Iterative Latent Refinement for Value Learning in Offline Goal-Conditioned RL

Daoxin Li[1,3*] **Songcheng Xu**[1,2*] **Guozhang Chen**[1,2†]

[1]State Key Laboratory of Multimedia Information Processing,
School of Computer Science, Peking University, Beijing, China
[2]National Key Laboratory for Multimedia Information Processing,
School of Computer Science, Peking University
[3]National University of Singapore

dawson.li@u.nus.edu xusongcheng26@stu.pku.edu.cn guozhang.chen@pku.edu.cn

Abstract

Offline goal-conditioned reinforcement learning (GCRL) enables goal reaching from static datasets, but the learned value, critic, or compatibility score must support both long horizon reachability inference and offline policy extraction. We study iterative latent refinement as an architectural replacement for the feedforward value or critic backbone used by existing offline GCRL algorithms. Instead of emitting a score after a single feedforward computation, the network repeatedly updates a latent representation using shared recurrent update weights and lightweight step specific parameters before exposing the final actor facing signal. On the evaluated OGBench datasets, iterative refinement variants improve performance across several algorithms without changing their losses, data, actors configuration. Across the 21 algorithm-dataset comparisons, the mean improvement is 10.5 percentage points, with the largest gains on stitching and bottleneck maze tasks. CRL diagnostics show that critics with latent refinement change normalized separation between matched and mismatched goals and produce actors that remain closer to dataset actions under several behavior support proxies. Depth and update capacity studies further suggest that the effect depends on how computation is allocated between refinement depth and per-step expressivity.

1 Introduction

Goal-conditioned reinforcement learning aims to learn policies that can solve many goals from shared experience [42, 2, 15]. In the offline setting, the learner must infer goal reaching behavior from a fixed dataset, without online interaction to correct value errors, collect missing transitions, or recover from out of distribution policy updates [23, 22, 13]. OGBench highlights these difficulties through tasks that probe trajectory stitching, long horizon reasoning, state-goal generalization, and policy extraction from offline data [37].

Prior offline GCRL methods address different parts of this problem. Contrastive and quasimetric methods learn reachability or compatibility signals [12, 54, 21]; supervised, advantage-weighted, and bootstrapped methods emphasize policy extraction from fixed data [57, 28, 53, 58]; and hierarchical or horizon reduction methods use subgoals, temporal abstraction, or multi-step targets to reduce long horizon difficulty [35, 1, 38]. These directions also point to different bottlenecks: faraway goal values can be hard to estimate [35], high level advantage estimates can have incorrect signs in long horizon

*Equal contribution.
†Corresponding author.
Source code: github.com/Criticality-Cognitive-Computation-Lab/refinement-rl.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

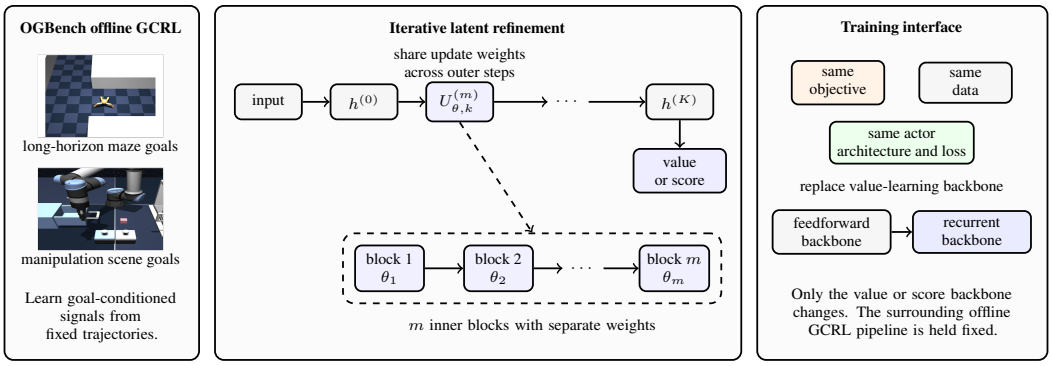


Figure 1: Overview of iterative latent refinement for offline GCRL. Left: OGBench task images adapted from Park et al. [37]. Middle: each outer step $U_{\theta,k}^{(m)}$ refines the latent state using m inner blocks with separate parameters $\theta_1, \ldots, \theta_m$. Their update weights are shared across the K outer steps. Right: the value or critic backbone changes while the dataset, training objective, and actor architecture and loss are held fixed.

regimes [1], cross trajectory stitching can overestimate unconnected state-goal pairs [21], and offline RL performance can be limited by value learning, policy extraction or generalization [36].

We focus on the learned signal that connects static trajectory evidence to policy improvement. Across offline GCRL algorithms, a value, critic, or compatibility network must encode long horizon reachability, avoid unsupported optimism, and expose a signal usable by the chosen actor extraction procedure [37, 36]. In CRL this interface is a contrastive score consumed by a DDPG+BC-style actor [12, 13]; in HIQL it is a scalar value used for hierarchical policy learning [35]; and in SAW it enters a subgoal-weighted actor update [58]. We ask whether changing the architecture of this learned signal, while leaving the surrounding algorithm fixed, can make the policy extraction interface more useful.

Standard implementations of these methods commonly use feedforward MLP backbones for the value, critic, or compatibility network [12, 35, 37, 58]. At the same time, architecture and compute allocation have become important levers in RL and representation learning, with depth, residual structure, regularization, and explicit compute scaling improving several settings [19, 45, 25, 31, 26, 52]. We therefore study iterative latent refinement (Figure 1): instead of producing a score in one feedforward pass, the network repeatedly revises a latent representation before exposing the final actor facing signal. This follows models that allocate computation through repeated latent updates or implicit depth [8, 17, 4, 34, 9, 51, 20, 14]. Section 4 gives the detailed design, including SwiGLU updates, step embeddings, LayerNorm, and LayerScale [44, 8, 3, 49]. We use this module as an architectural replacement for the MLP backbone in CRL, HIQL, and SAW, keeping losses, data, actors, and optimization unchanged. Since CRL exposes its compatibility score directly to the actor [12], we use it as the diagnostic case for measuring score separation and behavior support proxies.

Our contributions are threefold. **First**, we introduce iterative latent refinement as a value network architecture for offline GCRL and instantiate it in different algorithms without changing their objectives or actors. **Second**, we report empirical gains on a subset of OGBench locomotion and manipulation tasks, with large improvements on stitching and bottleneck mazes. **Third**, we characterize changes in the actor facing score and study how performance depends on refinement depth, per-step update capacity, and architectural components.

2 Related Work

Goal-conditioned reinforcement learning. Goal-conditioned RL learns policies and value functions that generalize across desired outcomes, building on goal indexed value approximation, temporal abstraction, hindsight relabeling, and supervised goal reaching [42, 47, 7, 2, 15]. Subsequent methods learn goal-conditioned distances, future state classifiers, latent goal spaces, replay buffer plans, or contrastive scores for long horizon and visual control [40, 30, 10, 41, 11, 12]. These approaches establish goal-conditioned values and reachability estimates as useful control signals. Offline GCRL adds the constraint that these signals and the resulting policies must be learned from a fixed dataset, without further environment interaction during training.

Offline goal-conditioned RL algorithms. Offline RL must learn from fixed data while controlling errors on out of distribution actions, motivating CQL, IQL, and TD3+BC [23, 22, 13]. Offline GCRL additionally requires generalization across state–goal pairs and stitching across trajectories. Supervised and advantage-weighted approaches include goal-conditioned behavioral cloning (GCBC), WGCSL, GoFAR, and DAWOG [37, 57, 28, 53]. Goal-conditioned implicit V-learning (GCIVL) and implicit Q-learning (GCIQL) learn bootstrapped values for policy extraction [37], while SAW bootstraps a flat policy from an auxiliary subgoal-conditioned policy [58]. CRL learns contrastive compatibility scores, QRL learns quasimetric distances, and CGCIVL penalizes values for unconnected state–goal pairs [12, 54, 21]. Hierarchical and multi-step methods, including HIQL, OTA, and SHARSA, reduce the effective horizon through subgoals or temporally extended targets [35, 1, 38]. OGBench evaluates these challenges across diverse datasets [37], and recent analysis identifies value learning, policy extraction, and generalization as distinct potential bottlenecks [36].

Complementary methods change planning or policy extraction. Test-Time Graph Search (TTGS) plans over dataset states using a frozen goal-conditioned policy [33]; Negative Energy as Reward (NEaR) optimizes trajectories under a learned energy model [50]. MeanFlowQL learns a one-step generative policy, while Flow Latent Policy (FLP) optimizes a behavior-regularized policy in a flow model's latent space [55, 56].

