Recursive Value Learning for Long-Horizon Offline Goal-Conditioned RL

Anonymous authors
Paper under double-blind review

Abstract

Scaling offline goal-conditioned reinforcement learning (GCRL) to long-horizon tasks is difficult because (1) long-range value learning depends on shorter-range estimates that may still be inaccurate, and (2) max-based value backups can amplify overestimation through repeated propagation. We propose DCRL (Divide-and-Conquer RL), which recursively decomposes each trajectory segment into a balanced binary tree and trains the values from leaves to root. Each parent is therefore updated only after its children, using an exact factorization of the observed route rather than selecting among noisy alternatives. Since this objective learns values along demonstrated routes that are not necessarily optimal, DCRL jointly propagates values across trajectories to discover shorter routes. Thanks to the balanced binary tree, DCRL reduces worst-case bootstrap depth from linear to logarithmic, and this shorter dependency structure empirically corresponds to much slower error accumulation. Across diverse goal-reaching tasks, DCRL substantially outperforms prior flat offline GCRL methods, and on the five most challenging long-horizon OGBench tasks, it improves the best prior average score from 55 to 64, surpassing all evaluated flat and hierarchical baselines.

1 Introduction

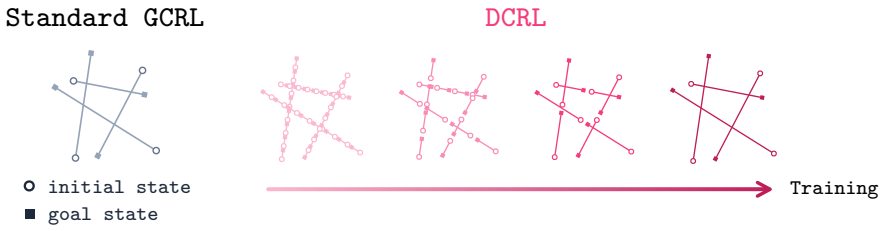


Figure 1. DCRL (right) learns short segments first, then uses them to learn longer ones, whereas standard GCRL (left) randomly samples pairs of all lengths in no particular order.

Long-horizon goal-conditioned value learning has a natural dependency structure: the value of a long trajectory segment is built from the values of its shorter constituent segments. Yet most offline goal-conditioned reinforcement learning (GCRL) methods form bootstrapped targets from independently sampled transitions or subgoals, without ensuring that shorter-horizon values are learned before the longer-horizon values that depend on them. A long-range estimate may therefore bootstrap from shorter-range values that are themselves still inaccurate. Moreover, standard Bellman-style and transitive value backups choose optimistically among actions or intermediate states (Bellman, 1966; Floyd, 1962). With finite offline data, this selection can favor overestimated candidates and propagate their errors through repeated backups, turning small local errors into large global inconsistencies.

We propose DCRL (Divide-and-Conquer RL), which makes this dependency explicit and enforces it through bottom-up training. The key idea is simple: learn shorter-range values (children) before the longer-range values (parents) that depend on them (Figure 1). Given a sampled trajectory segment, **Divide** recursively splits it at its *midpoint* until reaching single-step

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

segments. **Conquer** then trains a value function by processing the resulting segments from leaves to root. Thus, each parent is updated only after its children. A horizon-H segment therefore produces a balanced binary tree with $O(H)$ nodes but only $O(\log_2 H)$ sequential composition levels. This gives DCRL *logarithmic* bootstrap depth in the horizon, whereas standard backups have *linear* worst-case depth.

At each internal node of the tree, DCRL applies an exact trajectory factorization along the observed route. For any intermediate state w on a trajectory segment from s to g, the corresponding behavior value V_τ satisfies

$$V_\tau(s, g) = V_\tau(s, w) \cdot V_\tau(w, g).$$

Unlike standard max-based backups, this factorization evaluates a single route supported by the dataset. DCRL thus addresses the two sources of long-horizon error directly: bottom-up scheduling imposes the dependency order among value estimates, while exact route factorization avoids optimistic selection among noisy alternatives. We empirically show that DCRL accumulates less long-range error than standard max-based backups.

Explicit tree construction distinguishes DCRL from Transitive RL (TRL) (Park et al., 2026). TRL independently samples a trajectory segment and a random in-trajectory split at each update, thereby learning from isolated transitive decompositions. DCRL instead instantiates each sampled trajectory segment as a balanced binary tree, preserves its parent–child dependencies, and learns the value function on its segments from leaves to root. Thus, TRL samples individual transitive relations, whereas DCRL explicitly constructs and executes a divide-and-conquer computation. As illustrated in Figure 2, this structured learning order produces substantially more precise long-range value estimates than TRL.

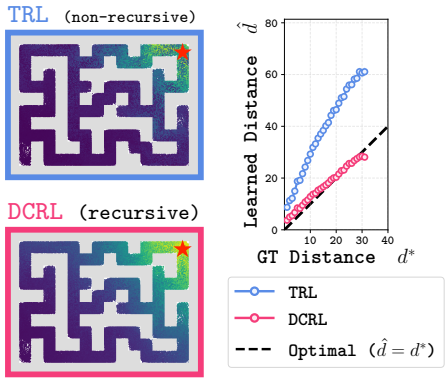


Figure 2. In humanoidmaze-giant, DCRL learns near-optimal distance estimates. We measure the distances predicted by DCRL and TRL from each state to a fixed goal (red star).

To preserve the ordering while maintaining diverse minibatches, DCRL interleaves multiple recursive trees through **slot scheduling**, which processes each tree bottom-up while mixing segment lengths within each minibatch.

Although recursive divide-and-conquer value learning mitigates error accumulation over the horizon, it recovers only behavior values. To recover optimal values, DCRL jointly trains the value function using multistep value propagation with goals globally relabeled from the dataset (Andrychowicz et al., 2017; Sutton & Barto, 2018). This propagation objective identifies shorter routes by combining segments from different trajectories. Thus, DCRL's two objectives serve complementary roles: recursive divide-and-conquer value learning provides reliable behavior routes, while multistep propagation recovers optimal routes by composing cross-trajectory routes.

In summary, our main contributions are threefold:

- **Algorithm.** We introduce DCRL, which explicitly instantiates divide-and-conquer value learning as a balanced binary tree over in-trajectory segments, trains each tree bottom-up using value factorization, and processes multiple trees in parallel through slot scheduling.
- **Analysis.** We prove that DCRL reduces worst-case bootstrap depth from *linear* to *logarithmic* in the horizon. In a controlled experiment, we show that this shorter dependency structure corresponds to substantially slower error accumulation.
- **Performance.** DCRL substantially outperforms prior flat offline GCRL algorithms across diverse goal-reaching tasks spanning different domains, horizons, and observation modalities, and surpasses evaluated hierarchical approaches on challenging long-horizon tasks.

2 Related Work

Goal-conditioned RL. Goal-conditioned RL aims to learn a policy that takes the optimal action from any state to reach a goal. We focus on the offline setting, which learns purely from a fixed dataset without online interaction. Prior work can be broadly categorized into Temporal Difference (TD) methods (Sutton & Barto, 2018; Haarnoja et al., 2018; Kostrikov et al., 2022; Park et al., 2025b), Monte Carlo (MC) methods (Sutton & Barto, 2018; Eysenbach et al., 2021; 2022), hierarchical methods (Barto & Mahadevan, 2003; Lynch et al., 2019; Park et al., 2023; 2025b), probabilistic methods (Eysenbach et al., 2021; Zheng et al., 2024), and quasimetric methods (Kaelbling, 1993; Dhiman et al., 2018; Jurgenson et al., 2020; Wang et al., 2023; Piękos et al., 2023; Myers et al., 2024; 2025; Park et al., 2026). `DCRL` builds on the quasimetric family, which we discuss next.

Quasimetrics in GCRL. In deterministic environments, GCRL reduces to learning a value function that estimates the shortest-path distance from any state to a goal. The induced optimal distance is a quasimetric (*i.e.*, an asymmetric distance obeying the triangle inequality), a property that GCRL algorithms exploit in three ways. "Explicit" methods impose the triangle inequality through quasimetric-constrained value networks (Wang et al., 2023; Myers et al., 2024; 2025). "Implicit" methods enforce it through triangle-inequality value backups (Kaelbling, 1993; Dhiman et al., 2018; Jurgenson et al., 2020; Piękos et al., 2023; Park et al., 2026). "Planning"-based methods use it to compose shortest paths at inference time (Eysenbach et al., 2019; Parascandolo et al., 2020; Jurgenson et al., 2020).

`DCRL` is most closely related to two "implicit" methods: Sub-goal tree dynamic programming (`TDP`) (Jurgenson et al., 2020) and Transitive RL (`TRL`) (Park et al., 2026). `TDP` recursively predicts intermediate subgoals to minimize decomposition cost and learns top-down. Greedy selection can overestimate the chosen decomposition. `DCRL` instead fixes the subgoal to the trajectory midpoint and learns bottom-up. `TRL` uses a similar in-trajectory backup, favoring optimistic splits via an upper-expectile loss. Because `TRL` learns values in no particular order, these optimistic updates rely on noisy estimates, amplifying overestimation through repeated propagation. This non-recursive update has *linear* worst-case bootstrap depth over long horizons. `DCRL` instead schedules a recursive midpoint decomposition with exact factorization, yielding *logarithmic* bootstrap depth and much slower empirical error growth.

3 Preliminaries

Problem Setting. We study GCRL in a deterministic controlled Markov process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mu, p, \gamma)$ comprising the state space, action space, initial-state distribution, deterministic transition function, and discount factor. Any state can be a goal. We are given an unlabeled dataset $\mathcal{D} = \{\tau^{(i)}\}_{i=1}^{N}$, where the i-th trajectory $\tau^{(i)} = (s_0^{(i)}, a_0^{(i)}, s_1^{(i)}, \dots, s_{H_i}^{(i)})$ has length H_i.

Following prior work (Wang et al., 2023; Park et al., 2026), we adopt the hitting-time formulation of the goal-conditioned value function: the agent receives a reward of 1 upon first reaching g and enters an absorbing state, so $V^\pi(s, g) = \mathbb{E}_\pi\left[\gamma^{T_g} \mid s_0 = s\right]$, where T_g is the first hitting time of g under a goal-conditioned policy π and $\gamma \in (0, 1)$ is the discount factor. The optimal value function is $V^* := \max_\pi V^\pi$. In deterministic environments, learning V^* is equivalent to learning the optimal distance $d^* := \log_\gamma V^*$, since $V^* = \gamma^{d^*}$. Here $d^*(s, g)$ is the minimum number of steps to reach g from s (*i.e.*, the shortest-path distance).

Triangle Inequality in GCRL. The optimal distance d^* satisfies the triangle inequality for all states $s, w, g \in \mathcal{S}$:

$$d^*(s, g) \;\le\; d^*(s, w) + d^*(w, g), \tag{1}$$

with equality when w lies on a shortest path from s to g. Through the relation $V^* = \gamma^{d^*}$, this is equivalent to a multiplicative triangle inequality on values:

$$V^*(s, g) \;\ge\; V^*(s, w) \cdot V^*(w, g). \tag{2}$$

Prior methods turn this property into a transitive backup $V(s, g) \leftarrow \max_w V(s, w) \cdot V(w, g)$ to learn the value function by maximizing over intermediate states w (Kaelbling, 1993; Dhiman et al., 2018; Jurgenson et al., 2020; Piękos et al., 2023; Park et al., 2026).

4 Recursive Value Learning for Long-Horizon Offline GCRL

Goal-conditioned value learning with standard max-based backups (*e.g.*, TD and transitive backups) suffers from error accumulation. While every long-range value inherently depends on shorter-range values, standard GCRL methods do not leverage this dependency structure. We address this by framing goal-conditioned value learning as a divide-and-conquer problem. We propose `DCRL` (Divide-and-Conquer RL), which recursively splits a trajectory segment and learns values from shorter segments to longer ones. Thus, `DCRL` exhibits far slower empirical error accumulation than standard backups and achieves strong goal-reaching performance.

In Section 4.1, we introduce recursive goal-conditioned value learning, which learns shorter-horizon (child) values before longer-horizon (parent) values. In Section 4.2, we describe the selection-free update rule of recursive value learning, which avoids maximizing over noisy intermediate states. Since this update rule is restricted to observed routes, it recovers only behavior values. To learn optimal values, `DCRL` also trains the same value function to compose segments across trajectories (Section 4.3). In Sections 4.4 and 4.5, we detail the practical implementation of `DCRL`.

4.1 Recursive Divide-and-Conquer Value Learning

We introduce a recursive divide-and-conquer strategy to enforce child-to-parent ordering in goal-conditioned value learning. `DCRL` recursively splits each trajectory segment (`Divide`) and learns values from leaves to root (`Conquer`). For a trajectory segment between randomly sampled states s_i and s_j, where $i < j$, we recursively bisect the segment at the midpoint $k = \lfloor (i+j)/2 \rfloor$ to build a balanced binary tree (Figure 3). By splitting at the midpoint, a horizon-H segment yields $O(H)$ tree nodes but only $O(\log_2 H)$ sequential composition levels.

Divide. For a root segment (s_i, s_j), we recursively split each segment into two children, (s_i, s_k) and (s_k, s_j), where $k = \lfloor (i+j)/2 \rfloor$, until reaching single-step leaves (*e.g.*, (s_i, s_{i+1})).

Conquer. We train child segments before their parents. Single-step leaves are grounded (*i.e.*, $V(s_i, s_{i+1}) = \gamma$), and each parent is then updated from its previously trained children.

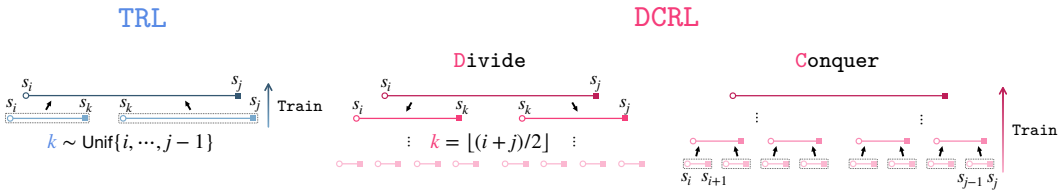


Figure 3. `DCRL` explicitly implements *divide-and-conquer*. `Divide`: Split a root segment (s_i, s_j) at its midpoint, then recursively split each half until reaching single-step leaves. `Conquer`: Train the value function on leaves first and then on their parents, proceeding up to the root. In contrast, `TRL` backs up from a single random subgoal s_k, where $k \in \{i, \ldots, j-1\}$.

4.2 Update Rule for Recursive Value Learning

For recursive value learning, we seek a selection-free update that avoids amplifying errors through maximization over noisy intermediate states. Unlike max-based backups, our update rule factorizes the value of a trajectory segment between s and g at its midpoint w:

$$V_\tau(s, g) = V_\tau(s, w) \cdot V_\tau(w, g). \tag{3}$$

Because Eq. (3) uses a fixed intermediate state without maximization, V_τ learns behavior values (Gulcehre et al., 2021). The corresponding distance $d_\tau(s, g) = d_\tau(s, w) + d_\tau(w, g)$ is the length of the trajectory's own route through w. It upper-bounds d^*: the trajectory's route need not be the shortest path, but it is supported by the data. This behavior value function V_τ admits a useful route-suboptimality decomposition. We prove the following:

Property 1. *Let $d_\tau := \log_\gamma V_\tau$, with V_τ satisfying the exact factorization in* Eq. (3)*, and define the route suboptimality $e(s, g) := d_\tau(s, g) - d^*(s, g)$. Then,*

$$e(s, g) \geq e(s, w) + e(w, g),$$

for ***any*** *intermediate state w on a trajectory segment from s to g (proof in Appendix A.1).*

Property 1 shows that route suboptimality can grow under composition: a parent route is at least as suboptimal as its two children combined. This motivates learning long-horizon behavior values recursively from shorter, less-suboptimal constituents. We therefore learn child values $(V_\tau(s, w), V_\tau(w, g))$ before using them to update their parent $(V_\tau(s, g))$, avoiding bootstrapping from unconverged child estimates. For recursive value learning (Section 4.1), we choose w as the segment midpoint, yielding a balanced binary tree.

4.3 Value Propagation for Optimality

Recursive divide-and-conquer value learning yields only in-trajectory behavior values (V_τ). To recover optimal values (V^*), we additionally train the value function with multistep value-propagation. Figure 4 illustrates a didactic example of the discrepancy between V_τ and V^*, confirming that the optimal value often requires composing segments across trajectories. Furthermore, the distribution of recursive tree samples is heavily skewed toward short segments. Decomposing a horizon-H root segment yields $2H - 1$ tree nodes, of which H are leaves. Models trained only on this distribution generalize poorly to distant relabeled goals.

Thus, we optimize our value function with an additional objective using i.i.d. samples with globally relabeled goals. Starting from the original triangle inequality (Eq. (2)), we fix the intermediate subgoal w to a future in-trajectory state s_{i+n} with propagation horizon n:

$$V^*(s_i, g) \geq \gamma^n \cdot V^*(s_{i+n}, g). \tag{4}$$

Because s_{i+n} lies n steps ahead on a demonstrated trajectory, $V^*(s_i, s_{i+n}) \geq \gamma^n$. Eq. (4) motivates pushing $V(s_i, g)$ up toward $\gamma^n \cdot V(s_{i+n}, g)$, which is equivalent to multistep propagation (Sutton & Barto, 2018). We optimize this objective using an upper-expectile loss because we seek the shortest route connecting s_i to g.

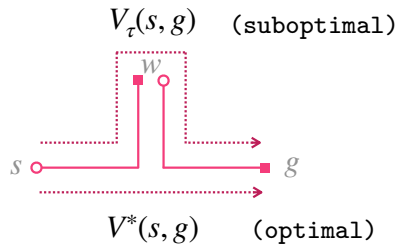


Figure 4. Optimal values require composing segments across trajectories. Given $s \to w$ and $w \to g$, the optimal route from s to g cannot be recovered from V_τ alone.

Importantly, divide-and-conquer supports n-step propagation by quickly learning long-range behavior values through a shallow dependency path. For a horizon-H segment, divide-and-conquer has $O(\log_2 H)$ bootstrap depth, whereas n-step propagation has $O(H/n)$ depth. Long-range behavior values therefore emerge through fewer stages, giving a reliable basis for propagation.

Division of Labor. The divide-and-conquer (Section 4.1) and propagation (Section 4.3) objectives train a shared value function but serve complementary roles: the former learns behavior values for within-trajectory pairs through a shallow dependency structure, whereas the latter composes segments across trajectories to recover optimal values. In Table 3, removing either objective degrades performance, confirming both roles are essential to `DCRL`.

