], DDP [138,139], **iLQR** [140], **SDDP** [141], PDDP [142,143], PD-DDP [43]
		Policy search	PILCO [144], GPS [81], SVG [145], **GDHP** [40]
	Sampling-based methods	MPC	RS [146], Mb-Mf [147], PETS [148], PlaNet [149]
		MCTS	ExIt [150], FBTS [151]
	Model-based data generation	Dyna-style	Dyna [152,153], MB-TRPO [154], MB-MPO [155], SLBO [156], MBPO [157]
		Value function	MVE [158], STEVE [159]

enhance the effectiveness and efficiency of RL. Asynchronous advantage actor-critic (A3C) [104] utilizes multiple parallel workers to independently explore the environment and update the central model asynchronously, achieving faster and more stable learning. Actor critic with experience replay (ACER) [105] and importance weighted actor learner architecture (IMPALA) [106] use the same parallel training structure as A3C but devise off-policy algorithms that utilize a replay buffer to achieve higher sample efficiency.

3.1.2.1 Natural policy gradient methods

One limitation of the policy gradient method is that it may not consistently follow the steepest ascent direction in the policy space, as it does not account for the underlying geometry of the policy space, which can lead to large, poorly directed updates. To address this issue, the natural policy gradient (NPG) method [107] employs the concept of the natural gradient [162], which adjusts the policy gradient by scaling it with the inverse of the Fisher information matrix. The modified policy gradient is expressed as shown in (26).

$$\tilde{\nabla}_\phi J(\phi) = F(\phi)^{-1} \nabla_\phi J(\phi), \tag{26}$$

where $F = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\phi}\left[\nabla_\phi \log \pi_\phi \nabla_\phi \log \pi_\phi^T\right]$. This adjustment aligns gradient steps with the true geometry of the policy space, leading to more efficient and stable updates, and ultimately resulting in faster convergence and improved performance [163].

Building on this foundation, several key algorithms

have emerged, each refining and extending the natural policy gradient approach. The natural actor-critic (NAC) algorithm [108] integrates natural policy gradients with an actor-critic framework to enhance efficiency. The natural evolution strategy (NES) [109] is proposed as an efficient black-box optimization method that leverages natural gradients for complex parameter spaces. Trust region policy optimization (TRPO) [110] employs natural policy gradients to optimize a surrogate objective function while constraining the policy update within a trust region defined by the KL divergence. Proximal policy optimization (PPO) [111] and actor-critic using Kronecker-factored trust region (ACKTR) [112] further refine TRPO by incorporating a clipped objective function and a Kronecker-factored approximation of the natural gradient, respectively, thus improving efficiency.

3.1.2.2 Deterministic policy gradient methods

The deterministic policy gradient (DPG) [113] builds on the policy gradient theorem by directly optimizing policies that deterministically map states to actions. Unlike stochastic policies that generate a distribution over actions, deterministic policies select a specific action for each state, leading to more sample-efficient learning. The deterministic policy gradient theorem [113] provides the foundation for this approach, stating that the gradient of the expected return with respect to the policy parameters ϕ can be expressed as follows:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\phi}\left[\nabla_\phi \pi_\phi(s) \nabla_a Q^\pi(s,a)\right]. \tag{27}$$

This formulation indicates that the gradient of the policy performance can be computed by the expected value of the product of the gradient of the policy and the gradient of the Q-function, facilitating efficient optimization in continuous action spaces.

A prominent algorithm employing the deterministic policy gradient is deep deterministic policy gradient (DDPG) [80], which uses DNNs to approximate policies and value functions. Subsequent algorithms have built upon DDPG with various improvements: Twin delayed DDPG (TD3) [114] mitigates overestimation bias in Q-learning, distributed distributional DDPG (D4PG) [115] uses distributional value estimates for better performance, and multi-agent DDPG (MADDPG) [116] extends DDPG to multi-agent environments, facilitating interaction and cooperation between multiple agents. These advancements demonstrate the continuous evolution and refinement of deterministic policy gradient methods.

3.1.3 Inference-based methods

The "RL as inference" approach distinguishes itself from conventional RL by framing the RL problem as a probabilistic inference task rather than maximizing the expected reward [164]. In this framework, the goal is to infer a distribution over optimal behaviors or policies that are most likely to achieve high rewards, based on the observed data. This probabilistic perspective provides a flexible and principled approach to RL, incorporating uncertainty and variability in the decision-making process. Reinterpreting RL as inference opens new avenues for policy search, particularly through the application of the expectation-maximization (EM) algorithm [165] and the principle of maximum entropy, both of which offer robust methods for optimizing policies within this probabilistic context.

3.1.3.1 Expectation-maximization for policy search

Unlike the conventional RL formulation, the inference-based formulation incorporates a prior distribution over trajectories and aims to estimate the posterior distribution over trajectories given the observed reward. In this framework, the reward signal is treated as evidence, while the policy is regarded as a distribution over actions that needs to be optimized. Formally, the likelihood of a trajectory τ under a policy $\pi(a|s)$ is given by (25). By modeling the probability of the reward as $p(O=1|\tau) \propto \exp(R(\tau))$, where O is a binary variable and $O=1$ denotes optimality, the objective is to maximize the marginal likelihood $p_\pi(O=1)$. This approach leads to the evidence lower bound (ELBO), expressed in (28), which can be optimized using the EM algorithm [165].

$$\log p_\pi(O=1) \geq \mathbb{E}_{q(\tau)}\left[R(\tau)\right] - D_{KL}\left(q(\tau)\|p_\pi(\tau)\right). \tag{28}$$

The E-step updates the variational distribution $q(\tau)$ to approximate the posterior, and the M-step updates the policy π to maximize the expected reward under $q(\tau)$ [121].

Several RL algorithms implement the EM algorithm within the "RL as inference" framework. Reward-weighted regression (RWR) [117] and policy learning by weighting exploration with the returns (PoWER) [118,119] update the policy through weighted regression, where the weights are exponential functions of cumulative rewards. Variational inference for policy search (VIP) [120] utilizes variational inference to approximate the posterior distribution over policy parameters. Maximum a posteriori policy optimization (MPO) [121] formulates policy optimization as a maximum a posteriori (MAP) estimation problem, integrating prior knowledge with observed data. These methods illustrate the flexibility of the EM algorithm in optimizing policies within the probabilistic inference framework of RL.

Similarly, algorithms such as relative entropy policy search (REPS) [122], policy improvement with path integrals (PI^2) [123], and guided policy search (GPS) [124-126,166] employ EM-like approaches. REPS balances expected returns with a relative entropy constraint to maintain policy stability, while PI^2 integrates path integral control methods into the policy improvement process. GPS alternates between optimizing trajectories for high reward and fitting a global policy to these trajectories. Although

these algorithms do not directly implement the EM algorithm, they incorporate EM-like procedures to achieve effective policy optimization.

3.1.3.2 Maximum entropy reinforcement learning

Maximum entropy reinforcement learning (MaxEnt RL) is a framework that extends conventional RL by incorporating an entropy maximization term into the objective function, as shown in (29).

$$J(\phi) = \sum_t \mathbb{E}_{s_t \sim d^\pi, a_t \sim \pi_\phi} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]. \quad (29)$$

This approach aims to maximize not only the expected return but also the entropy of the policy, which promotes more diverse and exploratory behaviors. The inherent stochasticity introduced by the entropy term facilitates efficient exploration, thereby preventing premature convergence to suboptimal policies and enabling the discovery of multiple high-reward strategies. This characteristic makes MaxEnt RL particularly effective in complex environments where exploration and robustness are crucial, enhancing the policy's adaptability in dynamic and uncertain scenarios.

MaxEnt RL can also be interpreted through the lens of the "RL as inference" framework. Levine [164] provided an analysis of MaxEnt RL from a probabilistic inference perspective, demonstrating how the entropy term can be seen as a form of regularization that promotes exploration. The robustness of MaxEnt RL approaches, particularly in diverse and challenging environments, was further emphasized by Eysenbach and Levine [167], who highlighted its ability to discover a wide range of effective strategies.

The MaxEnt RL framework has found applications across various domains. For optimal control, algorithms such as Z-learning [127] and approximate inference control (AICO) [128] have been developed, along with inverse RL algorithms that employ the maximum entropy objective [168,169]. Additionally, the maximum entropy objective has been applied in algorithms like G-learning [129], path consistency learning (PCL) [130] and policy gradient Q-learning (PGQL) [131]. However, these approaches predominantly address low-dimensional state and action spaces and utilize simple parametric representations, which may not scale effectively to complex environments and continuous action spaces.

To address these limitations, soft Q-learning [132] introduced a soft Bellman update to optimize the maximum entropy objective, incorporating a function approximator and Stein variational gradient descent (SVGD) [170] to extend the approach to continuous state and action spaces. Schulman *et al*. [171] demonstrated the equivalence between soft Q-learning and policy gradient methods, further solidifying the connection between MaxEnt RL and traidtional RL techniques. The Stein variational policy gradient (SVPG) [133] introduced a MaxEnt RL algorithm that integrates SVGD with policy gradient, similar to soft Q-learning. Subsequently, the soft actor-critic (SAC) [134,135] extended soft Q-learning to an actor-critic framework, showcasing robust performance across various tasks. Furthermore, the diversity is all you need (DIAYN) [136] illustrated the feasibility of learning diverse skills using a MaxEnt RL structure without relying on a reward function, demonstrating the flexibility and effectiveness of this approach.

3.2. Model-based methods

Model-based RL (MBRL) is an approach wherein a model of the environment is constructed to forecast future states and rewards, utilizing this model for planning and decision-making [1]. One of the primary advantages of model-based RL is its high data efficiency, as the agent can use the model to simulate numerous scenarios, thereby reducing the necessity for extensive real-world interactions. This is particularly valuable in contexts where interactions are risky, costly, or time-consuming. However, model-based RL also presents notable challenges. Achieving high model accuracy is difficult, especially in complex systems where the dynamics are intricate and hard to capture precisely. Inaccuracies in the model can lead to compounding errors during the planning phase, where small prediction errors accumulate over time, resulting in suboptimal decision-making. Additionally, model-based approaches often struggle with scalability, as the computational resources needed to maintain and update an accurate model can increase significantly with the complexity of the environment. These limitations highlight the trade-offs inherent in model-based RL, where the benefits of data efficiency must be weighed against the practical challenges of ensuring model fidelity and managing computational demands.

System identification [172] is crucial in MBRL as it involves constructing a model that accurately captures the dynamics of the environment. Broadly, the process of model construction can be divided into two approaches. The first is fundamental modeling [173,174], which utilizes known governing equations derived from physical laws to define the system's behavior. This approach is effective when the system's dynamics are well understood and can be represented by analytical expressions. The second approach is empirical modeling, where parameters are estimated from data through techniques like regression, maximum likelihood estimation, or Bayesian inference. While empirical methods are easier to implement, they may struggle with extrapolation beyond observed data. Linear models such as ARX, ARMAX, ARIMA, and N4SID can handle simple, linear systems but are inadequate for complex, nonlinear dynamics. In such cases, more flexible data-driven methods, such as Gaussian processes [175] or neural networks [176], allow the system to learn dynamics directly from high-dimensional data, mak-

ing method selection critical for MBRL performance and reliability.

3.2.1 Analytic gradient computation

Optimal control seeks to determine control inputs that minimize a cost function over a system's trajectory, with the linear quadratic regulator (LQR) [137] serving as a classical example for linear systems. Differential dynamic programming (DDP) [138,139] extends this approach to nonlinear systems by iteratively solving quadratic approximations of the value function, utilizing analytic gradients to optimize control policies more efficiently. This gradient-based optimization is central to DDP and its variants, such as iterative LQR (iLQR) [140], which simplifies DDP, stochastic DDP (SDDP) [141] and parameterized DDP (PDDP) [142,143], which handle uncertainties. Additionally, Primal-Dual DDP [43] incorporates a modified augmented Lagrangian framework, enabling the effective handling of nonlinear constraints.

Several MBRL algorithms have been developed that similarly leverage the analytic gradient of the model to enhance policy learning. Probabilistic inference for learning control (PILCO) [144] employs Gaussian processes for model construction, enabling efficient learning by incorporating uncertainty in model predictions. Guided policy search (GPS) [81] utilizes trajectory optimization [140] to generate guiding trajectories, which serve as references for training a global controller, effectively integrating local optimization with global policy learning. Stochastic value gradient (SVG) [145] adapts the value gradient method [177], initially designed for deterministic scenarios, to accommodate stochastic models and policies, thereby enabling gradient propagation through the model and improving policy updates. Finally, globalized dual heuristic programming (GDHP) [40] addresses the HJB equation by leveraging the gradient of the costate function, offering a principled approach to solve optimal control problems in a more global context.

3.2.2 Sampling-based planning

In nonlinear dynamical systems, computing the analytic gradient is challenging, making MBRL algorithms that utilize sampling-based planning particularly effective. One of the simplest approaches in this category is random shooting (RS) [146], where candidate action sequences are sampled from a fixed distribution, evaluated using a learned model, and the sequence with the highest predicted return is selected. To ensure robust decision-making, these algorithms often employ MPC [178-180], where only the first action of the chosen sequence is executed, and the process is repeated at each time step. An example of an algorithm utilizing RS is the model-based and model-free (Mb-Mf) algorithm [147]. More sophisticated sampling methods, such as the cross-entropy method (CEM) [181], further enhance planning efficiency and performance. Notable algorithms that leverage CEM include probabilistic ensembles with trajectory sampling (PETS) [148] and deep planning network (PlaNet) [149], both of which have demonstrated strong results in MBRL tasks.

Monte Carlo tree search (MCTS) [182,183] is a widely utilized heuristic search algorithm in discrete action environments, known for its ability to balance exploration and exploitation by simulating random action sequences to estimate the value of potential moves. A notable application of MCTS is its integration into AlphaGo [2,3], where it played a crucial role in achieving superhuman performance in the game of Go by efficiently navigating the vast search space of possible moves. Building on the foundational principles of MCTS, several MBRL algorithms, such as expert iteration (ExIt) [150] and feedback-based tree search (FBTS) [151], have been proposed to further advance this approach.

3.2.3 Model-based data generation

The Dyna algorithm [152,153] integrates both planning and learning by interleaving real-world interactions with simulated experiences generated from a learned model. Specifically, Dyna uses real-world experiences to update both the value function or policy and a model, which in turn is utilized to generate simulated experiences. These simulated experiences are then employed to further refine the policy or value function, effectively enhancing learning efficiency. By combining direct RL update with model-based planning, Dyna not only accelerates convergence towards optimal policies but also serves as the foundation for a variety of other MBRL algorithms that build upon its core principles.

Building upon the Dyna-style algorithm, several advanced MBRL approaches have emerged, each incorporating unique strategies to optimize policy performance using the model prediction. Model-based trust region policy optimization (MB-TRPO) [154] integrates an ensemble of learned models with the trust region policy optimization (TRPO) [110] algorithm, effectively balancing exploration and exploitation to refine policy updates. On the other hand, model-based meta-policy optimization (MB-MPO) [155] leverages meta-learning techniques to adaptively optimize policies, enhancing the agent's ability to generalize across different tasks. Stochastic lower bounds optimization (SLBO) [156] introduces a novel approach by focusing on optimizing the lower bounds of the expected return, incorporating a regularization term that accounts for the uncertainty in model predictions, thus ensuring more reliable policy improvements. Lastly, model-based policy optimization (MBPO) [157] also utilizes an ensemble of models but distinguishes itself by employing short-horizon model predictions rather than long-horizon ones, mitigating the potential inaccuracies that can accumulate over extended time steps. Additionally, model-based value expansion (MVE) [158] and stochastic en-

semble value expansion (STEVE) [159] utilize learned models to improve the target value estimates for TD learning, rather than generating synthetic data, thereby enhancing the accuracy of value function updates.

4. ALGORITHMS

In this paper, we implemented fourteen RL algorithms, selected from the algorithms highlighted in bold in Table 2. In this section, we briefly introduce each algorithm, outlining its fundamental concepts and the corresponding training process, including the specific loss functions used to train the agent. For a more detailed explanation of each algorithm, readers are referred to the original papers cited.

4.1. Deep Q-network

Deep Q-network (DQN) [4] is an extension of the conventional Q-learning algorithm to continuous state space using DNNs as a Q-function approximator. However, nonlinear function approximators such as a neural network, are considered to have low learning stability. Therefore, experience replay and target networks are introduced to improve the learning stability. Experience replay is a technique where past experience tuples (s,a,r,s') are stored in a replay buffer $\mathcal{D}$ and randomly sampled during training to break temporal correlations and improve the learning stability. A target network is a separate neural network used to estimate the target Q-values during training, which are updated less frequently than the primary Q-network to provide more stable targets for learning. The loss function for updating Q-functions in DQN is as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r+\gamma\max_{a'}\hat{Q}_{\bar{\theta}}(s',a')-Q_{\theta}(s,a)\right)^2\right], \tag{30}$$

where $\mathcal{D}$ is a uniform distribution from experience replay, and $\hat{Q}_{\bar{\theta}}$ is the target network with fixed parameters during Q-network update. The DQN agent chooses actions with ϵ-greedy policy to deal with the exploration-exploitation dilemma.

