Bridging Theory and Practice in Actor–Critic RL for Data Efficiency

Yifan Wu, Xiuqi Han, Chenhao Zhang, Fengze Zhang

Purdue University

ABSTRACT

Actor–critic (AC) methods balance policy optimization and value estimation, yet achieving high data efficiency in practice remains challenging. On-policy and off-policy AC algorithms exhibit complementary trade-offs between training stability and sample reuse. We propose a two-stage training framework that uses Natural Actor–Critic with Data Drop (NACDD) as a stable on-policy warm-up, followed by off-policy optimization with Deep Deterministic Policy Gradient (DDPG). In addition, we systematically analyze the Data Drop mechanism in NAC-DD and introduce an Adaptive Data Drop rule based on online TD-error correlation.

1 INTRODUCTION

Actor–critic (AC) methods form a central pillar of modern reinforcement learning (RL), combining parametric policy gradients with value-function estimation. Recent theoretical work has significantly strengthened our understanding of AC with deep function approximators [1, 3, 6]. Among all the attempts to improve the agent's performance, how to increase the efficiency of data exploitation remains an important challenge. This efficiency is formally characterized by sample complexity, which defines the total number of environment interactions required for an agent to learn an ε-optimal policy. In essence, it represents the "data price" an algorithm must pay to achieve a target precision. On-policy and off-policy deep AC methods exhibit complementary advantages and weaknesses, reflecting a fundamental trade-off between theoretical optimality and engineering efficiency. On one hand, Natural Actor–Critic with Data Drop (NAC-DD) [2] achieves a state-of-the-art theoretical sample complexity of $\tilde{\mathcal{O}}(\varepsilon^{-2})$. This efficiency stems from its use of Natural Policy Gradient (NPG), which incorporates second-order geometric information via the Fisher Information Matrix. From a theoretical perspective, NAC-DD possesses a more "advanced" model with stronger data "digestion" capabilities, enabling stable and rapid convergence in terms of update iterations. However, to satisfy the theoretical i.i.d. requirements for this optimal rate, NAC-DD employs a "Data Drop" mechanism that discards the majority of Markovian samples. Consequently, in engineering practice, its on-policy nature and poor sample reuse lead to extremely low wall-clock efficiency. By contrast, Deep Deterministic Policy Gradient (DDPG) [5] and its variants like TD3 [3] typically exhibit a higher theoretical sample complexity of $\tilde{\mathcal{O}}(\varepsilon^{-3})$ due to their reliance on first-order standard gradients. Nevertheless, the off-policy paradigm aggressively reuses samples via a large replay buffer, making DDPG significantly more efficient in practical engineering despite its "looser" theoretical bounds. Yet, this practical efficiency comes at a cost: the replay buffer initially contains "dirty" or uninformative random transitions, and the critic bootstraps from poorly parameterized state–action value approximations. Combined with additional target networks, this often leads to extremely unstable or divergent early learning dynamics. These complementary characteristics naturally motivate a *two-stage* approach: can we exploit NACDD's theoretically stable warm-up to (i) rapidly establish a meaningful action distribution with its superior "digestion" capability and (ii) generate higher-quality off-policy transitions, and then switch to DDPG to capitalize on its long-run sample reuse efficiency? This leads to the IRL-guided Two-Stage NACDD→DDPG framework introduced in this report.

In addition to the difference between on-policy and off-policy, which focuses on the problem of how many times and when we need to generate data, another aspect of data efficiency lies in how much data is needed to sample, which leads us to a trick called "Data Drop" proposed by NAC-DD. To decorrelate state-action-reward pairs, NAC-DD periodically discards samples on the on-policy trajectories. Theoretically, a data drop interval (M) greater than 1 can help models converge faster, yet to what extent this mechanism realizes the intended decorrelation effect in practice, and in different environments, remains to be evaluated. On top of that, given that a larger M makes data episodes longer, i.e, longer time, we propose an Adaptive Data Drop(ADD) to balance the tradeoff of data efficiency and training efficiency.

Therefore, the work of our project is two-fold, with a special focus on improving the data efficiency during agent training: 1) a two-stage warm-up training mechanism; and 2) a practical understanding of the effect of data drop. In summary, our work is as follows:

- Ensure a fair comparison, align the computational budget of both algorithms by equalizing the number of parameter updates (gradient steps) performed per unit of environment interaction, thus better understand sample complexity and its manifested differences during practical training.
- A two-stage AC agent training starting with an on-policy warm-up as an initialization before the off-policy trainin
- A Data Drop sweep for NAC-DD that reveals an environment-dependent trade-off: a clear sweet spot in `LunarLanderContinuous-v2` around $M = 2$–3, and a monotone degradation in `Pendulum-v1`.
- An Adaptive Data Drop rule that ties the drop interval M_t to online TD-error correlation.
- A comparison study of different ADD's design choices (TD-error vs. advantage correlation, exponential smoothing, and hysteresis), leading to practical guidelines: TD-error correlation is a suitable driver for Data Drop; advantage correlation is not; and simple smoothing or hysteresis mainly stabilise M_t without requiring delicate tuning.

2 BACKGROUND AND RELATED WORK

2.1 Actor–Critic and Natural Policy Gradient

We consider discounted Markov decision processes with continuous state and action spaces. Actor–critic methods maintain a parametric policy $\pi_\theta(a \mid s)$ and a critic $V_\omega(s)$ or $Q_\omega(s,a)$. The actor parameters θ are updated by stochastic gradients of a performance objective $J(\theta)$, while the critic parameters ω are fitted by temporal-difference (TD) regression.

Natural policy gradients precondition the vanilla gradient $\nabla_\theta J$ by the inverse Fisher information matrix F^{-1}, yielding a geometry-aware search direction. The natural actor–critic of Konda and Tsitsiklis [4] and its neural extensions [6, 3, 2] combine such natural gradients with critic-based advantage estimates. Recent theory shows that, under suitable conditions and with appropriate step-sizes, natural actor–critic methods can attain near-optimal sample complexity under Markov sampling.

2.2 Order-Optimal Actor–Critic and Data Drop

Ganesh *et al.* introduce NAC-DD, an order-optimal natural actor–critic algorithm that incorporates a *Data Drop* mechanism. Rather than using every on-policy transition, NAC-DD keeps only every M-th sample for the actor update, aiming to reduce temporal dependence while retaining enough data to maintain statistical efficiency. The analysis shows that appropriate choices of M yield improved convergence rates under multi-layer neural parameterisations.

The theory, however, does not prescribe a universal value of M. In practice M becomes another task-dependent hyperparameter, and the empirical behaviour of Data Drop has been explored only on a limited set of environments. Understanding how M interacts with temporal correlation and gradient noise in realistic neural settings is therefore of practical interest.

2.3 Practical Actor–Critic: DDPG and TD3

Deep Deterministic Policy Gradient (DDPG) is a canonical off-policy deterministic actor–critic method, combining a parameterised deterministic policy with a learned Q-function and a replay buffer. By leveraging experience replay and target networks, DDPG enables sample-efficient learning in continuous control, but is known to suffer from critic overestimation and training instability under function approximation.

Twin Delayed DDPG (TD3) [1] is a widely adopted refinement of DDPG that addresses these issues through: (i) twin critics with a minimum target, (ii) delayed policy updates, and (iii) target policy smoothing via clipped noise. These modifications substantially reduce overestimation bias and stabilise critic learning, leading to strong empirical performance on MuJoCo benchmarks.