Architecture and iterative computation. Architecture and regularization affect feature quality and control performance [45, 25, 24, 32, 46]. Bro and SimBa scale regularized actor–critic networks [31, 26], while SimBaV2 combines hyperspherical normalization with distributional value estimation and reward scaling [27]. Wang et al. [52] scale residual depth in self-supervised goal-conditioned RL, reporting strong online results but limited consistent gains in preliminary offline CRL experiments. Adaptive computation, Universal Transformers, deep equilibrium models, and recurrent reasoning networks provide broader precedents for repeated latent processing [17, 8, 4, 34, 43, 9, 51, 20, 14]. Within RL, Value Iteration Networks embed differentiable planning computation [48]; Deep Repeated ConvLSTM agents learn planning-like behavior in Sokoban [18], with subsequent interventions identifying causally relevant internal plans [5]. Closely related, Ghugare et al. [16] study how shared recurrent computation affects online and offline RL. We study step-conditioned SwiGLU refinement in the value and critic backbones of offline GCRL, focusing on goal discrimination and policy extraction while retaining the underlying objectives and actors.

3 Preliminaries

Offline goal-conditioned RL We consider an offline goal-conditioned Markov decision process with state space $\mathcal{S}$, action space $\mathcal{A}$, transition kernel P, and goal space $\mathcal{G}$. The learner is given a fixed dataset of trajectories $\mathcal{D} = \{\tau^{(n)}\}_{n=1}^{N}$, with $\tau = (s_0, a_0, s_1, \ldots, s_T)$, and receives no environment interaction during training. Given an initial state and goal g, the objective is to learn a goal-conditioned policy $\pi(a \mid s, g)$ that maximizes the discounted goal-reaching return $J(\pi) = \mathbb{E}_{g\sim p(g),\, \tau\sim p^{\pi}(\tau|g)}\left[\sum_{t=0}^{\infty} \gamma^t r_g(s_t, a_t, s_{t+1})\right]$, where r_g is the sparse goal-reaching reward or its environment-specific analogue. Training goals are relabeled using trajectory states or other dataset states, according to the underlying algorithm.

Contrastive RL Contrastive RL (CRL) learns a goal-conditioned compatibility score and then extracts a policy from that score [12]. We write the critic as $f_{\phi,\psi}(s, a, g)$ and the actor as $\pi_\theta(a \mid s, g)$. In our experiments, each critic head uses the scaled inner product $f_{\phi,\psi}(s, a, g) = \phi(s, a)^\top \psi(g)/\sqrt{d}$. For a minibatch $\{(s_i, a_i, g_i)\}_{i=1}^{B}$ with matched future goals, let $f_{ij} = f_{\phi,\psi}(s_i, a_i, g_j)$. The binary-NCE objective distinguishes diagonal positive pairs from off-diagonal negatives [29, 12, 37]:

$$\mathcal{J}_{\mathrm{CRL}}(f) = \frac{1}{B^2} \sum_{i,j=1}^{B} \left[\delta_{ij} \log \sigma(f_{ij}) + (1 - \delta_{ij}) \log(1 - \sigma(f_{ij}))\right]. \tag{1}$$

Here δ_{ij} is one when $i = j$ and zero otherwise. The implementation averages this objective over two critic heads. For policy extraction, let $f_{\min}$ denote their minimum. Continuous-action CRL uses the normalized DDPG+BC objective from OGBench:

$$\mathcal{J}_{\mathrm{actor}}(\pi_\theta) = \mathbb{E}_{(s,a)\sim p_{\mathcal{D}},\, g\sim p_{\mathcal{D}}(\cdot|s)}\left[\frac{f_{\min}(s, \pi_\theta^{\mu}(s, g), g)}{c_{\mathcal{B}}} + \alpha \log \pi_\theta(a \mid s, g)\right], \tag{2}$$

where $\pi_\theta^\mu(s, g)$ is the policy mean, clipped to the action bounds in the score term [13, 37]. The normalization is $c_\mathcal{B} = \mathrm{sg}(\mathrm{mean}_\mathcal{B} |f_{\min}(s, \pi_\theta^\mu(s, g), g)| + 10^{-6})$, computed over the actor minibatch, while sg stops gradients. Thus policy extraction depends directly on the learned score while remaining regularized toward behavior support, making CRL our main diagnostic setting.

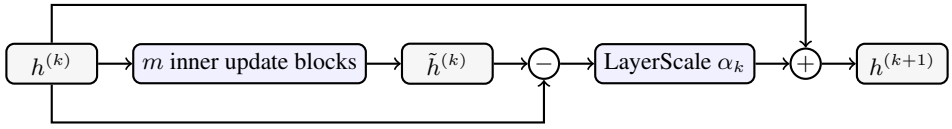
Neural network components LayerNorm normalizes a hidden vector across its feature dimensions and applies a learned scale and shift [3]. SwiGLU combines a SiLU-activated feature stream with multiplicative gating [44]:

$$\mathrm{SwiGLU}(z) = W_o[\mathrm{SiLU}(W_a z) \odot (W_b z)], \tag{3}$$

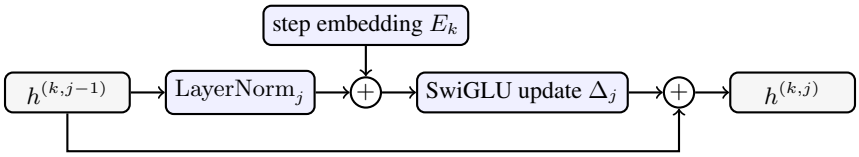
where $\mathrm{SiLU}(u) = u\sigma(u)$ and $\odot$ denotes elementwise multiplication. A residual connection adds a learned transformation to its input as $h + F(h)$ [19]. LayerScale introduces learned coefficients that scale the residual branch [49]. A learned step embedding encodes the computation index, allowing a module with shared weights to distinguish computation steps [8].

4 Iterative Latent Refinement

Our backbone refines an initial latent state $h^{(0)} \in \mathbb{R}^{d_h}$ through K refinement steps, each containing m inner blocks. The final latent state $h^{(K)}$ provides the value or critic signal used by the underlying algorithm. The latent state is initialized anew for each network input. Figure 2 shows how the building blocks form the inner and outer residual paths.



(a) Outer step: an m-block stack proposes $\tilde{h}^{(k)}$.



(b) Inner block j, reused at every refinement step.

Figure 2: Iterative latent refinement. (a) The m-block stack proposes $\tilde{h}^{(k)}$. The outer residual adds the scaled change $\alpha_k(\tilde{h}^{(k)} - h^{(k)})$. (b) Each inner block applies LayerNorm, step conditioning, and a SwiGLU residual update. The m blocks have distinct update weights, reused across all K refinement steps. E_k and α_k depend on the step.

Latent update. Let Δ_j denote a SwiGLU transformation (Equation 3) with block-specific weights, and set $h^{(k,0)} = h^{(k)}$. For inner block $j = 1, \ldots, m$, we compute

$$z^{(k,j)} = \mathrm{LN}_j\big(h^{(k,j-1)}\big) + E_k, \tag{4}$$

$$h^{(k,j)} = h^{(k,j-1)} + \Delta_j\big(z^{(k,j)}\big). \tag{5}$$

Here $E_k \in \mathbb{R}^{d_h}$ is a learned step embedding. Each inner block has its own SwiGLU and LayerNorm parameters, shared across all outer steps. After the inner stack, the proposal $\tilde{h}^{(k)} = h^{(k,m)}$ is combined with the previous outer state:

$$h^{(k+1)} = h^{(k)} + \alpha_k \left(\tilde{h}^{(k)} - h^{(k)}\right), \qquad k = 0, \ldots, K-1. \tag{6}$$

The scale α_k is a learned scalar initialized to 10^{-2}. For $m = 1$, the two residual paths simplify to $h^{(k+1)} = h^{(k)} + \alpha_k \Delta_1(z^{(k,1)})$. The step embeddings E_k and scales α_k are learned separately for each outer step. Thus K sets the refinement depth and m the capacity of each update.

5 Experiments

5.1 Experiment Setup

Environments We evaluate on state-based OGBench datasets [37] spanning AntMaze, Humanoid-Maze, Cube, and Scene. Maze tasks combine low-level locomotion with long-horizon navigation through layouts of varying size. The `navigate` datasets contain extended navigation trajectories, whereas `stitch` datasets contain short segments that must be combined to reach distant goals. Teleport variants introduce stochastic transitions. Cube evaluates pick-and-place skills and rearrangement of cubes into target configurations. Scene involves a cube, drawer, window, and buttons controlling locks, requiring ordered skill sequences such as unlocking a drawer before opening it. Both manipulation domains use unstructured play data. Training uses fixed offline datasets without additional interaction. Success is averaged over five evaluation tasks with 50 episodes per task. Appendix B lists the evaluated datasets.

Algorithms and architecture We study iterative latent refinement as an architecture level replacement inside existing offline GCRL algorithms. Our experiments include CRL, HIQL, and SAW as the main algorithmic settings. These methods expose different value-to-policy interfaces: CRL learns a contrastive state-action-goal score and extracts the actor with a DDPG+BC-style objective [12, 13]; HIQL uses hierarchical value learning with AWR-style actor extraction [35]; and SAW uses a different policy-extraction procedure based on its own subgoal-weighted training objective [58]. In each comparison, the recurrent variant replaces only the value or critic backbone that produces the learned signal consumed by the actor, while the actor networks themselves remain feedforward. The surrounding algorithm is otherwise held fixed, including replay data, goal sampling, losses, batch size, optimizer, and evaluation protocol. Appendix C specifies the replacement location for each algorithm and the recurrent hyperparameters.

