Adaptive Restart Distributions for Accelerating Reinforcement Learning

Adaptive Neustartverteilungen zur Beschleunigung des verstärkenden Lernens
Master thesis in the department of Computer Science by Michelle Saia
Date of submission: November 2, 2025

1. Review: Aryaman Reddi
2. Review: Nico Bohlinger
3. Review: Prof. Jan Peters, Ph.D.
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Michelle Saia, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 2. November 2025

M. Saia

Abstract

Reinforcement learning (RL) methods are commonly developed and evaluated in simulated environments, which makes it feasible to test algorithms before applying them to real-world systems. Despite steady progress, many RL algorithms still suffer from inefficient exploration: agents often revisit familiar states while struggling to reach task-relevant regions of the state space. This thesis addresses this challenge in difficult exploration problems by leveraging the flexibility of simulators to dynamically reshape the starting-state distribution. In the proposed two-agent framework, a primary agent learns to solve the task, while a secondary agent selects starting states from a set of previously visited states, guiding exploration toward regions that are difficult to reach from the default initial distribution. This interaction induces a curriculum over starting states, allowing the agent to encounter important or underexplored regions of the environment more efficiently during training. The results highlight when and why restart-based exploration is effective, and where it may fail, which helps clarify how restart-based exploration can be applied effectively in practice.

Zusammenfassung

Verstärkendes Lernen (Reinforcement Learning, RL) wird häufig in simulierten Umgebungen entwickelt und evaluiert, was es ermöglicht, Algorithmen vor ihrem Einsatz in realen Systemen zu testen. Trotz kontinuierlicher Fortschritte leiden viele RL-Algorithmen weiterhin unter ineffizienter Exploration: Die Agenten besuchen häufig bereits bekannte Zustände, während sie Schwierigkeiten haben, aufgaberelevante Bereiche des Zustandsraums zu erreichen. Diese Arbeit befasst sich mit dieser Herausforderung bei explorationsschwierigen Aufgaben, indem die Flexibilität von Simulatoren genutzt wird, um die Verteilung der Anfangszustände dynamisch anzupassen. Im vorgeschlagenen Zwei-Agenten-Rahmen lernt ein primärer Agent die eigentliche Aufgabe, während ein sekundärer Agent Anfangszustände aus einer Menge zuvor besuchter Zustände auswählt und so die Exploration gezielt auf schwer erreichbare Regionen lenkt. Diese Interaktion erzeugt ein Curriculum über Anfangszustände, wodurch der Agent wichtige oder bisher wenig erkundete Bereiche der Umgebung während des Trainings effizienter erreicht. Die Ergebnisse zeigen, wann und warum neustartbasierte Exploration wirksam ist und wo ihre Grenzen liegen, was verdeutlicht, wie solche Ansätze in der Praxis effektiv eingesetzt werden können.

Contents

1. Introduction

Reinforcement learning (RL) provides a general framework for sequential decision-making, in which an agent learns to act through interaction with an environment [1]. Over the past decade, RL has enabled major advances in control, robotics, and game-playing, demonstrating the potential of learning-based approaches to achieve complex behavior without explicit supervision. However, many RL methods still suffer from inefficient exploration, especially in high-dimensional or long-horizon environments [1, 2, 3]. Agents often spend large portions of training revisiting familiar states, while rarely reaching regions of the state space that are critical for solving the task [2, 4, 5].

This exploration inefficiency remains one of the main obstacles to applying RL reliably beyond standard benchmark tasks, which are often designed to simplify learning through dense rewards and structured dynamics [4, 6]. While intrinsic motivation or curiosity-based methods attempt to encourage novel behavior, these signals are often noisy and difficult to tune [2, 7]. In simulated environments, a more direct opportunity exists: the ability to reset the agent to arbitrary states during training. Unlike in the physical world, simulation allows full control over initial conditions, making it possible to guide exploration strategically by choosing where episodes begin.

This thesis builds on that idea and investigates how adaptive control of the starting-state distribution can accelerate reinforcement learning. Instead of always resetting the agent to the same initial position, the environment can occasionally restart from previously visited states that are informative or underexplored. By reshaping the distribution of starting states during training, the agent can focus its learning on regions that are difficult to reach from the default start, thereby improving both exploration and convergence.

To study this idea, the thesis introduces a two-agent framework in which a primary agent learns the task while a secondary agent selects restart states from experience. The secondary agent maintains a buffer of previously encountered states and adaptively samples from it to guide the next episode's start. This interaction implicitly creates a curriculum over starting states: early on, the agent explores nearby regions, while later it revisits and expands upon more complex or rewarding areas.

The central goal of this work is to analyze how different restart strategies influence exploration, stability, and learning efficiency in continuous-control environments. The thesis compares several mechanisms for selecting and maintaining restart states, investigates their interactions with intrinsic motivation, and identifies the conditions under which restart-based exploration provides the greatest benefit.

2. Foundations

This chapter provides the theoretical background for the thesis. We begin with a general overview of reinforcement learning (RL) as a framework for sequential decision making, then discuss policy gradient methods with a focus on Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC), and finally outline common approaches to exploration, including restart distributions, which motivate the method developed in this work.

2.1. Reinforcement Learning Basics

Reinforcement learning (RL) formalizes the interaction between an agent and an environment as a *Markov Decision Process* (MDP) [1, 8]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s, a)$ is the transition distribution, $r(s, a)$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor.

The agent follows a policy $\pi(a|s)$, which specifies the probability of selecting action a in state s. The objective is to maximize the expected discounted return:

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right].$$

Two central concepts are the *state-value function* $V^{\pi}(s)$ and the *action-value function* $Q^{\pi}(s, a)$, which represent expected returns from states or state–action pairs under policy π. Learning these value functions, directly or implicitly, underpins most RL algorithms.

A fundamental challenge in RL is the *exploration–exploitation dilemma* [1]. An agent must exploit known good actions to maximize reward, but also explore new actions to discover potentially better outcomes. Insufficient exploration can trap agents in suboptimal regions of the state space, a problem particularly acute in hard-exploration tasks.

2.2. Policy Gradient Methods

Classical value-based methods such as Q-learning do not scale well to continuous action spaces. Policy gradient methods address this limitation by directly optimizing a parameterized policy $\pi_\theta(a|s)$ with respect to the expected return [9]. The policy gradient theorem shows:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right],$$

where d^π is the discounted state distribution.

2.2.1. Actor–Critic Architectures

In practice, high variance in gradient estimates makes policy gradient methods unstable. Actor–critic methods reduce variance by learning an approximate value function (the critic) alongside the policy (the actor). This family of algorithms includes methods such as Advantage Actor-Critic (A2C) and Trust Region Policy Optimization (TRPO) [10, 11].

2.2.2. Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm [12] extends the deterministic policy gradient framework to deep neural networks and continuous action spaces. DDPG is an off-policy actor–critic method that combines ideas from Q-learning and policy gradients: the critic approximates the action-value function $Q_\phi(s,a)$, while the actor represents a deterministic policy $a = \mu_\theta(s)$ trained to maximize $Q_\phi(s,a)$.

The critic is updated using the Bellman target:

$$y_t = r_t + \gamma Q_{\phi^-}\big(s_{t+1}, \mu_{\theta^-}(s_{t+1})\big),$$

where ϕ^- and θ^- are slowly updated target network parameters used to stabilize learning. The actor parameters are optimized to maximize the critic's output:

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_a Q_\phi(s,a)\nabla_\theta \mu_\theta(s)],$$

where $\mathcal{D}$ is a replay buffer of past transitions.

DDPG introduced several key innovations for deep RL: the use of a replay buffer to enable off-policy learning, target networks for stability, and deterministic continuous-action policies. However, it is sensitive to hyperparameters and noise processes, and often suffers from overestimation bias and poor exploration in complex environments. These weaknesses later inspired more stable and expressive algorithms such as Soft Actor-Critic (SAC).

2.3. Soft Actor-Critic (SAC)

The *Soft Actor-Critic* (SAC) algorithm [13] is an off-policy actor–critic method that extends DDPG by incorporating entropy maximization to encourage more stable and efficient exploration. While DDPG learns a deterministic policy and is prone to instability, SAC optimizes a stochastic policy under a *maximum-entropy* objective that balances reward and uncertainty:

$$J(\pi) = \mathbb{E}_{\pi} \left[\sum_t \gamma^t \big(r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \big) \right],$$

where $\mathcal{H}(\pi(\cdot|s))$ denotes the entropy of the policy at state s, and α controls the trade-off between exploration and exploitation.

SAC alternates between updating the critic (Q-function) and the actor (policy). The critic learns to estimate expected returns by minimizing the temporal-difference residual, where the Q-update incorporates both reward and entropy:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Big[R(s,a) + \gamma \mathbb{E}_{a' \sim \pi} \big(Q(s', a') - \alpha \log \pi(a'|s') \big) - Q(s,a) \Big].$$

This *soft* Bellman update increases the value of actions that lead not only to higher rewards but also to higher policy entropy, promoting exploration directly through the value function.

The actor is updated to favor actions with higher Q-values while maintaining stochasticity in its output. In practice, the policy outputs a Gaussian distribution whose mean and variance are learned, and actions are squashed through a $\tanh$ function to remain within valid bounds. The temperature parameter α can be fixed or adjusted automatically to maintain a target entropy level, keeping the policy more random early in training and more deterministic as learning converges.

2.4. Exploration in Reinforcement Learning

Exploration remains a central challenge in RL. Simple strategies include ϵ-greedy exploration in discrete domains or adding Gaussian noise to actions in continuous domains. However, such local exploration strategies are ineffective in environments where important states are unlikely to be reached by chance.

2.4.1. Count-Based Exploration

Count-based methods reward agents for visiting novel states [14]. In tabular MDPs, the bonus can be inversely proportional to the square root of the visit count:

$$r'(s,a) = r(s,a) + \frac{\beta}{\sqrt{N(s,a)}},$$

where $N(s,a)$ is the number of visits to state–action pair (s,a). While effective in small domains, this does not scale to continuous or high-dimensional state spaces.

2.4.2. Pseudo-Counts and Intrinsic Motivation

To extend count-based ideas, pseudo-count methods use density models to approximate state novelty [15]. Other intrinsic motivation techniques reward agents for prediction error, surprise, or curiosity [2, 16]. Random Network Distillation (RND) is a particularly effective recent method, where prediction error on random features provides a novelty signal [16].

2.4.3. Random Network Distillation

Random Network Distillation (RND) is an intrinsic motivation method proposed by Burda et al. [16]. It addresses the exploration problem by rewarding agents for visiting novel states, measured through prediction error. RND employs two neural networks: a fixed, randomly initialized target network f, and a predictor network $\hat{f}_\theta$ trained to approximate f. Given a state s, the intrinsic reward is the squared prediction error:

$$r_{\text{int}}(s) = \|f(s) - \hat{f}_\theta(s)\|^2.$$

States that have not been frequently visited produce large prediction errors, yielding higher intrinsic rewards and encouraging the agent to explore new regions. As the predictor improves, the intrinsic reward for familiar states diminishes.

RND was originally validated on sparse-reward Atari benchmarks such as Montezuma's Revenge, demonstrating its effectiveness in guiding exploration where extrinsic signals are rare.

2.4.4. Curriculum Learning

Curriculum learning strategies shape the training process by gradually increasing task difficulty [17]. In RL, this often takes the form of adjusting the initial state distribution or goal distribution, allowing the agent to solve simpler problems before tackling harder ones. Reverse Curriculum Generation [18] is a notable example, starting agents near the goal and progressively moving initial states outward.

2.5. State Resets in Off-Policy Algorithms

In off-policy reinforcement learning, experience is collected from trajectories that may differ from the current policy. This property allows learning from transitions generated under arbitrary initial conditions, making state resets a natural and theoretically consistent modification. Changing the initial-state distribution $p_0(s)$ alters the regions of the state space explored during training but does not invalidate the Bellman-based update rules used by algorithms such as DDPG and SAC. This section outlines how resets influence these methods both mathematically and empirically.

2.5.1. Resets in DDPG

In Deep Deterministic Policy Gradient (DDPG) [12], the agent learns a deterministic policy $a = \mu_\theta(s)$ and a critic $Q_\phi(s, a)$ trained using transitions stored in a replay buffer $\mathcal{D}$. The critic update minimizes the Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[(Q_\phi(s,a) - (r + \gamma Q_{\phi^-}(s', \mu_{\theta^-}(s'))))^2\Big],$$

and the actor is updated by maximizing the critic's output:

$$J_\pi(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}[Q_\phi(s, \mu_\theta(s))].$$

Since both updates depend on expectations over the replay buffer, the data distribution in $\mathcal{D}$ determines which parts of the state space most influence learning. When the environment is reset to alternative starting states $s_0 \sim \tilde{p}_0(s)$, this effectively replaces the induced state-visitation distribution $d^\mu_{p_0}$ with $d^\mu_{\tilde{p}_0}$:

$$d^\mu_{\tilde{p}_0}(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s \mid s_0 \sim \tilde{p}_0, \mu, P).$$

The update equations remain valid because DDPG is off-policy: the critic and actor are trained on replayed transitions rather than full trajectories sampled from the current policy. Resets therefore modify only the empirical weighting of states in the updates, not the Bellman operator itself.

In practice, resetting to diverse or strategically chosen states increases the variety of transitions stored in $\mathcal{D}$, helping the critic generalize across a broader portion of the state space. However, neural Q-functions learn by interpolating between observed samples and thus rely on local continuity in the input space: states that are numerically close are expected to have similar values. When resets place the agent in distant or disconnected regions of the state space, this assumption breaks down—the critic must extrapolate across large gaps, often resulting in unstable or inaccurate value estimates [19, 20].

2.5.2. Resets in SAC

Soft Actor-Critic (SAC) [13] extends DDPG by optimizing a stochastic policy under a maximum-entropy objective. The critic update minimizes the soft Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[(Q_\phi(s,a) - (r + \gamma\,\mathbb{E}_{a'\sim\pi_\theta}[Q_{\bar{\phi}}(s',a') - \alpha\log\pi_\theta(a'|s')]))^2\Big],$$

and the policy is updated to minimize:

$$J_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\Big[\mathbb{E}_{a\sim\pi_\theta}[\alpha\log\pi_\theta(a|s) - Q_\phi(s,a)]\Big].$$

Both losses are expectations over transitions sampled from the replay buffer. Changing the initial-state distribution through resets thus modifies the empirical distribution of states $s \sim \mathcal{D}$, indirectly influencing the gradients in both the critic and actor updates.