From the present perspective, DDPG and TD3 represent a largely engineering-driven response to critic instability and correlation, whereas NAC-DD is motivated by tight sample-complexity analysis. A unified implementation enables a direct comparison between these two design philosophies.

3 METHOD

3.1 Variate Controlling: Sample Complexity Understanding

To rigorously compare the *Sample Complexity* of two algorithms, NAC-DD and DDPG, the experimental setup must neutralize variables related to computational load and training frequency. Simply plotting performance against environment steps is insufficient, as DDPG can theoretically reuse data indefinitely, and NAC-DD's single-update cost is significantly higher. Our objective is to establish an equivalence in the total number of gradient steps executed by both algorithms per unit of new environment interaction.

We assume the following default hyperparameters from the codebase for the analysis:

- NAC-DD Policy Collection Steps ($\text{NAC-DD}_{\text{Steps}}$): 4096
- Critic Training Epochs ($\text{Epochs}_{\text{Critic}}$): 5
- Minibatch Size ($\text{Batch}_{\text{Size}}$): 256 (assumed for both)

The NAC-DD algorithm utilizes an on-policy data batch for both Critic and Actor updates. The implementation applies sample subsampling via the drop_num hyperparameter to manage data correlation and memory.

We propose setting the subsampling rate hp.drop_num $= 2$. This halves the effective number of samples used for training.

$$N_{\text{eff}} = \frac{\text{NAC-DD}_{\text{Steps}}}{\text{hp.drop\_num}} = \frac{4096}{2} = 2048 \text{ samples} \tag{1}$$

The Critic network is trained for $\text{Epochs}_{\text{Critic}}$ epochs over the effective sample set. Each epoch consumes the entire dataset in $\text{Batch}_{\text{Size}}$ chunks. The total number of Critic parameter updates (N_{Critic}) is calculated as:

$$N_{\text{Critic}} = \text{Epochs}_{\text{Critic}} \times \left\lfloor \frac{N_{\text{eff}}}{\text{Batch}_{\text{Size}}} \right\rfloor = 40 \text{ updates} \tag{2}$$

The Actor network performs a single, high-cost policy update using the Conjugate Gradient method with Line Search. This process consumes significant computation (e.g., 10 Fisher-Vector Products, FVP, which involve second-order differentiation), but yields only one accepted parameter update. The Critic updates (40 steps) thus represent the dominant measure of training effort for this batch.

On the other hand, in the initial DDPG setup, train$(\cdot)$ is called immediately after every single environment step (time step). To achieve a fair comparison, DDPG must execute a total of $N_{\text{DDPG}} = 40$ parameter updates for every ≈ 4000 new environment steps collected (which approximates one NAC-DD batch period). The required DDPG training frequency ($\text{train}_{\text{freq}}$), defined as the number of new environment steps per single training call, is calculated as:

$$\text{train}_{\text{freq}} = \frac{\text{Environment Steps}}{\text{Target Updates}} = \frac{4000}{40} = 100 \tag{3}$$

3.1.1 Conclusion for comparison

The proposed parameter setup ensures that both the on-policy and off-policy algorithms are compared fairly:

`NAC-DD (hp.drop_num = 2)` Effectively uses 2048 new samples to perform 40 Minibatch Critic updates and 1 Actor update.

`DDPG (train_freq = 100)`: Every 100 new environment samples, it performs 1 training update (using 256 old samples from the buffer), resulting in 40 updates over a ~ 4000 step period.

3.2 Two-Stage Warm-Start: NAC-DD to DDPG with IRL

This method incorporates three major ideas: (1) a NAC-DD warm-up phase producing pretrained actor features and high-quality transitions, (2) a transfer mechanism initializing the DDPG actor from the NAC-DD actor, and (3) an imitation-learning-based reward augmentation (GAIL-style IRL) that is naturally compatible with actor–critic architectures.

We use the NAC-DD implementation from `github.com/LucasCJYSDL/NAC-DD`, and implement DDPG following the design of TD3/DDPG variants described in [**?**]. Additional training logic, buffer management, and IRL components are implemented by us.

3.2.1 Actor Parameter Transfer

A key observation is that NAC-DD and DDPG optimize fundamentally different policy parameterizations. NAC-DD models a stochastic Gaussian policy:

$$\pi_\theta(a \mid s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)),$$

whereas DDPG employs a deterministic policy:

$$a = \pi_\phi(s).$$

Thus the output heads of the two actor networks are *structurally incompatible*. However, their feature extractors—the lower layers that map states to latent representations—share the same functional role: both seek a smooth representation that correlates with action quality. Therefore, we transfer only the shared hidden layers:

$$\phi_{\text{shared}}^{\text{DDPG}} \leftarrow \theta_{\text{shared}}^{\text{NAC-DD}},$$

while reinitializing the DDPG output layer to preserve deterministic action semantics. This partial transfer provides DDPG with a warm-started feature space without imposing mismatched policy distributions.

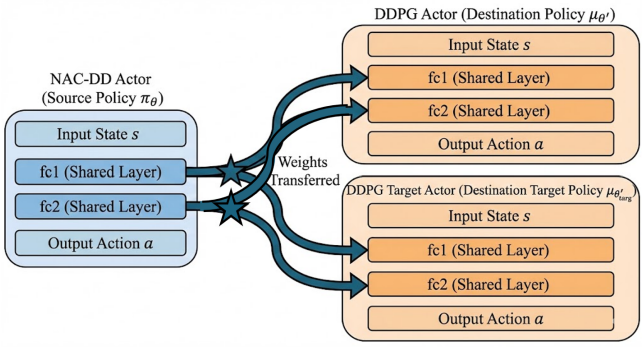


Figure 1: Parameter Transfer

3.2.2 Critic Non-Transferability

The critic architectures are fundamentally different:

- NAC-DD critic approximates a *state-value function*: $V(s)$.
- DDPG critic approximates a *state–action value function*: $Q(s,a)$.

Since these functions differ in both domain and training targets, transferring parameters would violate the functional semantics of the DDPG critic. Therefore:

$$Q^{\text{DDPG}} \text{ must be trained from scratch.}$$

3.2.3 Replay Buffer Warm-Up

In off-policy algorithms such as DDPG, the replay buffer is critical for stable learning. Standard practice fills the buffer with random transitions, which introduces two issues:

(1) many early samples do not reflect meaningful dynamics; (2) the critic bootstraps on poorly distributed samples.

Our two-stage design replaces this random prefill with transitions collected by NAC-DD after warm-up. Unlike random actions, NAC-DD's transitions reflect an already partially optimized policy. The resulting buffer:

$$\mathscr{B}_0 \approx \{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{N_{\text{warmup}}},$$

provides substantially better initialization for the DDPG critic and stabilizes the early learning dynamics.

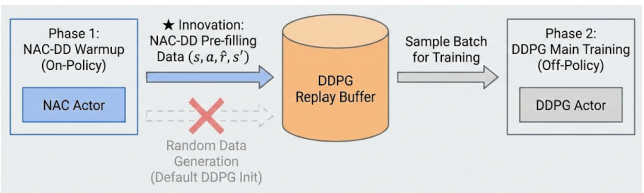


Figure 2: Buffer Preparation

3.2.4 IRL-Based Reward Shaping

We integrate an adversarial inverse reinforcement learning module (GAIL-style discriminator) to compute an auxiliary reward:

$$r_{\text{irl}}(s,a) = -\log(1 - D_\psi(s,a)).$$

This reward improves credit assignment by encouraging trajectories closer to expert demonstrations. Since both NAC-DD and DDPG are actor–critic architectures, the discriminator reward can be seamlessly inserted into their policy gradients without modifying structural assumptions. As validated in our experiments, IRL consistently improves performance across environments, especially in early stages where engineered rewards may be sparse or misleading.