Algorithm	Refinement config	Dataset	MLP	Ours
CRL	$K = 4,\ m = 1$	`antmaze-medium-navigate-v0`	**95**±**1**	**96**±**1**
CRL	$K = 4,\ m = 1$	`antmaze-large-navigate-v0`	83±4	**90**±**3**
CRL	$K = 4,\ m = 1$	`antmaze-giant-navigate-v0`	16±3	**33**±**6**
CRL	$K = 4,\ m = 1$	`antmaze-teleport-navigate-v0`	53±2	**56**±**3**
CRL	$K = 4,\ m = 1$	`antmaze-medium-stitch-v0`	53±6	**95**±**3**
CRL	$K = 4,\ m = 1$	`antmaze-large-stitch-v0`	11±2	**48**±**7**
CRL	$K = 4,\ m = 1$	`antmaze-teleport-stitch-v0`	31±4	**36**±**5**
CRL	$K = 4,\ m = 1$	`humanoidmaze-medium-navigate-v0`	**60**±**4**	**63**±**2**
CRL	$K = 4,\ m = 1$	`humanoidmaze-medium-stitch-v0`	36±2	**66**±**4**
CRL	$K = 4,\ m = 1$	`cube-single-play-v0`	19±2	**21**±**3**
CRL	$K = 4,\ m = 1$	`scene-play-v0`	19±2	**29**±**4**
HIQL	$K = 4,\ m = 2$	`antmaze-giant-navigate-v0`	65±5	**71**±**7**
HIQL	$K = 4,\ m = 2$	`antmaze-teleport-navigate-v0`	42±3	**47**±**3**
HIQL	$K = 4,\ m = 2$	`antmaze-large-stitch-v0`	67±5	**72**±**7**
HIQL	$K = 4,\ m = 2$	`antmaze-teleport-stitch-v0`	36±2	**43**±**2**
HIQL	$K = 4,\ m = 2$	`cube-single-play-v0`	**15**±**3**	**15**±**4**
HIQL	$K = 4,\ m = 2$	`scene-play-v0`	38±3	**57**±**7**
SAW	$K = 4,\ m = 1$	`antmaze-giant-navigate-v0`	73±4	**79**±**5**
SAW	$K = 4,\ m = 1$	`humanoidmaze-giant-navigate-v0`	35±4	**44**±**3**
SAW	$K = 4,\ m = 1$	`cube-double-play-v0`	40±7	**45**±**6**
SAW	$K = 4,\ m = 1$	`scene-play-v0`	**63**±**6**	**64**±**2**

Table 1: Main results on OGBench. Entries show mean success (%) ± standard deviation over eight seeds. CRL and HIQL MLP baselines are from OGBench [37], and SAW MLP baselines are from Zhou and Kao [58]. Bold denotes scores at or above 95% of the best mean in each row.

5.2 Main Results

The main comparison across locomotion and manipulation datasets is shown in Table 1. For later discussion, we abbreviate `antmaze-medium-stitch-v0` as AMS, `antmaze-large-stitch-v0` as ALS, `antmaze-giant-navigate-v0` as AGN, and `scene-play-v0` as SP. The table uses the default recurrent configurations selected for each algorithm. The roles of refinement depth K and per-step update capacity m are studied next in Section 5.3 and Section 5.4. Main results are reported over eight seeds, while the diagnostic, sweep, ablation, and compute studies below use four seeds unless stated otherwise.

5.3 Effect of Refinement Depth

We sweep the number of outer refinement steps $K \in \{1, 2, 4, 6, 8\}$ in CRL on AMS and AGN. Figure 3 compares refinement depths. Moving beyond one step improves success, but further depth does not yield monotonic gains. On AMS, $K = 6$ has the highest final mean, while $K = 4$ has the highest final mean on AGN. On AGN, final success is 32% for $K = 4$, compared with 23% for $K = 6$ and 15% for $K = 8$. On AMS, $K = 4$ and $K = 6$ both finish near 95%. These task-dependent responses favor a moderate refinement depth, and we use $K = 4$ in the main comparisons.

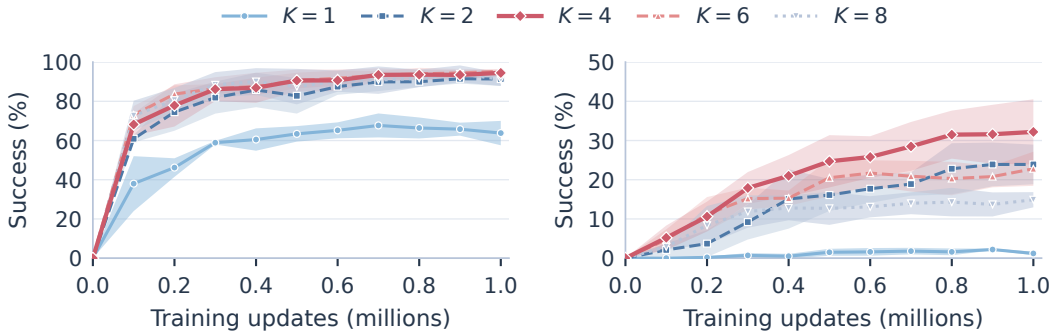


Figure 3: CRL refinement depth on AMS (left) and AGN (right). Curves compare $K = 1, 2, 4, 6, 8$ outer steps over 1M training updates, with evaluations every 100k. Markers show mean success and shaded bands show standard deviation. Moderate refinement depths outperform a single step, while further depth gives no consistent gain.

5.4 Effect of Per-Step Update Capacity

We first fix $K = 4$ and compare $m = 1$ with $m = 2$, where m is the number of inner update blocks at each refinement step. Figure 4 reports the HIQL and CRL results, with $\Delta = \text{Success}(m = 2) - \text{Success}(m = 1)$ in percentage points for each task. HIQL improves on all three tasks, whereas CRL decreases on ALS and AGN and is nearly unchanged on SP. The HIQL gains are 20 percentage points on ALS, 6 on AGN, and 9 on SP. Increasing m adds inner blocks with distinct weights, expanding the capacity of each refinement step. The effect of this additional capacity differs between HIQL and CRL.

We additionally examine the joint HIQL grid $m \in \{1, 2, 3\}$, $K \in \{2, 4, 6\}$ on AGN and ALS (Figure 5). At $K = 2$, ALS success increases from 47% to 59% to 73% as m increases. However, at $m = 3$, increasing K from 2 to 4 to 6 yields 73%, 70%, and 62%. Thus, more refinement is not uniformly beneficial, and useful computation depends on its allocation between depth and per-step capacity. These results suggest further increasing per-step capacity with moderate refinement depth could gain better performance.

5.5 What Does the Recurrent Critic Change?

End-task success alone does not explain what refinement changes. We therefore use CRL as the main diagnostic setting because its critic signal is exposed directly: CRL learns a compatibility score $f(s, a, g)$ and the DDPG+BC actor is trained to choose actions with high score under this critic. This lets us evaluate the learned signal at the policy-extraction interface. We focus on three state-based

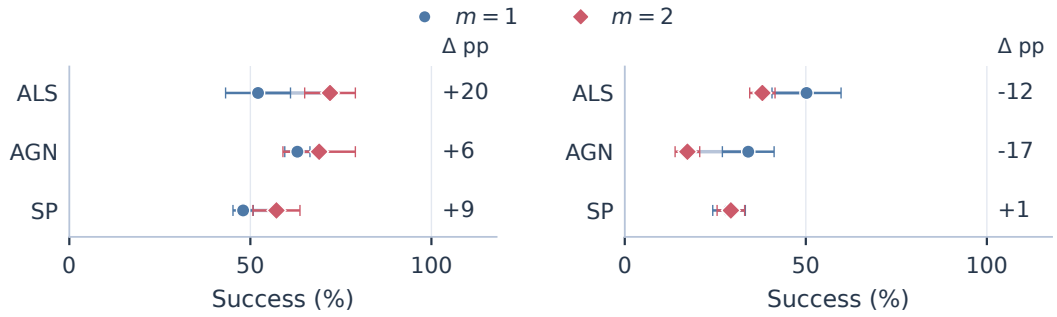


Figure 4: Per-step update capacity at $K = 4$ in HIQL (left) and CRL (right). Blue circles denote $m = 1$ and rose diamonds denote $m = 2$. Points and error bars show mean success and standard deviation. Each segment connects the two settings for one task. The Δ annotations give $m = 2$ minus $m = 1$ success in percentage points. Higher capacity benefits HIQL, while CRL shows no consistent improvement.

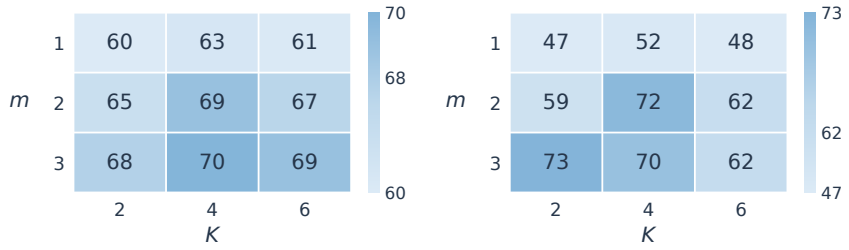


Figure 5: Joint capacity and refinement depth in HIQL on AGN (left) and ALS (right). Rows vary the number of inner blocks m, and columns vary the number of outer steps K. Cells show mean success (%). Darker blue indicates higher success within each panel, using separate nonlinear color scales ($\gamma = 2$). The highest success occurs at different (m, K) settings for the two tasks.