Similar to DDPG, the SAC critic depends on the state distribution present in the replay buffer. Resets that introduce states far from the agent's typical trajectories can induce extrapolation error in the critic, as the Q-function must estimate values in poorly represented regions of the state space. However, SAC is more robust to such distributional shifts due to its stochastic policy, entropy regularization, and use of double critics, which together mitigate overestimation bias and stabilize value learning [13, 19].

In practice, resets that sample reachable yet diverse states can accelerate learning by broadening state coverage and improving value propagation, while unrealistic or discontinuous resets may still lead to biased or unstable critic updates.

3. Related Work

Exploration in reinforcement learning (RL) is especially challenging when task-relevant states are unlikely to be visited under the environment's default start-state distribution. A growing line of research addresses this issue by *modifying the restart or start-state distribution* during training—via automated curricula, replay-driven resets, or demonstration-based initializations. This chapter reviews the most relevant approaches in this area.

3.1. Reverse Curriculum Generation

Florensa et al. [18] introduced *Reverse Curriculum Generation* (RCG), a method that automatically adapts the start-state distribution to accelerate learning in goal-conditioned tasks with sparse rewards. The central idea is to begin learning from states close to the goal and then gradually expand outward. Feasible "nearby" states are generated by short Brownian-motion rollouts from previously solvable (or partially solvable) states, producing a curriculum of "good starts"—states that the agent sometimes solves but not always.

RCG assumes access to (i) arbitrary resets, (ii) at least one goal state, and (iii) a communicating class containing the goal and all relevant starts. While these assumptions heuristically suggest that the agent can eventually cover the original start distribution, the method *does not provide a formal convergence guarantee*. Instead, the curriculum expands outward in a fixed number of training iterations, relying on empirical performance improvements to indicate progress. The paper focuses primarily on sparse-reward robotic manipulation and navigation tasks.

3.2. Go-Explore

Ecoffet et al. [3] proposed *Go-Explore*, an approach that separates exploration from policy learning by (1) maintaining an archive of promising states (or cells), (2) *returning* deterministically to one of these states, and then (3) *exploring* stochastically from it. Once a successful trajectory to the goal is discovered, a robust policy is obtained through imitation learning ("robustification"). This strategy achieved state-of-the-art results on challenging Atari exploration benchmarks such as Montezuma's Revenge and Pitfall by systematically revisiting previously explored regions of the environment.

Despite its success, Go-Explore has notable limitations. Its exploration phase relies on domain-specific cell representations to define the archive space, and these cell definitions frequently depend on *hand-crafted features* tailored to each environment (e.g., game-specific coordinates, room identifiers), which limits generality. Furthermore, the final policy is obtained through an imitation-learning stage rather than through end-to-end reinforcement learning. Extending the approach to high-dimensional continuous-control settings is also nontrivial: even when starting from promising states, purely random exploration rarely makes progress because competent locomotion and control are themselves hard prerequisites, making it unlikely to "stumble upon" successful behavior.

3.3. Demonstration-Based Restarts

Another line of work leverages demonstrations to modify the start-state distribution. Popov et al. [21] studied dexterous manipulation tasks such as grasping and stacking, using instructive start states sampled from successful demonstrations or manually chosen configurations. By initializing episodes closer to the goal (e.g., with an object already grasped), they reduced the difficulty of exploration and enabled efficient training with sparse rewards.

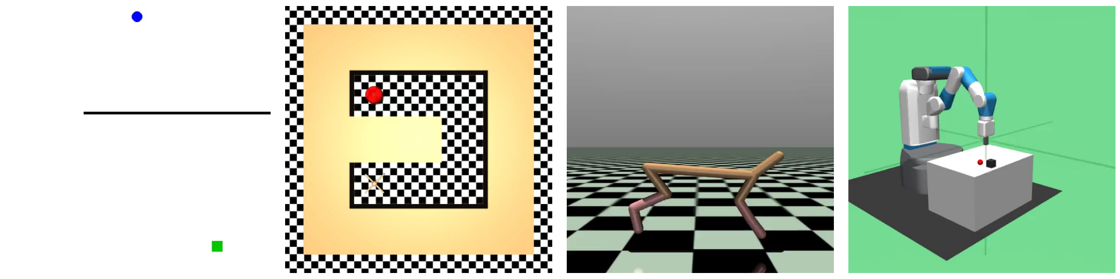
Salimans and Chen [22] applied a similar idea to hard-exploration Atari environments, particularly Montezuma's Revenge. Their method resets training episodes to states sampled along a single expert trajectory, starting near the end of the demonstration and gradually moving the reset point backward as the agent improves. This effectively constructs a reverse curriculum from one demonstration while using standard RL optimization rather than imitation learning. The approach achieved human-level performance from a single demonstration.

3.4. Exploring Restart Distributions

Tavakoli et al. [23] formalized the idea of adapting the start-state distribution through what they call *exploring restart distributions*. Their method maintains a memory of previously visited states and resets the agent to these states during training to improve state-space coverage and sample efficiency. Inspired by experience replay, they proposed three variants: *uniform*, *prioritized*, and *episodic* restarts. When combined with on-policy PPO, these strategies improved performance on both dense- and sparse-reward continuous-control environments, highlighting that adaptive restart mechanisms can benefit even standard policy-gradient methods without requiring demonstrations.

4. Environments

We evaluate our approach on four continuous-control environments: a custom **PointMass** toy environment created by us for controlled investigations, and three standard benchmarks from the *Gymnasium* (and Robotics) suites: **AntMaze**, **HalfCheetah**, and **Fetch Pick-and-Place**. Both **PointMass** and **AntMaze** are *goal-based* (fixed start/goal), **Fetch Pick-and-Place** is *multi-goal* (goal-conditioned with varying targets), and **HalfCheetah** has no explicit goal signal (forward locomotion). This diversity lets us systematically explore the *restart distribution*—comparing how alternative reset schemes influence exploration behavior, sample efficiency, and policy stability across goal-based (PointMass, AntMaze), multi-goal (Fetch Pick-and-Place), and non-goal (HalfCheetah) settings. A composite figure with real renderings is shown in Fig. 4.1a–d, which we reference throughout this section.



(a) PointMass (b) AntMaze (c) HalfCheetah (d) Fetch: Pick and Place

Figure 4.1.: Environments used in this study: (a) custom PointMass toy environment; (b) AntMaze; (c) HalfCheetah; (d) Fetch: Pick and Place. Panels (b–d) are standard Gymnasium / Robotics tasks.

4.0.1. PointMass (custom toy environment)

The **PointMass** environment *(Fig. 4.1a)* is a lightweight 2D *kinematic* setup (no contacts or complex dynamics), making it simple and interpretable and therefore well-suited for initial experiments on restart distributions. The agent starts at $(250, 20)$ and must reach a goal at $(400, 450)$. A horizontal wall blocks the direct path, so the agent must go around it. The x–y layout with start/goal and wall is shown in Fig. 4.2. Episodes last 500 steps.

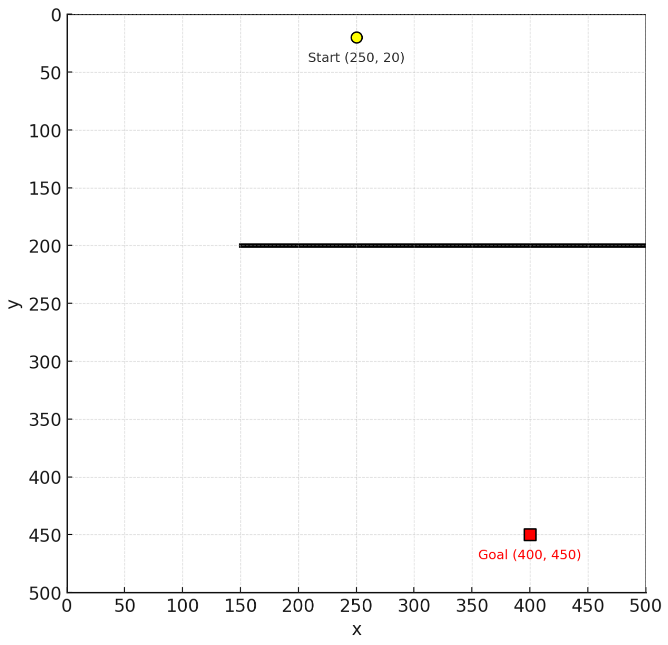


Figure 4.2.: PointMass layout: start (yellow), goal (red), and blocking wall at $y{=}200$.

Observations. We use a 4-dimensional *body state* vector $o_t = [\, x_t,\, v_{x,t},\, y_t,\, v_{y,t} \,]$.

Actions. The action is a 2D continuous command $a_t = (\Delta v_{x,t},\, \Delta v_{y,t}) \in [-1, 1]^2$ that adjusts the velocity along x and y. The environment applies a simple kinematic update with a small friction term proportional to velocity and clamps positions to the board; detailed integration is not essential for our discussion.

Reward. We use a distance-based shaping reward defined as the negative Euclidean distance to the goal:

$$r_t = -d_t, \quad \text{where} \quad d_t = \|(x_g - x_t,\ y_g - y_t)\|_2.$$

Episode. The episode terminates when the step limit is reached or the agent reached the goal.

4.0.2. AntMaze (MuJoCo, Gymnasium Robotics)

AntMaze is a hard exploration problem even with dense rewards due to deceptive gradients and the maze geometry. Locomotion with the Ant is also nontrivial: repeated foot–ground impacts and friction make the dynamics non-smooth and sensitive to small action changes. The task *(Fig. 4.1b)* features a quadruped ant navigating a maze to a fixed goal. We customize the environment so that each episode starts at $(-4, -4)$ and the goal is fixed at $(-4, 4)$; see the x–y layout in Fig. 4.3.

Observations. We use a 27-dimensional *body state* vector o_t (joint angles, joint velocities, and torso orientation/velocity). To make the goal information explicit, we *prepend* the relative displacement to the goal,

$$(\Delta x_t, \Delta y_t) = (x_g - x_t,\ y_g - y_t),$$

so the final input to the policy and value functions is

$$s_t = [\,\Delta x_t,\ \Delta y_t,\ o_t\,] \in \mathbb{R}^{29}.$$

Actions. The action is an 8-dimensional continuous vector $a_t \in [-1, 1]^8$ applied as joint torques to the ant's actuators.

Reward. We use exponential distance shaping,

$$r_t \;=\; e^{-d_t}, \qquad d_t \;=\; \big\|(x_g - x_t,\ y_g - y_t)\big\|_2.$$

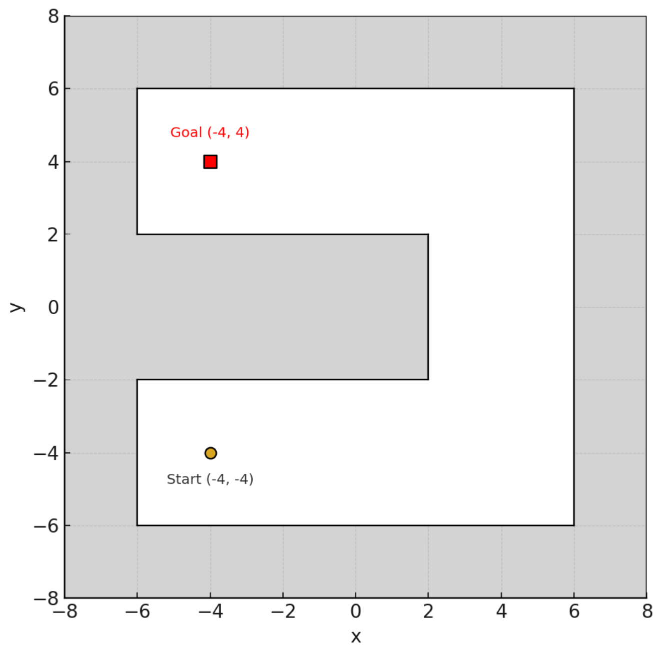


Figure 4.3.: AntMaze x–y layout. Gray regions denote walls. The episode starts at $(-4,-4)$ and the fixed goal is at $(-4,4)$.

Episode. The episode terminates when the step limit is reached; we set the horizon to 400 steps.

4.0.3. HalfCheetah (MuJoCo, Gymnasium)

The **HalfCheetah** environment *(Fig. 4.1c)* is a planar articulated robot optimized for fast forward locomotion.

Observations. A 17-D vector composed of $qpos$ (8 elements) and $qvel$ (9 elements). By default the absolute x position is excluded from the observation.

Actions. A 6-D continuous torque command $a \in [-1,1]^6$ applied at the hinge joints (back/front thigh, shin, and foot).

Reward. $r = \text{forward\_reward} - \text{ctrl\_cost}$ (Gymnasium defaults).

Episode. No terminal condition; episodes truncate at 1000 steps by default.

4.0.4. Fetch: Pick and Place (MuJoCo, Gymnasium Robotics)

The **Fetch Pick-and-Place** task *(Fig. 4.1d)* uses a 7-DoF arm with a parallel gripper to move a cube to a target over the table or in mid-air.

Observations. Goal-conditioned dictionary with `observation` $\in \mathbb{R}^{25}$ (robot & object state), `achieved_goal` $\in \mathbb{R}^3$, and `desired_goal` $\in \mathbb{R}^3$. For the policy input, we prepend two relative-position vectors to the default observation: gripper-to-block $\Delta_t^{gb} = b_t - g_t \in \mathbb{R}^3$ and block-to-goal $\Delta_t^{bg} = g_t^* - b_t \in \mathbb{R}^3$, giving

$$s_t = \left[\Delta_t^{gb},\ \Delta_t^{bg},\ o_t\right] \in \mathbb{R}^{31}.$$

Actions. A 4-D continuous command controlling end-effector Cartesian motion $(\Delta x, \Delta y, \Delta z)$ and the gripper open/close channel; $a_t \in [-1, 1]^4$.