3.2.5 Conclusion for Two Stage Algorithm

We sum up the procedure with pseudocode as follows. A more intuitive illustration is provided in 3

Initialize: Environment $\mathscr{E}$, State Filter $\mathscr{Z}$, Replay Buffer $\mathscr{D} \leftarrow \emptyset$
Initialize NAC-DD Actor π_ϕ and Critic V_ψ
Initialize Reward Network D_ω (GAIL) with expert data $\mathscr{D}_E$
Set warmup steps N_{warm} and total training steps N_{ddpg}

Phase 1: NAC-DD Warmup & Buffer Pre-filling
while $t_{\text{warm}} < N_{\text{warm}}$ **do**
  Collect $\mathscr{T}_{\text{on}} = \{(s,a,s',d)\}$ using π_ϕ (w. $\mathscr{Z}$)
  for each transition (s,a,s',d) in $\mathscr{T}_{\text{on}}$ **do**
    Compute IRL reward: $r \leftarrow D_\omega(s,a)$
    Store (s,a,r,s',d) in $\mathscr{D}$ (DDPG Buffer)
  end for
  Update NAC-DD: $(\phi,\psi) \leftarrow$ NAC-DD_Update$(\mathscr{T}_{\text{on}}, r)$
  $t_{\text{warm}} \leftarrow t_{\text{warm}} + |\mathscr{T}_{\text{on}}|$
end while

Phase 2: DDPG Main Training
Initialize DDPG Actor μ_θ and Critic Q_ξ
Parameter Transfer: $\theta_{\text{shared}} \leftarrow \phi_{\text{shared}}$
Initialize Target Networks $\mu_{\theta'}$ and $Q_{\xi'}$
$t_{\text{total}} \leftarrow 0$
while $t_{\text{total}} < N_{\text{ddpg}}$ **do**
  Collect transition (s,a,s',d) using μ_θ with exploration noise
  Compute IRL reward: $r \leftarrow D_\omega(s,a)$
  Store (s,a,r,s',d) in $\mathscr{D}$
  if Update condition met **then**
    Sample batch $\mathscr{B} \sim \mathscr{D}$
    Update (θ,ξ) using standard DDPG_Update$(\mathscr{B}, r)$
    ▷ Includes Critic/Actor updates and Target Soft Update
  end if
  $t_{\text{total}} \leftarrow t_{\text{total}} + 1$
end while
Output: Optimized Policy μ_θ

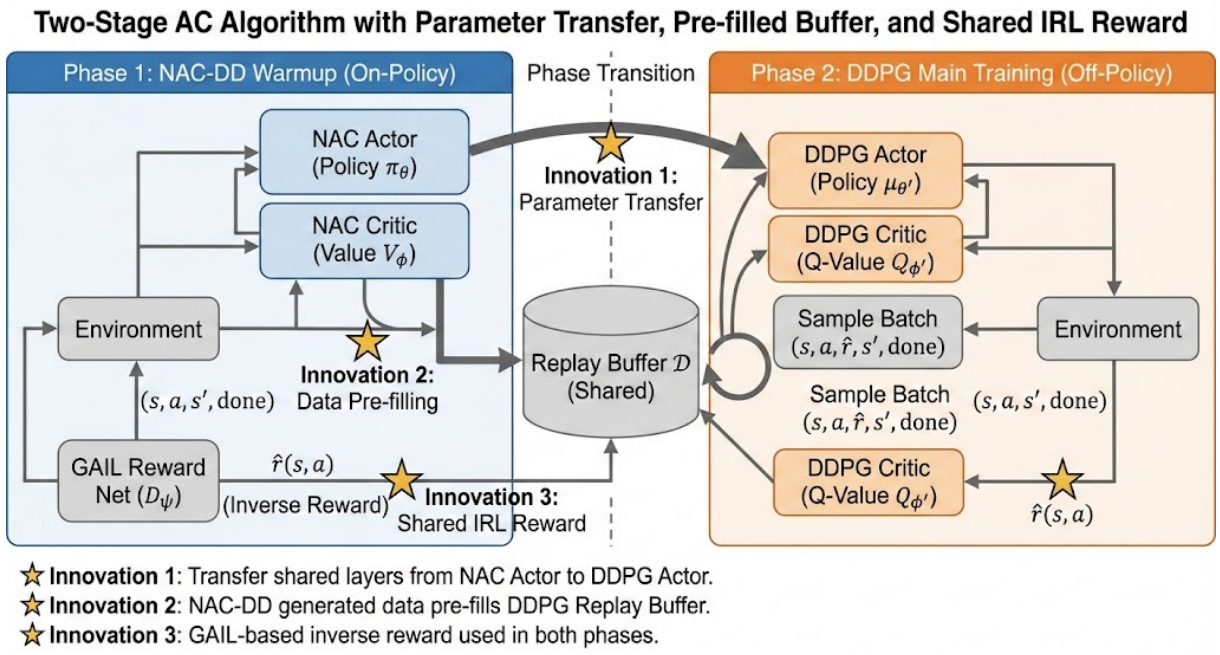
Algorithm 1: Concise IRL-Guided Two-Stage RL Framework

3.3 Data Drop in NAC

The second part of the study focuses on the Data Drop mechanism inside NAC-DD and its interaction with temporal correlation and sample efficiency.

Let $\{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{N_{\text{traj}}}$ denote the on-policy trajectory collected in one outer loop. Given a drop interval $M \in \mathbb{N}$, the critic update uses all transitions, while the actor update only accesses the sub-sequence

$$\{(s_{1+kM}, a_{1+kM}, r_{1+kM}, s_{2+kM})\}_k.$$



(a)

Figure 3: Two Stage Training Procedure

Intuitively, $M > 1$ increases the temporal spacing between the samples used for the policy update, which should reduce the effective Markovian correlation of the gradient estimates.

To quantify temporal dependence more directly, the critic logs the sequence of TD-errors

$$x_t = r_t + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)$$

for the entire on-policy trajectory. From this sequence a sliding window of length L is extracted, and the lag-1 autocorrelation coefficient is estimated as

$$\hat{\rho}_1 = \frac{\sum_{t=1}^{L-1}(x_t - \bar{x})(x_{t+1} - \bar{x})}{\sum_{t=1}^{L}(x_t - \bar{x})^2}, \qquad \bar{x} = \frac{1}{L}\sum_{t=1}^{L} x_t. \tag{4}$$

3.4 Adaptive Data Drop

To reduce manual tuning, an Adaptive Data Drop controller is introduced to adjust M online based on the observed temporal correlation.ADD replaces the fixed drop interval by a simple feedback controller driven by the correlation estimate $\hat{\rho}_1$. At each diagnostic step we map the current correlation estimate to a new drop interval M_{k+1} using two thresholds $0 \le \rho_{\text{low}} < \rho_{\text{high}}$ and bounds $1 \le M_{\min} < M_{\max}$.