AntMaze tasks: AMS, ALS, and AGN. Unless stated otherwise, the comparison is between the default MLP CRL critic and the four-step recurrent CRL critic.

For critic discrimination, we use the contrastive structure of CRL. A dataset transition paired with a future goal from the same trajectory is treated as a positive pair, while pairing the same transition with a sampled goal gives a negative pair. Let $q^+ = f(s, a, g)$ and $q^- = f(s, a, g^-)$ denote these matched and mismatched scores. We report the raw margin $\mathbb{E}[q^+ - q^-]$, ranking accuracy $\Pr[q^+ > q^-]$, and normalized separation $\mathrm{Sep} = \frac{\mathbb{E}[q^+ - q^-]}{\sqrt{\mathrm{Var}(q^+) + \mathrm{Var}(q^-) + \epsilon}}$. For actor-interface diagnostics, we compare the critic scores of the extracted actor action and the dataset action on the same state-goal pairs: $q_\Delta = \mathbb{E}_{(s, a_\mathcal{D}, g)} \left[f(s, \pi^\mu(s, g), g) - f(s, a_\mathcal{D}, g) \right]$. We also report the fraction of pairs where the critic prefers the actor action and policy–behavior MSE/L2 distance. Lower q_Δ and smaller policy–behavior distance are proxies for less extrapolative actor extraction. In particular, q_Δ is used as a within-critic action-preference diagnostic. Finally, graph-distance correlations test alignment with a shortest-path surrogate. Scores are correlated with negative shortest-path distance, so positive correlation associates higher scores with nearer goals. The hard-pair diagnostic restricts evaluation to the upper third of finite path distances.

Critic discrimination changes in normalized form. Iterative refinement increases normalized separation between matched and mismatched goals on ALS and AGN, while AMS remains nearly unchanged (Figure 6(a,b), with complementary statistics in Appendix E). Because rank accuracy is already high on several tasks, we treat the separation metric as the more informative critic-side diagnostic. The result suggests that recurrence changes the score distribution beyond a simple increase in raw score scale.

Actor-interface proxies indicate less extrapolation. The clearest diagnostic change appears at the actor interface. With the recurrent critic, the extracted actor reaches higher success while staying closer to dataset actions under the measured proxies. Across AMS, ALS, and AGN, q_Δ and policy

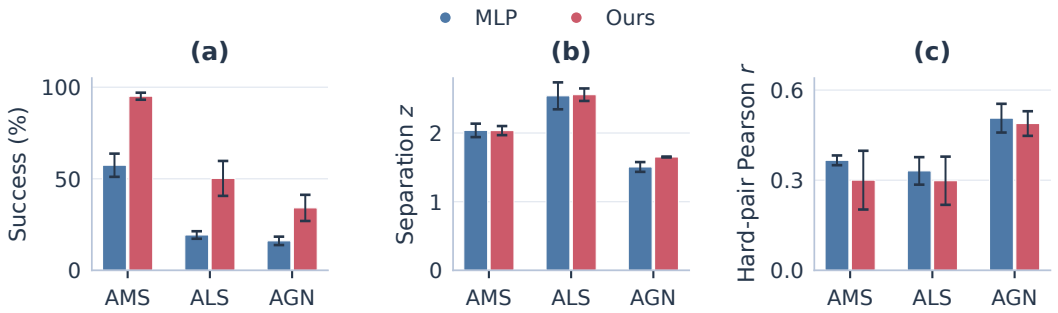
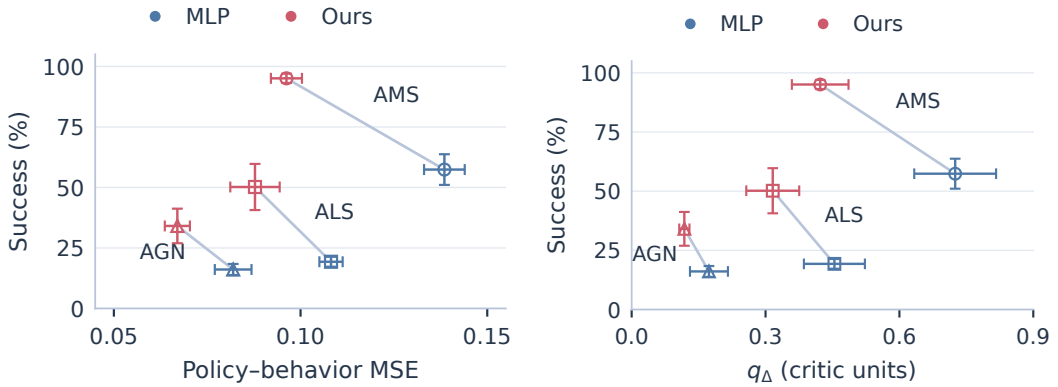


Figure 6: CRL diagnostics: (a) success, (b) normalized matched–mismatched goal-score separation, and (c) hard-pair Pearson correlation with negative graph distance. Blue and rose denote MLP and Ours. Hard pairs lie in the upper third of finite distances. Refinement improves success and lowers hard-pair graph correlations. Separation increases on ALS and AGN and remains similar on AMS.

MSE decrease (Figure 7a–7b), with complementary actor-interface statistics in Appendix E. These trends are consistent with the recurrent critic providing a more usable offline policy-extraction signal.



(a) Policy MSE (b) q_Δ

Figure 7: CRL actor-interface diagnostics: success versus (a) policy–behavior MSE and (b) q_Δ, the actor-minus-dataset critic score. Segments connect MLP and Ours on the same task. Refinement achieves higher success alongside lower policy MSE and q_Δ.

Graph-distance alignment is not the sole explanation. Success improves on all three tasks while hard-pair graph correlations decrease. Global correlations remain similar on AMS and decrease on ALS and AGN (Figure 6(c), Appendix E). These diagnostics probe different aspects of the learned signal. A graph correlation summarizes how critic scores order state–goal pairs across a collection of distances. Policy extraction instead compares actions at a fixed state and goal. Better action comparisons can therefore improve control even when the ordering across different state–goal pairs changes little. The actor-interface metrics complement graph alignment by examining this action-selection stage directly. Thus, improved control need not coincide with uniformly stronger global distance alignment.

Trajectory effects appear at bottlenecks. In maze layouts, the actor often needs to choose behavior-like actions that are not locally goal-directed, such as entering a corridor, completing a turn, or temporarily moving away from the goal to follow a U-shaped path. Figure 8 illustrates this interpretation. The MLP actor can reach a bottleneck region but stall or follow a locally tempting direction, while the recurrent-critic actor follows the corridor structure in the shown rollouts. The policies have the same feedforward architecture and differ in the critic used for training. The trajectory comparison thus illustrates how a change in value computation can affect action selection throughout an episode, including at successive corridor turns. The critic trains a feedforward actor, so the change in route selection does not require recurrent policy memory. Iterative computation affects the training signal, while the actor retains its original computation at deployment.

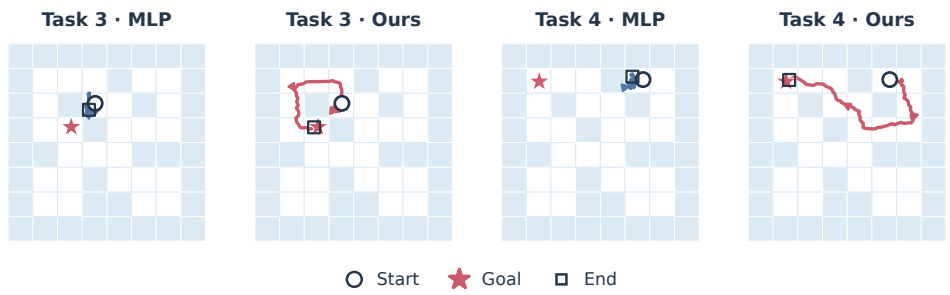


Figure 8: Selected CRL trajectories on AMS tasks 3 (first two panels) and 4 (last two panels). Each task compares MLP with Ours from the same initial state and goal. Circles, stars, and hollow squares mark starts, goals, and endpoints. Paths show how the two policies traverse the maze corridors and bottleneck regions.

5.6 Matched Parameter and Computation Comparison

We compare the recurrent CRL critic with a wider three-layer MLP of width 688 and a deeper five-layer MLP of width 512. Both controls preserve CRL's dense critic architecture, including GELU activations and LayerNorm. Their parameter counts are within 0.37% and 0.19% of the recurrent critic, respectively. Appendix C gives the parameter-counting details.

Neither feedforward control closes the success gap in Figure 9. The improvement is therefore not explained by parameter count alone. The two controls allocate their parameters differently: the wider network increases the feature dimension at each layer, while the deeper network adds successive transformations with independent weights. Evaluating both tests whether the gain can be recovered by expanding either representation size or feedforward depth. Refinement instead reuses its parameters as the latent state evolves across steps.