Reward. We use the *dense* option only: the reward is the negative Euclidean distance between the achieved and desired goals,

$$r_t = -\left\|\texttt{achieved\_goal}_t - \texttt{desired\_goal}_t\right\|_2.$$

Episode. The episode terminates when the step limit is reached (default 50 steps).

5. Methods

This chapter formalizes our setting and describes our restart mechanism: what we store, how we maintain a fixed-size *reset buffer*, and how we sample start states from it. We use **SAC** as the training backbone for all core experiments; a preliminary **DDPG** baseline is reported only for the initial *PointMass* study (see Foundations and Chapter 6).

5.1. Problem Setup

We model each task as an MDP $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ with default start distribution p_0. During training, an episode starts either from p_0 or from a *restart distribution* q_ψ over previously visited states:

$$s_0 \sim \begin{cases} p_0, & \text{with prob. } 1 - p_{\text{reset}}, \\ q_\psi, & \text{with prob. } p_{\text{reset}}, \end{cases} \qquad p_{\text{reset}} = 0.5.$$

We keep a dedicated reset buffer $\mathcal{B}$ of at most $N{=}1024$ items. Each item stores a state representation and a reward tag (defined below). The standard transition replay buffer for policy updates is unchanged.

5.2. Backbone

SAC (primary). All core experiments use Soft Actor–Critic (SAC); we follow the default configuration of our reference implementation. The only global change is the *training budget* (environment timesteps), specified per environment in Chapter 6.

Intrinsic motivation (AntMaze only). For *AntMaze* we add Random Network Distillation (RND) to encourage novelty:

$$r_t^{\text{tot}} = r_t^{\text{ext}} + \eta \left\| f(s_t) - \hat{f}_\theta(s_t) \right\|_2^2,$$

with default RND architecture/optimizer. We report the mixing coefficient η and any normalization alongside AntMaze results. RND is not used in PointMass, HalfCheetah, or Fetch.

Preliminary DDPG baseline (PointMass only). We include a DDPG baseline *only* in the initial PointMass experiments to motivate restarts; algorithmic details appear in Foundations and results in Chapter 6. DDPG is not part of the unified method below.

5.3. Restart Distributions

5.3.1. What we store (per-step vs. end-of-episode)

Entries are added to $\mathcal{B}$ in one of two logging modes:

- **Per-step:** at every env step, store the current observation and its immediate reward.
- **End-of-episode:** on termination, store selected observations (e.g., terminal/key states) tagged with an episode-level reward statistic (e.g., undiscounted return).

Each stored item keeps a representation z and a reward tag $\bar{r}$:

$$z = \phi(s), \qquad \phi(s) = \begin{cases} \text{raw observation } s & (\text{raw mode}), \\ \text{actor embedding } \phi_\theta(s) & (\text{embedded mode}). \end{cases}$$

5.3.2. How we keep the buffer full but diverse (capacity $N{=}1024$)

When $|\mathcal{B}| < N$ we insert the new item $(z, \bar{r})$. Once full, we use:

Random replace: With probability p_{replace}, we uniformly select an entry $b_r \in \mathcal{B}$ and replace it with $(z, \bar{r})$. Otherwise, the new item is discarded.

L2-diversify (diverse goal sampling, [24]): With probability p_{replace}, we consider $(z, \bar{r})$ for addition. A random resident b_r is sampled uniformly from $\mathcal{B}$, and we compare their average L_2 distances to the rest of the buffer:

$$d_{\text{new}} = \frac{1}{|\mathcal{B}| - 1} \sum_{b' \in \mathcal{B} \setminus \{b_r\}} \|z - b'.z\|_2,$$

$$d_{\text{old}} = \frac{1}{|\mathcal{B}| - 1} \sum_{b' \in \mathcal{B} \setminus \{b_r\}} \|b_r.z - b'.z\|_2.$$

If $d_{\text{new}} > d_{\text{old}}$, we replace b_r with $(z, \bar{r})$. Otherwise, we still replace it with probability $p_{\text{add-non-diverse}}$.

We use $p_{\text{replace}} = 10^{-3}$ and $p_{\text{add-non-diverse}} = 10^{-3}$ following [24].

5.3.3. How we sample starts from the buffer

When a reset is triggered (p_{reset}=0.5), we draw the new start state s_0 from the buffer $\mathcal{B}$ using one of the following strategies:

- **Uniform:** choose a buffer entry uniformly at random.
- **Reward-softmax:** sample proportionally to the (normalized) reward tag $\bar{r}$ associated with each stored state:

  $$p(b) \propto \exp\big(\beta\, \tilde{r}(b)\big),$$

  where $\tilde{r}(b)$ is a normalized version of $\bar{r}(b)$ (zero mean and unit variance within the current buffer), and β>0 is a temperature parameter controlling the sampling sharpness.

Each buffer entry stores a state and an associated reward tag $(s, \bar{r})$. The tag $\bar{r}$ represents either the instantaneous reward (for per-step mode) or an episode-level statistic such as total return or final distance (for end-of-episode mode); we specify the choice for each experiment in Chapter 6.

5.3.4. Manual (fixed) reset sets

We also evaluate small, hand-picked reset sets as a transparent baseline, sampled uniformly when used. Exact coordinates are reported with the corresponding figures in Chapter 6.

5.4. Unified Training Loop (SAC)

The following procedure is used in all core experiments (SAC backbone):

1. Initialize replay buffer $\mathcal{D}$ and reset buffer $\mathcal{B}$.
2. For each episode:
    a) With probability $p_{\text{reset}}=0.5$, sample s_0 from $\mathcal{B}$ using the chosen sampler; otherwise $s_0 \sim p_0$.
    b) For $t = 0, \ldots, H-1$:
        i. Sample $a_t \sim \pi_\theta(\cdot|s_t)$ (SAC).
        ii. Step env to get (r_t, s_{t+1}); push (s_t, a_t, r_t, s_{t+1}) to $\mathcal{D}$.
        iii. Update $\mathcal{B}$ according to the selected logging mode and maintenance policy.
        iv. Perform U SAC updates from $\mathcal{D}$ (defaults), including temperature tuning if enabled.

5.5. Implementation Notes

Defaults. We keep default hyperparameters of SAC (networks, optimizer, target updates, etc.). The only global change is the number of environment timesteps, specified per environment in Chapter 6.

Representations. We evaluate z as raw observations or actor embeddings $\phi_\theta(s)$; diversity and distances use whichever representation is chosen.

AntMaze RND. RND is enabled only on AntMaze with default module settings; we report η and any normalization in the AntMaze results.

5.6. Compared Variants (Naming)

Per environment we report combinations of:

- logging mode: per-step vs. end-of-episode,
- representation: raw vs. actor-embedded,
- buffer maintenance: FIFO vs. L2-diversify,
- sampler: uniform vs. reward-softmax,
- backbone: SAC (core), plus a preliminary DDPG baseline on PointMass only.

Unless stated otherwise, we use $p_{\text{reset}}{=}0.5$, buffer capacity $N{=}1024$, and sampler temperature $\beta{=}1$.

6. Experiments and Results

This chapter evaluates our restart strategies across four environments. We begin with a preliminary *PointMass* study using DDPG to motivate restarts, then switch to SAC for all core results on *AntMaze*, with transfer studies on *HalfCheetah* and *Fetch Pick-and-Place*.

6.1. Experimental Setup

Backbone and budgets. Unless stated otherwise, we use **SAC** with default hyperparameters (as in the reference implementation); the only global change is the *training budget* (environment timesteps), which we specify per environment below. We include a **DDPG** baseline *only* for the initial PointMass study to illustrate the effect of fixed resets.

Restart mechanism (shared). Reset probability $p_{\text{reset}}{=}0.5$. The reset buffer has fixed capacity $N{=}1024$ and stores either *raw* observations or *actor embeddings*. Entries are logged either *per step* (with the instantaneous reward) or *at episode end* (tagged by the episode return, i.e., the total reward accumulated over the episode). When full, the buffer is maintained either by **Random-Replace** or **L2-diversify** replacement (Sec. 5). Start states are sampled **uniformly** or by **reward-softmax** (temperature β; we report the value where used).

Intrinsic motivation (AntMaze only). For *AntMaze* we also report runs with RND (mixing coefficient η specified with the result).

6.1.1. Evaluation Protocol and Metric

Our primary metric is the **success rate during training**. Instead of running separate evaluation phases, we measure success continuously based on the `is_success` signal provided by each environment (when available). The success rate at time t is computed as:

$$\text{SuccessRate}(t) = \frac{\#\text{ successful episodes up to step } t}{\#\text{ episodes up to step } t}.$$

We plot success rate curves over training and report the mean $\pm$ standard error across random seeds.

Success criteria and horizons (see environment descriptions in Sections 4.0.1, 4.0.2, 4.0.4, and 4.0.3):

- **PointMass:** success is reaching the goal region before the horizon of 500 steps.
- **AntMaze:** success is reaching the goal location before the horizon of 400 steps.
- **Fetch Pick-and-Place:** success is reaching the goal position before the horizon of 50 steps.
- **HalfCheetah:** since no binary success signal is defined, we instead measure the episode return (cumulative reward) during training.

6.1.2. Compared Variants

We evaluate the following combinations:

- **Baselines**: SAC; SAC+FixedReset; (PointMass only) DDPG vs. DDPG+FixedReset.
- **Buffer logging**: {per-step, end-of-episode}.
- **Representation**: {raw, actor-embedded}.
- **Maintenance**: {Random-Replace, L2-diversify}.
- **Sampler**: {uniform, reward-softmax}.
- **RND (AntMaze)**: {off, on}.

External baselines. We compare to the *Exploring Restart Distributions* variants of Tavakoli et al. — **Uniform Restart** and **Prioritised Restart** — which adapt the initial-state distribution by restarting from previously encountered states. We use our own re-implementations that follow their definitions and settings (including the 10% ratio of augmented initial states and the shorter time limit for augmented starts), but use **SAC** instead of PPO as the backbone to match our off-policy training pipeline. Concretely, *Uniform Restart* samples initial states uniformly from a restart memory, whereas *Prioritised Restart* samples them by a softmax distribution over value-based TD errors. For implementation details, see Appendix A.1 and [23].

6.2. PointMass: Do Fixed Resets Help? (DDPG, preliminary)

6.2.1. Goal

Test whether adding manual fixed resets improves learning in a simple, controlled setting.

6.2.2. Setup

Training budget: $600{,}000$ environment steps.

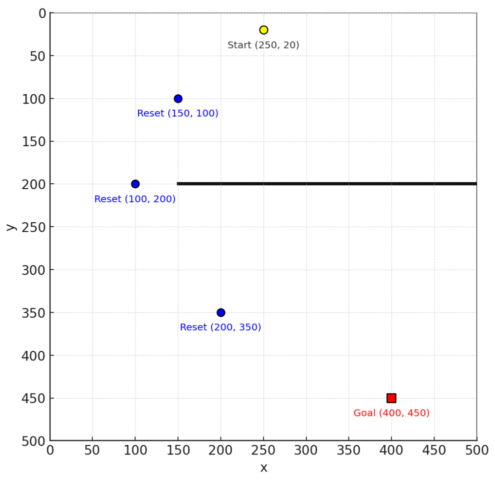
Task: Start $(250, 20)$, goal $(400, 450)$, horizon 500 steps (see Fig. 4.1a).

6.2.3. Methods Compared

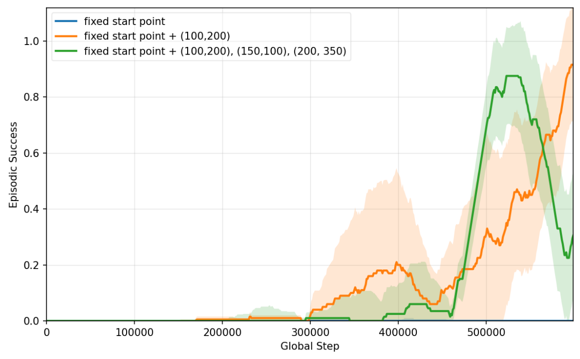
DDPG (fixed start only) vs. DDPG+FixedReset-1 (start $\cup\{(100, 200)\}$) vs. DDPG+FixedReset-3 (start $\cup\{(100, 200), (150, 100), (200, 350)\}$).

6.2.4. Results

Figure 6.1b shows episodic success versus training steps. Despite the dense reward, the baseline DDPG (blue) fails to make any measurable progress within the 600k-step budget. Adding a single reset point (orange) enables learning and leads to gradual improvement toward the end of training. With three reset points (green), learning becomes faster-the agent reaches successful episodes early-but also noticeably unstable, with performance



(a) PointMass layout with start/goal and fixed reset points.



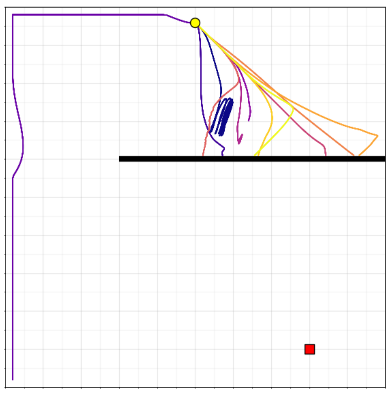
(b) Episodic success during training, measured only for episodes starting from the fixed start point. Curves show a 50-step moving average; shaded bands denote $\pm$ standard error across 5 seeds.

Figure 6.1.: Fixed resets on PointMass with DDPG.

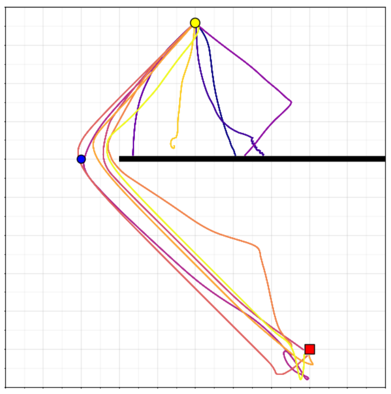
rising and falling after initial success. This indicates that additional reset locations help the agent escape local minima and expose more informative state–action transitions, but too many heterogeneous starting points can destabilize value estimation and lead to oscillatory performance.

