Title

Rule-based Policy Regularization for Reinforcement Learning-based Building Control

Authors

Liu, Hsin-Yu
Balaji, Bharathan
Gupta, Rajesh
et al.

Publication Date

2023-06-20

DOI

10.1145/3575813.3595202

Rule-based Policy Regularization for Reinforcement Learning-based Building Control

Hsin-Yu Liu*
hyl001@eng.ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Bharathan Balaji**
bhabalaj@amazon.com
Amazon
USA

Rajesh Gupta
gupta@ucsd.edu
University of California, San Diego
La Jolla, CA, USA

Dezhi Hong**
hondezhi@amazon.com
Amazon
USA

ABSTRACT

Rule-based control (RBC) is widely adopted in buildings due to its stability and robustness. It resembles a behavior cloning methodology refined by human experts; however, it is incapable of adapting to distribution drifts. Reinforcement learning (RL) can adapt to changes but needs to learn from scratch in the online setting. On the other hand, the learning ability is limited in offline settings due to extrapolation errors caused by selecting out-of-distribution actions. In this paper, we explore how to incorporate RL with a rule-based control policy to combine their strengths to continuously learn a scalable and robust policy in both online and offline settings. We start with representative online and offline RL methods, TD3 and TD3+BC, respectively. Then, we develop a dynamically weighted actor loss function to selectively choose which policy for RL models to learn from at each training iteration. With extensive experiments across various weather conditions in both deterministic and stochastic scenarios, we demonstrate that our algorithm, rule-based incorporated control regularization (RUBICON), outperforms state-of-the-art methods in offline settings by 40.7% and improves the baseline method by 49.7% in online settings with respect to a reward consisting of thermal comfort and energy consumption in building-RL environments.

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**.

ACM Reference Format:
Hsin-Yu Liu*, Bharathan Balaji**, Rajesh Gupta, and Dezhi Hong**. 2023. Rule-based Policy Regularization for Reinforcement Learning-based Building Control. In *The 14th ACM International Conference on Future Energy Systems (e-Energy '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3575813.3595202

*Corresponding author.
** Work unrelated to Amazon.

e-Energy '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0032-3/23/06.
https://doi.org/10.1145/3575813.3595202

1 INTRODUCTION

Buildings typically implement rule-based control, which adjusts the setpoints of actuators to co-optimize occupants' thermal comfort and energy efficiency. These rule-based control (RBC) systems codify the problem-solving know-how of human experts, akin to behavioral cloning policy learned from expert demonstration [13]. RBC is stable, robust, and without uncertainty, but lacks the flexibility to evolve over time.

Reinforcement learning (RL) can adapt to changes in the environment with a data-driven approach and improves the performance of HVAC systems control [45]. In online RL, the training of the control policy relies on a simulator that models the HVAC system. We use established building-RL simulation environments – *Sinergym* [17] for our experiments. However, when such a simulation model is not available, offline RL can be used to train a policy based on historical data [50]. We focus on improving upon existing RL algorithms for HVAC control where a rule-based policy already exists, which is a common scenario in real-world implementations. By combining the advantages of RL and rule-based methods, we aim to develop a stable and scalable algorithm without learning from scratch and utilize the existing knowledge.

In our work, we seek to answer the following research questions: ***How can we incorporate reinforcement learning models with an existing rule-based control policy to improve models' performance? Could this method be implemented in both online and offline settings as a unified approach?***

RL regularization methods are typically tailored specifically to online or offline settings. For example, online methods encourage exploration to either improve estimations of non-greedy actions' values or to encourage the exploration to find an optimal policy [11]. On the other hand, offline methods favor exploitation since it is unlikely for offline models to accurately estimate uncharted state-action values with a static dataset [10, 46].

Our method builds on TD3+BC [8], a representative offline RL algorithm. TD3+BC makes minimal changes by adding a behavior cloning term to regularize the online TD3 [9] policy. In TD3+BC, the only policy to learn from is the behavioral policy that generates the buffer. Our dynamically-weighted algorithm regularizes RL policy using the better policy between an existing RBC policy and the behavioral policy. It can be incorporated into any existing actor-critic RL algorithms with minimal changes.

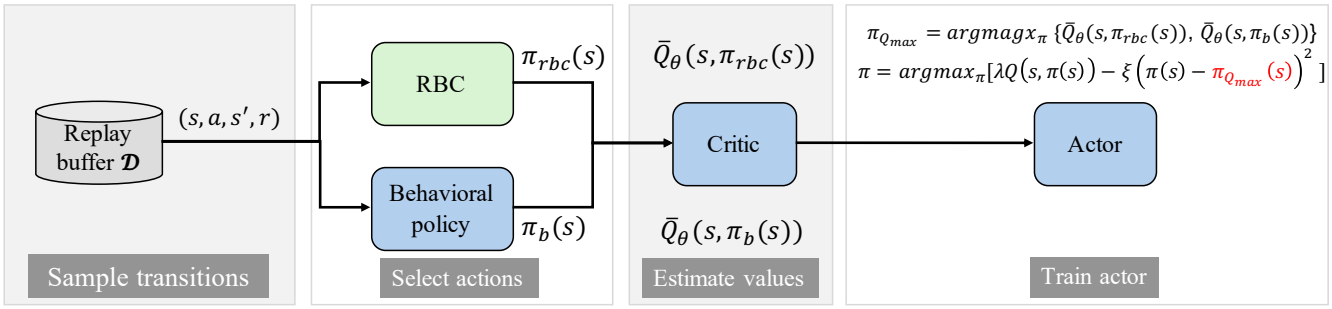

Figure 1: The flow of RUBICON: We incorporate the RBC policy and selectively update the actor with the policy (between RBC and behavioral) that has a higher estimated mean Q-value. It is a unified method for both online and offline approaches.

RUBICON considers RBC as a safe reference policy in which RL training can learn and improve. The actor selectively trains on either RBC or behavioral policy, depending on which policy yields a higher averaged Q-value in a mini-batch estimated by the critic network. The flow of RUBICON is shown in Fig. 1. Our proposed approach is distinct from prior work in the following aspects:

- We develop a unified regularization approach for both online and offline RL methods with minimal algorithmic modification.
- Rule-based control policy is directly incorporated into the policy update step to provide stability and robustness.
- We introduce a dynamic weighting method in actor-critic settings. The actor loss is varied from time step to time step depending on the average Q-value estimate of behavioral policy and RBC policy predicted from the value networks.

To our knowledge, previously RBC is only used as hard constraints or heuristics in RL settings, and we are the first to incorporate an existing RBC policy directly into actor-critic algorithms.

2 RELATED WORK

Building RL control Prior research has demonstrated that building RL control policy could outperform RBC in both online and offline settings. Researchers have studied extensively for HVAC control with online RL methods [12, 44, 48]. Zhang et al. [51] developed a framework for whole building HVAC (heating, ventilation, air-conditioning) control in online settings to achieve a 16.7% heating demand reduction cf. RBC control. OCTOPUS holistically controls subsystems in modern buildings to get a 14.26% energy saving cf. RBC policy [6]. Yang et al. [47] implemented an RL control for LowEx building systems with a 11.47% improvement on cumulative net power output than RBC.

With offline RL, Zhang et al. [50] applied a state-of-the-art method and demonstrated a 12 ∼ 35% of reduction in ramping. Liu et al. [26] incorporated a Kullback-Leibler (KL) divergence constraint during the training of an offline RL agent to penalize policies that are far away from the previous updates for stability, and achieve a 16.7% of energy reduction cf. the default RBC control.

RL + RBC The combination of RL and RBC has been explored in many studies, where RBCs are primarily used as auxiliary constraints or guiding mechanism. Lee et al. [23] propose to use two modules in their control flow, one for continuous control with RL agent and a discrete one controlled by RBC. Wang et al. [43] improve RL with low-level rule-based trajectory modification to achieve a safe and efficient lane-change behavior. Zhu et al. [52] incorporate RBC for generating the closed-loop trajectory and reducing the exploration space for RL pre-processing. Berenji [4] use a learning process to fine-tune the performance of a rule-based controller. Radaideh and Shirvan [30] first train RL proximal policy optimization (PPO) [32] agents to master matching some of the problem rules and constraints, then RL is used to inject experiences to guide various evolutionary/stochastic algorithms. Likmeta et al. [24] learn RBC parameters via RL methods. These previous methods incorporate RBC in the flow as heuristics or as hard constraints. Instead, we directly incorporate RBC policy in RL training in an algorithmic way.

Online RL regularization The online baseline we compare to in the evaluation is a state-of-the-art algorithm: TD3. It applies target policy smoothing regularization to avoid overfitting in the value estimate with deterministic policies. TRPO [31] uses a trust region constraint based on KL-divergence between old and new policy distributions for robust policy updates. SAC [11] uses soft policy iteration for learning optimal maximum entropy policies. Munchausen-RL [42] regularizes policy updates with a KL-divergence penalty similar to TRPO, and adds a scaled entropy term to penalize policy that is far from uniform policy.

Offline RL regularization Offline RL is more conservative compared with online methods as it depends only on the logged interactions generated by unknown policies. It suffers from extrapolation errors induced by selecting out-of-distribution actions. Since offline RL policies are learned entirely from a static dataset, it is unlikely for value networks to accurately estimate values when there is no sufficient state-action visitation. Thus, regularization methods become more prominent in offline settings. Batch-constrained deep Q-learning (BCQ) [10], one of the pioneers of offline RL, ascribes extrapolation errors to three main factors: absent data, model bias, and training mismatch. It mitigates the errors by deploying a variational autoencoder (VAE) to reconstruct the action given a state using the data collected by the behavioral policy. Then regularize the divergence between the learned policy and the behavioral policy. The offline baseline method we compare to in our study is TD3+BC. It starts from the online method TD3, and adds a behavior cloning term in the policy update to regularize the actor to imitate the behavioral policy and avoid selecting out-of-distribution

actions. BRAC [46] studies both value penalty and policy regularization with multiple divergence metrics (KL, maximum mean discrepancy (MMD), and Wasserstein) to regularize the actor's policy based on the behavioral policy. FisherBRC [20] incorporates a gradient penalty regularizer for the state-action value offset term and demonstrates the equivalence to Fisher divergence regularization. CQL [21] learns a conservative, lower-bound estimate in the value network via regularizing Q-values. Model-based method, e.g. COMBO [49] regularizes the value function on out-of-support transitions generated via environment dynamic models' rollouts.

Conservative RL Turchetta et al. [40] use a priori unknown safety constraint that depends on state-action and satisfies certain regularity conditions with a Gaussian prior. Alshiekh et al. [3] propose to synthesize a reactive system called a *shield* to monitor the actions and correct them if violations are caused. Sui et al. [37] use a pre-specified "safety" threshold as a requirement and express it via a Gaussian process prior.

All of these prior works use data collected by a behavioral policy and do not assume access to any existing policy. The behavioral policy used in experiments is typically an unknown or partially trained agent. In contrast, we assume direct access to a robust behavioral policy in the form of rule-based control. While this assumption may not hold for other applications where there might not be pre-existing policies, rule-based control policies are routinely deployed in industrial control settings, such as building HVAC control.

We incorporate a robust reference policy derived by human experts to improve RL policy. The rule-based control policy reduces uncertainty due to its deterministic behavior. On the opposite, the deep learning model is affected by random initialization conditions, even if trained on the same dataset, as varied initialization conditions might lead to different policies. RUBICON demonstrates a substantial reduction of standard deviations between different randomly initialized conditions across varied tasks.

3 BACKGROUND

In reinforcement learning, an agent interacts with the environment and sequentially selects actions based on its policy at every time step. The problem can be formulated as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R}$, transition dynamics p, and discount factor $\gamma \in [0, 1)$. The goal is to maximize the expectation of the cumulative discounted rewards, denoted by $R_t = \sum_{i=t+1}^{\infty} \gamma^i r(s_i, a_i, s_{i+1})$ [38]. The agent's behavior is determined by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which maps states to actions either in a deterministic approach or with a probability distribution. The expected return following the policy from a given state s is the action-value function $Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | s_0 = s, a_0 = a]$ by taking action a.

We conduct our experiments with the building RL environments [17]. The objective of the agent is to maintain a comfortable thermal environment with minimal energy use. The state consists of indoor/outdoor temperatures, time/day, occupant count, thermal comfort, and related sensor data. The action adjusts the temperature setpoint of the thermostat. The reward is a linear combination of occupants' thermal comfort and energy consumption. The environment is a single-floor building divided into 5 zones, with 1 interior and 4 exterior rooms.