The recurrent model requires approximately $3.4\times$ the default MLP's FLOPs per gradient update (Appendix F). At approximately matched training FLOPs, success increases from 51% to 86% on AMS, from 15% to 29% on ALS, and from 12% to 18% on AGN.

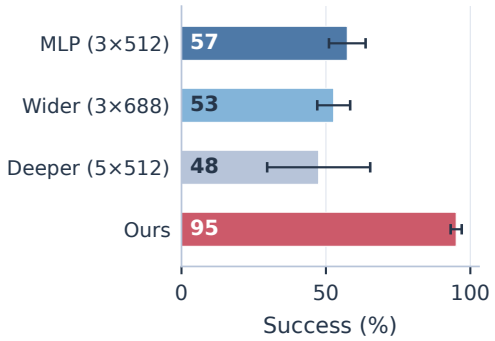


Figure 9: Parameter-matched CRL critics on AMS. The wider 3×688 and deeper 5×512 MLPs are compared with the default 3×512 MLP and our $K = 4, m = 1$ critic. The wider and deeper controls approximately match our critic's parameter count.

5.7 Ablation on Model Components

We isolate the components of the recurrent update while keeping the CRL objective and refinement depth fixed. Each variant preserves the recurrent critic but removes one architectural choice: the refinement step embedding, normalization in the update block, LayerScale on the residual update, or the SwiGLU update MLP.

Figure 10 shows that the complete update reaches 95% on AMS and has the highest mean success on ALS at 50%. Removing step conditioning leaves AMS success nearly unchanged at 95% but lowers ALS success to 39%. Removing LayerNorm lowers success to 88% and 28%, while removing LayerScale yields 80% and 20%. Replacing SwiGLU with a dense update yields 75% and 30%.

The relative effects differ across tasks. Replacing SwiGLU causes the largest mean reduction on AMS, while removing LayerScale is most damaging on ALS. Step conditioning has little effect on AMS success but contributes more on ALS. This pattern points to complementary roles for the components: gated feature transformations are particularly useful on AMS, whereas controlling residual-update magnitude is especially important on ALS. Reusing the update means that each component affects a sequence of latent transformations. Its influence therefore depends on both the update itself and how often it is applied.

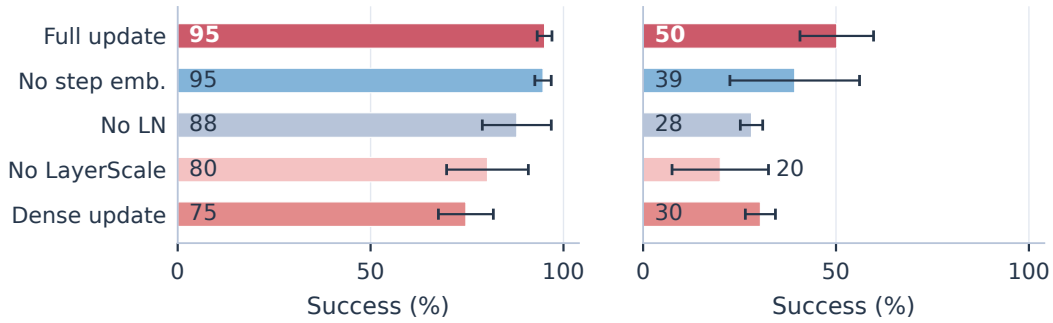


Figure 10: CRL component ablations on AMS (left) and ALS (right). The full update is compared with variants that remove step embeddings, LayerNorm, or LayerScale, or replace SwiGLU with a dense update. Refinement depth and the CRL objective are fixed.

Step embeddings let the shared update distinguish refinement stages, with a larger benefit on ALS. LayerNorm controls feature scale and LayerScale regulates residual changes, providing complementary controls on stability under repeated updates. SwiGLU uses input-dependent gating to select which features to modify. The performance losses from removing these components support combining step-specific computation, normalization, residual scaling, and gating within the shared recurrent update.

6 Conclusion

We studied iterative latent refinement as an architectural choice for value learning in offline GCRL. The backbone repeatedly updates an input-conditioned latent state with shared weights while retaining the underlying algorithm's objectives and actor architecture. Across CRL, HIQL, and SAW, this replacement improves mean success on many evaluated OGBench tasks, with the largest gains on stitching and bottleneck mazes. CRL diagnostics show stronger normalized goal discrimination and smaller actor deviations from dataset actions alongside higher success. Depth and capacity studies reveal that additional refinement is not uniformly beneficial. Its effectiveness depends on how computation is divided between the number of steps and the capacity of each update. These findings motivate treating the architecture of the value-to-policy interface as a practical design choice in offline GCRL.

Limitations and future work. Sharing update weights couples the transformations and training gradients across refinement steps. A central open question is how this coupling shapes the inductive bias and optimization of value learning. Separating the effects of reusable computation, parameter tying, and step conditioning calls for controlled comparisons of shared and unshared updates, together with analyses of how latent trajectories preserve reachability information and propagate estimation errors.

Viewing refinement as a learned dynamical system offers a route to understanding these effects. Studies of sequential cognition [39] and recurrent spatial representations [6] provide perspectives on how recurrent dynamics organize representations. Analyses of latent trajectories and their sensitivity to perturbations could connect these dynamics to score discrimination and policy extraction. Such an account could guide the choice of refinement depth and update capacity, as well as adaptive stopping criteria [17].

Acknowledgements and Disclosures

This work was supported by the National Key R&D Program of China (Project No. 2025YFA1016700), the National Natural Science Foundation of China (Grant No. 62576011), the Beijing Major Science and Technology Project (Contract No. Z251100008125055), and the Beijing Academy of Artificial Intelligence (BAAI). We thank the School of Computing at the National University of Singapore for providing cluster computing resources used in the experiments.

The authors declare no competing interests.

References

[1] Hongjoon Ahn, Heewoong Choi, Jisu Han, and Taesup Moon. Option-aware temporally abstracted value for offline goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.48550/arXiv.2505.12737.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Neural Information Processing Systems*, volume 30, 2017.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[4] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Neural Information Processing Systems*, volume 32, 2019.

[5] Thomas Bush, Stephen Chung, Usman Anwar, Adrià Garriga-Alonso, and David Krueger. Interpreting emergent planning in model-free reinforcement learning. In *International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=DzGe40glxs`.

[6] Yunxiang Chen, Tao Wang, Wei Wang, and Feng Liu. Emergent attractor-like dynamics support stable and generalizable path integration in recurrent neural networks. *Cognitive Neurodynamics*, 20:179, 2026. doi: 10.1007/s11571-026-10555-8.

[7] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Neural Information Processing Systems*, volume 5, pages 271–278, 1992.

[8] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

[9] Yilun Du, Jiayuan Mao, and Joshua B. Tenenbaum. Learning iterative reasoning through energy diffusion. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11764–11776, 2024. doi: 10.48550/arXiv.2406.11179.

[10] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Neural Information Processing Systems*, volume 32, 2019.

[11] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*, 2021.

[12] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Neural Information Processing Systems*, volume 35, pages 35603–35620. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.48550/arXiv.2206.07568.

[13] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Neural Information Processing Systems*, volume 34, 2021.

[14] Zitian Gao, Lynx Chen, Yihao Xiao, He Xing, Ran Tao, Haoming Luo, Joey Zhou, and Bryan Dai. Universal reasoning model. arXiv preprint arXiv:2512.14693, 2025.

[15] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021.

[16] Raj Ghugare, Michał Bortkiewicz, Alicja Ziarko, and Benjamin Eysenbach. On the role of computation in reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 34884–34898, 2026. URL `https://proceedings.mlr.press/v306/ghugare26a.html`.

[17] Alex Graves. Adaptive computation time for recurrent neural networks. arXiv preprint arXiv:1603.08983, 2016.

[18] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An investigation of model-free planning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2464–2473, 2019. URL `https://proceedings.mlr.press/v97/guez19a.html`.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, pages 770–778, 2016. doi: 10.1109/cvpr.2016.90.

[20] Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. arXiv preprint arXiv:2510.04871, 2025.

[21] Kaiqiang Ke, Qian Lin, Zongkai Liu, Shenghong He, and Chao Yu. Conservative offline goal-conditioned implicit V-learning. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 29591–29607, 2025.

[22] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022.

[23] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *Neural Information Processing Systems*, volume 33, pages 1179–1191, 2020.

[24] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

[25] Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-based deep reinforcement learning requires explicit regularization. In *International Conference on Learning Representations*, 2022.

[26] Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R. Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. SimBa: Simplicity bias for scaling up parameters in deep reinforcement learning. In *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2410.09754.

[27] Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 33352–33403, 2025. URL `https://proceedings.mlr.press/v267/lee25u.html`.

[28] Jason Yecheng Ma, Jason Yan, Dinesh Jayaraman, and Osbert Bastani. Offline goal-conditioned reinforcement learning via f-advantage regression. In *Advances in Neural Information Processing Systems 35*, volume 35, pages 310–323. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-0023.

[29] Zhuang Ma and Michael Collins. Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. In *Conference on Empirical Methods in Natural Language Processing*, pages 3698–3707. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1405.

[30] Ashvin V. Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Neural Information Processing Systems*, volume 31, 2018.