The baseline controller uses the magnitude $|\hat{\rho}_1|$:

$$M_{k+1} = \begin{cases} M_{\min}, & |\hat{\rho}_1| \le \rho_{\text{low}}, \\ M_{\max}, & |\hat{\rho}_1| \ge \rho_{\text{high}}, \\ \text{round}\Big(M_{\min} + \frac{|\hat{\rho}_1| - \rho_{\text{low}}}{\rho_{\text{high}} - \rho_{\text{low}}}(M_{\max} - M_{\min})\Big), & \text{otherwise.} \end{cases} \tag{5}$$

When the estimated correlation is small, the controller sets $M_{k+1} = 1$ and retains all transitions for the actor. When $|\hat{\rho}_1|$ is large, it assigns the maximum drop interval $M_{\max}$ to aggressively thin out highly correlated samples. Intermediate values of $|\hat{\rho}_1|$ lead to intermediate drop intervals via linear interpolation.

Within each outer loop, NAC-DD collects trajectories using the current interval M_k, updates critic and actor exactly as in the fixed-M case, and only afterwards updates M using the controller. ADD therefore introduces no additional optimiser-related hyperparameters and modifies only the sampling pattern.

To isolate which design choices in ADD are essential, four variants of the controller are considered:

- **Baseline (TD with linear mapping)**: the controller uses TD-error correlation and the linear mapping above.
- **Advantage correlation**: identical to Baseline, but uses the advantage sequence instead of TD-errors when forming $\hat{\rho}_1$.
- **EMA smoothing**: applies exponential moving-average smoothing to the correlation estimates before mapping: $\tilde{\rho}_k = \alpha\hat{\rho}_k + (1-\alpha)\tilde{\rho}_{k-1}$
- **Hysteresis mapping**: uses TD-errors but replaces the linear mapping by a discrete hysteresis rule: if $|\hat{\rho}_1| > \rho_{\text{high}}$ then $M_{k+1} = \min(M_k + 1, M_{\max})$; if $|\hat{\rho}_1| < \rho_{\text{low}}$ then $M_{k+1} = \max(M_k - 1, M_{\min})$; otherwise $M_{k+1} = M_k$.

All four variants share the same NAC-DD backbone, optimiser settings, and evaluation protocol, and differ only in how the drop interval M is adapted. This allows performance differences in the experiments to be attributed directly to the design of the ADD controller.

4 RESULTS AND ANALYSIS

4.1 Experiment Setup

4.1.1 Environments

Our experiments are conducted on continuous-control benchmarks provided by Gymnasium and MuJoCo. We focus on two representative tasks with different state-action dimensions and reward scales:

- **Pendulum-v1**: A classic low-dimensional control task with a 3-dimensional state space (angle sine, angle cosine, angular velocity) and a 1-dimensional continuous action (torque). The reward at each step is a quadratic penalty on angle deviation and control effort. We define the success threshold as an average return of -200.
- **Hopper-v5**: A higher-dimensional locomotion task from the MuJoCo suite with an 11-dimensional state space (body positions, velocities, and joint angles) and a 3-dimensional continuous action (joint torques). The agent receives a reward combining forward velocity, alive bonus, and control cost. We define the success threshold as an average return of 3000.

Each episode is truncated at a maximum horizon of $H = 1000$ steps.

4.1.2 Network Architecture

Unless otherwise stated, actors and critics use the same two-layer MLP architecture as in all methods with the following architecture:

- **Hidden layers**: Two fully-connected layers, each with 64 hidden units.
- **Activation**: Hyperbolic tangent (`tanh`) is applied after each hidden layer.
- **Actor output**: A linear layer outputs the mean $\mu(s)$ of a Gaussian policy. The log standard deviation $\log \sigma$ is parameterized as a separate learnable vector (state-independent), initialized to 0 so that $\sigma = 1$ at the start of training.
- **Critic output**: A single linear unit outputs the scalar state-value estimate $V(s)$.

All network weights are initialized using Xavier uniform initialization, and biases are initialized to zero.

4.1.3 Hyperparameters

The hyperparameters are kept consistent across all experiments unless otherwise noted. Key training hyperparameters are summarized in Table 1.

Table 1: Training hyperparameters used in all experiments.

Hyperparameter	Value
Discount factor γ	0.99
Actor learning rate α_π	3×10^{-4}
Critic learning rate α_V	1×10^{-3}
Batch size (before dropping)	2048
Maximum training steps	2×10^7
Episode horizon H	1000
Evaluation episodes	10
Logging interval (iterations)	5

For the Adaptive Data Drop (ADD) controller, we use the following default parameters:

- Correlation window size: $W = 4096$ samples.
- Minimum samples required before updating M: 512.
- Correlation thresholds: $\rho_{\text{low}} = 0.1$, $\rho_{\text{high}} = 0.5$.
- Drop number range: $M_{\min} = 1$, $M_{\max} = 5$.
- EMA smoothing coefficient (for EMA variant): $\alpha = 0.1$.
- Hysteresis step size (for Hyst variant): $\Delta M = 1$.

4.2 Sample Complexity Comparison

This experiment is corresponding to discussion in Section 3.1. The configuration standardizes the total number of gradient descent operations (40 updates) per collection epoch (≈ 4000 steps), making the resulting Sample Complexity vs. Reward curves a valid measure of each algorithm's inherent sample efficiency.

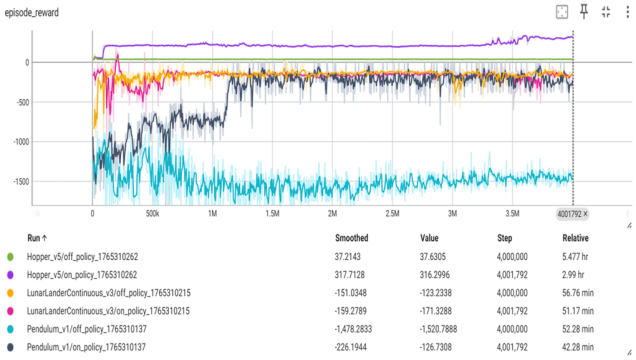


Figure 4: On-/off-policy comparison in different environments

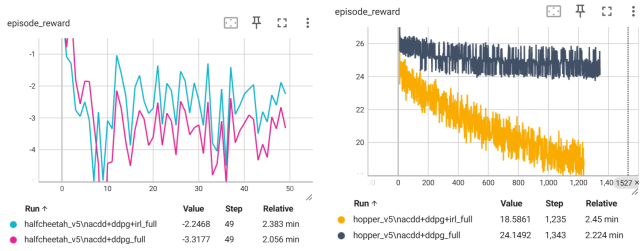
As illustrated in 4, after aligning the computational budgets, NAC-DD significantly outperforms DDPG in terms of reward across all environments given the same number of interaction steps.

4.3 Two-Stage Implementation

We evaluate the method on `Hopper-v5` and `HalfCheetah-v5`. Each experiment uses identical random seeds for fair comparison. The warm-up budget is fixed at 60000 NACDD steps, followed by 40000 DDPG steps. Since our focus is warm-start effectiveness, we intentionally keep DDPG steps moderate.

4.3.1 IRL Effectiveness Check

We first compare using the environment reward r versus the IRL reward r_{irl}. Training always uses one of the rewards exclusively, but evaluation rollouts report environment reward for consistency. In both environments, IRL improves evaluation performance, confirming that integrating imitation-guided reward shaping is beneficial for this two-stage framework. All subsequent experiments therefore incorporate IRL.