DQN algorithm has demonstrated its superiority over traditional RL algorithms in the context of Atari games, achieving performance levels comparable to those of human players. Moreover, it exhibits robustness across diverse environments, thus establishing its versatility in the realm of RL applications. DQN's flexibility and effectiveness have sparked new developments, giving rise to different approaches aimed at making it even better. These variations, including double DQN [96], dueling DQN [97], among others, have contributed to its reputation for achieving proficient control performance in environments characterized by continuous state spaces and discrete action spaces.

4.2. Quantile regression deep Q-network

Quantile regression deep Q-network (QR-DQN) [101] is a type of distributional RL algorithm proposed to bridge the gap between theory and practice, addressing limitations of the previously introduced C51 [100] algorithm. To manage the stochastic nature of the environment, rather than estimating expected returns through value function approximation, distributional RL learns the value distribution by treating rewards as random variables. This process involves defining the distributional Bellman operator, denoted as (31), corresponding to the conventional RL's Bellman operator, and iteratively progressing through approximate distribution learning.

$$\mathcal{T}^{\pi}Z(s,a) \overset{D}{=} r(s,a)+\gamma Z(s',a'), \quad s'\sim\mathcal{P}(\cdot|s,a),\ a'\sim\pi(\cdot|s), \tag{31}$$

where $Z(s,a)$ represents the random return whose expectation is the value $V(s)$, and $P_1 \overset{D}{=} P_2$ denotes for the equal probability distribution. Previous study have demonstrated that minimizing the Wasserstein distance between the probability distribution Z and its Bellman update $T^{\pi}Z$ results in the optimal value distribution [100]. The p-Wasserstein metric is defined as the L_p metric on the inverse cumulative distribution function, as depicted in (32), for distributions U and Y.

$$d_p(U,Y) = \left(\int_0^1 |F_Y^{-1}(\omega)-F_U^{-1}(\omega)|^p d\omega\right)^{1/p}. \tag{32}$$

However, utilizing the Wasserstein metric as a loss function presents challenges as it cannot be directly minimized via stochastic gradient descent.

QR-DQN, therefore, aims to minimize the Wasserstein distance to a target distribution by employing quantile regression. Unlike C51, QR-DQN does not predetermine lower and upper bounds by uniformly fixing probability values; instead, it calculates an adjustable support. Additionally, it utilizes quantile regression loss to minimize the Wasserstein distance through stochastic gradient descent. This approach defines the quantile Huber loss, depicted in (33), as the original quantile regression loss is non-differentiable at zero.

$$\mathcal{L}_{\tau}^{\kappa}(u) = \begin{cases} \dfrac{u^2}{2\kappa}|\tau-\delta_{u<0}|, & \text{if } |u|\le\kappa, \\ (|u|-\dfrac{1}{2}\kappa)|\tau-\delta_{u<0}|, & \text{otherwise.} \end{cases} \tag{33}$$

QR-DQN has been observed to outperform previous algorithms.

4.3. Deep deterministic policy gradient

In continuous action spaces, conventional value-based methods like DQN struggle to find greedy policies due to their discrete-action-focused approach. Deep deterministic policy gradient (DDPG) [80] addresses this challenge

by adopting an actor-critic method capable of handling continuous action domains. DDPG employs a parameterized policy to directly map states to deterministic actions, updating policy parameters through the deterministic policy gradient (DPG) algorithm [113]. However, similar to DQN, the use of nonlinear function approximators in DDPG can lead to convergence issues. To overcome this, DDPG uses a replay buffer, alleviating the assumption of independently and identically distributed samples. Furthermore, DDPG introduces a soft update mechanism for the target network, gradually updating its parameters over time instead of directly transferring them. This soft update is governed by the parameter $\tau \ll 1$, with the soft update equations for the critic and actor networks represented as (34) and (35).

$$\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}, \tag{34}$$

$$\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}. \tag{35}$$

The loss function for updating the critic network is similar to that of DQN, as shown in (30), but with the target Q-value computed using the actor network, as depicted in (36).

$$\begin{aligned}&\mathcal{L}(\theta)\\ &= \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r+\gamma\hat{Q}_{\bar{\theta}}(s',\pi_\phi(s')) - Q_\theta(s,a)\right)^2\right].\end{aligned} \tag{36}$$

For updating the actor network, DDPG employs a deterministic policy gradient, illustrated in (37).

$$\nabla_\phi \mathcal{L}(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_a Q_\theta(s,a)|_{s=s,a=\pi(s)} \nabla_\phi \pi_\phi(s)|_{s=s}\right]. \tag{37}$$

Furthermore, DDPG employs batch normalization to mitigate performance degradation stemming from unit differences of features, while also tackling exploration challenges through the introduction of noise derived from the Ornstein-Uhlenbeck process into the deterministic policy. By leveraging these methods, DDPG demonstrates comparable control performance across a diverse array of continuous action domains.

4.4. Twin delayed deep deterministic policy gradient

The utilization of a nonlinear function approximator in DDPG often results in approximation errors, which consequently lead to the overestimation bias of the Q-function and heightened variance during the learning process. These challenges significantly impede learning speed and overall performance. In response, the twin delayed deep deterministic policy gradient (TD3) [114] algorithm emerges as a proposed solution to supplement the DDPG framework.

Firstly, to address the overestimation bias issue, the TD3 algorithm adopts clipped double Q-learning. In standard Q-learning, overestimation bias arises from the utilization of the maximum Q-function value during updates, as depicted in (30). While double Q-learning [96] proposes the usage of two critic networks and two actor networks to mitigate this bias, it doesn't entirely eliminate overestimation. Hence, TD3 introduces a clipped double Q-learning approach which trains the optimal policy for a single actor network by selecting the estimated Q value corresponding to the minimum of the two estimated Q values $\hat{Q}_{\theta_1}, \hat{Q}_{\theta_2}$ as the target, as illustrated in (38).

$$\begin{aligned}\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Bigg[&\left(r+\gamma\min_{i=1,2}\hat{Q}_{\bar{\theta}_i}(s',\pi_\phi(s'))\right.\\ &\left.- Q_{\theta_i}(s,a)\right)^2\Bigg],\ \forall i\in\{1,\,2\}.\end{aligned} \tag{38}$$

Secondly, to address the high variance issue, TD3 suggests delaying updates to the actor network, allowing it to update at a slower pace than the critic network. In an actor-critic framework like DDPG, if the critic network fails to converge adequately and exhibits a significant approximation error, updating the actor network concurrently with the critic network often leads to divergence. Therefore, rather than updating both networks simultaneously, TD3 introduces a delay in updating the actor network. This ensures that the actor network learns at a slower rate than the critic network, thereby enhancing the stability of the learning process.

4.5. Soft actor-critic

Soft actor-critic (SAC) [134] is an off-policy algorithm that integrates actor-critic methods with entropy maximization to acquire policies tailored for continuous action spaces, thereby enhancing stability and exploration. SAC addresses the challenges of high sample complexity and susceptibility to hyperparameters by incorporating the MaxEnt RL framework. This framework aims to concurrently maximize expected return and entropy, as depicted in (39).

$$\pi^* = \arg\max_\pi \sum_t \mathbb{E}_{s_t,a_t\sim\mathcal{D}}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right]. \tag{39}$$

Here $\mathcal{H}$ denotes the entropy of the policy. By maximizing entropy, SAC facilitates more efficient exploration of the action space, consequently enabling the agent to acquire proficient policies with diminished sample requirements in comparison to alternative algorithms.

In SAC, the loss function for updating the critic network is the same as (36). The actor network aims to minimize an objective function denoted as (40), for entropy maximization.

$$\mathcal{L}(\phi) = \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\phi}\left[\alpha\log(\pi_\phi(a|s)) - Q_\theta(s,a)\right]. \tag{40}$$

Specifically, SAC employs a reparameterization trick for actor updates, as illustrated in (41), to reduce gradient variance, thus enhancing the stability and effectiveness of neural network training.

$$\mathcal{L}(\phi) = \mathbb{E}_{s\sim\mathcal{D},\epsilon\sim\mathcal{N}}\Big[\alpha\log\pi_\phi(f_\phi(\epsilon;s)|s) - Q_\theta(s,f_\phi(\epsilon;s))\Big]. \tag{41}$$

Additionally, it is proposed to automatically adjust the temperature by minimizing an objective function, such as (42), rather than manual selection.

$$\mathcal{L}(\alpha) = \mathbb{E}_{a\sim\pi_\phi}\Big[-\alpha\log(\pi(a|s)) - \alpha\bar{\mathcal{H}}\Big]. \tag{42}$$

SAC has demonstrated comparable or superior performance to existing state-of-the-art model-free RL algorithms on benchmark problems.

4.6. Advantage actor-critic

To model and optimize the policy directly without using the value function, the policy gradient method [91] can be employed, expressed as follows:

$$\nabla_\phi\mathcal{L}(\phi) = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1}\nabla_\phi\log\pi_\phi(a_t|s_t)G_t\Big], \tag{43}$$

where $G_t = \sum_{t'=t+1}^{T} r_{t'}$. Policy gradients are utilized in algorithms such as REINFORCE [93], which is a typical policy-based method, as well as in actor-critic algorithms that combine policy and value functions. However, when using the return G_t as shown in (43), both the environment and the policy can be stochastic. Consequently, the same starting state can lead to very different returns, resulting in a high variance problem. To reduce variance and increase stability, the value function can be introduced as a baseline b_t.

There are various alternatives to the return G_t. Among them, the advantage actor-critic (A2C) [104] algorithm is an actor-critic method that uses an advantage function derived from the action value function $Q_\theta(s,a)$ and the state value function V_w as a baseline. The advantage function $A(s,a)$ is defined as shown in (44), and it indicates whether executing a specific action a is better than the average value for a given state.

$$A(s,a) = Q_\theta(s,a) - V_w(s). \tag{44}$$

To approximate the advantage function, the Bellman optimality equation can be used to replace the Q function with the state value function. Thus, the advantage function can be expressed in terms of the state value function alone, as shown in (45).

$$A(s,a) = r(s,a) + \gamma V_w(s') - V_w(s). \tag{45}$$

Finally, the modified policy gradient method can be applied to update the policy, as shown in (46).

$$\nabla_\phi\mathcal{L}(\phi) = \mathbb{E}_\tau\Big[\sum_{t=0}^{T-1}\nabla_\phi\log\pi_\phi(a_t|s_t)A(s_t,a_t)\Big]. \tag{46}$$

4.7. Trust region policy optimization

Trust region policy optimization (TRPO) is an algorithm designed to optimize stochastic control policies using a trust region method [110], which complements the conservative policy iteration (CPI) [184]. The conservative policy iteration, while theoretically sound, is impractical due to its reliance on a mixture policy - a convex combination of old and new policies. Therefore, in order to develop a more practical policy update method, KL-divergence is employed as the distance metric for the policy. A surrogate objective is defined based on KL-divergence, forming a lower bound on policy performance. It has been proven that optimizing this surrogate objective leads to monotonic improvement in policy performance [110].

To construct a parameterized policy-based algorithm, an unconstrained problem with a penalty is transformed into a constrained problem with a trust region constraint. It is proposed to use expected KL divergence as a constraint instead of maximum KL-divergence. Additionally, a sample-based method for estimating the objective and constraint of an optimization problem is introduced.

$$\max_\theta\ \mathbb{E}_{s\sim\rho_{\theta_{old}},\,a\sim\pi_{\theta_{old}}}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}_{\theta_{old}}(s,a)\Big], \tag{47}$$

$$\text{s.t. } \mathbb{E}_{s\sim\rho_{\theta_{old}}}\big[D_{KL}(\pi_{\theta_{old}}(\cdot|s)\|\pi_\theta(\cdot|s))\big] \le \delta. \tag{48}$$

For updating policy parameters, the conjugate gradient method with a backtracking line search is used.

4.8. Proximal policy optimization

Although the lower bound of policy performance has been shown in CPI, naive optimization leads to excessively large policy updates. Proximal policy optimization (PPO) [111], unlike TRPO, applies a method to limit policy updates by clipping the degree of policy updates to prevent excessively large updates. The claim in TRPO is that a surrogate objective that uses KL divergence as a penalty rather than a constraint forms a lower bound on policy performance. However, since determining the appropriate penalty coefficient is problem specific and time-consuming, TRPO transforms the unconstrained optimization problem into the constrained optimization problem based on the trust region method. On the other hand, PPO applies a simple modification of clipping the surrogate objective to deal with the same problem, and it is empirically verified that it performs better than other

algorithms.

$$\max_{\theta} \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \tag{49}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. There are also methods that adaptively update the penalty coefficient, but they are known to perform worse than clipping-based PPO [111].

4.9. Policy learning by weighting exploration with the returns

Policy learning by weighting exploration with the returns (PoWER) [118] is a policy search method inspired by EM algorithms, proposed for high-dimensional motor control tasks. PoWER extends the immediate reward framework, primarily addressed in previous literature [117], into an episodic RL framework. Specifically, it demonstrates that policy gradients [91] and natural policy gradients [107,163] can be derived by maximizing the lower bound of the expected return. However, due to the sensitivity of gradient-based methods to the learning rate, the PoWER algorithm, inspired by EM algorithms, proposes the application of reward-weighted regression (RWR) [117] over a larger horizon while maximizing the lower bound.

Conventional state-independent exploration methods have been characterized by high variance, frequent perturbations, and potential damage to the system executing the trajectory. Therefore, PoWER introduces state-dependent exploration, formulated in (50):

$$a \sim \pi(a_t|s_t) = \mathcal{N}(a|\theta^T\phi(s_t), \hat{\Sigma}(s_t)). \tag{50}$$

This approach proposes a methodology to update the policy parameters, as demonstrated in (51)

$$\theta' = \theta + \mathbb{E}\left[\sum_{t=1}^{T} Q^\pi(s_t,a_t)W(s_t)\right]^{-1} \times \mathbb{E}\left[\sum_{t=1}^{T} Q^\pi(s_t,a_t)W(s_t)\epsilon_t\right], \tag{51}$$

where $W(s_t) = \frac{\phi(s_t)\phi(s_t)^T}{\phi(s_t)^T\phi(s_t)}$. PoWER demonstrates superior performance compared to existing algorithms in motor learning for robotics.

4.10. Relative entropy policy search

Relative entropy policy search (REPS) [122] is a policy search algorithm designed to tackle the optimal control problem by maximizing the expected return while limiting information loss. Traditional policy search algorithms focus on finding the optimal policy based on sampled data. However, they may overlook the importance of staying close to the sampled data during policy improvement, potentially leading to significant information loss [107,163]. To address this concern, REPS introduces a novel approach by defining an optimal control problem that controls the relative entropy between path distributions generated by old and new policies as follows:

$$\max_{\pi,\mu^\pi} L(\pi) = \sum_{s,a} \mu^\pi(s)\pi(a|s)r(s,a), \tag{52}$$

$$\text{s.t. } \epsilon \geq \sum_{s,a} \mu^\pi(s)\pi(a|s)\log\frac{\mu^\pi(s)\pi(a|s)}{q(s,a)}, \tag{53}$$

$$\sum_{s'} \mu^\pi(s')\phi(s') = \sum_{s,a,s'} \mu^\pi(s)\pi(a|s)\mathcal{P}(s'|s,a)\phi(s'), \tag{54}$$

$$1 = \sum_{s,a} \mu^\pi(s)\pi(a|s). \tag{55}$$

The constraint depicted in (53) sets REPS apart from conventional optimal control methods, ensuring a balance between exploration and exploitation.

REPS has a structure that iteratively updates actor and critic based on samples. The closed-form solution of the optimization problem defined in (52)-(55) is given by (56). From this, we can compute the new policy $\pi(a|s)$.

$$\pi(a|s) = \frac{q(s,a)\exp\left(\frac{1}{\eta}\delta_\theta(s,a)\right)}{\sum_b q(s,b)\exp\left(\frac{1}{\eta}\delta_\theta(s,b)\right)}. \tag{56}$$

To find the value function $V(s) = \theta^T\phi(s)$, the dual function as shown in (57) is optimized. Since the dual function g is convex and smoothly differentiable, it can be optimized using a solver such as Broyden-Fletcher-Goldfarb-Shannon (BFGS).

$$\min_{\theta,\eta} g(\theta,\eta) = \eta\log\left(\sum_{s,a} q(s,a)\exp\left(\epsilon + \frac{1}{\eta}\delta_\theta(s,a)\right)\right). \tag{57}$$

By setting this bound on the loss of information, REPS performs well by implementing a smoother and more stable learning process, preventing excessive exploration of the state-action space.

