Reward-Conditioned Reinforcement Learning

Michal Nauman
University of Warsaw
nauman.mic@gmail.com

Marek Cygan
University of Warsaw, Nomagic

Pieter Abbeel
UC Berkeley, Amazon FAR

Abstract

Sample-efficient reinforcement learning typically uses a fixed reward function both to guide exploration and to supervise learning. However, the same experience can provide supervision under multiple related reward specifications, even when the objective of interest remains fixed. We introduce *Reward-Conditioned Reinforcement Learning* (RCRL), an off-policy approach that collects experience under a nominal objective while training a reward-conditioned agent using counterfactual rewards recomputed from replay. RCRL constructs alternative reward specifications by perturbing the nominal reward, providing diverse supervision without additional environment interaction. Across single-task, multi-task, and vision-based control benchmarks, RCRL improves nominal task performance and sample efficiency when integrated with strong RL backbones. As a secondary capability, the learned policies support zero-shot behavioral adjustment toward related objectives without further training. These results demonstrate that diverse reward supervision can improve learning of a predefined objective while also enabling policy steerability.

1 Introduction

Training on diverse tasks and objectives is a common strategy for improving representations and performance in machine learning [2, 26]. In reinforcement learning (RL), however, agents trained for a known task typically use a single reward to both guide exploration and supervise learning [23, 29]. Yet choosing one objective for interaction does not require learning from only one reward. To this end, we ask whether diverse reward supervision can improve learning on a predefined task of interest.

Many control problems use rewards that combine task progress, tracking, and control costs [63, 52] and thus provide an opportunity to broaden supervision without collecting additional experience. Standard RL combines these components into a scalar reward and uses only that reward to supervise learning [62]. But these components can be recombined, and the same transition can be evaluated under many objectives. Such counterfactual rewards can provide alternative learning targets from the same experience. However, it is unclear if such diverse reward supervision improves the original task.

We study this question through **Reward-Conditioned Reinforcement Learning (RCRL)**. RCRL relabels replayed transitions with alternative reward specifications and conditions the agent on the specifications used for each update. We construct these either by perturbing the nominal rewards or by reusing reward functions from the related tasks. Because rewards are recomputed from components stored in replay, RCRL adds diverse supervision without additional environment interactions, while

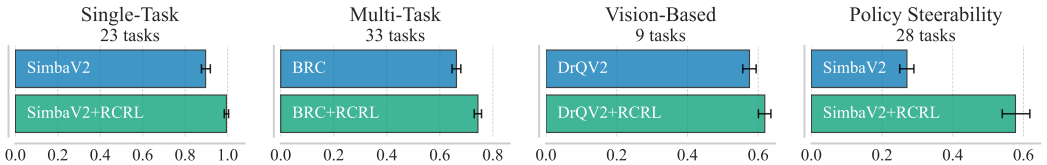


Figure 1: ***Results summary.*** RCRL improves the performance of strong baselines in both single and multi-task settings (Figure 3). As a secondary capability, RCRL increases the steerability of the trained policy without further training (Figure 10).

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

exploration focuses fixed *nominal objective*. We evaluate RCRL with strong backbones across single-task, multi-task, and vision-based control, including SIMBAV2 [45], BRC [48], and DRQV2 [75]. Across these, RCRL improves nominal task sample efficiency, while allowing for steering toward related objectives without further training. This shows that diverse reward supervision can improve learning of the target objective while making the policy more steerable. Our findings are as follows:

- **Improved Sample Efficiency** – Training with diverse reward supervision via RCRL improves sample efficiency of strong RL backbones, despite evaluation being focused only on the nominal task. (Figures 1, 3 & 4).
- **Improved Steerability** – As a secondary capability, RCRL enables a trained policy to adjust its behavior without further training. Diverse exploration strengthens this steerability at cost to nominal performance (Figures 10 & 11).

2 Related Work

In RCRL, environment interactions follow a predefined objective, while diverse rewards are used for supervision. This setup is motivated by improving learning of the predefined task, while also allowing for steerability within the reward family. Table below summarizes the main differences between the proposed and related approaches. We expand the related work discussion in Appendix C.

Method	Interaction	Supervision	Primary use
Single-task RL	Single objective	Single reward	Nominal task learning
RCRL	**Single objective**	**Diverse rewards**	**Nominal task learning**
MTRL / MORL	Diverse objectives	Diverse rewards	Policy steerability
USFA	Diverse objectives	Successor features	Policy steerability
FB	Broad reward-free data	Occupancy representations	Policy steerability

Sample-efficient single-task RL. A body of work aims to improve the efficiency and stability of deep RL when learning a fixed objective, through techniques such as increased data reuse [18], improved value estimation [10, 23], exploration [17], and scalable network architectures [47, 11]. Recent methods achieve strong performance by combining several of these advances [45]. RCRL operates in the same setting and is complementary to these improvements, but rather than changing how transitions are collected or increasing the number of gradient steps, it extracts additional supervision from existing replay by evaluating each transition under diverse reward specifications. Our experiments therefore treat strong single-task algorithms as backbones and ask whether diverse reward supervision can further improve their learning efficiency.