(a) `HalfCheetah-v5` (b) `Hopper-v5`

Figure 5: IRL Effectiveness Check

4.3.2 Parameter Transfer vs. Buffer Warm-Up

To isolate effects, we design controlled conditions:

1. **No warm-up (baseline).**
2. **Actor-parameter transfer-only warm up.**

3. **Buffer preparation-only warm up.**

4. **Combined parameter transfer + buffer preparation warm-up.**

For fairness, even when DDPG receives no transfer, we still execute NACDD warm-up but do not use its outputs. This ensures identical environmental randomness before the DDPG phase.

Actor-only transfer. In `HalfCheetah-v5`, actor transfer yields negligible improvement; in `Hopper-v5`, slight gains appear but remain modest. See (a) and (b) in 6.

Buffer-only warm-up. In sharp contrast, buffer transfer significantly boosts performance in both environments. This confirms that high-quality early transitions are crucial for stabilizing the DDPG critic. See (c) and (d) in 6.

Combined transfer. When applying both transfer mechanisms, `HalfCheetah-v5` demonstrates clear synergy and obtains the best reward curves. `Hopper-v5` shows only mild improvements, but still outperforms baseline in most cases. See 7 and 8.

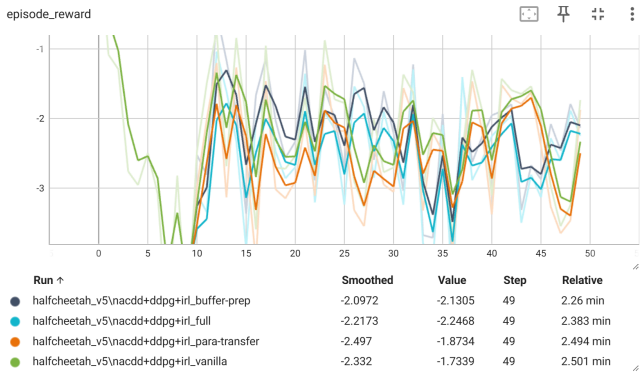


Figure 7: Comprehensive Comprison in `env HalfCheetah-v5`

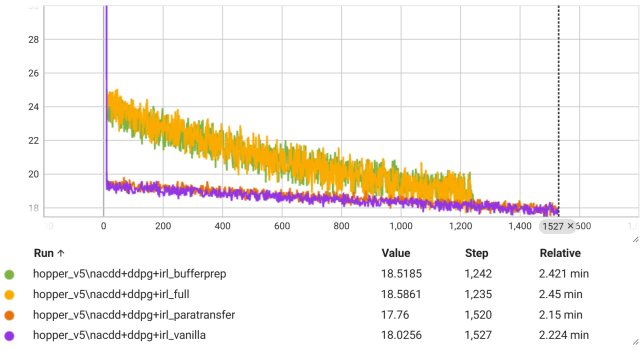


Figure 8: Comprehensive Comprison in `env Hopper-v5`

4.3.3 Discussion for Two-Stage Training

A striking empirical finding is that *buffer warm-up contributes substantially more than actor parameter transfer*. We provide several explanations:

1. Critic-dominated early dynamics. DDPG training is highly critic-driven: poor critic estimates directly distort policy gradients. Since the critic is reinitialized from scratch, transferred actor features offer limited benefit until the critic becomes accurate. In contrast, high-quality replay transitions allow the critic to learn meaningful Q-estimates from the first updates, dramatically improving stability.

2. On-/off-policy distribution mismatch. NACDD warm-up produces on-policy transitions concentrated around the NACDD behavior policy. These transitions define a much more realistic state–action visitation distribution than random exploration. DDPG, being off-policy, benefits directly from this shift in data distribution; random-prefilled buffers induce a strong initial mismatch that destabilizes bootstrapping.

3. Deterministic vs. stochastic policy mismatch. Transferred actor layers come from a stochastic NACDD policy whose geometry may not align well with deterministic DDPG updates. Even if the latent representation is helpful, the mismatch in output semantics can limit effective transfer.

4. Buffer improves both critic and policy, actor transfer only affects policy. Warm-up buffer contributes to two learning channels:

$$\mathscr{B}_0 \longrightarrow Q\text{-learning} \longrightarrow \pi\text{-update}.$$

Actor transfer affects only the forward policy network. Consequently, buffer transfer has a multiplicative effect on learning dynamics, while actor transfer provides only additive improvement.

Conclusion. Across environments, the evidence consistently supports that NACDD warm-up—primarily through improved buffer initialization, and secondarily through feature transfer—provides a stable and efficient mechanism for mitigating early-stage off-policy instability. This demonstrates that combining theoretically stable on-policy methods with sample-efficient off-policy RL is a promising direction for future hybrid RL research.

4.4 Analysis of the Data Drop Mechanism

We next turn to the Data Drop mechanism inside NAC-DD and study how the choice of the drop interval M affects temporal correlation and sample efficiency. On Pendulum-v1, Hopper-v5, and HalfCheetah-v5 we run NAC-DD with fixed $M \in \{1,2,3,8\}$. For each configuration and environment, we record the mean lag-1 TD-error autocorrelation $\bar{\rho}_1$, learning curves versus outer-loop iterations, and learning curves versus environment steps.

4.4.1 TD-error autocorrelation vs. drop number

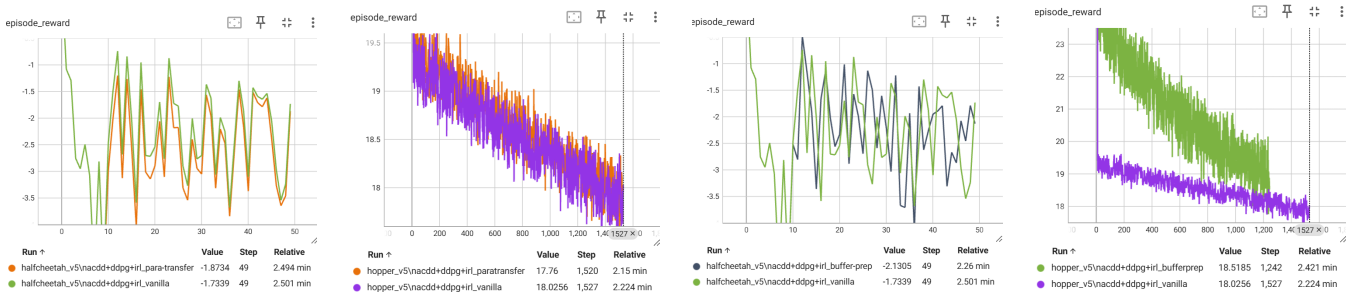
Figure 9 summarises how the mean lag-1 TD-error autocorrelation $\bar{\rho}_1$ varies with the drop interval M. On Hopper-v5, $\bar{\rho}_1$ is strongly negative at $M = 1$, and its magnitude decreases as M increases; by $M = 8$ the TD-errors are approximately uncorrelated. Pendulum-v1 follows a similar overall trend: increasing M reduces $\bar{\rho}_1$ towards zero, and the TD-errors remain close to uncorrelated at larger drop intervals. Overall, larger M progressively attenuates temporal dependence in the TD-error signal across tasks.

4.4.2 Convergence vs. iterations

Figures 10 plot the learning curves for Hopper-v5 and Pendulum-v1 from two complementary viewpoints: average return versus outer-loop iteration (left), and versus environment steps (right). Understanding the distinction between these two axes is key to interpreting the effect of Data Drop.