4.4 Implementation: Parallel Slot Scheduling

We introduce **slot scheduling** to efficiently execute recursive divide-and-conquer value learning. Training on one tree at a time would be slow and produce highly correlated batches with limited trajectory and horizon diversity. To process multiple root segments concurrently, we maintain multiple independent slots. As illustrated in Figure 5 and Algorithm 1, each slot manages the recursive tree of a single root segment. A slot $\mathcal{T}$ is initialized as a FIFO queue of recursive samples ordered from leaves to root. Samples dequeued from $\mathcal{T}$ are processed from leaves to root, enforcing our principle: every child is scheduled before its parent.

During training, we maintain $S = 128$ slots $\{\mathcal{T}_\ell\}_{\ell=1}^{S}$ (for batch size 1024), drawing samples uniformly across them at each step. Because slots drain and refill independently, they become

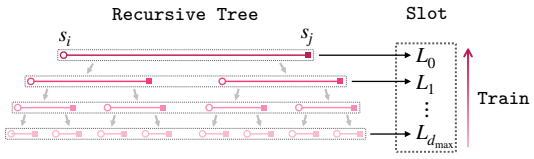


Figure 5. A **recursive tree** is flattened into a slot, ordered from leaves ($L_{d_{\max}}$) to the root (L_0), so every child is learned before its parent.

Algorithm 1 `SampleSlot`($\mathcal{D}$)

1: Initialize per-depth lists $\{L_d\}_{d=0}^{d_{\max}}$
2: Sample $\tau = (s_0, a_0, s_1, \ldots, s_H) \sim \mathcal{D}$
3: Sample $i < j \sim \mathrm{Unif}\{0, \ldots, H\}$
4: Build a **recursive tree** with $(s_i, \ldots, s_j)$
5: **return** `Queue`$(L_{d_{\max}}, \ldots, L_0)$

desynchronized, so later batches span a range of lengths while preserving child-to-parent ordering within each slot. We study how the number of slots (S) affects `DCRL`'s performance in Appendix H.3 and compare `DCRL` with other divide-and-conquer strategies in Section 5.3.

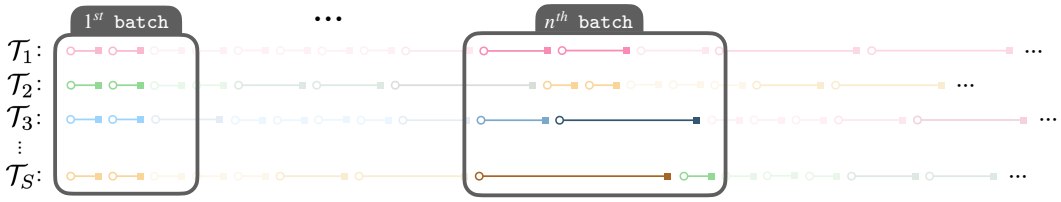


Figure 6. Maintaining S independent **slots** $(\mathcal{T}_1, \ldots, \mathcal{T}_S)$ enables (i) learning child segments before their parents and (ii) mixing diverse sample lengths within batches. Each slot begins by emitting single-step leaves, as illustrated by the 1st batch; as slots drain and refill at different rates, they become desynchronized, mixing short-horizon and long-horizon segments, as in the n^{th} batch. Each slot schedules a new tree once its current one is consumed.

4.5 Implementation: Critic Learning

In practice, we train the action-value function $Q(s, a, g) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ by minimizing a divide-and-conquer loss and a propagation loss simultaneously:

$$\mathcal{L}^{\texttt{DCRL}}(Q) = \mathcal{L}^{\texttt{dc}}(Q) + \mathcal{L}^{\texttt{prop}}(Q). \tag{5}$$

Divide-and-Conquer Loss. $\mathcal{L}^{\texttt{dc}}(Q)$ learns behavior values from recursive tree samples τ_{DC} using the exact value factorization:

$$\mathcal{L}^{\texttt{dc}}(Q) = \mathbb{E}_{\tau_{\text{DC}} \sim \boldsymbol{\mathcal{T}}}\Big[\mathcal{L}_{\text{BCE}}\big(Q(s_i, a_i, s_j),\ \texttt{sg}[Q(s_i, a_i, s_k)]\, \bar{Q}(s_k, a_k, s_j)\big)\Big], \tag{6}$$

where $k = \lfloor (i + j)/2 \rfloor$, $\texttt{sg}$ denotes stop-gradient, $\bar{Q}$ is the EMA target critic, and $\mathcal{L}_{\text{BCE}}$ is the binary cross-entropy loss. To obtain each divide-and-conquer sample τ_{DC}, we uniformly select a slot $\mathcal{T} \in \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$ and dequeue its next sample. We evaluate the first child value $Q(s_i, a_i, s_k)$ using the online critic and the second child value $\bar{Q}(s_k, a_k, s_j)$ using the EMA critic. Single-step samples use target γ; single-step child values are likewise replaced by γ. Appendix H.1 ablates the critic used for each child (online or EMA) and finds that using the online critic for the first and the EMA critic for the second (*i.e.*, $\texttt{sg}[Q] \cdot \bar{Q}$) performs best.

Propagation Loss. $\mathcal{L}^{\texttt{prop}}(Q)$ learns optimal values from globally relabeled samples τ_{prop} using multistep propagation:

$$\mathcal{L}^{\texttt{prop}}(Q) = \mathbb{E}_{\tau_{\text{prop}} \sim \boldsymbol{\mathcal{D}}}\Big[\mathcal{L}^{\kappa_{\texttt{prop}}}_{\text{BCE}}\big(Q(s_i, a_i, g),\ \gamma^n \cdot \bar{Q}(s_{i+n}, a_{i+n}, g)\big)\Big], \tag{7}$$

where $\mathcal{L}^{\kappa_{\texttt{prop}}}_{\text{BCE}}$ is the BCE loss with expectile $\kappa_{\texttt{prop}} \in (0.5, 1)$, and g is a future, random, or current state. For an in-trajectory goal $g = s_j$ within n steps of s_i ($j - i \le n$), we use γ^{j-i} instead of $\gamma^n \cdot \bar{Q}$. The propagation horizon n is task-specific and studied in Section 5.2.

The two losses differ by design: $\mathcal{L}^{\texttt{dc}}$ learns behavior routes with plain BCE ($\kappa_{\texttt{dc}} = 0.5$), whereas $\mathcal{L}^{\texttt{prop}}$ favors shorter routes through upper-expectile BCE ($\kappa_{\texttt{prop}} > 0.5$), since the optimal value requires the shortest of many routes. We ablate $\kappa_{\texttt{dc}}$ and $\kappa_{\texttt{prop}}$ in Appendix H.2; performance peaks at $\kappa_{\texttt{dc}} = 0.5$, consistent with optimistic weighting introducing bias when learning behavior routes. Appendix B summarizes the full training procedure for `DCRL`.

5 Experiments

Our experiments and analysis address three questions:

- **Q1:** How does DCRL compare with prior offline GCRL methods across diverse task domains, horizons, dataset scales, and observation modalities? (→ Section 5.1)
- **Q2:** How does DCRL scale with the propagation horizon n and the task horizon H compared with standard Bellman-style and transitive backups? (→ Section 5.2)
- **Q3:** How does DCRL compare with alternative divide-and-conquer strategies in terms of goal-reaching performance and training efficiency? (→ Section 5.3)

We compare DCRL against prior offline GCRL algorithms spanning four value-learning families:

- **Temporal Difference**: IQL (Kostrikov et al., 2022), IVL (Kostrikov et al., 2022; Park et al., 2023), SAC+BC (Haarnoja et al., 2018), TD, TD-n (Sutton & Barto, 2018)
- **Monte Carlo**: CRL (Eysenbach et al., 2022), MC (Tian et al., 2021; Shah et al., 2021)
- **Hierarchical**: POR (Xu et al., 2022), HIQL (Park et al., 2023), SHARSA (Park et al., 2025b)
- **Triangle Inequality**: QRL (Wang et al., 2023), TMD (Myers et al., 2025), TDP (Kaelbling, 1993; Jurgenson et al., 2020), COE (Piękos et al., 2023), TRL (Park et al., 2026)

We present 95% confidence intervals in plots and standard deviations in tables. In tables, mean scores at least 95% of each row's best are marked in red. Descriptions of each baseline, experimental details, and full results are provided in Appendices C, E, and J, respectively.

5.1 How Well Does DCRL Compare with Prior Offline GCRL Methods?

We evaluate DCRL and prior offline GCRL methods on goal-reaching tasks spanning diverse domains, horizons, dataset scales, and observation modalities (state and pixel). First, Table 1 compares these methods on the five most challenging long-horizon tasks from OGBench (Park et al., 2025a). DCRL achieves the best average score, outperforming all flat and hierarchical baselines. Notably, DCRL is the only flat method to achieve nonzero success on `cube-octuple`, while attaining the highest scores on `humanoidmaze-giant` and both puzzle tasks.

Table 1. DCRL achieves the best average performance on **long-horizon** tasks.

		Temporal Difference				Monte Carlo		Hierarchical		Triangle Inequality				
Environment	FBC	IQL	SAC+BC	TD	TD-n	CRL	MC	HIQL	SHARSA	QRL	TDP	COE	TRL	DCRL
humanoidmaze-giant-1B	0 ± 0	3 ± 4	5 ± 0	3 ± 2	78 ± 4	62 ± 42	79 ± 4	25 ± 6	42 ± 3	3 ± 2	2 ± 0	2 ± 1	54 ± 5	93 ± 3
puzzle-4x5-1B	0 ± 0	20 ± 0	19 ± 1	19 ± 1	92 ± 8	1 ± 1	47 ± 7	12 ± 5	90 ± 4	0 ± 0	0 ± 0	0 ± 0	97 ± 1	100 ± 0
puzzle-4x6-1B	0 ± 0	17 ± 2	11 ± 8	13 ± 3	53 ± 11	0 ± 0	37 ± 4	4 ± 3	55 ± 3	0 ± 0	0 ± 0	0 ± 0	51 ± 5	87 ± 5
cube-quadruple-100M	1 ± 0	41 ± 3	39 ± 4	20 ± 2	21 ± 6	22 ± 2	1 ± 1	62 ± 6	74 ± 2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	37 ± 4
cube-octuple-1B	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	15 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	5 ± 1
Average	0	16	15	11	49	17	33	21	55	1	0	0	40	64

Next, Figure 7 compares aggregate performance on standard-horizon OGBench tasks (10 state-based and 10 pixel-based). DCRL performs best in both modalities.

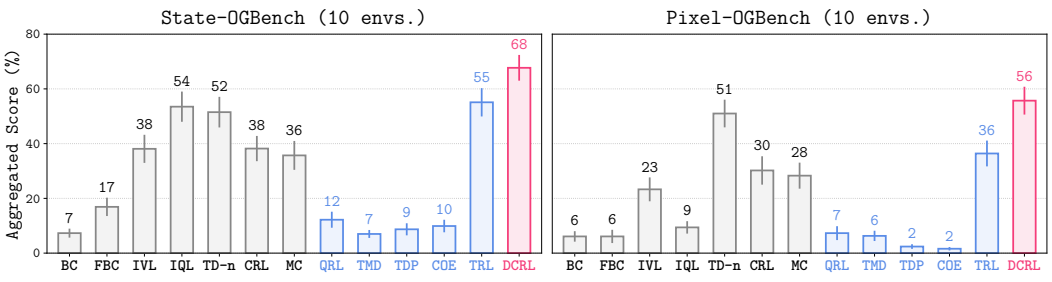


Figure 7. DCRL achieves state-of-the-art performance on standard OGBench tasks across diverse task **domains** and observation **modalities**. Full results are in Tables 15 and 16.

Finally, we test DCRL and prior methods on CALVIN (Mees et al., 2022), a compositional, long-horizon manipulation environment. We use the in-domain setup, where four subtasks must be solved in sequence using 1,239 task-agnostic trajectories spanning 34 subtasks. Figure 8 reports `Return` (subtasks completed), `Completion` (full-task success), and task progress. In the state setting, DCRL is the only flat method that completes all four subtasks.

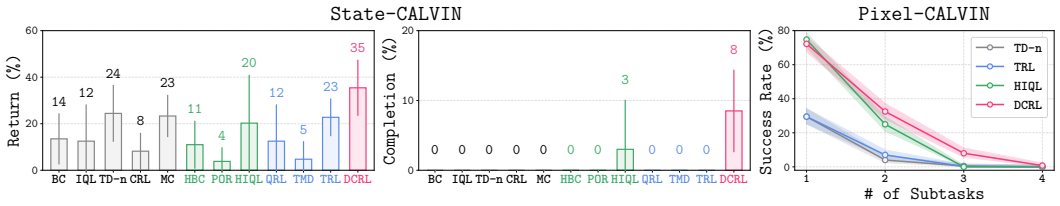


Figure 8. DCRL achieves the highest four-subtask completion rate on state-based CALVIN (8%), versus 3% for the best hierarchical baseline and 0% for all flat baselines. On pixel-based CALVIN, which excludes privileged object poses, DCRL reaches at least three subtasks in 8% of rollouts, but full-sequence completion remains rare (1% for DCRL and 0% for all baselines).

5.2 Is DCRL Better Than TD and TRL Across Propagation and Task Horizons?

We first study how the propagation horizon n affects DCRL's performance. While DCRL relies on value propagation to recover optimal values, this mechanism can also be applied to TRL. We therefore compare DCRL-n against TD-n and TRL-n (TRL with n-step propagation) across a range of horizons n, isolating the effect of DCRL's recursive value learning.

Figure 9 (left) shows the results on two long-horizon tasks across $n \in \{1, 5, 10, 25, 50, 100\}$. DCRL-n achieves the best performance at larger n (≥ 25) on both tasks, while TD-n performs especially poorly on the puzzle task and TRL-n struggles on the cube-manipulation task.

At $n \in \{1, 5\}$, DCRL-n performs similarly to TD-n and worse than TRL-n. With small n, DCRL-n's propagation chain is long ($O(H/n)$), so it cannot fully leverage its recursively learned long-horizon behavior values to recover optimal values. For $n \geq 25$, DCRL-n maintains a clear margin over both baselines. See Figure 12 for the full comparison.

To understand how error accumulation scales with task horizon H, we analyze each method's bootstrap depth. In Appendix A.2.1, we prove that DCRL-n has **logarithmic** bootstrap depth in H, whereas TD-n and TRL-n have **linear** worst-case depth. We then test whether this shallower dependency structure yields slower error growth under a fixed training budget.

For this controlled comparison, we use a discrete environment (`combination-lock`) (Park et al., 2025b) to measure error accumulation. In Figure 9 (right), we report mean long-range Q-error ($|\hat{Q} - Q^*|$) across task horizons $H \in \{256, \ldots, 2048\}$ at a fixed $n = 64$. DCRL-n remains relatively flat while TD-n and TRL-n accumulate error rapidly. Appendix A.2.2 further separates the effects of n-step propagation, expectile selection, and recursive scheduling.

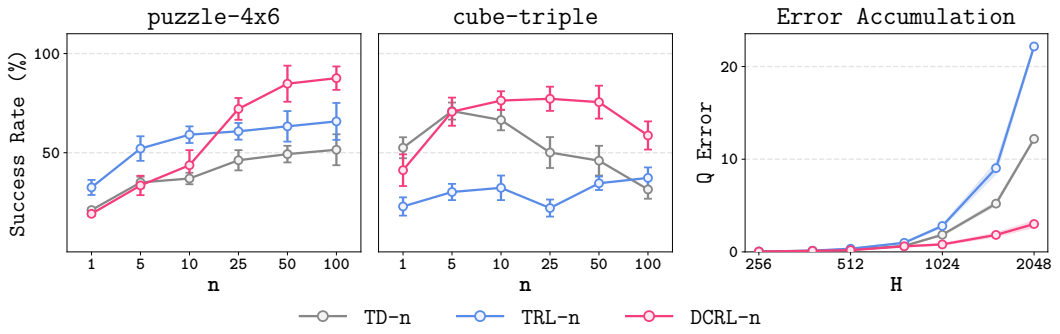


Figure 9. Left: DCRL-n surpasses TD-n and TRL-n at long **propagation horizons** ($n \geq 25$). Right: Q-error in DCRL-n grows far more slowly with **task horizon** H than in the others.

5.3 Comparison with Alternative Divide-and-Conquer Strategies

We compare DCRL with alternative strategies for organizing or modifying divide-and-conquer value learning. Descriptions and more experiments are in Appendices D and I, respectively.

- Curriculum (Δ): learns values on short segments first using H/Δ stages.
- TRL (λ): up-weights short state–goal pairs and down-weights long ones in the value loss.
- DCRL (w/ random split): uses a random in-trajectory subgoal rather than the midpoint.
- DCRL (w/ reverse order): learns recursive values in reverse (parent-to-child) order.
- DCRL (w/o recursion): replaces recursion with i.i.d. sampling and adds one-step grounding.
- DCRL (w/o propagation): uses recursive value learning alone to recover optimal values.
- DCRL (w/ max-based backup): uses $V(s,g) \leftarrow \max_w V(s,w) \cdot V(w,g)$ to select subgoals.

Table 2. Comparison of divide-and-conquer strategies in humanoidmaze-giant.

Method	Success ↑	Throughput ↑
Curriculum ($\Delta = 25$)	28 ± 7	96 ± 2
Curriculum ($\Delta = 50$)	39 ± 8	99 ± 4
Curriculum ($\Delta = 100$)	29 ± 5	99 ± 2
TRL ($\lambda = 0$)	54 ± 5	110 ± 1
TRL ($\lambda = 0.01$)	53 ± 2	107 ± 1
TRL ($\lambda = 0.1$)	52 ± 3	106 ± 2
DCRL (w/ random split)	92 ± 1	66 ± 1
DCRL (w/ reverse order)	18 ± 6	82 ± 4
DCRL	93 ± 3	84 ± 3

Table 3. Ablation studies of DCRL on state-based OGBench (10 environments).

Method	Success ↑	Throughput ↑
DCRL	68 ± 2	193 ± 3
DCRL (w/o recursion)	39 ± 3	173 ± 1
DCRL (w/o propagation)	45 ± 2	201 ± 5
DCRL (w/ max-backup)	62 ± 2	157 ± 2

Table 2 reports success and throughput (gradient steps per second) at a fixed batch size of 1024. DCRL achieves over 1.7× TRL's success rate at 24% lower throughput. Random splitting creates deeper, less balanced trees and has lower throughput, but performance remains unchanged, likely because exact factorization and child-to-parent scheduling are retained. Reversing the learning order reduces success from 93% to 18%, consistent with parents bootstrapping from not-yet-updated child estimates. Curriculum learning is highly sensitive to Δ, and its best performance trails DCRL and TRL. Finally, TRL's distance re-weighting does not improve the unweighted baseline.