4.11. Policy improvement with path integral

Policy improvement with path integral (PI^2) is a model-free sampling-based policy search method [123,185]. Among the policy search methods, PI^2 can explicitly consider the stochasticity of the dynamics by performing the stochastic integral in the path integral framework. Path integral has become an increasingly popular approach in RL because it can be easily combined with other methods such as CEM, MPC, covariance matrix adaptation (CMA) and backstepping control [186,187].

Suppose the continuous-time dynamics with the stochastic differential equation form

$$dx = (f(x_t) + G(x_t)u)dt + B(x_t)dw. \tag{58}$$

The value function $V(x_t)$ for this optimal control problem is given as (59).

$$V(x_t) = \min_u \mathbb{E}_{\mathbb{Q}} \left[r(x_T) + \int_t^T \left(q(x_t) + \frac{1}{2} u^T R u \right) dt \right]. \tag{59}$$

The stochastic HJB equation for the system is given as

$$\begin{aligned} -V_t &= q(x_t) + f(x_t)^T V_x - \frac{1}{2} V_x^T G R^{-1} G^T V_x \\ &\quad + \frac{1}{2} Tr(BB^T V_{xx}), \end{aligned} \tag{60}$$

with the boundary condition $V(x_T) = \phi(x_T)$. The optimal control policy is $u^*(x_t) = -R^{-1} G V_x$. Equation (60) is a nonlinear partial differential equation (PDE), intractable to solve in a high-dimensional state space. We take the exponential transformation to the value function that transforms (60) to the tractable linear PDE. The desirability function $\Psi(x_t)$ is defined as $V(x_t) = -\lambda \log \Psi(x_t)$, where λ is a temperature parameter. We need an assumption, $BB^T = \lambda G R^{-1} G^T$ to derive a linear PDE. Then, we have

$$\Psi_t = \frac{\Psi(x_t)}{\lambda} q(x_t) - f(x_t)^T \Psi_x - \frac{1}{2} Tr(BB^T \Psi xx). \tag{61}$$

This linear PDE is referred to as the backward Chapman-Kolmogorov PDE. The application of the Feynman-Kac formula provides the solution to the PDE as

$$\begin{aligned} \Psi(x_{t_0}) &= \mathbb{E}_{\mathbb{P}} \left[\exp\left(-\frac{1}{\lambda} S(\tau) \right) \right], \\ S(\tau) &= \phi(x_T) + \int_{t_0}^T q(x_t) dt. \end{aligned} \tag{62}$$

Note that the expectation is taken with respect to the probability distribution $\mathbb{P}$ of the uncontrolled dynamics ($u = 0$). Equation (62) yields the path integral for the value function. The optimal policy can be derived as

$$\begin{aligned} u^*(x_t) &= \mathcal{G}(x_{t_0}) \frac{\mathbb{E}_{\mathbb{P}}[\exp(-S(\tau)/\lambda) B(x_{t_0}) dw]}{\mathbb{E}_{\mathbb{P}}[\exp(-S(\tau)/\lambda)]}, \\ \mathcal{G}(x_{t_0}) &= R^{-1} G^T (G R^{-1} G^T)^{-1}. \end{aligned} \tag{63}$$

It is often inefficient to sample the trajectories from uncontrolled dynamics. The optimal trajectory might be highly deviated from the uncontrolled dynamics, and therefore the sampled trajectories are likely to exhibit high costs and be unsafe. The important sampling technique can alleviate this issue, such that one can choose the sampling trajectories as arbitrarily controlled dynamics. Then the path integral optimal control law in (63) can be transformed into the iterative form. Here, the Girsanov theorem is applied.

Most of the PI^2 algorithms consider the special case where the discretized dynamics have the form

$$\Delta x_i = f(x_i) \Delta t + G(x_i) \left(u(x_i) + \frac{1}{\rho} \frac{\epsilon}{\sqrt{\Delta t}} \right) \Delta t, \tag{64}$$

which is, the uncertainty enters the system only through the actuator channels. Then (63) can be written as the path integral of the sampled trajectories with respect to the controlled dynamics $\mathbb{Q}$

$$u^*(x_t) = u(x_t) + \frac{\mathbb{E}_{\mathbb{Q}}[\exp(-\tilde{S}(\tau)/\lambda) \delta u]}{\mathbb{E}_{\mathbb{Q}}[\exp(-\tilde{S}(\tau)/\lambda)]}, \tag{65}$$

where δu denotes the random change in the control input

$$\delta u = \frac{1}{\rho} \frac{\epsilon}{\sqrt{\Delta t}}. \tag{66}$$

$\tilde{S}(\tau)$ is the modified cost function that includes control cost and the control randomness cost as follows:

$$\begin{aligned} \tilde{S}(\tau) &= \phi(x_N) + \Sigma_{i=1}^{N-1} \tilde{q}(x_i, u_i), \\ \tilde{q}(x_i, u_i) &= q(x_i) + \frac{1}{2} u_i^T R_i u + u_i^T R \delta u_i. \end{aligned} \tag{67}$$

4.12. Iterative linear quadratic regulation

Iterative linear quadratic regulation (iLQR) is a model-based RL method that can be applied to the optimal control of nonlinear dynamics and cost functions [140]. The dynamics and cost functions are expanded up to the second order, and LQR finds the local optimal policy that is valid within the expansion region. The procedure is iteratively applied until convergence. ILQR is therefore analogous to the Gauss-Newton method for nonlinear least-squared trajectory optimization.

ILQR requires the first and second-order derivatives of the dynamics and cost functions with respect to the state and control variables. Consider the discrete-time dynamics

$$x_{t+1} = f(x_t, u_t). \tag{68}$$

The value function and Q function are given as

$$V(x_i) = \min_u \left(\phi(x_N) + \sum_{i=1}^{N-1} r(x_i, u_i) \right), \tag{69}$$

$$Q(x_i, u_i) = r(x_i, u_i) + V(x_{i+1}). \tag{70}$$

The Bellman equation for the discrete-time optimal control is written as

$$V(x_i) = \min_{u_i} \left[r(x_i, u_i) + V(x_{i+1}) \right]. \tag{71}$$

The both-hand-sides of the Bellman equation (71) are expanded to second order as

$$\begin{aligned} &V + V_x^T \delta x + \frac{1}{2} \delta x^T V_{xx} \delta x \\ &= \min_{\delta u} \left(r + \begin{bmatrix} Q_x \\ Q_u \end{bmatrix}^T \begin{bmatrix} \delta x \\ \delta u \end{bmatrix} \right. \end{aligned}$$

$$+\frac{1}{2}\begin{bmatrix}\delta x\\ \delta u\end{bmatrix}^T\begin{bmatrix}Q_{xx} & Q_{xu}\\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta x\\ \delta u\end{bmatrix}\Bigg), \tag{72}$$

where δx and δu are the state and control variables perturbations. The expansion coefficients are given as

$$Q_x = r_x + f_x^T V_x, \tag{73a}$$
$$Q_u = r_u + f_u^T V_x, \tag{73b}$$
$$Q_{xx} = r_{xx} + f_x^T V_{xx} f_x + \sum_{k=1}^{S} V_{x,k} f_{xx,k}, \tag{73c}$$
$$Q_{uu} = r_{uu} + f_u^T V_{xx} f_u + \sum_{k=1}^{S} V_{x,k} f_{uu,k}, \tag{73d}$$
$$Q_{xu} = r_{xu} + f_x^T V_{xx} f_u + \sum_{k=1}^{S} V_{x,k} f_{xu,k}. \tag{73e}$$

The optimizer δu^* of the quadratic form is written as

$$\begin{aligned}\delta u^* &= \arg\min_u Q(\delta x, \delta u)\\ &= -Q_{uu}^{-1}(Q_u + Q_{ux}\delta x) = k + K\delta x,\end{aligned} \tag{74}$$

where $k(i)$ and $K(i)$ are the feedforward and feedback gains. Substituting (74) into (72) yields the backward pass (recursion) of the value function and its derivatives

$$\Delta V(i) = -\frac{1}{2}Q_u^T Q_{uu}^{-1} Q_u, \tag{75a}$$
$$V_x(i) = Q_x - \frac{1}{2}Q_u^T Q_{uu}^{-1} Q_{ux}, \tag{75b}$$
$$V_{xx}(i) = Q_{xx} - \frac{1}{2}Q_{xu}^T Q_{uu}^{-1} Q_{ux}. \tag{75c}$$

The control gains k and K obtained in the backward pass are used in the forward pass to generate a new trajectory.

4.13. Stochastic differential dynamic programming

Stochastic differential dynamic programming (SDDP) is an extension of the iLQR to the stochastic nonlinear systems [141]. The stochastic dynamics are written as

$$dx = f(x_t, u_t)dt + B(x_t, u_t)dw. \tag{76}$$

The value function, Q function, and Bellman equation are identical to the iLQR as in (69)) and (71). We perform the second-order expansion same as (72). The expansion coefficients of the SDDP become

$$Q_x = r_x + f_x^T V_x + U_x, \tag{77a}$$
$$Q_u = r_u + f_u^T V_x + U_u, \tag{77b}$$
$$Q_{xx} = r_{xx} + f_x^T V_{xx} f_x + U_{xx}, \tag{77c}$$
$$Q_{uu} = r_{uu} + f_u^T V_{xx} f_u + U_{uu}, \tag{77d}$$
$$Q_{xu} = r_{xu} + f_x^T V_{xx} f_u + U_{xu}. \tag{77e}$$

Here, we can observe the difference of the coefficients compared to (73). The difference arises from the influence of the diffusion term $B(x_t, u_t)dw$ and its derivatives to the value function, given as

$$U_x = \sigma^2 \Delta t \sum_{k=1}^{S} B_{x,k}^T V_{xx} B_k, \tag{78a}$$
$$U_u = \sigma^2 \Delta t \sum_{k=1}^{S} B_{u,k}^T V_{xx} B_k, \tag{78b}$$
$$U_{xx} = \sigma^2 \Delta t \sum_{k=1}^{S} B_{x,k}^T V_{xx} B_{x,k}, \tag{78c}$$
$$U_{uu} = \sigma^2 \Delta t \sum_{k=1}^{S} B_{u,k}^T V_{xx} B_{u,k}, \tag{78d}$$
$$U_{xu} = \sigma^2 \Delta t \sum_{k=1}^{S} B_{x,k}^T V_{xx} B_{u,k}. \tag{78e}$$

The local optimal policy δu^* is identical to the iLQR as (74).

4.14. Globalized dual heuristic programming

Globalized dual heuristic programming (GDHP) is a 'partial' model-based RL method focusing on the control-affine system with quadratic control cost function [188]. The solution of Bellman equation in this system involves value, costate, and policy functions. DNNs are then used to approximate the three functions [40]. GDHP requires only a partial knowledge of the model; a system control gain and a control cost function. The functions are usually easily accessible, making GDHP more useful than the full model-based RL methods such as iLQR and DDP.

The discrete-time control-affine dynamics are given as

$$x_{t+1} = f(x_t) + G(x_t)u_t. \tag{79}$$

The path cost function and value function are given as

$$r(x_i, u_i) = q(x_i) + \frac{1}{2}u_i^T R u_i, \tag{80}$$
$$V(x_i) = \min_u \left(\phi(x_N) + \sum_{i=1}^{N-1} r(x_i, u_i)\right). \tag{81}$$

The Bellman equation takes the same form as in (71). Here, we define the costate function $\lambda(x)$ as

$$\lambda(x) = \frac{\partial V(x)}{\partial x}. \tag{82}$$

By differentiating both sides of the Bellman equation, we obtain the following iteration rule for the value, costate, and policy functions as

$$V^{(i+1)}(x_t) = r_t + V^{(i)}(x_{t+1}), \tag{83}$$
$$\lambda^{(i+1)}(x_t) = \frac{\partial r_t}{\partial x_t} + \left(\frac{\partial x_{t+1}}{\partial x_t}\right)^T \lambda^{(i)}(x_{t+1}), \tag{84}$$
$$\pi^{(i+1)}(x_t) = -\frac{1}{2}R^{-1}G^T(x_t)\lambda^{(i+1)}(x_{t+1}), \tag{85}$$

where i denotes the iteration number. Finally, DNNs $\hat{V}(x)$, $\hat{\lambda}(x)$, and $\hat{\pi}(x)$ are introduced to approximate $V^{(i+1)}(x)$, $\lambda^{(i+1)}(x)$, and $\pi^{(i+1)}(x)$, respectively.

5. ENVIRONMENTS

In this study, six chemical and bioprocess units are utilized as environments for the application of RL algorithms. This section introduces the variables, governing equations, and parameters that define each environment, as well as the control objectives and corresponding reward functions. The application of RL to each environment requires a MDP formulation, where the state space is defined by the state variables x, time t, and manipulated variables u, while the action space is defined by the changes Δu in the manipulated variables.

5.1. Van de Vusse reactor

The Van de Vusse process [189] is a benchmark problem in nonlinear control systems, consisting of a continuous stirred tank reactor (CSTR) and a cooling jacket. The primary reaction involves the transformation of liquid cyclopentadiene (substance A) into cyclopentanol (substance B) under acid-catalyst conditions. Unwanted consecutive and parallel reactions lead to the formation of byproducts, including cyclopentanediol (substance C) and dicyclopentadiene (substance D).

The dynamic model used to simulate the CSTR can be represented by the following set of equations

$$\dot{C}_A = \frac{\dot{V}}{V_R}\left(C_{A_0} - C_A\right) - k_1(T)C_A - k_3(T)C_A^2, \tag{86}$$

$$\dot{C}_B = -\frac{\dot{V}}{V_R}C_B + k_1(T)C_A - k_2(T)C_B, \tag{87}$$

$$\dot{T} = \frac{\dot{V}}{V_R}(T_0 - T) - \frac{1}{\rho C_p}\Big(k_1(T)C_A\Delta H_{R_{AB}} + k_2(T)C_B\Delta H_{R_{BC}} + k_3(T)C_A^2\Delta H_{R_{AD}}\Big) + \frac{k_w A_R}{\rho C_p V_R}\left(T_C - T\right), \tag{88}$$

$$\dot{T}_C = \frac{1}{m_C C_{pC}}\Big(\dot{Q}_C + k_w A_R\left(T - T_C\right)\Big), \tag{89}$$

subject to the conditions $C_A \geq 0$ and $C_B \geq 0$. Here, C_A and C_B represent the concentrations of substances A and B, respectively, while T and T_C denote the temperatures in the reactor and the cooling jacket, respectively. The reaction rates k_i are assumed to follow the Arrhenius equation and depend on temperature as follows:

$$k_i = k_{i0}\exp\left(\frac{E_i}{T + 273.15}\right),\ i = 1,\ 2,\ 3. \tag{90}$$

The physico-chemical parameters of the CSTR are provided in Table 3.

The state variables for this system are the concentrations C_A and C_B, along with the temperatures T and T_C. The manipulated variables include the feed flow rate $\dot{V}/V_R$ and the heat removal rate $\dot{Q}_C$. The control objective is to regulate the concentration of substance B to a specified reference value C_B^{ref}. The reward function is formulated accordingly, as shown in (91).

$$r(s,a) = -\left(y^{ref} - g(s)\right)^T Q\left(y^{ref} - g(s)\right) - a^T R a, \tag{91}$$

where $g(s) = C_B$, $y^{ref} = C_B^{ref}$, and Q and R are weight coefficients that scale the control objective term and the input regularization term, respectively. This notation is used

Table 3. Physico-chemical parameters of the CSTR.

Parameter	Description	Value	Units
k_{10}	Collision factor for reaction k_1	1.287×10^{12}	h^{-1}
k_{20}	Collision factor for reaction k_2	1.287×10^{12}	h^{-1}
k_{30}	Collision factor for reaction k_3	9.043×10^{9}	1/mol A·h
E_1	Activation energy for reaction k_1	-9758.3	K
E_2	Activation energy for reaction k_2	-9758.3	K
E_3	Activation energy for reaction k_3	-8560	K
$\Delta H_{R_{AB}}$	Enthalpies of reaction k_1	4.2	kJ/molA
$\Delta H_{R_{BC}}$	Enthalpies of reaction k_2	-11.0	kJ/molB
$\Delta H_{R_{AD}}$	Enthalpies of reaction k_3	-41.85	kJ/molA
ρ	Density	0.9342	kg/l
C_p	Heat capacity	3.01	kJ/kg·K
k_w	Heat transfer coefficient	4032	kJ/h·m^2·K
A_R	Surface of cooling jacket	0.215	m^2
V_R	Reactor volume	0.01	m^3
m_C	Coolant mass	5.0	kg
C_{pC}	Heat capacity of coolant	2.0	kJ/kg·K

Table 4. State and action values of the CSTR.