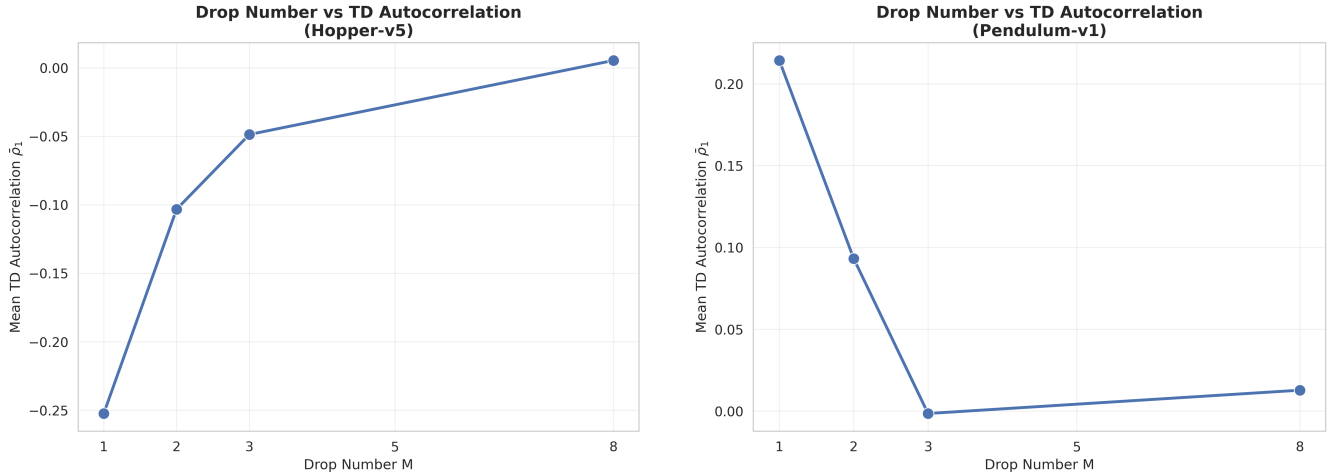
By construction, each outer-loop iteration in NAC-DD performs one critic update and one actor update using mini-batches of fixed size. The only quantity that changes with the drop interval M is how many raw transitions must be collected to assemble those batches: larger M means that more on-policy steps are rolled out and then subsampled at stride M. Consequently, plots against *iterations* compare different choices of M under a matched optimisation budget: after k iterations, all configurations have taken the same number of gradient steps and processed the same number of training samples, but those samples differ in their temporal spacing.

In contrast, plots against *environment steps* match the amount of interaction with the environment. For a fixed step budget, small-M runs can perform many more outer-loop updates than large-M runs,



(a) `HalfCheetah-v5` (b) `Hopper-v5` (c) `HalfCheetah-v5` (d) `Hopper-v5`

Figure 6: Controlled Conditions for Two-Stage Training



(a) Hopper-v5 (b) Pendulum-v1

Figure 9: Mean lag-1 TD-error autocorrelation $\bar{\rho}_1$ versus drop interval M.

simply because each update requires fewer fresh transitions. Thus, curves on the step axis entangle two effects: (i) how efficiently an algorithm uses the samples it has collected, and (ii) how many gradient updates it has been able to afford within the same interaction budget. From this perspective, the step axis primarily reflects *sample efficiency*, whereas the iteration axis isolates *per-update optimisation efficiency*.

On Hopper-v5, the iteration-wise curves show that $M = 1$, $M = 3$, and $M = 5$ all make comparable progress per update: they cross the performance threshold within roughly 80–110 iterations, with $M = 1$ and $M = 5$ slightly ahead and $M = 3$ close behind. In contrast, $M = 2$ exhibits slower progress over many iterations before a late improvement, and $M = 8$ also shows noticeably slower convergence per iteration. However, when the same runs are viewed on the step axis, $M = 1$ is the most sample-efficient configuration: it reaches the threshold after the fewest environment interactions, followed by $M = 3$. Settings with larger M consume substantially more steps before achieving similar returns, and $M = 8$ remains far below the threshold for most of training.

Pendulum-v1 shows a consistent pattern. When measured per iteration, $M = 2$ converges the fastest, with $M = 5$ close behind; $M = 1$ and $M = 3$ require a few additional iterations, and $M = 8$ reaches the threshold later than the best settings. On the step axis, however, $M = 1$ and $M = 2$ become the most sample-efficient choices, while $M = 3$ and $M = 5$ need more environment interactions to reach the same return.

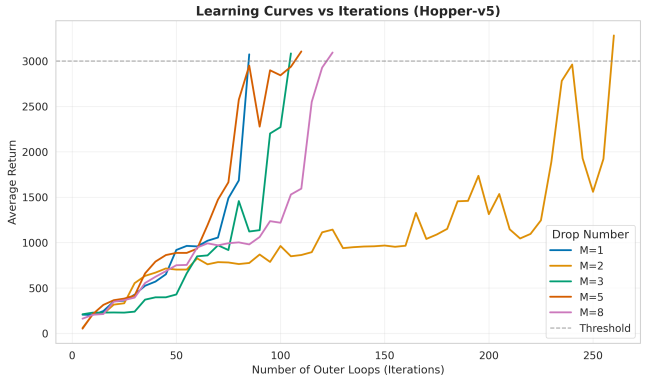
Overall, these results lead to two conclusions. First, Data Drop has a real algorithmic effect: for a fixed number of gradient updates, moderate drop intervals ($M \approx 2$–5) can yield smoother and sometimes faster convergence than using every sample ($M = 1$), especially in the early phases of training. Second, this benefit comes with a cost in interaction budget: very large M values are consistently sample-inefficient, and even moderate M trades some sample efficiency for improved decorrelation. In the remainder of this work we therefore focus primarily on iteration-wise convergence—which directly reflects the optimisation behaviour of NAC-DD and ADD—while reporting step-based metrics as a complementary view of the sample-efficiency trade-offs.

4.5 Adaptive Data Drop

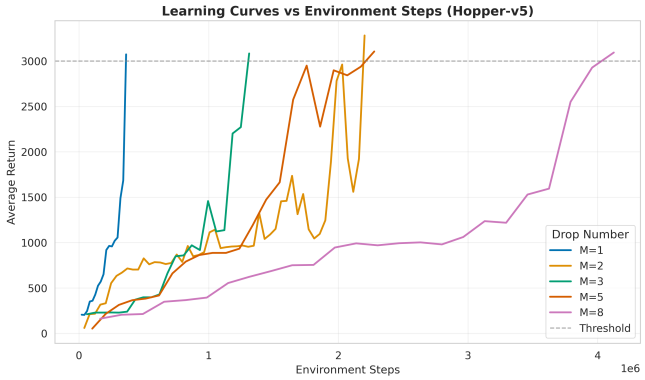
The fixed-M experiments suggest that choosing M is crucial yet task-dependent. We now evaluate Adaptive Data Drop, which adjusts M online based on the TD-error correlation, and study both its overall performance and the effect of its internal design choices.