Table 3 ablates DCRL's recursion, propagation, and value-update rule. Removing recursion causes the largest performance drop, highlighting the recursive tree's central role. Removing propagation sharply degrades performance, showing that one optimistic divide-and-conquer objective cannot replace separate behavior-value learning and optimal-value propagation. Max-based backups underperform exact factorization, consistent with selection-induced overestimation, and lower throughput due to subgoal-search overhead. Overall, these ablations validate DCRL's division of labor: recursive factorization learns reliable behavior values, while propagation recovers optimality.

6 Conclusion

We introduce DCRL, which exploits a natural dependency structure in goal-conditioned value learning: longer-segment values are built from shorter ones. DCRL realizes recursive value learning through a balanced binary tree over in-trajectory segments, learns behavior values from leaves to root, and propagates them across trajectories toward optimality. Across offline GCRL benchmarks, DCRL substantially outperforms prior flat methods and surpasses evaluated hierarchical baselines on challenging long-horizon tasks. By mitigating horizon-induced error accumulation, DCRL provides a foundation for scaling GCRL to longer horizons.

Limitations. Like other triangle-inequality methods, DCRL assumes deterministic dynamics. Under stochastic dynamics, exact factorization generally fails because the expectation of a product need not factorize (*i.e.*, $\mathbb{E}[XY] \neq \mathbb{E}[X] \cdot \mathbb{E}[Y]$). Additionally, hierarchical approaches outperform DCRL on cube-manipulation tasks, despite these tasks having shorter horizons than humanoidmaze-giant, where DCRL excels. Thus, their difficulty does not stem solely from horizon length. Extending recursive sampling to stochastic dynamics and understanding when hierarchy remains beneficial are important directions for future work.

AI use statement

In this work, we used generative AI tools to help develop theoretical models and conceptual frameworks, provide critical ingredients for proving mathematical claims, and assist in writing proofs. We did not use generative AI tools to formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, support qualitative or thematic data analysis, or interpret results. Generating synthetic datasets and cleaning or reformatting datasets were not applicable to this work. Additionally, we used generative AI tools to summarize or analyze existing literature, search for information, improve the readability of the paper, and identify relevant literature. We reviewed all AI-assisted work, verified all prior work and references suggested by the LLM, and checked every LLM-refined sentence. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

Reproducibility statement

We provide our source code and instructions for reproducing the reported results in the supplementary code archive. Appendix B presents the full algorithm, and Appendix K reports all experimental hyperparameters. Appendix A provides detailed theoretical analysis and empirical validation, including assumptions and proofs, while Appendix E describes the experimental setup.

References

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 (NeurIPS)*, 2017.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

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

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems (RSS)*, 2023.

Luc Devroye. A note on the height of binary search trees. *Journal of the ACM*, 33(3): 489–498, 1986.

Vikas Dhiman, Shurjo Banerjee, Jeffrey M Siskind, and Jason J Corso. Floyd–Warshall reinforcement learning: Learning from past experiences to reach new goals. *arXiv preprint arXiv:1809.09318*, 2018.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning (ICML)*, 2018.

Ben Eysenbach, Russ Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

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

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*, 2018.

Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345–345, 1962.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

Caglar Gulcehre, Sergio Gómez Colmenarejo, Ziyu Wang, Jakub Sygnowski, Thomas Paine, Konrad Zolna, Yutian Chen, Matthew Hoffman, Razvan Pascanu, and Nando de Freitas. Regularized behavior value estimation. *arXiv preprint arXiv:2103.09575*, 2021.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

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

Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. IDQL: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. Sub-goal trees – a framework for goal-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

Leslie Pack Kaelbling. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1993.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

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

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning (CoRL)*, 2019.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7327–7334, 2022.

Vivek Myers, Chongyi Zheng, Anca Dragan, Sergey Levine, and Benjamin Eysenbach. Learning temporal distances: Contrastive successor features can provide a metric structure for decision-making. In *International Conference on Machine Learning (ICML)*, 2024.

Vivek Myers, Bill Zheng, Benjamin Eysenbach, and Sergey Levine. Offline goal-conditioned reinforcement learning with quasimetric representations. In *Neural Information Processing Systems (NeurIPS)*, 2025.

Giambattista Parascandolo, Lars Buesing, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica B Hamrick, Nicolas Heess, Alexander Neitz, and Theophane Weber. Divide-and-conquer Monte Carlo tree search for goal-directed planning. *arXiv preprint arXiv:2004.11410*, 2020.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. HIQL: Offline goal-conditioned RL with latent states as actions. In *Neural Information Processing Systems (NeurIPS)*, 2023.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025a.

Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes RL scalable. In *Neural Information Processing Systems (NeurIPS)*, 2025b.

Seohong Park, Aditya Oberai, Pranav Atreya, and Sergey Levine. Transitive RL: Value learning via divide and conquer. In *International Conference on Learning Representations (ICLR)*, 2026.

Piotr Piękos, Aditya Ramesh, Francesco Faccio, and Jürgen Schmidhuber. Efficient value propagation with the compositional optimality equation. In *NeurIPS 2023 Workshop on Goal-Conditioned Reinforcement Learning*, 2023.

Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. Rapid exploration for open-world navigation with latent goal models. In *Conference on Robot Learning (CoRL)*, 2021.

Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2018.

Stephen Tian, Suraj Nair, Frederik Ebert, Sudeep Dasari, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Model-based visual planning with self-supervised functional distances. In *International Conference on Learning Representations (ICLR)*, 2021.

Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal goal-reaching reinforcement learning via quasimetric learning. In *International Conference on Machine Learning (ICML)*, 2023.

Haoran Xu, Li Jiang, Li Jianxiong, and Xianyuan Zhan. A policy-guided imitation approach for offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2022.

Chongyi Zheng, Ruslan Salakhutdinov, and Benjamin Eysenbach. Contrastive difference predictive coding. In *International Conference on Learning Representations (ICLR)*, 2024.

A Theoretical Analysis

A.1 Proof of Property 1

Proof.

$$\begin{aligned} d^*(s,g) + e(s,g) &= d_\tau(s,g) \\ &= d_\tau(s,w) + d_\tau(w,g) \\ &= [d^*(s,w) + e(s,w)] + [d^*(w,g) + e(w,g)] \\ &= [d^*(s,w) + d^*(w,g)] + [e(s,w) + e(w,g)] \\ &\geq d^*(s,g) + e(s,w) + e(w,g) \end{aligned}$$

Therefore, we have $e(s,g) \geq e(s,w) + e(w,g)$. □

A.2 Error Accumulation Analysis

We analyze error accumulation theoretically and empirically. The theoretical subsection (Appendix A.2.1) formalizes backup dependencies through bootstrap depth and derives upper error bounds for `TD-n`, `TRL-n`, and `DCRL-n`, highlighting the effects of selection-free factorization and child-to-parent scheduling. The empirical subsection (Appendix A.2.2) tests these predictions in a controlled `combination-lock` environment by measuring long-range error across horizons and ablating the recursive schedule.

A.2.1 Theoretical Validation

We compare `DCRL-n` with Bellman-style and transitive-backup methods (`TD-n`, `TRL-n`). Both `DCRL-n` and `TRL-n` include the same additional n-step propagation objective, whereas vanilla `TRL` uses only its transitive objective.

For the dependency analysis, each sample $p = (s_i, s_j)$ denotes the critic value associated with (s_i, a_i, s_j), where a_i is the dataset action at s_i. We suppress this action argument and write $Q_\bullet(p) := Q_\bullet(s_i, a_i, s_j)$. For $\gamma \in (0,1)$, define the corresponding distance estimate

$$\hat{d}_\bullet(p) := \log_\gamma Q_\bullet(p), \qquad e_\bullet(p) := \hat{d}_\bullet(p) - d^*(p),$$

where $d^*(p) := \log_\gamma Q^*(s_i, a_i, s_j)$ is the optimal distance conditioned on first taking the dataset action a_i. For a fixed trajectory $\tau^{(\ell)}$, we suppress the trajectory superscript and let $\mathcal{P} = \{(s_i, s_j) : 0 \leq i \leq j \leq H_\ell\}$ denote the set of in-trajectory pairs, with $\mathcal{P}_\circ \subset \mathcal{P}$ denoting the grounded pairs. For the structural analysis, pairs within n steps are treated as grounded because their targets require no further bootstrapping: `TD-n` grounds them through n-step returns, while the shared propagation objective provides the same grounding to `DCRL-n` and `TRL-n`. Vanilla `TRL` grounds only one-step pairs. The error bounds below additionally idealize these grounded targets as exact relative to d^*; the bootstrap-depth results require only that recursion terminates at them.

A possibly stochastic backup operator $B : \mathcal{P} \setminus \mathcal{P}_\circ \to 2^{\mathcal{P}}$ maps each non-grounded sample $p = (s, g)$ to the learned constituents in its bootstrap target:

$$B_{\texttt{TD-n}}(s_i, g) = \{(s_{i+n}, g)\}, \qquad B_{\texttt{DCRL-n}}(s_i, s_j) = B_{\texttt{TRL-n}}(s_i, s_j) = \{(s_i, s_k), (s_k, s_j)\},$$

where $k = \lfloor (i+j)/2 \rfloor$ for `DCRL-n` and $k \sim \mathrm{Unif}\{i+1, \ldots, j-1\}$ for `TRL-n`. `TRL`'s original sampling also permits $k = i$; because this yields an identity decomposition that does not advance the dependency graph, we condition on the effective nontrivial backup $i < k < j$.

Definition 1. *For a root sample p and a realization ω of all stochastic backup choices, the* **realized dependency graph** *$G_B(p;\omega)$ is obtained by recursively adding an edge (p', q) for every $q \in B(p';\omega)$, stopping when $q \in \mathcal{P}_\circ$. For `TRL-n`, this graph represents the recursive unrolling of successive random backups, not an explicitly stored tree.*

Definition 2. *The realized bootstrap depth $\mathrm{D}_B(p;\omega)$ is the longest path from p to a grounded leaf in $G_B(p;\omega)$. For a stochastic operator, define*

$$\mathrm{D}_B^{\mathrm{avg}}(p) := \mathbb{E}_\omega[\mathrm{D}_B(p;\omega)], \qquad \mathrm{D}_B^{\max}(p) := \sup_\omega \mathrm{D}_B(p;\omega).$$

For deterministic operators, we simply write $\mathrm{D}_B(p)$.

Proposition 1. *Given a non-grounded sample $p = (s_i, s_j)$ with $H := j - i$ and fixed n,*

$$\begin{aligned} \mathrm{D}_{\texttt{TD-n}}(p) &= \lceil H/n \rceil - 1, \\ \mathrm{D}_{\texttt{DCRL-n}}(p) &= \lceil \log_2(H/n) \rceil, \\ \mathrm{D}^{\mathrm{avg}}_{\texttt{TRL-n}}(p) &= \Theta(\log(H/n)), & \mathrm{D}^{\max}_{\texttt{TRL-n}}(p) &= \Theta(H - n), \\ \mathrm{D}^{\mathrm{avg}}_{\texttt{TRL}}(p) &= \Theta(\log H), & \mathrm{D}^{\max}_{\texttt{TRL}}(p) &= \Theta(H). \end{aligned}$$

Proof sketch. Let $\mathrm{D}_\bullet(L)$ denote the bootstrap depth of a segment of length L, with $\mathrm{D}_\bullet(L) = 0$ for $L \le n$.

For `TD-n`, each backup reduces the remaining length by n, so

$$\mathrm{D}_{\texttt{TD-n}}(L) = 1 + \mathrm{D}_{\texttt{TD-n}}(L - n),$$

which gives $\mathrm{D}_{\texttt{TD-n}}(H) = \lceil H/n \rceil - 1$.

For `DCRL-n`, midpoint splitting yields

$$\mathrm{D}_{\texttt{DCRL-n}}(L) = 1 + \max\{\mathrm{D}_{\texttt{DCRL-n}}(\lfloor L/2 \rfloor), \mathrm{D}_{\texttt{DCRL-n}}(\lceil L/2 \rceil)\}\,.$$

After d levels, the longest remaining segment has length $\lceil H/2^d \rceil$. The smallest d for which this length is at most n is $d = \lceil \log_2(H/n) \rceil$.

For `TRL-n`, recursively choosing uniformly random splits yields the same partition structure as a random binary search tree, stopped when every segment has length at most n. For fixed n, this truncation does not change the asymptotic height, so standard random-tree results give expected depth $\Theta(\log(H/n))$ (Devroye, 1986). In the worst case, every split is adjacent to an endpoint, reducing the longest remaining segment by one per level. Hence, the maximum depth is $H - n = \Theta(H - n)$. Vanilla `TRL` stops only at one-step segments; setting $n = 1$ gives expected depth $\Theta(\log H)$ and maximum depth $H - 1 = \Theta(H)$. □

The midpoint split halves the longest remaining segment at each level. Recursive uniform splitting has the structure of a random binary search tree, giving logarithmic average depth, whereas repeatedly splitting next to an endpoint gives linear worst-case depth. Accordingly, `TD-n` has $\Theta(H/n)$ sequential propagation levels, `DCRL-n` has $\Theta(\log(H/n))$, and `TRL-n` lies between these cases depending on its realized splits. The transitive branch of vanilla `TRL` has the same random-split structure but stops only at one-step leaves.

Error transmission. We characterize how each backup transmits the error of its children.

Assumption 1. *For each method $\bullet$, there exist $\epsilon \ge 0$ and $\zeta_\bullet \ge 1$ such that, for every non-grounded p and backup realization ω,*

$$|e_\bullet(p)| \le \epsilon + \zeta_\bullet \sum_{q \in B_\bullet(p;\omega)} |e_\bullet(q)|, \qquad e_\bullet(p) = 0 \quad \textit{for } p \in \mathcal{P}_\circ.$$

We use $\zeta = 1$ for nonselective symmetric backups and $\zeta > 1$ to model amplification from optimistic selection. In `TRL-n`, an upper expectile ($\kappa > 0.5$) can preferentially weight overestimated transitive targets; we model this effect with $\zeta_{\texttt{TRL-n}} > 1$. At $\kappa = 0.5$, the regression is symmetric, and we set $\zeta_{\texttt{TRL-n}} = 1$. Vanilla `TRL` uses the same transitive backup but has no additional n-step propagation objective and therefore only one-step grounding. `DCRL-n` instead uses the deterministic midpoint $k = \lfloor (i + j)/2 \rfloor$ and the factorized distance target $\hat{d}(s_i, s_k) + \hat{d}(s_k, s_j)$ under a symmetric loss. Because there is no candidate set to select from, we model this nonselective backup with $\zeta_{\texttt{DCRL-n}} = 1$. This objective cannot itself discover shortcuts, which is why `DCRL-n` pairs it with a separate multistep propagation objective. `TD-n`'s expectile ($\kappa > 0.5$) is applied across the sample distribution and therefore acts as an implicit maximum over the n-step continuations available from (s_i, g) in the data, in the manner of IQL (Kostrikov et al., 2022). Where several continuations of differing value are supported, this selection can amplify child error, which we model with $\zeta_{\texttt{TD-n}} > 1$; when the continuation is effectively unique, we use $\zeta_{\texttt{TD-n}} = 1$.

Upper bounds. First, we discuss upper error bounds for `TD-n`, `TRL-n`, and `DCRL-n`. Applying Assumption 1 repeatedly, from a sample to the grounded leaves on which it ultimately rests, bounds that sample's error. Doing so for each method gives the following.

At $\zeta = 1$, `TD-n` forms a chain of $\mathrm{D}_{\texttt{TD-n}} = \lceil H/n \rceil - 1$ backups, giving $|e| \leq \epsilon\, \mathrm{D}_{\texttt{TD-n}}(p) = O(\epsilon H/n)$. `DCRL-n` forms a binary tree of depth $d = \lceil \log_2(H/n) \rceil$. Unrolling Assumption 1 over this tree gives $|e| \leq \epsilon(2^d - 1) = O(\epsilon H/n)$. Thus, both methods admit the same asymptotic upper bound, despite having different sequential depths. For `TRL-n` with $\kappa > 0.5$, consider a realized decomposition of depth $D_\omega := \mathrm{D}_{\texttt{TRL-n}}(p; \omega)$. Let E_r bound the error at depth r, with $E_0 = 0$. Its selection gain $\zeta_{\texttt{TRL-n}} > 1$ gives

$$E_r \leq 2\zeta_{\texttt{TRL-n}} E_{r-1} + \epsilon,$$

and therefore

$$E_{D_\omega} \leq \epsilon \frac{(2\zeta_{\texttt{TRL-n}})^{D_\omega} - 1}{2\zeta_{\texttt{TRL-n}} - 1} = O\big(\epsilon (2\zeta_{\texttt{TRL-n}})^{D_\omega}\big)\,.$$

Thus, logarithmic-depth realizations yield a polynomial error bound in H/n, whereas highly unbalanced realizations can produce a much looser bound. Selection amplification therefore compounds with decomposition depth.

The upper bound cannot separate `TD-n` from `DCRL`, since a horizon-H value ultimately depends on the same H/n grounded segments. However, their dependency structures have different sequential depths.

Propagation depth. The static upper bounds above do not account for the order in which grounded information propagates through learned targets. Consider an idealized local backup process in which information traverses at most one dependency edge per update stage. By definition, information from a grounded leaf requires at least $\mathrm{D}_B(p)$ sequential stages to reach a sample p. Therefore,

$$\begin{aligned} \texttt{TD-n}: &\quad \Theta(H/n) \text{ stages}, \\ \texttt{DCRL}: &\quad \Theta(\log(H/n)) \text{ stages}. \end{aligned}$$

Thus, `DCRL` has an asymptotically shorter propagation path from grounded segments to long-horizon values. Bootstrap depth alone does not determine estimation error; however, a shorter path requires fewer sequential stages for grounded information to reach long-horizon values. We empirically test whether this structural advantage corresponds to lower finite-budget error in Appendix A.2.2.

Summary. The analysis highlights two complementary mechanisms:

1. **Fixed-split factorization.** The exact factorization in Eq. (3) avoids the selection bias introduced by upper-expectile transitive backups such as `TRL` and `TRL-n` ($\kappa > 0.5$).
2. **Recursive scheduling.** Balanced recursion reduces the dependency depth from linear to logarithmic, while the leaf-first schedule trains child values before using them in parent updates.

The first mechanism avoids maximization-induced selection error, whereas the second shortens and explicitly orders the propagation path from grounded to long-horizon values.