The details about the RL settings in our problem are described below:

- **State**: Site outdoor air dry bulb temperature, site outdoor air relative humidity, site wind speed, site wind direction, site diffuse solar radiation rate per area, site direct solar radiation rate per area, zone thermostat heating setpoint temperature, zone thermostat cooling setpoint temperature, zone air temperature, zone thermal comfort mean radiant temperature, zone air relative humidity, zone thermal comfort clothing value, zone thermal comfort Fanger model PPD (predicted percentage of dissatisfied), zone people occupant count, people air temperature, facility total HVAC electricity demand rate, current day, current month, and current hour.
- **Action**: Heating setpoint and cooling setpoint in continuous settings for the interior zones.
- **Reward**: We follow the default linear reward setting, which considers the energy consumption and the absolute difference to temperature comfort.
- **Environment**: A single floor building with an area of $463.6m^2$ divided into 5 zones, 1 interior, and 4 exteriors. The HVAC system is a packaged VAV (variable air volume) (DX (direct expansion) cooling coil and gas heating coils) with fully autosized input. And the simulation period of one episode is a full year. The weather types are classified according to the U.S. Department of Energy (DOE) standard [28]. The weather type details and their representative geometric locations are listed below based on TMY3 datasets [22]:
  - **Cool marine**: Washington, USA. The mean annual temperature and mean annual relative humidity are 9.3°C and 81.1% respectively.
  - **Hot dry**: Arizona, USA with mean annual temperature of 21.7°C and a mean annual relative humidity of 34.9%
  - **Mixed humid**: New York, USA with a mean annual temperature of 12.6°C and a mean annual relative humidity of 68.5%

4 RULE-BASED INCORPORATED CONTROL REGULARIZATION

Our goal is to improve an agent's ability to learn with the assistance of human experts' domain knowledge in both online and offline settings. In certain problems, we could configure accurate simulators as oracles so we can safely learn with online RL methods or there might be existing simulators. For example, in robotic control [39], Go [35], and video games [27]. However, for most real-world problems, it is unlikely or it is time-consuming and requires a domain expertise to build a functional simulator for each environment (e.g. building thermal simulators), or it can be dangerous or risky to evaluate partially trained policy directly in real environments(e.g. healthcare and financial trading). Offline RL algorithms, on the other hand, rely on historical data collected by an existing but unknown behavioral policy. The objective is to learn a policy that improves on the behavioral policy measured by episodic rewards. In

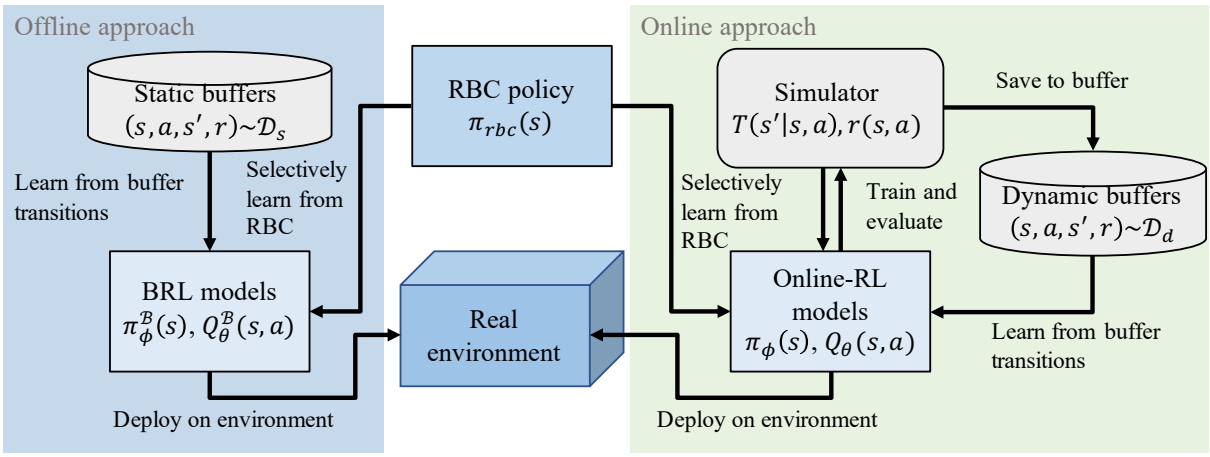

Figure 2: Our proposed method, RUBICON, incorporates RBC into RL to improve stability in building HVAC control. It could be applied to both online and offline approaches.

Fig. 2, we illustrate how RUBICON accommodates both the online and offline training paradigms.

Our algorithm builds on existing actor-critic algorithms TD3 and TD3+BC. We only modify the policy update with the incorporated rule-based control policy selectively and use the critic as-is. Therefore, we focus our discussion on the policy update of the algorithm. TD3 is derived from DDPG [34], it mitigates the function approximation error with double Q-learning and delayed policy updates. TD3+BC is an offline RL algorithm adapted from TD3, and is one of the state-of-the-art offline RL methods evaluated with D4RL datasets [7]. TD3+BC adds a behavior cloning term to the policy update step to penalize the policy that is far away from the behavioral policy (Eq. 1). The blue-colored terms indicate the changes from TD3 to TD3+BC.

$$\pi = \arg\max_{\pi} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\lambda Q(s,\pi(s)) - (\pi(s)-a)^2\right] \quad (1)$$

$$\lambda = \frac{\alpha}{\frac{1}{N}\sum_{(s_i,a_i)} |Q(s_i,a_i)|} \quad (2)$$

In Eq. 1, λ is decided by the averaged mini-batch Q-estimate and a hyperaparameter α to adjust between RL and imitation learning (Eq. 2), where N is the size of the batch.

Our method, RUBICON, dynamically weighs both TD3 and TD3+BC's policy update steps with either RBC policy or behavioral policy in each training iteration. In Eq. 3, we replace the actions a sampled from the buffers in Eq. 1 with $\pi_{Q_{max}}(s)$ and add a hyperparameter ξ to integrate TD3 and TD3+BC methods as one. Red-colored terms (in Eq. 3) indicate the changes from TD3 and TD3+BC to our method. We replace the notation of sampled actions a in TD3+BC with behavioral policy $\pi_b(s)$ to avoid confusion. Details of the hyperparameter settings in our work are in Appendix C.

$$\pi = \arg\max_{\pi} \mathbb{E}_{s\sim\mathcal{D}} \left[\lambda Q(s,\pi(s)) - \xi(\pi(s) - \pi_{Q_{max}}(s))^2\right] \quad (3)$$

$$\pi_{Q_{max}}(s) = \arg\max_{\pi} \{\bar{Q}(s,\pi_b(s)), \bar{Q}(s,\pi_{rbc}(s))\} \quad (4)$$

Every time when the policy is being updated, given the states s of the sampled mini-batch, the behavioral policy $\pi_b(s)$ and the RBC policy $\pi_{rbc}(s)$ select actions in a deterministic fashion. The state-action pairs' Q-values are estimated by the critic, the average of the Q-value estimations in the mini-batches are $\bar{Q}(s,\pi_b(s))$ and $\bar{Q}(s,\pi_{rbc}(s))$. The actor models dynamically choose the selected actions decided by the policy with a higher average Q-value to be regularized from in each policy update step, i.e. the actor loss function is dynamically weighted. By describing it as dynamically weighted, it means that the actor loss is changing from one iteration to another since it is automatically decided by Eq. 4. In offline settings the actor loss is either regularized with the behavioral policy or the RBC policy; in online settings, it is either regularized with the RBC policy or learning as is without behavioral cloning term. In online settings, the behavioral policies are the older versions of the policy used to generate the transitions in the buffer, and in offline settings, it is an unknown policy.

The reason we choose the average as the metric to decide which set of transitions to learn from instead of selecting each transition with higher estimated value (each batch is a combination of $\pi_b(s)$ and $\pi_{rbc}(s)$) is that if we choose by each transition we will lose the information on which state-action visitations lead to worse values, the model will then suffer from the imbalanced data problem. The credit-blame assignment is essential in RL learning convergence and the experience replay can help speed up the propagation process [25]. Furthermore, the over-estimation of Q-values would be more severe. The algorithm of our method is given in Alg. 1. Changes from baselines to our method are highlighted in blue. Where d is the policy update frequency, the noise ϵ added to the policy is sampled with Gaussian $\mathcal{N}(0,\sigma)$ and clipped by c. In both online and offline approaches, the policy update follows Eq. 3 and 4 with different hyperparameter settings.

Our rule-based control algorithm is described in Alg. 2. It is derived from the rule-based controller in Sinergym's [17] example. For the purpose of computation efficiency and to fit the batch settings in our algorithm, we vectorize the original RBC policy. The

rules are simple and intuitive, and could generalize well: First, we get the datetime information we need from the states. Then, we get the seasonal comfort temperature zone for every transition. If the indoor air temperature (IAT) is below the lower bound of the comfort zone, then we set both cooling and heating setpoints a degree higher (measured in Celcius degrees). On the opposite, if the IAT is above the upper bound of the comfort zone, then we set both the heating and cooling setpoints a degree lower than the current setpoints. Finally, we examine if the current datetime is in the office hours. If not, then the setpoints are set to be (18.33, 23.33) (°C) for the purpose of energy reduction since occupants' thermal comfort is not important in these time periods assuming zero occupancy.

Algorithm 1: RUBICON

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$,actor network π_ϕ, with random parameters θ_1, θ_2, ϕ, target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$, RBC policy π_{rbc}, and replay buffer or load buffer $\mathcal{B}$
for $t = 1$ *to* T **do**
  if *online* **then**
    Select action with exploration noise
    $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$
    Observe reward r and next state s'
    Store transition (s, a, r, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{j=1,2} Q_{\theta'_j}(s', \tilde{a})$
  Update critics $\theta_j \leftarrow \arg\min_{\theta_j} N^{-1} \sum (y - Q_{\theta_j}(s, a))^2$
  if t *mod* d **then**
    Update ϕ by policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Policy update follows Eq. 3 and 4
    Calculate $\nabla_\phi J(\phi)$
    Update target networks:
    $\theta'_j \leftarrow \tau\theta_j + (1 - \tau)\theta'_j$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

Algorithm 2: Rule-based control policy

Input : Current datetime $current\_dt$, indoor air temperature IAT, zone thermostat heating setpoint temperature a_h, and zone thermostat cooling setpoint temperature a_c, obtained from the states with size N
Output : Actions selected by RBC
for i *in* N **do**
  $season\_comfort\_zone_i = get\_season\_comfort(current\_dt_i)$
  if $IAT_i >= max(season\_comfort\_zone_i)$ **then**
    $a_{h_i} = a_{h_i} - 1$
    $a_{c_i} = a_{c_i} - 1$
  if $IAT_i < min(season\_comfort\_zone_i)$ **then**
    $a_{h_i} = a_{h_i} + 1$
    $a_{c_i} = a_{c_i} + 1$
  $a_i = (a_{h_i}, a_{c_i})$
  if $current\_dt_i.weekday \geq 5$ *or* $current\_dt_i.hour$ *in range(22,6)* **then**
    $a_i = (18.33, 23.33)$(°C)

5 EXPERIMENTS

In our experiments, there are two environment response types: deterministic and stochastic. A Gaussian noise with μ=0 and σ=2.5 is added to the outside temperature from episode to episode in the stochastic environments. And three weather types: hot, cool, and mixed. In the results of all the tables and figures, "*hot-deterministic*" indicates that the task is learned and evaluated with the hot weather condition and deterministic environment. Similarly, we have all six combinations such as "*cool-stochastic*", etc. All scores in this paper are normalized with expert policy as 100 and random policy as 0.

5.1 Offline approach

5.1.1 Main Experiment. First, we consider the offline approach, where no simulator exists but historical data is available. We follow the standard procedure for BRL evaluation [7]:

(1.) Train behavioral agents for 500K time steps, then compare the most representative algorithms DDPG, TD3, and SAC (learning curves are shown in Fig. 12). The online methods we compare are described below:

- **DDPG**: Deep deterministic policy gradient is a method that combines the actor-critic approach and deep Q-network (DQN) [27]. It is capable of dealing with continuous action space problems via policy gradient in a deterministic approach which outperforms the stochastic policy methods in high-dimensional tasks.
- **SAC**: Soft actor-critic, an off-policy maximum entropy RL algorithm that encourages exploration. They empirically show that SAC yields a better sample efficiency than DDPG.
- **TD3**: Twin delayed deep deterministic policy gradient algorithm, it reduces overestimation with double Q-learning, combines with target networks to limit errors from imprecise function approximation.

(2.) Select the best agent as our expert agent and generate buffers with it for 500K time steps. A medium agent is trained "halfway", which means that an agent is trained most closely to an agent with the evaluation performance half the performance as the expert agent. And a random agent which samples actions randomly and generates buffers.

(3.) Train BRL models for 500K time steps and evaluate the policy every 25K time steps in all buffers mentioned above in step (2.). We show the detailed learning curves in Appendix B. Normalized and averaged scores across runs are shown in Table 1. The offline methods we compare with are listed below:

- **TD3+BC**: An offline version of TD3, it adds a behavior cloning term to regularize policy towards behavioral policy combined with mini-batch Q-values and buffer states normalization for stability improvement.
- **CQL**: Conservative Q-learning, derived from SAC, it learns a lower-bound estimate of the value function by regularizing the Q-values during training.
- **BCQ**: Batch-constrained deep Q-learning, it implements a variational autoencoder (VAE) [19] to reconstruct the action given the state. And adds perturbation in actor on the policy, the degree of perturbation and size of mini-batch can be adjusted in order to behave more like a traditional RL method or imitation learning.

- **BC**: Behavior cloning, we train a VAE to reconstruct action given state. It simply imitates the behavioral agent without reward signals.