[31] Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: Scaling for compute and sample-efficient continuous control. In *Advances in Neural Information Processing Systems*, volume 37, pages 113038–113071. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.48550/arXiv.2405.16158.

[32] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847, 2022. doi: 10.48550/arXiv.2205.07802.

[33] Evgenii Opryshko, Junwei Quan, Claas A. Voelcker, Yilun Du, and Igor Gilitschenski. Test-time graph search for goal-conditioned reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 94754–94774, 2026. URL `https://proceedings.mlr.press/v306/opryshko26a.html`.

[34] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Neural Information Processing Systems*, volume 31, 2018.

[35] Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. HIQL: Offline goal-conditioned RL with latent states as actions. In *Neural Information Processing Systems*, volume 36, pages 34866–34891. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.48550/arXiv.2307.11949.

[36] Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline RL? In *Neural Information Processing Systems*, volume 37, pages 79029–79056. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.48550/arXiv.2406.09329.

[37] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2410.20092.

[38] Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes RL scalable. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.48550/arXiv.2506.04168.

[39] Thomas Parr, Karl Friston, and Giovanni Pezzulo. Generative models for sequential dynamics in active inference. *Cognitive Neurodynamics*, 18:3259–3272, 2024. doi: 10.1007/s11571-023-09963-x.

[40] Vitchyr H. Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. In *International Conference on Learning Representations*, 2018.

[41] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7783–7792, 2020.

[42] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[43] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *Neural Information Processing Systems*, volume 34, pages 6695–6706, 2021.

[44] Noam Shazeer. GLU variants improve transformer. arXiv preprint arXiv:2002.05202, 2020.

[45] Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2RL: Deep dense architectures in reinforcement learning. arXiv preprint arXiv:2010.09163, 2020.

[46] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32145–32168, 2023. doi: 10.48550/arXiv.2302.12902.

[47] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2): 181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

[48] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016. URL `https://proceedings.neurips.cc/paper_files/paper/2016/hash/c21002f464c5fc5bee3b98ced83963b8-Abstract.html`.

[49] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *IEEE International Conference on Computer Vision*, pages 32–42. IEEE, 2021. doi: 10.1109/ICCV48922.2021.00010.

[50] Carlos Vélez García, Miguel Cazorla, and Jorge Pomares. Negative energy as reward: Optimizing beyond demonstrations in offline goal-conditioned control. In *RL4IL Workshop at the IEEE International Conference on Robotics and Automation*, 2026. URL `https://openreview.net/forum?id=r38GkjaxYh`.

[51] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi-Yadkori. Hierarchical reasoning model. arXiv preprint arXiv:2506.21734, 2025.

[52] Kevin A. Wang, Ishaan Javali, Michał Bortkiewicz, Tomasz Trzciński, and Benjamin Eysenbach. 1000 layer networks for self-supervised RL: Scaling depth can enable new goal-reaching capabilities. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.48550/arXiv.2503.14858.

[53] Mianchu Wang, Yue Jin, and Giovanni Montana. Goal-conditioned offline reinforcement learning through state space partitioning. *Machine Learning*, 113:2435–2465, 2024. doi: 10.1007/s10994-023-06500-z.

[54] Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 36411–36430, 2023. doi: 10.48550/arXiv.2304.01203.

[55] Zeyuan Wang, Da Li, Yulin Chen, Ye Shi, Liang Bai, Tianyuan Yu, and Yanwei Fu. One-step generative policies with q-learning: A reformulation of MeanFlow. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 26751–26759, 2026. doi: 10.1609/aaai.v40i31.39885.

[56] Yulong Xia and Fuchun Sun. Behavior regularization with flow latent policy for offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 27028–27036, 2026. doi: 10.1609/aaai.v40i32.39916.

[57] Rui Yang, Yiming Lu, Wenzhe Li, Hao Sun, Meng Fang, Yali Du, Xiu Li, Lei Han, and Chongjie Zhang. Rethinking goal-conditioned supervised learning and its connection to offline RL. In *International Conference on Learning Representations*, 2022.

[58] John L. Zhou and Jonathan C. Kao. Flattening hierarchies with policy bootstrapping. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.48550/arXiv.2505.14975.

A Algorithm Background

This section summarizes the value-to-policy interfaces of HIQL and SAW and specifies where recurrent refinement is inserted. We use the OGBench implementation of HIQL [37, 35] and the state-based SAW formulation [58].

Shared value learning. Both methods learn two goal-conditioned value heads $V_{\psi,i}(s,g)$, $i \in \{1,2\}$, with target heads $V_{\bar{\psi},i}$. Write $V_\psi = (V_{\psi,1} + V_{\psi,2})/2$ and $V_{\bar{\psi}} = (V_{\bar{\psi},1} + V_{\bar{\psi},2})/2$ for the online and target ensemble means. For a relabeled dataset transition (s, a, s', g), let r_g denote the goal-conditioned reward and M_g the continuation mask. The reference implementation uses

$$y_i = r_g + \gamma M_g V_{\bar{\psi},i}(s', g), \qquad A_V = \min_{i\in\{1,2\}} y_i - V_{\bar{\psi}}(s, g), \tag{7}$$

$$\mathcal{L}_V = \mathbb{E}\left[\sum_{i=1}^{2} w_\kappa(A_V)\big(y_i - V_{\psi,i}(s,g)\big)^2\right], \qquad w_\kappa(u) = \begin{cases}\kappa, & u \geq 0,\\ 1-\kappa, & u < 0.\end{cases} \tag{8}$$

Here κ is the expectile coefficient. Targets and expectile weights are treated as constants during differentiation. In particular, the weight is determined by the target-network advantage, rather than by the current-head regression residual. Target parameters are updated by exponential moving averaging with rate τ_{EMA}. For the actor objectives below, define the clipped advantage weight $\omega_c(u) = \mathrm{sg}(\min\{\exp(cu), 100\})$, where sg stops gradients through the value-derived weight during actor optimization.

HIQL. HIQL decomposes long-horizon goal reaching into high-level subgoal prediction and low-level action generation [35]. Let $\phi(s, w)$ be the learned, length-normalized representation of a subgoal w. The high-level actor $\pi_h(z \mid s, g)$ predicts a subgoal representation, and the low-level actor $\pi_\ell(a \mid s, z)$ maps that representation to a primitive action. For a sampled low-level goal g_ℓ and high-level target state w, the advantages are

$$A_\ell = V_\psi(s', g_\ell) - V_\psi(s, g_\ell), \qquad A_h = V_\psi(w, g) - V_\psi(s, g). \tag{9}$$

The actors are trained by advantage-weighted behavioral cloning:

$$\mathcal{L}_\ell^{\mathrm{HIQL}} = -\mathbb{E}\left[\omega_{\alpha_\ell}(A_\ell) \log \pi_\ell(a \mid s, \phi(s, g_\ell))\right], \tag{10}$$

$$\mathcal{L}_h^{\mathrm{HIQL}} = -\mathbb{E}\left[\omega_{\alpha_h}(A_h) \log \pi_h(\phi(s, w) \mid s, g)\right]. \tag{11}$$

For the state-based tasks considered here, actor losses do not update ϕ: the low-level conditioning representation and high-level regression target are detached. At evaluation, the predicted subgoal representation is length-normalized before being passed to the low-level actor. Refinement replaces the online and target value backbones. The goal-representation module, both actors, and their training objectives retain the reference architecture and formulation.

SAW. SAW bootstraps a flat goal-conditioned policy $\pi(a \mid s, g)$ from an auxiliary subgoal-conditioned policy $\pi_\ell(a \mid s, w)$ [58]. The auxiliary policy is used during training. Evaluation queries only the flat policy. In addition to the shared value loss, the two policies use one-step advantage-weighted losses

$$\mathcal{L}_\ell^{\mathrm{SAW}} = -\mathbb{E}\left[\omega_{\alpha_\ell}(A_\ell) \log \pi_\ell(a \mid s, g_\ell)\right], \qquad A_\ell = V_\psi(s', g_\ell) - V_\psi(s, g_\ell), \tag{12}$$

$$\mathcal{L}_\pi^{\mathrm{SAW}} = -\mathbb{E}\left[\omega_{\alpha_{\mathrm{awr}}}(A_\pi) \log \pi(a \mid s, g)\right], \qquad A_\pi = V_\psi(s', g) - V_\psi(s, g). \tag{13}$$

For a dataset waypoint w, the policy-bootstrapping term is

$$\mathcal{L}_{\mathrm{wp}}^{\mathrm{SAW}} = \mathbb{E}\left[\omega_\beta(A_w) D(\pi_\ell(\cdot \mid s, w), \pi(\cdot \mid s, g))\right], \qquad A_w = V_\psi(w, g) - V_\psi(s, g). \tag{14}$$

For the fixed-unit-variance Gaussian policies used here, $D = \|\mu_\pi(s,g) - \mathrm{sg}[\mu_\ell(s,w)]\|_2^2$, where sg denotes stop-gradient. This is proportional to the KL divergence between the two policies. The total loss is $\mathcal{L}_V + \mathcal{L}_\ell^{\mathrm{SAW}} + \mathcal{L}_\pi^{\mathrm{SAW}} + \mathcal{L}_{\mathrm{wp}}^{\mathrm{SAW}}$. Recurrent refinement replaces the online and target value backbones, while the feedforward goal encoders, auxiliary policy, flat policy, and policy-bootstrapping objective retain the SAW formulation.

B OGBench Tasks Used in This Study

We use the state-based OGBench datasets listed in Table 2 [37]. Training uses fixed offline trajectories. Environment interaction is used only for evaluation. Each dataset provides five predefined evaluation tasks with specified initial-state and goal configurations. For each evaluation, we run 50 episodes per task and average the binary episode-success indicators over all five tasks, giving 250 episodes per dataset and seed.

The main comparison contains 21 algorithm–dataset pairs spanning 13 distinct datasets in two domains. Locomotion covers AntMaze and HumanoidMaze navigation and stitching tasks. Manipulation covers Cube and Scene rearrangement tasks. Navigation evaluates goal reaching over different maze sizes, while stitching datasets require combining fragments of offline trajectories. The teleport variants additionally introduce stochastic transitions. Manipulation datasets contain play trajectories and evaluate reaching target object or scene configurations.

Table 2: State-based OGBench datasets in Table 1, grouped by domain and environment family.

Domain	Environment	Datasets used	Main evaluation stressor
Locomotion	AntMaze navigate	`antmaze-medium-navigate-v0`, `antmaze-large-navigate-v0`, `antmaze-giant-navigate-v0`, `antmaze-teleport-navigate-v0`	Long-horizon navigation to held-out maze goals
Locomotion	AntMaze stitch	`antmaze-medium-stitch-v0`, `antmaze-large-stitch-v0`, `antmaze-teleport-stitch-v0`	Goal reaching by stitching trajectory fragments
Locomotion	HumanoidMaze	`humanoidmaze-medium-navigate-v0`, `humanoidmaze-medium-stitch-v0`, `humanoidmaze-giant-navigate-v0`	High-dimensional humanoid locomotion in maze layouts
Manipulation	Cube	`cube-single-play-v0`, `cube-double-play-v0`	Moving one or more cubes to target configurations
Manipulation	Scene	`scene-play-v0`	Compositional object and scene-element manipulation

Abbreviations used in the main text are AMS for `antmaze-medium-stitch-v0`, ALS for `antmaze-large-stitch-v0`, AGN for `antmaze-giant-navigate-v0`, and SP for `scene-play-v0`.

C Hyperparameters and Implementation Details

We use the state-based OGBench benchmark settings for CRL and HIQL [37], and the corresponding state-based settings from SAW [58]. Table 3 lists the shared optimization and evaluation settings. Tables 4–6 give the algorithm-specific and dataset-specific settings for Table 1. The feedforward actors retain the corresponding reference implementations. Recurrent-backbone implementation details are given at the end of this appendix.

Table 3: Shared settings for the state-based main experiments.

Hyperparameter	Setting
Optimizer	Adam
Learning rate	3×10^{-4}
Batch size	1024
Training updates	10^6
Feedforward hidden dimensions	$(512, 512, 512)$
Feedforward activation	GELU
Actor distribution	Gaussian with fixed unit variance
Evaluation actions	Deterministic policy mean
Evaluation tasks per dataset	5
Evaluation episodes per task	50

CRL. CRL uses the OGBench DDPG+BC actor objective with behavioral-cloning coefficient α. Each critic branch outputs a 512-dimensional embedding, and the critic uses an ensemble of two members. Critic training samples goals geometrically from future states in the same trajectory, with

sampling probability $1-\gamma$ at each step. Actor goals are sampled uniformly from future trajectory states or uniformly from the dataset, with the mixture probabilities in Table 4. The current-state actor-goal probability is zero for every row. Thus, critic-goal sampling uses $(p_{\mathrm{cur}}, p_{\mathrm{traj}}, p_{\mathrm{rand}}) = (0, 1, 0)$, with geometric sampling for the trajectory component, while actor-goal sampling uses uniform trajectory sampling.

Table 4: Dataset-specific CRL settings from the OGBench benchmark configuration. α is the DDPG+BC behavioral-cloning coefficient. Actor trajectory goals are sampled uniformly.

Dataset	α	Discount	Actor traj-goal prob.	Actor random-goal prob.
`antmaze-medium-navigate-v0`	0.1	0.99	1.0	0.0
`antmaze-large-navigate-v0`	0.1	0.99	1.0	0.0
`antmaze-giant-navigate-v0`	0.1	0.995	1.0	0.0
`antmaze-teleport-navigate-v0`	0.1	0.99	1.0	0.0
`antmaze-medium-stitch-v0`	0.1	0.99	0.5	0.5
`antmaze-large-stitch-v0`	0.1	0.99	0.5	0.5
`antmaze-teleport-stitch-v0`	0.1	0.99	0.5	0.5
`humanoidmaze-medium-navigate-v0`	0.1	0.995	1.0	0.0
`humanoidmaze-medium-stitch-v0`	0.1	0.995	0.5	0.5
`cube-single-play-v0`	3.0	0.99	1.0	0.0
`scene-play-v0`	3.0	0.99	1.0	0.0

HIQL. HIQL uses expectile coefficient $\kappa = 0.7$, target update rate $\tau_{\mathrm{EMA}} = 0.005$, and low-level and high-level AWR inverse temperatures $\alpha_\ell = \alpha_h = 3.0$. The value ensemble has two members, and the length-normalized subgoal representation has dimension 10. Value goals are sampled with probabilities $(0.2, 0.5, 0.3)$ for the current state, geometrically sampled future trajectory states, and uniformly sampled dataset states, respectively. Actor trajectory goals are sampled uniformly, and the current-state actor-goal probability is zero. The subgoal horizon k is 25 for the AntMaze datasets and 10 for Cube and Scene, as specified in Table 5. Actor losses do not backpropagate through the shared subgoal representation for these state-based tasks.

Table 5: Dataset-specific HIQL settings from the OGBench benchmark configuration. Both AWR inverse temperatures are 3.0.

Dataset	Discount	k	Actor traj-goal prob.	Actor random-goal prob.
`antmaze-giant-navigate-v0`	0.995	25	1.0	0.0
`antmaze-teleport-navigate-v0`	0.99	25	1.0	0.0
`antmaze-large-stitch-v0`	0.99	25	0.5	0.5
`antmaze-teleport-stitch-v0`	0.99	25	0.5	0.5
`cube-single-play-v0`	0.99	10	1.0	0.0
`scene-play-v0`	0.99	10	1.0	0.0

SAW. SAW uses a two-member value ensemble with target update rate $\tau_{\mathrm{EMA}} = 0.005$ and the same value-goal mixture as HIQL. All four reported datasets use uniformly sampled future actor goals, with zero current-state and random-goal probabilities. Following the state-based SAW implementation, the value and flat-policy goal encoders are separate feedforward modules with 10-dimensional length-normalized outputs. The auxiliary subgoal-conditioned actor receives the raw state and subgoal. Table 6 follows Table 2 of Zhou and Kao [58]. Its AWR coefficient α applies to both the auxiliary policy and the flat policy ($\alpha_\ell = \alpha_{\mathrm{awr}}$), while β is the inverse temperature of the waypoint-advantage weight.

Table 6: Dataset-specific state-based SAW settings. κ denotes the value expectile, α the shared AWR inverse temperature, β the waypoint-weight inverse temperature, and k the subgoal horizon.

Dataset	Discount	κ	$\alpha_\ell = \alpha_{\mathrm{awr}}$	β	k	Actor traj-goal prob.	Actor random-goal prob.
`antmaze-giant-navigate-v0`	0.995	0.7	3.0	3.0	25	1.0	0.0
`humanoidmaze-giant-navigate-v0`	0.995	0.7	3.0	3.0	100	1.0	0.0
`cube-double-play-v0`	0.99	0.7	3.0	1.0	10	1.0	0.0
`scene-play-v0`	0.99	0.7	3.0	1.0	10	1.0	0.0

Recurrent implementation. The backbone in Section 4 uses a SwiGLU block with three 512×512 weight matrices. The learned per-step LayerScale parameters are initialized to 10^{-2}. For CRL, we replace both the state–action branch ϕ and the goal branch ψ, preserving the bilinear score and 512-dimensional branch outputs. For HIQL and SAW, we replace the scalar online-value and target-value backbones. The goal-representation encoders and actors remain feedforward. Each ensemble member has its own recurrent parameters, with update weights shared across refinement steps within that member. No auxiliary training loss is added.

Parameter counting. We count the scalar entries in trainable parameter tensors, counting each shared tensor once. The matched-parameter study in Figure 9 compares complete critics, including both ensemble members and both embedding branches. The actor is identical across these controls. Table 9 reports combined actor and critic counts.

D Additional CRL Backbone Results

We evaluate BroNet, SimBa, and IRU-4 as alternative critic backbones within CRL on AMS, ALS, and Scene. Each run uses 10^6 gradient updates and a batch size of 1024. Table 7 summarizes success rates, parameter counts, and update FLOPs.

Table 7: CRL alternative-backbone comparison. Success entries report mean (%) $\pm$ sample standard deviation over eight training seeds. Parameter counts include the actor and both critic ensemble members. GFLOPs/update are static XLA estimates for full FP32 training updates at batch size 1024, including backpropagation and Adam. AMS and ALS have identical network dimensions.