Scope. The propagation objective uses an upper-expectile loss to favor high-value n-step trajectory segments observed in the offline dataset. In an idealized finite, deterministic, tabular setting with complete transition coverage and exact optimization, taking $\kappa_{\texttt{prop}} \to 1$ recovers the hard maximum over these segments. The resulting n-step Bellman optimality operator is a γ^n-contraction and converges to V^*. Such a guarantee generally does not hold in practical offline RL settings because of incomplete data coverage and function approximation. Moreover, using a finite expectile ($\kappa_{\texttt{prop}} = 0.7$ in our experiments) may favor overestimated targets. The exact factorization in $\mathcal{L}^{\texttt{dc}}$ avoids intermediate-state selection when learning demonstrated routes, but does not eliminate optimistic selection from $\mathcal{L}^{\texttt{prop}}$. This separation is intentional: $\mathcal{L}^{\texttt{dc}}$ learns behavior values through selection-free factorization, whereas $\mathcal{L}^{\texttt{prop}}$ favors shorter routes composed across trajectories. Our analysis therefore characterizes bootstrap depth and error transmission rather than guaranteeing convergence of `DCRL` under function approximation.

A.2.2 Empirical Validation

In this section, we empirically test how bootstrap depth and recursive scheduling affect finite-budget error accumulation. We examine whether (i) shallower dependency structures correspond to slower long-range error growth and (ii) the depth advantage is realized when children are trained before their parents.

Didactic environment. Measuring error accumulation in benchmarks (*e.g.*, OGBench) is confounded because the optimal value $d^*(s, g)$ is unavailable in closed form and any proxy introduces a discretization error that itself grows with the horizon. We therefore use the `combination-lock` environment of Park et al. (2025b), introduced for exactly this purpose. It consists of H states in a line and two discrete actions; each state has one *answer* action, fixed by a random seed, that advances one step, while the other returns the agent to state 0. States are encoded as $\lceil \log_2 H \rceil$-dimensional binary vectors under a random permutation of indices, so the ordering cannot be read off the representation. Following Park et al. (2025b), we use the undiscounted step-cost setting ($\gamma = 1$) and learn distances directly rather than through $\log_\gamma Q$, which is defined only for $\gamma \in (0, 1)$. Across environments of increasing horizon, distances and errors are therefore not capped by the discounted value range. Under this convention, $Q^*(s, g) = -d^*(s, g)$ and $\hat{Q}(s, g) = -\hat{d}(s, g)$, giving $|\hat{Q} - Q^*| = |\hat{d} - d^*|$. Thus, Q-error and distance error have identical magnitudes and units.

We adapt the task to the goal-conditioned setting required by `DCRL` and `TRL` because their backups factor through an intermediate state and therefore require values for state–goal *pairs* rather than a single fixed goal. Along the forward path, the optimal distance is available exactly, $d^*(s, g) = g - s$, giving ground truth at every horizon.

All variants learn $\hat{d}(s, g)$ by squared-error regression in distance space and use the same network ($[512, 512, 512]$ MLP with LayerNorm and GELU), optimizer (Adam, $3{\times}10^{-4}$, batch 512, target-update rate 0.005), gradient budget (5M steps), and uniform sampling of (s, g). Let $h := g - s$ and set $n = 64$.

Distance-space expectile convention. All expectile parameters are reported using the value-space convention employed by the main algorithm. Let y_d denote a distance target and define the residual $\delta_d = y_d - \hat{d}(s, g)$. For non-grounded samples, we use

$$\ell_\kappa(\delta_d) = 2w_\kappa(\delta_d)\delta_d^2, \qquad w_\kappa(\delta_d) = \begin{cases} \kappa, & \delta_d < 0, \\ 1 - \kappa, & \delta_d \geq 0. \end{cases}$$

Exact grounded targets are fitted symmetrically with $w_\kappa = 0.5$. Because $Q = -d$ in this experiment, the value residual has the opposite sign of the distance residual. Consequently, weighting negative distance residuals by $\kappa > 0.5$ is equivalent to weighting positive value residuals by κ, thereby favoring higher values and shorter predicted distances. Under the standard distance-space expectile convention, our reported parameter κ therefore corresponds to the expectile level $1 - \kappa$; in particular, $\kappa = 0.7$ corresponds to a lower distance expectile of 0.3, not an upper distance expectile of 0.7.

Accordingly, `TRL-n` ($\kappa = 0.7$) uses optimistic transitive regression that favors shorter distances, whereas `TRL-n` ($\kappa = 0.5$) uses symmetric transitive regression. Only the transitive expectile differs between these two variants; their added n-step propagation objective is identical and uses the same value-space optimism convention. `DCRL`'s fixed-split transitive objective uses symmetric regression ($\kappa_{\texttt{dc}} = 0.5$).

We compare five variants. `TD-n`, both `TRL-n` variants, and `DCRL-n` use $n = 64$. Each `TRL-n` variant retains `TRL`'s transitive objective and adds the same n-step propagation objective used by `DCRL-n`; vanilla `TRL` uses only the transitive objective. `TRL-n` ($\kappa = 0.7$) uses optimistic transitive regression under the value-space convention defined above, whereas `TRL-n` ($\kappa = 0.5$) uses symmetric transitive regression. Here, κ applies only to the transitive objective; the added n-step propagation objective is unchanged.

Measuring error accumulation. Let $\mathcal{E}(h) := \mathbb{E}_{g-s=h}|\hat{d}(s, g) - d^*(s, g)|$ be the mean absolute distance error at horizon h. As H increases, the number of state–goal pairs grows

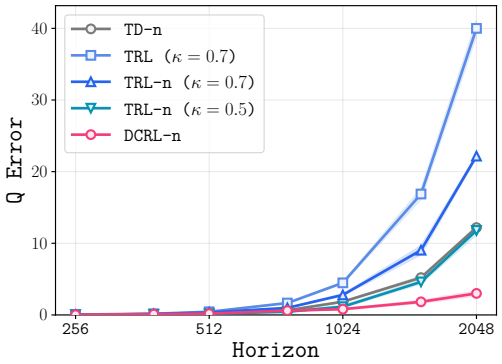


Figure 10. On `combination-lock`, accumulated error $\mathcal{A}(H)$ from Eq. (8) (3 seeds; shading: ± 1 s.d.). Adding n-step propagation reduces `TRL`'s error, while symmetric transitive regression ($\kappa = 0.5$) further lowers `TRL-n`'s error to the `TD-n` level. `DCRL-n` remains lowest and grows slowest, consistent with its logarithmic bootstrap depth and child-to-parent scheduling.

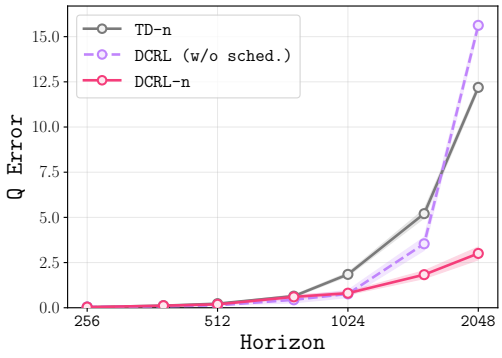


Figure 11. On `combination-lock`, we isolate child-to-parent scheduling while holding the factorized backup and n-step propagation fixed (3 seeds; shading: ± 1 s.d.). I.i.d. sampling performs comparably for $H \lesssim 1024$, but its error rises sharply at longer horizons and approaches `TD-n`'s. `DCRL-n` remains flat, showing that ordering becomes important as dependency chains deepen.

as H^2, making the value function increasingly difficult to fit. We capture the resulting total long-range error directly, defining *error accumulation* as the mean error on pairs spanning at least half of the environment horizon:

$$\mathcal{A}(H) \;:=\; \mathbb{E}_{(s,g):g-s\geq H/2}\left[\left|\hat{d}(s,g) - d^*(s,g)\right|\right]. \tag{8}$$

This metric reports long-range error in the original distance units without normalization by short-range fitting error. All variants use the same architecture and optimization budget, while the four n-step variants also receive identical exact supervision for $h \leq n$. Differences among these four therefore isolate their backup mechanisms and schedules; vanilla `TRL` additionally measures the effect of omitting n-step propagation.

Separating propagation, selection, and recursion. Figure 10 reports $\mathcal{A}(H)$ over seven horizons spanning 256–2,048. At long horizons, error follows `TRL`> `TRL-n` ($\kappa = 0.7$) > `TRL-n` ($\kappa = 0.5$) $\approx$ `TD-n` > `DCRL-n`. Adding n-step propagation substantially reduces `TRL`'s error. With this propagation fixed, lowering the transitive expectile from $\kappa = 0.7$ to 0.5 further reduces error, showing that optimistic subgoal selection amplifies estimation noise. `TRL-n` ($\kappa = 0.5$) performs similarly to `TD-n`, whereas `DCRL-n` remains relatively flat and achieves the lowest error. Since `TRL-n` ($\kappa = 0.5$) and `DCRL-n` both use symmetric regression and the same n-step propagation, the remaining gap reflects `DCRL`'s fixed-midpoint recursion and child-to-parent scheduling.

Child-to-parent scheduling realizes the depth advantage. Figure 11 isolates the second mechanism by holding the backup operator fixed and varying only the schedule. The benefit of scheduling grows with the horizon. Up to $H \approx 1024$, the two variants perform comparably (`DCRL (w/o sched.)`: $\mathcal{A} = 1.61$ versus `DCRL`: 1.78 at $H = 1024$), suggesting that both can adequately learn relatively shallow dependencies. Beyond this point, `DCRL (w/o sched.)` degrades sharply—reaching 3.72 at $H = 1536$ and 9.14 at $H = 2048$, nearly matching `TD-n` (9.10)—while `DCRL` remains flat. This result suggests that unordered sampling becomes increasingly prone to bootstrapping from unconverged child estimates as the horizon grows. The ablation therefore supports the practical importance of child-to-parent scheduling for long-horizon value learning.

Scope. This experiment isolates errors induced by backup mechanisms and scheduling and differs from our full implementation in three respects. First, values are regressed directly in

distance space with squared loss, whereas the deployed agents parameterize $Q \in (0, 1)$ and use multiplicative factorization under binary cross-entropy loss. Distance regression leaves error unbounded and therefore observable. Second, the didactic variants optimize only the backup objectives: `TRL` uses its transitive objective, `TRL-n` adds n-step propagation, and `DCRL-n` combines recursive factorization with the same propagation objective. They omit policy learning and implementation details of the full agents. Third, action and intermediate-state selection behave differently in `combination-lock`. Over actions, the correct successor and reset state differ in value by $\Theta(s)$, far exceeding approximation noise; hence, `TD-n`'s expectile has no ambiguous choice and $\zeta_{\texttt{TD-n}} = 1$. The experiment does not measure action-selection amplification under the multimodal benchmark data. Over intermediate states, every split is exactly tied, $d^*(s, k) + d^*(k, g) = g - s$, so upper-expectile transitive regression favors estimation noise. Accordingly, the gap between `TRL` and `TRL-n` ($\kappa = 0.7$) reflects the addition of n-step propagation, while the gap between the two `TRL-n` variants isolates transitive selection amplification. The gap between `TRL-n` ($\kappa = 0.5$) and `DCRL-n` reflects fixed-midpoint recursion and child-to-parent scheduling. These conclusions concern backup mechanisms and scheduling rather than end-to-end performance, which Section 5 evaluates separately.

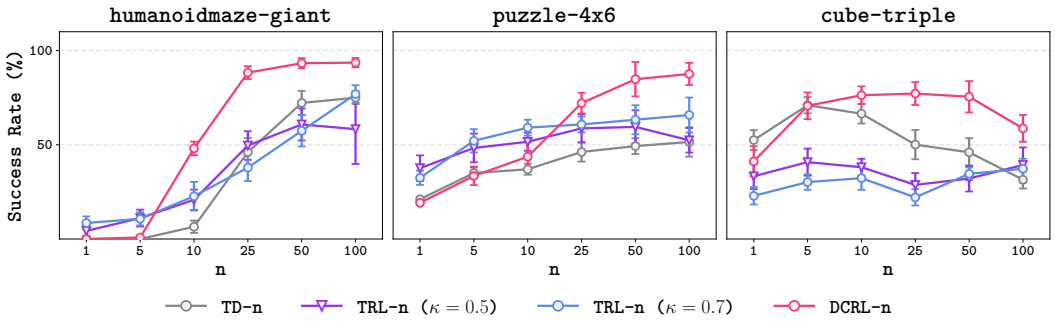


Figure 12. Exhaustive comparison across propagation horizons n on three long-horizon environments. We report success rate across $n \in \{1, 5, 10, 25, 50, 100\}$. `DCRL-n` performs best for $n \geq 25$ on all tasks, while the two `TRL-n` variants perform similarly overall.

End-to-end propagation-horizon comparison. Figure 12 complements the controlled error analysis with benchmark success rates across propagation horizons. `DCRL-n` performs best for $n \geq 25$ on all three tasks. The two `TRL-n` variants perform similarly overall, with neither consistently outperforming the other. Thus, although symmetric transitive regression reduces error in the controlled environment, this benefit does not consistently translate into higher benchmark performance.

B Full Algorithm

Algorithm 2 Divide-and-Conquer Reinforcement Learning (`DCRL`)

Input: Dataset $\mathcal{D}$, number of slots S, propagation horizon n
Initialize critic $Q(s, a, g)$, slots $\{\mathcal{T}_\ell \leftarrow \texttt{SampleSlot}(\mathcal{D})\}_{\ell=1}^{S}$
while not converged **do**
  ▷ Divide-and-Conquer
  Sample $\ell \sim \mathrm{Unif}(\{1, \dots, S\})$
  Sample $\tau_{\mathrm{DC}} \leftarrow \texttt{dequeue}(\mathcal{T}_\ell)$
  if $\mathcal{T}_\ell = \emptyset$ **then**
    $\mathcal{T}_\ell \leftarrow \texttt{SampleSlot}(\mathcal{D})$
  end if
  ▷ Propagation
  Sample $\tau_{\mathrm{prop}} = (s_i, a_i, s_k, a_k, g) \sim \mathcal{D}$, where $k = i + n$ is clipped at the trajectory end
  Train Q by minimizing $\mathcal{L}^{\texttt{dc}}(Q) + \mathcal{L}^{\texttt{prop}}(Q)$ (Eqs. 6, 7)
end while

Expectile-weighted BCE. The BCE objectives in Eqs. (6) and (7) use the same residual-based expectile-weighting rule: Eq. (6) uses the symmetric case $\kappa_{\texttt{dc}} = 0.5$, whereas Eq. (7) uses $\kappa_{\texttt{prop}} > 0.5$. For either objective, let z be the critic logit, $q = \sigma(z)$ its predicted value, and $y \in [0, 1]$ its stop-gradient target. Define

$$\ell_{\mathrm{BCE}}(q, y) = -y \log q - (1 - y) \log(1 - q), \qquad w_\kappa(q, y) = \begin{cases} \kappa, & y \geq q, \\ 1 - \kappa, & y < q. \end{cases}$$

We use

$$\mathcal{L}^{\texttt{dc}}(Q) = \mathbb{E}_{\tau_{\mathrm{DC}}} \left[2 w_{\kappa_{\texttt{dc}}}(q, y) \ell_{\mathrm{BCE}}(q, y) \right], \qquad \mathcal{L}^{\texttt{prop}}(Q) = \mathbb{E}_{\tau_{\mathrm{prop}}} \left[w_{\kappa_{\texttt{prop}}}(q, y) \ell_{\mathrm{BCE}}(q, y) \right].$$

Thus, $\kappa_{\texttt{dc}} = 0.5$ recovers standard BCE, while $\kappa_{\texttt{prop}} > 0.5$ emphasizes samples whose targets exceed their current predictions. For analytically grounded propagation targets, we instead use the symmetric weight $w = 0.5$.

Boundary cases. A recursive sample τ_{DC} is either a single-step leaf (s_i, a_i, s_{i+1}) or an internal-node sample $(s_i, a_i, s_k, a_k, s_j)$. For a leaf, we use target γ and minimize $\mathcal{L}_{\mathrm{BCE}}(Q(s_i, a_i, s_{i+1}), \gamma)$; for an internal node, we use the factorized target in Eq. (6), replacing either child value by γ when its segment is a single step. For propagation samples, both s_{i+n} and a_{i+n} in Eq. (7) are clipped to the final stored state-action index of the sampled trajectory, so bootstrapping never crosses a trajectory boundary; the multiplier remains γ^n.

Intra-batch ordering. Slot scheduling enforces a topological order over emitted samples, rather than a strict ordering of optimizer updates. Because a minibatch may contain multiple samples from the same slot, children and parents near a depth boundary can be included in the same batch and evaluated using the same pre-update critic. In this case, the parent target does not yet incorporate the child's within-batch update. Nevertheless, a child is never emitted after its parent, and co-batching can occur only when sampling crosses a depth boundary; all earlier descendants have already appeared in preceding batches. Thus, slot scheduling provides an approximate child-to-parent curriculum under minibatch training. The $O(\log H)$ bootstrap depth characterizes the structural dependency chain, not the number of optimizer steps required for convergence.

C Baselines

Offline GCRL methods generally comprise two components: goal-conditioned value estimation and policy extraction. We describe the main value-learning mechanism of each baseline below and note its policy-extraction mechanism when it is essential to the method. Our implementations of `TD`, `TD-n`, `TDP`, `COE`, and `TRL` follow Park et al. (2026).

Temporal-difference methods. `IQL` (Kostrikov et al., 2022) fits a goal-conditioned state-value function by asymmetric expectile regression over dataset actions and trains its critic with Bellman backups. The expectile approximates action maximization without evaluating out-of-distribution actions; at $\kappa = 0.5$, the objective reduces to behavioral SARSA value estimation. `IVL` (Kostrikov et al., 2022; Park et al., 2023) denotes goal-conditioned implicit V-learning, which fits only a goal-conditioned state-value function using expectile-weighted temporal-difference regression and extracts its policy through advantage-weighted regression. `SAC+BC` (Haarnoja et al., 2018) extracts a stochastic policy by maximizing the learned critic through reparameterized gradients while adding an entropy or behavior-cloning term that constrains the policy toward dataset actions. `TD` is a one-step, IQL-style goal-conditioned temporal-difference baseline, whereas `TD-n` replaces the one-step bootstrap with an n-step return and clips n when the sampled goal occurs sooner (Sutton & Barto, 2018). `DCRL` uses the same n as its propagation horizon on each task; we report these values in Appendix E.

Monte Carlo methods. `CRL` (Eysenbach et al., 2022) learns a goal-conditioned critic by contrastively distinguishing future states from independently sampled states and derives the policy from this learned reachability signal. `MC` learns a behavioral goal-conditioned value by directly regressing $Q(s_i, a_i, s_j)$ onto the empirical trajectory return γ^{j-i}, avoiding bootstrapping (Tian et al., 2021; Shah et al., 2021). This estimator is simple and stable but recovers behavior rather than optimal values under suboptimal data and can have high variance under stochastic dynamics.

Hierarchical methods. `POR` (Xu et al., 2022) is a policy-guided imitation method that uses a learned policy to select useful intermediate targets from the offline data and imitates the resulting improved decisions. `HIQL` (Park et al., 2023) treats latent future states as high-level actions: a high-level policy proposes a waypoint and a low-level goal-conditioned policy reaches it, with both policies extracted by advantage-weighted regression. `SHARSA` (Park et al., 2025b) reduces the effective decision horizon by training a high-level subgoal policy together with a low-level policy that executes each temporally extended subgoal.