Multi-task and multi-objective RL. Multi-task RL (MTRL) trains shared agents across multiple tasks [64, 33, 77, 43, 48], while multi-objective RL (MORL) learns policies across combinations of potentially competing objectives, often by conditioning on preferences or reward weights [55, 73, 1, 32]. RCRL uses related conditioning machinery but studies a different setting. Alternative reward specifications are used to relabel experience collected for a predefined nominal objective, rather than to define the objectives under which that experience must be collected. This separates reward diversity in supervision from reward diversity in exploration and lets us ask whether learning across objectives can improve the objective of interest itself.

Goal-conditioned RL. This approach defines tasks in terms of goal states, typically using sparse reward functions that indicate success with respect to a given goal [40, 56]. In techniques like HER, the agent is conditioned on a goal representation, allowing a single agent to learn behaviors for multiple goals within the same environment [5, 4], supporting adaptation to new goals [19, 20, 51]. RCRL applies a closely related idea to reward specifications rather than goal states. RCRL stores the components required to evaluate a family of rewards, recomputes scalar rewards on replay, and optimizes them through the same objective as the nominal reward. Unlike HER, which is primarily motivated by learning from goal-reaching rewards, RCRL studies whether relabeling with alternative dense rewards can improve a known objective when exploring under a single objective.

Successor features and reward-conditioned representations. Successor features (SFs) factor value functions into expected state-action features and reward weights, enabling transfer across rewards that are linear in a shared feature representation [8, 9]. Universal successor feature approximators (USFAs) extend this idea by conditioning successor features on task descriptors and using generalized

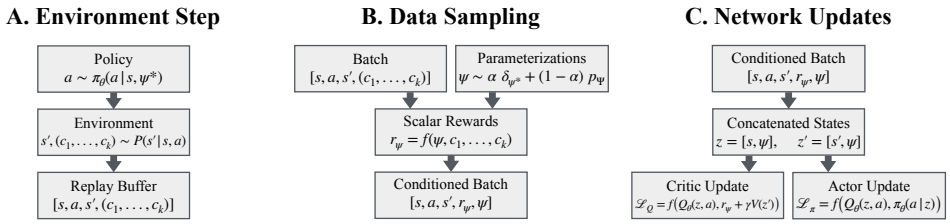


Figure 2: ***Overview of RCRL.*** During interaction, actions are sampled from the policy conditioned on the nominal reward $\psi^\star$, with reward components stored in the buffer. During training, each transition is paired with an independently sampled reward specification $\psi \sim \mathcal{P}_\Psi$ to compute the scalar reward r_ψ. The training then follows with the agent conditioned on ψ.

policy improvement for transfer [12]. Forward-backward (FB) representations similarly learn reward-independent occupancy representations that can be mapped to new rewards after training [66, 67, 78]. Additionally, recent approaches learn representations from broad offline data, often without rewards [22, 36]. RCRL shares the goal of representing multiple reward-dependent behaviors, but uses a different learning mechanism and data regime. Firstly, it directly trains scalar reward-conditioned value functions from recomputed rewards. Furthermore, RCRL learns from data generated online by the known nominal objective rather than from broad reward-free or multi-objective exploration.

3 Reward-Conditioned Reinforcement Learning

We consider rewards of the form $r_\psi(s, a) = f\big(\psi, c_1(s, a), \ldots, c_k(s, a)\big)$, where $c_1, \ldots, c_k$ are measurable reward components and ψ specifies how they are combined into a scalar reward. Such structure is common in control, where rewards often combine terms for task progress, pose or velocity tracking, and control costs. Usually, single-task RL optimizes a fixed specification $\psi^\star$. Once the underlying components are available the same transition can be evaluated under many alternative specifications $\psi \in \Psi$. RCRL exploits this structure, as summarized in Figure 2 and Pseudocode 1.