Trajectory Visualization. For qualitative reference, we visualize the best evaluation trajectory across checkpoints for one representative seed under the three conditions: no resets, one reset, and three resets (Fig. 6.2). The trajectories reflect the trends discussed above: without resets, the agent initially explores toward the nearby wall and becomes stuck in a local maximum; a single reset point enables consistent progress toward the goal; and with three resets, the agent explores more widely but follows less stable paths, mirroring the oscillatory learning dynamics seen in the quantitative results.

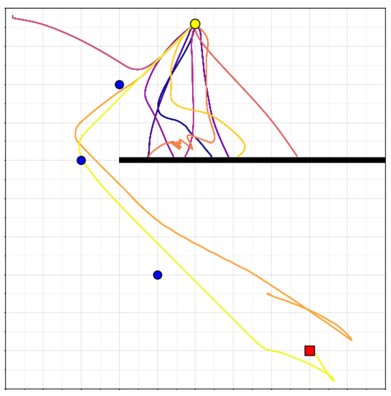
Takeaway. Hand-picked resets significantly improved performance in this simple navigation task, enabling the agent to reach the goal where standard training failed. However,



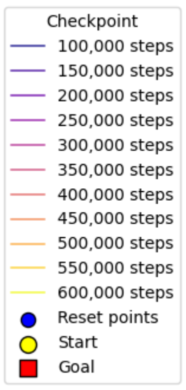
(a) No resets



(b) 1 reset



(c) 3 resets



(d) legend

Figure 6.2.: Evaluation trajectories for a single representative seed under no resets, one reset point, and three reset points. Trajectories are collected periodically during training and visualized with a shared legend.

adding multiple diverse reset points introduced instability in learning, likely because the policy and critic had to adapt to very different initial conditions, making value estimation inconsistent across trajectories.

6.3. AntMaze

6.3.1. Goal

We now turn to the more challenging *AntMaze* environment (see subsection 4.0.2), which provides dense rewards yet remains difficult due to long-horizon credit assignment and complex dynamics. We first establish a backbone that reliably makes progress, then study how different restart distributions and samplers affect learning.

6.3.2. Setup

Environment. The *AntMaze* task (see Figure 4.1b) requires a quadruped agent to navigate a U-shaped maze from a fixed start to a fixed goal. The environment provides a *dense, unshaped* reward based on the negative distance to the goal. Each episode lasts up to 400 environment steps.

Evaluation. We measure performance during training by *episodic success*—the fraction of episodes in which the agent reaches the goal region from the fixed start state before the horizon—using a 500-step moving average with shaded $\pm$ standard error over 5 seeds.

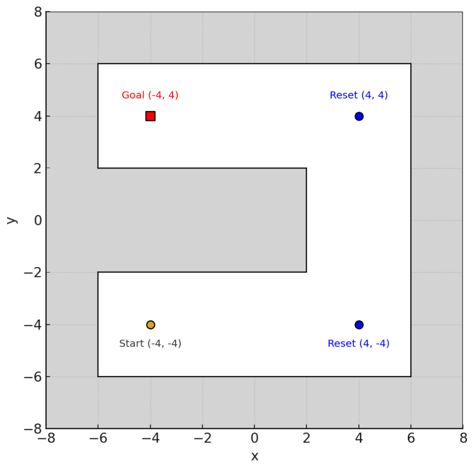
Training. All experiments share this environment configuration and are trained for a total budget of 4×10^6 environment steps. Later subsections specify the backbone algorithm (SAC or SAC+RND) and restart mechanism used in each variant.

6.3.3. Baselines and Manual Resets

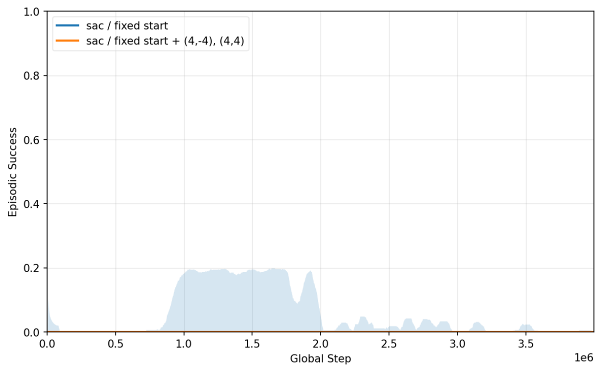
Goal. Establish baseline performance and test whether a few hand-picked reset states improve learning.

Methods compared. SAC (fixed start only) and SAC+FixedReset-2 (two manual reset states at $(4,-4)$ and $(4,4)$).

Results (overview). Figure 6.3 shows (a) the map with start/goal and reset points and (b) the training curves. With a fixed start, SAC achieves success in only a single seed and otherwise fails to learn within the training budget. Adding two manual reset points *worsens* performance overall relative to the fixed-start baseline. The per-seed visitation heatmaps in Fig. 6.4a show almost no exploration beyond the immediate start region, while Fig. 6.4b reveals coverage clustered around the reset states with weak connectivity along the corridor, explaining the degraded and inconsistent outcomes.



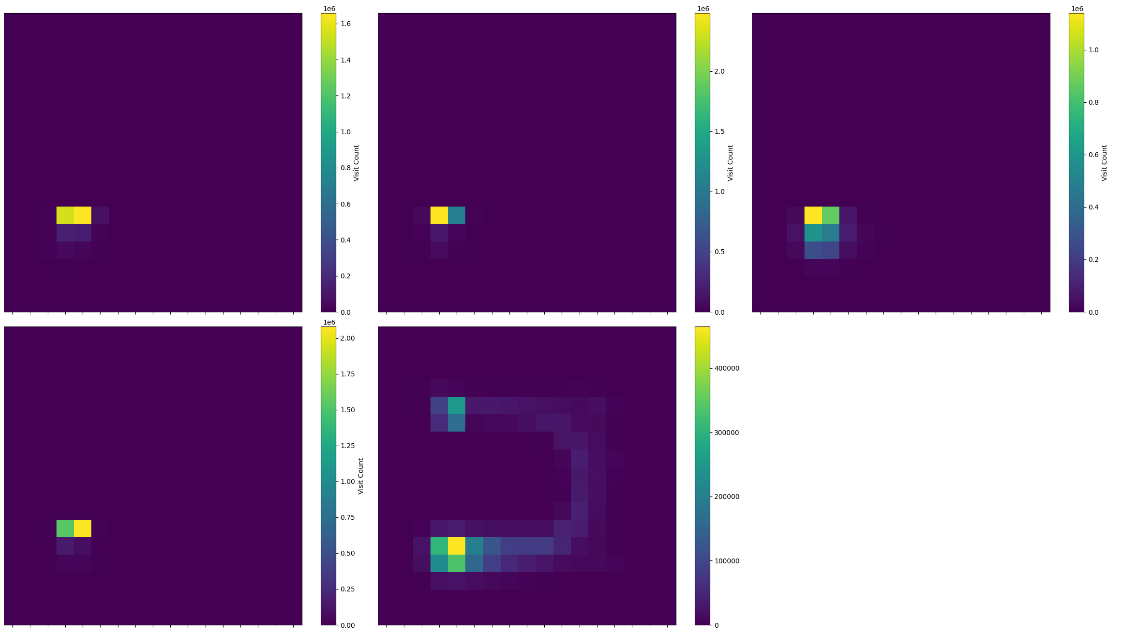
(a) Layout with start $(-4,-4)$, goal $(-4,4)$, and two reset points $(4,-4)$, $(4,4)$.



(b) Episodic success vs. steps (50-step moving average; shaded ±SE over 5 seeds).

Figure 6.3.: AntMaze baselines: fixed start vs. fixed start + manual resets.

Discussion. With a fixed start, SAC rarely reaches the goal, indicating a genuine exploration bottleneck. Surprisingly, adding two manual reset points worsens performance: the agent spends most of its time around those reset locations without learning how to travel between them and the goal. As a result, progress becomes disconnected, and the agent struggles to link actions across longer trajectories to their eventual outcomes. A likely contributing factor is the way resets were implemented: the agent was always reset to the default configuration while only its (x, y) position changed and velocity was set to zero. These resets create states that do not correspond to realistic trajectories, where both orientation and velocity evolve smoothly. Such physically inconsistent restarts may confuse the policy and critic, reducing value accuracy and further harming performance.


(a) SAC (seeds 1–5). *Note:* the first four plots use a 1e6 colorbar scale.

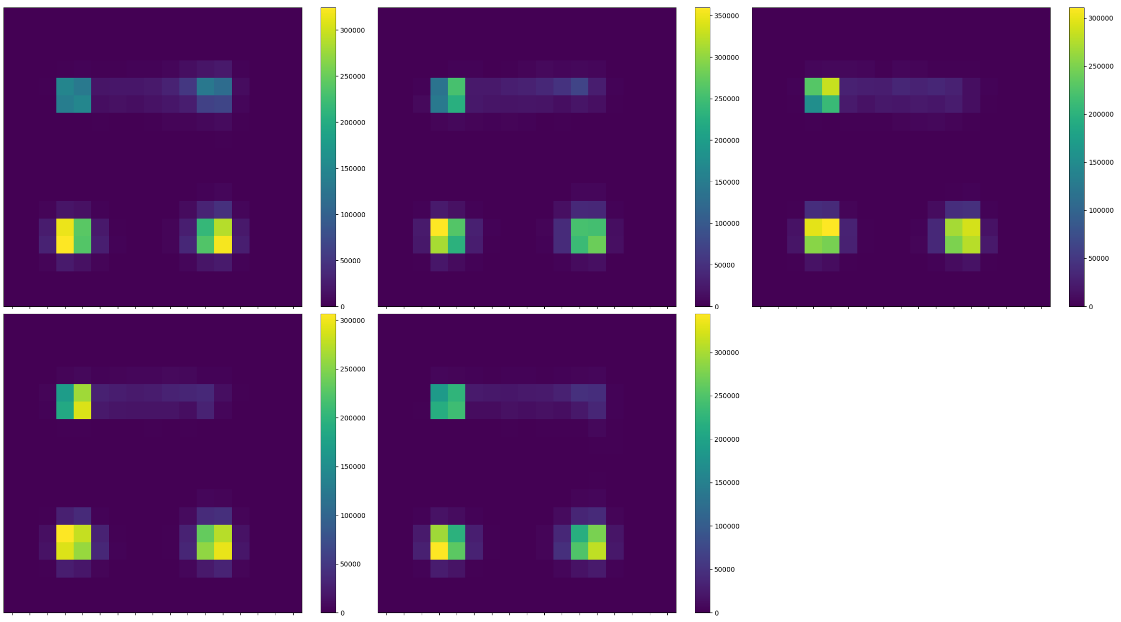

(b) SAC+FixedReset-2 (seeds 1–5).

Figure 6.4.: AntMaze state-visitation heatmaps at the end of training, shown per seed for each baseline method. Each block is aligned column-wise across experiments.

Trajectory Visualization. To illustrate the agent's behavioral progress, we evaluate one representative seed throughout training. At each checkpoint, the agent performs ten evaluation episodes, and we plot the *best trajectory*—defined as the episode achieving the highest return—among them (Fig. 6.5). Without resets or intrinsic motivation, the agent remains confined near the start and repeatedly converges to local optima, failing to explore the maze or reach the goal.

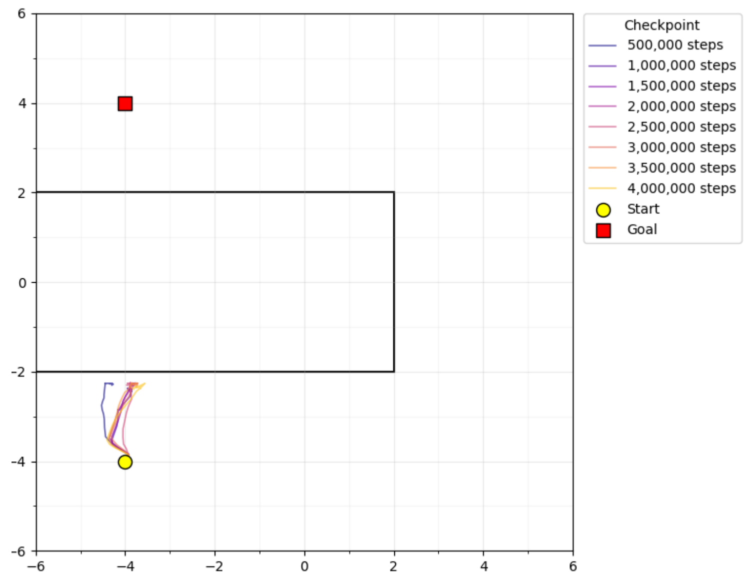


Figure 6.5.: Evaluation trajectories for a single representative seed under *no resets* and *no RND*. At each checkpoint, the best trajectory out of ten evaluation episodes is shown. The agent remains near its initial position, indicating poor exploration and convergence to local optima.

Takeaway. In AntMaze, naive manual resets hurt: they increase local coverage but fail to promote corridor connectivity, degrading success relative to starting from the default state only. In the following experiments, we therefore focus on more realistic and dynamically consistent reset states—those sampled from actual trajectories—so that the agent restarts in states that match feasible motion and velocity patterns.

6.3.4. Choosing SAC+RND as the Baseline

Goal. Decide on a backbone that reliably overcomes AntMaze's exploration bottleneck for all subsequent experiments.

Method. SAC+RND (SAC augmented with Random Network Distillation for novelty).

Results. Figure 6.6 shows a slight improvement in task performance with SAC+RND compared to plain SAC. The agent demonstrates more directed exploration within the maze. The state-visitation heatmaps in Fig. 6.7 confirm this effect, showing broader coverage across the maze, even though the agent still fails to consistently reach the goal.