	Success (%)			Parameters (M)		GFLOPs/update	
Backbone	AMS	ALS	Scene	AntMaze	Scene	AntMaze	Scene
BroNet	64$\pm$11	23$\pm$7	27$\pm$4	5.90	5.93	59.60	59.78
SimBa	39$\pm$6	16$\pm$9	24$\pm$5	18.49	18.52	175.52	175.69
IRU-4	70$\pm$9	28$\pm$6	27$\pm$9	5.89	5.92	176.47	176.64

Integration with CRL. Each backbone replaces the state–action branch $\phi(s, a)$ and goal branch $\psi(g)$ in both critic ensemble members. Hidden width and branch output dimension are 512, and the score remains $\phi(s, a)^\top \psi(g)/\sqrt{512}$. CRL's three-layer, 512-wide feedforward actor, contrastive BCE objective, and DDPG+BC actor objective are retained.

BroNet. The BroNet-based backbone [31] uses a Dense–LayerNorm–ReLU input stem and two residual blocks. Each residual branch applies Dense–LayerNorm–ReLU–Dense–LayerNorm–ReLU, with LeCun normal initialization for hidden-layer weights.

SimBa. The SimBa-based backbone [26] uses two pre-LayerNorm residual blocks with fourfold expansion, ReLU, and contraction, followed by a final LayerNorm. Critic observations and goals are used without running-statistics input normalization. Actor inputs and actions are unchanged.

IRU-4. The IRU-based backbone [16] uses a normalized, input-conditioned context and applies an interpolation recurrent unit for four steps with shared weights. Sigmoid gates interpolate the recurrent state with a tanh candidate. The recurrent hidden state is updated through a residual connection.

E Additional Diagnostics

Table 8 provides complementary critic, actor-interface, and graph-distance statistics for the comparison in Figures 6 and 7.

Table 8: Complementary CRL diagnostics, reported as mean $\pm$ standard deviation. Actor preference is the fraction of state-goal pairs on which the critic scores the actor action above the dataset action. Graph correlations use negative shortest-path distance. Hard pairs lie in the upper third of finite path distances. Raw margins are in critic-specific score units.

	AMS		ALS		AGN	
Metric	MLP	Ours	MLP	Ours	MLP	Ours
Raw margin	18.544 ± 2.075	20.020 ± 1.541	19.835 ± 2.915	20.206 ± 1.651	11.943 ± 0.944	11.199 ± 0.657
Rank acc.↑	0.988 ± 0.002	0.991 ± 0.003	0.994 ± 0.002	0.996 ± 0.003	0.969 ± 0.006	0.973 ± 0.003
Actor preference	0.706 ± 0.006	0.669 ± 0.006	0.669 ± 0.011	0.659 ± 0.018	0.676 ± 0.029	0.674 ± 0.020
Policy L2↓	0.920 ± 0.015	0.781 ± 0.007	0.830 ± 0.009	0.756 ± 0.025	0.743 ± 0.014	0.681 ± 0.014
Graph Pearson r	0.515 ± 0.023	0.517 ± 0.027	0.505 ± 0.016	0.488 ± 0.030	0.693 ± 0.013	0.689 ± 0.006
Graph Spearman ρ	0.535 ± 0.031	0.549 ± 0.017	0.532 ± 0.019	0.515 ± 0.032	0.766 ± 0.012	0.762 ± 0.011
Hard-pair Pearson r	0.366 ± 0.016	0.300 ± 0.098	0.331 ± 0.046	0.298 ± 0.080	0.507 ± 0.048	0.489 ± 0.041
Hard Spearman ρ	0.384 ± 0.014	0.320 ± 0.101	0.355 ± 0.049	0.319 ± 0.085	0.507 ± 0.048	0.486 ± 0.049

F Computation Comparison

Table 9 compares parameter counts, training FLOPs, and success at approximately matched training FLOPs.

Table 9: CRL computation comparison. Parameter counts include the actor and both critic ensemble members. FLOPs are static XLA estimates for FP32 updates with batch size 1024, including forward computation, backpropagation, and Adam. Total training FLOPs cover 1M updates. Compute-matched success compares 1M MLP updates with 300k recurrent updates, with the latter using approximately 103% of the MLP's total FLOPs. Success entries show mean $\pm$ sample standard deviation.

Task	Critic	Params (M)	GFLOPs/update	Training PFLOPs	Compute-matched success (%)
`AMS`	MLP	3.79	40.08	40.08	$51_{\pm 10}$
`AMS`	Ours	5.89	137.36	137.36	$86_{\pm 6}$
`ALS`	MLP	3.79	40.08	40.08	$15_{\pm 1}$
`ALS`	Ours	5.89	137.03	137.03	$29_{\pm 9}$
`AGN`	MLP	3.79	40.08	40.08	$12_{\pm 4}$
`AGN`	Ours	5.89	137.03	137.03	$18_{\pm 4}$

The recurrent model requires approximately $3.4\times$ as many FLOPs per gradient update as the MLP on all three tasks. At approximately matched training FLOPs, refinement improves mean success on all three tasks.

G Computational Resources

Most experiments were run on NVIDIA H20 GPUs with 96 GB of memory, typically with two independent training jobs sharing one GPU. The implementation was also tested on NVIDIA RTX 3090, RTX 4090, and H100 GPUs. Under the typical two-job-per-GPU setting, a run of 10^6 gradient updates took approximately 2–3 hours for an MLP baseline and 3–4 hours for a recurrent model with $K = 4$. Runs with $K = 8$ took up to approximately 6 hours. These durations vary with the algorithm, dataset, hardware, and concurrent load. Evaluation can run on either a GPU or a CPU. A suite covering the five evaluation tasks typically took 2–3 minutes on a GPU.

NeurIPS Paper Checklist

1. **Claims**
   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   Answer: [Yes]
   Justification: The abstract and introduction describe iterative latent refinement for offline GCRL value and critic backbones. Sections 5.2 and 5.5 support the empirical claims, and Section 6 discusses the underlying research questions.
2. **Limitations**
   Question: Does the paper discuss the limitations of the work performed by the authors?
   Answer: [Yes]
   Justification: Section 6 discusses how parameter sharing, optimization, and step conditioning shape latent dynamics. It identifies controlled comparisons and dynamical analyses as directions for understanding refinement depth and update capacity.
3. **Theory assumptions and proofs**
   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?
   Answer: [N/A]
   Justification: The contribution is empirical. Sections 3 and A specify the setting and objectives.
4. **Experimental result reproducibility**
   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?
   Answer: [Yes]
   Justification: Sections 4 and 5.1 describe the architecture and protocol. Appendices A–C give the objectives, datasets, hyperparameters, and replacement locations. Seed counts and uncertainty measures accompany the results.
5. **Open access to data and code**
   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?
   Answer: [Yes]
   Justification: The repository linked on the first page contains the backbone, dependencies, OGBench integration instructions, hyperparameters, and a training example. Datasets are available through OGBench [37].
6. **Experimental setting/details**
   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?
   Answer: [Yes]
   Justification: Section 5.1 and Appendices B–C specify the algorithms, datasets, optimization settings, goal sampling, evaluation settings, and backbone configurations.
7. **Experiment statistical significance**
   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?
   Answer: [Yes]
   Justification: Reported error bars and $\pm$ terms denote across-seed standard deviations. Main comparisons use eight seeds, while diagnostic, depth, capacity, ablation, and compute studies use four seeds unless stated otherwise.
8. **Experiments compute resources**
   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?
   Answer: [Yes]
   Justification: Appendix G reports GPU types, memory, and typical training and evaluation times. Appendix F reports parameter counts, estimated training FLOPs, and success at approximately matched training FLOPs.

9. **Code of ethics**
   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?
   Answer: [Yes]
   Justification: The study uses public benchmark datasets and simulation environments. It involves no human participants or private data.
10. **Broader impacts**
    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?
    Answer: [Yes]
    Justification: Learning from existing data can reduce costly environment interaction. Physical deployment requires validation of goal reaching and safety because inaccurate values can produce unsupported actions. Repeated latent updates also increase computational cost.
11. **Safeguards**
    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?
    Answer: [N/A]
    Justification: The work concerns simulation-based reinforcement learning and does not introduce high-risk pretrained models or scraped datasets.
12. **Licenses for existing assets**
    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?
    Answer: [Yes]
    Justification: OGBench datasets and the CRL, HIQL, and SAW reference methods are credited through their original papers. Code derived from OGBench retains its MIT license and copyright notice in the code package.
13. **New assets**
    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?
    Answer: [Yes]
    Justification: The linked repository documents the module, installation, integration with OGBench, hyperparameters, and a training example. No new dataset is introduced.
14. **Crowdsourcing and research with human subjects**
    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?
    Answer: [N/A]
    Justification: The study does not involve crowdsourcing or human participants.
15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**
    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?
    Answer: [N/A]
    Justification: The study does not involve human participants and requires no IRB approval.
16. **Declaration of LLM usage**
    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.
    Answer: [N/A]
    Justification: Large language models are not components of the proposed network, training objectives, or policy inference.