	Symbol	Description	Initial value	Minimum	Maximum	Units
States	C_A	Concentration of substance A	2.14	0	3.5	g/L
	C_B	Concentration of substance B	1.09	0	1.8	g/L
	T	Temperature in the reactor	114.2	80	140	°C
	T_C	Temperature in the cooling jacket	112.9	90	135	°C
Actions	$\dot{V}/V_R$	Feed flow	14.19	3	35	h^{-1}
	$\dot{Q}_C$	Heat removal	-1113.5	-9000	0	kJ/h

throughout the remainder of this section in the context of the reward function. The initial, minimum and maximum values for each state and action are detailed in Table 4.

5.2. Industrial batch polymerization reactor

The batch polymerization reactor utilized in this case study [190] comprises a reactor into which monomer is injected, a jacket that regulates the temperature within the reactor, and an external heat exchanger. The injected monomer is converted into polymer through an exothermic chemical reaction.

The dynamic model of the industrial batch polymerization reactor is described by mass balances for the water, monomer and product hold-ups (m_W, m_A, m_P), as well as energy balances for the reactor (T_R), the vessel (T_S), the jacket (T_M), the mixture in the external heat exchanger (T_{EK}), and the coolant exiting the external heat exchanger (T_{AWT}). These eight ordinary differential equations are formulated as follows:

$$\dot{m}_W = \dot{m}_F w_{W,F}, \tag{92}$$

$$\dot{m}_A = \dot{m}_F w_{A,F} - k_{R1} m_{A,R} - k_{R2} m_{AWT} m_A / m_{ges}, \tag{93}$$

$$\dot{m}_P = k_{R1} m_{A,R} + p_1 k_{R2} m_{AWT} m_A / m_{ges}, \tag{94}$$

$$\dot{T}_R = \frac{1}{c_{p,R}, m_{ges}} \Big(\dot{m}_F c_{p,F} \left(T_F - T_R\right) + \Delta H_R k_{R1} m_{A,R} - k_K A \left(T_R - T_S\right) - \dot{m}_{AWT} c_{p,R} \left(T_R - T_{EK}\right) \Big), \tag{95}$$

$$\dot{T}_S = \frac{1}{c_{p,S} m_S} \Big(k_K A \left(T_R - T_S\right) - k_K A \left(T_S - T_M\right) \Big), \tag{96}$$

$$\dot{T}_M = \frac{1}{c_{p,W} m_{M,KW}} \Big(\dot{m}_{M,KW} c_{p,W} \left(T_M^{\text{in}} - T_M\right) + k_K A \left(T_S - T_M\right) \Big), \tag{97}$$

$$\dot{T}_{EK} = \frac{1}{c_{p,R} m_{AWT}} \Big(\dot{M}_{AWT} c_{p,W} \left(T_R - T_{EK}\right) - \alpha \left(T_{EK} - T_{AWT}\right) + k_{R2} m_A m_{AWT} \Delta H_R / m_{ges} \Big), \tag{98}$$

$$\dot{T}_{AWT} = \frac{1}{c_{p,W} m_{AWT,KW}} \Big(\dot{m}_{AWT,KW} c_{p,W} \left(T_{AWT}^{\text{in}} - T_{AWT}\right) - \alpha \left(T_{AWT} - T_{EK}\right) \Big). \tag{99}$$

Here, $\dot{m}_F$ denotes the feed flow rate, while T_M^{in} and T_{AWT}^{in} are the coolant temperatures at the inlets of the jacket and the external heat exchanger, respectively. The remaining variables are defined as follows:

$$U = \frac{m_P}{m_A + m_P}, \tag{100}$$

$$m_{ges} = m_W + m_A + m_P, \tag{101}$$

$$k_{R1} = k_0 \exp\left(\frac{-E_a}{R(T_R + 273.15)}\right) \times \left(k_{U1}\left(1 - U\right) + k_{U2} U\right), \tag{102}$$

$$k_{R2} = k_0 \exp\left(\frac{-E_a}{R(T_{EK} + 273.15)}\right) \times \left(k_{U1}\left(1 - U\right) + k_{U2} U\right), \tag{103}$$

$$k_K = \left(m_W k_{WS} + m_A k_{AS} + m_P k_{PS}\right)/m_{ges}, \tag{104}$$

$$m_{A,R} = m_A - m_A m_{AWT}/m_{ges}, \tag{105}$$

where U denotes the polymer-monomer ratio in the reactor, m_{ges} represents the total mass, k_{R1} and k_{R2} are the reaction rates in the reactor and the external heat exchanger, k_K is the total heat transfer coefficient of the mixture inside the reactor, and $m_{A,R}$ indicates the current amount of monomer within the reactor.

To operate the polymerization reactor, it is essential to consider both safety constraints and phase separation issues. The primary safety constraint involves the maximum temperature that the reactor could reach in the event of cooling failure, denoted as T_{adiab}. This temperature must be constrained below 109°C and is governed by the following differential equation

$$\dot{T}_{adiab} = \frac{\Delta H_R}{m_{ges} c_{p,R}} \dot{m}_A + \dot{T}_R - \left(\dot{m}_W + \dot{m}_A + \dot{m}_P\right) \left(\frac{m_A \Delta H_R}{m_{ges}^2 c_{p,R}}\right). \tag{106}$$

The operation of the polymerization reactor is divided into two phases: the feeding phase, where monomer is injected, and the holding phase, where the remaining monomer undergoes reaction. Rather than treating each phase separately, we introduce the variable m_A^{acc}, representing the accumulated monomer, leading to the following expression:

$$\dot{m}_A^{acc} = \dot{m}_F. \tag{107}$$

Table 5. Parameters of the polymerization reactor.

Parameter	Description	Value	Units
R	Gas constant	8.314	kJ/kmolK
$c_{p,M}$	Specific heat capacity of the coolant	4.2	kJ/kgK
$c_{p,S}$	Specific heat capacity of the steel	0.47	kJ/kgK
$c_{p,F}$	Specific heat capacity of the feed	3.0	kJ/kgK
$c_{p,R}$	Specific heat capacity of the reactor contents	5.0	kJ/kgK
T_F	Feed temperature	25	°C
A	Heat exchange surface of the jacket	65	m^2
$m_{M,KW}$	Mass of coolant in the jacket	5,000	kg
m_S	Mass of reactor steel	39,000	kg
m_{AWT}	Mass of the product in the external heat exchanger	200	kg
$m_{AWT,KW}$	Mass of the coolant in the external heat exchanger	1,000	kg
$\dot{m}_{M,KW}$	Coolant flow rate of the jacket	300,000	kg/h
$\dot{m}_{AWT,KW}$	Coolant flow rate of the external heat exchanger	100,000	kg/h
$\dot{m}_{AWT}$	Product flow rate to the external heat exchanger	20,000	kg/h
E_a	Activation energy	8,500	kJ/kmol
ΔH_R	Specific reaction enthalpy	950	kJ/kg
k_0	Specific reaction rate	7	-
k_{U1}	Reaction parameter 1	32	-
k_{U2}	Reaction parameter 2	4	-
$w_{W,F}$	Mass fraction of water in the feed	0.333	-
$w_{A,F}$	Mass fraction of A in the feed	0.667	-
k_{WS}	Heat transfer coefficient between water and steel	4,800	W/m^2K
k_{AS}	Heat transfer coefficient between monomer and steel	1,000	W/m^2K
k_{PS}	Heat transfer coefficient between product and steel	100	W/m^2K
α	Experimental coefficient	3,600,000	s^{-1}

The parameter values for the industrial polymerization reactor are provided in Table 5.

The state variables consist of ten variables, including the mass variables m_W, m_A, m_P, m_A^{acc}, and the temperature variables T_R, T_S, T_M, T_{EK}, T_{AWT}, T_{adiab}. The system has three manipulated variables: the feed flow rate and the coolant temperatures at the inlets of the jacket and the external heat exchanger. The reward function, as represented in (108), aims to maximize the polymer product while penalizing control actions to smooth the variations in the manipulated variables.

$$r(s,a) = -(1-\tanh(m_P))Q - a^T R a. \tag{108}$$

The state and action variables of the polymerization reactor are summarized in Table 6.

5.3. Fed-batch bioreactor

The model of a fed-batch bioreactor producing penicillin [191] is a realistic representation of the process, accounting for factors such as temperature and pH in the actuation system. The mathematical model of the bioreactor comprises ordinary differential equations that represent five variables.

The growth of biomass can be described by (109), where the specific growth rate μ is assumed to follow the Contois kinetics, as computed by (110).

$$\frac{dX}{dt} = \mu X - \frac{X}{V}\cdot\frac{dV}{dt}, \tag{109}$$

$$\mu = \frac{\mu_X}{1+K_1/H^+ + H^+/K_2} \times \left(\frac{S}{K_X X + S}\right)\left(\frac{C_L}{K_{OX}X + C_L}\right) \times \left(k_g \exp\left(-\frac{E_g}{RT}\right) - k_d \exp\left(-\frac{E_d}{RT}\right)\right). \tag{110}$$

The penicillin production rate is described by the following equation

$$\frac{dP}{dt} = \mu_{pp}X - KP - \frac{P}{V}\cdot\frac{dV}{dt}, \tag{111}$$

$$\mu_{pp} = \mu_p\left(\frac{S}{K_p + S(1+S/K_I)}\right)\left(\frac{C_L^p}{K_{op}X + C_L^p}\right), \tag{112}$$

where μ_{pp} represents the specific penicillin production rate, which is described by the substrate-inhibition model.

Glucose and oxygen are utilized during biomass growth and product formation. The dynamics of the substrate and

Table 6. State and action values of the polymerization reactor.

	Symbol	Description	Initial value	Minimum	Maximum	Units
States	m_W	Mass of water	10000	0	3×10^4	kg
	m_A	Mass of monomer	853	0	3×10^4	kg
	m_P	Mass of product	26.5	0	3×10^4	kg
	T_R	Temperature in the reactor	90.0	88	92	°C
	T_S	Temperature in the vessel	90.0	0	100	°C
	T_M	Temperature in the jacket	90.0	0	100	°C
	T_{EK}	Temperature of the mixture in the external heat exchanger	35.0	0	100	°C
	T_{AWT}	Temperature of the coolant leaving the external heat exchanger	35.0	0	100	°C
	T_{adiab}	Temperature if the cooling system fails	104.897	0	109	°C
	m_F^{acc}	Mass of the accumulated monomer	0	0	30000	kg
Actions	$\dot{m}_F$	Feed flow	0	0	30000	kg/h
	T_M^{in}	Coolant temperature at the inlet of the jacket	60	60	100	°C
	T_{AWT}^{in}	Coolant temperature at the inlet of the external heat exchanger	60	60	100	°C

dissolved oxygen are described by the following equations

$$\frac{dS}{dt} = -\frac{\mu}{Y_{X/S}}X - \frac{\mu_{pp}}{Y_{p/s}}X - m_X X + F\frac{s_f}{V} - \frac{S}{V}\frac{dV}{dt}, \tag{113}$$

$$\frac{dC_L}{dt} = -\frac{\mu}{Y_{X/O}}X - \frac{\mu_{pp}}{Y_{p/O}}X - m_O X + K_{la}\left(C_L^* - C_L\right) - \frac{C_L}{V}\cdot\frac{dV}{dt}, \tag{114}$$

$$K_{la} = \alpha\sqrt{f_g}\left(\frac{P_w}{V}\right)^{\beta}. \tag{115}$$

Here, the overall mass transfer coefficient K_{la} is a function of the agitation power P_w and the oxygen flow rate f_g, with α and β representing empirical constants.

The volume of the bioreactor changes in response to the substrate flow rate and the evaporation loss F_{loss}. This can be represented by the following equations

$$\frac{dV}{dt} = F - F_{loss}, \tag{116}$$

$$F_{loss} = V\lambda\left(\exp\left(5\times\frac{T-T_0}{T_v-T_0}\right) - 1\right). \tag{117}$$

The parameter values are provided in Table 7.

The controller aims to maximize the productivity of the penicillin (i.e., PV) and its yield concerning the cumulative substrate (i.e., $PV/\left(S(0)V(0) + s_f\int_0^t F(t)dt\right)$). Two states are additionally augmented to a state space model to formulate the objective function as a standard RL form as

$$\frac{dS_a}{dt} = s_f F, \quad \frac{dF}{dt} = \dot{F}(t). \tag{118}$$

Now we define the reward function as

$$r(s,a) = -\left[1 - \tanh\left(PVR_0 + \frac{PV}{S_a}R_1\right)\right]Q - a^T Ra. \tag{119}$$

Tanh function is used to design the reward function to have zero value when maximized. The initial, minimum and maximum values for each state and action variable of the bioreactor are summarized in Table 8.

5.4. Crystallization reactor

The crystallization reactor [192] is a tube crystallizer in which the evaporative crystallization reaction of an ammonium sulfate-water system is conducted using a fed-batch process. The reactor features an inlet for a crystal-free stream, compensating for volume loss due to solvent evaporation and slurry sampling, and two outlets for the product removal stream and vapor stream. The inlet stream consists of a saturated ammonium sulfate solution, which is injected at a constant temperature of 50°C to maintain isothermal conditions.

The dynamics of the crystallization process can be rigorously modeled using a population balance equation in conjunction with conservation laws and kinetic relations. The population balance equations are reduced to ordinary differential equations by employing the method of moments. The temporal evolution of moments of the crystal size distribution (CSD) can be described as follows:

$$\frac{dm_i}{dt} = 0^i\cdot B_0 + i\cdot G m_{i-1} - \frac{m_i F_p}{V}, \tag{120}$$

$$m_i(0) = m_{i,0},\ i = 0,\ \cdots,\ 4, \tag{121}$$

where m_i denotes the i-th moment of the CSD, G represents the crystal growth rate, B_0 is the nucleation rate, V is the reactor volume, and F_p is the product flow rate.

Table 7. Parameters of the fed-batch bioreactor.

	Parameter	Description	Value	Units
Feed	s_f	Biomass concentration	600	g/L
	T_f	Feed temperature	298	K
Biomass	μ_x^*	Maximum specific growth rate	0.092	h^{-1}
	K_x	Contois saturation constant	0.15	g/L
	K_{ox}	Oxygen limitation constant	2×10^{-2}	-
	k_g	Arrhenius constants for growth	7×10^3	K
	k_d	Arrhenius constants for cell death	1×10^{33}	K
	E_g	Activation energies for cell death	5100	cal/mol
	E_d	Activation energies for cell death	50000	cal/mol
	K_1	pH dependency constant	1×10^{-10}	mol/L
	K_2	pH dependency constant	7×10^{-5}	mol/L
Penicillin	μ_p	Specific rate of penicillin production	0.005	h^{-1}
	K_p	Inhibition constant	0.0002	g/L
	K_I	Dissociation constant	0.10	g/L
	K_{op}	Oxygen limitation constant	5×10^{-4}	-
	p	Dissolved oxygen dependency constant	3	-
	K	Penicillin hydrolysis rate constant	0.04	h^{-1}
Substrate	$Y_{x/s}$	Yield (biomass/glucose)	0.45	g/g
	$Y_{x/o}$	Yield (biomass/oxygen)	0.04	g/g
	$Y_{p/s}$	Yield (penicillin/glucose)	0.90	g/g
	$Y_{p/o}$	Yield (penicillin/oxygen)	0.20	g/g
	m_x	Maintenance coefficients on substrate	0.014	h^{-1}
	m_o	Maintenance coefficients on oxygen	0.467	h^{-1}
	C_L^*	Saturation concentration of C_L	1.16	g/L
	α, β	Mass transfer modeling constants	70, 0.4	-
	P_w	Agitation power	29.9	kW
	f_g	Oxygen flow rate	0.1	g/L
Volume	λ	Volume loss constant	2.5×10^{-4}	h^{-1}
	T_0	Freezing temperatures	273.15	K
	T_v	Boiling temperatures	373.15	K

Table 8. States and action values of the bioreactor.