Triangle-inequality methods. `QRL` (Wang et al., 2023) parameterizes the goal-reaching value as a quasimetric, architecturally imposing non-negativity, asymmetry, and the triangle inequality. `TMD` (Myers et al., 2025) learns an asymmetric temporal-distance representation from offline data while enforcing quasimetric structure, then uses this distance to extract a goal-conditioned policy. `TDP` (Kaelbling, 1993; Jurgenson et al., 2020) applies a Floyd–Warshall-style backup that chooses the best intermediate subgoal; in continuous domains, we approximate the hard maximum over the state space using subgoal candidates sampled from the dataset. `COE` (Piękos et al., 2023) enforces the compositional optimality equation with a separate generator that proposes the maximizing intermediate state; for offline training, the generator is regularized toward in-distribution states. `TRL` (Park et al., 2026) propagates reachability by composing two shorter goal-conditioned values through an intermediate state using a transitive backup, yielding an implicit divide-and-conquer structure.

Policy-only methods. We additionally include behavior cloning (`BC`), hierarchical behavior cloning (`HBC`) (Gupta et al., 2019), and flow-matching behavior cloning (`FBC`) (Chi et al., 2023). `BC` directly imitates dataset actions, `HBC` separately imitates high-level waypoints and low-level actions, and `FBC` uses an expressive flow-matching policy to model potentially multimodal behavior distributions.

D Alternative Divide-and-Conquer Strategies

Here we describe alternative divide-and-conquer strategies that we compare with DCRL in Section 5.3.

Curriculum learning. The simplest divide-and-conquer approach learns values on short trajectories first, gradually increasing their length via a manual curriculum. For length-T trajectories with curriculum step Δ, the i-th stage ($i = 1, \dots, T/\Delta$) learns values for sub-trajectories of length $(i-1)\cdot\Delta + 1$ to $i \cdot \Delta$, sampled uniformly from full trajectories.

Distance re-weighting. Park et al. (2026) implement divide-and-conquer implicitly by downweighting distant pairs in the loss. Each sample is weighted by $w(s_i, s_j) := \big(1 + \log_\gamma Q(s_i, a_i, s_j)\big)^{-\lambda}$, where $\log_\gamma Q$ is the learned distance. The weight decays with distance, so short pairs dominate; λ controls the decay rate, and $\lambda = 0$ recovers uniform weighting.

Random subgoal sampling. While DCRL splits each problem at the trajectory midpoint, an alternative samples a random in-trajectory subgoal. We implement DCRL (w/ random split), which is identical to Algorithm 2 but uses a random split $k \sim \mathrm{Unif}(i+1, j-1)$ instead of the midpoint during recursive tree construction (Algorithm 1).

Max-based backup. DCRL backs up each node through the trajectory midpoint with an exact factorization, $Q(s_i, a_i, s_j) \leftarrow Q(s_i, a_i, s_k)\, Q(s_k, a_k, s_j)$ with $k = \lfloor (i+j)/2 \rfloor$. A natural alternative replaces this fixed split with a *learned* one: at each node we sample $M = 10$ in-trajectory subgoal candidates $m \sim \mathrm{Unif}(i+1, j-1)$ and both split and back up at the candidate maximizing the factorized value under the current critic,

$$m^\star = \arg\max_m Q(s_i, a_i, s_m) \cdot Q(s_m, a_m, s_j),$$
$$Q(s_i, a_i, s_j) \leftarrow Q(s_i, a_i, s_{m^\star}) \cdot Q(s_{m^\star}, a_{m^\star}, s_j).$$

When a segment has fewer than M interior states, we simply enumerate all of them. We implement DCRL (w/ max-based backup), identical to Algorithm 2 except that the midpoint split k in the recursive tree construction (Algorithm 1) is replaced by $m^\star$. Because the split now depends on Q and determines the node's children, the tree is constructed top-down using the current critic, and the schedule is rebuilt every 5000 steps so the splits track the critic as it improves; the recursion, the base-case grounding $Q(s_i, a_i, s_{i+1}) \leftarrow \gamma$, and the value-propagation loss are all unchanged from DCRL. For the demonstrated route, the factorized target $\gamma^{m-i}\gamma^{j-m} = \gamma^{j-i}$ is independent of m. Maximizing learned values may nevertheless favor either shortcuts introduced by propagation or estimation error. This ablation tests whether adaptive split selection improves upon the fixed midpoint.

Without recursive scheduling. To test whether the depth-stratified, child-to-parent schedule is necessary, we ablate the recursion entirely while retaining the factorization backup. Instead of recursively decomposing each trajectory into a tree (Algorithm 1), we sample pairs (s_i, s_j) i.i.d. per batch with $j - i \sim \mathrm{Geom}(1-\gamma)$, split once at the midpoint $k = \lfloor (i+j)/2 \rfloor$, and apply the same backup $Q(s_i, a_i, s_j) \leftarrow Q(s_i, a_i, s_k)\, Q(s_k, a_k, s_j)$.

Without recursion, the sub-problems $Q(s_i, a_i, s_k)$ and $Q(s_k, a_k, s_j)$ are no longer guaranteed to have been trained before their parent. Moreover, i.i.d. sampling does not explicitly provide the one-step leaves that ground each recursive tree. Without reliable base-case supervision, the factorization target becomes self-referential and the critic collapses ($\bar{Q} \to 0$). We therefore add an explicit one-step loss on adjacent transitions,

$$\mathcal{L}_1(Q) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}}\big[\, \mathcal{L}_{\mathrm{BCE}}\big(Q(s, a, s'), \gamma\big)\,\big], \tag{9}$$

which restores the exact γ^1 base case that the recursion supplies implicitly. We implement DCRL (w/o recursion) with this grounding term and the unchanged value-propagation loss. This construction isolates the absence of recursive scheduling while retaining the same factorization and propagation objectives with equivalent one-step grounding.

Without propagation. To test whether divide-and-conquer alone can recover optimal values, we remove $\mathcal{L}^{\texttt{prop}}$ and increase the divide-and-conquer expectile from $\kappa_{\texttt{dc}} = 0.5$ to 0.7. The upper expectile favors higher-valued routes across samples, tasking the recursive objective with learning optimal values rather than behavior values. All other components remain unchanged.

Reverse scheduling. We implement `DCRL` (w/ reverse order) by retaining the same midpoint tree, factorized backup, one-step groundings, and propagation loss as `DCRL`, but reversing each slot's traversal order so that parents are emitted before their children. This ablation isolates traversal direction while preserving the recursive tree structure.

E Experimental Setup

Policy Extraction. After training the action-value function $Q(s, a, g)$, we extract a goal-conditioned policy $\pi(s, g) : \mathcal{S} \times \mathcal{S} \to \mathcal{A}$ that takes the highest-Q action. Policy extraction can be performed either during or after value training. We describe policy extraction procedures in Appendix F.

Evaluation. To evaluate offline goal-conditioned RL algorithms, we use a suite of environments from OGBench (Park et al., 2025a) and CALVIN (Mees et al., 2022).

For OGBench, each environment is evaluated on five test-time tasks corresponding to distinct start–goal pairs, with 15 rollouts per state-based task and 50 rollouts per pixel-based task.

For CALVIN, we use the in-domain Task D→Task D setup and the dataset provided by Shi et al. (2022). We evaluate the target sequence of four consecutive subtasks—opening the drawer, turning on the lightbulb, moving the slider to the left, and turning on the LED—using 50 rollouts for the state-based variant and 100 for the pixel-based variant.

Pixel-based CALVIN. The dataset of Shi et al. (2022) provides proprioceptive and scene states but no images, so we re-render it: every transition is replayed in the CALVIN simulator and an image is captured by the static third-person camera, yielding $128 \times 128 \times 3$ RGB observations. Rendering is done once offline and cached, so all methods consume identical pixel trajectories. Each observation comprises the image and a 15-dimensional proprioceptive vector and is normalized using dataset statistics; the goal is the rendered image of the target scene, in which all four subtasks are complete. The evaluation protocol is otherwise unchanged from the state-based variant: we use a single fixed far goal, and a rollout succeeds at level k if at least k of the four subtasks are completed in the prescribed order.

Visual encoder. For pixel-based CALVIN, we use a smaller version of the Impala CNN (Espeholt et al., 2018), matching the encoder used for the pixel-based OGBench environments, so that architecture is not a confounding factor across benchmarks. Observations and goals are encoded by separate encoder instances, both of which are trained end-to-end with the rest of the network at the same learning rate. We apply random-crop augmentation with probability 0.5. The padding is increased from 3 to 6 pixels: OGBench's pixel observations are 64×64, whereas ours are 128×128, so a 3-pixel pad would shift the image by half the relative amount and cover a correspondingly smaller fraction of the receptive field. Scaling the padding with the resolution keeps the effective augmentation strength, in units of the CNN's coverage, comparable to the OGBench setting.

Long-horizon OGBench tasks. For large-scale, long-horizon tasks (Table 1), we report the mean performance over four random seeds, averaged across checkpoints at 800k, 900k, and 1M training steps. For `DCRL`'s value propagation, we use $n = 100$ on `humanoidmaze-giant-1B` and `puzzle-{4x5,4x6}-1B`, and $n = 25$ on `cube-quadruple-100M` and `cube-octuple-1B`. Cube tasks decompose into single-block pick-and-place subtasks, so a smaller n suffices even when the full task is long. Full long-horizon results are reported in Table 8.

Table 4. Long-horizon environment specifications.

Environment	State Dim. ($\mathcal{S}$)	Goal Domain ($\mathcal{G}$)	Action Dim. ($\mathcal{A}$)	Episode length
humanoidmaze-giant	69	$\mathbb{R}^2$	21	4000
puzzle-4x5	99	$\{0,1\}^{20}$	5	1000
puzzle-4x6	115	$\{0,1\}^{24}$	5	1000
cube-triple	46	$\mathbb{R}^9$	5	1000
cube-quadruple	55	$\mathbb{R}^{12}$	5	1000
cube-octuple	91	$\mathbb{R}^{24}$	5	1500

Standard OGBench tasks. For state-based tasks (Figure 7 (*left*)), we report mean performance over four random seeds, averaged across the 800k, 900k, and 1M checkpoints. We use $n = 25$ for DCRL's value propagation on all standard OGBench tasks, both state-based and pixel-based. The state-based environments include:

- antmaze-large, antsoccer-arena, pointmaze-large, scene
- cube-{single,double}
- humanoidmaze-{medium,large}
- puzzle-{3x3,4x4}

For pixel-based tasks (Figure 7 (*right*)), we report mean performance over four random seeds, averaged across the 300k, 400k, and 500k checkpoints. The pixel-based environments include:

- visual-scene
- visual-antmaze-{medium,large,giant}
- visual-cube-{single,double,triple}
- visual-puzzle-{3x3,4x4,4x5}

Full state-based results and pixel-based results are reported in Table 15 and Table 16.

CALVIN benchmark. In Figure 8, we report mean performance over eight random seeds at the 500k checkpoint for state-based CALVIN, and over four random seeds at the 200k checkpoint for pixel-based CALVIN. On state-based CALVIN, we train HIQL without a latent representation, so its high-level policy predicts subgoals directly in the 21-dimensional observation space and is therefore comparable to the other value-learning methods, none of which learn a separate goal representation. On pixel-based CALVIN, we use HIQL's standard latent-subgoal formulation, in which the high-level policy predicts $\phi(g)$ in the space produced by the value function's goal encoder. Following the recommended configuration for pixel-based environments, ϕ is trained by both the value objective and the low-level actor's AWR loss. We use $n = 25$ for DCRL's value propagation. Full CALVIN results are reported in Table 6 and Table 7.

n-step comparison. For the n-step comparison (Figure 9), we use 10 random seeds and report mean performance at the 1M checkpoint for each $n \in \{1, 5, 10, 25, 50, 100\}$. We use humanoidmaze-giant-1B, puzzle-4x6-1B, and cube-triple-100M. Full task-level results for TD-n, TRL-n ($\kappa = 0.7$), and DCRL-n are reported in Appendix J.3.

F Policy Extraction

We consider the following two standard policy-extraction techniques.

Reparameterized gradients (Fujimoto & Gu, 2021) extract a Gaussian policy by maximizing the following combined DDPG (Lillicrap et al., 2016) and behavioral cloning (BC) objective:

$$J_{\text{DDPG+BC}}(\pi) = \mathbb{E}_{(s,a,g)\sim\mathcal{D},\, a^\pi\sim\pi(\cdot|s,g)}\left[Q(s,a^\pi,g) + \alpha \log \pi(a|s,g)\right], \tag{10}$$

where a^π is a reparameterized action from the policy, a is a dataset action, and α determines the strength of the BC regularization. This objective maximizes the predicted value while penalizing deviation from the dataset distribution to prevent out-of-distribution exploitation.

Rejection sampling (Chen et al., 2023; Hansen-Estruch et al., 2023) defines a nonparametric policy by drawing N action candidates and selecting the candidate with the highest value:

$$\pi(s,g) \overset{d}{=} \arg \max_{\{a_i\}_{i=1}^N \text{ s.t. } a_i\sim\pi_\beta(\cdot|s,g)} Q(s,a_i,g), \tag{11}$$

where N denotes the number of candidates, $\overset{d}{=}$ denotes equality in distribution, and π_β is a goal-conditioned BC policy. In practice, the BC policy is parameterized as an expressive diffusion or flow-matching model to capture multimodal behavior distributions effectively.

In our experiments, we use both reparameterized gradients and rejection sampling, depending on the environment. We report the exact extraction method and hyperparameters (*e.g.*, α and N) in Appendix K.

G Additional Related Work

Curriculum Learning. Our recursive schedule, which learns shorter samples before longer ones, resembles a curriculum over horizon length. Curriculum learning orders tasks by difficulty (Bengio et al., 2009; Graves et al., 2017; Florensa et al., 2018), requiring a task space and difficulty estimates. In contrast, `DCRL` decomposes purely by horizon: it recursively bisects each trajectory at its temporal midpoint, independent of the task or its difficulty. The short-to-long ordering thus emerges from the decomposition, and all lengths are trained concurrently across parallel slots rather than in sequential stages.

H Ablation Studies

In this section, we present ablation studies on the key components of DCRL.

H.1 Target Value Sources for the Divide-and-Conquer Loss

In our divide-and-conquer loss (Eq. (6)), DCRL computes the target value by multiplying two critic estimates corresponding to the first child and the second child. We investigate the effect of using either the *online* network (Q) or the *EMA* target network ($\bar{Q}$) for each child estimate. As shown in Figure 13, we evaluate all four combinations: $Q \cdot \bar{Q}$, $\bar{Q} \cdot \bar{Q}$, $Q \cdot Q$, and $\bar{Q} \cdot Q$. Across the humanoidmaze-giant and puzzle-4x6 environments, the combination of the online first child and the EMA second child ($Q \cdot \bar{Q}$) achieves the best success rate. Relying entirely on the online network ($Q \cdot Q$) causes value estimates to collapse, while the other configurations ($\bar{Q} \cdot \bar{Q}$ and $\bar{Q} \cdot Q$) degrade performance on at least one task.

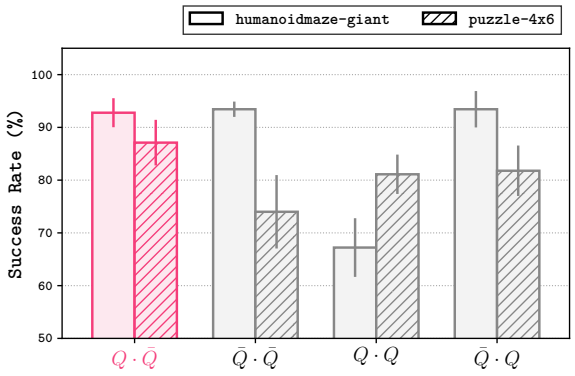


Figure 13. Using the online first child and the EMA second child ($Q \cdot \bar{Q}$) yields the most robust performance on humanoidmaze-giant and puzzle-4x6.

The two factors play complementary freshness and stability roles. The first factor, $Q(s_i, a_i, s_k)$, represents the shorter prefix from the parent state–action pair to the midpoint. Because leaf-first scheduling processes shorter segments before their parents, the online critic contains the freshest prefix updates; using $\bar{Q}$ would delay their propagation through the EMA lag. We stop gradients through this factor, so the online critic is used only as a fresh target snapshot. The second factor, $\bar{Q}(s_k, a_k, s_j)$, represents the remaining continuation and plays a role analogous to the bootstrapped future value in an n-step TD target. Evaluating it with the EMA critic stabilizes the recursively constructed target. Thus, $Q \cdot \bar{Q}$ balances rapid child-to-parent propagation with target stability: $\bar{Q} \cdot \bar{Q}$ propagates updates more slowly, whereas $Q \cdot Q$ multiplies two rapidly changing estimates and can amplify correlated errors or self-referential drift. This interpretation motivates the asymmetric design, while Figure 13 provides its empirical validation.

H.2 Expectile Parameters

We evaluate the sensitivity of DCRL to the expectile parameters $\kappa_{\texttt{dc}}$ (divide-and-conquer loss) and $\kappa_{\texttt{prop}}$ (propagation loss). As shown in Figure 14 and Figure 15, DCRL achieves the best performance with $\kappa_{\texttt{dc}} = 0.5$ and $\kappa_{\texttt{prop}} = 0.7$ on both humanoidmaze-giant and cube-triple. Deviating from these values generally degrades performance.

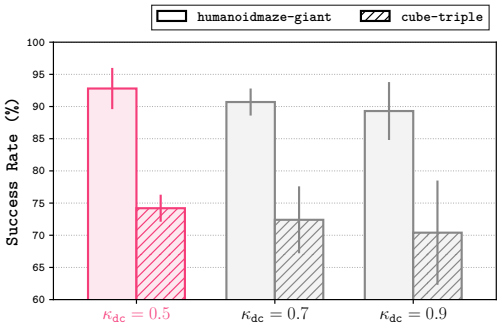


Figure 14. Ablation of the expectile parameter $\kappa_{\texttt{dc}}$ for the divide-and-conquer loss $\mathcal{L}^{\texttt{dc}}$. Performance peaks at $\kappa_{\texttt{dc}} = 0.5$ (our default).