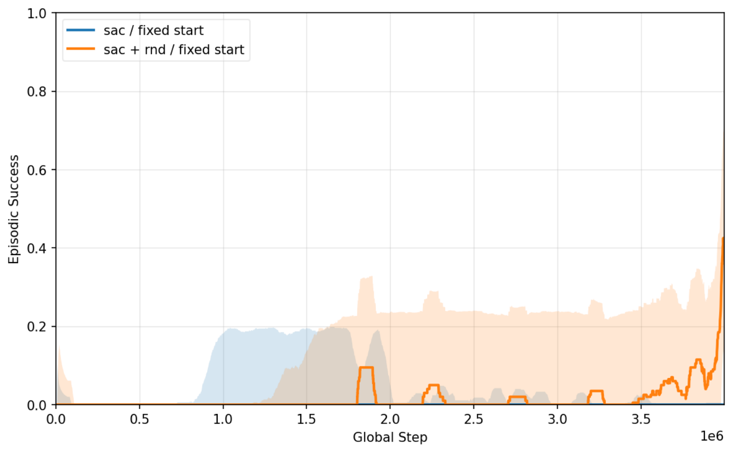


Figure 6.6.: SAC+RND: episodic success vs. steps (50-step moving average; shaded $\pm$SE over 5 seeds).

Decision. Since effective exploration is crucial for the restart-based methods proposed in this work, we adopt SAC+RND as the default backbone for all subsequent AntMaze experiments. The intrinsic reward from RND promotes broader state coverage, which provides a stronger foundation for evaluating the impact of different restart distributions and sampling strategies. When studying these mechanisms next, we keep the RND module fixed and vary only the reset mechanism.

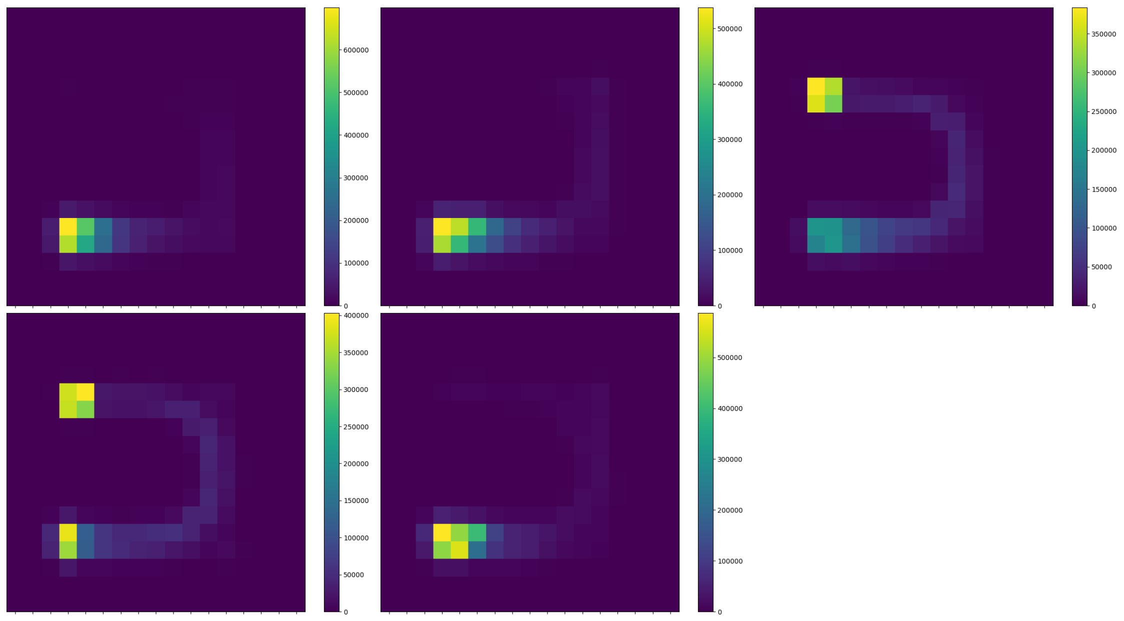


Figure 6.7.: SAC+RND state-visitation heatmaps at the end of training (five seeds). Brighter regions indicate higher visitation frequency, while darker regions indicate less exploration. The agent explores a larger portion of the maze compared to plain SAC.

Trajectory Visualization. To visualize the effect of intrinsic motivation, we repeat the same evaluation protocol as before: one representative seed, ten evaluation episodes per checkpoint, and plotting the best trajectory—the episode that achieved the highest return. As shown in Fig. 6.8, adding RND to SAC encourages the agent to explore a wider variety of configurations and positions compared to the plain SAC baseline (Fig. 6.5). This increased behavioral diversity enables the agent to make gradual progress through the maze.

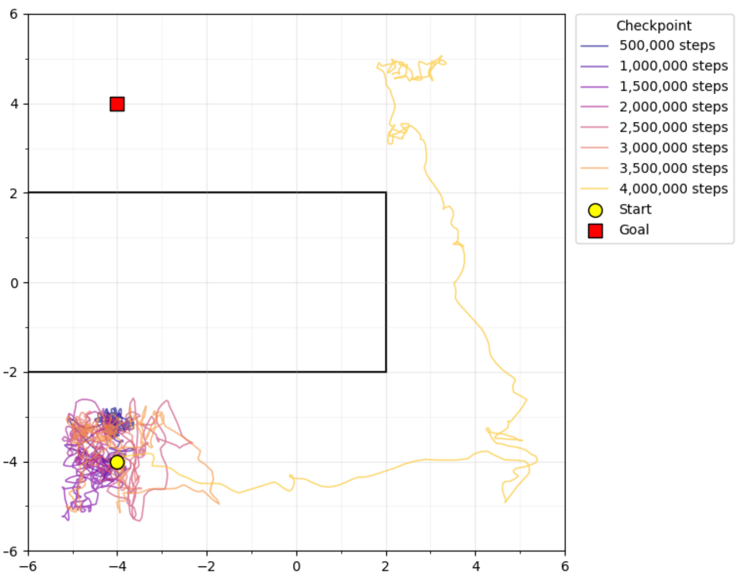


Figure 6.8.: Evaluation trajectories for a single representative seed under *SAC+RND* without resets. At each checkpoint, the best trajectory out of ten evaluation episodes is shown. The addition of RND significantly improves exploration, allowing the agent to traverse new regions of the maze.

6.3.5. Restart Methods

We next evaluate restart mechanisms while keeping SAC+RND as the backbone. Each method modifies only the initial-state distribution.

6.3.6. Restart Buffer (Method #1)

Goal. Improve exploration by starting some episodes from a learned *restart buffer* instead of always from the default start.

What we store. We push *raw observations* o_t (no embedding). Logging can be: (i) *per-step* (push o_t every step) or (ii) *end-of-episode* (push a subset at episode end).

How we add elements (maintenance). The buffer has a fixed capacity $|\mathcal{B}| \leq N{=}1024$ and is maintained according to the strategies introduced in subsection 5.3.2, namely **Random-Replace** and **L2-diversify**. These mechanisms ensure the buffer remains full while retaining diverse samples.

How we sample restarts (independent of maintenance). Sampling follows the schemes introduced in Section 5.3.3, namely **Uniform** and **Softmax**. At each episode start, with probability p_{reset} we draw the initial state $\tilde{s}_0 \sim \pi_{\text{reset}}(\cdot;\mathcal{B})$; otherwise, the environment's default start is used. In this experiment, the Softmax weights are computed using a score based on the combined reward signal— the sum of the environment reward and the RND intrinsic reward, each with coefficient 1.

Note: Either sampler can be paired with either maintenance rule (e.g., Random-Replace+Softmax or L2-div+Uniform).

Integration. Unless stated otherwise, we use SAC+RND as the backbone; the restart buffer changes only the *initial state distribution*.

Variants (names used in plots). For each logging unit (per-step / end-ep), we report all maintenance–sampling combinations:

- **SAC+RND+Reset (per-step, raw, Random-Replace / Uniform)**
- **SAC+RND+Reset (per-step, raw, Random-Replace / Softmax)**
- **SAC+RND+Reset (per-step, raw, L2-div / Uniform)**
- **SAC+RND+Reset (per-step, raw, L2-div / Softmax)**

- **SAC+RND+Reset (end-episode, raw, Random-Replace / Uniform)**
- **SAC+RND+Reset (end-episode, raw, Random-Replace / Softmax)**
- **SAC+RND+Reset (end-episode, raw, L2-div / Uniform)**
- **SAC+RND+Reset (end-episode, raw, L2-div / Softmax)**

We additionally include two baseline restart strategies from [23]—**Prioritized (TD-error)** and **Uniform Restart**—as well as a **No Reset (SAC)** baseline, which corresponds to the same SAC+RND setup but without any reset mechanism.

Reporting. In figure Fig. 6.9 we plot episodic success vs. environment steps (500-step moving average; shaded $\pm$ standard error over 5 seeds).

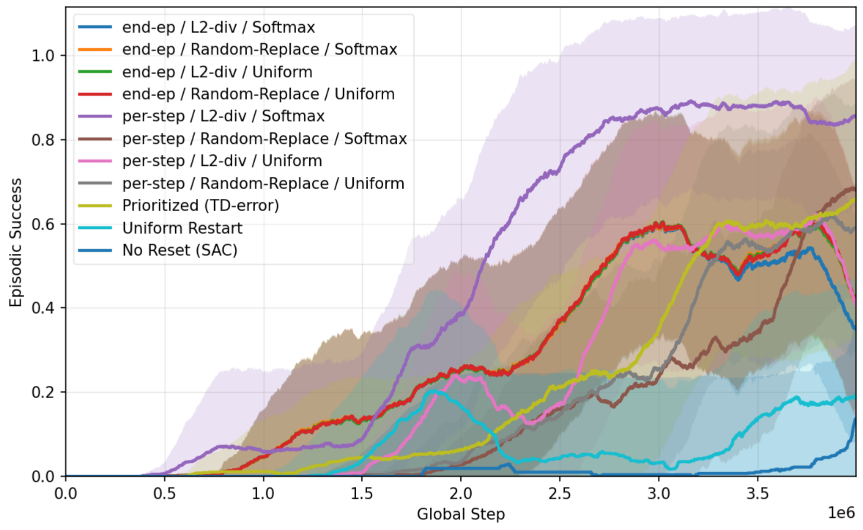


Figure 6.9.: Restart Buffer ablation on AntMaze with SAC+RND. Curves: 500-step moving average; bands: $\pm$ standard error over 5 seeds.

Reset State Distributions (Method #1)

To visualize where restarts actually occurred during training, we plot the (x, y) locations sampled as reset states for each variant (five seeds per method).

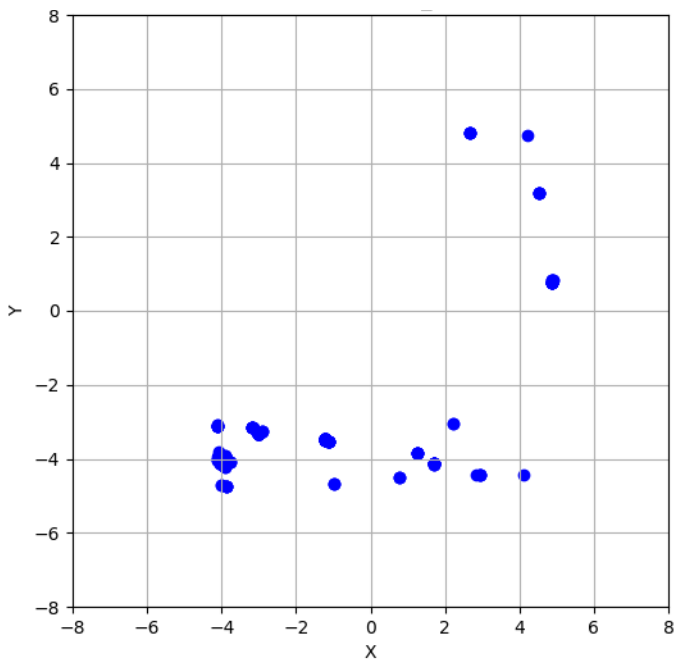


Figure 6.10.: Large view of sampled reset states for the best Method #1 configuration (Per-step / L2-div / Softmax). Shown for a representative seed at the end of training.

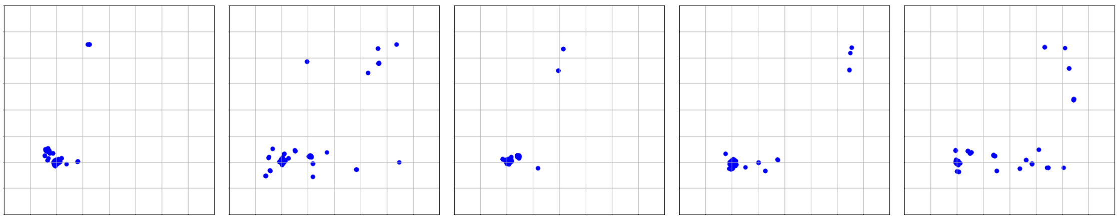


Figure 6.11.: Per-step / L2-div / Softmax — sampled reset states (seeds 1–5).

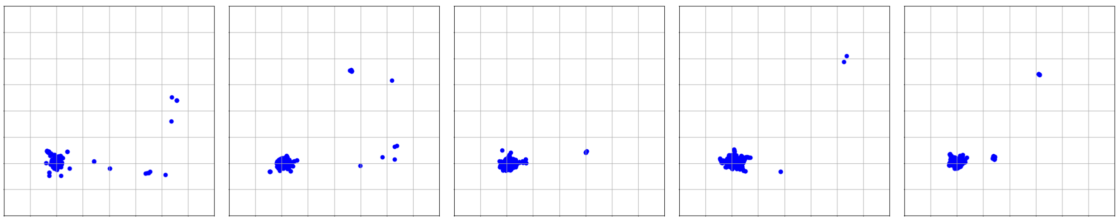


Figure 6.12.: Per-step / Random-Replace / Softmax — sampled reset states (seeds 1–5).

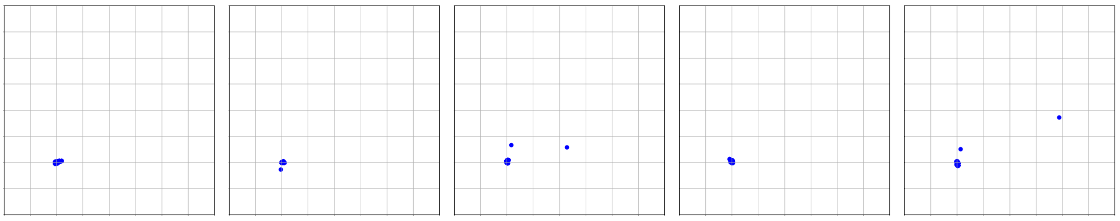


Figure 6.13.: Per-step / L2-div / Uniform — sampled reset states (seeds 1–5).