	Symbol	Description	Initial value	Minimum	Maximum	Units
States	X	Biomass concentration	0.1	0	50	g/L
	S	Substrate concentration	15	0	25	g/L
	C_L	Dissolved oxygen concentration	1.16	0	1.5	g/L
	P	Penicillin concentration	0	0	4	g/L
	V	Culture volume	100	90	110	L
Action	F	Substrate feed flow rate	0.05	0	0.05	L/h

Under isothermal conditions, the mass and energy balance equations can be simplified as shown in (122). The constant coefficients k_1 and k_2 are expressed in (123) and (124), respectively.

$$\frac{dC}{dt} = \frac{1}{1 - K_V m_3}\left(\frac{F_p\left(C^* - C\right)}{V} + 3K_V G m_2\left(k_1 + C\right) + k_2 Q_v\right), \tag{122}$$

$$k_1 = \frac{H_V C^*}{H_V - H_L}\left(\frac{\rho_C}{\rho_L} - 1 + \frac{\rho_L H_L - \rho_C H_C}{\rho_L H_V}\right) - \frac{\rho_C}{\rho_L}, \tag{123}$$

$$k_2 = \frac{C^*}{V\rho_L\left(H_V - H_L\right)}. \tag{124}$$

Additionally, the total nucleation rate and the size-independent crystal growth rate can be modeled as follows:

$$B_0 = k_b m_3 G, \tag{125}$$

Table 9. Parameters of the crystallization reactor.

Parameters	Description	Value	Unit
C^*	Saturation concentration	0.46	kg/kg
g	Growth rate exponent	1	-
H_C	Specific enthalpy of crystals	60.75	kJ/kg
H_L	Specific enthalpy of liquid	69.86	kJ/kg
H_V	Specific enthalpy of vapor	2.59×10^{3}	kJ/kg
K_V	Volumetric shape factor	0.43	-
k_b	Nucleation rate constant	1.02×10^{15}	#/m^4
k_g	Growth rate constant	7.50×10^{-5}	m/s
F_P	Product flow rate	1.73×10^{-6}	m^3/s
V	Crystallizer volume	7.50×10^{-2}	m^3
ρ_C	Crystal density	1767.35	kg/m^3
ρ_L	Solution density	1248.93	kg/m^3

Table 10. States and action values of the crystallization reactor.

	Symbol	Description	Initial value	Minimum	Maximum	Units
States	m_0	0^{th} Moment	1×10^{11}	1×10^{11}	1.2×10^{11}	1/m^3
	m_1	1^{st} Moment	4×10^{6}	4×10^{6}	4×10^{7}	m/m^3
	m_2	2^{nd} Moment	400	400	4000	m^2/m^3
	m_3	3^{rd} Moment	0.1	0.1	1.0	m^3/m^3
	m_4	4^{th} Moment	0.2×10^{-4}	0.2×10^{-4}	0.2×10^{-3}	m^4/m^3
	C	Solute concentration	0.461	0.460	0.461	kg/kg
Action	Q_V	Heat input	8.5	7	20	kW

$$G = k_g\left(C - C^*\right)^g. \tag{126}$$

The parameters for the crystallization process are provided in Table 9.

The optimization objective of the batch crystallizer is to maximize the crystal growth, characterized by the Sauter mean diameter $d_{32} = m_3/m_2$. At the same time, the amount of heat input Q_v should be minimized. The reward function is designed as

$$r(s,a) = -\left[1 - \tanh\left(\frac{m_3}{m_2}\right)\right] Q_1 - Q_v^2 Q_2 - a^T R a. \tag{127}$$

The initial, minimum and maximum values for each state and action variable of crystallization reactor are provided in Table 10.

5.5. Distillation column

In this study, a binary distillation column with 32 trays is considered [193], where the vapor and liquid flow rates are assumed to be constant across all trays. The energy balance is not taken into account. The feed stream is introduced at the 17th tray, and the relative volatility, γ, is assumed to be constant.

The vapor-liquid equilibrium equations are represented as algebraic equations as follows:

$$L_t = rr_t D, \tag{128}$$

$$V_t = L_t + D, \tag{129}$$

$$S_t = F + L_t, \tag{130}$$

$$y_{n,t} = \frac{\alpha x_{n,t}}{1 + (\alpha - 1)x_{n,t}}, \quad \forall t \in \{t_0, \cdots, t_f\},\ \forall n \in \{1, \cdots, 32\}. \tag{131}$$

Here, V_t denotes the vapor flow rate, while L_t and S_t represent the liquid flow rates in the rectification and the stripping sections, respectively. The reflux ratio rr_t is constrained by the condition $1 \le u_t \le 5$. $x_{n,t}$ and $y_{n,t}$ are the mole fractions of liquid and vapor phases, respectively.

The material balances on each tray n for all t in $\{t_0, \cdots, t_f\}$ are described by the following ordinary differential equations

$$\dot{x}_{1,t} = \frac{1}{M_1} V_t (y_{2,1} - x_{1,t}), \tag{132}$$

$$\dot{x}_{n,t} = \frac{1}{M_n}\left(L_t(x_{n-1,t} - x_{n,t}) - V_t(y_{n,t} - y_{n+1,t})\right) \quad \forall n \in \{2,\ \cdots,\ 16\}, \tag{133}$$

$$\dot{x}_{17,t} = \frac{1}{M_{17}}(F x_f + L_t x_{16,t} - S_t x_{17,t} - V_t(y_{17,t} - y_{18,t})), \tag{134}$$

$$\dot{x}_{n,t} = \frac{1}{M_n}\left(S_t(x_{n-1,t} - x_{n,t}) - V_t(y_{n,t} - y_{n+1,t})\right) \quad \forall n \in \{18,\ \cdots,\ 31\}, \tag{135}$$

Table 11. Parameters of the distillation column.

Parameter	Description	Value	Units
F	Feed flow rate	0.4	kg/h
D	Distillate flow rate	0.2	kg/h
α	Relative volatility	1.6	-
M_1	Holdup at tray 1	0.5	kg
M_{32}	Holdup at tray 32	1.0	kg
M_n	Holdup at tray n ($n \notin \{1,32\}$)	0.25	kg

$$\dot{x}_{32,t} = \frac{1}{M_{32}}(S_t x_{31,t} - (F - D_t)x_{32,t} - V_t y_{32,t}). \quad (136)$$

In these equations, M_n represents the holdup on tray n. The values of the parameters used are provided in Table 11.

The state variable in the distillation column environment is the liquid-phase composition x_n for each of the 32 trays, while the manipulated variable is the reflux ratio rr. The control objective of this system is to adjust the manipulated variable such that the mole fraction of the first tray reaches its reference value with minimal manipulation of the reflux ratio. The reward function used for this purpose is provided in (137).

$$r(s,a) = -\big(y^{ref} - g(s)\big)^T Q\big(y^{ref} - g(s)\big) - a^T R a, \quad (137)$$

where $y^{ref} = x_1^{ref}$, $g(s) = x_1$. The initial, minimum, and maximum values of the state and manipulated variables are provided in Table 12.

5.6. Plug flow reactor

The plug flow reactor (PFR) [194] is a type of distributed parameter system, characterized as a nonlinear tubular reactor with both diffusion and convection. The dynamic behavior of the PFR can be described by dimensionless partial differential equations, as formulated in (138) and (139).

$$\frac{\partial T}{\partial t} = \frac{1}{Pe_h}\frac{\partial^2 T}{\partial z^2} - \frac{1}{Le}\frac{\partial T}{\partial z} + \eta c \exp\left[\gamma\left(1 - \frac{1}{T}\right)\right] + \mu\big(T_w(z,t) - T(z,t)\big), \quad (138)$$

$$\frac{\partial c}{\partial t} = \frac{1}{Pe_m}\frac{\partial^2 c}{\partial z^2} - \frac{\partial c}{\partial z} - Dac \exp\left[\gamma\left(1 - \frac{1}{T}\right)\right]. \quad (139)$$

Here, T and c denote the dimensionless temperature and concentration, respectively. The boundary conditions are specified as Neumann boundary conditions as follows:

$$z = 0 \begin{cases} -\dfrac{\partial T}{\partial z} = Pe_h\big(T_i(t) - T(z,t)\big), \\ -\dfrac{\partial c}{\partial z} = Pe_m\big(c_i(t) - c(z,t)\big), \end{cases} \quad (140)$$

$$z = 1 \begin{cases} \dfrac{\partial T}{\partial z} = 0, \\ \dfrac{\partial c}{\partial z} = 0. \end{cases} \quad (141)$$

The dimensionless parameters are provided in Table 13.

To solve the partial differential equations, the reactor interior is partitioned into six compartments, transforming the equation into six ordinary differential equations for concentration and temperature, respectively. The state variables consist of the concentration and temperature for each compartment. The system's manipulated variable is the reactor's wall temperature T_w. In this case study, the reactor is divided into two partitions, with the wall temperature of each partition serving as the manipulated variable. It is assumed that these temperatures can be adjusted independently without influencing one another. The control objective is to regulate the concentration exiting the last of the six compartments to a reference value, while minimizing fluctuations in the manipulated variables. The reward function for this system is defined accordingly.

$$r(s,a) = -\big(y^{ref} - g(s)\big)^T Q\big(y^{ref} - g(s)\big) - a^T R a, \quad (142)$$

where $y^{ref} = c_5^{ref}$, $g(s) = c_5$. The initial, minimum, and maximum values for the state and manipulated variables are presented in Table 14.

6. SIMULATION RESULTS

The RL algorithms introduced in Section 4 were applied to the benchmark process control problems outlined in Section 5. These algorithms were tested on the benchmarks to evaluate their effectiveness, with performance assessed using various metrics. The results highlight the capabilities and limitations of each algorithm, offering valuable insights into their potential for real-world process control applications. The implementations of the algorithms and environments are available at http://github.com/skjw1224/epelRL.

We established five metrics to numerically compare the performance of each algorithm. These metrics were designed to evaluate algorithm convergence, the performance of the trained agents, and computational efficiency. The values for each metric, reflecting the performance of each algorithm across different environments, are provided in Table 15.

To assess algorithm convergence, we defined a specific convergence criterion. Since most algorithms learn a critic

Table 12. States and action values of the distillation column.

	Symbol	Description	Initial value	Minimum	Maximum	Units
States	x_1	Liquid-phase mole fraction at tray 1	0.9354	0	1	–
	x_2	Liquid-phase mole fraction at tray 2	0.9005	0	1	–
	x_3	Liquid-phase mole fraction at tray 3	0.8623	0	1	–
	x_4	Liquid-phase mole fraction at tray 4	0.8216	0	1	–
	x_5	Liquid-phase mole fraction at tray 5	0.7799	0	1	–
	x_6	Liquid-phase mole fraction at tray 6	0.7385	0	1	–
	x_7	Liquid-phase mole fraction at tray 7	0.6988	0	1	–
	x_8	Liquid-phase mole fraction at tray 8	0.6618	0	1	–
	x_9	Liquid-phase mole fraction at tray 9	0.6285	0	1	–
	x_{10}	Liquid-phase mole fraction at tray 10	0.5992	0	1	–
	x_{11}	Liquid-phase mole fraction at tray 11	0.5741	0	1	–
	x_{12}	Liquid-phase mole fraction at tray 12	0.5531	0	1	–
	x_{13}	Liquid-phase mole fraction at tray 13	0.5357	0	1	–
	x_{14}	Liquid-phase mole fraction at tray 14	0.5216	0	1	–
	x_{15}	Liquid-phase mole fraction at tray 15	0.5103	0	1	–
	x_{16}	Liquid-phase mole fraction at tray 16	0.5012	0	1	–
	x_{17}	Liquid-phase mole fraction at tray 17	0.4941	0	1	–
	x_{18}	Liquid-phase mole fraction at tray 18	0.4854	0	1	–
	x_{19}	Liquid-phase mole fraction at tray 19	0.4742	0	1	–
	x_{20}	Liquid-phase mole fraction at tray 20	0.4598	0	1	–
	x_{21}	Liquid-phase mole fraction at tray 21	0.4416	0	1	–
	x_{22}	Liquid-phase mole fraction at tray 22	0.4191	0	1	–
	x_{23}	Liquid-phase mole fraction at tray 23	0.3920	0	1	–
	x_{24}	Liquid-phase mole fraction at tray 24	0.3602	0	1	–
	x_{25}	Liquid-phase mole fraction at tray 25	0.3240	0	1	–
	x_{26}	Liquid-phase mole fraction at tray 26	0.2846	0	1	–
	x_{27}	Liquid-phase mole fraction at tray 27	0.2432	0	1	–
	x_{28}	Liquid-phase mole fraction at tray 28	0.2018	0	1	–
	x_{29}	Liquid-phase mole fraction at tray 29	0.1617	0	1	–
	x_{30}	Liquid-phase mole fraction at tray 30	0.1251	0	1	–
	x_{31}	Liquid-phase mole fraction at tray 31	0.0924	0	1	–
	x_{32}	Liquid-phase mole fraction at tray 32	0.0645	0	1	–
Action	rr	Reflux ratio	3.0	1	5	–

Table 13. Parameters of the plug flow reactor.

Parameter	Description	Value	Units
T_i	Inlet temperature	1.0	-
c_i	Inlet concentration	1.0	-
Pe_h	Peclet number (energy)	5.0	-
Pe_m	Peclet number (mass)	5.0	-
Le	Lewis number	1.0	-
Da	Damkohler number	0.875	-
γ	Activation energy	15.0	-
η	Heat of reaction	0.8375	-
μ	Heat transfer coefficient	13.0	-

or value function, we monitored critic loss throughout the training process. The convergence criterion was determined by calculating the standard deviation of critic loss over a moving window of 50 episodes. Convergence was considered achieved when the variation in critic loss fell to 10% of the initial standard deviation, calculated from the first two episodes. When the episode count was less than 50, we calculated the standard deviation using avail-

Table 14. States and action values of the plug flow reactor.

	Symbol	Description	Initial value	Minimum	Maximum	Units
States	T_0	Temperature at $z = 0$	1.0	0.5	2.0	–
	T_1	Temperature at $z = 1$	1.0	0.5	2.0	–
	T_2	Temperature at $z = 2$	1.0	0.5	2.0	–
	T_3	Temperature at $z = 3$	1.0	0.5	2.0	–
	T_4	Temperature at $z = 4$	1.0	0.5	2.0	–
	T_5	Temperature at $z = 5$	1.0	0.5	2.0	–
	c_0	Concentration at $z = 0$	0.4	0	1.0	–
	c_1	Concentration at $z = 1$	0.4	0	1.0	–
	c_2	Concentration at $z = 2$	0.4	0	1.0	–
	c_3	Concentration at $z = 3$	0.4	0	1.0	–
	c_4	Concentration at $z = 4$	0.4	0	1.0	–
	c_5	Concentration at $z = 5$	0.4	0	1.0	–
Action	$T_{w,1}$	Wall temperature at 1^{st} partition	1.0	0.3	2.0	–
	$T_{w,2}$	Wall temperature at 2^{nd} partition	1.0	0.3	2.0	–

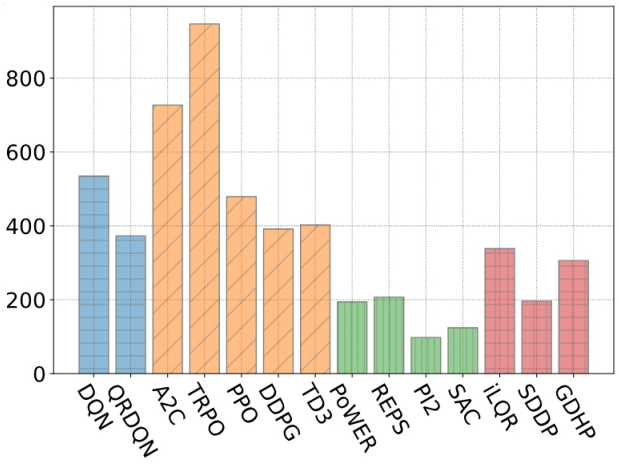

Fig. 1. Comparison of converged episodes across the implemented algorithms.

able data up to that point and continued training until the episode count exceeded 50, even if the convergence criterion reached below 10% of the initial value. We then measured the number of training episodes required for each algorithm to meet the defined convergence criterion. Using this approach, we evaluated whether the critic loss decreased sufficiently during training and how efficiently the training process was conducted. The average values of the number of converged episodes can be found in Fig. 1.

The convergence behavior of various algorithms differed based on their underlying approaches. Value-based algorithms, such as DQN and QR-DQN, tended to converge more slowly than those utilizing actor networks. Model-based algorithms, including iLQR, SDDP, and GDHP, converged relatively quickly due to their use of models, while PI^2 and SAC outperformed model-based algorithms in terms of convergence speed. Among policy-based algorithms, on-policy methods such as A2C, TRPO, and PPO generally required more episodes to converge compared to off-policy methods like DDPG and TD3. In this study, however, on-policy algorithms were implemented with a single update at the end of each episode, suggesting they required fewer updates overall to reach convergence than off-policy algorithms.

We tuned the hyperparameters of each algorithm with a focus on optimizing learning progress and minimizing critic loss according to the defined convergence criterion. To streamline the tuning process, we applied common values to fundamental hyperparameters and those related to network structure, as shown in Table 16. Algorithm-specific hyperparameters are detailed in Tables 17-20. It is important to note that these parameter values are not necessarily the optimal choice for each environment.