Algorithm 1 Reward-Conditioned Reinforcement Learning

Input: nominal objective $\psi^\star$, alternative specifications Ψ, mixture distribution $\mathcal{P}_\Psi$

$a \sim \pi(a|s, \psi^\star)$ `# explore via a policy conditioned on` $\psi^\star$
$s', c_1, ..., c_k = \text{ENV.STEP}(a)$ `# gather the reward components` $c_1, ...c_k$ `when acting`
$\mathcal{D} \leftarrow \mathcal{D} \cup (s, a, c_1, ..., c_k, s')$ `# store rewards components in the buffer`
$\{s, a, c_1, ..., c_k, s'\}_i \sim \mathcal{D}$ with $i = 1, ..., B$ `# sample B transitions from the buffer`
$r_i = f(\psi_i, c_1, ..., c_k)$ with $\psi_i \sim \mathcal{P}_\Psi$ `# evaluate sampled reward specifications`
$\text{AGENT.UPDATE}(\{s, a, r_\psi, s', \psi\}_i)$ `# update the agent while conditioning on` ψ

The nominal specification $\psi^\star$ defines both the target task and the reward used for environment interaction. RCRL additionally trains on alternative specifications $\psi \in \Psi$ by conditioning the policy and critic on the active specification, $\pi_\theta(a \mid s, \psi)$ and $Q_\phi(s, a, \psi)$. Transitions are stored together with their reward components, allowing their rewards to be recomputed under different specifications during replay. For each sampled transition, RCRL draws a reward specification according to:

$$\mathcal{P}_\Psi = \alpha\ \delta_{\psi^\star} + (1 - \alpha)\ p_\Psi,$$

where $\delta_{\psi^\star}$ is a point mass on the nominal specification $\psi^\star$ and p_Ψ samples alternative reward specifications ψ. The coefficient $\alpha \in [0, 1]$ controls the balance between nominal and counterfactual supervision, with $\alpha = 1$ recovering standard single-task RL. After sampling ψ, the corresponding reward is recomputed from the stored components and the transition is used to update the agent conditioned on ψ. We sample reward specifications independently for each transition in the batch. The alternative specifications affect learning but not the objective used to collect experience, and the agent always conditions on $\psi^\star$ (the nominal objective) for the environment interaction. RCRL therefore learns alternative reward-conditioned behaviors off-policy from nominal task experience. The primary question that we study *is whether this counterfactual supervision improves learning of* $\psi^\star$. As we show in Section 4, it improves nominal performance across several settings, while the learned reward-conditioned policy also supports steering toward related reward specifications.

3.1 Constructing the Set of Reward Specifications

A central design choice in RCRL is how to construct the alternative reward specifications Ψ. We consider two complementary strategies that provide different forms of reward diversity.

Perturbed Reward Conditioning (PRC). In the first approach, we generate continuous variations of the nominal reward specification $\psi^\star$. Many control rewards combine multiple components whose relative contributions are determined by reward parameters, naturally admitting variations. We exploit this structure by applying nonnegative elementwise perturbations $\Delta \in \mathbb{R}^k$ according to $\psi = \psi^\star \odot \Delta$. Sampling Δ from a distribution over nonnegative values centered at unity produces a family of rewards around the nominal objective while preserving the signs of its components. The spread of this distribution controls the diversity of reward supervision and nonnegative perturbations ensure that the qualitative structure of the reward is preserved, including component signs.

Auxiliary Reward Conditioning (ARC). In the second approach, we construct the set of alternative reward specifications Ψ using reward functions corresponding to distinct tasks defined in the same state and action space. Unlike PRC, which generates variations of the nominal reward, ARC considers qualitatively different objectives applied to the same embodiment. This approach is natural in multi-task and multi-objective settings, where related tasks or objectives provide a readily available set of reward functions for relabeling. For example, when the nominal reward $\psi^\star$ defines a `run` objective, additional reward specifications in Ψ may correspond to related behaviors such as `standing` or `walking`. As such, the auxiliary objectives might provide complementary signal or form learning curricula that facilitate the learning (e.g. standing facilitates walking, which in turn supports running).

3.2 Reward Conditioning and Training Stability

As illustrated in Figure 2, RCRL conditions the agent on the reward specification used for each update. In this subsection, we discuss different representations used for this conditioning.

Conditioning on the Reward Specification. For continuously parameterized rewards, such as PRC, the reward parameters ψ or perturbation vector Δ can be concatenated directly to the environment state. For finite reward sets, such as ARC, we assign each reward specification a learnable embedding and provide this representation to both the actor and critic. The embeddings are learned jointly with the value function through the critic objective, following common practice in multi-task RL [48]. In our main experiments, unless stated otherwise, we use embedding-based conditioning. We compare a variety of related conditioning approaches in Subsection 4.2.

Training Stability. Learning across reward specifications can be challenging when their reward scales or induced value functions differ substantially. We therefore retain the stabilization mechanisms of each backbone, including reward normalization, categorical value learning, and scaled critics in SIMBAV2 and BRC. Our ablations show that RCRL benefits particularly from these scaled critic designs (Figure 7), indicating a strong interaction between reward conditioning and critic capacity. However, as our vision-based experiments show, DRQV2 which uses a conventional scalar critic also benefits from RCRL. We provide further analysis in Subsection 4.2 and Appendix D.

4 Experiments

Our experiments ask whether diverse reward supervision improves learning of a fixed nominal objective. We first measure nominal task gains across benchmarks, then study key RCRL design choices and reward compatibility, and finally examine policy steerability beyond the nominal objective. All presented figures show 95% confidence intervals calculated using bootstrapping according to the RLiable methodology [3]. Additional experimental settings are provided in Appendix F.

RCRL Setup. In our experiments, the set of alternative reward specifications Ψ is constructed using one of two strategies introduced in Section 3.1, PRC or ARC. Unless otherwise stated, we set $\alpha = 0.5$, so that approximately half of the training updates use the nominal reward specification $\psi^\star$ and half use alternative specifications. For PRC, Ψ is generated by multiplicatively perturbing the coefficients of the nominal reward. The perturbations Δ_i are sampled from a stratified log-uniform distribution with support $[0.25, 4.0]$. For ARC, Ψ is constructed from task-specific reward functions that can be evaluated on the same state-action transitions as the nominal task. We discuss other implementation details, including exact counterfactual reward definitions for both ARC and PRC in Appendix E.

4.1 RCRL Improves Learning Efficiency

We first evaluate whether diverse reward supervision improves learning of the nominal objective. Across all experiments in this section, we follow the training protocols and hyperparameters of the corresponding backbone and evaluate performance exclusively under the nominal reward. We summarize these runs in Figure 3 and present detailed training curves in Appendix G.

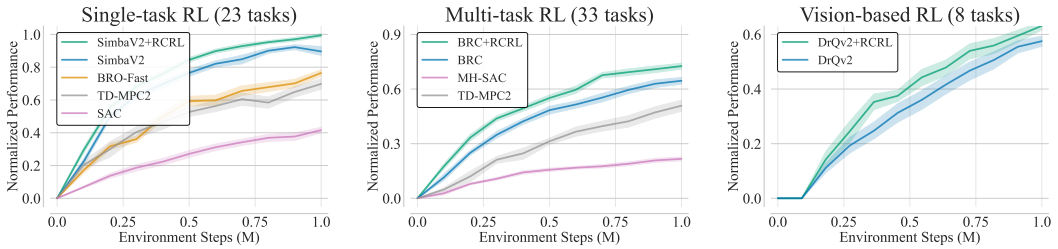


Figure 3: ***RCRL improves learning efficiency.*** The figure presents aggregate training performance across single-task, multi-task, and vision-based control. Adding counterfactual reward supervision via RCRL improves the corresponding backbone while evaluating under the nominal reward. We provide additional experimental details in Appendix F.

Single-task RL. We first augment SIMBAV2 [45] with PRC and evaluate it on the 23-task continuous-control tasks used in the original study, spanning DMC [63], HumanoidBench [60], and OpenAI Gym [14], and using 10 seeds. We list all tasks in Appendix F.1. We preserve the original SIMBAV2 training setup, including the training horizon and hyperparameters, and introduce only the reward-conditioning components required by RCRL. As shown in the left panel of Figure 3, RCRL improves both final and AUC performance by around 10%. These results show that diverse reward supervision via RCRL can improve learning of a fixed nominal objective even when both data collection and evaluation remain focused on the nominal objective.

Multi-task RL. We next augment BRC [48] with ARC and evaluate it on DMC Dogs, HumanoidBench, and HumanoidBench Hard following the original work. We list all tasks in Appendix F.1 and use 5 seeds. We preserve the original BRC training protocol and hyperparameters, and introduce only the components required by RCRL. In this setting, each task continues to collect experience under its own nominal objective, while RCRL additionally evaluates the reward functions of related tasks on replayed transitions. As shown in the middle panel of Figure 3, RCRL improves learning, showing that counterfactual supervision remains beneficial even when applied on top of a strong multi-task learner. ARC is attractive in this setting because trajectories collected for one task can also be evaluated under the reward functions of the other tasks. For N tasks, this creates up to $N(N-1)$ additional pairings from the same environment interactions. As shown on Figure 4, the effect is pronounced on the DMC Dogs. There, BRC+RCRL learns substantially faster than the original multi-task learner, leading to more than twofold improvement in sample efficiency and reducing the variance across seeds. This result shows that diverse reward supervision via RCRL can complement the benefits of multi-task training.

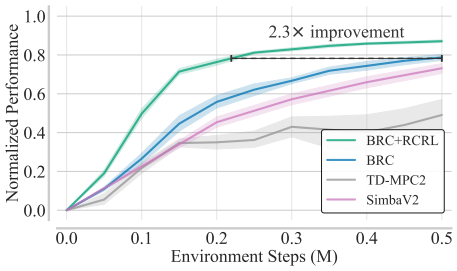


Figure 4: ***Learning on DMC Dogs.*** RCRL yields a 2.3× improvement in sample efficiency, reaching 80% performance after approximately 200k environment steps.

Vision-based RL. We also apply PRC to DRQV2 [75] on 9 image-based DMC tasks listed in Appendix F.1. As shown in the right panel of Figure 3, RCRL improves final performance and AUC by 10% and 17% respectively. We find that RCRL improves learning performance in this setup as well, showing that the effect is not limited to proprioceptive observations or to the architectural choices used by SIMBAV2 or BRC. This provides complementary evidence that counterfactual reward supervision remains beneficial across observation modalities and learning architectures.

Takeaway

Broad, diverse reward supervision via RCRL can outperform specialized training even when the evaluation is restricted to a single predefined nominal objective.

4.2 What Makes RCRL Effective?

Previous subsection shows that diverse reward supervision via RCRL improves nominal task learning. In this subsection, we investigate design choices that drive these improvements. We first disentangle the benefits of diverse reward supervision from broader state-action coverage, then investigate the roles of reward conditioning, exploration, sampling balance, and model capacity.

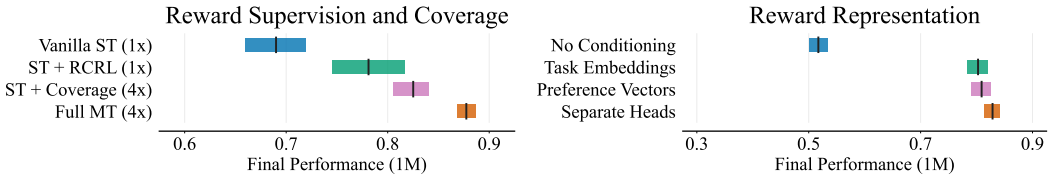


Figure 5: ***Ablation studies.*** The left panel shows that improvements from multi-task learning come from both broader state-action coverage and additional reward supervision; RCRL captures the latter without additional environment interaction. The right panel shows ablations on conditioning the agent on Ψ. Removing conditioning degrades performance significantly, however, all tested methods that condition the agent on the reward information perform similarly well.

Reward supervision and coverage. We next disentangle two sources of multi-task gains: broader state-action coverage and diverse reward supervision. We test BRC on the four DMC Dogs tasks, training for 1M environment steps per task with 5 random seeds. We compare single-task training (Vanilla ST), single-task RCRL (ST+RCRL), single-task training with expanded state-action coverage (ST+Coverage), and full multi-task training (Full MT). Vanilla ST and ST+RCRL use only experience collected under the nominal task and are matched in interaction. ST+Coverage additionally receives replay from single-task runs on the other three tasks, giving it access to $4\times$ as many transitions while still optimizing only the nominal reward. Following tandem RL [50], these transitions are progressively added during training. Full MT uses the same broader experience while also learning from the reward associated with each task. As shown in the left panel of Figure 5, both ST+RCRL and ST+Coverage outperform Vanilla ST. Full MT, which combines broader coverage and diverse reward supervision performs best. This shows that both factors contribute to multi-task gains, while RCRL captures the benefit of diverse reward supervision from the original interaction budget.

Reward representation. RCRL requires the agent to distinguish which reward specification is active, but this conditioning can be implemented in several ways. We compare no conditioning, fixed preference vectors, learned reward embeddings following BRC recipe [48], and separate reward-specific heads, while keeping the rewards and training data fixed under ARC variant of SIMBAV2+RCRL. We evaluate all variants across the 18 tasks from DMC Dogs and HumanoidBench for 1M steps using 10 seeds. As shown in the right panel of Figure 5, removing conditioning degrades performance, whereas the three conditioned variants perform similarly. This indicates that identifying the active reward is essential for benefiting from diverse supervision, while the particular representation of that identity is comparatively unimportant. In practice, RCRL therefore does not depend on a specialized conditioning architecture.

Reward supervision and exploration. We next test whether alternative rewards should also influence environment interaction. Across the same 18 tasks, we compare single-task SIMBAV2, SIMBAV2+RCRL, and SIMBAV2+RCRL+EXPLORE, evaluating both ARC and PRC approaches, and training each for 500k environment steps with 10 random seeds. Standard RCRL collects experience only under the nominal reward, whereas RCRL+Explore samples the reward function used for exploration from the same distribution used for training, and resampling every environment step. We evaluate both ARC and PRC while keeping other settings fixed, isolating the effect of reward-diverse exploration. As shown in Figure 6, Standard RCRL achieves the highest nominal performance for both ARC and PRC. At the same interaction budget, RCRL+Explore also slightly outperforms SIMBAV2. Thus, reward diversity can be beneficial even when it affects exploration, but using diverse reward supervision paired with task-focused exploration yields the strongest nominal performance.

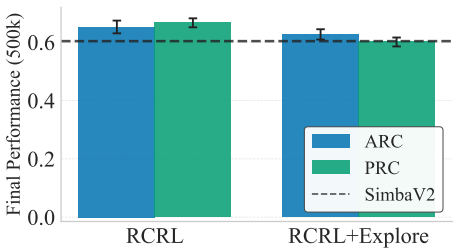


Figure 6: ***RCRL and exploration.*** RCRL performs best when exploration remains nominal. RCRL+Explore performs worse than RCRL but outperforms vanilla SIMBAV2.

Sensitivity to counterfactual supervision. We next investigate how the balance between nominal and counterfactual updates affects RCRL. The mixture coefficient α controls the probability of sampling the nominal reward for each update, allowing us to vary the amount of counterfactual supervision. We evaluate $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, with $\alpha = 0.5$ used by default, using SIMBAV2 backbone on the same 18 tasks, 500k steps and 10 seeds. As shown in Figure 14, RCRL improves performance across a broad range of sampling ratios, with the strongest results at $\alpha = 0.7$, corresponding to 70% nominal and 30% samples relabeled with counterfactual rewards. This suggests that counterfactual supervision remains beneficial even when it replaces a substantial fraction of nominal updates, although emphasizing the nominal objective yields the best performance. We further examine the choice and spread of the alternative reward sampling distribution in Appendix F.2.2, finding that RCRL remains effective across various distributions.

Model capacity. Finally, we study how RCRL interacts with critic capacity. Diverse reward supervision requires a critic to represent value functions for many specifications, suggesting that it may only be useful when the model has sufficient capacity. Figure 7 compares SIMBAV2 and SIMBAV2+RCRL across 23 tasks considered in the single-task benchmark (Appendix F.1), trained for 1M steps, using widths of 128 (0.3M parameters) and 512 (2.5M parameters), with the latter corresponding to the default. At smaller scale, RCRL provides no improvement over the baseline. In contrast, RCRL improves performance significantly when paired with a scaled model. Interestingly, increased capacity benefits both methods, but the gain is larger for RCRL. This suggests that scale enables the agent to exploit the richer supervision provided by counterfactual rewards. Thus, RCRL and model capacity appear to be complementary, with RCRL being the most effective once the learner has sufficient capacity to represent it.

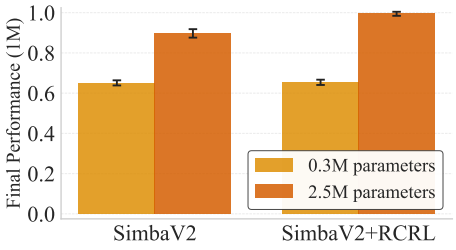


Figure 7: ***RCRL and model capacity.*** Diverse supervision improves performance when the critic has sufficient capacity to model counterfactual rewards.

Takeaway

RCRL achieves the strongest nominal task performance with task-focused exploration, although reward-diverse exploration can still outperform standard single-task RL. Explicit reward conditioning is essential, and sufficient model capacity helps exploit the benefits of diverse supervision.

4.3 RCRL and Reward Compatibility

Although RCRL improves nominal task learning, not all alternative rewards have to be equally useful. This raises the question of what makes a counterfactual reward beneficial for a given task. In this subsection, we investigate this question by measuring the contributions of various reward functions.

Leave-one-out reward analysis. We first measure the contribution of individual alternative rewards across the 8 HumanoidBench tasks listed in Table 1, using 1M steps and 10 seeds. For each nominal task, we compare SIMBAV2+RCRL trained with the full ARC reward set against variants in which one alternative reward is removed. We keep all other aspects of training fixed. Since `stair` and `slide` tasks share the same reward function, the 8 tasks correspond to seven distinct alternative reward specifications. We define the contribution of reward j to nominal task i as $C_{i,j} = \mathrm{AUC}_i - \mathrm{AUC}_{i\setminus j}$, where positive values indicate that including reward j improves nominal task learning and negative values indicate interference. As shown in the left panel of Figure 8, the mean contribution across task-alternative reward pairs is positive, corresponding to a 2% improvement, but individual effects vary substantially from -5% to 10%. Compatibility is task dependent, with `crawl` benefiting from every reward considered, and some rewards reducing performance on `stand` or `run`. Thus, the benefit of diverse reward supervision depends on the nominal task and the choice of counterfactual rewards.

Reward similarity. We next ask whether the most useful counterfactual rewards are simply those that most closely resemble the nominal reward. For all 48 task-alternative reward pairs, we measure the correlation $R_{i,j}$ between the nominal and counterfactual reward signals evaluated on the same transitions. We aggregate both reward correlation and leave-one-out contribution over the training. As shown in the middle panel of Figure 8, the opposite pattern emerges, with low correlation rewards making larger contributions, with a Pearson correlation of $r = -0.41$. This suggests that useful

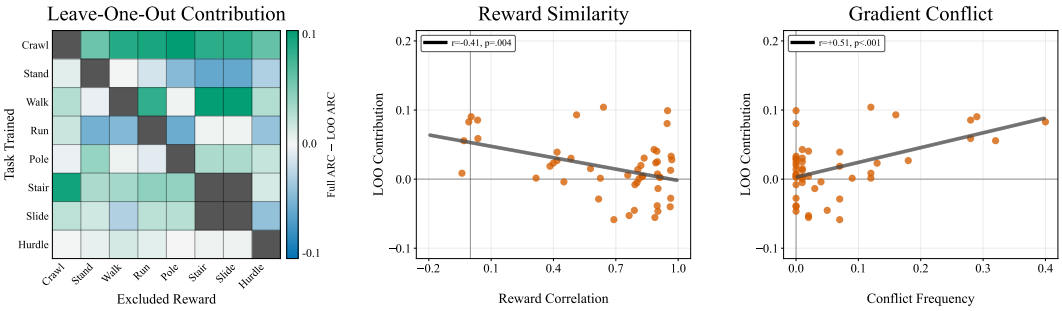


Figure 8: ***RCRL and reward compatibility.*** The left panel shows the leave-one-out contribution $C_{i,j}$ of each counterfactual reward to nominal task learning. The middle and right panels relate these contributions to reward correlation and gradient conflict frequency respectively. In both cases, less similar signals tend to make larger contributions in RCRL.

counterfactual rewards need not reproduce the nominal reward, and may improve learning the nominal objective by providing distinct supervision that is qualitatively different from the nominal reward.

Gradient compatibility. Counterfactual rewards may be useful precisely when they induce optimization signals that differ from the nominal objective. In particular, gradient conflict may indicate that an alternative reward provides complementary rather than redundant supervision. To examine this, for each of the 48 task-alternative reward pairs considered in the previous paragraph we sample the same transitions, compute critic gradients under the nominal and counterfactual rewards, and measure the frequency of gradient conflict with respect to the critic parameters calculated according to Yu et al. [77]. Specifically, we calculate the dot product between nominal and counterfactual reward minibatches gradients in the critic backbone, and we denote conflict when this dot product is negative. Similarly to the previous paragraph, we aggregate conflict frequency and leave-one-out contribution over training for the same task-alternative reward pairs. As shown in the right panel of Figure 8, alternative rewards that induce gradient conflict more frequently tend to make larger contributions to nominal task learning. This relationship is strong with a correlation of $r = 0.51$. Interestingly, this contrasts with the intuition that useful auxiliary objectives should produce gradients aligned with the nominal objective. Instead, our results suggest that diverse and even conflicting signals can provide complementary supervision and improvements.

RCRL improves gradient consistency. The previous analysis shows that beneficial counterfactual rewards often induce gradient conflicts with the nominal objective. However, this raises a question: *does training with potentially conflicting rewards lead to a critic whose gradients become more consistent over time?* To investigate this, we compare gradient conflict under single-task and RCRL training across the same eight tasks, considering both nominal–counterfactual gradients and gradients computed from independent nominal-reward minibatches (Figure 9). We find that nominal–counterfactual conflict decreases from 0.51 to 0.07 under RCRL. Interestingly, conflict between independent nominal-reward minibatches also decreases, from 0.09 to 0.04. These results suggest that learning from diverse reward specifications is associated with more consistent critic updates, even when evaluated exclusively on the nominal objective. One possible explanation is that counterfactual supervision encourages representations that accommodate multiple reward signals while improving gradient alignment within the nominal task. Although this analysis does not establish a causal mechanism, it provides evidence that the benefits of diverse reward supervision are accompanied by changes in the critic's optimization dynamics.

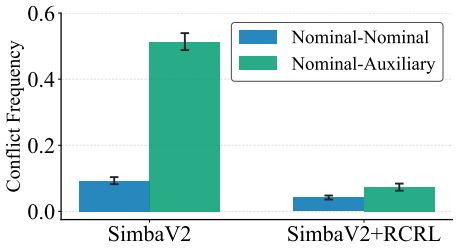


Figure 9: ***RCRL and gradient consistency.*** RCRL reduces nominal-counterfactual conflict, but also reduces nominal-nominal conflict between independently sampled batches.

Takeaway

Useful alternative rewards need not resemble the nominal objective. Less correlated rewards and more conflicting gradients are associated with larger nominal task improvements. Despite this conflicting supervision, RCRL ultimately reduces gradient conflict even in nominal task batches.

4.4 Steerability Beyond the Nominal Objective

The main goal of RCRL is to improve a fixed predefined objective, but reward conditioning paired with diverse reward supervision also makes the policy steerable. In this subsection, we investigate whether behaviors associated with Ψ can be recovered without further training.

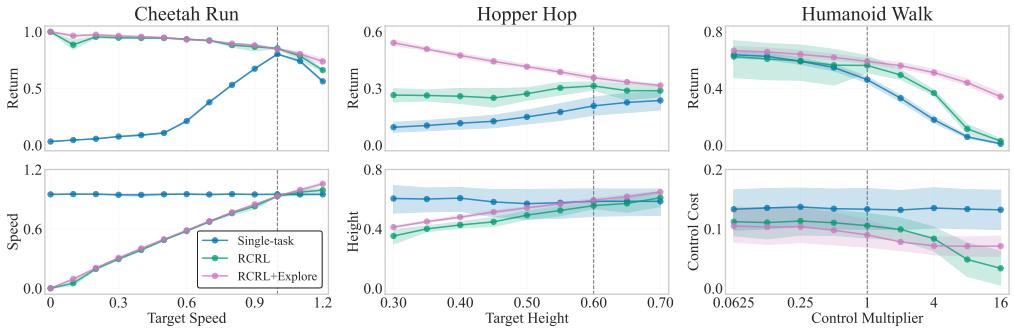


Figure 10: ***Zero-shot steerability.*** Alternative reward specifications induce different behaviors in `cheetah-run`, `hopper-hop`, and `humanoid-walk` tasks. We compare single-task SIMBAV2, RCRL, and RCRL+Explore, which trains and explores across multiple rewards, as well as uses $3\times$ more environment interactions. The dashed line denotes the nominal reward specification which is used to generate the experience for single-task and RCRL. RCRL produces steerable behavior, despite learning the other objectives entirely off-policy from nominal task experience.

Zero-shot steerability. We evaluate RCRL on 3 DM Control tasks: `cheetah-run`, with rewards targeting different running speeds; `hopper-hop`, with different standing heights; and `humanoid-walk`, with different action penalties. We compare SIMBAV2, SIMBAV2+RCRL, and SIMBAV2+RCRL+EXPLORE. Single-task and RCRL agents use 1M environment steps, whereas RCRL+Explore uses 3M steps, with 10 seeds per domain. RCRL interacts only under the nominal objective and learns alternative rewards through relabeling, while RCRL+Explore corresponds to a full multi-objective training with expanded exploration under the alternative objectives. At evaluation, we vary only the reward conditioning, without further policy updates. As shown in Figure 10, RCRL achieves the highest average nominal performance while providing greater zero-shot steerability than single-task training. On alternative objectives, average performance increases from 0.26 for single-task to 0.58 with RCRL, approaching the performance of full multi-objective training with expanded state-action coverage. Consistent with off-policy learning, the steerability decreases as the requested objective moves further from the nominal task. As such, RCRL can recover behaviors when they are sufficiently supported by the nominal objective, whereas multi-task training directly explores conditioned on alternative objectives.

Multi-objective and reward-free methods. We next compare RCRL with methods designed for learning across multiple objectives, including Preference Conditioned Control implemented as a variant of MORL with relabeling [73], USFA [12], and FB [67]. Since FB usually assumes access to an offline dataset, we initialize it with replay data collected during an RCRL training run. Using SIMBAV2 as the backbone, we evaluate all methods on all `walker`, `dog`, and `humanoid` tasks from DMC, leading to 10 tasks in total, under a budget of 500k environment interactions, reporting AUC on both nominal and alternative objectives from 5 random seeds. RCRL achieves the strongest nominal task performance, improving over single-task SIMBAV2 while substantially increasing alternative task performance. PCC with counterfactual reward relabeling achieves slightly better alternative performance, but lower nominal performance, illustrating the trade-off between broad objective coverage and task-focused exploration. USFA and FB perform substantially worse in this low-budget setting, although FB is typically trained with much larger reward-free datasets. Overall, these results show that reward-diverse exploration favors broader alternative task coverage, whereas nominally directed exploration better optimizes the predefined objective while retaining useful steerability.

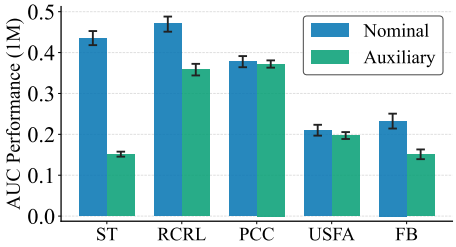


Figure 11: ***Multi-objective and reward-free methods.*** Despite exploring only under the nominal task, RCRL achieves the strongest nominal performance while allowing for steerability which is comparable to multi-objective PCC.

Transfer to alternative objectives. Finally, we investigate whether alternative reward supervision also improves transfer adaptation to these alternative tasks. While improving nominal task remains the primary goal of RCRL, exposure to alternative rewards may facilitate learning representations aligned with these rewards. