Revisiting TD Target Aggregation under Uncertainty in Q-Learning

Lipeng Zu[1] **Xiaonan Zhang**[1]

[1]Computer Science Department, Florida State University, Tallahassee, Florida, USA

Abstract

Deep Q-Networks (DQNs) learn value functions through bootstrapped temporal-difference updates, where future returns are approximated using a greedy maximization over next-state action values. While effective, this aggregation rule is inherently sensitive to estimation noise: when Q-values are uncertain, the maximization operator deterministically favors the largest estimate, regardless of its reliability, leading to amplified errors through bootstrapping. In this work, we propose the **S**uccessor Rollout **A**ggregation **D**eep **Q**-Network (SADQ), a simple modification to Q-learning that regularizes how the TD target is formed. SADQ uses one-step rollout predictions from a learned dynamics model to guide the comparison among candidate next-state actions, introducing additional structure into the aggregation step without altering the underlying learning framework. The resulting mixed Bellman update attenuates unreliable maxima while preserving the standard fixed point under diminishing model error. We provide theoretical analysis showing that SADQ reduces bootstrap-induced overestimation in a pointwise manner. Empirically, SADQ consistently improves training stability across classical control tasks, real-world vector-based environments, and Atari benchmarks when compared to strong DQN variants.

1 INTRODUCTION

Q-learning and its deep variants, such as Deep Q-Networks (DQN), learn action-value functions through bootstrapped temporal-difference (TD) updates [Mnih et al., 2015, Liang et al., 2022]. In these methods, future returns are approximated using a greedy backup, $\max_{a'} Q(s', a')$, rather than being directly observed [Moore and Atkeson, 1993, Ghiassian et al., 2020]. This mechanism has proven effective in practice, but it also introduces a well-known source of instability when combined with function approximation [Maei et al., 2009, Lee and He, 2019, Rowland et al., 2023].

A large body of work has focused on improving the quality of Q-value estimation. Strategies such as prioritized sampling [Schaul et al., 2016], multi-step returns [Mnih et al., 2016], architectural refinements including Double and Dueling DQN [Van Hasselt et al., 2016, Wang et al., 2016], and distributional methods [Bellemare et al., 2017, Dabney et al., 2018b] have significantly improved empirical performance and stability. These methods primarily aim to reduce estimation uncertainty or better characterize predictive dispersion at the level of individual value predictions. However, even when value estimates are improved, the TD target is still formed by a hard maximization over next-state actions. This maximization implicitly assumes that the relative ordering of Q-values is reliable. In practice, especially during learning, Q-values are often noisy and uncertain. Under such conditions, selecting actions solely based on their maximum estimated value can be brittle: the greedy operator may favor actions that appear optimal due to transient overestimation rather than genuine long-term advantage. This uncertainty is recursively amplified through bootstrapped updates, leading to biased targets and unstable learning dynamics.

This observation suggests that the challenge is not only about estimating Q-values accurately, but also about how candidate actions are ranked under uncertainty when forming the TD target. In particular, the standard greedy backup lacks any mechanism to assess the reliability of competing action values. A natural question then arises: can we introduce additional structure into the aggregation step of the TD update, without abandoning the simplicity and efficiency of Q-learning? One source of such structure comes from predicting the immediate consequences of actions under the environment dynamics. Recent advances in learned dynamics models [Ha and Schmidhuber, 2018, Hafner et al., 2025] have shown that short-horizon rollouts can provide meaningful information about action outcomes. Yet, in most

Proceedings of the 42nd Conference on Uncertainty in Artificial Intelligence (UAI 2026), PMLR 337:8330–8348.

existing work, these models are used for planning or data generation, rather than for shaping the TD target itself.

In this paper, we propose *Successor Rollout Aggregation Deep Q-Networks* (SADQ), a simple modification to the TD target construction that incorporates one-step rollout information into action comparison. Instead of relying solely on greedy value maximization, SADQ uses rollout-based evaluations to guide the selection of the next-state action used in the target. Importantly, the rollout is not used to replace Q-values, but to regularize the aggregation process, reducing sensitivity to unreliable maxima while preserving the standard bootstrap framework.

Our contributions can be summarized as follows:

- We identify brittle action aggregation under noisy value estimates as a structural source of instability in Q-learning.
- We propose SADQ, a rollout-guided aggregation mechanism that regularizes greedy TD target construction without altering the underlying learning paradigm.
- We provide theoretical analysis showing that the resulting update reduces maximization-induced bias while preserving the optimal fixed point under diminishing model error.
- We demonstrate consistent empirical improvements across a range of benchmark and real-world decision-making tasks.

2 RELATED WORK

Stabilization Strategies for Q Estimation. The core instability in Q-learning arises from the bootstrapped target, where the hard maximization operator introduces systematic overestimation bias and amplifies estimation variance [Mnih et al., 2015, Sutton and Barto, 2018]. Early work such as Double DQN mitigates overestimation bias in Q-value predictions [Van Hasselt et al., 2016], while Dueling DQN separates the state-value and advantage functions [Wang et al., 2016]. Advancing further, distributional RL techniques, such as C51 [Bellemare et al., 2017] and QR-DQN [Dabney et al., 2018b], model the full return distribution before applying maximization, thereby altering how uncertainty propagates through the max operator [Tang et al., 2022, Schwarzer et al., 2023, Kastner et al., 2025]. Bayesian approaches also emerge, integrating prior knowledge and adaptive exploration strategies into the DQN framework [Cao and Ray, 2012]. Complementary to these innovations, sampling techniques are used to enhance data efficiency and exploration at the sample level [Osband et al., 2016, Schaul et al., 2016, Elvira and Martino, 2021]. Despite these advances, most existing methods eventually revert to the max operation for Q updates. In contrast, our work focuses explicitly on restructuring the max-based TD target itself.

Model-Based and Generative Approaches in RL. Model-based RL exploits learned dynamics models for planning and decision-making [Ha and Schmidhuber, 2018, Schwarzer et al., 2021, Mondal et al., 2024]. For example, Recurrent State-Space Model (RSSM) [Hafner et al., 2019, 2025] combines variational inference with recurrent dynamics to infer compact latent states. Subsequent approaches such as RePo [Zhu et al., 2023] and Denoised MDP [Wang et al., 2022] remove pixel reconstruction and emphasize reward-relevant features to improve tractability and task alignment. To enhance generalization and robustness, several methods introduce information bottlenecks through mutual information regularization across latent representations [Bai et al., 2021, Saanum et al., 2023a, Tang et al., 2023]. Diffusion-based generative models have recently been incorporated into RL as powerful tools for trajectory modeling and synthetic data generation [Janner et al., 2022, Lu et al., 2023b, Wang et al., 2025]. Some approaches integrate policy or value guidance into diffusion processes to bias generation toward task-relevant behaviors [Rigter et al., 2024, Chen et al., 2023]. Others adopt energy-based objectives to guide generative modeling for RL learning [Lu et al., 2023a, Liu et al., 2024]. In contrast to approaches that leverages predictive models for long-horizon planning, our method employs one-step successor rollouts and integrates them directly into TD target aggregation, thereby modifying the update rule rather than the planning process.

Relation to Path-Consistency Methods. The proposed aggregation bears conceptual similarity to path-consistency approaches such as PCZero [Zhao et al., 2023], which reduce variance by enforcing consistency along selected trajectories. PCZero constructs a window over historical and search-expanded states and minimizes value variation within that window, reflecting the principle that values along an optimal path should agree [Zhao et al., 2022]. The mixed TD target in Eq. (17) can be viewed as a single-step analogue of this idea. Given the set of rollout-informed estimates $\{\widetilde{Q'}_\phi(s', a')\}_{a' \in \mathcal{A}}$, we identify the action whose predicted successor outcome is most aligned with the greedy backup, and interpolate between this action and the standard maximizer. In contrast to multi-step or window-based consistency, our approach operates at the finest temporal granularity of a single transition, directly regularizing the aggregation step in the TD update.

Relation to Model-based Value Expansion. SADQ is related to model-based value expansion methods in that both use predictive model information to improve value learning. Methods such as MVE [Palenicek et al., 2022] and STEVE [Buckman et al., 2018] use learned dynamics and reward models to construct expanded value targets by accumulating predicted rewards over rollout horizons and then bootstrapping from the terminal predicted state. In contrast, SADQ does not use the model to expand the

Bellman backup over multiple rollout steps. The auxiliary model provides a one-step lookahead score for each candidate next action, which is used to refine the action-selection step inside the max-based TD target. Thus, SADQ shares the motivation of leveraging predictive structure for value learning, but differs in where the model enters the update. Value expansion modifies the target value through rollout-based reward accumulation, whereas SADQ modifies the action aggregation process while keeping the final bootstrapped value evaluation within the Q-learning framework.

3 PRELIMINARIES

3.1 REINFORCEMENT LEARNING

RL is a machine learning paradigm where an agent learns to make decisions through the interaction with the environment [Matsuo et al., 2022, Saanum et al., 2023b, Zhang et al., 2024]. Typically, the RL problem is modeled as a Markov Decision Process (MDP), characterized by a tuple (S, A, P, R, γ). Here, S represents the set of states, A indicates the set of actions, P denotes the state transition probabilities, R specifies the reward function, and γ serves as the discount factor. The objective of the agent is to learn an optimal policy $\pi^*(s)$ that maximizes the expected cumulative reward over time.

RL methods can be broadly categorized into policy-based [Schulman et al., 2015], value-based [Mnih et al., 2015], and actor-critic approaches [Haarnoja et al., 2018]. Policy-based methods directly learn a policy $\pi(a|s)$ that maps states to actions, whereas value-based methods focus on estimating a value function, such as the Q-function $Q(s, a)$, to estimate the quality of actions. Given that this work centers on DQN, our discussion primarily emphasizes value-based methods and their relevance to optimal policy learning.

3.2 DEEP Q-BASED RL

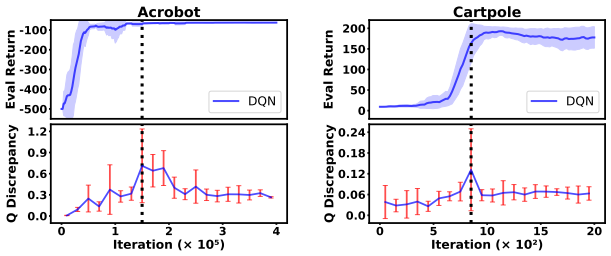
As a foundational deep Q-based RL algorithm, Deep Q-Network (DQN) [Mnih et al., 2015] integrates Q-learning with deep neural networks to facilitate decision-making in high-dimensional state spaces. Particularly, DQN is comprised of two distinct neural networks: the primary Q-network and the target Q-network. The primary Q-network, simply referred to as the Q-network, approximates the Q-value $Q(s, a)$. To stabilize training, a separate target Q network, denoted as $Q'(s, a)$, computes the target Q-value y independently. The optimization objective for training the Q-network is defined as minimizing the TD error, given by:

$$\mathcal{L}_Q = \mathbb{E}_{(s,a,r,s')\in D}[y - Q(s, a)], \tag{1}$$

where (s, a, r, s') is the sampled transition from replay buffer $\mathcal{D}$. The target Q-value y in Eq. (1) is computed by the target Q-network Q' and is formulated as:

$$y = r + \gamma \max_{a'} Q_{\text{target}}(s', a'), \tag{2}$$

The maximization operator in Eq. (2) determines how future value estimates are aggregated into the TD target, making it a critical component of the learning dynamics.



(a) (b)

Figure 1: Upper: Training performance of DQN on Acrobot and Cartpole environments. Lower: The evaluation of Q discrepancy ($\max Q(s, a) - \min Q(s, a)$).

4 MOTIVATION

Observation: Fig. 1 shows that DQN training typically exhibits a rapid improvement phase followed by a stable phase. To examine how value estimation evolves during this transition, we track the Q discrepancy $\Delta_Q(s) = \max_a Q(s, a) - \min_a Q(s, a)$, which directly depends on the max-Q term used in the TD target. As shown in the lower panels, Δ_Q increases during the performance rising phase, reaches a peak near the transition point (dashed line), and then decreases to a stable level together with the evaluation return. Since $\max_a Q(s, a)$ is the quantity selected in the bootstrap target, the alignment between performance dynamics and Δ_Q indicates that training behavior is closely tied to the magnitude of the maxQ statistic. In particular, the overshoot and subsequent contraction of Δ_Q suggest that excessive amplification of the maximized Q-value is not sustained in the stable regime, motivating a closer examination of how max-Q enters the TD update.

Insight: The key point is the large separation among action values is temporary. Once learning stabilizes, the discrepancy contracts rather than remaining large. This indicates that aggressively amplifying the largest Q-value is not a prerequisite for good performance, but instead emerges during learning when value estimates are still uncertain. During this phase, action values are noisy and continually reshaped, yet the maximization operator treats their ordering as fully reliable. By selecting a single largest estimate, the TD target can be dominated by transient peaks that are not well supported by the underlying dynamics. These peaks are then reinforced through bootstrapping, even though they

eventually fade as learning progresses. The difficulty therefore lies not only in estimating Q-values accurately, but in how actions are compared when those estimates are still unreliable.

Such observation raises a simple question: can the next-state evaluation used in the TD target be made more robust than a single maximized value, without abandoning the basic structure of Q-learning? Addressing this question motivates reconsidering how predicted future consequences are incorporated into the TD update, and leads to the rollout-guided aggregation approach developed in this work.

5 METHOD

In this section, we present the Successor Rollout Aggregation Mechanism (SADQ). We first introduce the learned dynamics model to generate one-step successor states and rewards (Section 5.1 for vector-based tasks; Section 5.2 for image-based tasks). Based on these rollout predictions, we construct a structurally aggregated TD target for DQN and its distributional variants (Section 5.3 and Section 5.4, respectively). In the end, the theoretical analysis of the proposed mechanism is provided (Section 5.5).

5.1 VECTOR-STATE DYNAMICS MODEL

Our goal is not to perform multi-step rollout or long-horizon planning, but to obtain a lightweight, action-conditional signal that reflects how different actions are expected to unfold in the next step. In environments where the state is represented as a low-dimensional vector, this can be achieved by modeling the dynamics directly in the original state space.

We therefore learn a stochastic dynamics model $\mathcal{M}$ that predicts both the successor state distribution $P(s' \mid s, a)$ and the immediate reward. For state transitions, we model the dominant stochasticity using a Gaussian distribution,

$$P(s' \mid s, a) \sim \mathcal{N}(\mu(s, a), \sigma^2(s, a)), \tag{3}$$

where the mean $\mu(s, a)$ and variance $\sigma^2(s, a)$ are parameterized by neural networks f_μ and f_σ:

$$\mu(s, a) = f_\mu(s, a), \tag{4}$$

$$\sigma^2(s, a) = \exp(f_\sigma(s, a)). \tag{5}$$

Given a state–action pair (s, a), a successor state $s'_{\mathcal{M}}$ is sampled using the reparameterization trick,

$$s'_{\mathcal{M}} = \mu(s, a) + \sigma(s, a) \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \tag{6}$$

and the immediate reward is predicted by a separate head,

$$r_{\mathcal{M}} = f_r(s, a). \tag{7}$$

The dynamics model is trained using supervised losses on both the successor state and the reward,

$$\mathcal{L}_{\mathcal{M}} = \text{MSE}(s'_{\mathcal{M}}, s') + \text{MSE}(r_{\mathcal{M}}, r), \tag{8}$$

where (s, a, r, s') are sampled from the replay buffer. Once trained, $\mathcal{M}$ provides one-step rollout predictions $(s'_{\mathcal{M}}, r_{\mathcal{M}})$ that serve as auxiliary signals when forming the TD target, complementing the maximized Q-value.

5.2 IMAGE-STATE DYNAMICS MODEL

In image-based environments, the system state is not directly observable and must be inferred from high-dimensional observations. To obtain one-step successor predictions, we operate in a learned latent space. While recurrent state-space models (RSSMs) [Hafner et al., 2019] are commonly used for long-horizon dynamics modeling, our objective here is more limited. We require a compact latent representation that supports reliable one-step prediction for TD target aggregation, rather than multi-step imagination or planning.

Accordingly, we adopt a simplified RSSM-style latent dynamics model focusing exclusively on immediate transition consequences. The model takes the current latent state x_s and action a as input, and produces predictions of the successor latent state and reward. By restricting the temporal scope to a single step, we avoid introducing recurrent structure that would add parameterization without providing corresponding benefit for our setting. Specifically, given the one-hot action embedding e_a, the latent dynamics are defined as

$$\begin{aligned} \textit{Feature model:} \quad & h = f_\phi(x_s, e_a), \\ \textit{Encoder:} \quad & z \sim q_\phi(z \mid h, x_{s'}), \\ \textit{Dynamics predictor:} \quad & \hat{z} \sim p_\phi(\hat{z} \mid h), \\ \textit{Reward predictor:} \quad & \hat{r} \sim p_\phi(\hat{r} \mid h, z), \\ \textit{Decoder:} \quad & \hat{x}_{s'} \sim p_\phi(\hat{x}_{s'} \mid h, z). \end{aligned} \tag{9}$$

The encoder $q_\phi(z \mid h, x_{s'})$ and the dynamics predictor $p_\phi(\hat{z} \mid h)$ together capture the stochastic transition structure in the latent space, while the decoder maps latent predictions back to the observation space. A separate reward head predicts the immediate reward associated with the transition. Importantly, this model is trained and used strictly for one-step prediction, and is not employed for trajectory rollout or policy optimization.

The dynamic model parameters ϕ are learned end-to-end using a weighted combination of prediction, dynamics, and representation losses, following the general structure of Dreamer-style training [Hafner et al., 2025]:

$$\mathcal{L}(\phi) = \beta_{\text{pred}}\, \mathcal{L}_{\text{pred}} + \beta_{\text{dyn}}\, \mathcal{L}_{\text{dyn}} + \beta_{\text{rep}}\, \mathcal{L}_{\text{rep}}, \tag{10}$$

where we set $\beta_{\text{pred}} = 1$, $\beta_{\text{dyn}} = 1$, and $\beta_{\text{rep}} = 0.1$. The

individual loss terms are given by

$$\begin{aligned}\mathcal{L}_{\text{pred}}(\phi) &= \mathcal{L}_{\text{recon}}(\phi) + \mathcal{L}_{\text{rew}}(\phi)\\ &= -\ln p_\phi(x_{s'} \mid h, z) - \ln p_\phi(r \mid h, z),\\ \mathcal{L}_{\text{dyn}}(\phi) &= \max\Big(1,\ \mathrm{KL}\big[\mathrm{sg}(q_\phi(z \mid h, x_{s'}))\ \big\|\ p_\phi(z \mid h)\big]\Big),\\ \mathcal{L}_{\text{rep}}(\phi) &= \max\Big(1,\ \mathrm{KL}\big[q_\phi(z \mid h, x_{s'})\ \big\|\ \mathrm{sg}(p_\phi(z \mid h))\big]\Big).\end{aligned} \tag{11}$$

To reduce the computational overhead of dynamics model training, we adapt the update frequency of the dynamics model based on the reconstruction component of $\mathcal{L}_{\text{pred}}$, controlled by a threshold $\tau_{\mathcal{M}}$. Specifically, if

$$\mathcal{L}_{\text{recon}}(\phi) = -\ln p_\phi(x_{s'} \mid h, z) < \tau_{\mathcal{M}}, \tag{12}$$

the update frequency is reduced to a fraction $t_f \in (0, 1)$ of the default schedule; otherwise, the dynamics model follows the default update schedule. This adaptive schedule preserves frequent updates when the model has not yet learned reliable observation reconstruction, while reducing redundant updates after the reconstruction quality becomes sufficiently accurate.

Once the dynamics model is trained, the latent dynamics model provides one-step successor state and reward predictions that are used solely to inform TD target aggregation. The model does not alter the underlying Q-learning update, but supplies auxiliary predictive structure that helps regularize action comparison under uncertain value estimates.

5.3 STRUCTURAL AGGREGATION TARGET

In standard Q-learning, the TD target is formed by aggregating future action values through a hard maximization over next-state actions. Rather than modifying the off-policy learning framework itself, we focus on how this aggregation is carried out in the bootstrapped update. Our goal is to introduce additional structure into the comparison among candidate next actions, while leaving the underlying value function approximation unchanged. To this end, we use the learned dynamics model $\mathcal{M}$ to obtain one-step, action-conditional predictions from the observed next state s'. For each candidate action $a' \in \mathcal{A}$, the model produces a predicted successor state $s''_{\mathcal{M}}(a')$ together with an immediate reward estimate. These predictions are not used for planning or policy improvement, but serve as auxiliary evidence when comparing candidate actions in the TD target.

We begin by recalling the Bellman optimality equation for the action-value function,

$$Q^*(s, a) = \mathbb{E}\Big[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\Big]. \tag{13}$$

In practice, Q-learning approximates this relation using the greedy TD target $\max_{a'} Q'_\phi(s', a')$. As discussed earlier, the reliability of this update depends entirely on the quality of the maximized estimate at the observed next state s'.

To regularize this aggregation step, we additionally evaluate the immediate consequences of each candidate next action using the dynamics model $\mathcal{M}_\theta$. For every $a' \in \mathcal{A}$, the model generates a one-step rollout,

$$(r'_{\mathcal{M}}, s''_{\mathcal{M}}) \sim \mathcal{M}_\theta(\cdot \mid s', a'), \tag{14}$$

from which we compute a model-based Bellman estimate,

$$\widetilde{Q'}_\phi(s', a') = r'_{\mathcal{M}} + \gamma \max_{a''} Q'_\phi(s''_{\mathcal{M}}, a''). \tag{15}$$

This quantity provides a rollout-informed evaluation of action a', grounded in its predicted transition outcome rather than solely in the value estimate at s'.

We use these rollout-based estimates only to guide action comparison. Specifically, we select

$$\hat{a}' = \arg\max_{a' \in \mathcal{A}} \widetilde{Q'}_\phi(s', a'), \tag{16}$$

and form a mixed TD target,

$$\hat{y} = \alpha \max_{a'} Q'_\phi(s', a') + (1 - \alpha) Q'_\phi(s', \hat{a}'), \tag{17}$$

where $\alpha \in [0, 1]$ controls the interpolation between the standard greedy evaluation and the model-guided action selection. Importantly, the rollout does not replace the value function in the target; it only influences which action is used when forming the update. In this way, SADQ separates action ranking from value evaluation, reducing sensitivity to unreliable maxima while preserving the standard bootstrap structure.

5.4 EXTENSION TO DISTRIBUTIONAL DQN

We next extend the proposed aggregation mechanism to the distributional setting, where the target network predicts a return distribution $Z'_\phi(s, a)$ rather than a scalar value. In distributional DQN, greedy evaluation is typically performed with respect to the expected return,

$$\max_{a'} \mathbb{E}\big[Z'_\phi(s', a')\big], \tag{18}$$

and the TD target is formed by selecting the action with the largest expectation.

As in the scalar case, our objective is not to alter the underlying distributional learning framework, but to regularize how candidate actions are compared when forming the target. To this end, we use the learned dynamics model $\mathcal{M}_\theta$ to obtain one-step, action-conditional successor predictions from the observed next state s'. For each $a' \in \mathcal{A}$, the model produces

$$(r'_{\mathcal{M}}, s''_{\mathcal{M}}) \sim \mathcal{M}_\theta(\cdot \mid s', a'). \tag{19}$$

Based on these predictions, we construct a rollout-informed distributional Bellman estimate,

$$\widetilde{Z'}_\phi(s', a') = r'_{\mathcal{M}} + \gamma\, Z'_\phi(s''_{\mathcal{M}}, a^{**}), \tag{20}$$

where $a^{**} = \arg\max_{a''} \mathbb{E}\left[Z'_\phi(s''_{\mathcal{M}}, a'')\right]$. This estimate reflects the predicted immediate transition outcome of action a', while retaining the distributional representation of future returns.

We use the expectation of these rollout-informed distributions to guide action comparison,

$$\hat{a}' = \arg\max_{a' \in \mathcal{A}} \mathbb{E}\left[\widetilde{Z'}_\phi(s', a')\right], \tag{21}$$

and construct a mixed distributional target,

$$\hat{Z} = \alpha\, Z'_\phi(s', a^*) + (1-\alpha)\, Z'_\phi(s', \hat{a}'), \tag{22}$$

where $a^* = \arg\max_{a'} \mathbb{E}\left[Z'_\phi(s', a')\right]$.

This formulation preserves the same structural aggregation principle as in the scalar case. The dynamics model is used only to influence which action is selected for the target, while the value distribution itself is always provided by the distributional critic. In this way, the proposed mechanism extends naturally from scalar Q-values to return distributions, regularizing greedy action comparison without modifying the distributional update rule.

5.5 THEORETICAL ANALYSIS

We provide a theoretical analysis to clarify how the proposed aggregation affects bootstrap bias, and to verify that it does not alter the optimal solution asymptotically. All formal proofs are deferred to **Appendix A**.

Let $\mathcal{M}_\theta(\cdot \mid s, a)$ denote a dynamics model trained on the support of the replay buffer $\mathcal{D}$. For notational convenience, define the standard and model-based TD targets

$$y_{\mathcal{M}} = r_{\mathcal{M}} + \gamma \max_{a'} Q'_\phi(s'_{\mathcal{M}}, a'), \tag{23}$$

$$y = r + \gamma \max_{a'} Q'_\phi(s', a'), \tag{24}$$

where $(r, s') \sim \mathcal{T}(\cdot \mid s, a)$ follows the true environment dynamics and $(r_{\mathcal{M}}, s'_{\mathcal{M}}) \sim \mathcal{M}_\theta(\cdot \mid s, a)$ is generated by the learned dynamics model.

Lemma 1 (Local state extrapolation). Under the neural tangent kernel (NTK) regime [Jacot et al., 2018], for any in-sample state-action pair $(s, a) \in \mathcal{D}$ and in-neighborhood state-action pair $(s_{\mathcal{M}}, a)$ such that $\|s - s_{\mathcal{M}}\| \le \epsilon$, the value difference of the deep Q function can be bounded as:

$$\|Q_\phi(s'_{\mathcal{M}}, a') - Q_\phi(s', a')\| \le C\left(\sqrt{\min\left(\|s' \oplus a'\|, \|s'_{\mathcal{M}} \oplus a'\|\right)}\sqrt{\epsilon} + 2\epsilon\right), \tag{25}$$

where $\oplus$ denotes the vector concatenation operation, and C is a finite constant. The **Lemma 1** is a direct corollary of Theorem 1 in Li et al. [2023], specialized to the case of local state extrapolation.

Algorithm 1 SADQ

Input: Online Q-network $Q(s, a; \theta)$, target Q-network $Q(s, a; \theta')$, replay buffer $\mathcal{D}$, dynamics model $\mathcal{M}$.
Output: Optimized Q-network $Q(s, a; \theta^*)$.

Training loop:
while not done **do**
  Sample transitions $\{(s, a, r, s')\}$ from $\mathcal{D}$ for $\mathcal{M}$.
  Train model $\mathcal{M}$ by Eq. (8) or Eq. (10).
  Resample transitions $\{(s, a, r, s')\}$ from $\mathcal{D}$ for Q.
  Compute successor rollout in Eq. (14).
  Get candidate action $\hat{a}'$ in Eq. (16) or Eq. (21).
  Compute target Q-value y in Eq. (17) or Eq. (22).
  Update Q-networks.
  Periodically update target network Q'.
end while
return Optimized Q-network parameters θ.

Assumption 1 (Controlled model-induced perturbation). We assume that for all $(s, a) \in \mathcal{D}$,

$$\mathbb{E}[\|y_{\mathcal{M}} - y\| \mid (s, a)] \;\le\; \epsilon_r + \gamma\epsilon_s, \tag{26}$$

where ϵ_r bounds the reward prediction error of the learned model, and ϵ_s bounds the state-induced value perturbation characterized by **Lemma 1**, respectively.

Theorem 1 (Bootstrap bias reduction and ideal-limit target consistency). Let $\mathcal{T}_{\text{std}}$ and $\mathcal{T}_{\text{mix}}$ denote the standard Bellman operator and the mixed operator induced by Eq. (17), respectively. Then, the following properties hold:

1. For any bounded action-value function Q, the mixed operator satisfies
$$\mathcal{T}_{\text{mix}} Q \;\le\; \mathcal{T}_{\text{std}} Q \quad \text{pointwise.}$$
2. In the ideal limiting case where the model-induced perturbation approach zero in **Assumption 1**, then
$$\lim_{k \to \infty} \left\| \mathcal{T}_{\text{mix}}^{(k)} Q^* - \mathcal{T}^* Q^* \right\|_\infty = 0.$$

The first property shows that, for any given Q, the mixed operator produces a TD target no larger than the standard greedy backup. This indicates that SADQ attenuates the upward bias caused by unreliable maximized value estimates. The second property is an ideal-limit target consistency result. When the model-induced perturbation approaches zero, the mixed target approaches the optimal Bellman target. This does not imply a finite-error fixed-point guarantee; with nonzero residual model error, the induced operator may differ from the optimal Bellman operator.

5.6 SADQ ALGORITHM

The Algorithm 1 summarizes how SADQ modifies TD target construction in standard Q-learning. Instead of relying

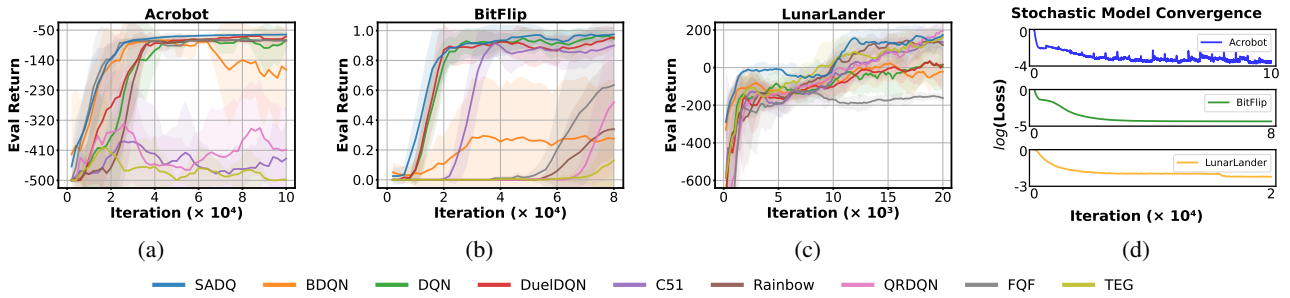


Figure 2: Performance comparison of SADQ with other baselines across conventional RL tasks. The legend above describes the corresponding methods in (a) Acrobot, (b) BitFlip, (c) LunarLander, and (d) Convergency of the stochastic model.

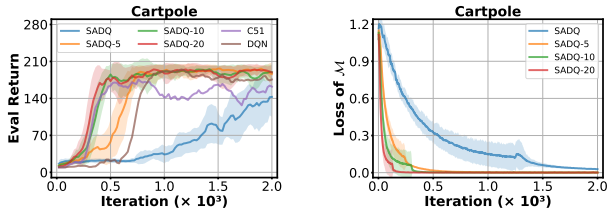


Figure 3: Effects of stochastic model convergence to performance. (a) Effects of updating frequency to SADQ and compare with C51 and DQN; (b) The Loss of stochastic model among different updating configurations.

solely on the max operator over next-state Q-values, SADQ uses one-step successor rollout information from the auxiliary dynamics model to guide action aggregation. The Q-network is then updated with the resulting mixed TD target, while the remaining replay-based training pipeline follows the standard DQN-style procedure.

6 EXPERIMENTS

We first evaluate SADQ on discrete-action vector-based benchmarks implemented in DI-Engine[1]. In this setting, SADQ is built upon the Dueling DQN backbone [Wang et al., 2016]. The environments include Acrobot [Sutton, 1995], BitFlip, Cartpole [Barto et al., 1983], and LunarLander, along with two real-world-inspired scenarios, CityFlow [Tang et al., 2019] and O-Cloud. Detailed environment descriptions are provided in **Appendix B**.

We compare SADQ against representative value-based baselines from two major categories. **(1) Classical DQN variants:** DQN [Mnih et al., 2015], Double DQN [Van Hasselt et al., 2016], Dueling DQN [Wang et al., 2016], BDQN [Azizzadenesheli et al., 2018], and SUNRISE [Lee et al., 2021]. **(2) Distributional DQN methods:** C51 [Bellemare et al., 2017], Rainbow [Hessel et al., 2018], QRDQN [Dabney et al., 2018b], FQF [Yang et al., 2019], and TEG [Dabney et al., 2021]. This setup allows us to evaluate SADQ against both classical value estimators and distributional formulations.

To further assess generality in high-dimensional visual domains, we conduct experiments on the Atari100K benchmark [Bellemare et al., 2013], implemented in Jax-baseline [2]. In the image-based setting, SADQ is applied as a plug-in modification to multiple value-based agents. Specifically, we replace the standard TD target in QRDQN [Dabney et al., 2018b], IQN [Dabney et al., 2018a], and CoAct Korkmaz [2025b] with the proposed mixed target, while preserving their original architectures, distributional parameterizations, and optimization procedures. This design isolates the effect of SADQ at the target-construction level and enables evaluation across different quantile-based estimators.

All experiments are conducted with five independent random seeds. We report mean performance across seeds, and full hyperparameter configurations are provided in **Appendix C** for reproducibility.

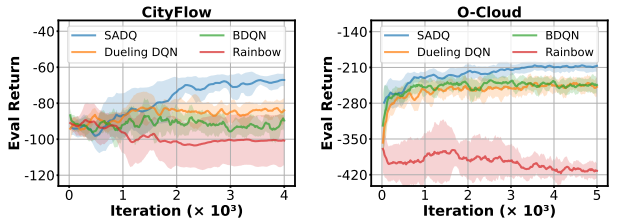
6.1 CONVENTIONAL RL TASKS

We evaluate SADQ across multiple control environments. On Acrobot, BitFlip, and LunarLander (Figs. 2a–2c), SADQ attains the highest or competitive final returns while exhibiting more stable training dynamics than representative DQN variants, which often show rapid early gains followed by performance degradation. To isolate the contribution of SADQ, we compare multiple baselines with and without SADQ under identical experimental settings, where the only difference is the max-operator-related component introduced by SADQ. As shown in **Appendix D**, SADQ consistently improves all baselines.

To examine the role of the vector-based stochastic model

[1] https://github.com/opendilab/DI-engine

[2] https://github.com/tinker495/jax-baseline



(a) (b)

Figure 4: Performance comparison of SADQ with baselines across CityFlow and O-Cloud scenarios. (a) Performance in CityFlow scenario; (b) Performance in O-Cloud scenario.

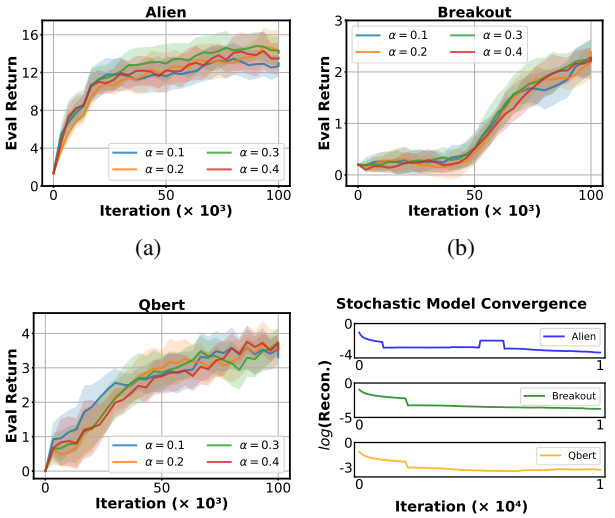
$\mathcal{M}$, we vary its update frequency k in Cartpole. As shown in Fig. 3a, larger k consistently accelerates convergence and improves final performance, with $k = 20$ achieving the fastest learning. The corresponding loss curves (Fig. 3b) confirm faster and more stable model convergence as k increases. Together, these results indicate that more accurate successor prediction leads to improved stability and effectiveness of the SADQ update. See **Appendix E** for further discussion of Cartpole.

6.2 REAL WORLD SCENARIOS

We further evaluate SADQ in two real-world scenarios: CityFlow for traffic signal control and O-Cloud for dynamic resource allocation. These experiments are designed to assess the practical applicability of SADQ in complex, structured decision-making environments and to compare its performance against representative baseline methods under realistic operational conditions. In both scenarios (Figs. 4a-4b), SADQ demonstrates superior performance compared to other baselines, including Dueling DQN, BDQN, and Rainbow. During the whole training process, SADQ achieves the highest evaluation return, consistently outperforming the baselines throughout training.

6.3 ATARI100K GAMES

Tab. 1 reports results on Atari100K games for QRDQN, IQN, and CoAct with and without SADQ. Across all three estimators, SADQ improves both mean and median performance, with gains on 20/24 games for QRDQN, 18/24 for IQN, and 22/24 for CoAct. These results demonstrate that successor-guided target formation consistently enhances robustness in quantile-based value learning. We further analyze the effect of the mixing coefficient α on three representative games (Alien, Breakout, and Qbert). As shown in Figs 5a–5c, SADQ yields smoother learning dynamics and reduced variance for all tested α, indicating robustness to the choice of mixing weight. Fig 5d shows that the simplified RSSM-style model converges rapidly (∼2k iterations),



(a) (b) (c) (d)

Figure 5: Case study on Atari environments. (a) α-ablation on Alien; (b) α-ablation on Breakout; (c) α-ablation on Qbert; (d) Convergence of the stochastic dynamics model measured by log reconstruction loss across three games.

suggesting that the rollout signal becomes reliable early and effectively regularizes TD target construction.

In our implementation, $\mathcal{M}$ is instantiated as the simplified one-step RSSM-style model described in Sec. 5.2. The full RSSM contains approximately 62M parameters, whereas our one-step dynamics model contains approximately 34M parameters. To further examine the computational overhead of training the dynamics model $\mathcal{M}$, we report the wall-clock training time of SADQ under different dynamics-model update frequencies. Tab. 2 compares the runtime with $t_f = 40\%$ and $t_f = 20\%$ on Atari-100K and Atari-10M. The results show that the adaptive scheduling mechanism maintains sufficient state-prediction accuracy while substantially reducing the computational overhead of dynamics-model training.

Table 2: Runtime comparison of QR-DQN and SADQ with different reduced update fractions t_f.

	QR-DQN	**+SADQ** ($t_f = 40\%$)	**+SADQ** ($t_f = 20\%$)
Atari-100K	4 mins	6 mins	5 mins
Atari-10M	434 mins	598 mins	522 mins

7 DISCUSSION

SADQ relies on an auxiliary dynamics model $\mathcal{M}$ for one-step successor prediction, which introduces a trade-off between prediction reliability and computational overhead. If

Table 1: Performance on Atari100K. Normalized scores are averaged over 20 evaluations per seed.

	QRDQN		IQN		CoAct	
Game	Base	+SADQ	Base	+SADQ	Base	+SADQ
Alien	12.49	**14.22**	10.08	**10.98**	10.74	**11.54**
Amidar	2.48	**3.15**	2.61	**3.04**	2.06	**2.85**
Assault	4.82	**4.86**	**4.80**	4.63	3.79	**4.03**
Asterix	2.27	**2.45**	1.62	**1.74**	1.75	**2.07**
BankHeist	**0.30**	0.23	0.08	0.08	0.15	**0.21**
BattleZone	0.41	**0.61**	0.42	**0.46**	0.70	**0.99**
Boxing	2.62	**3.92**	-31.27	**-26.91**	2.84	**3.43**
Breakout	2.00	**2.38**	2.30	**2.31**	1.38	**1.43**
ChopperCommand	1.83	**2.02**	**1.63**	1.37	1.80	**1.88**
CrazyClimber	15.23	**23.11**	16.12	**29.93**	24.17	**28.81**
DemonAttack	3.82	**4.09**	**4.12**	4.01	**3.46**	3.23
Gopher	3.91	**4.99**	2.83	**3.41**	7.18	**7.56**
Hero	1.63	**3.30**	1.06	**2.08**	3.09	**6.40**
Jamesbond	0.16	**0.24**	0.18	**0.22**	0.17	**0.18**
Kangaroo	**0.26**	0.22	**0.04**	0.02	**0.35**	0.24
Krull	76.34	**77.04**	67.17	**69.77**	74.60	**77.81**
KungFuMaster	7.70	**8.93**	1.60	**4.65**	1.74	**2.05**
MsPacman	16.23	**17.30**	15.46	**17.09**	12.93	**13.21**
Pong	-14.96	**-14.27**	-20.70	-20.70	-16.88	**-16.66**
PrivateEye	**-1.35**	-1.85	-21.00	**-8.46**	-4.91	**-2.95**
Qbert	3.42	**3.54**	2.25	**3.04**	2.11	**2.30**
RoadRunner	**1.66**	1.65	2.08	**2.69**	0.91	**0.93**
Seaquest	2.80	**3.22**	1.84	**2.59**	2.48	**2.63**
UpNDown	5.28	**5.55**	4.35	**4.57**	4.29	**4.56**
Mean	6.31	**7.05**	2.90	**4.69**	5.89	**6.63**

$\mathcal{M}$ has not sufficiently converged, its predictions may perturb action comparison, especially in simple tasks such as Cartpole where standard DQN-style updates already provide a strong learning signal. This trade-off comes from introducing auxiliary predictive information, rather than from any particular model architecture. SADQ only requires a one-step predictive signal, and the relative overhead can be further mitigated in ensemble-DQN settings or pre-collected data scenarios, where the auxiliary model can be trained offline or reused more efficiently.

The scope of SADQ is motivated by the continuing role of bootstrapped TD targets in value-based reinforcement learning. Our experiments focus on DQN-based agents because they expose the max-based action selection step most directly, and distributional variants preserve the same target-selection issue when return distributions are used for action evaluation. This focus does not make the problem narrow. Recent work still revisits the stability, efficiency, and statistical behavior of deep TD learning and Q-based methods, suggesting that target construction remains a relevant design point in modern RL [Korkmaz, 2025a, Gallici et al., 2025, Kastner et al., 2025]. SADQ contributes to this line by addressing how bootstrapped targets can be made less sensitive to noisy value estimates during action aggregation.

8 CONCLUSION

In this work, we revisited TD target construction through the perspective of action aggregation under uncertainty and identified hard maximization over noisy Q-values as a structural source of instability in deep Q-learning. Rather than redesigning value estimators, we introduced SADQ, a lightweight successor-rollout aggregation mechanism that regularizes greedy target formation while preserving the standard bootstrap framework. Theoretically, we show that the resulting mixed Bellman operator is pointwise no more optimistic than the standard backup and recovers the optimal Bellman target in the ideal limit as model error approaches zero. Empirically, across classical control tasks, real-world scenarios, and Atari100K benchmarks, SADQ consistently improves training stability and final performance, and integrates seamlessly as a plug-in modification to both DQN and distributional variants. These results suggest that restructuring the aggregation step of the TD update itself offers a simple and principled path toward more robust Q-learning.

Acknowledgements

The work of Xiaonan Zhang is partially supported by NSF under grants CCF-2312617, CNS-2431553, and IIS-2544108. We thank the anonymous reviewers for their helpful feedback.

References

Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. In *Information Theory and Applications Workshop*, pages 1–9. IEEE, 2018.

Chenjia Bai, Lingxiao Wang, Lei Han, Animesh Garg, Jianye Hao, Peng Liu, and Zhaoran Wang. Dynamic bottleneck for robust self-supervised exploration. *Advances in Neural Information Processing Systems*, 34: 17007–17020, 2021.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 834–846, 1983.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in Neural Information Processing Systems*, 31, 2018.

Feng Cao and Soumya Ray. Bayesian hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 25, 2012.

Huayu Chen, Cheng Lu, Chengyang Ying, Hang Su, and Jun Zhu. Offline reinforcement learning via high-fidelity generative behavior modeling. In *International Conference on Learning Representations*, 2023.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pages 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Will Dabney, Georg Ostrovski, and Andre Barreto. Temporally-extended ε-greedy exploration. In *International Conference on Learning Representations*, 2021.

Víctor Elvira and Luca Martino. *Advances in Importance Sampling*, pages 1–14. John Wiley & Sons, Ltd, 2021.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Foerster, and Mario Martin. Simplifying deep temporal difference learning. In *International Conference on Learning Representations*, 2025.

Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient temporal-difference learning with regularized corrections. In *International Conference on Machine Learning*, pages 3524–3534. PMLR, 2020.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in Neural Information Processing Systems*, 31, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pages 9902–9915. PMLR, 2022.

Tyler Kastner, Mark Rowland, Yunhao Tang, Murat A Erdogdu, and Amir-massoud Farahmand. Categorical distributional reinforcement learning with kullback-leibler divergence: Convergence and asymptotics. In *International Conference on Machine Learning*, pages 29294–29320. PMLR, 2025.

Ezgi Korkmaz. Counteractive rl: Rethinking core principles for efficient and scalable deep reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 24330–24348, 2025a.

Ezgi Korkmaz. Counteractive RL: Rethinking core principles for efficient and scalable deep reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 24330–24348, 2025b.

Donghwan Lee and Niao He. Target-based temporal-difference learning. In *International Conference on Machine Learning*, pages 3713–3722. PMLR, 2019.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pages 6131–6141. PMLR, 2021.

Jianxiong Li, Xianyuan Zhan, Haoran Xu, Xiangyu Zhu, Jingjing Liu, and Ya-Qin Zhang. When data geometry meets deep function: Generalizing offline reinforcement learning. In *International Conference on Learning Representations*, 2023.

Litian Liang, Yaosheng Xu, Stephen McAleer, Dailin Hu, Alexander Ihler, Pieter Abbeel, and Roy Fox. Reducing variance in temporal-difference value estimation via ensemble of deep networks. In *International Conference on Machine Learning*, pages 13285–13301. PMLR, 2022.

Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *International Conference on Distributed Computing Systems*, pages 372–382. IEEE, 2017.

Xu-Hui Liu, Tian-Shuo Liu, Shengyi Jiang, Ruifeng Chen, Zhilong Zhang, Xinwei Chen, and Yang Yu. Energy-guided diffusion sampling for offline-to-online reinforcement learning. In *International Conference on Machine Learning*, pages 31541–31565. PMLR, 2024.

Cheng Lu, Huayu Chen, Jianfei Chen, Hang Su, Chongxuan Li, and Jun Zhu. Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. In *International Conference on Machine Learning*, pages 22825–22855. PMLR, 2023a.

Cong Lu, Philip Ball, Yee Whye Teh, and Jack Parker-Holder. Synthetic experience replay. *Advances in Neural Information Processing Systems*, 36:46323–46344, 2023b.

Hamid Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. *Advances in Neural Information Processing Systems*, 22, 2009.

Yixiu Mao, Yun Qu, Qi Wang, and Xiangyang Ji. Adaptive neighborhood-constrained q learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 38:26008–26041, 2025.

Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. Deep learning, reinforcement learning, and world models. *Neural Networks*, 152:267–275, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

Arnab Kumar Mondal, Siba Smarak Panigrahi, Sai Rajeswar, Kaleem Siddiqi, and Siamak Ravanbakhsh. Efficient dynamics modeling in interactive environments with koopman theory. In *International Conference on Learning Representations*, 2024.

Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in Neural Information Processing Systems*, 29, 2016.

Daniel Palenicek, Michael Lutter, and Jan Peters. Revisiting model-based value expansion. In *ICLR Workshop on Generalizable Policy Learning in Physical World*, 2022.

Marc Rigter, Jun Yamada, and Ingmar Posner. World models via policy-guided trajectory diffusion. *Transactions on Machine Learning Research*, 2024.

Mark Rowland, Yunhao Tang, Clare Lyle, Rémi Munos, Marc G Bellemare, and Will Dabney. The statistical benefits of quantile temporal-difference learning for value estimation. In *International Conference on Machine Learning*, pages 29210–29231. PMLR, 2023.

Tankred Saanum, Noémi Éltető, Peter Dayan, Marcel Binz, and Eric Schulz. Reinforcement learning with simple sequence priors. *Advances in Neural Information Processing Systems*, 36:61985–62005, 2023a.

Tankred Saanum, Noémi Éltető, Peter Dayan, Marcel Binz, and Eric Schulz. Reinforcement learning with simple sequence priors. *Advances in Neural Information Processing Systems*, 36, 2023b.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021.

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8, 1995.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2018.

Yunhao Tang, Remi Munos, Mark Rowland, Bernardo Avila Pires, Will Dabney, and Marc Bellemare. The nature of temporal difference errors in multi-step distributional reinforcement learning. *Advances in Neural Information Processing Systems*, 35:30265–30276, 2022.

Yunhao Tang, Zhaohan Daniel Guo, Pierre Harvey Richemond, Bernardo Avila Pires, Yash Chandak, Rémi Munos, Mark Rowland, Mohammad Gheshlaghi Azar, Charline Le Lan, Clare Lyle, et al. Understanding self-predictive learning for reinforcement learning. In *International Conference on Machine Learning*, pages 33632–33656. PMLR, 2023.

Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8797–8806, 2019.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Likun Wang, Xiangteng Zhang, Yinuo Wang, Guojian Zhan, Wenxuan Wang, Haoyu Gao, Jingliang Duan, and Shengbo Eben Li. Off-policy reinforcement learning with model-based exploration augmentation. *Advances in Neural Information Processing Systems*, 38:92121–92151, 2025.

Tongzhou Wang, Simon Du, Antonio Torralba, Phillip Isola, Amy Zhang, and Yuandong Tian. Denoised mdps: Learning world models better than the world itself. In *International Conference on Machine Learning*, pages 22591–22612. PMLR, 2022.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.

Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu. Fully parameterized quantile function for distributional reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Shuai Zhang, Heshan Devaka Fernando, Miao Liu, Keerthiram Murugesan, Songtao Lu, Pin-Yu Chen, Tianyi Chen, and Meng Wang. Sf-dqn: provable knowledge transfer using successor feature for deep reinforcement learning. In *International Conference on Machine Learning*, pages 58897–58934. PMLR, 2024.

Dengwei Zhao, Shikui Tu, and Lei Xu. Efficient learning for alphazero via path consistency. In *International Conference on Machine Learning*, pages 26971–26981. PMLR, 2022.

Dengwei Zhao, Shikui Tu, and Lei Xu. Generalized weighted path consistency for mastering atari games. *Advances in Neural Information Processing Systems*, 36: 50346–50357, 2023.

Chuning Zhu, Max Simchowitz, Siri Gadipudi, and Abhishek Gupta. Repo: Resilient model-based reinforcement learning by regularizing posterior predictability. *Advances in Neural Information Processing Systems*, 36: 32445–32467, 2023.

Revisiting TD Target Aggregation under Uncertainty in Q-Learning (Supplementary Material)

Lipeng Zu[1] **Xiaonan Zhang**[1]

[1]Computer Science Department, Florida State University, Tallahassee, Florida, USA

A THEORETICAL ANALYSIS

Lemma 1 (Local state extrapolation). Under the neural tangent kernel (NTK) regime [Jacot et al., 2018], for any in-sample state-action pair $(s, a) \in \mathcal{D}$ and in-neighborhood state-action pair $(s_{\mathcal{M}}, a)$ such that $\|s - s_{\mathcal{M}}\| \leq \epsilon$, the value difference of the deep Q function can be bounded as:

$$\|Q_\phi(s'_{\mathcal{M}}, a') - Q_\phi(s', a')\| \leq C\left(\sqrt{\min\left(\|s' \oplus a'\|, \|s'_{\mathcal{M}} \oplus a'\|\right)}\sqrt{\epsilon} + 2\epsilon\right), \tag{27}$$

where $\oplus$ denotes the vector concatenation operation, and C is a finite constant.

Proof. The lemma follows directly from Theorem 1 or Lemma 4 in Li et al. [2023]. Please refer to Li et al. [2023] for detailed proofs. In addition, NTK remains one of the most influential theoretical frameworks for analyzing the generalization of deep neural networks [Mao et al., 2025]. □

Assumption 1 (Controlled model-induced perturbation). Let $\mathcal{M}_\theta(\cdot \mid s, a)$ be a stochastic dynamics model trained on the replay buffer support $\mathcal{D}$. Assume that for all $(s, a) \in \mathcal{D}$,

$$\mathbb{E}\Big[\|r_{\mathcal{M}} + \gamma \max_{a'} Q'_\phi(s'_{\mathcal{M}}, a') - \big(r + \gamma \max_{a'} Q'_\phi(s', a')\big)\| \,\Big|\, (s, a)\Big] \;\leq\; \epsilon_r + \gamma\epsilon_s, \tag{28}$$

where $(r, s') \sim \mathcal{T}(\cdot \mid s, a)$ denotes the true transition dynamic and $(r_{\mathcal{M}}, s'_{\mathcal{M}}) \sim \mathcal{M}_\theta(\cdot \mid s, a)$ denotes the learned stochastic model. Moreover, ϵ_r bounds the reward prediction error of the learned model, and ϵ_s bounds the state-induced value perturbation characterized by **Lemma 1**, respectively.

Definition 1: (Successor-state aggregation). The standard TD update relies on the greedy evaluation $\max_{a'} Q'_\phi(s', a')$ at the observed next state s'. Here, we additionally employ a stochastic dynamics model $\mathcal{M}_\theta$ to predict the one-step outcomes of all next state-action pairs. For each $a' \in \mathcal{A}$, let $(r'_{\mathcal{M}}, s''_{\mathcal{M}}) \sim \mathcal{M}_\theta(\cdot \mid s', a')$. We define the model-based Bellman estimate

$$\widetilde{Q'}_\phi(s', a') = r'_{\mathcal{M}} + \gamma \max_{a''} Q'_\phi(s''_{\mathcal{M}}, a''). \tag{29}$$

Let $\hat{a}' = \arg\max_{a' \in \mathcal{A}} \widetilde{Q'}_\phi(s', a')$, then the resulting mixed TD target is defined as

$$\hat{y} = \alpha \max_{a'} Q'_\phi(s', a') + (1 - \alpha) Q'_\phi(s', \hat{a}'), \tag{30}$$

where $\alpha \in [0, 1]$ controls the trade-off between the standard greedy target and the model-guided action selection.

Remark 1: Path-consistency methods such as PCZero construct a sliding window that contains historical states and MCTS-scouted nodes [Zhao et al., 2023]. They then minimize the value variance within that window. The goal is to enforce the principle that "values on one optimal path should be identical" [Zhao et al., 2022]. *Definition 1* realizes a one-step aggregation analogue: given the model-generated successor evaluations $\{\widetilde{Q}_\phi(s', a')\}_{a' \in \mathcal{A}}$, we select $\hat{a}' = \arg\max_{a' \in \mathcal{A}} \widetilde{Q}_\phi(s', a')$, that is, the action whose model-based Bellman estimate is most aligned with the greedy backup. The mixed target in Eq. (30) then interpolates between the standard TD greedy evaluation and this model-guided successor aggregation. In this sense, the aggregation step plays a role analogous to the low-variance window averaging in PCZero, but at the finest temporal granularity of a single transition.

Theorem 1 (Bootstrap bias reduction and ideal-limit target consistency). Let $\mathcal{T}_{\mathrm{std}}$ and $\mathcal{T}_{\mathrm{mix}}$ denote the standard Bellman operator and the mixed operator induced by Eq. (17), respectively. Then, the following properties hold:

1. For any bounded action-value function Q, the mixed operator satisfies
$$\mathcal{T}_{\mathrm{mix}} Q \;\le\; \mathcal{T}_{\mathrm{std}} Q \quad \text{pointwise.}$$
2. In the ideal limiting case where the model-induced perturbation approach zero in **Assumption 1**, then
$$\lim_{k \to \infty} \left\| \mathcal{T}_{\mathrm{mix}}^{(k)} Q^* - \mathcal{T}^* Q^* \right\|_\infty = 0.$$

Proof. Define the following Bellman operators acting on bounded functions $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:
$$(\mathcal{T}_{\mathrm{std}} Q)(s, a) := r(s, a) + \gamma\, \mathbb{E}_{s'} \big[\max_{a'} Q(s', a') \big], \tag{31}$$
$$(\mathcal{T}_{\mathrm{mix}} Q)(s, a) := r(s, a) + \gamma\, \mathbb{E}_{s'} \big[\alpha \max_{a'} Q(s', a') + (1 - \alpha) Q(s', \hat{a}'(s')) \big], \tag{32}$$
where $\hat{a}'(s')$ is defined via successor-state aggregation (Definition 1), and $\mathcal{T}^*$ denotes the optimal Bellman operator.

(I) Bias monotonicity.

For any fixed Q and any s',
$$Q(s', \hat{a}'(s')) \le \max_{a'} Q(s', a').$$
Hence,
$$(\mathcal{T}_{\mathrm{mix}} Q)(s, a) - (\mathcal{T}_{\mathrm{std}} Q)(s, a) = \gamma(1 - \alpha)\, \mathbb{E}_{s'} \big[Q(s', \hat{a}'(s')) - \max_{a'} Q(s', a') \big] \le 0.$$
Therefore,
$$\mathcal{T}_{\mathrm{mix}} Q \le \mathcal{T}_{\mathrm{std}} Q \quad \text{pointwise.}$$
This establishes that the mixed operator cannot increase the bootstrap bias relative to the greedy operator.

(II) Ideal case consistency.

By Definition 1, the model-based one-step score is
$$\widetilde{Q}(s', a') = r'_{\mathcal{M}} + \gamma \max_{a''} Q(s''_{\mathcal{M}}, a'').$$
Assumption 1 implies that for all (s', a') in the replay support,
$$\big| \widetilde{Q}(s', a') - \big(r' + \gamma \max_{a''} Q(s'', a'') \big) \big| \le \epsilon_r + \gamma \epsilon_s.$$
If
$$\lim_{k \to \infty} (\epsilon_r^{(k)} + \gamma \epsilon_s^{(k)}) = 0,$$
then
$$\sup_{s, a} \big| (\mathcal{T}_{\mathrm{mix}}^{(k)} Q)(s, a) - (\mathcal{T}_{\mathrm{std}} Q)(s, a) \big| \;\longrightarrow\; 0.$$
In particular,
$$\lim_{k \to \infty} \| \mathcal{T}_{\mathrm{mix}}^{(k)} Q^* - \mathcal{T}^* Q^* \|_\infty = 0.$$
Since Q^* is the unique fixed point of the optimal Bellman operator $\mathcal{T}^*$, the limiting result indicates that the mixed target recovers the optimal Bellman target in the ideal zero-error case. □

B SCENARIOS OF CITYFLOW AND O-CLOUD

In Appendix B, we illustrate the scenarios of CityFlow and O-Cloud that are used in our evaluations in detail, including the system model, state representation, action space, reward function, and setup.

B.1. CITYFLOW SCENARIO

We consider a traffic management scenario within the CityFlow simulation environment[1], where a set of intersections, denoted as $\mathcal{C} = \{1, \cdots, c, \cdots, C\}$, are managed by a centralized traffic control system. Each intersection is equipped with traffic lights that regulate the flow of vehicles across various road networks. The goal is to manage traffic flow efficiently by adjusting the traffic light phases at each intersection based on real-time traffic conditions.

State Representation: The state space, denoted as $\mathcal{S}$, encapsulates crucial operational metrics:

- Traffic Light Phases: Each intersection $c \in \mathcal{C}$ has multiple phases representing different traffic light states (e.g., green, yellow, red) that control the flow of vehicles. The state captures the current phase at each intersection.
- Vehicle Count on Lanes: The number of vehicles present on each lane leading into an intersection, represented by a vector V_{lane}. This includes both the total number of vehicles and those waiting to pass through the intersection.
- Waiting Vehicle Count on Lanes: The number of vehicles waiting at each lane, represented by a vector W_{lane}. This reflects the congestion level at the intersection.

Action Space: The action space, $\mathcal{A}$, involves selecting an appropriate traffic light phase for each intersection $c \in \mathcal{C}$. The action taken at each step is to choose the phase that will be applied to control the traffic flow.

Reward Function: The reward function is designed to minimize traffic congestion and vehicle wait times at intersections. The reward is computed based on the difference in the number of waiting vehicles before and after a traffic light phase change:

$$r^{c}_{\text{congestion}} = -\sum_{l \in \mathcal{L}} \left(W^{\text{after}}_{lane} - W^{\text{before}}_{lane} \right), \tag{33}$$

where $\mathcal{L}$ represents the set of lanes at intersection c. This reward structure encourages actions that reduce the number of waiting vehicles, thus alleviating congestion.

The overall reward for the environment at any given time is the sum of the rewards across all intersections:

$$r_{\text{total}} = \sum_{c=1}^{C} r^{c}_{\text{congestion}}. \tag{34}$$

B.2. O-CLOUD SCENARIO

We consider the computational task management on O-Cloud clusters, where a set of servers $\mathcal{M} = \{1, \cdots, m, \cdots, M\}$ with limited computing resources handle computing requests from applications (herein and after: users). The requests from different users are with various attributes in terms of CPU and RAM demands as well as the required processing latency. Upon the arrival of a user request at the O-Cloud, we assign it to an appropriate server for execution. In instances where a server is operating at full load to process concurrent requests, incoming requests are queued for temporary storage.

This scenario leverages the Alibaba cluster-trace-v2018 dataset [2], an open-source collection of real production cluster workload traces. Spanning an 8-day period and encompassing data from 4000 machines, this dataset provides a detailed view of server characteristics, including CPU, memory, and communication bandwidth. Each task trace records arrival time, duration, and CPU resource demand, with tasks arranged in chronological order of arrival.

State Representation: The state space, denoted as $\mathcal{S}$, encapsulates vital operational metrics:

[1] https://github.com/cityflow-project/CityFlow/

[2] https://github.com/alibaba/clusterdata

- Demands of the incoming task (user request), including requirement of CPU (c_{req}) and RAM (r_{req}), as well as the estimated occupation time (t_{occ}).
- The CPU and RAM utilization rates for each server, represented as the vector $U_{\text{cpu}} = \{u^m_{\text{cpu}} | m \in \mathcal{M}\}$ and $U_{\text{ram}} = \{u^m_{\text{ram}} | m \in \mathcal{M}\}$, respectively. U_{cpu} and U_{ram} indicate the current resource load.
- The length of the pending queue in each server, represented as the vector $L_{\text{queue}} = \{l^m_{\text{queue}} | m \in \mathcal{M}\}$, reflecting the count of pending tasks when CPU and RAM are used to handle other tasks in server $m \in \mathcal{M}$.
- A dynamically calculated queue penalty vector $P_{\text{queue}} = \{p^m_{\text{queue}} | m \in \mathcal{M}\}$ from each server. P_{queue} quantifies the delay-induced penalty associated with tasks in the pending queue, where $p^m_{\text{queue}} = \sum_{l=1}^{l^m_{\text{queue}}} t^l_{\text{occ}}$.

The queue penalty, P_{queue}, for server m is calculated based on the resource demands and occupation time of tasks in each server's pending queue. Specifically,

$$P^m_{\text{queue}} = \sum_i (c^i_{\text{req}} + r^i_{\text{req}}) \cdot t^i_{\text{occ}}. \tag{35}$$

This penalty measure helps in understanding the resource demand and processing backlog of tasks queued for execution, aiding in making more informed server selection decisions.

Action Space: The action space, $\mathcal{A}$, involves selecting an appropriate server for each incoming task, which can be mathematically represented as choosing a server $a \in \{1, 2, ..., M\}$ for task allocation.

Reward Function: The design of the reward function seeks to concurrently minimize power consumption and reduce the latency for users. For a set of M servers, instantaneous power P_{power} is calculated based on the CPU utilization rates of the individual servers. This is given by [Liu et al., 2017] and we modify it as:

$$P_{\text{power}} = \sum_{m=1}^{M} \left(P_0 + (P_1 - P_0) * (2u^m_{\text{cpu}} - (u^m_{\text{cpu}})^{1.4})/P_1\right) \tag{36}$$

Suppose that the server m is selected to handle the incoming task, the delay penalty P_{latency} is defined as a normalized measure based on the queuing penalties across all servers:

$$P^{cur}_{\text{latency}} = \sum_{i=0}^{cur_{time}} \begin{cases} t^i_{latency}, & if\ task\ starts \\ cur - t^i_{arr}, & else \end{cases} \tag{37}$$

where cur denotes the current time, $t^i_{latency}$ is the latency of task i, and t^i_{arr} is the arrival time of task i. Then, we can compute the reward for latency at current time:

$$r_{latency} = P^{cur}_{\text{latency}} - P^{cur-1}_{\text{latency}} \tag{38}$$

The reward function r_x is then expressed as a weighted sum of the instantaneous power among all servers and the current latency penalty to a selected server, defined as:

$$r_x = -\left(w_1 \cdot P_{\text{power}} + w_2 \cdot r_{latency}\right), \tag{39}$$

where w_1 and w_2 are weighting coefficients.

C EXPERIMENTAL CONFIGURATIONS

All vector-based RL environments and baselines are implemented in DI-Engine[3]. All image-based RL environments and baselines are implemented in Jax-baseline [4]. All experiments are conducted on a machine equipped with an NVIDIA RTX 5090 GPU, an Intel Core i9-14900k processor, and 128 GB of DDR5 RAM.

Vector-based Tasks: Table 3 provides detailed configurations for both Q and $\mathcal{M}$ on vector-based tasks. The term *Update per Collect* refers to the number of training steps performed after every *Replay Frequency* steps. The term *Target Update Interval* indicates the frequency, in steps, at which the target Q-network is updated. Parameters under *Epsilon* correspond to the settings of the epsilon-greedy exploration strategy.

[3] https://github.com/opendilab/DI-engine

[4] https://github.com/tinker495/jax-baseline

Table 3: Training Configurations for Q and $\mathcal{M}$ on vector-based tasks.

Parameter	Acrobot-V1	LunarLander-V2	Cartpole-V0	BitFlip	O-Cloud	CityFlow
Configurations for Q						
Discount	0.99	0.99	0.97	0.99	0.8	0.99
Hidden Size	[256, 256]	[512, 64]	[128, 128, 64]	[128, 128, 64]	[64, 64]	[256, 256]
Batch Size	128	64	64	128	32	64
Learning Rate	1e-4	1e-3	1e-3	5e-4	5e-5	5e-5
Update per Collect	10	10	1	10	1	1
Target Update Interval	2400	640	8000	4800	2000	2000
Total Steps	960000	128000	160000	960000	500000	400000
Buffer Size	100000	100000	100000	4000	100000	100000
Replay Frequency	96	64	80	96	100	100
Epsilon Start	1	0.95	0.95	0.2	0.05	0.05
Epsilon End	0.05	0.1	0.1	0.2	0.05	0.05
Epsilon Decay	250000	50000	10000	100	10000	10000
Configurations for $\mathcal{M}$						
Hidden Size	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[64, 64]	[256, 256]
Batch Size	256	128	128	256	64	128
Learning Rate	4e-5	4e-5	4e-5	4e-4	5e-4	5e-4
Update per Collect	1	1	[1,5,10,20]	1	1	1
State Norm	1	1	1	1	50	15
Factor α	0.2	0.2	0.2	0.2	0.5	0.5

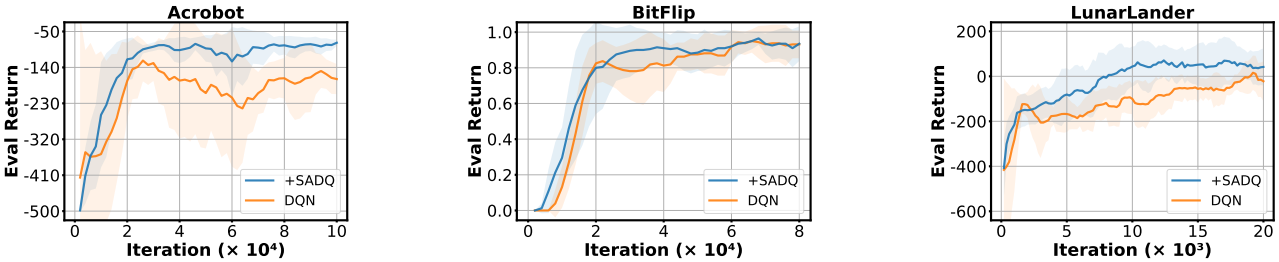
For the BitFlip environment, we set $n_{\text{bits}} = 8$. The cloud environment comprises 10 servers, resulting in an action space size of $|\mathcal{A}| = 10$. The reward function is parameterized by weights $w_1 = 0.1$ and $w_2 = 0.005$. The system begins with a warm-up phase of 1000 tasks, followed by 200 user requests. In the CityFlow simulation environment, there are 4 intersections, each with 4 control phases, yielding an action space of size $|\mathcal{A}| = 256$. The simulation operates over a fixed episode duration, where each step represents a predefined time period during which traffic light phases can be adjusted.

Image-based Tasks (Atari100K): For Atari100K experiments, we follow a standard value-based training setup. We use AdamW optimizer with a learning rate of 1×10^{-4}. The Q-network consists of a 2048-dimensional hidden layer. Training is performed for 100k environment steps with batch size 32 and training frequency 4. We use $\gamma = 0.99$, replay buffer size 10^5, and start learning after 1,600 steps. Exploration is annealed over 40% of training to a final ϵ of 0.001. The target network is updated every 1,000 steps, and two gradient updates are performed per environment interaction.

For the one-step RSSM-style dynamics model, the latent dynamics consists of a deterministic state of dimension 4096 and a categorical stochastic state with 32 variables and 32 classes per variable. The model uses a hidden size of 512, with AdamW using learning rate 3×10^{-4} and weight decay 10^{-4}. The training objective combines dynamics, representation, reconstruction, and reward terms with weights $\lambda_{\text{dyn}} = 1.0$, $\lambda_{\text{rep}} = 0.1$, $\lambda_{\text{recon}} = 2.0$, and $\lambda_{\text{rew}} = 1.0$, and uses free nats of 1.0. For the adaptive dynamics-model update schedule, we set the reconstruction-loss threshold to $\tau_{\mathcal{M}} = 0.001$ and the reduced relative update frequency to $t_f = 0.4$. For SADQ, we set the mixed-target coefficient to $\alpha = 0.2$ in all image-based experiments.

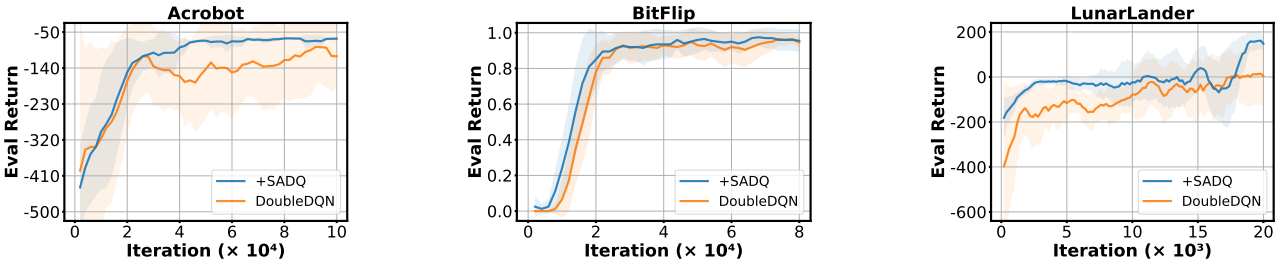
D CONTROLLED BASELINE COMPARISONS

To isolate the contribution of SADQ, we compare multiple baselines with and without SADQ under identical experimental settings. The only difference is the max-operator-related component introduced by SADQ. Figs 6–9 show the controlled comparisons between each baseline and its SADQ variant. SADQ consistently improves all baselines, suggesting that the proposed component is not tied to a specific DQN variant.



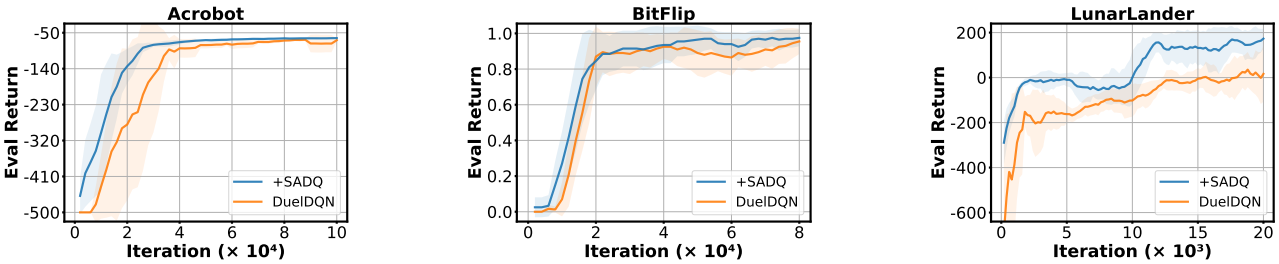
(a) Acrobot (b) BitFlip (c) LunarLander

Figure 6: Controlled comparison of DQN and DQN+SADQ across Acrobot, BitFlip, and LunarLander.



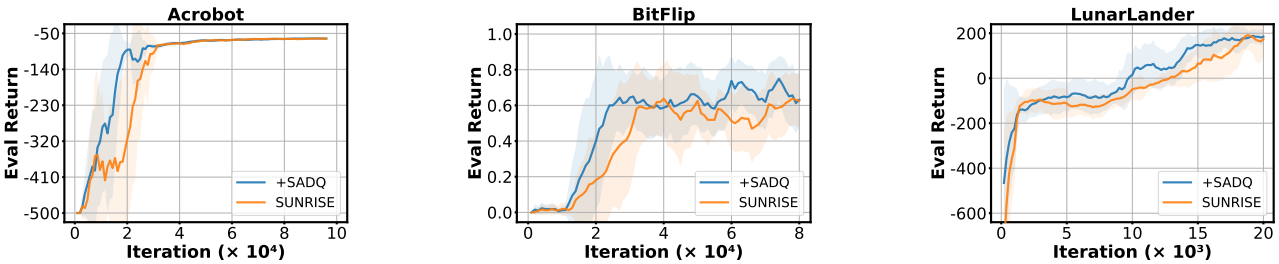
(a) Acrobot (b) BitFlip (c) LunarLander

Figure 7: Controlled comparison of Double DQN and Double DQN+SADQ across Acrobot, BitFlip, and LunarLander.



(a) Acrobot (b) BitFlip (c) LunarLander

Figure 8: Controlled comparison of Dueling DQN and Dueling DQN+SADQ across Acrobot, BitFlip, and LunarLander.



(a) Acrobot (b) BitFlip (c) LunarLander

Figure 9: Controlled comparison of SUNRISE and SUNRISE+SADQ across Acrobot, BitFlip, and LunarLander. SUNRISE uses an ensemble size of 5.

E PERFORMANCE COMPARISON OF SADQ WITH OTHER BASELINES

As discussed in the main text, SADQ is shown to significantly underperform compared to all baseline algorithms in the Cartpole environment in Fig. 10. While baselines like DQN and others achieve acceptable results, SADQ struggles to learn an effective policy and fails to match even the simplest algorithms. The simplicity of the Cartpole environment explains this outcome: with only two discrete actions and a state space of four dimensions, Cartpole poses minimal complexity for learning. Most baseline methods converge to a satisfactory policy within a few hundred iterations. Even advanced DQN variants, despite potentially introducing noise due to their complexity, are able to learn an acceptable policy given the short training time required by Cartpole.

On the other hand, SADQ's reliance on the stochastic model $\mathcal{M}$ for successor state predictions introduces a critical limitation. For $\mathcal{M}$ to make accurate predictions, it requires sufficient training time and data to stabilize. In environments with shorter learning horizons like Cartpole, $\mathcal{M}$ does not have enough iterations to converge effectively, leading to poor performance. This dependency on the stochastic model highlights a trade-off in SADQ's design: while it excels in tasks requiring detailed environment dynamics modeling, it may struggle in tasks where learning needs to occur rapidly. This observation sheds light on a key aspect of SADQ.

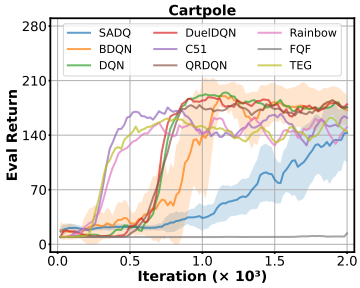


Figure 10: Performance comparison of SADQ with other baselines in Cartpole.