To assess the performance of the trained agents, we evaluated the mean and standard deviation of cost. Since state noise was added to each environment, we conducted a test run of five episodes and calculated the mean and standard deviation of cost across these episodes. A smaller mean value indicated better performance, while a lower standard deviation indicated less variability in operation despite state noise. The mean and standard deviation of cost for each algorithm are presented in Figs. 2 and 3.

When comparing average performance and variability, no algorithm showed a clear superiority over the others. Among them, GDHP, DQN, SAC, and DDPG achieved the lowest average costs, indicating strong performance on process control. Additionally, SDDP, DQN, iLQR, and TRPO demonstrated reduced variability, with relatively low standard deviations.

The six environments used in this study consist of three regulation problems (Subsections 5.1, 5.5, 5.6) and three product maximization problems (Subsections 5.2, 5.3, 5.4). The average cost per algorithm for each problem

Table 15. Performance metrics for each algorithm and environment. Each cell in the table contains, from top to bottom, the convergence criteria, number of converged episodes, average cost with standard deviation, and episodic computation time.

	Env 5.1	Env 5.2	Env 5.3	Env 5.4	Env 5.5	Env 5.6
DQN	0.1999 1000 episodes 0.063 ± 0.008 5.3999 sec	113.8759 1000 episodes 1.105 ± 0.025 0.9617 sec	0.0901 106 episodes 1.224 ± 0.000 7.6381 sec	1.0945 1000 episodes 3.949 ± 0.371 3.1935 sec	0.0265 51 episodes 0.009 ± 0.000 18.1019 sec	0.0607 51 episodes 0.002 ± 0.000 5.0433 sec
QR-DQN	0.0988 66 episodes 0.017 ± 0.001 5.4520 sec	2.2334 1000 episodes 2.247 ± 0.016 1.3189 sec	0.1941 1000 episodes 1.219 ± 0.001 7.5637 sec	0.0995 56 episodes 5.803 ± 0.412 3.2277 sec	0.0973 63 episodes 0.035 ± 0.001 18.5190 sec	0.0586 52 episodes 0.002 ± 0.000 4.3187 sec
DDPG	0.1416 1000 episodes 0.013 ± 0.002 7.2863 sec	0.0931 99 episodes 0.752 ± 0.013 1.1491 sec	0.0943 83 episodes 1.220 ± 0.002 9.4014 sec	0.6341 1000 episodes 4.471 ± 0.449 4.4043 sec	0.0519 51 episodes 0.092 ± 0.001 19.2306 sec	0.0958 117 episodes 0.003 ± 0.001 5.0845 sec
TD3	0.6737 1000 episodes 0.019 ± 0.002 7.8937 sec	0.0976 158 episodes 2.238 ± 0.033 1.1930 sec	0.0996 154 episodes 1.225 ± 0.000 9.8651 sec	0.8632 1000 episodes 4.336 ± 0.577 3.6289 sec	0.0961 54 episodes 0.245 ± 0.003 20.2628 sec	0.0132 51 episodes 0.002 ± 0.001 3.8714 sec
SAC	0.0972 92 episodes 0.025 ± 0.006 11.7791 sec	0.0968 109 episodes 1.123 ± 0.057 1.8184 sec	0.0774 77 episodes 1.225 ± 0.001 14.4175 sec	0.0978 318 episodes 3.904 ± 0.380 6.9238 sec	0.0199 51 episodes 0.073 ± 0.006 25.9213 sec	0.0993 99 episodes 0.003 ± 0.000 7.9324 sec
A2C	2.2090 1000 episodes 0.018 ± 0.005 3.818 sec	0.0917 103 episodes 1.809 ± 0.148 0.974 sec	0.0994 259 episodes 1.225 ± 0.001 5.938 sec	3.6867 1000 episodes 4.064 ± 0.387 2.502 sec	29.0082 1000 episodes 0.023 ± 0.011 7.301 sec	0.5288 1000 episodes 0.004 ± 0.002 3.877 sec
TRPO	54.9355 1000 episodes 0.036 ± 0.003 4.2199 sec	2.3964 1000 episodes 2.241 ± 0.036 3.4647 sec	53.2963 1000 episodes 1.226 ± 0.001 6.4006 sec	1.6830 1000 episodes 3.834 ± 0.370 2.3150 sec	45.8139 1000 episodes 0.053 ± 0.001 8.6221 sec	16550.06 1000 episodes 0.004 ± 0.000 1.6219 sec
PPO	0.0989 422 episodes 0.023 ± 0.002 58.6378 sec	0.0951 167 episodes 2.048 ± 0.071 0.9579 sec	184.7986 1000 episodes 1.223 ± 0.001 5.8210 sec	0.2496 1000 episodes 6.309 ± 0.467 37.3324 sec	0.0993 100 episodes 0.090 ± 0.001 12.1681 sec	0.0974 184 episodes 0.003 ± 0.000 3.8789 sec
REPS	0.0005 52 episodes 0.046 ± 0.001 23.2553 sec	0.0721 919 episodes 2.256 ± 0.022 179.0177 sec	0.0038 52 episodes 1.224 ± 0.000 34.4216 sec	0.0924 67 episodes 6.703 ± 0.450 305.2800 sec	0.0367 52 episodes 0.016 ± 0.000 216.4288 sec	0.0973 100 episodes 0.001 ± 0.000 284.1908 sec
PI^2	0.0994 138 episodes 0.048 ± 0.026 10.0617 sec	0.0535 52 episodes 2.224 ± 0.026 15.5519 sec	0.0526 52 episodes 1.224 ± 0.000 22.8700 sec	0.0769 52 episodes 6.012 ± 0.547 10.2794 sec	0.0623 52 episodes 0.015 ± 0.000 24.4315 sec	0.0945 241 episodes 0.001 ± 0.000 6.3884 sec
iLQR	0.0945 259 episodes 0.002 ± 0.000 32.7227 sec	0.0999 397 episodes 1.906 ± 0.077 133.9712 sec	0.0989 338 episodes 1.224 ± 0.001 137.5543 sec	0.0990 387 episodes 5.250 ± 0.349 37.6597 sec	0.0851 456 episodes 0.001 ± 0.000 586.0436 sec	0.0992 194 episodes 0.001 ± 0.000 69.7655 sec
SDDP	0.0990 126 episodes 0.002 ± 0.000 1040.0260 sec	0.0989 230 episodes 2.082 ± 0.025 186.8030 sec	0.0947 196 episodes 1.225 ± 0.000 167.4444 sec	0.0970 215 episodes 3.838 ± 0.366 60.3685 sec	0.1000 356 episodes 0.001 ± 0.000 889.9191 sec	0.0688 56 episodes 0.001 ± 0.000 2341.6320 sec
GDHP	0.0967 390 episodes 0.003 ± 0.000 80.6878 sec	0.0996 69 episodes 0.697 ± 0.011 99.8510 sec	0.0863 79 episodes 1.220 ± 0.001 424.3303 sec	0.0989 242 episodes 4.310 ± 0.606 70.7151 sec	0.0962 57 episodes 0.002 ± 0.000 985.3778 sec	0.6190 1000 episodes 0.001 ± 0.000 132.1814 sec

Table 16. Common hyperparameters across all algorithms.

Category	Hyperparameter	Value
Common parameters	Maximum episode	1000
	Buffer size	1×10^6
	Mini-batch size	1024
	Discount (γ)	0.99
	Number of warm-up episode	1
	Number of hidden nodes	128
	Number of hidden layers	2
	Soft update (τ)	0.005
	Gradient clipping magnitude	5
	Actor learning rate	1×10^{-4}
	RBF type	Gaussian

Table 17. Hyperparameter settings specific to value-based algorithms.

Category	Algorithm	Hyperparameter	Value
Value-based	DQN	Critic learning rate	1×10^{-2}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
		Max number of action grid	200
	QR-DQN	Critic learning rate	1×10^{-4}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-1}
		Max number of action grid	200
		N quantiles	21

Table 18. Hyperparameter settings specific to inference-based algorithms.

Category	Algorithm	Hyperparameter	Value
Inference-based	SAC	Critic learning rate	1×10^{-3}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
		Automatic temperature tuning	True
	REPS	RBF dimension	50
		Number of roll-outs	50
		Max KL divergence	10
		Critic regularization	0.01
		Actor regularization	1
	PI^2	RBF dimension	100
		Number of roll-outs	5
		h	10
		Initial lambda	0.1
		Exploration decay	0.99
	PoWER	RBF dimension	1000
		Number of roll-outs	1000
		Variance update	True

category is shown in Fig. 4. As indicated in Fig. 4, model-based algorithms performed significantly better on regulation problems. By contrast, model-free algorithms generally exhibited similar control performance, with value-based algorithms performing particularly well. In product maximization problems, certain algorithms, including GDHP, DQN, SAC, and DDPG, achieved notably high performance.

To further compare the computational efficiency of each algorithm, we calculated the average and median values of episodic computation time, as shown in Fig. 5. Overall, model-free RL algorithms demonstrated lower

Table 19. Hyperparameter settings specific to policy-based algorithms.

Category	Algorithm	Hyperparameter	Value
Policy-based	DDPG	Critic learning rate	1×10^{-3}
		Adam epsilon	1×10^{-4}
		L2 regularization	1×10^{-3}
	TD3	Critic learning rate	1×10^{-2}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
		Policy noise	0.2
		Noise clip	0.5
		Policy delay	2
	A2C	Critic learning rate	1×10^{-2}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
	TRPO	Critic learning rate	1×10^{-4}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
		GAE lambda	0.99
		GAE gamma	0.99
		Number of conjugate gradient iterations	100
		Number of line search iterations	10
		Max KL divergence	0.001
	PPO	Critic learning rate	1×10^{-3}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
		GAE lambda	0.99
		GAE gamma	0.99
		Number of conjugate gradient iterations	10
		Number of line search iterations	10
		Max KL divergence	0.01
		Clip epsilon	0.1

Table 20. Hyperparameter settings specific to model-based algorithms.

Category	Algorithm	Hyperparameter	Value
Model-based	iLQR	α	0.1
		Learning rate	0.99
	SDDP	α	0.5
		Learning rate	0.99
	GDHP	Critic learning rate	1×10^{-4}
		Adam epsilon	1×10^{-6}
		L2 regularization	1×10^{-3}
		Costate learning rate	1×10^{-4}

computational costs during the learning process. In contrast, model-based RL algorithms required significantly higher computational resources due to the need to calculate derivative terms, resulting in substantially longer computation times than model-free algorithms.

A radar plot of the evaluation metric values for each algorithm category - value-based, policy-based, inference-based, and model-based - is shown in Figs. 6-9. The algorithms within each category exhibit similar convergence properties, with computation times generally low across categories, except for the model-based methods. However, performance metrics are widely distributed across algorithms, regardless of category. This distribution suggests that to achieve optimal performance when applying RL to process control, it is essential to select a method specific to the environment.

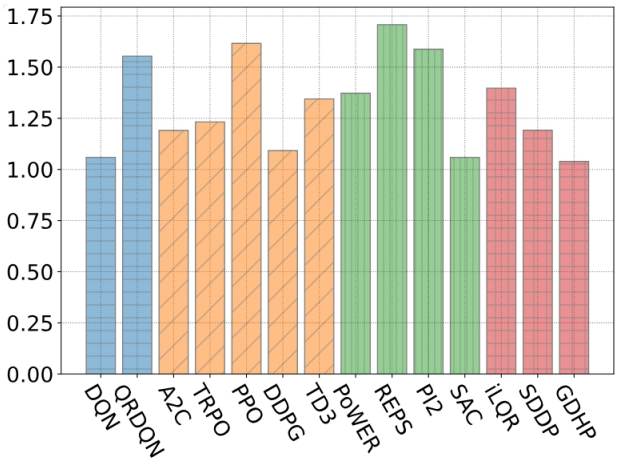

Fig. 2. Comparison of average cost across the implemented algorithms.

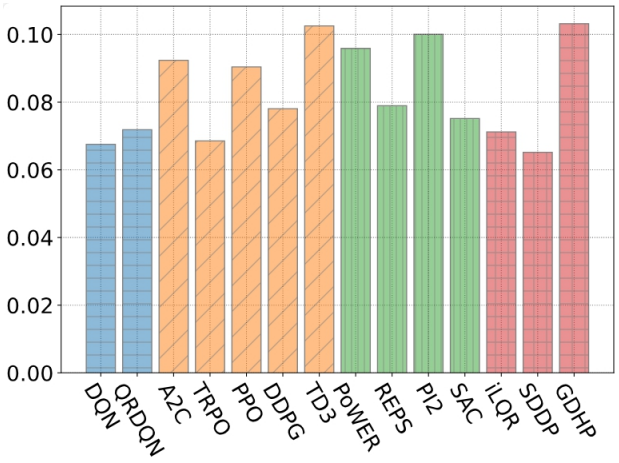

Fig. 3. Comparison of cost standard deviation across the implemented algorithms.

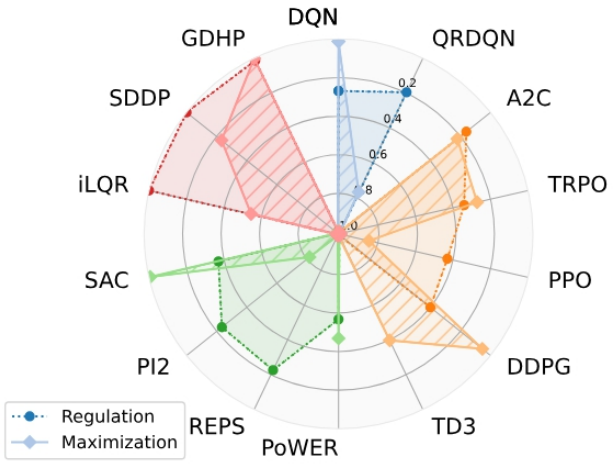

Fig. 4. Average cost of each algorithm for both regulation and product maximization problems.

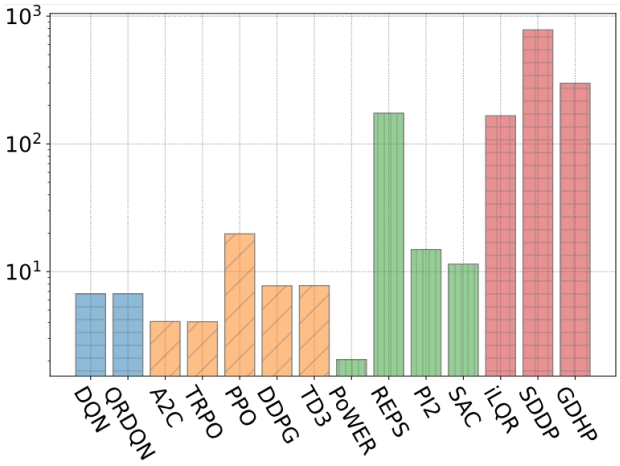

Fig. 5. Comparison of episodic computation time across the implemented algorithms on a log scale.

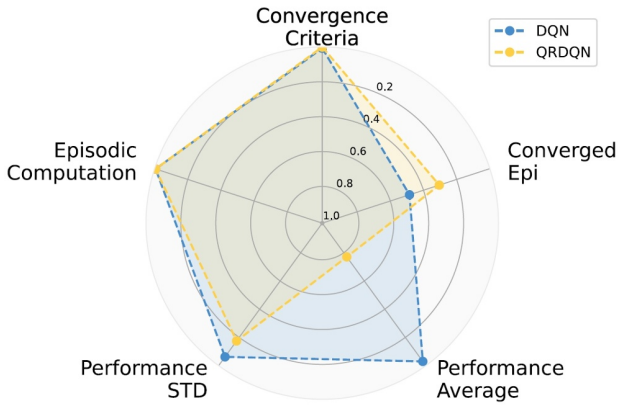

Fig. 6. Radar chart illustrating the performance metrics of value-based methods.

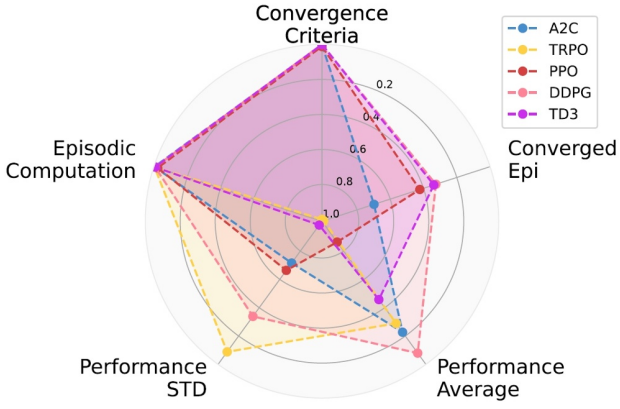

Fig. 7. Radar chart illustrating the performance metrics of policy-based methods.