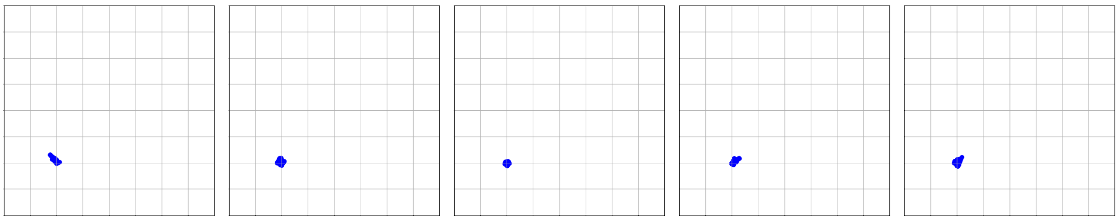


Figure 6.14.: Per-step / Random-Replace / Uniform — sampled reset states (seeds 1–5).

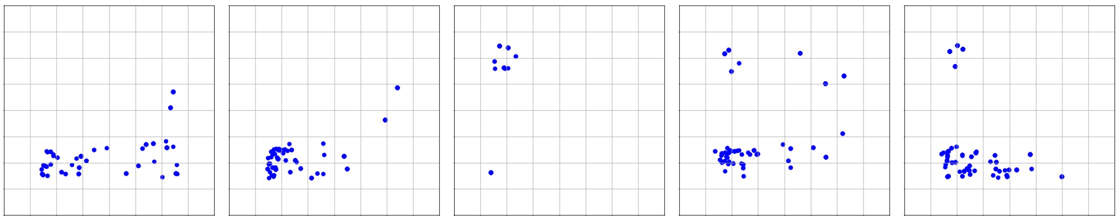


Figure 6.15.: End-episode / L2-div / Softmax — sampled reset states (seeds 1–5).

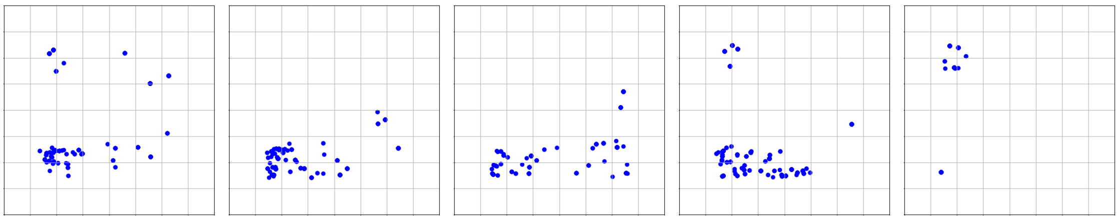


Figure 6.16.: End-episode / Random-Replace / Softmax — sampled reset states (seeds 1–5).

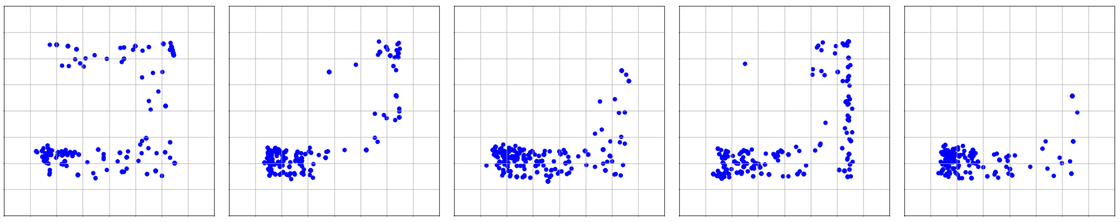


Figure 6.17.: End-episode / L2-div / Uniform — sampled reset states (seeds 1–5).

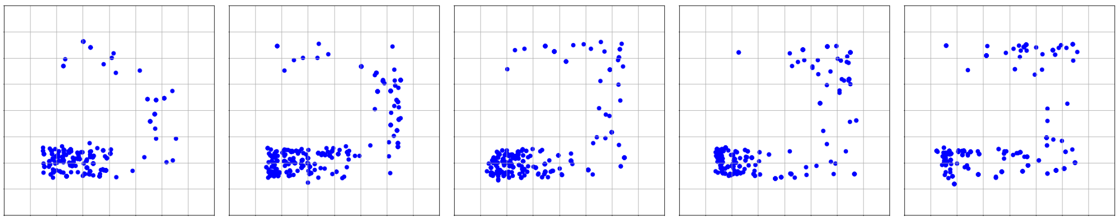


Figure 6.18.: End-episode / Random-Replace / Uniform — sampled reset states (seeds 1–5).

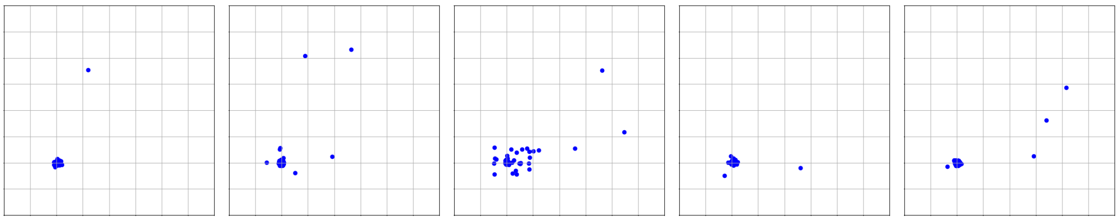


Figure 6.19.: Prioritized (TD-error) — sampled reset states (seeds 1–5).

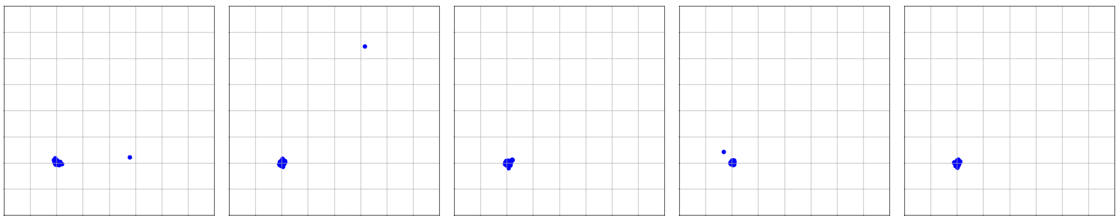


Figure 6.20.: Uniform Restart — sampled reset states (seeds 1–5).

Discussion. Figure 6.9 shows the performance of all restart buffer variants on AntMaze. The **per-step / L2-div / Softmax** configuration is the only one that consistently reaches high success, achieving both faster learning and higher final performance. Most other restart variants—including **Random-Replace / Uniform** and **L2-div / Uniform**—show moderate improvements over the no-reset SAC baseline, suggesting that simple restarts from replayed states can already help exploration to some extent. The **Prioritized (TD-error)** baseline performs slightly better than these mid-tier variants, whereas the external **Uniform Restart** baseline performs poorly, nearly matching the no-reset baseline, indicating that uniform restarts alone are insufficient to drive meaningful exploration.

Examining the restart-state distributions (Figure 6.11–Figure 6.20) reveals clear structural differences between variants. Per-step methods tend to collect and sample a large number of points concentrated near the start region, whereas end-of-episode variants exhibit a more even spread along the agent's trajectories. However, this broader spatial coverage did not translate into higher success—none of the end-episode variants outperformed the best per-step configuration. As shown in Figure 6.11, the best-performing variant (**Per-step / L2-div / Softmax**) displays many restart states clustered near the starting point but also several distributed along the trajectory toward the goal. This combination may be particularly effective, as it reinforces early exploration while still exposing the agent to progress along successful paths. Interestingly, several methods with very similar restart maps—such as **Uniform Restart**, **Per-step / Random-Replace / Uniform**, and **Per-step / L2-div / Uniform**— achieved notably different final success rates (approximately 0.2, 0.63, and 0.41, respectively). The reasons for these discrepancies are not entirely clear and may stem from subtle interactions between sampling frequency, buffer diversity, and value estimation stability.

Trajectory Visualization. We follow the same evaluation setup as before: one representative seed, ten evaluation episodes per checkpoint, and plotting the best trajectory (the episode achieving the highest return). As shown in Fig. 6.21, integrating the restart mechanism with SAC+RND—specifically using the best-performing configuration (*Per-step / L2-div / Softmax*)—significantly enhances the agent's efficiency in reaching the goal. The trajectories illustrate smoother and more directed exploration compared to the plain SAC+RND case (Fig. 6.8), confirming that targeted restarts can accelerate progress once intrinsic rewards have established basic exploratory behavior.

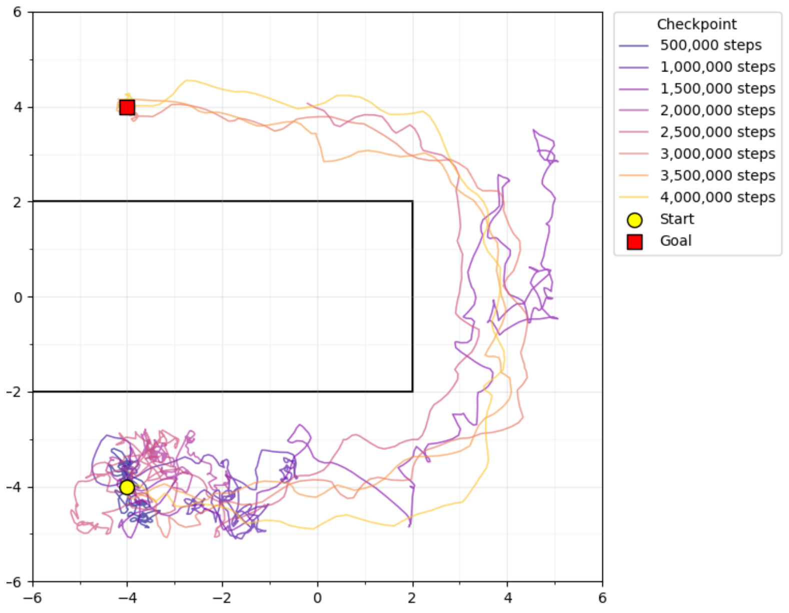


Figure 6.21.: Evaluation trajectories for a single representative seed under the best-performing restart configuration (*Per-step / L2-div / Softmax*). At each checkpoint, the best trajectory—the episode with the highest return—is shown. Compared to the *SAC+RND* baseline, the addition of state resets enables faster progress toward the goal and more consistent traversal of the maze.

Takeaway. Substantial gains arise only when combining diversity-aware maintenance (*L2-diversify*) with reward-weighted Softmax sampling and per-step logging, highlighting the importance of both diversity and reward-guided prioritization for effective restart-driven exploration. At the same time, many sampled states still cluster around the same few regions, likely because the ant's motion is limited early in training. In the following experiments, we therefore explore ways to increase diversity further and broaden the range of restart states.

6.3.7. Method #1 Focused Ablation: Reset Ratio and 10-Step Cap

Goal. Test two restart-specific knobs (inspired by prior work) on our best Method #1 configuration.

Base configuration. SAC+RND+Reset with *per-step* logging, *raw* observations, *L2-diversify* maintenance, and *Softmax* sampling.

Settings. We vary the reset probability $p_{\text{reset}} \in \{0.1, 0.5\}$. When an episode starts from the buffer, we cap its length at a *fixed* $H_{\text{ep}} = 10$ steps (default-start episodes keep the full horizon $H{=}400$).

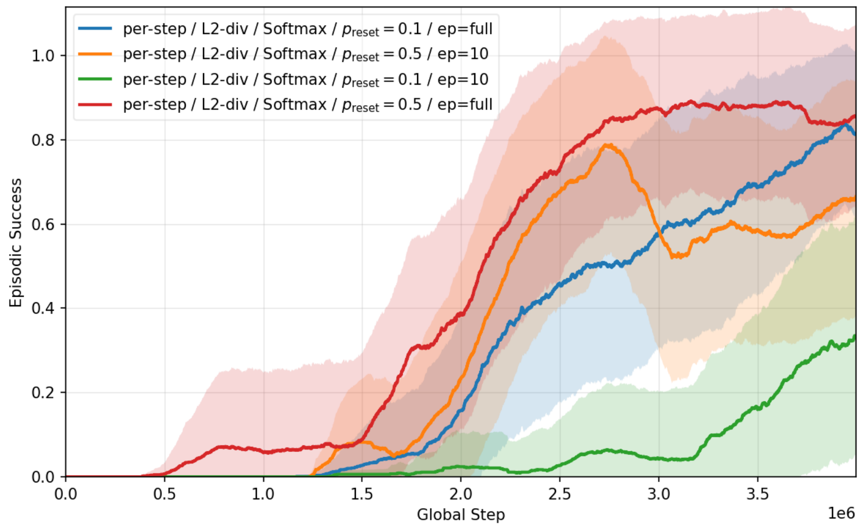


Figure 6.22.: Restart Buffer ablation on AntMaze with SAC+RND. Curves: 500-step moving average; bands: $\pm$ standard error over 5 seeds. Here, **rc** denotes the reset probability (p_{reset}), and **ep** the episode length after a restart to a sampled state.

Discussion. As shown in Fig. 6.22, none of the tested configurations yield a significant improvement over the base setup. While increasing the reset ratio ($p_{\text{reset}} = 0.5$) or limiting reset episodes to 10 steps slightly affects learning dynamics, overall performance remains comparable across settings. This suggests that within our training budget, these hyperparameters have limited influence on the effectiveness of the restart buffer.

6.3.8. Restart Buffer with Actor Embeddings (Method #2)

Goal. Extend Method #1 (see 6.3.6) by using actor network embeddings to represent stored observations, aiming to further diversify the restart buffer. The hypothesis is that increased diversity in the stored states can promote broader exploration and, in turn, more effective problem solving.