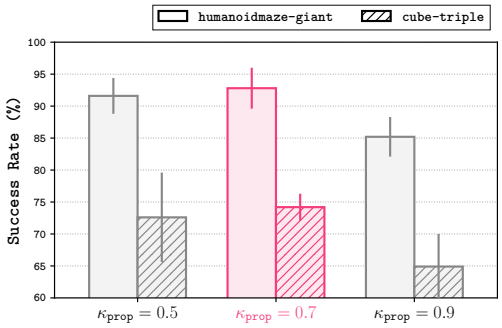


Figure 15. Ablation of the expectile parameter $\kappa_{\texttt{prop}}$ for the propagation loss $\mathcal{L}^{\texttt{prop}}$. Performance peaks at $\kappa_{\texttt{prop}} = 0.7$ (our default).

H.3 Number of Slots

We study how the number of slots affects DCRL's performance and training time. Larger numbers of slots incur greater CPU overhead because they must be maintained concurrently, increasing training time. Figures 16 and 17 provide a Pareto analysis of DCRL and TRL ($\lambda = 0$) in humanoidmaze-giant and cube-triple. In humanoidmaze-giant, DCRL reaches 90% success in under 3 hours, whereas TRL remains below 80% even after 8 hours. In cube-triple, DCRL reaches 70% success in under 2 hours, whereas TRL stalls below 50% and does not improve with a larger training budget. We measure training time on the same GPU (NVIDIA L40S).

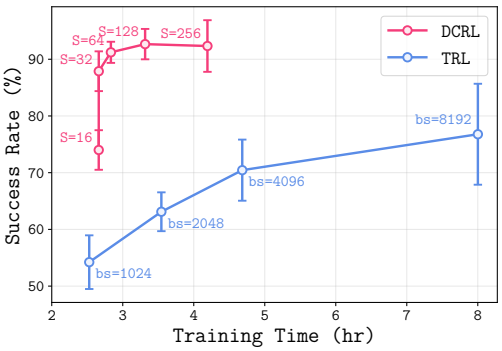


Figure 16. Across training budgets, DCRL outperforms TRL. In humanoidmaze-giant, we plot success rate (%) against training time (hours), while varying the number of slots (S=16,32,64,128,256) for DCRL at a fixed batch size (1024), and varying the batch size (bs=1024,2048,4096,8192) for TRL.

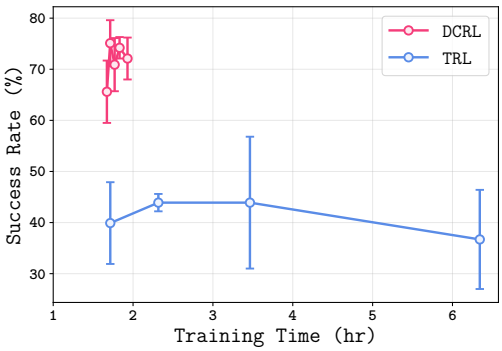


Figure 17. Same Pareto analysis as Fig. 16 on cube-triple. DCRL maintains a better performance–efficiency trade-off and reaches 70% success within 2 hours. DCRL varies slots (S) at a fixed batch size (1024), whereas TRL varies batch size (bs); the points follow the same left-to-right configuration order.

I Divide-and-Conquer Strategies on Cube Manipulation

We provide additional comparisons of divide-and-conquer strategies on `cube-triple`. As shown in Table 5, the results are mostly consistent with our findings on `humanoidmaze-giant` (Table 2). `DCRL` achieves performance comparable to random splitting and about 1.7× the success rate of the best `TRL` variant, with approximately 7% lower throughput. The same pattern holds for the ablations: the random split performs no better while cutting throughput by 43%, curriculum learning remains highly sensitive to Δ, and distance re-weighting fails to improve upon `TRL`. We measure training throughput for Tables 2, 3, and 5 on the same GPU (NVIDIA L40S).

Table 5. Comparison of divide-and-conquer strategies on `cube-triple` using the same batch size (1024). We report success rate (%) and training throughput (gradient steps per second).

Method	Success ↑	Throughput ↑
Curriculum ($\Delta = 25$)	7 ± 4	155 ± 2
Curriculum ($\Delta = 50$)	35 ± 5	154 ± 2
Curriculum ($\Delta = 100$)	15 ± 6	155 ± 2
TRL ($\lambda = 0$)	37 ± 8	161 ± 2
TRL ($\lambda = 0.01$)	42 ± 3	163 ± 2
TRL ($\lambda = 0.1$)	34 ± 4	160 ± 1
DCRL (w/ random split)	72 ± 4	86 ± 1
DCRL	71 ± 3	152 ± 1

J Full Experimental Results

J.1 CALVIN Benchmark

Table 6. Full results on state-based CALVIN.

		Temporal Difference		Monte Carlo		Hierarchical			Triangle Inequality			
Metric	BC	IQL	TD-n	MC	CRL	HBC	POR	HIQL	QRL	TMD	TRL	DCRL
Return (%)	13.5 ± 13.2	12.5 ± 18.9	24.4 ± 14.6	23.3 ± 10.9	8.2 ± 9.5	11.0 ± 12.3	3.8 ± 7.2	20.2 ± 24.8	12.5 ± 18.9	4.8 ± 9.3	22.8 ± 9.7	35.4 ± 14.4
Completion (%)	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	3.0 ± 8.5	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	8.5 ± 7.1

Table 7. Full results on pixel-based CALVIN. We report the success rate (%) of completing at least k consecutive subtasks, together with the average number of subtasks completed per rollout.

	# of Subtasks				
Method	1	2	3	4	Avg. Subtasks
TD-n	30 ± 22	4 ± 3	0 ± 1	0 ± 0	0.34 ± 0.24
TRL	30 ± 30	7 ± 9	0 ± 0	0 ± 0	0.37 ± 0.39
HIQL	75 ± 50	25 ± 50	0 ± 0	0 ± 0	1.00 ± 0.82
DCRL	72 ± 26	32 ± 22	8 ± 7	1 ± 2	1.14 ± 0.54

J.2 Long-Horizon OGBench Tasks

Table 8. Full results on large-scale, long-horizon OGBench tasks.

			Temporal Difference				Monte Carlo		Hierarchical		Triangle Inequality				
Environment	Task	FBC	IQL	SAC+BC	TD	TD-n	MC	CRL	HIQL	SHARSA	QRL	TDP	COE	TRL	DCRL
humanoidmaze-giant-1B	task1	0 ± 0	0 ± 0	5 ± 3	2 ± 4	68 ± 14	64 ± 9	52 ± 38	16 ± 8	23 ± 8	1 ± 1	0 ± 0	1 ± 1	12 ± 17	91 ± 6
	task2	0 ± 0	3 ± 7	12 ± 3	8 ± 4	90 ± 13	87 ± 5	63 ± 46	38 ± 12	46 ± 14	2 ± 2	3 ± 3	2 ± 2	76 ± 6	97 ± 3
	task3	0 ± 0	5 ± 6	8 ± 6	1 ± 1	65 ± 12	83 ± 8	68 ± 46	26 ± 14	16 ± 3	2 ± 3	3 ± 2	4 ± 2	11 ± 5	90 ± 5
	task4	0 ± 0	2 ± 3	2 ± 3	2 ± 2	78 ± 14	78 ± 8	57 ± 41	12 ± 6	35 ± 7	0 ± 0	0 ± 0	1 ± 2	76 ± 9	96 ± 4
	task5	0 ± 0	3 ± 4	0 ± 0	3 ± 1	89 ± 10	84 ± 11	68 ± 46	32 ± 7	89 ± 3	8 ± 8	4 ± 2	5 ± 2	96 ± 4	94 ± 6
	overall	0 ± 0	3 ± 4	5 ± 0	3 ± 2	78 ± 4	79 ± 4	62 ± 42	25 ± 6	42 ± 3	3 ± 2	2 ± 0	2 ± 1	54 ± 5	93 ± 3
puzzle-4x5-1B	task1	0 ± 0	100 ± 0	95 ± 3	84 ± 7	100 ± 0	100 ± 0	7 ± 5	58 ± 25	100 ± 0	0 ± 0	0 ± 0	0 ± 0	100 ± 0	100 ± 0
	task2	0 ± 0	0 ± 0	0 ± 0	4 ± 3	98 ± 4	67 ± 12	0 ± 0	0 ± 0	99 ± 1	0 ± 0	0 ± 0	0 ± 0	99 ± 1	100 ± 0
	task3	0 ± 0	0 ± 0	0 ± 0	2 ± 2	84 ± 28	8 ± 10	0 ± 0	0 ± 0	93 ± 5	0 ± 0	0 ± 0	0 ± 0	100 ± 0	100 ± 0
	task4	0 ± 0	0 ± 0	0 ± 0	2 ± 2	98 ± 3	50 ± 9	0 ± 0	0 ± 0	87 ± 5	0 ± 0	0 ± 0	0 ± 0	99 ± 1	99 ± 1
	task5	0 ± 0	0 ± 0	0 ± 0	2 ± 2	81 ± 17	8 ± 11	0 ± 0	0 ± 0	72 ± 11	0 ± 0	0 ± 0	0 ± 0	88 ± 8	99 ± 1
	overall	0 ± 0	20 ± 0	19 ± 1	19 ± 1	92 ± 8	47 ± 7	1 ± 1	12 ± 5	90 ± 4	0 ± 0	0 ± 0	0 ± 0	97 ± 1	100 ± 0
puzzle-4x6-1B	task1	0 ± 0	87 ± 9	48 ± 39	61 ± 16	100 ± 0	98 ± 4	0 ± 0	17 ± 12	99 ± 1	0 ± 0	0 ± 0	0 ± 0	100 ± 0	100 ± 0
	task2	0 ± 0	0 ± 0	5 ± 10	2 ± 2	39 ± 21	46 ± 30	0 ± 0	4 ± 2	61 ± 13	0 ± 0	0 ± 0	0 ± 0	66 ± 13	99 ± 2
	task3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	81 ± 21	34 ± 10	0 ± 0	0 ± 0	71 ± 15	0 ± 0	0 ± 0	0 ± 0	67 ± 21	100 ± 0
	task4	0 ± 0	0 ± 0	0 ± 0	1 ± 1	46 ± 28	5 ± 6	0 ± 0	0 ± 0	45 ± 7	0 ± 0	0 ± 0	0 ± 0	23 ± 7	100 ± 0
	task5	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	37 ± 24
	overall	0 ± 0	17 ± 2	11 ± 8	13 ± 3	53 ± 11	37 ± 4	0 ± 0	4 ± 3	55 ± 3	0 ± 0	0 ± 0	0 ± 0	51 ± 5	87 ± 5
cube-quadruple-100M	task1	3 ± 2	94 ± 3	59 ± 13	66 ± 6	51 ± 10	5 ± 4	56 ± 6	88 ± 7	100 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1	59 ± 7
	task2	0 ± 1	39 ± 4	77 ± 8	25 ± 9	44 ± 9	0 ± 0	46 ± 8	79 ± 1	100 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	69 ± 8
	task3	0 ± 0	63 ± 5	34 ± 9	8 ± 3	9 ± 4	0 ± 0	2 ± 1	69 ± 13	77 ± 3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	25 ± 12
	task4	0 ± 0	5 ± 2	5 ± 2	1 ± 1	4 ± 3	0 ± 0	1 ± 0	32 ± 9	34 ± 4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	7 ± 4
	task5	0 ± 0	1 ± 1	21 ± 8	1 ± 1	12 ± 7	0 ± 0	2 ± 2	38 ± 5	61 ± 6	0 ± 0	0 ± 0	0 ± 0	0 ± 0	27 ± 3
	overall	1 ± 0	41 ± 3	39 ± 4	20 ± 2	21 ± 6	1 ± 1	22 ± 2	62 ± 6	74 ± 2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	37 ± 4
cube-octuple-1B	task1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 2	74 ± 5	0 ± 0	0 ± 0	0 ± 0	0 ± 0	23 ± 7
	task2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	task3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	task4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	task5	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	overall	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	15 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	5 ± 1

J.3 Propagation Horizon Comparison

Table 9. Results with $n = 1$.

Environment	Task	TD-1	TRL-1	DCRL-1
humanoidmaze-giant-1B	task1	0 ± 0	0 ± 0	0 ± 0
	task2	0 ± 0	0 ± 0	0 ± 0
	task3	0 ± 0	0 ± 0	0 ± 0
	task4	0 ± 0	0 ± 0	0 ± 0
	task5	0 ± 0	42 ± 18	0 ± 0
	overall	0 ± 0	8 ± 4	0 ± 0
puzzle-4x5-1B	task1	100 ± 0	100 ± 1	100 ± 0
	task2	1 ± 2	85 ± 16	0 ± 0
	task3	0 ± 0	67 ± 29	0 ± 0
	task4	0 ± 0	96 ± 6	0 ± 0
	task5	0 ± 0	63 ± 24	0 ± 0
	overall	20 ± 0	82 ± 15	20 ± 0
puzzle-4x6-1B	task1	99 ± 1	99 ± 2	99 ± 2
	task2	6 ± 5	28 ± 10	3 ± 4
	task3	0 ± 0	23 ± 13	0 ± 0
	task4	0 ± 0	13 ± 12	0 ± 0
	task5	0 ± 0	0 ± 0	0 ± 0
	overall	21 ± 1	33 ± 4	20 ± 1
cube-triple-100M	single_pnp	95 ± 7	68 ± 14	89 ± 8
	triple_pnp	61 ± 14	15 ± 8	42 ± 13
	from_stack	54 ± 9	27 ± 11	30 ± 11
	cycle	26 ± 8	2 ± 2	5 ± 2
	stack	26 ± 9	2 ± 3	36 ± 6
	overall	52 ± 5	23 ± 5	40 ± 5

Table 10. Results with $n = 5$.

Environment	Task	TD-5	TRL-5	DCRL-5
humanoidmaze-giant-1B	task1	0 ± 0	0 ± 0	0 ± 0
	task2	0 ± 0	0 ± 0	2 ± 2
	task3	0 ± 0	0 ± 0	1 ± 2
	task4	0 ± 0	0 ± 0	1 ± 2
	task5	0 ± 0	54 ± 18	0 ± 0
	overall	0 ± 0	11 ± 4	1 ± 1
puzzle-4x5-1B	task1	100 ± 0	100 ± 0	100 ± 0
	task2	80 ± 12	95 ± 6	82 ± 12
	task3	34 ± 17	87 ± 14	27 ± 17
	task4	67 ± 14	97 ± 5	42 ± 17
	task5	15 ± 10	75 ± 21	10 ± 9
	overall	59 ± 9	91 ± 9	52 ± 8
puzzle-4x6-1B	task1	100 ± 0	96 ± 6	100 ± 0
	task2	70 ± 13	31 ± 17	61 ± 14
	task3	5 ± 4	76 ± 17	4 ± 5
	task4	0 ± 0	58 ± 20	0 ± 0
	task5	0 ± 0	0 ± 0	0 ± 0
	overall	35 ± 3	52 ± 6	33 ± 2
cube-triple-100M	single_pnp	85 ± 8	75 ± 8	82 ± 8
	triple_pnp	77 ± 10	31 ± 10	79 ± 9
	from_stack	69 ± 9	24 ± 10	73 ± 10
	cycle	47 ± 11	4 ± 4	40 ± 10
	stack	77 ± 8	17 ± 8	65 ± 10
	overall	71 ± 4	30 ± 4	68 ± 6

Table 11. Results with $n = 10$.

Environment	Task	TD-10	TRL-10	DCRL-10
humanoidmaze-giant-1B	task1	12 ± 9	3 ± 4	59 ± 10
	task2	4 ± 7	20 ± 21	84 ± 7
	task3	2 ± 3	2 ± 4	45 ± 8
	task4	13 ± 10	14 ± 18	47 ± 10
	task5	2 ± 4	74 ± 17	5 ± 4
	overall	6 ± 3	23 ± 8	48 ± 4
puzzle-4x5-1B	task1	100 ± 0	100 ± 0	100 ± 0
	task2	99 ± 2	99 ± 4	100 ± 0
	task3	95 ± 7	95 ± 7	100 ± 0
	task4	96 ± 4	99 ± 2	95 ± 8
	task5	48 ± 12	91 ± 10	94 ± 6
	overall	88 ± 3	97 ± 4	98 ± 2
puzzle-4x6-1B	task1	100 ± 0	100 ± 1	100 ± 0
	task2	72 ± 9	32 ± 20	58 ± 24
	task3	13 ± 12	81 ± 20	55 ± 22
	task4	0 ± 1	83 ± 11	14 ± 11
	task5	0 ± 0	0 ± 0	0 ± 0
	overall	37 ± 3	59 ± 4	45 ± 8
cube-triple-100M	single_pnp	77 ± 9	70 ± 9	86 ± 7
	triple_pnp	69 ± 8	34 ± 14	88 ± 8
	from_stack	67 ± 9	32 ± 9	86 ± 8
	cycle	42 ± 14	4 ± 4	42 ± 11
	stack	78 ± 7	21 ± 6	83 ± 9
	overall	67 ± 5	32 ± 6	77 ± 5

Table 12. Results with $n = 25$.

Environment	Task	TD-25	TRL-25	DCRL-25
humanoidmaze-giant-1B	task1	41 ± 14	15 ± 13	89 ± 5
	task2	54 ± 16	52 ± 12	97 ± 3
	task3	42 ± 11	12 ± 12	80 ± 11
	task4	43 ± 16	27 ± 18	88 ± 8
	task5	50 ± 21	83 ± 6	88 ± 14
	overall	46 ± 7	38 ± 7	88 ± 3
puzzle-4x5-1B	task1	100 ± 0	100 ± 0	100 ± 0
	task2	99 ± 2	98 ± 3	100 ± 0
	task3	99 ± 2	91 ± 16	100 ± 1
	task4	98 ± 2	97 ± 2	99 ± 2
	task5	92 ± 6	93 ± 12	100 ± 1
	overall	98 ± 2	96 ± 6	100 ± 1
puzzle-4x6-1B	task1	100 ± 0	99 ± 2	100 ± 0
	task2	63 ± 16	33 ± 16	62 ± 28
	task3	56 ± 17	89 ± 4	99 ± 2
	task4	12 ± 11	83 ± 14	88 ± 11
	task5	0 ± 0	0 ± 0	6 ± 7
	overall	46 ± 5	61 ± 4	71 ± 7
cube-triple-100M	single_pnp	65 ± 15	57 ± 14	82 ± 15
	triple_pnp	44 ± 15	22 ± 7	73 ± 8
	from_stack	49 ± 11	19 ± 11	76 ± 6
	cycle	20 ± 10	2 ± 2	41 ± 10
	stack	72 ± 5	12 ± 4	83 ± 8
	overall	50 ± 8	22 ± 4	71 ± 5

Table 13. Results with $n = 50$.

Environment	Task	TD-50	TRL-50	DCRL-50
humanoidmaze-giant-1B	task1	60 ± 9	38 ± 16	89 ± 8
	task2	80 ± 16	75 ± 11	97 ± 3
	task3	60 ± 19	35 ± 9	89 ± 6
	task4	64 ± 22	46 ± 19	95 ± 3
	task5	96 ± 5	93 ± 6	96 ± 3
	overall	72 ± 6	57 ± 8	93 ± 3
puzzle-4x5-1B	task1	100 ± 0	99 ± 1	100 ± 0
	task2	100 ± 0	97 ± 5	100 ± 0
	task3	93 ± 20	98 ± 4	100 ± 1
	task4	97 ± 4	98 ± 4	100 ± 1
	task5	83 ± 28	91 ± 8	100 ± 1
	overall	95 ± 10	97 ± 2	100 ± 0
puzzle-4x6-1B	task1	100 ± 0	98 ± 3	100 ± 0
	task2	27 ± 8	35 ± 28	90 ± 11
	task3	68 ± 16	91 ± 9	100 ± 0
	task4	52 ± 14	92 ± 6	96 ± 2
	task5	0 ± 0	0 ± 1	30 ± 11
	overall	49 ± 4	63 ± 8	83 ± 4
cube-triple-100M	single_pnp	65 ± 12	76 ± 8	87 ± 6
	triple_pnp	45 ± 19	32 ± 12	81 ± 7
	from_stack	42 ± 10	32 ± 10	73 ± 9
	cycle	16 ± 7	6 ± 6	39 ± 10
	stack	61 ± 14	27 ± 8	70 ± 11
	overall	46 ± 7	35 ± 3	70 ± 5

Table 14. Results with $n = 100$.

Environment	Task	TD-100	TRL-100	DCRL-100
humanoidmaze-giant-1B	task1	59 ± 12	68 ± 10	87 ± 8
	task2	89 ± 6	95 ± 4	97 ± 3
	task3	57 ± 11	52 ± 12	92 ± 5
	task4	79 ± 11	74 ± 15	94 ± 6
	task5	90 ± 7	95 ± 7	97 ± 4
	overall	75 ± 3	77 ± 5	94 ± 2
puzzle-4x5-1B	task1	100 ± 0	98 ± 3	100 ± 0
	task2	96 ± 6	99 ± 2	100 ± 1
	task3	78 ± 28	96 ± 6	100 ± 1
	task4	97 ± 2	98 ± 2	99 ± 3
	task5	74 ± 21	91 ± 9	97 ± 3
	overall	89 ± 11	96 ± 3	99 ± 1
puzzle-4x6-1B	task1	100 ± 0	99 ± 2	100 ± 0
	task2	42 ± 19	54 ± 29	91 ± 15
	task3	74 ± 20	92 ± 6	100 ± 0
	task4	42 ± 18	81 ± 16	99 ± 2
	task5	0 ± 0	3 ± 4	31 ± 16
	overall	51 ± 8	66 ± 9	84 ± 6
cube-triple-100M	single_pnp	60 ± 10	79 ± 9	83 ± 9
	triple_pnp	29 ± 10	44 ± 7	57 ± 11
	from_stack	27 ± 13	38 ± 9	51 ± 6
	cycle	3 ± 4	7 ± 2	16 ± 7
	stack	37 ± 12	18 ± 12	53 ± 13
	overall	31 ± 5	37 ± 5	52 ± 5

J.4 Standard OGBench Tasks

Table 15. Full results on state-based OGBench tasks.

Environment	Task	BC	FBC	IVL	IQL	TD-n	CRL	MC	QRL	TMD	TDP	COE	TRL	DCRL
pointmaze-large	task1	53 ± 24	98 ± 3	94 ± 6	92 ± 9	74 ± 13	23 ± 18	13 ± 12	31 ± 28	18 ± 25	40 ± 19	67 ± 26	52 ± 19	82 ± 9
	task2	3 ± 4	17 ± 6	0 ± 0	0 ± 0	0 ± 0	42 ± 28	0 ± 0	5 ± 6	4 ± 9	0 ± 0	0 ± 0	1 ± 2	0 ± 0
	task3	17 ± 9	76 ± 11	100 ± 0	76 ± 13	40 ± 13	67 ± 14	3 ± 4	0 ± 0	8 ± 9	4 ± 4	12 ± 8	7 ± 5	53 ± 16
	task4	23 ± 10	79 ± 8	3 ± 5	0 ± 0	9 ± 9	13 ± 18	3 ± 6	1 ± 2	12 ± 10	56 ± 12	44 ± 27	41 ± 13	42 ± 27
	task5	30 ± 22	87 ± 5	44 ± 39	0 ± 0	31 ± 8	21 ± 8	0 ± 0	0 ± 0	12 ± 15	49 ± 16	48 ± 16	64 ± 14	74 ± 9
	overall	25 ± 3	71 ± 4	48 ± 10	34 ± 3	31 ± 7	33 ± 7	4 ± 4	7 ± 7	11 ± 8	30 ± 5	34 ± 7	33 ± 5	50 ± 6
antmaze-large	task1	3 ± 3	13 ± 11	11 ± 13	23 ± 13	63 ± 16	87 ± 9	30 ± 8	51 ± 15	7 ± 2	6 ± 5	6 ± 6	57 ± 13	73 ± 11
	task2	22 ± 12	24 ± 8	19 ± 5	44 ± 20	76 ± 5	64 ± 25	47 ± 8	69 ± 12	20 ± 6	23 ± 6	26 ± 11	66 ± 4	83 ± 4
	task3	53 ± 10	48 ± 6	55 ± 6	82 ± 6	87 ± 6	91 ± 3	86 ± 6	96 ± 3	64 ± 16	69 ± 9	67 ± 4	80 ± 4	90 ± 5
	task4	17 ± 9	7 ± 3	4 ± 3	15 ± 7	21 ± 8	92 ± 4	17 ± 10	57 ± 16	14 ± 2	17 ± 10	14 ± 1	8 ± 8	24 ± 14
	task5	17 ± 5	16 ± 4	18 ± 12	20 ± 7	28 ± 6	90 ± 6	13 ± 5	61 ± 12	23 ± 13	17 ± 4	18 ± 5	18 ± 11	31 ± 14
	overall	22 ± 4	22 ± 4	21 ± 6	37 ± 8	55 ± 6	85 ± 7	39 ± 5	67 ± 8	26 ± 4	27 ± 3	26 ± 3	46 ± 5	60 ± 6
humanoidmaze-medium	task1	10 ± 7	7 ± 9	29 ± 4	31 ± 9	95 ± 2	91 ± 8	79 ± 13	8 ± 12	7 ± 2	4 ± 3	6 ± 4	76 ± 6	98 ± 2
	task2	6 ± 4	8 ± 5	37 ± 12	82 ± 9	94 ± 1	96 ± 1	88 ± 5	22 ± 29	11 ± 6	7 ± 3	9 ± 2	96 ± 2	96 ± 1
	task3	8 ± 5	15 ± 5	16 ± 7	6 ± 6	8 ± 7	72 ± 17	1 ± 1	30 ± 21	15 ± 5	12 ± 3	13 ± 3	8 ± 8	21 ± 14
	task4	2 ± 1	3 ± 3	0 ± 0	0 ± 0	8 ± 12	8 ± 7	2 ± 3	9 ± 9	1 ± 2	0 ± 0	1 ± 1	11 ± 3	2 ± 2
	task5	11 ± 5	12 ± 2	39 ± 16	63 ± 12	97 ± 3	93 ± 4	76 ± 7	20 ± 19	14 ± 10	14 ± 4	14 ± 7	92 ± 3	98 ± 2
	overall	7 ± 2	9 ± 3	24 ± 4	36 ± 3	60 ± 3	72 ± 5	49 ± 3	18 ± 17	9 ± 2	7 ± 0	9 ± 2	57 ± 1	63 ± 3
humanoidmaze-large	task1	1 ± 1	0 ± 0	9 ± 3	10 ± 1	34 ± 14	49 ± 17	6 ± 5	3 ± 3	1 ± 1	0 ± 0	0 ± 0	9 ± 6	39 ± 6
	task2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	2 ± 3
	task3	3 ± 1	2 ± 1	3 ± 2	11 ± 6	64 ± 14	63 ± 25	22 ± 11	9 ± 6	3 ± 3	3 ± 4	4 ± 3	29 ± 9	66 ± 13
	task4	3 ± 3	1 ± 1	1 ± 1	2 ± 1	0 ± 0	9 ± 8	1 ± 1	0 ± 0	2 ± 2	2 ± 2	4 ± 2	4 ± 4	5 ± 6
	task5	2 ± 2	2 ± 1	2 ± 1	0 ± 0	0 ± 0	20 ± 17	0 ± 0	1 ± 1	0 ± 0	0 ± 0	2 ± 2	0 ± 0	1 ± 2
	overall	2 ± 1	1 ± 1	3 ± 1	5 ± 1	20 ± 2	28 ± 5	6 ± 2	3 ± 2	1 ± 1	1 ± 1	2 ± 0	8 ± 1	22 ± 2
antsoccer-arena	task1	7 ± 4	30 ± 4	76 ± 5	86 ± 6	78 ± 8	52 ± 11	71 ± 4	20 ± 5	6 ± 4	12 ± 1	6 ± 5	89 ± 3	92 ± 6
	task2	6 ± 3	24 ± 7	62 ± 5	92 ± 4	69 ± 9	36 ± 4	49 ± 11	21 ± 3	8 ± 3	7 ± 6	11 ± 2	85 ± 5	78 ± 14
	task3	2 ± 3	14 ± 6	82 ± 8	87 ± 5	87 ± 5	49 ± 10	55 ± 6	13 ± 6	3 ± 4	1 ± 2	4 ± 3	89 ± 5	86 ± 6
	task4	2 ± 1	11 ± 7	34 ± 6	59 ± 2	35 ± 11	13 ± 7	22 ± 7	3 ± 3	2 ± 2	3 ± 1	2 ± 1	48 ± 12	41 ± 10
	task5	1 ± 1	6 ± 3	54 ± 4	63 ± 7	52 ± 12	22 ± 8	12 ± 5	8 ± 3	1 ± 1	3 ± 1	3 ± 1	53 ± 3	68 ± 15
	overall	3 ± 1	17 ± 2	62 ± 2	77 ± 3	64 ± 4	35 ± 5	42 ± 3	13 ± 1	4 ± 2	5 ± 1	5 ± 2	73 ± 4	73 ± 2
cube-single	task1	7 ± 9	17 ± 5	87 ± 3	97 ± 3	91 ± 5	64 ± 6	98 ± 2	6 ± 7	13 ± 8	4 ± 3	7 ± 7	98 ± 2	97 ± 2
	task2	8 ± 6	18 ± 12	93 ± 4	96 ± 3	96 ± 4	61 ± 8	99 ± 1	8 ± 4	9 ± 5	5 ± 4	6 ± 12	97 ± 3	97 ± 4
	task3	9 ± 6	22 ± 5	93 ± 2	99 ± 1	98 ± 1	69 ± 9	98 ± 2	9 ± 3	14 ± 4	1 ± 2	2 ± 3	99 ± 1	98 ± 1
	task4	6 ± 2	20 ± 6	85 ± 7	92 ± 4	93 ± 2	56 ± 10	97 ± 1	2 ± 2	4 ± 3	3 ± 2	27 ± 17	93 ± 6	94 ± 5
	task5	5 ± 5	14 ± 7	82 ± 4	89 ± 5	94 ± 3	66 ± 18	98 ± 3	3 ± 2	3 ± 3	0 ± 0	24 ± 21	87 ± 7	94 ± 7
	overall	7 ± 2	18 ± 5	88 ± 2	95 ± 1	95 ± 2	63 ± 5	98 ± 1	6 ± 3	9 ± 3	3 ± 1	13 ± 8	95 ± 2	96 ± 3
cube-double	task1	6 ± 5	17 ± 8	88 ± 4	92 ± 6	48 ± 10	77 ± 8	40 ± 8	7 ± 4	8 ± 2	6 ± 2	1 ± 2	73 ± 5	84 ± 14
	task2	0 ± 0	1 ± 1	78 ± 5	84 ± 7	0 ± 0	42 ± 9	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	23 ± 7	81 ± 8
	task3	0 ± 0	0 ± 0	75 ± 6	85 ± 2	0 ± 0	39 ± 10	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	30 ± 11	73 ± 5
	task4	0 ± 0	1 ± 2	8 ± 5	12 ± 9	0 ± 0	1 ± 1	0 ± 0	0 ± 0	1 ± 1	0 ± 0	0 ± 0	3 ± 3	2 ± 3
	task5	0 ± 0	2 ± 2	47 ± 14	45 ± 8	1 ± 1	17 ± 4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	18 ± 7	29 ± 8
	overall	1 ± 1	4 ± 2	59 ± 2	64 ± 4	10 ± 2	35 ± 3	8 ± 2	1 ± 1	2 ± 0	1 ± 0	0 ± 0	30 ± 5	54 ± 4
scene	task1	14 ± 8	60 ± 6	97 ± 3	99 ± 1	96 ± 4	71 ± 13	72 ± 14	19 ± 8	13 ± 3	43 ± 4	32 ± 8	97 ± 2	98 ± 4
	task2	2 ± 1	13 ± 3	92 ± 4	96 ± 2	94 ± 2	14 ± 4	26 ± 8	2 ± 2	1 ± 2	9 ± 2	2 ± 2	95 ± 3	94 ± 7
	task3	2 ± 3	21 ± 9	83 ± 8	89 ± 8	78 ± 6	33 ± 9	25 ± 15	1 ± 1	0 ± 0	3 ± 4	2 ± 3	97 ± 3	96 ± 3
	task4	3 ± 1	19 ± 8	43 ± 28	13 ± 11	79 ± 7	12 ± 3	4 ± 3	6 ± 3	5 ± 5	2 ± 2	3 ± 1	76 ± 17	88 ± 4
	task5	0 ± 0	3 ± 4	23 ± 16	10 ± 9	31 ± 13	2 ± 2	1 ± 1	1 ± 1	0 ± 0	0 ± 0	0 ± 0	18 ± 9	57 ± 4
	overall	4 ± 2	23 ± 2	68 ± 10	61 ± 3	75 ± 3	26 ± 5	25 ± 5	6 ± 2	4 ± 1	12 ± 1	8 ± 2	77 ± 2	87 ± 3
puzzle-3x3	task1	4 ± 4	10 ± 1	4 ± 2	99 ± 1	99 ± 1	15 ± 8	93 ± 12	3 ± 3	8 ± 2	5 ± 1	7 ± 3	99 ± 1	99 ± 2
	task2	1 ± 2	1 ± 1	2 ± 2	99 ± 1	99 ± 1	6 ± 3	90 ± 10	0 ± 0	2 ± 2	1 ± 1	2 ± 2	99 ± 1	96 ± 4
	task3	1 ± 1	1 ± 1	2 ± 1	99 ± 1	99 ± 1	1 ± 1	69 ± 15	0 ± 0	1 ± 2	0 ± 0	1 ± 2	100 ± 0	98 ± 2
	task4	0 ± 0	1 ± 1	1 ± 1	98 ± 2	99 ± 1	1 ± 1	67 ± 15	0 ± 0	1 ± 1	2 ± 3	1 ± 1	98 ± 1	98 ± 2
	task5	1 ± 1	2 ± 2	1 ± 1	95 ± 2	99 ± 1	2 ± 2	83 ± 14	0 ± 0	1 ± 1	1 ± 1	0 ± 0	99 ± 1	98 ± 2
	overall	1 ± 1	3 ± 1	2 ± 1	98 ± 0	99 ± 0	5 ± 1	81 ± 12	1 ± 1	3 ± 1	2 ± 0	2 ± 0	99 ± 0	98 ± 1
puzzle-4x4	task1	1 ± 1	1 ± 1	6 ± 4	33 ± 14	9 ± 7	1 ± 1	7 ± 2	0 ± 0	0 ± 0	1 ± 1	0 ± 0	47 ± 5	94 ± 1
	task2	0 ± 0	1 ± 1	4 ± 4	0 ± 0	6 ± 3	0 ± 0	3 ± 4	0 ± 0	0 ± 0	0 ± 0	1 ± 1	17 ± 5	34 ± 4
	task3	0 ± 0	1 ± 1	6 ± 1	58 ± 10	4 ± 4	1 ± 1	4 ± 3	0 ± 0	0 ± 0	1 ± 1	0 ± 0	38 ± 13	85 ± 2
	task4	1 ± 1	1 ± 1	6 ± 6	22 ± 5	8 ± 4	1 ± 1	4 ± 3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	34 ± 2	77 ± 2
	task5	0 ± 0	0 ± 0	6 ± 1	29 ± 9	3 ± 3	0 ± 0	7 ± 4	0 ± 0	0 ± 0	1 ± 1	0 ± 0	32 ± 6	76 ± 3
	overall	0 ± 0	1 ± 0	5 ± 2	28 ± 4	6 ± 3	0 ± 0	5 ± 2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	34 ± 4	73 ± 1

Table 16. Full results on pixel-based OGBench tasks.