In Table 1, we observe that RUBICON outperforms all other benchmarks in overall score across weather types, random seeds, and environment types. To breakdown the reward scores into the optimization objectives, RUBICON achieves an overall 10.11% of energy reduction and 34.44% in comfort penalty cf. to the state-of-the-art method CQL. Other BRL methods show good performance either in specific tasks or with a specific randomly initialized configuration; however, overall they are more unstable cf. RUBICON. Our method provides more robust and more consistent performance across all variants and demonstrates the ability to generalize across various weather types and response modes of tasks. Also, as we can see in Fig. 3, learning from both medium buffer and RBC policy, RUBICON improves on both their best performances. The standard deviation of RUBICON's scores is the least among the policies evaluated, which means our policy is the most stable one cf. others. We include the BRL learning curves with expert and random buffers in Appendix B

In the following sections, we conducted several robustness and ablation experiments to demonstrate the necessity of our enhancements.

5.1.2 Transfer experiment. In a real-world scenario where we might have existing building control data in one building, but no data for a new building with different weather data distribution. Given existing buffers and we want to use them as prior knowledge to combine with RBC policy and transfer the model to another weather type where we have no data. We experiment with the medium buffers in stochastic environments. The results shown in Table 5 and Fig. 4 indicate that our method is capable of transferring from one weather condition to another with comparable performance without any hyperparameter or RBC policy change. As the results demonstrate, due to the diversity of the mixed weather, RUBICON improves learning in cool and hot weather. On the other hand, transfer from monotonic weather conditions leads to worse returns.

5.1.3 Ablation Experiments.

RBC buffer experiments. To differentiate RUBICON from directly learning from RBC buffers, we conduct the experiment of BRL models learning from the RBC buffers, i.e. learns from a buffer generated by RBC policy. The learning curves are illustrated in Figure 5. The results in Table 6 indicate that even when learning with RBC buffers with RBC policy itself, RUBICON could still outperform RBC policy due to its learning ability. However, due to the monotony of RBC policy, the improvement is limited. It shows the importance of incorporating RBC policy and RL policy.

Mixed buffer experiments. In order to evaluate if mixing the buffers (of RBC buffer and the original buffer to learn from) is equivalent to RUBICON, we conduct experiments by mixing 50% of transitions in RBC buffer with 50% of transition in the original buffer to learn from. The result is shown in Figure 6 and Table 9. It indicates our selective algorithm is necessary to dynamically decide if RBC policy or the behavioral policy to learn from instead of randomly trained on both.

5.1.4 Robustness Experiments.

Data efficiency experiment. We conduct the experiments with buffers of only one year of data (35, 040 transitions). Data efficiency is a challenge for model-free RL to yield accurate value estimation as it is considered data inefficient generally. This experiment is designed to observe how RUBICON adapts in a scenario where there is insufficient data. In Table 2, we observe that our method still outperforms its baseline overall. Although it dominates with random buffers and has comparable performance with expert buffers, it does not learn well with medium buffers. The root cause is the similarity of the quality of actions between medium buffers and RBC policy, which causes the critic to misjudge which action to pick between them. However, RUBICON still outperforms the baseline in other two types of buffers since the value estimation differences between $(\pi_b(s), s)$ and $(\pi_{rbc}(s), s)$ are more distinguishable in these scenarios.

Policy analysis experiment. Since Q-value prediction is usually overestimated, we use immediate rewards as references to examine the quality of the inferences of Q-networks. We pre-train a reward model $R_\psi(s, a)$ using the data in the buffer to predict reward $\hat{r}$ given state s and selected action a with 200K iterations with the buffer as our training data. At each iteration of the policy update, we record the policy $\pi(s)$ and the predicted rewards in each batch, i.e., $\hat{r} = R_\psi(s, \pi(s))$. We plot the distributions of reward in action spaces in Fig. 7. It demonstrates that RUBICON selects the actions in a wider range cf. TD3+BC, nonetheless, with a reward distribution of higher values. The distribution shown is with 10% of data randomly selected from the entire training for better visualization.

5.2 Online approach

5.2.1 Main Experiment. In the online approach, it is assumed that an oracle exists for accurate simulations. In real-world applications, researchers train online models in simulation environments before deployment in real buildings. Experimental results comparing TD3 and our method can be found in Table 3. In five out of six tasks, our method outperforms TD3 in averaged scores and with a substantially smaller standard deviation across runs. The learning curves are illustrated in Fig. 8. In terms of the optimization objectives, RUBICON yields a 13.16% of energy reduction and 17.86% reduction in comfort penalty.

These results empirically show that our method strengthens the learning process not only in the offline approach but also in the online approach.

5.2.2 Hyperparameter experiment. Deep-RL is sensitive to hyperparameter tuning [2], thus, we keep the original neural network architectures and hyperparameter settings for a fair comparison. Since all the authors of these methods have optimized the hyperparameters across various tasks and randomly initialized conditions. However, for the online settings, we introduce the behavioral cloning term in the actor loss. We conduct hyperparameter optimization experiments for the optimized settings. When the behavioral policy's mean Q-value of the batch is higher than RBC policy, the

[1]Some scores with a standard deviation of 0 is caused by the round down of normalized scores, they are negligible numbers. But not exact zeros.

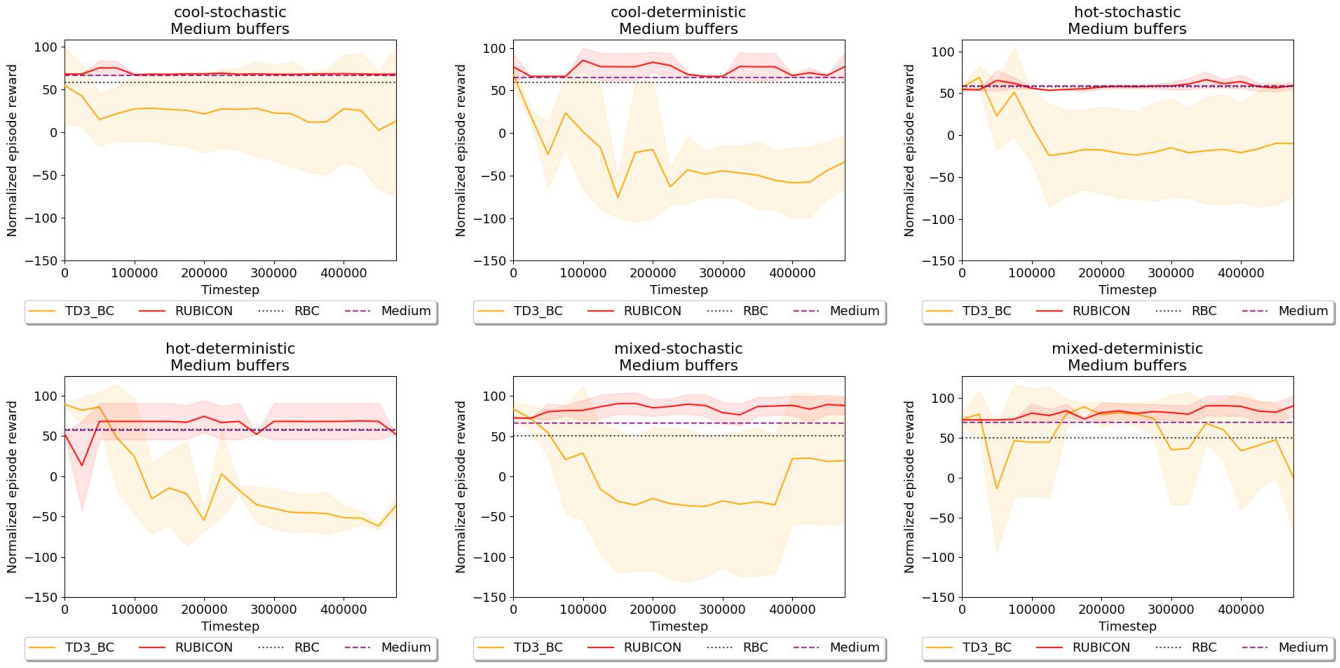

Figure 3: Learning curves of RUBICON and the baseline method TD3+BC with medium buffers. All learning curves are plotted with solid lines indicating averaged values and the half-transparent region is one standard deviation.

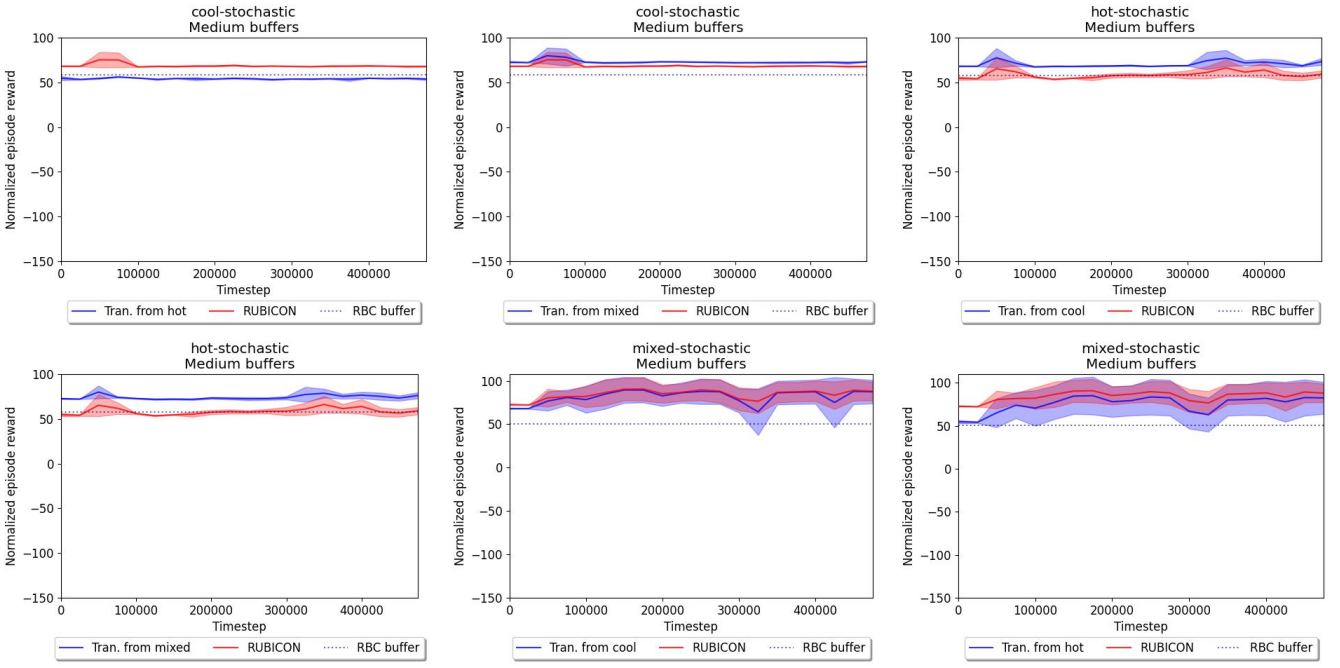

Figure 4: Learning curves of BRL models transferred from other weather types

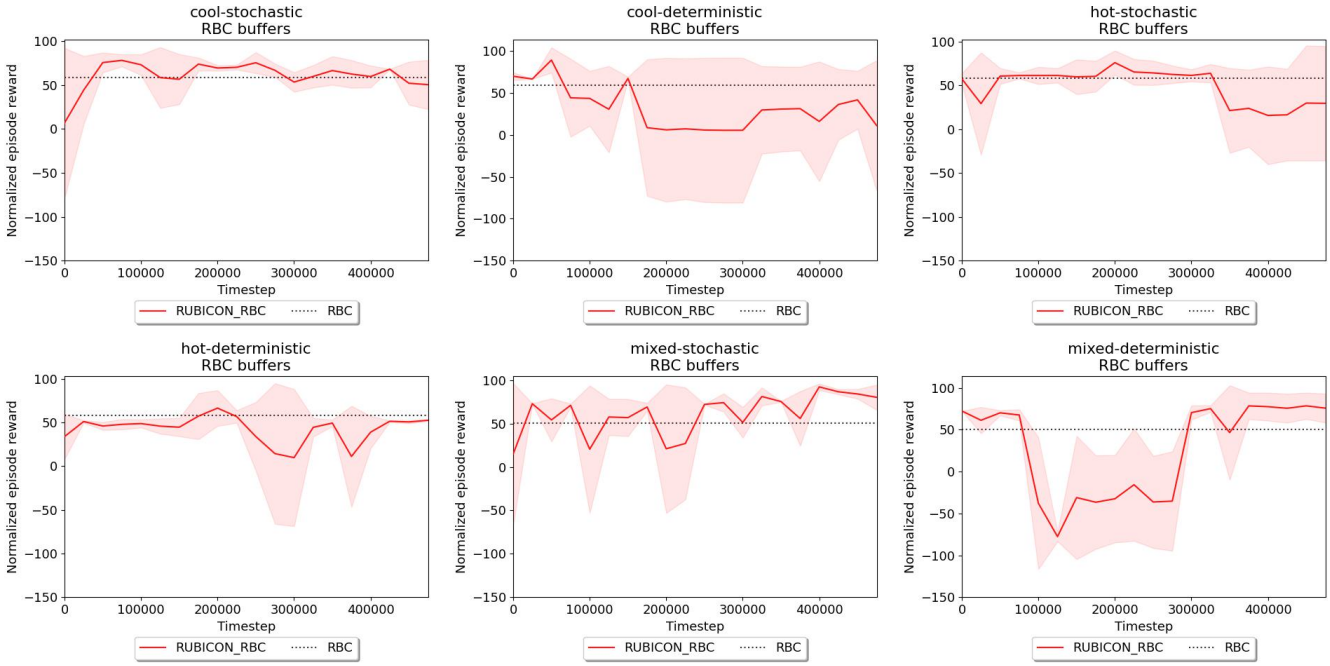

Figure 5: Learning curves of RUBICON learns from RBC buffers

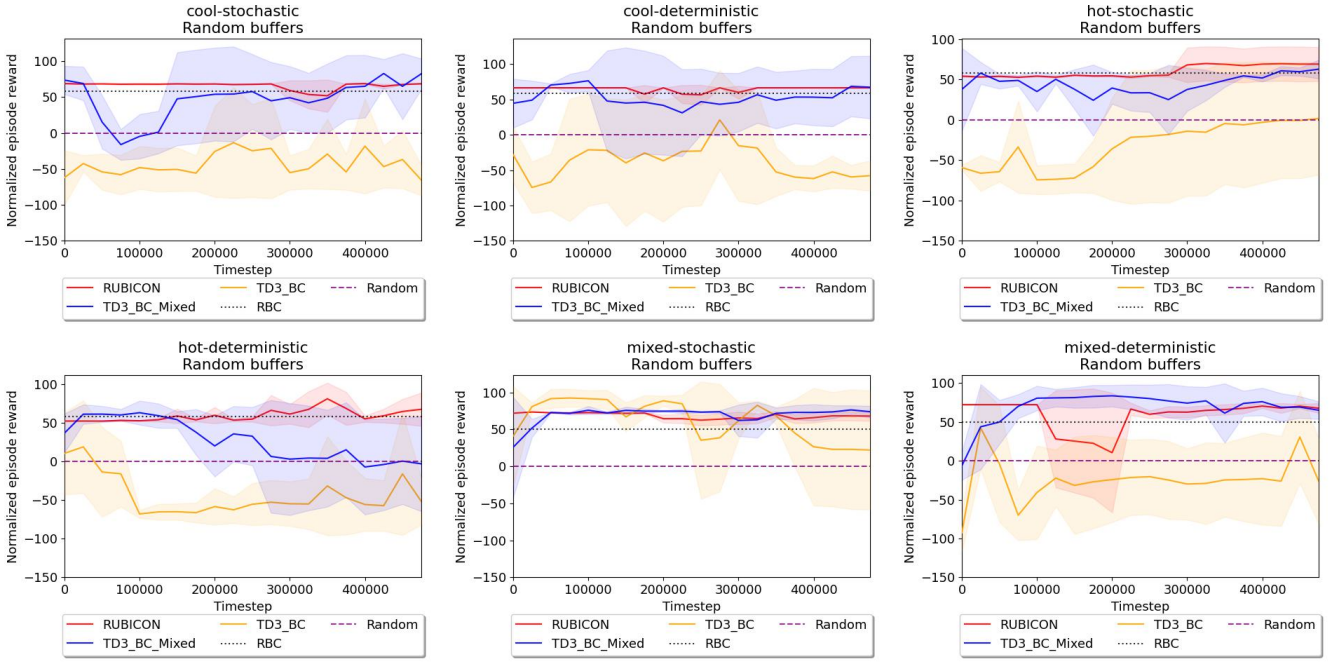

Figure 6: Learning curves comparing RUBICON and TD3+BC to TD3+BC learns from a mixture of 50% amount of transitions from the random buffer and 50% amount of transitions from the RBC buffer in stochastic environments

Table 1: BRL methods benchmark: Average normalized score over the final 5 evaluations and 3 random seeds. Values followed by ± correspond to standard deviation over the last 5 evaluations across runs. [1]

Environment	Buffer	RUBICON	TD3+BC	CQL	BCQ	BC
hot-deterministic	Expert	86.13±17.83	99.72±0.42	**100±0**	-32.01±95.46	-89.2±14.84
hot-deterministic	Medium	64.91±18.02	-49.58±13.52	**67.64±32.83**	13.4±51.24	-26.74±26.47
hot-deterministic	Random	62.7±14.36	-45.73±44.8	-23.19±76.76	**69.2±33.61**	-12.55±74.63
mixed-deterministic	Expert	81±25.94	94.66±7.36	**100±0**	-6.22±84.27	-95.46±14.78
mixed-deterministic	Medium	**86.84±12.39**	36.23±56.33	37.36±86.8	64.45±37.4	-27.82±63.16
mixed-deterministic	Random	**68.83±4.93**	-13.71±57.06	-23.46±83.61	-65.29±48.84	-103.4±7.45
cool-deterministic	Expert	98±2.78	81.11±16.88	**100±0**	-29.74±95.89	27.76±102.15
cool-deterministic	Medium	**72.2±8.07**	-49.97±36.4	55.44±49	70.18±14.42	8.62±45.32
cool-deterministic	Random	**66.5±0**	-58.4±19.25	12.98±73.04	27.77±63.67	10.48±70.79
hot-stochastic	Expert	99.01±0.56	77.69±30.48	**99.49±0.24**	-15.34±84.51	-72.86±38.25
hot-stochastic	Medium	**59.72±5.29**	-14.84±66.04	39.92±56.67	-62.2±5.25	31.22±66.26
hot-stochastic	Random	**68.83±21.26**	-1.82±73.31	36.64±67.61	-1.23±68.86	-10.45±61.91
mixed-stochastic	Expert	94.16±8.12	96.6±2.14	**99.77±0.21**	-108.38±2.83	-102.02±9.26
mixed-stochastic	Medium	**87.23±12.34**	9.48±81.06	80.13±20.78	70.75±9.9	38.66±48.02
mixed-stochastic	Random	67.03±6.26	28.01±72.79	**94.04±5.87**	-109.46±0.77	-107.41±4.36
cool-stochastic	Expert	53.58±65.53	78.27±31.08	**99.97±0.32**	-115.85±0.98	28.15±101.52
cool-stochastic	Medium	68.07±0.46	16.09±69.41	**81.56±18.01**	-11.55±56.13	25.44±35.57
cool-stochastic	Random	**67.55±1.14**	-44.33±36.36	-97.35±11.07	-53.92±78.06	-50.37±83.99
Sum		**1352.37±225.38**	339.49±714.77	960.98±582.88	-295.48±832.17	-527.93±868.81

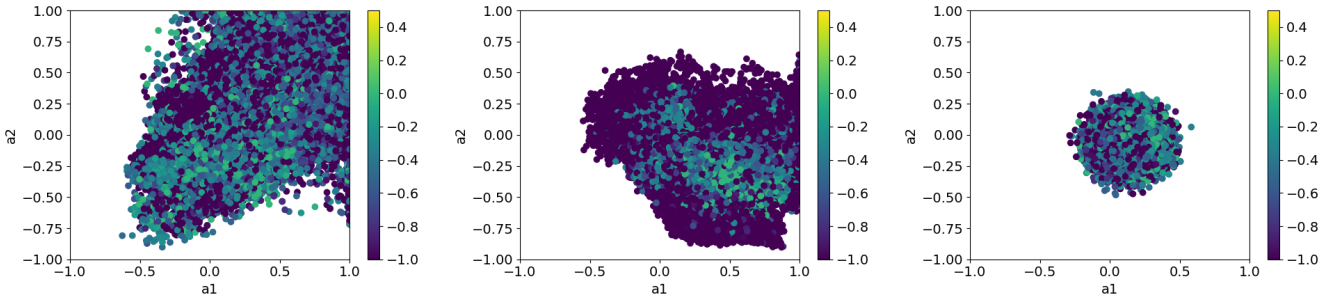

Figure 7: Reward distribution in action spaces of *hot-continuous* environment learns from medium buffer, from left to right: RUBICON (1.842/1.978/-0.577), TD3+BC (1.534/1.332/-0.668), and buffer (0.908/0.915/-0.799); tuples are the values of (action1 range/action2 range/reward mean).

actor loss follows the original TD3 algorithm. On the opposite, with RBC policy having a higher mean Q-value, the weighting λ and its hyperparameter α (see Eq. 2 and Alg. 1) should be optimized since we cannot assume the model's behavior is similar to the offline setting. It is mentioned in the TD3+BC paper that the value of α decides if the model learns similarly to RL (α=4) or imitation learning (α=1) and the default value set in TD3+BC is α =2.5. We experiment on the values $\{1, 2.5, 4\}$ to observe how it affects the performance of our models in all tasks. The result (See Table 4) shows that when α=1 the model gives the highest scores and the least variance. This indicates that the agent should imitate RBC policy even more than the offline setting ($\alpha = 2.5$) in order to achieve a more optimal policy.

6 CONCLUSION AND FUTURE WORKS

In this paper, we explored how rule-based control policies can be incorporated into reinforcement learning as regularization to improve both of their performance. Our method can be implemented on the baseline methods with minimal changes and is straightforward and intuitive. We applied our method in building HVAC control simulation environments in both online and offline settings, demonstrating its practical usage regardless of the existence of a valid environment simulator. We empirically demonstrate that our method outperforms state-of-the-art offline/batch reinforcement learning methods and improves from its online baseline by a substantial amount in building HVAC control tasks where rule-based control is robust and a standard in real-world settings. We expect

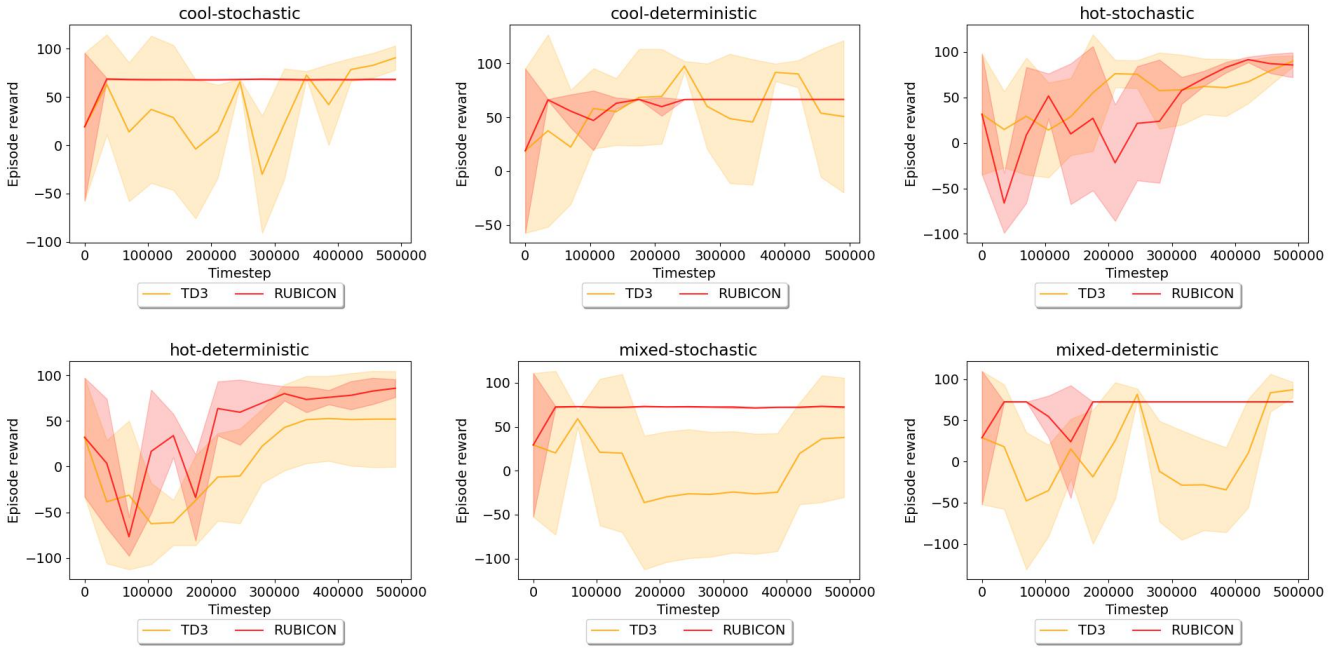

Figure 8: Learning curves comparing online RUBICON and TD3

Table 2: Data reduction experiment

Environment	Buffer	RUBICON	TD3+BC
hot-deterministic	Expert	**89.65±9.12**	74.96±12.38
hot-deterministic	Medium	-6.92±78.43	**79.37±25.5**
hot-deterministic	Random	**90.41±5.47**	-24.18±59.62
mixed-deterministic	Expert	43.09±60.36	**93.77±8.71**
mixed-deterministic	Medium	48.24±54.32	**98.67±2.28**
mixed-deterministic	Random	**82.89±10.19**	72.58±6.91
cool-deterministic	Expert	**89.52±11.28**	86.45±8.62
cool-deterministic	Medium	-11.56±62.82	**47.61±17.66**
cool-deterministic	Random	35.92±43.24	**42.59±72.43**
hot-stochastic	Expert	80.35±25.8	**89.02±5.51**
hot-stochastic	Medium	5.3±43.33	**10.79±56.47**
hot-stochastic	Random	**62.26±18.59**	26.43±52.33
mixed-stochastic	Expert	78.08±25.78	**85.25±12.96**
mixed-stochastic	Medium	66.02±17.68	**72.17±16.14**
mixed-stochastic	Random	**55.26±29.19**	-86.21±25.92
cool-stochastic	Expert	**98.18±2.56**	39.04±86.35
cool-stochastic	Medium	**73.16±19.56**	38.68±39.93
cool-stochastic	Random	**69.76±5.46**	-49.5±9.75
Sum	.	**1049.61±523.18**	797.49±519.47

Table 3: Online RUBICON and TD3 comparison 1

Environment	RUBICON	TD3
hot-deterministic	**79.08±12.24**	51.83±49.93
mixed-deterministic	**72.34±0.00**	23.46±41.01
cool-deterministic	**66.52±0.00**	**66.3±41.66**
hot-stochastic	**83.64±8.25**	71.8±21.51
mixed-stochastic	**72.24±0.49**	8.5±66.61
cool-stochastic	68.14±0.65	**73.38±16.61**
Sum	**441.99±21.64**	295.29±237.36

our study, open-sourced code bases and dataset[2] would encourage both domains and RL experts to explore more opportunities for the combination of existing policies and RL and extend this concept to more real-world applications.

For future works, we plan to enhance the interpretability of the decision-making process in our experiments, we aim to develop transparent and interpretable algorithms for RL agents via Explainable RL (XRL) [1, 16, 36]. Also, using the ensemble Q-networks for a more accurate Q-value estimation.

ACKNOWLEDGMENTS

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

[2]For the dataset please refer to our B2RL (https://github.com/HYDesmondLiu/B2RL) repository. And for the codes please refer to our RUBICON (https://github.com/HYDesmondLiu/RUBICON) repository.

REFERENCES

[1] Tameem Adel, Alexander Rosenberg, and Been Kim. 2019. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In *Advances in Neural Information Processing Systems (NeurIPS)*. 10027–10036.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. 2021. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems* 34 (2021), 29304–29320.

[3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[4] Hamid R Berenji. 1992. A reinforcement learning—based architecture for fuzzy logic control. *International Journal of Approximate Reasoning* 6, 2 (1992), 267–292.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[6] Xianzhong Ding, Wan Du, and Alberto Cerpa. 2019. OCTOPUS: Deep reinforcement learning for holistic smart building control. In *BuildSys*. 326–335.

[7] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219* (2020).

[8] Scott Fujimoto and Shixiang Shane Gu. 2021. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems* 34 (2021), 20132–20145.

[9] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*. PMLR, 1587–1596.

[10] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *ICML*. PMLR, 2052–2062.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[12] Mengjie Han, Ross May, Xingxing Zhang, Xinru Wang, Song Pan, Da Yan, Yuan Jin, and Liguo Xu. 2019. A review of reinforcement learning methodologies for controlling occupant comfort in buildings. *Sustainable Cities and Society* 51 (2019), 101748.

[13] Frederick Hayes-Roth. 1985. Rule-based systems. *Commun. ACM* 28, 9 (1985), 921–932.

[14] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[15] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. 2021. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *arXiv preprint arXiv:2111.08819* (2021).

[16] Lu Jiang, Tong Xiao, and Thomas Huang. 2018. Learning to Explain: A Framework for Machine Learning Explanations. In *Advances in Neural Information Processing Systems (NeurIPS)*. 9810–9820.

[17] Javier Jiménez-Raboso, Alejandro Campoy-Nieves, Antonio Manjavacas-Lucas, Juan Gómez-Romero, and Miguel Molina-Solana. 2021. Sinergym: a building simulation and control framework for training reinforcement learning agents. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 319–323.

[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[20] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. 2021. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*. PMLR, 5774–5783.

[21] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191.

[22] National Renewable Energy Laboratory. 2008. *TMY3 Datasets*.

[23] Daeil Lee, Awwal Mohammed Arigi, and Jonghyun Kim. 2020. Algorithm for autonomous power-increase operation using deep reinforcement learning and a rule-based system. *IEEE Access* 8 (2020), 196727–196746.

[24] Amarildo Likmeta, Alberto Maria Metelli, Andrea Tirinzoni, Riccardo Giol, Marcello Restelli, and Danilo Romano. 2020. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics and Autonomous Systems* 131 (2020), 103568.

[25] Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8, 3 (1992), 293–321.

[26] Hsin-Yu Liu, Bharathan Balaji, Sicun Gao, Rajesh Gupta, and Dezhi Hong. 2022. Safe HVAC Control via Batch Reinforcement Learning. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 181–192.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[28] Department of Energy. 2023. *Prototype Building Models*. https://www.energycodes.gov/prototype-building-models#TMY3

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[30] Majdi I Radaideh and Koroush Shirvan. 2021. Rule-based reinforcement learning methodology to inform evolutionary algorithms for constrained optimization of engineering applications. *Knowledge-Based Systems* 217 (2021), 106836.

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[33] Takuma Seno and Michita Imai. 2021. d3rlpy: An Offline Deep Reinforcement Learning Library. *arXiv preprint arXiv:2111.03788* (2021).

[34] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. PMLR, 387–395.

[35] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[36] Przemysław Spurek, Damian Szymański, and Tomasz Tajmajer. 2019. Towards Interpretable Reinforcement Learning Using Attention Augmented Agents. In *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 3239–3245.

[37] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. 2015. Safe exploration for optimization with Gaussian processes. In *International conference on machine learning*. PMLR, 997–1005.

[38] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 5026–5033.

[40] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. 2016. Safe exploration in finite markov decision processes with gaussian processes. *Advances in Neural Information Processing Systems* 29 (2016).

[41] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in science & engineering* 13, 2 (2011), 22–30.

[42] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. 2020. Munchausen reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 4235–4246.

[43] Junjie Wang, Qichao Zhang, Dongbin Zhao, and Yaran Chen. 2019. Lane change decision-making through deep reinforcement learning with rule-based constraints. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–6.

[44] Zhe Wang and Tianzhen Hong. 2020. Reinforcement learning for building controls: The opportunities and challenges. *Applied Energy* 269 (2020), 115036.

[45] Tianshu Wei, Yanzhi Wang, and Qi Zhu. 2017. Deep reinforcement learning for building HVAC control. In *Proceedings of the 54th annual design automation conference 2017*. 1–6.

[46] Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361* (2019).

[47] Lei Yang, Zoltan Nagy, Philippe Goffin, and Arno Schlueter. 2015. Reinforcement learning for optimal control of low exergy buildings. *Applied Energy* 156 (2015), 577–586.

[48] Liang Yu, Shuqi Qin, Meng Zhang, Chao Shen, Tao Jiang, and Xiaohong Guan. 2020. Deep Reinforcement Learning for Smart Building Energy Management: A Survey. *arXiv preprint arXiv:2008.05074* (2020).

[49] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. 2021. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems* 34 (2021), 28954–28967.

[50] Chi Zhang, Sanmukh Rao Kuppannagari, and Viktor K Prasanna. 2022. Safe Building HVAC Control via Batch Reinforcement Learning. *IEEE Transactions on Sustainable Computing* 7, 4 (2022), 923–934.

[51] Zhiang Zhang, Adrian Chong, Yuqi Pan, Chenlu Zhang, and Khee Poh Lam. 2019. Whole building energy model for HVAC optimal control: A practical framework based on deep reinforcement learning. *Energy and Buildings* 199 (2019), 472–490.

[52] Yuanyang Zhu, Zhi Wang, Chunlin Chen, and Daoyi Dong. 2021. Rule-based reinforcement learning for efficient robot navigation with space reduction. *IEEE/ASME Transactions on Mechatronics* 27, 2 (2021), 846–857.

A EXPERIMENT DETAILS

- **Software**
  - **Python**: 3.9.12
  - **Pytorch**: 1.12.1+cu113 [29]
  - **Sinergym**: 1.9.5 [17]
  - **Gym**: 0.21.0 [5]
  - **Numpy**: 1.23.1 [41]
  - **CUDA**: 11.2
- **Hardware**
  - **CPU**: Intel Xeon Gold 6230 (2.10 GHz)
  - **GPU**: NVidia RTX A6000
- **Average training/evaluation time for RUBICON**: 4 hours 45 minutes 18 seconds
- **Benchmark implementations**
  - **DDPG**: We adopt the DDPG implementation in TD3 author-provided implementation
  - **TD3**: Author-provided implementation
  - **SAC**: We adopt CleanRL [15] implementation due to software version conflict with author-provided repository
  - **TD3+BC**: Author-provided implementation
  - **CQL**: We adopt d3rlpy [33] implementation due to software version conflict with the author-provided repository
  - **BCQ**: Author-provided implementation

B LEARNING CURVES, DETAILED SCORES, AND ADDITIONAL EXPERIMENTS

- **BRL learning** We illustrate the learning curves of BRL methods learn from different quality of buffers for better visualization of comparison in Fig. 9, 10 and 11
- **Behavioral agents** The behavioral agents' learning curves are demonstrated in Fig. 12
- **CQL+RUBICON** Since CQL demonstrates better performance compared to other methods except for RUBICON (see Table 1), we conduct experiments combining CQL and our RUBICON method to learn from random and medium buffers since the performance of CQL-expert is already extremely well. The results in Figure 13, 14 and Table 7, 8 indicate that RUBICON also improves CQL. However, it does not consistently improve CQL's performance from task to task, which is not as we observe from TD3+BC to RUBICON learn from random/medium buffers (see Figure 11, 3). Also, the improvement is limited and does not even reach the RBC policy performance, thus we did not continue exploring the possibility of combining CQL with RUBICON.
- **Learn from worsened RBC policies** We run another ablation experiment to observe how the quality of RBC policy affects the performance compared with RUBICON and baseline TD3+BC. We design two worsened RBC policies: The first one is a biased RBC where we modify the change in setpoints (a_{h_i} and a_{c_i}) from 1 to 5 in Algo. 2, we name this method "RBC_CB" in Figure 15. The other is to replace RBC with random policy, it is named as "RBC_Random". From the results in Table 10 we could find that even with constantly worsened RBC policy it still improves from baseline, However, it is still too aggressive for the models to learn a robust policy. And with random policy as a worsened RBC it is almost equivalent as no reference policy, the performance is similar to our baseline TD3+BC.
- **Non-selective experiments** In this experiment, we remove the dynamically weighted regularization. Instead, we regularize the behavioral policy and RBC policy simultaneously in every iteration of training (see Eq. 5). The experimental results are shown in Table 11. We observe that regularizing both policies at the same time deteriorates the model performance cf. RUBICON. Since in each iteration, one of RBC policy $\pi_{rbc}(s)$ and behavioral policy $\pi_b(s)$ yields a better action selection compared to the other. It emphasizes the necessity of dynamic weighting in the policy update steps.

$$\pi = \arg\max_{\pi} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\lambda Q(s, \pi(s)) - (\pi(s) - a)^2 - (\pi(s) - \pi_{rbc}(s))^2\right] \tag{5}$$

All learning curves are normalized with random policy as 0 and expert policy as 100, averaged with 3 random seeds and the scores shown in tables are the average and standard deviation last 5 evaluations.

C MODEL PARAMETERS

We list the hyperparameters used in this paper for reproducibility. Unless mentioned otherwise, we keep the original hyperparameters setups as the implementations listed in Appendix A since DRL methods are sensitive to hyperparameter tuning [14] (see Table 12, 13, 14, and 15).

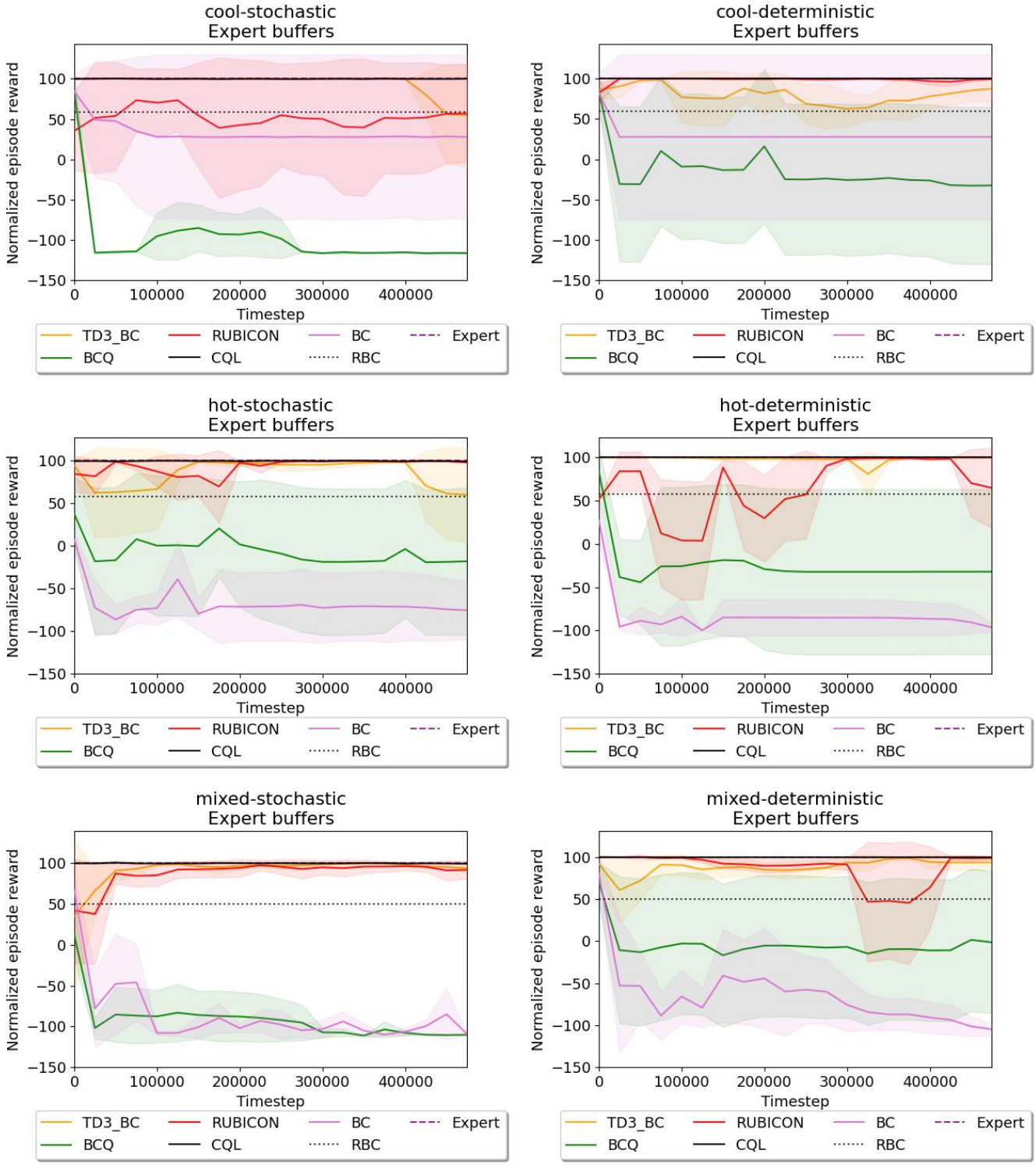

Figure 9: Learning curves of BRL models learn from expert buffers.

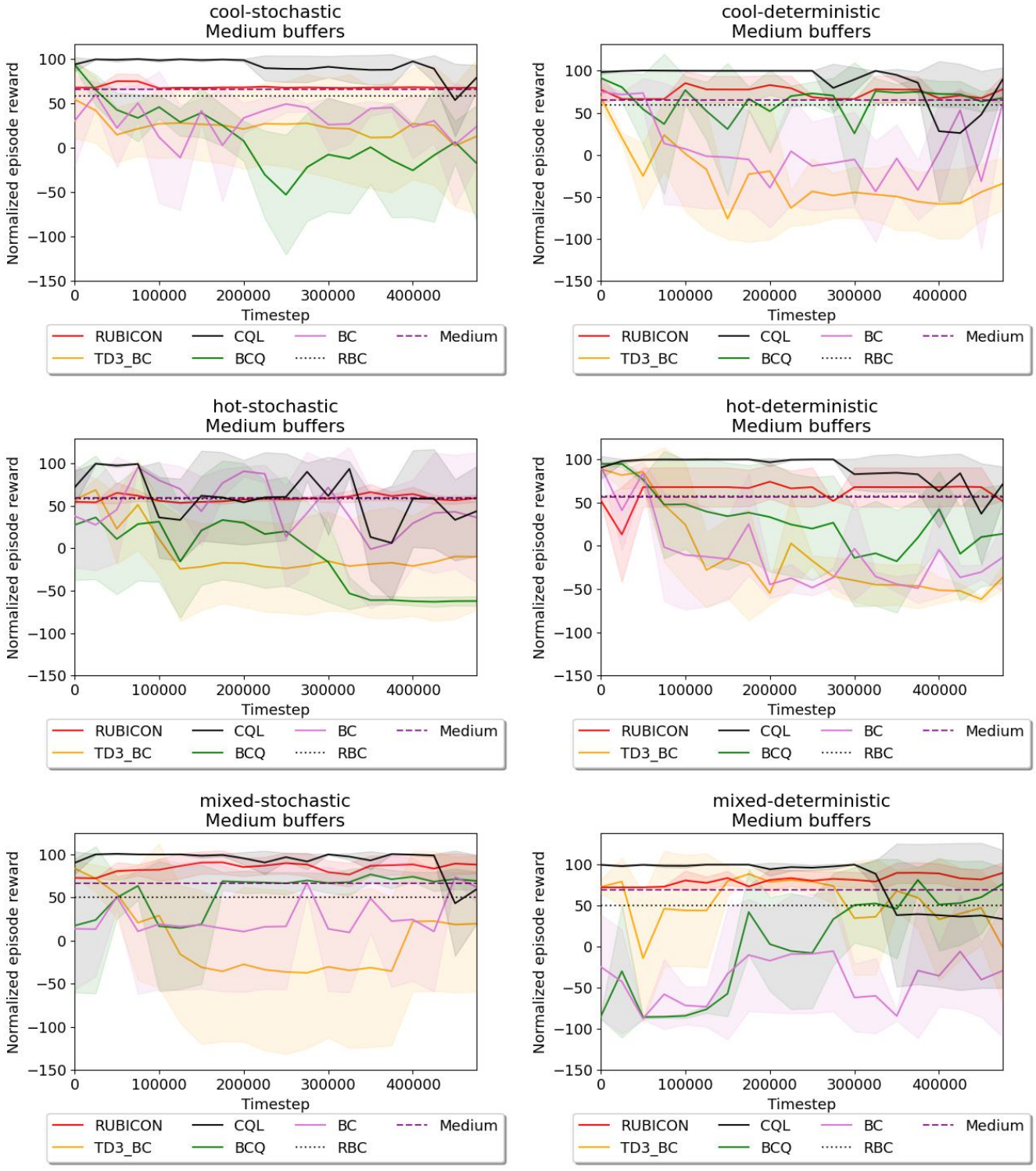

Figure 10: Learning curves of BRL models learn from medium buffers.

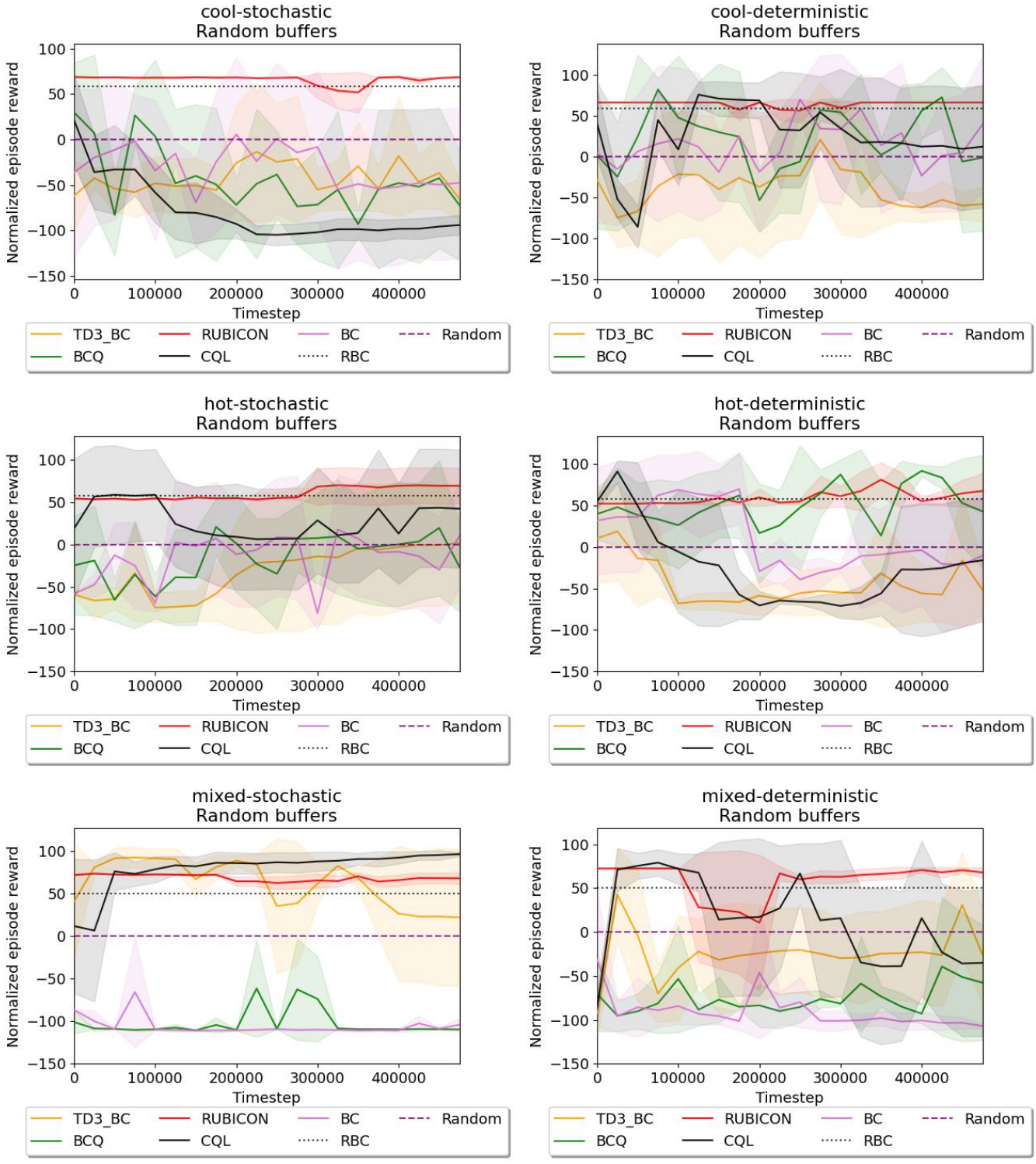

Figure 11: Learning curves of BRL models learn from random buffers

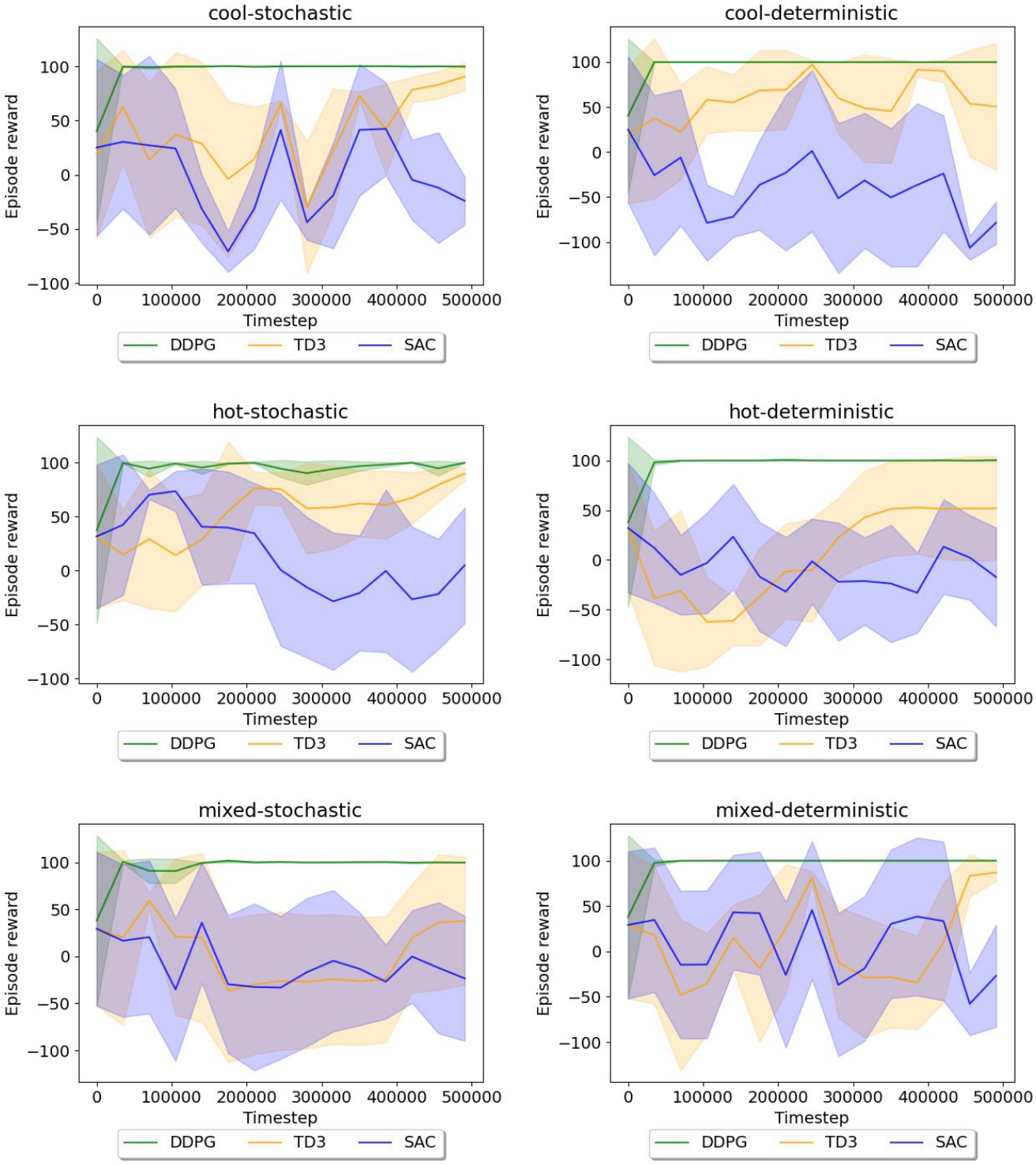

Figure 12: Learning curves of behavioral model training, behavioral models are trained with 500K time steps before generating buffers.

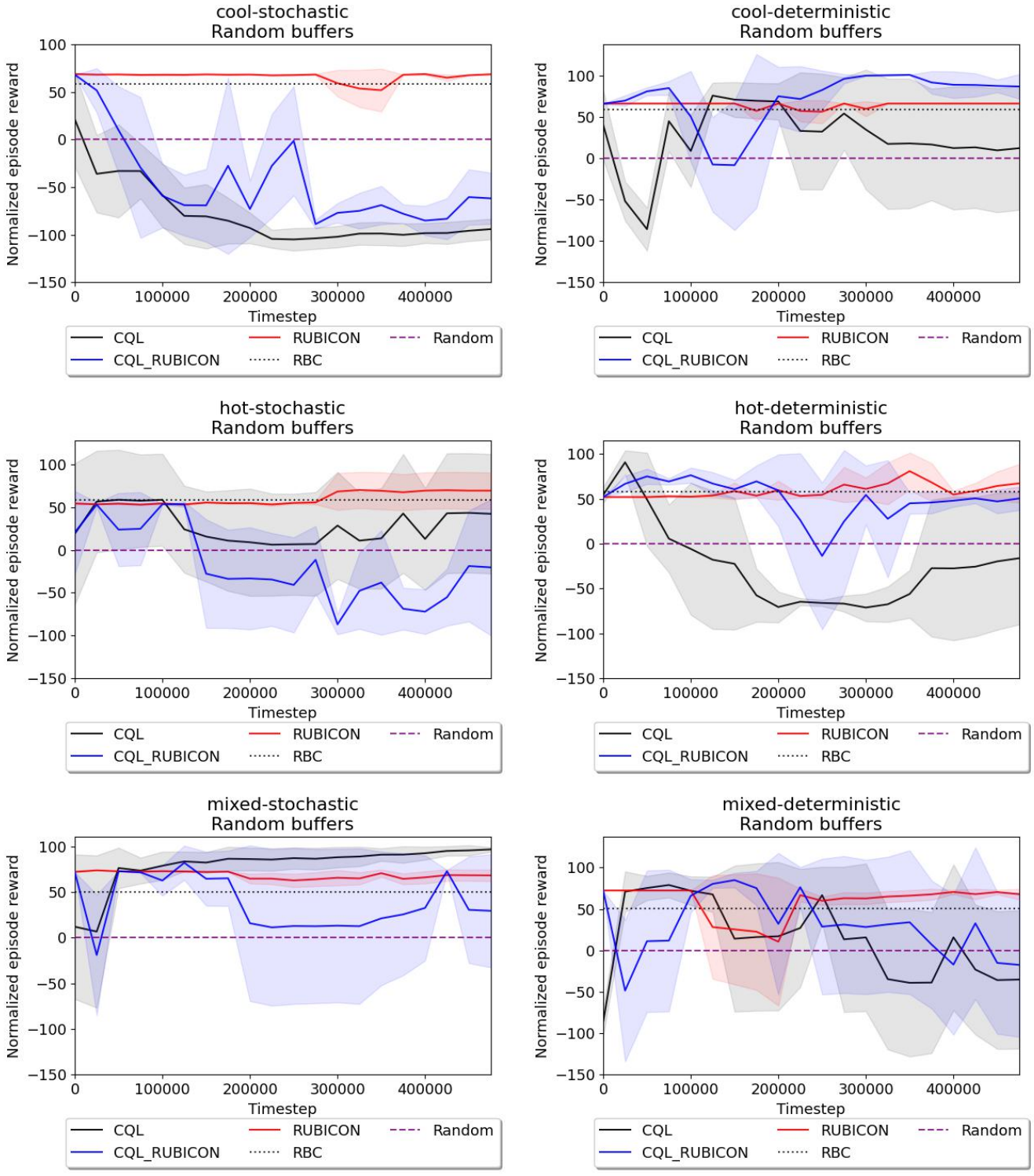

Figure 13: Learning curves of CQL, CQL+RUBICON, and RUBICON learn from random buffers

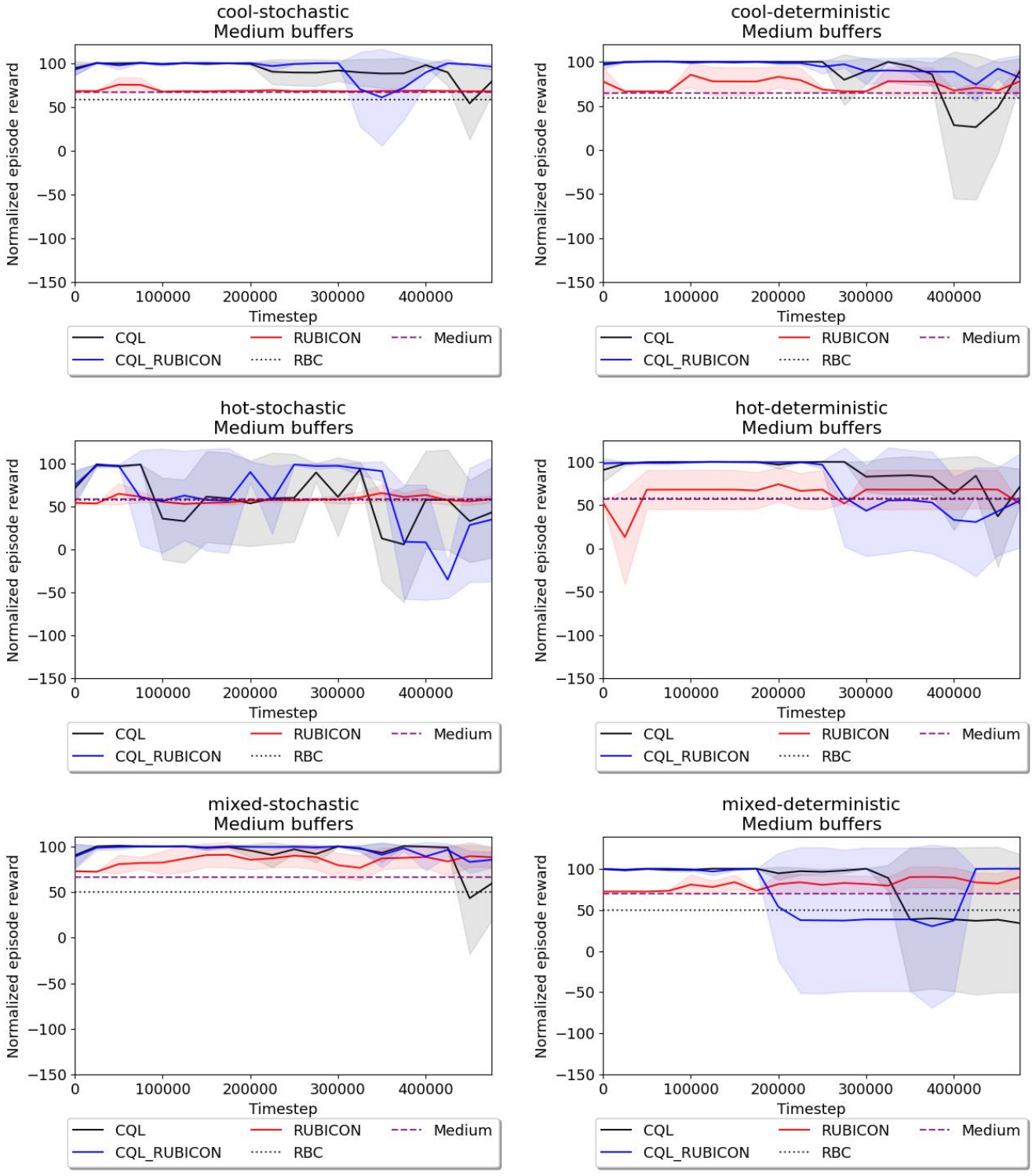

Figure 14: Learning curves of CQL, CQL+RUBICON, and RUBICON learn from medium buffers

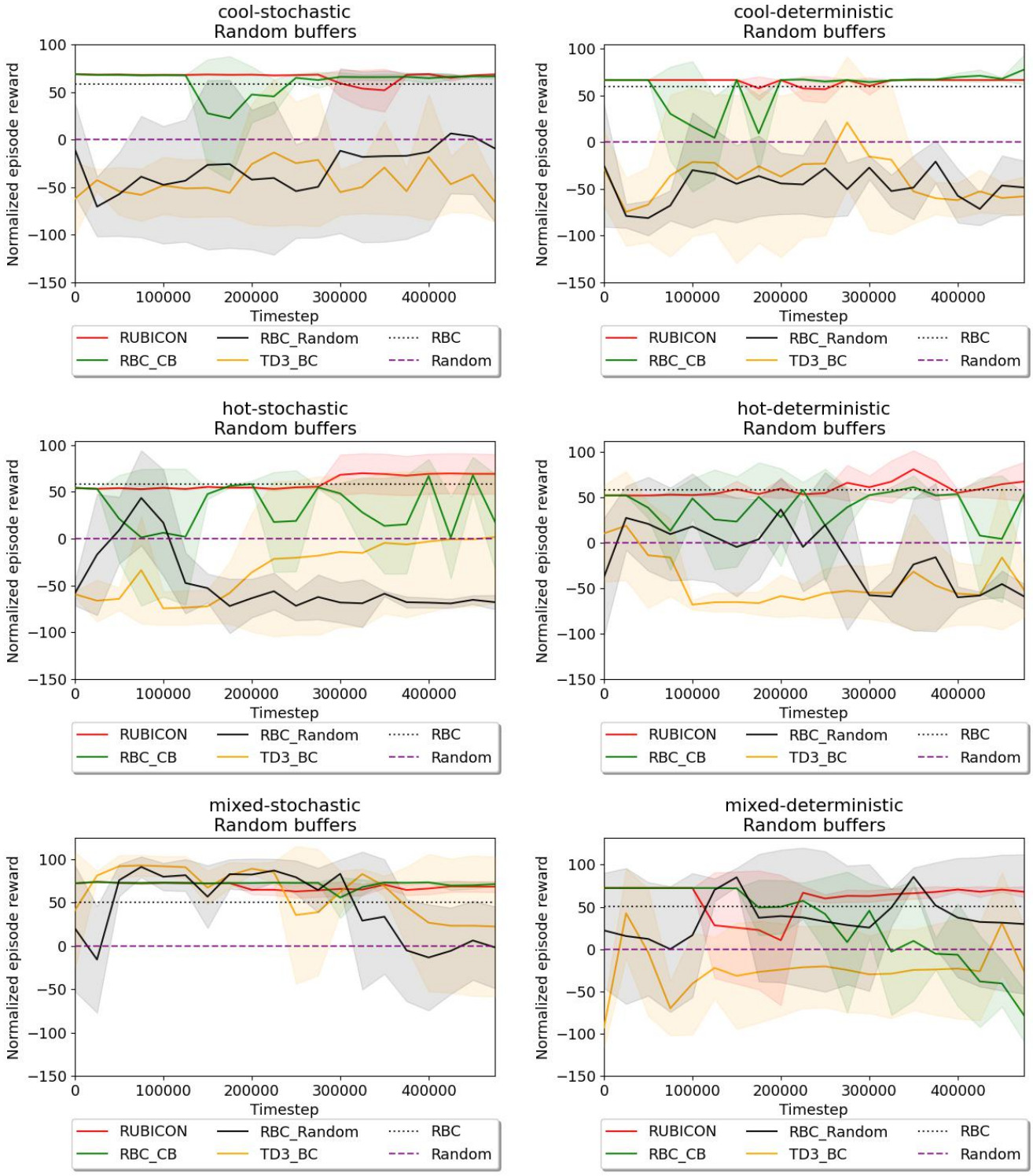

Figure 15: Learning curves of RUBICON learns from worsened RBC compared with TD3+BC and RUBICON

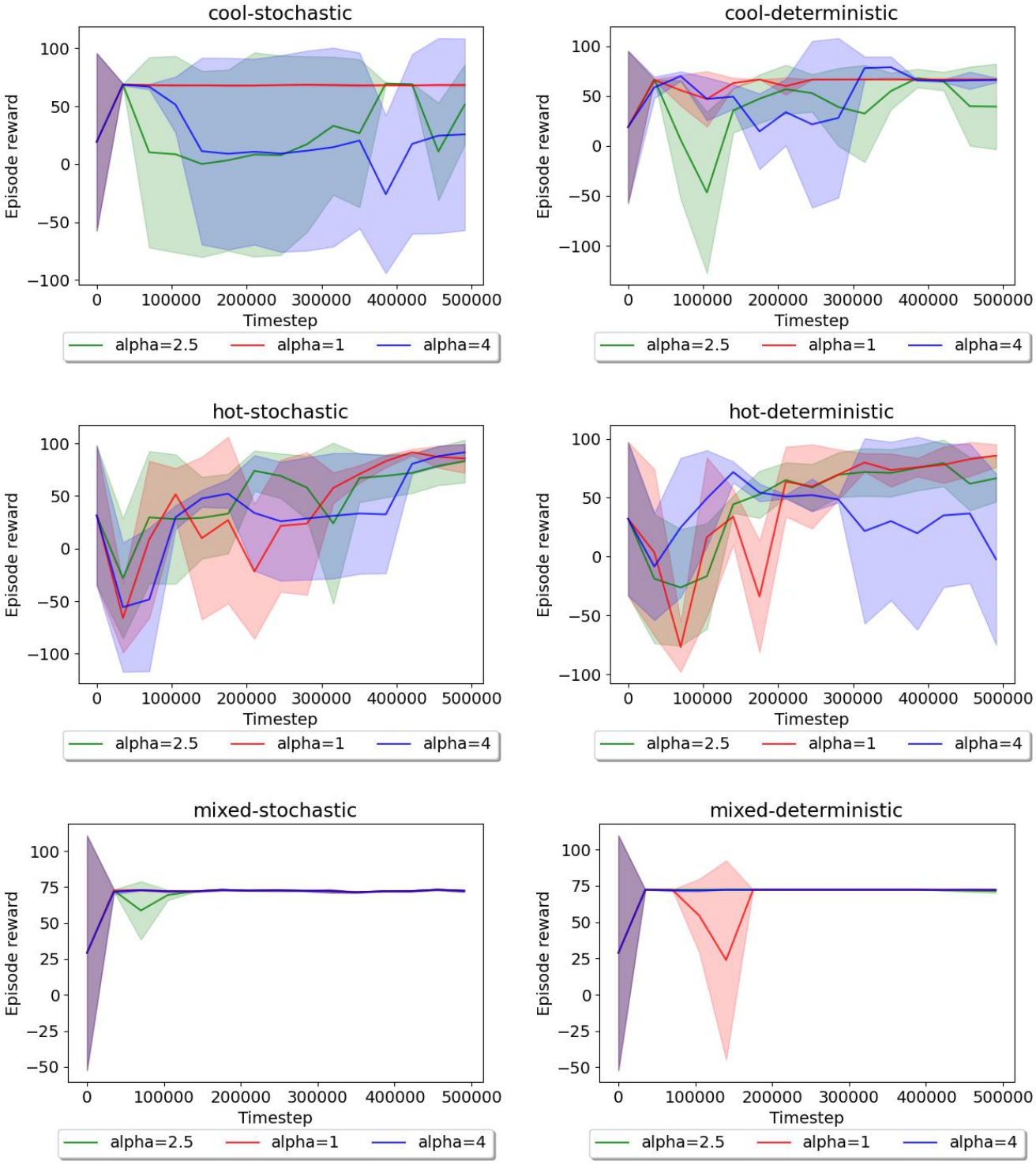

Figure 16: Learning curves of online RUBICON hyperparameter optimization

Table 4: Hyperparameter experiment. 1

Environment	$\alpha = 1$	$\alpha = 2.5$	$\alpha = 4$
hot-deterministic	**79.08±12.24**	70.76±20.28	23.83±68.3
mixed-deterministic	**72.34±0.00**	71.92±0.52	**72.26±0.05**
cool-deterministic	**66.52±0.00**	53.28±23.81	**68.16±4.69**
hot-stochastic	**83.64±8.25**	73.92±20.24	65.13±28.08
mixed-stochastic	**72.24±0.49**	**72.24±0.49**	**72.24±0.49**
cool-stochastic	**68.14±0.65**	45.46±28.4	12.38±77.7
Sum	**441.99±21.64**	387.58±93.74	347.08±130.01

Table 5: Transfer experiment

Environment	Trans. from	RUBICON_Trans.	RUBICON
hot-stochastic	cool-stochastic	**71.42±3.1**	59.72±5.29
mixed-stochastic	cool-stochastic	84.93±16.92	**87.23±12.34**
cool-stochastic	hot-stochastic	54.07±0.85	**68.07±0.46**
mixed-stochastic	hot-stochastic	81.02±20.09	**87.23±12.34**
cool-stochastic	mixed-stochastic	**72.39±0.68**	68.07±0.46
hot-stochastic	mixed-stochastic	**75.26±3.3**	59.72±5.29
Sum		**439.09±44.94**	430.04±36.18

Table 6: RUBICON learns from buffers generated by RBC compared with RBC buffer performance

Environment	RUBICON	RBC
hot-deterministic	**67.92±22.51**	57.9
mixed-deterministic	**73.68±1.96**	50.12
cool-deterministic	**72.28±6.53**	59.15
hot-stochastic	53.83±0.8	**57.92**
mixed-stochastic	**72.46±0.68**	50.22
cool-stochastic	53.35±20.36	**58.48**
Sum	**393.53±52.85**	333.79

Table 7: CQL+RUBICON learns from random buffer compared with CQL and RUBICON 1

Environment	RUBICON	CQL_RUBICON	CQL
hot-deterministic	**62.7±14.36**	48.37±10.12	-23.19±76.76
mixed-deterministic	**68.83±4.93**	-2±85.18	-23.46±83.61
cool-deterministic	66.5±0	**88.98±13.24**	12.98±73.04
hot-stochastic	**68.83±21.26**	-47.04±45.48	36.64±67.61
mixed-stochastic	67.03±6.26	38.08±49.17	**94.04±5.87**
cool-stochastic	**67.55±1.14**	-73.76±20.51	-97.35±11.07
Sum	**401.47±47.98**	52.64±223.72	-0.32±317.97

Table 8: Scores of CQL+RUBICON learns from medium buffer compared with CQL and RUBICON

Environment	RUBICON	CQL_RUBICON	CQL
hot-deterministic	64.91±18.02	43.03±55.26	**67.64±32.83**
mixed-deterministic	**86.84±12.39**	73.4±37.59	37.36±86.8
cool-deterministic	72.2±8.07	**85.24±17.33**	55.44±49
hot-stochastic	**59.72±5.29**	9.39±58.72	39.92±56.67
mixed-stochastic	87.23±12.34	**90.14±9.37**	80.13±20.78
cool-stochastic	68.07±0.46	**91.05±11.39**	81.56±18.01
Sum	**438.98±56.59**	392.26±189.69	362.07±264.11

Table 9: Scores of TD3+BC learns from a mixture of random buffer and RBC buffer compared with RUBICON learns from random buffer 1

Environment	TD3+BC_Mixed	RUBICON	TD3+BC
hot-deterministic	0.02±59.76	**62.7±14.36**	-45.73±44.8
mixed-deterministic	**70.66±15.45**	68.83±4.93	-13.71±57.06
cool-deterministic	59.01±40.92	**66.5±0**	-58.4±19.25
hot-stochastic	57.93±5.6	**68.83±21.26**	-1.82±73.31
mixed-stochastic	**74.08±8.7**	67.03±6.26	28.01±72.79
cool-stochastic	**71.67±35.04**	67.55±1.14	-44.33±36.36
Sum	333.4±165.5	**401.47±47.98**	-135.98±303.60

Table 10: Comparison between RUBICON, TD3+BC, and worsened RBCs 1

Environment	RUBICON	TD3+BC	RBC_CB	RBC_Random
hot-deterministic	**62.7±14.36**	-45.73±44.8	34.06±27.13	-47.72±25.11
mixed-deterministic	**68.83±4.93**	-13.71±57.06	-33.89±38.47	36.39±72.08
cool-deterministic	66.5±0	-58.4±19.25	**70.64±5.85**	-48.84±25.93
hot-stochastic	**68.83±21.26**	-1.82±73.31	33.81±36.94	-67.7±5.66
mixed-stochastic	67.03±6.26	28.01±72.79	**71.22±2.64**	-4.07±52.46
cool-stochastic	**67.55±1.14**	-44.33±36.36	65.84±3.06	-5.9±74.12
Sum	**401.47±47.98**	-135.98±303.60	241.69±114.12	-137.85±255.38

Table 11: Non-selective experiment 1

Environment	Buffer	RUBICON	RUBICON w/o DW
hot-deterministic	Expert	**86.13±17.83**	-19.8±63.89
hot-deterministic	Medium	**64.91±18.02**	47.26±12.89
hot-deterministic	Random	**62.7±14.36**	-19.8±63.89
mixed-deterministic	Expert	**81±25.94**	-75.6±29.46
mixed-deterministic	Medium	**86.84±12.39**	42.99±48.04
mixed-deterministic	Random	**68.83±4.93**	-75.6±29.46
cool-deterministic	Expert	**98±2.78**	41.3±20.53
cool-deterministic	Medium	**72.2±8.07**	36.54±67.84
cool-deterministic	Random	**66.5±0**	41.3±20.53
hot-stochastic	Expert	**99.01±0.56**	57.68±22.3
hot-stochastic	Medium	**59.72±5.29**	29.26±45.5
hot-stochastic	Random	**68.83±21.26**	57.68±22.3
mixed-stochastic	Expert	**94.16±8.12**	40.57±44.91
mixed-stochastic	Medium	**87.23±12.34**	55.6±33.53
mixed-stochastic	Random	**67.03±6.26**	40.57±44.91
cool-stochastic	Expert	**53.58±65.53**	-68.84±27.46
cool-stochastic	Medium	**68.07±0.46**	8.01±61.1
cool-stochastic	Random	**67.55±1.14**	-68.84±27.46
Sum		**1352.37±225.38**	170.3±686.1

Table 12: TD3, TD3+BC, and RUBICON hyperparameters

	Hyperparameter	Value
Algorithm hyperparameters	Optimizer	Adam [18]
	Critic learning rate	$3e^{-4}$
	Actor learning rate	$3e^{-4}$
	Mini-batch size	256
	Discount factor	0.99
	Target update rate	$5e^{-3}$
	Policy noise	0.2
	Policy noise clipping	(-0.5, 0.5)
	Policy update frequency	2
	TD3+BC α	2.5
	RUBICON online α	1
	RUBICON offline α	2.5
	RUBICON online ξ	0 if $\bar{Q}(s, \pi_b(s)) \geq \bar{Q}(s, \pi_{rbc}(s))$ else 1
	RUBICON offline ξ	1
Network architecture	Critic hidden dimension	256
	Critic hidden layers	2
	Critic activation function	ReLU
	Actor hidden dimension	256
	Actor hidden layers	2
	Actor activation function	ReLU

Table 13: SAC/CQL hyperparameters

	Hyperparameter	Value
Algorithm hyperparameters	Optimizer	Adam
	Critic learning rate	$1e^{-3}$
	Actor learning rate	$3e^{-4}/1e^{-4}$
	Mini-batch size	256
	Discount factor	0.99
	Target update rate	$5e^{-3}$
	Policy noise	0.2
	Policy noise clipping	(-0.5, 0.5)
	Policy update frequency	2
	SAC entropy auto-tuning	True
	CQL α threshold	10
	CQL conservative weight	5.0
	CQL number of sampled actions	10
Network architecture	Critic hidden dimension	256
	Critic hidden layers	3
	Critic activation function	ReLU
	Actor hidden dimension	256
	Actor hidden layers	3
	Actor activation function	ReLU

Table 14: DDPG hyperparameters

	Hyperparameter	Value
Algorithm hyperparameters	Optimizer	Adam
	Critic learning rate	$1e^{-3}$
	Actor learning rate	$1e^{-4}$
	Mini-batch size	64
	Discount factor	0.99
	Target update rate	$1e^{-3}$
	Policy noise	$\mathcal{N}(0, 0.1)$
	Policy noise clipping	(-0.5, 0.5)
	Policy update frequency	1
Network architecture	Critic hidden dimension	400/300
	Critic hidden layers	2
	Critic activation function	ReLU
	Actor hidden dimension	400/300
	Actor hidden layers	2
	Actor activation function	ReLU

Table 15: BCQ/BC hyperparameters

	Hyperparameter	Value
Algorithm hyperparameters	Optimizer	Adam
	Critic learning rate	$1e^{-3}$
	Actor learning rate	$1e^{-4}$
	Mini-batch size	100
	Discount factor	0.99
	Target update rate	$5e^{-3}$
	Minimum weighting	0.75
	Max perturbation	0.05
Network architecture	Critic hidden dimension	400/300
	Critic hidden layers	2
	Critic activation function	ReLU
	Actor hidden dimension	400/300
	Actor hidden layers	2
	Actor activation function	ReLU
	VAE hidden dimension	750
	VAE latent vector clipping	(-0.5, 0.5)