Method. Each observation o_t is embedded via the actor network as $z_t = \phi_{\text{actor}}(o_t)$ (detached and ℓ_2-normalized), and both (o_t, z_t) are stored in the buffer. Embeddings are refreshed every $K{=}10{,}000$ environment steps by recomputing $z \leftarrow \phi_{\text{actor}}(o)$ for all entries in $\mathcal{B}$. All other components—maintenance, sampling, and integration—follow the same procedure as in Method #1 (see subsection 6.3.6).

Variants (names used in plots). For each logging unit (per-step / end-ep), we report:

- **SAC+RND+Reset (per-step, raw+emb, L2-div / Uniform)**
- **SAC+RND+Reset (per-step, raw+emb, L2-div / Softmax)**
- **SAC+RND+Reset (end-ep, raw+emb, L2-div / Uniform)**
- **SAC+RND+Reset (end-ep, raw+emb, L2-div / Softmax)**

Reset State Distributions (Method #2: Actor Embeddings)

Discussion. As shown in Fig. 6.23, using actor embeddings to diversify the restart buffer does not yield improvements over the best Method #1 configuration. While the **end-episode / L2-div / Uniform** variant (orange) shows somewhat similar performance, none of the embedding-based approaches surpass the raw observation baseline. This suggests that, in this setup, embedding-based diversity does not provide a clear exploration advantage. A possible limitation of this approach is that the actor network must first be sufficiently trained before its embeddings become meaningful; during early training, the representation is still unstable and may not reflect useful state similarities. Additionally, in the raw observation space, the first two coordinates (x, y) dominate the distance metric due to their larger scale compared to other features, whereas in the embedding space all dimensions are normalized to similar ranges. This difference changes how diversity is measured and may partly explain why the embedding-based variants behave

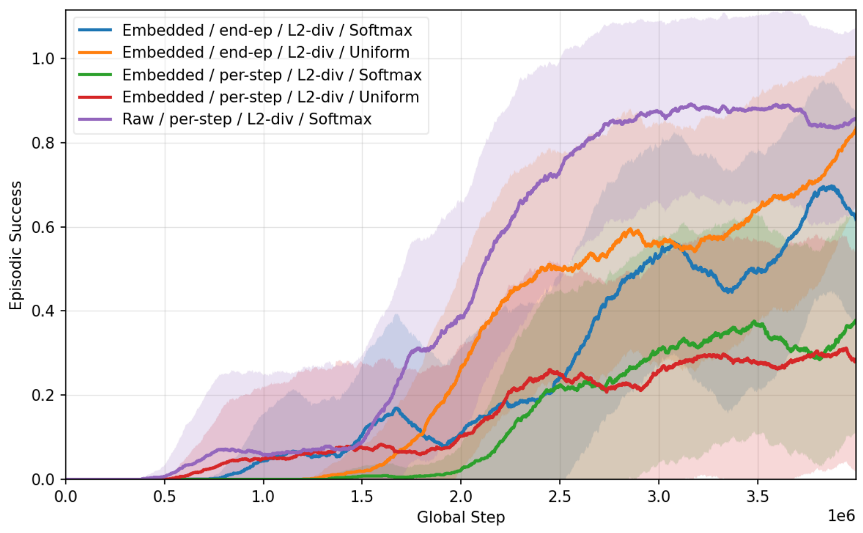


Figure 6.23.: Restart Buffer on AntMaze with SAC+RND. Curves: 500-step moving average; bands: $\pm$ standard error over 5 seeds.

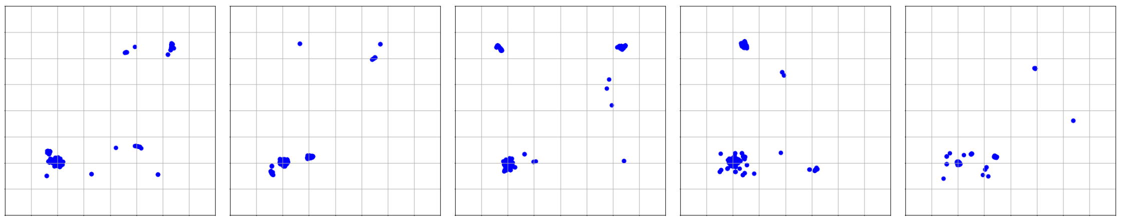


Figure 6.24.: Embedded / per-step / L2-div / Softmax — sampled reset states (seeds 1–5).

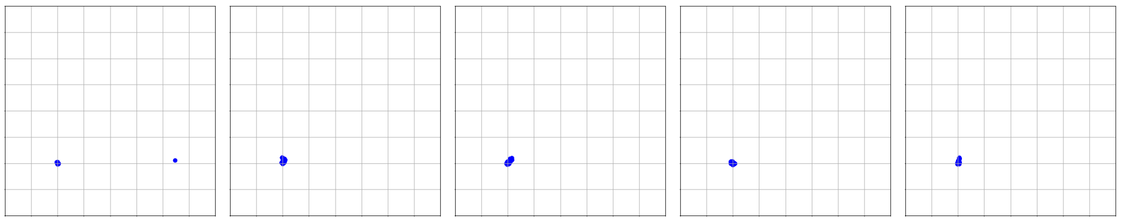


Figure 6.25.: Embedded / per-step / L2-div / Uniform — sampled reset states (seeds 1–5).

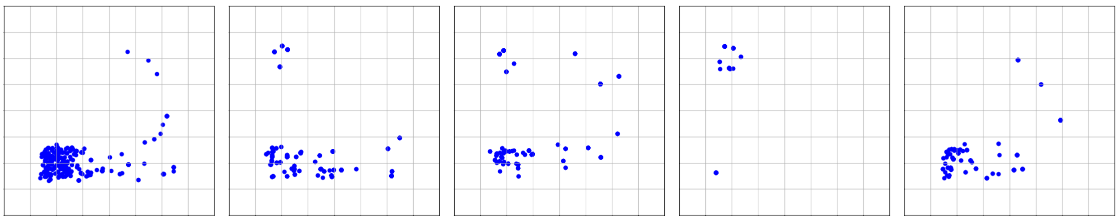


Figure 6.26.: Embedded / end-episode / L2-div / Softmax — sampled reset states (seeds 1–5).

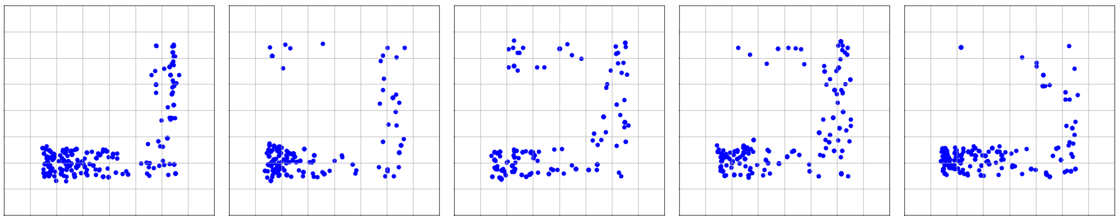


Figure 6.27.: Embedded / end-episode / L2-div / Uniform — sampled reset states (seeds 1–5).

differently from those using raw observations. As seen in the reset distributions (Figure 6.24–Figure 6.27), the spatial spread of restart points in (x, y) is not noticeably broader than in Method #1—particularly for the per-step variants, which remain concentrated near the start area. This indicates that using embeddings did not lead to more diverse exploration in physical space, even though it increased diversity in representation space.

Takeaway. Using actor embeddings to represent restart states did not improve exploration in this setting. The need for a warm-up period before the embeddings become meaningful, together with the normalization of all features to similar scales, likely weakens the role of spatial information such as (x, y). As a result, the embedding-based buffer captures diversity in representation space but not necessarily in physically relevant terms, limiting its effectiveness for guiding exploration.

6.3.9. Successful-Trajectory Restarts (Method #3)

Goal. Investigate whether resetting to states sampled along a previously successful trajectory can accelerate convergence and improve learning efficiency.

Overview. This method extends the restart buffer of Method #1 (see 6.3.6) by introducing resets from a discovered successful trajectory. Training begins identically to Method #1—using raw observations, per-step or end-episode logging, Random-Replace or L2-div maintenance, and Uniform or Softmax sampling. While no successful episode is found, resets are drawn from the buffer $\mathcal{B}$ as usual. Once a complete start-to-goal rollout is discovered, the system switches to a *trajectory-restart* mode that samples reset states uniformly from that trajectory and assigns the remaining episode budget accordingly.

Successful-trajectory memory. Let the episode horizon be H ($H{=}400$ in AntMaze). When the agent first reaches the goal, we store the successful trajectory

$$\tau^\star = (o_0, a_0, \dots, o_{L^\star}),$$

where $L^\star \le H$ is its length. If a later run finds a *higher-reward* successful trajectory, we replace $\tau^\star$.

Curriculum resets along the trajectory (uniform). At the start of an episode (with prob. p_{traj}), pick an index

$$k \sim \text{Uniform}\{0, \dots, L^\star{-}1\}$$

and reset to o_k. We cap the episode length to the remaining budget

$$H_{\text{ep}} \;=\; \min\bigl(H,\; L^\star - k\bigr).$$

Example: if $k{=}100$ on a $L^\star{=}400$ trajectory, the episode gets 300 steps. This encourages stitching the known suffix and prevents long detours.

Interaction with the restart buffer. Before discovering $\tau^\star$, we use Method #1 exactly (resets drawn from $\mathcal{B}$ with probability $p_{\text{reset}}{=}0.5$). After discovery, with the same probability $p_{\text{reset}}{=}0.5$ we *always* reset to a state sampled from $\tau^\star$ (uniform k), with the episode length capped to the remaining budget $H_{\text{ep}} = \min(H,\, L^\star - k)$; otherwise we start from the default initial state. The buffer $\mathcal{B}$ continues to be updated during training but is no longer used as a reset source once $\tau^\star$ is available.

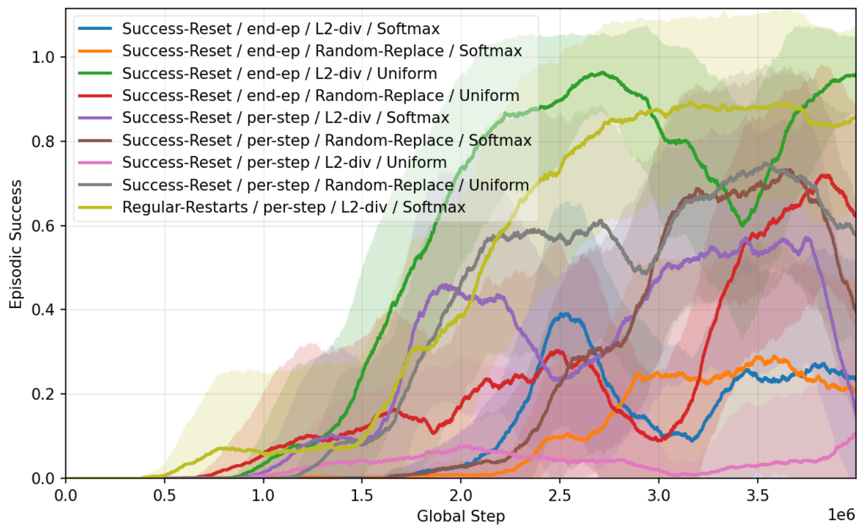
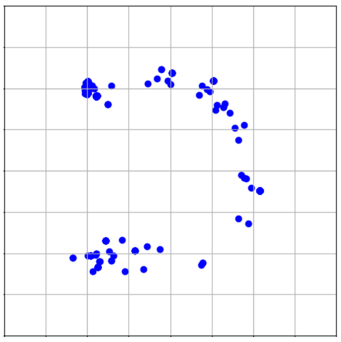
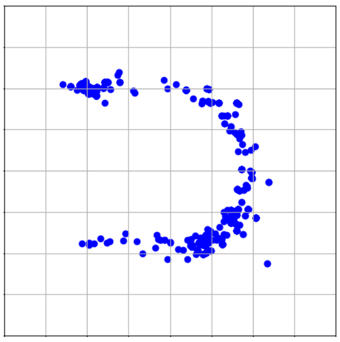


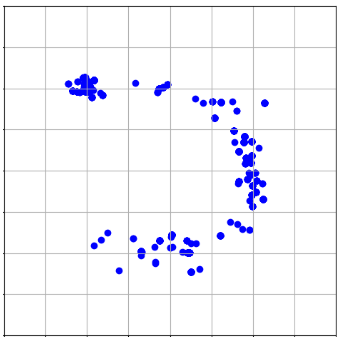
Figure 6.28.: SAC+RND+TrajRestart on AntMaze. Episodic success vs. steps (500-step moving average; shaded $\pm$ SE over 5 seeds).

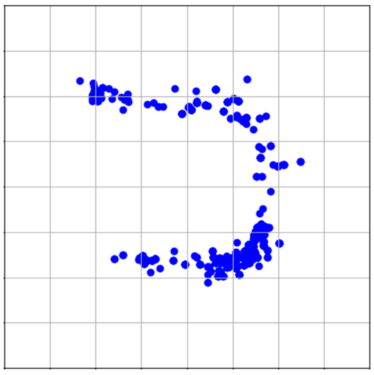
Discussion. As shown in Fig. 6.28, only one Success-Reset configuration—*end-episode / L2-div / Uniform*—outperforms the best Method #1 baseline (per-step / L2-div / Softmax), achieving a faster rise and a higher final success rate. All other variants perform worse, converging more slowly or less consistently. Furthermore, most Success-Reset curves exhibit sharp mid-training drops, indicating that resetting to successful trajectories can introduce instability. Consistent with Fig. 6.29, the reset states are diverse and span the entire path from the initial point to the goal, which appears to support quicker and more effective learning in this best-performing configuration.

Takeaway. Restarting from successful trajectories sometimes improves performance but behaves inconsistently. Only one variant showed a clear benefit, while others led to unstable or weaker learning, suggesting that the method's effectiveness depends on factors not yet well understood.


(a) Seed 1


(b) Seed 2


(c) Seed 3


(d) Seed 4

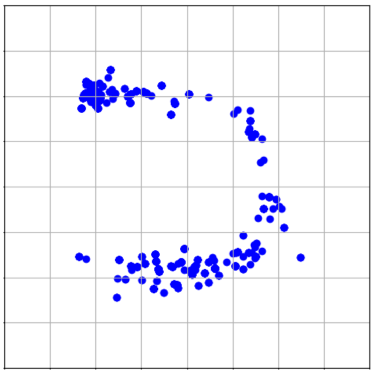

(e) Seed 5

Figure 6.29.: Sampled reset states per seed for the best Success-Reset configuration (*end-ep / L2-div / Uniform*). Points are states sampled from successful trajectories during training.