Environment	Task	BC	FBC	IVL	IQL	TD-n	CRL	MC	QRL	TMD	TDP	COE	TRL	DCRL
visual-cube-single	task1	12 ± 4	15 ± 8	70 ± 3	42 ± 12	73 ± 13	47 ± 20	52 ± 7	68 ± 9	9 ± 3	1 ± 1	0 ± 0	80 ± 9	77 ± 10
	task2	4 ± 4	5 ± 3	65 ± 9	44 ± 11	64 ± 16	40 ± 20	55 ± 11	35 ± 34	5 ± 1	1 ± 1	0 ± 0	67 ± 12	68 ± 11
	task3	6 ± 5	4 ± 4	49 ± 2	24 ± 9	60 ± 11	33 ± 17	57 ± 5	41 ± 27	10 ± 3	2 ± 1	0 ± 0	78 ± 9	62 ± 11
	task4	0 ± 0	0 ± 1	60 ± 13	21 ± 6	52 ± 15	18 ± 16	39 ± 6	32 ± 10	0 ± 1	0 ± 0	0 ± 0	67 ± 11	48 ± 10
	task5	0 ± 0	2 ± 2	55 ± 6	20 ± 7	56 ± 15	16 ± 10	39 ± 11	30 ± 8	0 ± 1	0 ± 0	0 ± 0	67 ± 10	53 ± 13
	overall	5 ± 1	5 ± 3	60 ± 5	30 ± 5	61 ± 12	31 ± 15	48 ± 2	41 ± 15	5 ± 1	1 ± 0	0 ± 0	72 ± 6	62 ± 8
visual-cube-double	task1	4 ± 3	7 ± 1	44 ± 8	6 ± 4	47 ± 17	7 ± 4	16 ± 3	20 ± 3	3 ± 2	0 ± 1	0 ± 0	50 ± 14	45 ± 16
	task2	0 ± 0	2 ± 1	0 ± 1	0 ± 0	9 ± 7	0 ± 0	0 ± 0	2 ± 2	0 ± 0	0 ± 0	0 ± 0	5 ± 4	12 ± 9
	task3	0 ± 1	2 ± 3	0 ± 0	0 ± 0	11 ± 9	0 ± 0	0 ± 0	2 ± 0	0 ± 0	0 ± 0	0 ± 0	3 ± 4	10 ± 7
	task4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1
	task5	0 ± 0	0 ± 0	4 ± 2	0 ± 0	3 ± 2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	2 ± 3	8 ± 7
	overall	1 ± 1	2 ± 0	10 ± 2	1 ± 1	14 ± 6	2 ± 1	3 ± 0	5 ± 0	1 ± 0	0 ± 0	0 ± 0	12 ± 3	15 ± 6
visual-cube-triple	task1	73 ± 8	98 ± 1	68 ± 8	76 ± 6	92 ± 5	85 ± 12	64 ± 16	81 ± 3	66 ± 2	1 ± 0	0 ± 0	67 ± 7	94 ± 5
	task2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1
	task3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	2 ± 2	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	4 ± 3
	task4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
	task5	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1	1 ± 1
	overall	15 ± 2	20 ± 0	14 ± 2	15 ± 1	19 ± 1	17 ± 2	13 ± 3	16 ± 1	13 ± 0	0 ± 0	0 ± 0	13 ± 1	20 ± 1
visual-scene	task1	59 ± 7	73 ± 6	84 ± 4	56 ± 4	85 ± 9	52 ± 6	62 ± 7	44 ± 6	58 ± 6	34 ± 6	0 ± 1	60 ± 11	83 ± 7
	task2	0 ± 0	6 ± 4	24 ± 8	1 ± 1	51 ± 14	1 ± 1	7 ± 4	2 ± 2	1 ± 0	0 ± 0	0 ± 0	17 ± 9	45 ± 14
	task3	0 ± 0	0 ± 0	16 ± 8	0 ± 0	32 ± 15	0 ± 0	3 ± 1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	14 ± 8	40 ± 14
	task4	2 ± 1	4 ± 1	0 ± 0	3 ± 4	23 ± 10	1 ± 1	6 ± 3	2 ± 1	3 ± 2	1 ± 1	0 ± 0	2 ± 3	29 ± 6
	task5	0 ± 0	0 ± 0	0 ± 0	0 ± 0	3 ± 3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1	6 ± 5
	overall	12 ± 2	17 ± 1	25 ± 3	12 ± 2	39 ± 5	11 ± 2	16 ± 2	10 ± 1	12 ± 1	7 ± 1	0 ± 0	19 ± 3	41 ± 5
visual-puzzle-3x3	task1	1 ± 1	1 ± 0	97 ± 2	5 ± 9	97 ± 3	2 ± 1	94 ± 4	3 ± 3	2 ± 1	1 ± 1	0 ± 0	96 ± 3	98 ± 2
	task2	0 ± 0	0 ± 1	1 ± 1	0 ± 0	94 ± 8	0 ± 0	49 ± 6	0 ± 0	0 ± 0	0 ± 0	0 ± 0	83 ± 10	95 ± 5
	task3	0 ± 0	0 ± 0	0 ± 1	0 ± 0	89 ± 10	0 ± 0	12 ± 3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	61 ± 16	87 ± 8
	task4	0 ± 0	0 ± 0	2 ± 3	0 ± 0	84 ± 12	0 ± 0	8 ± 4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	44 ± 21	85 ± 8
	task5	0 ± 0	0 ± 0	3 ± 2	0 ± 0	93 ± 6	0 ± 0	24 ± 6	0 ± 0	0 ± 0	0 ± 0	0 ± 0	78 ± 10	93 ± 4
	overall	0 ± 0	0 ± 0	21 ± 1	1 ± 2	92 ± 7	0 ± 0	37 ± 4	1 ± 1	0 ± 0	0 ± 0	0 ± 0	73 ± 10	91 ± 4
visual-puzzle-4x4	task1	11 ± 3	2 ± 2	86 ± 7	18 ± 5	94 ± 6	10 ± 7	17 ± 9	0 ± 0	11 ± 3	0 ± 0	0 ± 0	68 ± 9	96 ± 3
	task2	19 ± 5	6 ± 7	8 ± 5	28 ± 12	82 ± 9	19 ± 13	4 ± 2	0 ± 0	18 ± 6	0 ± 0	0 ± 0	38 ± 8	91 ± 5
	task3	8 ± 3	1 ± 2	80 ± 4	15 ± 4	91 ± 5	8 ± 7	5 ± 1	0 ± 0	9 ± 3	0 ± 0	0 ± 0	56 ± 13	91 ± 4
	task4	6 ± 2	2 ± 1	65 ± 9	10 ± 5	92 ± 4	7 ± 5	3 ± 2	0 ± 0	6 ± 1	0 ± 0	0 ± 0	48 ± 9	95 ± 3
	task5	6 ± 2	2 ± 2	61 ± 8	9 ± 3	88 ± 6	4 ± 4	2 ± 1	0 ± 0	7 ± 2	0 ± 0	0 ± 0	46 ± 13	90 ± 7
	overall	10 ± 1	2 ± 2	60 ± 5	16 ± 4	89 ± 4	10 ± 6	6 ± 2	0 ± 0	10 ± 1	0 ± 0	0 ± 0	51 ± 8	93 ± 3
visual-puzzle-4x5	task1	22 ± 8	14 ± 4	86 ± 4	31 ± 8	99 ± 2	31 ± 6	17 ± 7	0 ± 0	36 ± 9	0 ± 0	0 ± 0	96 ± 5	99 ± 1
	task2	1 ± 1	0 ± 1	0 ± 0	1 ± 1	55 ± 36	0 ± 1	0 ± 0	0 ± 0	0 ± 1	0 ± 0	0 ± 0	19 ± 19	96 ± 5
	task3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	31 ± 25	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	19 ± 20	81 ± 16
	task4	0 ± 0	0 ± 0	0 ± 0	0 ± 0	33 ± 23	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 2	64 ± 20
	task5	0 ± 0	0 ± 0	0 ± 0	0 ± 0	20 ± 23	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	2 ± 3	73 ± 20
	overall	5 ± 2	3 ± 1	17 ± 1	7 ± 2	48 ± 21	6 ± 1	3 ± 1	0 ± 0	7 ± 2	0 ± 0	0 ± 0	27 ± 9	83 ± 11
visual-antmaze-medium	task1	17 ± 6	16 ± 4	30 ± 7	16 ± 3	89 ± 4	92 ± 2	89 ± 4	0 ± 0	18 ± 4	18 ± 4	24 ± 4	80 ± 6	86 ± 6
	task2	8 ± 2	2 ± 2	21 ± 6	7 ± 2	81 ± 11	94 ± 2	94 ± 3	0 ± 0	6 ± 3	4 ± 1	9 ± 3	80 ± 7	87 ± 7
	task3	17 ± 1	14 ± 4	24 ± 5	16 ± 4	90 ± 5	98 ± 1	93 ± 3	0 ± 0	13 ± 2	16 ± 3	11 ± 1	82 ± 8	89 ± 4
	task4	12 ± 2	8 ± 3	21 ± 3	9 ± 2	86 ± 6	94 ± 2	90 ± 4	0 ± 0	14 ± 4	13 ± 4	11 ± 5	51 ± 14	84 ± 6
	task5	4 ± 2	3 ± 1	16 ± 5	6 ± 2	86 ± 5	94 ± 2	96 ± 1	0 ± 0	4 ± 1	6 ± 1	11 ± 2	74 ± 10	91 ± 6
	overall	11 ± 2	9 ± 2	22 ± 2	11 ± 1	86 ± 4	94 ± 1	92 ± 0	0 ± 0	11 ± 1	11 ± 1	13 ± 2	73 ± 4	87 ± 3
visual-antmaze-large	task1	3 ± 1	3 ± 1	7 ± 2	4 ± 3	70 ± 7	78 ± 5	64 ± 7	0 ± 0	5 ± 2	3 ± 1	4 ± 3	53 ± 8	73 ± 6
	task2	4 ± 3	0 ± 1	4 ± 1	2 ± 1	18 ± 10	80 ± 3	35 ± 10	0 ± 0	2 ± 1	2 ± 1	2 ± 1	2 ± 2	22 ± 9
	task3	4 ± 2	0 ± 0	6 ± 2	4 ± 1	68 ± 11	90 ± 3	92 ± 4	1 ± 1	3 ± 2	2 ± 2	4 ± 3	44 ± 12	64 ± 7
	task4	4 ± 2	3 ± 0	5 ± 3	6 ± 1	59 ± 12	88 ± 3	61 ± 14	0 ± 1	3 ± 1	5 ± 1	3 ± 2	10 ± 8	59 ± 10
	task5	4 ± 2	2 ± 1	5 ± 1	4 ± 2	56 ± 12	83 ± 2	49 ± 17	0 ± 0	4 ± 2	5 ± 2	4 ± 2	13 ± 9	54 ± 11
	overall	4 ± 0	2 ± 0	5 ± 1	4 ± 1	54 ± 6	84 ± 1	60 ± 4	0 ± 0	4 ± 1	3 ± 0	3 ± 2	25 ± 5	54 ± 4
visual-antmaze-giant	task1	0 ± 0	0 ± 0	0 ± 0	0 ± 0	1 ± 1	17 ± 2	1 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1	0 ± 1
	task2	1 ± 1	0 ± 0	2 ± 1	1 ± 1	14 ± 5	73 ± 9	16 ± 5	0 ± 0	2 ± 1	1 ± 1	0 ± 0	2 ± 2	14 ± 4
	task3	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1	22 ± 6	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 1	0 ± 1
	task4	0 ± 1	0 ± 0	0 ± 1	0 ± 0	2 ± 2	47 ± 5	1 ± 1	0 ± 0	0 ± 1	0 ± 0	0 ± 0	0 ± 0	3 ± 2
	task5	1 ± 1	0 ± 0	2 ± 3	1 ± 1	28 ± 10	77 ± 5	9 ± 5	0 ± 1	1 ± 1	1 ± 1	0 ± 0	2 ± 2	35 ± 14
	overall	0 ± 0	0 ± 0	1 ± 1	0 ± 0	9 ± 3	47 ± 2	5 ± 1	0 ± 0	1 ± 0	1 ± 0	0 ± 0	1 ± 1	11 ± 3

K Hyperparameters

Table 17. Common hyperparameters.

Hyperparameter	Value
Optimizer	Adam (Kingma & Ba, 2015)
Learning rate	0.0003
Batch size	1024 (state-based), 256 (pixel-based)
Number of slots	128 (state-based), 64 (pixel-based)
Nonlinearity	GELU (Hendrycks & Gimpel, 2016)
Target network update rate	0.005
Critic ensemble size	2
Critic target	$\texttt{min}(Q_1, Q_2)$ for `cube-*` and CALVIN, otherwise $\texttt{mean}(Q_1, Q_2)$
Number of flow steps	10
Policy extraction	Reparameterized gradients or rejection sampling (see Table 21)

Goal sampling. For a state s_i in a trajectory $(s_0, \dots, s_H)$, we sample each policy or value goal from the mixture $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$, whose entries sum to one. The four components select, respectively: (i) the current state s_i; (ii) a future state $s_{\min(i+\Delta, H)}$, where $\Delta \sim \text{Geom}(1-\gamma)$ on $\{1, 2, \dots\}$; (iii) a future state s_j with $j \sim \text{Unif}\{i+1, \dots, H\}$; and (iv) a state sampled uniformly from the offline dataset $\mathcal{D}$. Policy and value goals are sampled independently using the ratios reported in Tables 18 to 20.

Table 18. Hyperparameters for long-horizon OGBench tasks.

Hyperparameter	Value
Gradient steps	10^6
MLP size	$[1024, 1024, 1024, 1024]$
Discount factor γ	0.999
n-step length (n) (`TD-n`, `DCRL`)	100 (`humanoidmaze`, `puzzle`), 25 (`cube`)
Policy $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio	$(0, 0, 1, 0)$ (`humanoidmaze`), $(0, 0.5, 0, 0.5)$ (`puzzle`)
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`TRL`, `TD`, `MC`)	$(0, 0, 1, 0)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`CRL`)	$(0, 1, 0, 0)$
Behavior Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`DCRL`)	$(0, 0, 1, 0)$
Optimal Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`DCRL`)	$(0.2, 0.5, 0, 0.3)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (others)	$(0.2, 0.5, 0, 0.3)$

Table 19. Hyperparameters for standard OGBench tasks.

Hyperparameter	Value
Gradient steps	10^6 (state-based), 5×10^5 (pixel-based)
MLP size	$[512, 512, 512]$
Discount factor γ	0.99 (default), 0.995 (`humanoidmaze`)
n-step length (n) (`TD-n`, `DCRL`)	25
Image augmentation probability (pixel-based)	0.5 (manipulation), 0.0 (locomotion)
Random-crop padding (pixel-based)	3
Policy $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio	$(0, 0, 1, 0)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`TRL`, `CRL`)	$(0, 1, 0, 0)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`TD`, `MC`)	$(0, 0, 1, 0)$
Behavior Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`DCRL`)	$(0, 0, 1, 0)$
Optimal Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (`DCRL`)	$(0.2, 0.5, 0, 0.3)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (others)	$(0.2, 0.5, 0, 0.3)$

Table 20. Hyperparameters for CALVIN.

Hyperparameter	Value
Gradient steps	5×10^5 (state-based), 2×10^5 (pixel-based)
Learning rate	3×10^{-4}
Actor MLP size	$[256, 256]$
Value MLP size	$[512, 512, 512]$
Discount factor γ	0.99
n-step length (n) (TD-n, DCRL)	25
Image augmentation probability (pixel-based)	0.5
Random-crop padding (pixel-based)	6
Policy $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio	$(0, 0.5, 0, 0.5)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (TRL, TD, MC, CRL)	$(0, 1, 0, 0)$
Behavior Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (DCRL)	$(0, 1, 0, 0)$
Optimal Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (DCRL)	$(0.2, 0.5, 0, 0.3)$
Value $(p^{\mathcal{D}}_{\text{curr}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{rand}})$ ratio (others)	$(0.2, 0.5, 0, 0.3)$
Gradient clipping norm	1.0
Subgoal (waypoint) steps (HIQL)	25
AWR temperature (low-level / high-level) (HIQL)	1 / 1
High-level random-goal probability (HIQL)	0.3
Latent subgoal representation (HIQL)	False (state-based), True (pixel-based)
High-level action dimension (HIQL)	21 (state-based), 527 (pixel-based)

Table 21. Task-specific hyperparameters for all methods. (α: BC coefficient for reparameterized gradients, N: sample count for rejection sampling, M: subgoal count, P: random goal distance, β: goal regularization weight, κ: expectile, λ: distance re-weighting factor.) For DCRL, κ denotes κ_{prop}; we fix $\kappa_{\text{dc}} = 0.5$. For hyperparameters not listed here, we adopt the default values from Eysenbach et al. (2022); Park et al. (2023; 2025a; 2026).

Environment	QRL α	TDP $(\alpha, N), M, P$	COE $(\alpha), \beta$	TD (α, N)	MC (α, N)	TMD α	{TD-n,TRL,DCRL} (α, N)	{TRL,DCRL} κ	TRL λ
humanoidmaze-giant-1B	0.001	(3, -), 8, 1000	1, 0.5	(0.3, -)	(0.3, -)	-	(-, 32)	0.7	0
puzzle-4x5-1B	3	(-, 32), 8, 1000	3, 10	(-, 32)	(-, 32)	-	(-, 32)	0.7	0
puzzle-4x6-1B	3	(-, 32), 8, 1000	3, 10	(-, 32)	(-, 32)	-	(-, 32)	0.7	0
cube-quadruple-100M	3	(-, 4), 8, 1000	3, 10	(-, 4)	(-, 4)	-	(-, 4)	0.7	0
cube-octuple-1B	3	(-, 4), 8, 1000	3, 10	(-, 4)	(-, 4)	-	(-, 4)	0.7	0
pointmaze-large	0.0003	(5, -), 8, 500	1, 1	(10, -)	(2, -)	0.03	(2, -)	0.7	0.7
antmaze-large	0.003	(10, -), 8, 100	3, 10	(0.7, -)	(0.7, -)	0.1	(0.7, -)	0.7	0
antsoccer-arena	0.003	(10, -), 8, 200	0.3, 1	(0.3, -)	(0.3, -)	0.3	(0.3, -)	0.7	0.5
humanoidmaze-medium	0.001	(5, -), 8, 500	0.1, 3	(0.1, -)	(0.1, -)	0.1	(0.1, -)	0.7	0
humanoidmaze-large	0.001	(5, -), 8, 500	0.1, 1	(0.1, -)	(0.1, -)	0.1	(0.1, -)	0.7	0.1
cube-single	0.3	(5, -), 8, 500	0.3, 10	(1, -)	(1, -)	3.0	(1, -)	0.7	0.7
cube-double	0.3	(5, -), 8, 500	0.3, 10	(10, -)	(2, -)	3.0	(2, -)	0.7	1
scene	0.3	(1, -), 16, 200	1, 10	(1, -)	(1, -)	3.0	(1, -)	0.7	1
puzzle-3x3	0.3	(5, -), 8, 500	3, 10	(2, -)	(2, -)	3.0	(2, -)	0.7	0.5
puzzle-4x4	0.3	(5, -), 8, 500	1, 10	(2, -)	(1, -)	3.0	(1, -)	0.7	2
visual-cube-single	0.3	(5, -), 16, 500	0.3, 10	-	(2, -)	3.0	(-, 8)	0.7	0
visual-cube-double	0.3	(5, -), 16, 500	0.3, 10	-	(0.7, -)	3.0	(-, 8)	0.7	0
visual-cube-triple	0.3	(5, -), 16, 500	0.3, 10	-	(0.7, -)	3.0	(-, 8)	0.7	0
visual-scene	0.3	(1, -), 16, 200	1, 10	-	(1, -)	3.0	(-, 8)	0.7	0
visual-puzzle-3x3	0.3	(5, -), 16, 500	3, 10	-	(2, -)	3.0	(-, 8)	0.7	0
visual-puzzle-4x4	0.3	(5, -), 16, 500	1, 10	-	(1, -)	3.0	(-, 8)	0.7	0
visual-puzzle-4x5	0.3	(5, -), 16, 500	1, 10	-	(0.5, -)	3.0	(-, 8)	0.7	0
visual-antmaze-medium	0.003	(10, -), 8, 100	3, 10	-	(0.3, -)	0.1	(-, 8)	0.7	0
visual-antmaze-large	0.003	(10, -), 8, 100	3, 10	-	(0.3, -)	0.1	(-, 8)	0.7	0
visual-antmaze-giant	0.003	(10, -), 8, 100	3, 10	-	(0.3, -)	0.1	(-, 8)	0.7	0
CALVIN	0.003	-	-	-	(-, 32)	1.0	(-, 32)	0.7	0