4.5.1 ADD dynamics and performance

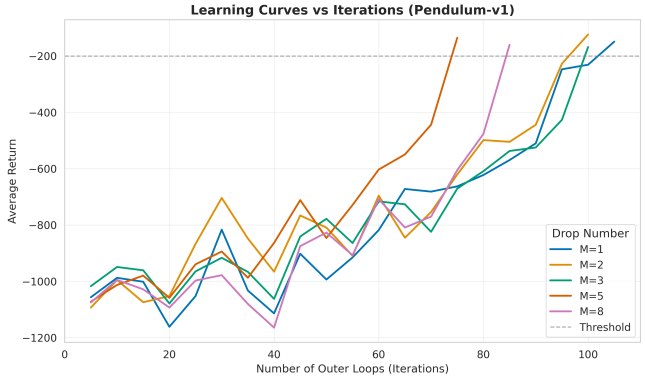
Figure 11 shows the ADD dynamics on Hopper and Pendulum. On Hopper, the TD-error correlation stays consistently negative and varies within a moderate range throughout training. The ADD controller therefore keeps M in the small-to-moderate regime ($M \in$



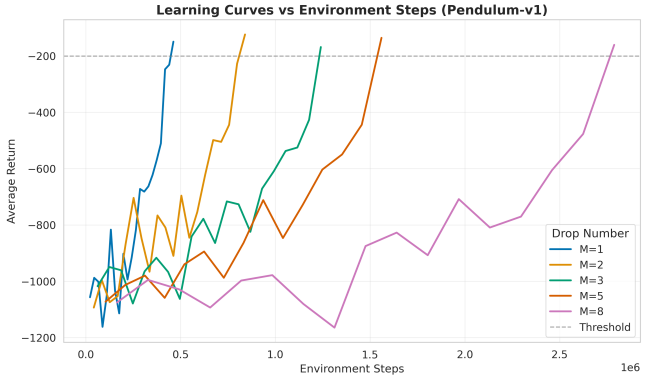
(a) Hopper-v5: return vs. iterations



(b) Hopper-v5: return vs. env steps

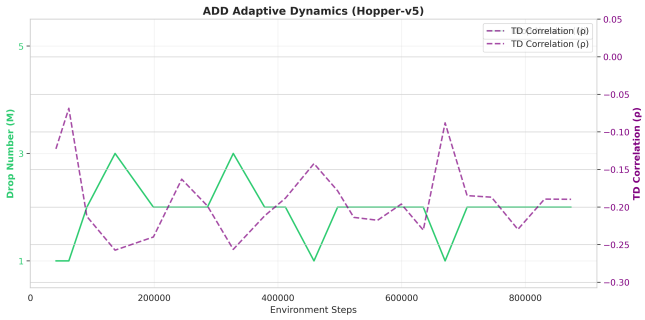


(c) Pendulum-v1: return vs. iterations

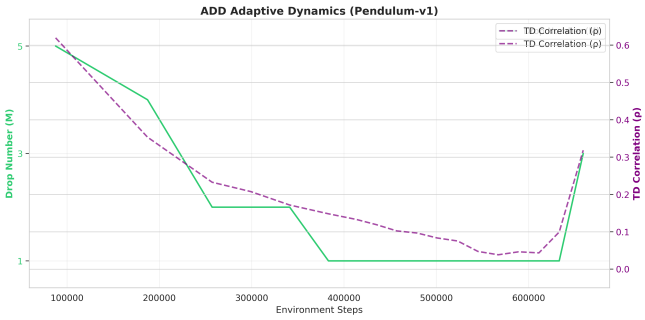


(d) Pendulum-v1: return vs. env steps

Figure 10: Learning curves from two environments: average return versus outer-loop iterations (left column) and versus environment steps (right column), shown for Hopper-v5 (top row) and Pendulum-v1 (bottom row).



(a) Hopper-v5

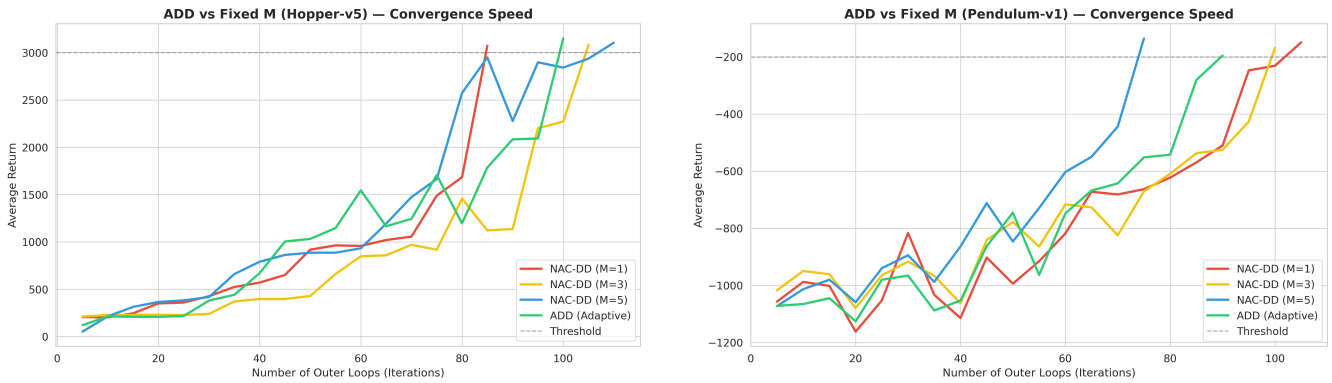


(b) Pendulum-v1

Figure 11: ADD dynamics: drop number M (green, left axis) tracks TD autocorrelation (purple, right axis).

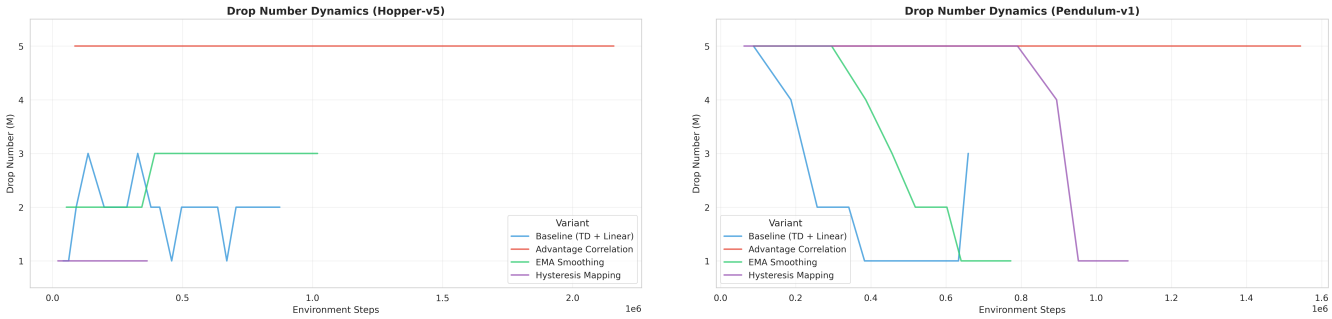
$\{1,2,3\}$), switching between these values as $|\hat{\rho}_1|$ moves around the correlation thresholds. Overall, the adaptation is stable and the resulting M range aligns with the "good" region identified by the fixed-M sweep.

On Pendulum, the TD-error correlation starts high and decreases steadily as the policy improves. Accordingly, ADD initially selects a large drop interval close to $M_{\max} = 5$ to counteract strong temporal dependence, then reduces M in stages towards 1 as $\hat{\rho}_1$ approaches zero. Near the end of training, a brief increase in the estimated correlation is accompanied by a short-term increase of M, indicating that the controller continues to respond to correlation changes rather than remaining fixed once M reaches its minimum. Performance-wise, Figure 12 reports learning curves *versus outer-loop iterations*, i.e., under a matched optimisation budget where each configuration executes the same number of critic/actor updates per iteration. Under this iteration-wise view, ADD achieves convergence comparable to the best fixed-M NAC-DD configuration on both environments: it reaches the task-specific return threshold within a similar number of outer loops as the best-tuned fixed-M setting and substantially faster than clearly suboptimal choices.