6.4. Fetch Pick-and-Place

Goal. Test whether Method #1 (restart buffer with raw observations; see Sec. 6.3.6) improves sample efficiency on a dense, multi-goal, goal-conditioned manipulation task.

Setup. Environment horizon $H=50$ steps with the standard `is_success` signal (goal achieved within the horizon). Curves use a 50-step moving average; shaded bands denote $\pm$ standard error over 5 seeds. Backbone is SAC (no RND).

Method. SAC+Reset with the Method #1 design (Sec. 6.3.6): raw observations, *per-step* logging into a capacity-N buffer, maintenance via either **Random-Replace** or **L2-diversify**, and restart sampling by either **Uniform** or **Softmax**. The restart probability is $p_{\text{reset}}=0.5$ unless otherwise noted.

Results. Figure 6.30 shows that none of the restart-based variants outperform the baseline SAC without resets. While several configurations achieve moderate success, all restart methods ultimately lag behind the baseline in both convergence speed and final performance. This suggests that in the dense, multi-goal Fetch Pick-and-Place task, resetting to previously visited states does not aid exploration and instead makes learning harder, likely by disrupting the natural goal-conditioned progression of the policy.

Takeaway. In this dense, multi-goal setting, the restart-based mechanisms do not align well with the task structure. By biasing resets toward states that happened to be closer to previous goals, the buffer introduces an unintended advantage that becomes meaningless once the goal changes each episode, ultimately hindering learning rather than improving it.

6.5. HalfCheetah

Goal. Probe whether Method #1 (Sec. 6.3.6) offers benefits on a dense-reward locomotion task where exploration is not the main bottleneck.

Setup. No binary success is defined; we report **episode return** during training (as in Sec. 6.1.1). Curves show a 50-step moving average with $\pm$ standard error over 5 seeds. Backbone is SAC (no RND).

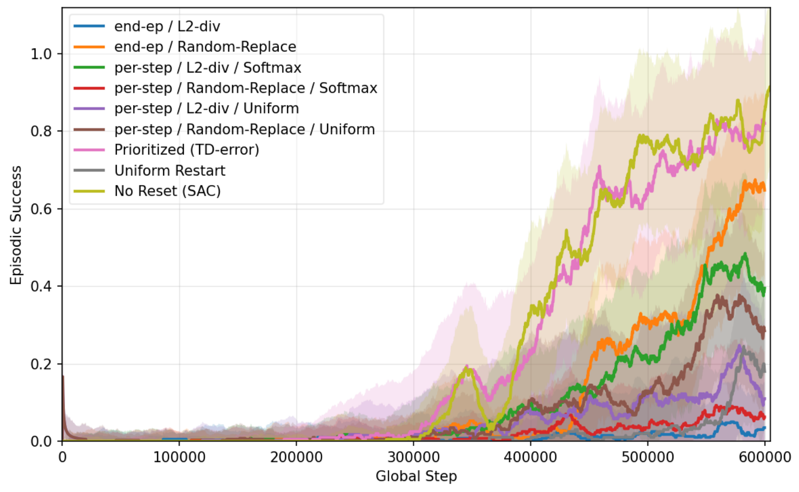


Figure 6.30.: **Fetch Pick-and-Place** with Method #1 (SAC backbone). Success rate during training (50-step moving average; $\pm$SE over 5 seeds).

Method. As in Method #1 with raw observations and *per-step* logging; maintenance via **Random-Replace** or **L2-diversify**; restart sampling by **Uniform** or **Softmax**.

Results. Figure 6.31 shows that during early and mid training, most restart-based variants achieve slightly higher returns than the baseline SAC. However, this advantage diminishes as training progresses, and by convergence, all methods reach comparable performance. Only two variants maintain a marginal lead at the end, but the difference appears statistically insignificant. Overall, the restart mechanisms do not meaningfully improve long-term performance on this dense locomotion task.

Discussion. Restart variants show small early gains that fade by convergence, with only minor differences in final performance. Unlike maze navigation tasks, *HalfCheetah* has no rare or hard-to-reach states—the agent can easily reach high-velocity, high-reward configurations from the default start because locomotion is simple and smooth. Each step already provides informative velocity-based rewards, so exploration is not a bottleneck. As a result, restarting from previous states adds little new experience and mostly perturbs the natural training dynamics, giving only a short-term boost without improving the final outcome.

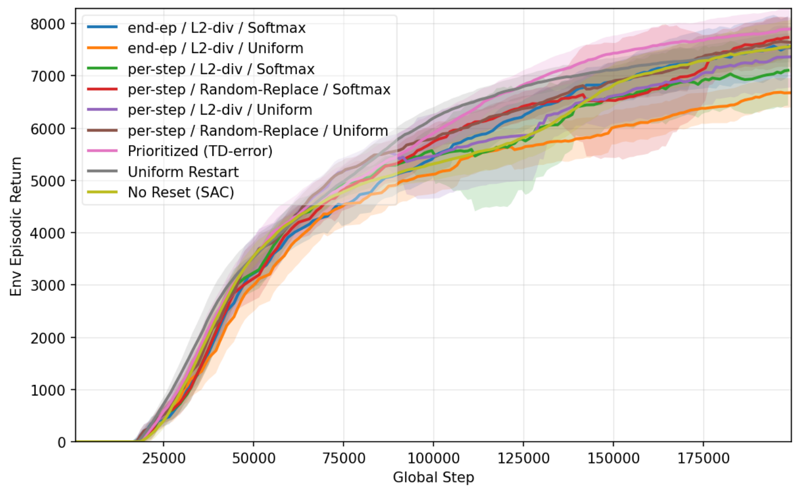


Figure 6.31.: **HalfCheetah** with Method #1 (SAC backbone). Episode return during training (50-step moving average; $\pm$SE over 5 seeds). Diversity-aware maintenance with Softmax sampling modestly improves both learning speed and final return.

7. Discussion

The previous chapter presented experimental results for several restart-based reinforcement learning strategies across four dense-reward environments: *PointMass*, *AntMaze*, *Fetch Pick-and-Place*, and *HalfCheetah*. This chapter takes a broader view of those findings. It discusses when and why restart mechanisms help, when they fail, and which design choices matter most. Finally, it summarizes key lessons, practical guidelines, limitations, and potential directions for future research.

7.1. When and Why Restarts Help

Restart mechanisms proved most beneficial in environments that combine dense rewards with challenging exploration or long-horizon credit assignment. This effect was most evident in the maze-based tasks, particularly **PointMass** and **AntMaze**, where the agent must navigate complex spatial layouts and overcome exploration bottlenecks. In such settings, learning from a single fixed start can severely limit coverage of the state space, whereas restarting from previously visited states exposes the agent to a broader range of experiences and more informative feedback within the same training budget.

In **AntMaze**, the combination of *diversity-aware maintenance* (**L2-diversify**) and *reward-weighted sampling* (**Softmax**) consistently achieved the best results. Together, these mechanisms ensured that the restart buffer contained a balanced mix of informative and varied states, while sampling favored regions that contributed most to progress. The resulting training dynamics resembled a self-organized curriculum, where the agent gradually extended its exploration frontier without manual intervention.

7.2. When Restarts Fail

In contrast, restart strategies did not help, and sometimes hindered, learning in environments that were either easy to explore or involved frequently changing goals.

In the **HalfCheetah** task, the dense reward already provides a strong learning signal that encourages steady improvement. Here, restart mechanisms offered only a small initial benefit and converged to the same final performance as the baseline. Since exploration is not a bottleneck, additional resets merely reintroduced familiar states without adding useful experience.

The **Fetch Pick-and-Place** task showed a stronger negative effect. Although its reward is dense, each episode uses a new goal position. The restart buffer therefore stored states that were useful for previous goals, which became irrelevant in the next episode. Sampling from these outdated states biased the learning process and disrupted the goal-conditioned policy updates. This demonstrates that restarts can be harmful when the task's objective or goal distribution changes over time.

A similar limitation appeared in the **trajectory-based restarts** tested on AntMaze. Once the agent found a successful trajectory, repeatedly restarting along it sometimes accelerated short-term progress but often destabilized training. The sudden change in the initial-state distribution made value estimates inconsistent and caused temporary drops in performance. These observations highlight that restart mechanisms must be applied carefully to avoid unintended shifts in the training distribution. A more structured curriculum approach, where restart locations progress gradually along the discovered trajectory instead of being sampled at random, might lead to more stable and consistent learning outcomes.

7.3. Conclusions

The experimental results demonstrate that restart-based exploration can improve reinforcement learning performance in certain dense-reward settings, but its effectiveness strongly depends on the task structure and the specific design of the restart mechanism.

Restarts were most beneficial in long-horizon, exploration-heavy environments such as maze-navigation tasks (**PointMass** and **AntMaze**). In these settings, restarting from previously visited states allowed the agent to revisit meaningful regions of the state space and accelerate progress beyond what was possible from a single fixed start. Among all tested configurations, the combination of *diversity-aware maintenance* (**L2-diversify**) and *reward-weighted sampling* (**Softmax**) consistently achieved the best results in **AntMaze**, highlighting the importance of maintaining both diversity and relevance in the restart buffer.

In contrast, restarts provided little or no benefit in environments where exploration is naturally easy or where the goal changes frequently, such as **HalfCheetah** and **Fetch Pick-and-Place**. In these cases, restarting from previously visited states mostly reintroduced familiar or outdated configurations, which added noise without contributing useful experience.

Overall, these findings show that the success of restart mechanisms depends on achieving a careful balance between diversity, relevance, and stability in the initial-state distribution. When designed appropriately, restarts can serve as an effective tool for accelerating learning and improving exploration in dense-reward environments that still present significant challenges in credit assignment and coverage.

7.4. Limitations and Future Work

Although the study provides consistent qualitative trends across environments, the results show considerable variance and randomness between runs, making it difficult to draw definite quantitative conclusions. This variability suggests that the effectiveness of restart mechanisms can be sensitive to stochastic factors such as initialization, exploration noise, and environment dynamics.

Another limitation concerns the limited range of tested environments. Two of the tasks (**PointMass** and **AntMaze**) are maze-like navigation problems with strong exploration challenges, while the other two (**HalfCheetah** and **Fetch Pick-and-Place**) are comparatively easy from an exploration perspective. It would therefore be valuable to evaluate restart strategies on a broader set of exploration-heavy domains, including more diverse locomotion or manipulation tasks.

A further limitation lies in the restricted hyperparameter coverage. Restart-based methods introduce several additional design choices, such as buffer size, reset frequency, sampling strategy, diversity metric, and the method used to score or rank states within the buffer, that were only partially explored in this work. In particular, the state-score function was based solely on the environment reward, while alternative scoring criteria (for example, value estimates, intrinsic rewards, or hybrid metrics) were not examined. Due to hardware constraints, only a small number of representative variants were tested rather than a full hyperparameter sweep. Different combinations of these factors might lead to stronger or more stable improvements.

Future research could explore adaptive restart probabilities that adjust dynamically during training or learned restart policies that predict which states are most useful to revisit. Finally, extending restart-based exploration to real robotic systems, where physical resets are costly, could provide valuable insights into their practical utility beyond simulation.

7.5. Summary

Restart mechanisms can enhance exploration and efficiency in dense-reward environments with complex dynamics or long horizons. However, they are not universally beneficial; in tasks with simple exploration or changing goals, they may even hinder learning. Their effectiveness depends on maintaining a diverse and informative restart buffer while balancing exploration coverage with relevance to the current objective. Overall, these results clarify the potential and the limits of learning from previously visited states, and they highlight restart design as an important tool, when used with care, for improving reinforcement learning performance.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2 ed., 2018.

[2] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *International Conference on Machine Learning*, pp. 2778–2787, 2017.

[3] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “Go-explore: a new approach for hard-exploration problems,” 2021.

[4] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “First return, then explore,” *Nature*, vol. 590, p. 580–586, Feb. 2021.

[5] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, “Deep exploration via bootstrapped dqn,” 2016.

[6] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” 2019.

[7] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” 2018.

[8] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in Neural Information Processing Systems*, pp. 1057–1063, 2000.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International Conference on Machine Learning*, pp. 1928–1937, 2016.

[11] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

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

[14] A. L. Strehl and M. L. Littman, "An analysis of model-based interval estimation for reinforcement learning," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1309–1331, 2008.

[15] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

[16] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2019.

[17] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *Journal of Machine Learning Research*, vol. 21, no. 181, pp. 1–50, 2020.

[18] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on Robot Learning*, pp. 482–495, PMLR, 2017.

[19] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[20] A. Kumar, J. Fu, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," 2019.

[21] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," in *International Conference on Learning Representations*, 2017.

[22] T. Salimans and R. Chen, “Learning montezuma’s revenge from a single demonstration,” in *NeurIPS Deep Reinforcement Learning Workshop*, 2018.

[23] A. Tavakoli, V. Levdik, R. Islam, C. M. Smith, and P. Kormushev, “Exploring restart distributions,” in *Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2019.

[24] D. Warde-Farley, T. V. de Wiele, T. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih, “Unsupervised control through non-parametric discriminative rewards,” 2018.

A. Appendix

A.1. External Baselines: Re-implementation Details

We re-implemented **Uniform Restart** and **Prioritised Restart** from Tavakoli et al. ("Exploring Restart Distributions"). Our code follows their mechanisms and hyperparameters where applicable: (i) maintain a restart memory of previously encountered states; (ii) sample augmented initial states with fixed ratio (0.1 of interactions) and apply a shorter time limit for augmented starts; (iii) for *Uniform*, sample uniformly; (iv) for *Prioritised*, compute state priorities via value TD error and sample with proportional/softmax weighting; Differences: we use **SAC** as the backbone (not PPO) and match our observation design (relative-goal prepend) and horizon $H{=}400$. All other training budgets, seed counts, and evaluation protocols are aligned with our main experiments. [23]