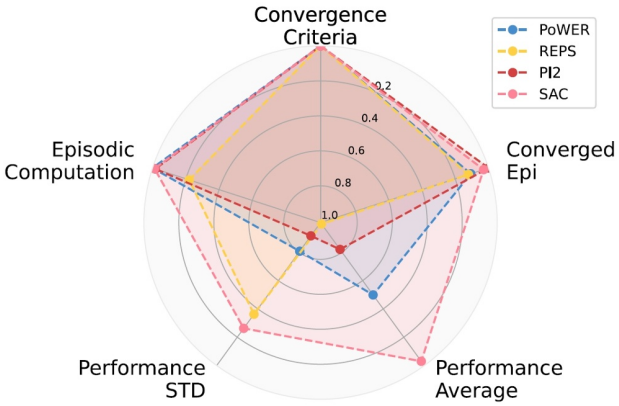

Fig. 8. Radar chart illustrating the performance metrics of inference-based methods.

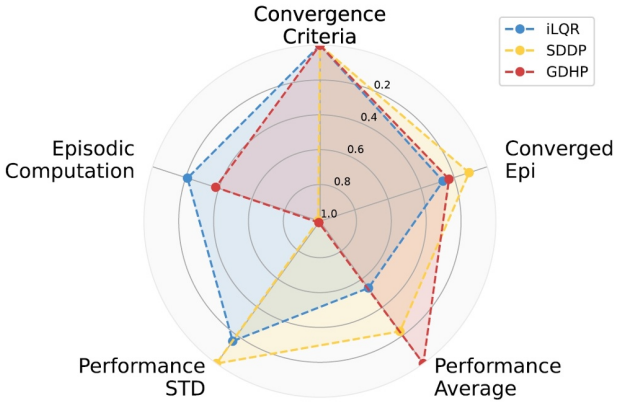

Fig. 9. Radar chart illustrating the performance metrics of model-based methods.

7. FUTURE RESEARCH DIRECTIONS

It is essential to study how to formulate control problems into appropriate MDPs for applying RL to process control. In conventional model-based process control methods, defining the system's state variables is straightforward. When applying RL, however, a user-specific state space must be defined, and the choice of variables can significantly influence both performance and efficiency. For systems with time delays, for example, the state space may need to be augmented with past histories to satisfy the Markov property, or a time-sequence-specific approximator, such as RNNs, may be required [195].

Moreover, the design of the reward function has a crucial impact on learning performance [196], because it is the primary means by which the agent and user interact in the MDP. Further research is required to determine appropriate reward structures that align with the objectives of process control. Approaches such as apprenticeship learning [197] or inverse RL [198,199] could offer promising directions for defining optimal rewards.

In the practical application of RL to chemical and biomanufacturing processes, safety considerations are paramount [200,201]. RL requires sufficient exploration to learn optimal policies, but uncontrolled exploration in real environments poses significant risks, including safety hazards and potential economic losses. Research into methods that enable necessary exploration within a safe range is critical. Additionally, RL's inability to explicitly handle safety constraints presents a limitation. To address this, research should explore integrating mathematical programming techniques to incorporate constraint satisfaction during the control process.

Offline RL algorithms, which have gained traction recently, offer another promising approach for safe learning [202-204]. By utilizing historical operational data, offline RL enables optimal policy learning without direct interaction with the environment, thus enhancing safety [205]. However, given its limited exploration capability, it is important to consider fine-tuning the learned policy online to further improve performance [206,207].

The integration of well-established control theory and methods with RL algorithms is critical for designing controllers that deliver strong performance. To enhance the credibility of RL methods, it is essential to analyze the convergence and learning stability of RL algorithms. In particular, for algorithms that employ DNNs, establishing a mathematical foundation is necessary to understand the conditions required for stable learning. Additionally, closed-loop stability of RL-driven controllers should be analyzed mathematically to ensure performance guarantees.

Further, research should focus on integrating RL with conventional control methods, such as MPC, to address the limitations of each approach while leveraging their respective strengths. Both conventional control techniques and modern RL algorithms have distinct advantages and drawbacks. By synthesizing these methodologies into complementary algorithms, it is possible to design more effective and robust controllers.

Finally, research is needed to develop practical guidelines for RL implementation. For instance, guidelines on selecting neural network architectures, learning rate schedules, optimizers, and other design choices should be tailored to the specific characteristics of each environment. Moreover, methods for tuning algorithm-specific hyperparameters and analyzing hyperparameter sensitivity are critical for comparative assessments of new and existing algorithms. Although such guidelines may seem minor, they are expected to play a pivotal role in guiding future researchers toward more effective RL applications.

8. CONCLUSION

Reinforcement learning (RL) holds immense potential for advancing process control, offering adaptive and autonomous solutions to complex control challenges. In this

review, we introduced the foundational concepts of RL, categorized key approaches, and demonstrated the implementation of select RL algorithms across benchmark problems in chemical and biological engineering, supported by numerical analyses. Additionally, we discussed key considerations for future research, highlighting the opportunities and challenges in integrating RL into the process control domain. While significant hurdles remain, RL's potential to enhance the adaptability, efficiency, and intelligence of process control systems makes it a promising direction for future innovation.

CONFLICT OF INTEREST

The authors declare that there is no competing financial interest or personal relationship that could have appeared to influence the work reported in this paper. Jong Min Lee is a Senior Editor of International Journal of Control, Automation, and Systems. Senior Editor status has no bearing on editorial consideration.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT press, 2018.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al*., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484-489, 2016.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al*., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354-359, 2017.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al*., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al*., "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350-354, 2019.

[6] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238-1274, 2013.

[7] J. Kim, D. Jang, and H. J. Kim, "Distributed multi-agent target search and tracking with gaussian process and reinforcement learning," *International Journal of Control, Automation, and Systems*, vol. 21, no. 9, pp. 3057-3067, 2023.

[8] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909-4926, 2021.

[9] H. D. Nguyen and K. Han, "Safe reinforcement learning-based driving policy design for autonomous vehicles on highways," *International Journal of Control, Automation, and Systems*, vol. 21, no. 12, pp. 4098-4110, 2023.

[10] A. Coronato, M. Naeem, G. De Pietro, and G. Paragliola, "Reinforcement learning for intelligent healthcare applications: A survey," *Artificial Intelligence in Medicine*, vol. 109, 101964, 2020.

[11] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Science Advances*, vol. 4, no. 7, eaap7885, 2018.

[12] J. Shin, T. A. Badgwell, K.-H. Liu, and J. H. Lee, "Reinforcement learning - Overview of recent progress and implications for process control," *Computers & Chemical Engineering*, vol. 127, pp. 282-294, 2019.

[13] S. Spielberg, A. Tulsyan, N. P. Lawrence, P. D. Loewen, and R. Bhushan Gopaluni, "Toward self-driving processes: A deep reinforcement learning approach to control," *AIChE journal*, vol. 65, no. 10, e16689, 2019.

[14] R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," *Computers & Chemical Engineering*, vol. 139, 106886, 2020.

[15] H. Yoo, H. E. Byun, D. Han, and J. H. Lee, "Reinforcement learning for batch process control: Review and perspectives," *Annual Reviews in Control*, vol. 52, pp. 108-119, 2021.

[16] T. H. Oh, "Quantitative comparison of reinforcement learning and data-driven model predictive control for chemical and biological processes," *Computers & Chemical Engineering*, vol. 181, 108558, 2024.

[17] N. P. Lawrence, S. K. Damarla, J. W. Kim, A. Tulsyan, F. Amjad, K. Wang, B. Chachuat, J. M. Lee, B. Huang, and R. B. Gopaluni, "Machine learning for industrial sensing and control: A survey and practical perspective," *Control Engineering Practice*, vol. 145, 105841, 2024.

[18] J. Hoskins and D. Himmelblau, "Process control via artificial neural networks and reinforcement learning," *Computers & Chemical Engineering*, vol. 16, no. 4, pp. 241-251, 1992.

[19] J. Wilson and E. Martinez, "Neuro-fuzzy modeling and control of a batch process involving simultaneous reaction and distillation," *Computers & Chemical Engineering*, vol. 21, pp. S1233-S1238, 1997.

[20] C. W. Anderson, D. C. Hittle, A. D. Katz, and R. M. Kretchmar, "Synthesis of reinforcement learning, neural networks and pi control applied to a simulated heating coil," *Artificial Intelligence in Engineering*, vol. 11, no. 4, pp. 421-429, 1997.

[21] E. C. Martínez, "Batch process modeling for optimization using reinforcement learning," *Computers & Chemical Engineering*, vol. 24, no. 2-7, pp. 1187-1193, 2000.

[22] T. I. Ahamed, P. N. Rao, and P. Sastry, "A reinforcement learning approach to automatic generation control," *Electric Power Systems Research*, vol. 63, no. 1, pp. 9-26, 2002.

[23] J. M. Lee and J. H. Lee, "Simulation-based learning of cost-to-go for control of nonlinear processes," *Korean Journal of Chemical Engineering*, vol. 21, pp. 338-344, 2004.

[24] J. M. Lee and J. H. Lee, "Approximate dynamic programming-based approaches for input-output data-driven control of nonlinear processes," *Automatica*, vol. 41, no. 7, pp. 1281-1288, 2005.

[25] J. H. Lee and J. M. Lee, "Approximate dynamic programming based approach to process control and scheduling," *Computers & Chemical Engineering*, vol. 30, no. 10-12, pp. 1603-1618, 2006.

[26] H. Nosair, Y. Yang, and J. M. Lee, "Min-max control using parametric approximate dynamic programming," *Control Engineering Practice*, vol. 18, no. 2, pp. 190-197, 2010.

[27] Y. Yang and J. M. Lee, "A switching robust model predictive control approach for nonlinear systems," *Journal of Process Control*, vol. 23, no. 6, pp. 852-860, 2013.

[28] S. Spielberg, R. Gopaluni, and P. Loewen, "Deep reinforcement learning approaches for process control," *Proc. of 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, IEEE, pp. 201-206, 2017.

[29] T. H. Oh, J. W. Kim, S. H. Son, H. Kim, K. Lee, and J. M. Lee, "Automatic control of simulated moving bed process with deep Q-network," *Journal of Chromatography A*, vol. 1647, 462073, 2021.

[30] Y. Bao, Y. Zhu, and F. Qian, "A deep reinforcement learning approach to improve the learning performance in process control," *Industrial & Engineering Chemistry Research*, vol. 60, no. 15, pp. 5504-5515, 2021.

[31] T. Joshi, S. Makker, H. Kodamana, and H. Kandath, "Twin actor twin delayed deep deterministic policy gradient (TATD3) learning for batch process control," *Computers & Chemical Engineering*, vol. 155, 107527, 2021.

[32] T. Joshi, H. Kodamana, H. Kandath, and N. Kaisare, "TASAC: A twin-actor reinforcement learning framework with a stochastic policy with an application to batch process control," *Control Engineering Practice*, vol. 134, 105462, 2023.

[33] J. Deng, S. Sierla, J. Sun, and V. Vyatkin, "Reinforcement learning for industrial process control: A case study in flatness control in steel industry," *Computers in Industry*, vol. 143, 103748, 2022.

[34] B. J. Pandian and M. M. Noel, "Control of a bioreactor using a new partially supervised reinforcement learning algorithm," *Journal of Process Control*, vol. 69, pp. 16-29, 2018.

[35] O. Dogru, N. Wieczorek, K. Velswamy, F. Ibrahim, and B. Huang, "Online reinforcement learning for a continuous space system with experimental validation," *Journal of Process Control*, vol. 104, pp. 86-100, 2021.

[36] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas *et al*., "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414-419, 2022.

[37] N. P. Lawrence, M. G. Forbes, P. D. Loewen, D. G. McClement, J. U. Backström, and R. B. Gopaluni, "Deep reinforcement learning with shallow controllers: An experimental application to pid tuning," *Control Engineering Practice*, vol. 121, 105046, 2022.

[38] O. Dogru, K. Velswamy, F. Ibrahim, Y. Wu, A. S. Sundaramoorthy, B. Huang, S. Xu, M. Nixon, and N. Bell, "Reinforcement learning approach to autonomous PID tuning," *Computers & Chemical Engineering*, vol. 161, 107760, 2022.

[39] L. Zhu, Y. Cui, G. Takami, H. Kanokogi, and T. Matsubara, "Scalable reinforcement learning for plant-wide control of vinyl acetate monomer process," *Control Engineering Practice*, vol. 97, 104331, 2020.

[40] J. W. Kim, B. J. Park, H. Yoo, T. H. Oh, J. H. Lee, and J. M. Lee, "A model-based deep reinforcement learning method applied to finite-horizon optimal control of nonlinear control-affine system," *Journal of Process Control*, vol. 87, pp. 166-178, 2020.

[41] J. W. Kim, T. H. Oh, S. H. Son, D. H. Jeong, and J. M. Lee, "Convergence analysis of the deep neural networks based globalized dual heuristic programming," *Automatica*, vol. 122, 109222, 2020.

[42] J. W. Kim, B. J. Park, T. H. Oh, and J. M. Lee, "Model-based reinforcement learning and predictive control for two-stage optimal control of fed-batch bioreactor," *Computers & Chemical Engineering*, vol. 154, 107465, 2021.

[43] J. W. Kim, T. H. Oh, S. H. Son, and J. M. Lee, "Primal-dual differential dynamic programming: A model-based reinforcement learning for constrained dynamic optimization," *Computers & Chemical Engineering*, vol. 167, 108004, 2022.

[44] K. Alhazmi, F. Albalawi, and S. M. Sarathy, "A reinforcement learning-based economic model predictive control framework for autonomous operation of chemical reactors," *Chemical Engineering Journal*, vol. 428, 130993, 2022.

[45] T. H. Oh, H. M. Park, J. W. Kim, and J. M. Lee, "Integration of reinforcement learning and model predictive control to optimize semi-batch bioreactor," *AIChE Journal*, vol. 68, no. 6, e17658, 2022.

[46] H. Yoo, V. M. Zavala, and J. H. Lee, "A dynamic penalty approach to state constraint handling in deep reinforcement learning," *Journal of Process Control*, vol. 115, pp. 157-166, 2022.

[47] F. Tang, Z. Feng, Y. Li, C. Yang, and B. Sun, "A constrained multi-objective deep reinforcement learning approach for temperature field optimization of zinc oxide rotary volatile kiln," *Advanced Engineering Informatics*, vol. 58, 102197, 2023.

[48] E. Pan, P. Petsagkourakis, M. Mowbray, D. Zhang, and E. A. del Rio-Chanona, "Constrained model-free reinforcement learning for process optimization," *Computers & Chemical Engineering*, vol. 154, 107462, 2021.

[49] P. Petsagkourakis, I. O. Sandoval, E. Bradford, F. Galvanin, D. Zhang, and E. A. del Rio-Chanona, "Chance constrained policy optimization for process control and optimization," *Journal of Process Control*, vol. 111, pp. 35-45, 2022.

[50] M. Mowbray, P. Petsagkourakis, E. A. del Rio-Chanona, and D. Zhang, "Safe chance constrained reinforcement learning for batch process control," *Computers & Chemical Engineering*, vol. 157, 107630, 2022.

[51] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, "Constrained differential dynamic programming: A primal-dual augmented Lagrangian approach," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 13371-13378, 2022.

[52] Y. Kim and J. M. Lee, "Model-based reinforcement learning for nonlinear optimal control with practical asymptotic stability guarantees," *AIChE Journal*, vol. 66, no. 10, e16544, 2020.

[53] Y. Kim and J. W. Kim, "Safe model-based reinforcement learning for nonlinear optimal control with state and input constraints," *AIChE Journal*, vol. 68, no. 5, e17601, 2022.

[54] Y. Kim and T. H. Oh, "Model-based safe reinforcement learning for nonlinear systems under uncertainty with constraints tightening approach," *Computers & Chemical Engineering*, vol. 183, 108601, 2024.

[55] S. Hwangbo and G. Sin, "Design of control framework based on deep reinforcement learning and monte-carlo sampling in downstream separation," *Computers & Chemical Engineering*, vol. 140, 106910, 2020.

[56] M. Mowbray, R. Smith, E. A. Del Rio-Chanona, and D. Zhang, "Using process data to generate an optimal control policy via apprenticeship and reinforcement learning," *AIChE Journal*, vol. 67, no. 9, e17306, 2021.

[57] D. G. McClement, N. P. Lawrence, J. U. Backström, P. D. Loewen, M. G. Forbes, and R. B. Gopaluni, "Meta-reinforcement learning for the tuning of PI controllers: An offline approach," *Journal of Process Control*, vol. 118, pp. 139-152, 2022.

[58] D. Dutta and S. R. Upreti, "A reinforcement learning-based transformed inverse model strategy for nonlinear process control," *Computers & Chemical Engineering*, vol. 178, 108386, 2023.

[59] H. Hassanpour, P. Mhaskar, and B. Corbett, "A practically implementable reinforcement learning control approach by leveraging offset-free model predictive control," *Computers & Chemical Engineering*, vol. 181, 108511, 2024.

[60] K. Alhazmi and S. M. Sarathy, "Direct learning of improved control policies from historical plant data," *Computers & Chemical Engineering*, vol. 185, 108662, 2024.

[61] P. Petsagkourakis, I. O. Sandoval, E. Bradford, D. Zhang, and E. A. del Rio-Chanona, "Reinforcement learning for batch bioprocess optimization," *Computers & Chemical Engineering*, vol. 133, 106649, 2020.

[62] H.-E. Byun, B. Kim, and J. H. Lee, "Embedding active learning in batch-to-batch optimization using reinforcement learning," *Automatica*, vol. 157, 111260, 2023.

[63] H. Yoo, B. Kim, J. W. Kim, and J. H. Lee, "Reinforcement learning based optimal control of batch processes using monte-carlo deep deterministic policy gradient with phase segmentation," *Computers & Chemical Engineering*, vol. 144, 107133, 2021.

[64] J. M. Lee and J. H. Lee, "An approximate dynamic programming based approach to dual adaptive control," *Journal of Process Control*, vol. 19, no. 5, pp. 859-864, 2009.

[65] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, 2014.

[66] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34-37, 1966.

[67] D. Bertsekas, *Dynamic Programming and Optimal Control: Volume I*, vol. 4, Athena scientific, 2012.

[68] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, vol. 703, John Wiley & Sons, 2007.

[69] J.-M. Lee and J. H. Lee, "Approximate dynamic programming strategies and their applicability for process control: A review and future directions," *International Journal of Control, Automation, and Systems*, vol. 2, no. 3, pp. 263-278, 2004.

[70] G. A. Rummery and M. Niranjan, *On-line Q-learning Using Connectionist Systems*, vol. 37, University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[71] C. J. C. H. Watkins, *Learning from Delayed Rewards*, King's College, 1989.

[72] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279-292, 1992.

[73] M. G. Lagoudakis, R. Parr, and M. L. Littman, "Least-squares methods in reinforcement learning for control," *Proc. of Methods and Applications of Artificial Intelligence: Second Hellenic Conference on AI*, SETN 2002 Thessaloniki, Greece, pp. 249-260, 2002.

[74] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Online least-squares policy iteration for reinforcement learning control," *Proc. of the 2010 American Control Conference*, IEEE, pp. 486-491, 2010.

[75] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, no. 1, pp. 33-57, 1996.

[76] J. A. Boyan, "Least-squares temporal difference learning," *Proc. of International Conference on Machine Learning*, pp. 49-56, 1999.

[77] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107-1149, 2003.

[78] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, "A survey on modern trainable activation functions," *Neural Networks*, vol. 138, pp. 14-32, 2021.

[79] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92-108, 2022.

[80] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.

[81] S. Levine and V. Koltun, "Guided policy search," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1-9, 2013.

[82] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15084-15097, 2021.

[83] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1273-1286, 2021.

[84] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747, 2016.

[85] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121-2159, 2011.

[86] M. D. Zeiler, "Adadelta: An adaptive learning rate method," arXiv preprint arXiv:1212.5701, 2012.

[87] D. P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[88] J. M. Lee, N. S. Kaisare, and J. H. Lee, "Choice of approximator and design of penalty function for an approximate dynamic programming based control approach," *Journal of Process Control*, vol. 16, no. 2, pp. 135-156, 2006.

[89] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with gaussian processes," *Proc. of the 22nd International Conference on Machine Learning*, pp. 201-208, 2005.

[90] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, pp. 161-178, 2002.

[91] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[92] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," arXiv preprint arXiv:1506.02438, 2015.

[93] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229-256, 1992.

[94] I. H. Witten, "An adaptive optimal controller for discrete-time markov environments," *Information and Control*, vol. 34, no. 4, pp. 286-295, 1977.

[95] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834-846, 1983.

[96] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[97] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1995-2003, 2016.

[98] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," *Proc. of International Conference on Learning Representations*, 2018.

[99] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, pp. 3215-3222, 2018.

[100] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *Proc. of International Conference on Machine Learning*, PMLR, pp. 449-458, 2017.

[101] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[102] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1096-1105, 2018.

[103] D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, and T.-Y. Liu, "Fully parameterized quantile function for distributional reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[104] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1928-1937, 2016.

[105] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. De Freitas, "Sample efficient actor-critic with experience replay," arXiv preprint arXiv:1611.01224, 2016.

[106] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1407-1416, 2018.

[107] S. M. Kakade, "A natural policy gradient," *Advances in Neural Information Processing Systems*, vol. 14, 2001.

[108] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180-1190, 2008.

[109] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 949-980, 2014.

[110] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1889-1897, 2015.

[111] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[112] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[113] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," *Proc. of International Conference on Machine Learning*, PMLR, pp. 387-395, 2014.

[114] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1587-1596, 2018.

[115] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," arXiv preprint arXiv:1804.08617, 2018.

[116] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[117] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," *Proc. of the 24th International Conference on Machine Learning*, pp. 745-750, 2007.

[118] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Advances in Neural Information Processing Systems*, vol. 21, 2008.

[119] N. L. Roux, "Efficient iterative policy optimization," arXiv preprint arXiv:1612.08967, 2016.

[120] G. Neumann, "Variational inference for policy search in changing situations," *Proc. of the 28th International Conference on Machine Learning*, pp. 817-824, 2011.

[121] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," arXiv preprint arXiv:1806.06920, 2018.

[122] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, pp. 1607-1612, 2010.

[123] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137-3181, 2010.

[124] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[125] W. H. Montgomery and S. Levine, "Guided policy search via approximate mirror descent," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[126] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 3381-3388, 2017.

[127] E. Todorov, "Linearly-solvable Markov decision problems," *Advances in Neural Information Processing Systems*, vol. 19, 2006.

[128] M. Toussaint, "Robot trajectory optimization using approximate inference," *Proc. of the 26th Annual International Conference on Machine Learning*, pp. 1049-1056, 2009.

[129] R. Fox, A. Pakman, and N. Tishby, "Taming the noise in reinforcement learning via soft updates," arXiv preprint arXiv:1512.08562, 2015.

[130] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[131] B. O'Donoghue, R. Munos, K. Kavukcuoglu, and V. Mnih, "Combining policy gradient and Q-learning," arXiv preprint arXiv:1611.01626, 2016.

[132] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1352-1361, 2017.

[133] Y. Liu, P. Ramachandran, Q. Liu, and J. Peng, "Stein variational policy gradient," arXiv preprint arXiv:1704.02399, 2017.

[134] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *Proc. of International Conference on Machine Learning*, PMLR, pp. 1861-1870, 2018.

[135] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," arXiv preprint arXiv:1812.05905, 2018.

[136] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," arXiv preprint arXiv:1802.06070, 2018.

[137] D. E. Kirk, *Optimal Control Theory: An Introduction*, Courier Corporation, 2004.

[138] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85-95, 1966.

[139] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*, American Elsevier Publishing Company, 1970.

[140] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 4906-4913, 2012.

[141] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," *Proc. of the 2010 American Control Conference*, IEEE, pp. 1125-1132, 2010.

[142] A. Oshin, M. D. Houghton, M. J. Acheson, I. M. Gregory, and E. A. Theodorou, "Parameterized differential dynamic programming," arXiv preprint arXiv:2204.03727, 2022.

[143] H. Jung, J. W. Kim, and J. M. Lee, "Two-stage dynamic real-time optimization framework using parameter-dependent differential dynamic programming," *Computers & Chemical Engineering*, vol. 192, 108896, 2024.

[144] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," *Proc. of the 28th International Conference on Machine Learning (ICML-11)*, pp. 465-472, 2011.

[145] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[146] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497-528, 2009.

[147] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 7559-7566, 2018.

[148] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[149] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *Proc. of International Conference on Machine Learning*, PMLR, pp. 2555-2565, 2019.

[150] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[151] D. Jiang, E. Ekwedike, and H. Liu, "Feedback-based tree search for reinforcement learning," *Proc. of International Conference on Machine Learning*, PMLR, pp. 2284-2293, 2018.

[152] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *Machine Learning Proceedings 1990*, Elsevier, pp. 216-224, 1990.

[153] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160-163, 1991.

[154] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," arXiv preprint arXiv:1802.10592, 2018.

[155] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," *Proc. of Conference on Robot Learning*, PMLR, pp. 617-629, 2018.

[156] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma, "Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees," arXiv preprint arXiv:1807.03858, 2018.

[157] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[158] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," arXiv preprint arXiv:1803.00101, 2018.

[159] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[160] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253-279, 2013.

[161] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv preprint arXiv:1511.05952, 2015.

[162] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251-276, 1998.

[163] J. A. Bagnell and J. Schneider, "Covariant policy search," *Proc. of the 18th International Joint Conference on Artificial Intelligence*, pp. 1019-1024, 2003.

[164] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," arXiv preprint arXiv:1805.00909, 2018.

[165] R. M. Neal and G. E. Hinton, "A view of the em algorithm that justifies incremental, sparse, and other variants," *Learning in Graphical Models*, Springer, pp. 355-368, 1998.

[166] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[167] B. Eysenbach and S. Levine, "Maximum entropy RL (provably) solves some robust RL problems," arXiv preprint arXiv:2103.06257, 2021.

[168] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." *Proc. of the 23rd National Conference on Artificial Intelligence*, vol. 8, pp. 1433-1438, USA, 2008.

[169] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," arXiv preprint arXiv:1507.04888, 2015.

[170] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose bayesian inference algorithm," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[171] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft Q-learning," arXiv preprint arXiv:1704.06440, 2017.

[172] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1-12, 2010.

[173] S. H. Son, J. J. Han, and J. M. Lee, "Modeling of the polymerization of linear monomers in the presence of multifunctional units," *Polymer*, vol. 126, pp. 74-86, 2017.

[174] M.-J. Kim, H.-J. Cho, and C.-G. Kang, "Mathematical modeling and analysis of a piston air compressor of a railway vehicle for abnormal data generation," *International Journal of Control, Automation, and Systems*, vol. 22, no. 2, pp. 360-372, 2024.

[175] H. Kim, H. Chang, and H. Shim, "Evaluating mr-gpr and mr-nn: An exploration of data-driven control methods for nonlinear systems," *International Journal of Control, Automation, and Systems*, vol. 22, pp. 2934-2941, 2024.

[176] D. M. Himmelblau, "Applications of artificial neural networks in chemical engineering," *Korean Journal of Chemical Engineering*, vol. 17, pp. 373-392, 2000.

[177] M. Fairbank and E. Alonso, "Value-gradient learning," *Proc. of International Joint Conference on Neural Networks*, IEEE, pp. 1-8, 2012.

[178] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667-682, 1999.

[179] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733-764, 2003.

[180] J. H. Lee, "Model predictive control: Review of the three decades of development," *International Journal of Control, Automation, and Systems*, vol. 9, pp. 415-424, 2011.

[181] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, pp. 19-67, 2005.

[182] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," *Proc. of International Conference on Computers and Games*, Springer, pp. 72-83, 2006.

[183] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1-43, 2012.

[184] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," *Proc. of the 19th International Conference on Machine Learning*, pp. 267-274, 2002.

[185] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344-357, 2017.

[186] M. Kazim, J. Hong, M.-G. Kim, and K.-K. K. Kim, "Recent advances in path integral control for trajectory optimization: An overview in theoretical and algorithmic perspectives," *Annual Reviews in Control*, vol. 57, 100931, 2024.

[187] S. Bae, T. H. Oh, J. W. Kim, Y. Kim, and J. M. Lee, "Integrating path integral control with backstepping control to regulate stochastic system," *International Journal of Control, Automation, and Systems*, vol. 21, no. 7, pp. 2124-2138, 2023.

[188] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits and Systems Magazine*, vol. 9, no. 3, pp. 32-50, 2009.

[189] H. Chen, A. Kremling, and F. Allgöwer, "Nonlinear predictive control of a benchmark CSTR," *Proc. of 3rd European Control Conference*, pp. 3247-3252, 1995.

[190] S. Lucia, J. A. Andersson, H. Brandt, M. Diehl, and S. Engell, "Handling uncertainty in economic nonlinear model predictive control: A comparative case study," *Journal of Process Control*, vol. 24, no. 8, pp. 1247-1259, 2014.

[191] G. Birol, C. Ündey, and A. Cinar, "A modular simulation package for fed-batch fermentation: penicillin production," *Computers & Chemical Engineering*, vol. 26, no. 11, pp. 1553-1565, 2002.

[192] A. Mesbah, J. Landlust, A. Huesman, H. Kramer, P. Jansens, and P. Van den Hof, "A model-based control framework for industrial batch crystallization processes," *Chemical Engineering Research and Design*, vol. 88, no. 9, pp. 1223-1233, 2010.

[193] R. Parker, B. Nicholson, J. Siirola, C. Laird, and L. Biegler, "An implicit function formulation for optimization of discretized index-1 differential algebraic systems," *Computers & Chemical Engineering*, vol. 168, 108042, 2022.

[194] K. A. Hoo and D. Zheng, "Low-order control-relevant models for a class of distributed parameter systems," *Chemical Engineering Science*, vol. 56, no. 23, pp. 6683-6710, 2001.

[195] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," arXiv preprint arXiv:1904.12901, 2019.

[196] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," *Proc. of the 16th International Conference on Machine Learning*, pp. 278-287, 1999.

[197] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Proc. of the twenty-first international conference on Machine learning*, 2004.

[198] A. Y. Ng, S. Russell *et al.*, "Algorithms for inverse reinforcement learning," *Proc. of International Conference on Machine Learning*, vol. 1, no. 2, 2000.

[199] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[200] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437-1480, 2015.

[201] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411-444, 2022.

[202] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179-1191, 2020.

[203] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20132-20145, 2021.

[204] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit Q-learning," arXiv preprint arXiv:2110.06169, 2021.

[205] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," arXiv preprint arXiv:2005.01643, 2020.

[206] S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin, "Offline-to-online reinforcement learning via balanced replay and pessimistic Q-ensemble," *Proc. of Conference on Robot Learning*, PMLR, pp. 1702-1712, 2022.

[207] M. Nakamoto, S. Zhai, A. Singh, M. Sobol Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine, "Cal-QL: Calibrated offline rl pre-training for efficient online fine-tuning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

Joonsoo Park received his B.Sc. and M.Sc. degrees from the School of Chemical and Biological Engineering, Seoul National University, Seoul, Korea, in 2021 and 2023, respectively. He is currently pursuing a Ph.D. degree in the School of Chemical and Biological Engineering at Seoul National University. His research interests include advanced process control, reinforcement learning, and optimization.

Hyein Jung received her B.Sc. degree from the School of Chemical and Biological Engineering, Seoul National University, Seoul, Korea, in 2019. She is currently pursuing a Ph.D. degree in the School of Chemical and Biological Engineering, Seoul National University. Her research interests include model predictive control, dynamic programming, and system identification.

Jong Woo Kim is an Assistant Professor in the Department of Energy and Chemical Engineering at Incheon National University, Incheon, Korea. He obtained his B.Sc. and Ph.D. degrees from the School of Chemical and Biological Engineering, Seoul National University, Seoul, Korea, in 2014 and 2020, respectively. He was a postdoctoral researcher in Chair of Bioprocess Engineering, Technische Universität Berlin, Berlin, Germany, from 2020 to 2022. His research interests are in the areas of process automation and development, including reinforcement learning, stochastic optimal control, and high-throughput bioprocess development.

Jong Min Lee is a Professor in the School of Chemical and Biological Engineering at Seoul National University, SNU, Seoul, Korea. From September 2016 to August 2017, he was a Visiting Associate Professor in the Department of Chemical Engineering at MIT. He also held the Samwha Motors Chaired Professorship from 2015 to 2017. He obtained his B.Sc. degree in chemical engineering from SNU, in 1996 and completed his Ph.D. degree in chemical engineering at Georgia Institute of Technology, Atlanta, US, in 2004. He also held a research associate position in biomedical engineering at the University of Virginia, Charlottesville, US, from 2005 to 2006. He was an assistant professor of chemical and materials engineering at University of Alberta, Edmonton, Canada, from 2006 before joining SNU in 2010. He is also a registered professional engineer with APEGA in Alberta, Canada. His current research interests include modeling, control, and optimization of large-scale chemical process, energy, and manufacturing systems with uncertainty and reinforcement learning-based control.