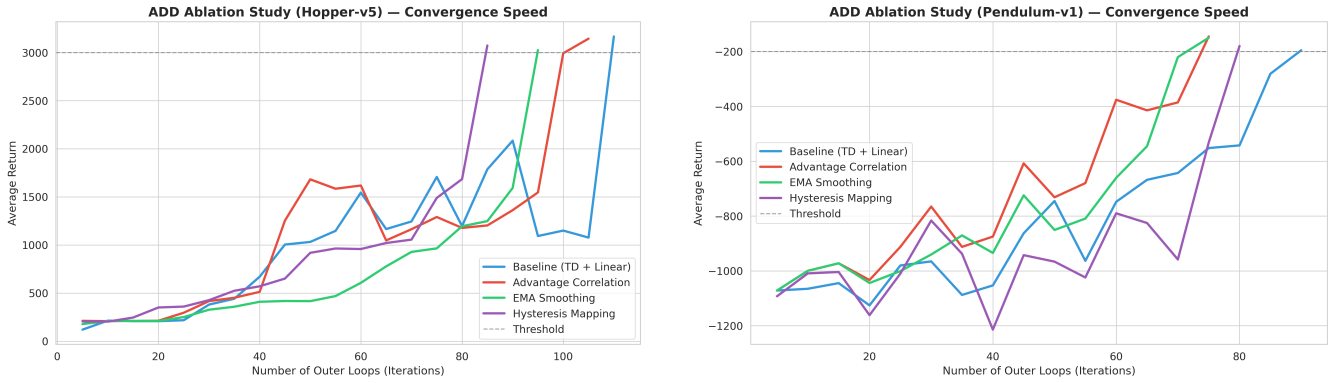
(a) Hopper-v5 (b) Pendulum-v1

Figure 12: ADD achieves convergence comparable to the best fixed-M NAC-DD configuration.



(a) Hopper-v5 (b) Pendulum-v1

Figure 13: Drop-number dynamics for ADD variants on Hopper-v5 and Pendulum-v1.



(a) Hopper-v5 (b) Pendulum-v1

Figure 14: Convergence versus outer-loop iterations for ADD variants on Hopper-v5 and Pendulum-v1.

4.5.2 Comparison of ADD design choices

Finally, we compare the four ADD variants defined in the Methods section: Baseline (TD with linear mapping), Advantage correlation, EMA smoothing, and Hysteresis mapping (Figures 13 and 14).

On Hopper, the TD-based variants exhibit distinct adaptation behaviours. Baseline and EMA keep M in a small-to-moderate range (roughly 1–3) and ultimately reach similar final performance. Baseline and EMA show relatively smooth adjustments of M; in terms of iteration-wise convergence, EMA reaches the threshold earlier than Baseline. Hysteresis, in contrast, quickly drives M down to 1 and keeps it almost constant thereafter, leading to behaviour close to a fixed small-M scheme; correspondingly, it achieves the fastest convergence among the four variants in this setting. The Advantage-correlation variant often saturates at $M = 5$ for extended periods; its learning curve is more variable than the TD-based variants and, on Hopper, it converges later than Hysteresis/EMA but remains com-

petitive with Baseline, suggesting that advantage-based correlation can be a less reliable signal for producing a well-adapted M schedule.

On Pendulum, Baseline and EMA again provide a balanced trade-off in how they adapt M: both reduce M relatively quickly once the estimated correlation falls, which is consistent with faster late-stage improvement. Hysteresis eventually achieves similar returns but requires more iterations before decreasing M, reflecting its more conservative adjustment rule. The Advantage-correlation variant maintains M close to $M_{\max} = 5$ for most of training, yet still achieves near-best iteration-wise convergence on Pendulum, indicating that good return can sometimes be obtained even when the controller does not strongly adapt M.

Overall, the results indicate that: (i) TD-error correlation provides a more consistent driver than advantage correlation for inducing meaningful online adaptation of M in our settings, and (ii) simple linear or EMA-smoothed mappings already capture most of the benefit; more conservative rules such as strong hysteresis can reduce adaptation responsiveness and yield environment-dependent gains.

5 LIMITATION

Two-stage framework introduces additional hyperparameters (e.g., warm-up length and transfer timing) whose optimal choices are environment-dependent. The theoretical guarantees associated with NAC-DD do not directly extend to the subsequent off-policy DDPG phase, so the overall method lacks a unified end-to-end convergence guarantee. Empirically, the performance gains are not uniform across environments, and in some tasks the two-stage strategy provides only marginal improvements over standard off-policy baselines. Transferring actor parameters across algorithms may introduce a representation mismatch, which could limit performance in tasks where the optimal policy structure differs substantially between the two phases.

As for the evaluation of ADD, one hypothetical advantage lies in its ability to generalize and adapt across multiple environments. Our experiments have revealed that an optimal fixed-M value is different for the various environments, as some are sensitive to the correlation between samples, while others are not. As ADD dynamically changes M based on the TD-error correlation, we don't need to rely on grid search for the hyperparameter. As mentioned by the professor during our poster session, one helpful metric will be the average training efficiency across different environments using a fixed M, compared to using ADD. Because of the limited time in the final season, this experiment has not yet been conducted to validate the effect of using ADD.

While the two aspects of our project aimed to tackle the data efficiency problem during AC agent training in different dimensions, we didn't have the chance to combine them, again due to the time budget. A series of experiments should be conducted to better evaluate our new training schema of the combination of on-policy warm-up and ADD.

6 WORK DISTRIBUTION

Our work is distributed as follows:

- Yifan Wu: implementation of ADD and running experiment
- Xiuqi Han: sample complexity comparison, implementation of the two-stage training and running experiment
- Chenhao Zhang: running experiments
- Fengze Zhang: running experiments

All team members participate in making the poster and report.

7 CONCLUSION

This work investigates data efficiency in actor–critic reinforcement learning. We propose a two-stage training framework that uses NAC-DD as a stable on-policy warm-up, followed by off-policy optimization with DDPG. Empirical results demonstrate that NAC-DD–generated transitions significantly improve early off-policy stability, with replay-buffer warm-up contributing more than direct actor parameter transfer. In addition, we provide a systematic analysis of the Data Drop mechanism in NAC-DD and introduce an Adaptive Data Drop strategy based on online TD-error correlation. Our experiments show that Adaptive Data Drop achieves performance comparable to the best fixed drop intervals without manual tuning, highlighting the importance of correlation-aware sampling. Overall, this study illustrates that combining theoretically grounded on-policy methods with sample-efficient off-policy learning is a promising direction for improving data efficiency in practical actor–critic systems.

ACKNOWLEDGMENTS

REFERENCES

[1] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018. Twin Delayed DDPG (TD3).

[2] S. Ganesh, J. Chen, W. Mondal, and V. Aggarwal. Order-optimal global convergence for actor-critic with general policy and neural critic parametrization, 2024. NAC-DD preprint.

[3] M. Gaur, A. S. Bedi, D. Wang, and V. Aggarwal. Closing the gap: Achieving global convergence (last iterate) of actor-critic under markovian sampling with neural network parametrization. In *International Conference on Machine Learning (ICML), PMLR*, 2024.

[4] V. Konda and J. Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, eds., *Advances in Neural Information Processing Systems*, vol. 12. MIT Press, 1999.

[5] 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, 2019.

[6] H. Tian, I. C. Paschalidis, and A. Olshevsky. Convergence of actor-critic methods with multi-layer neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.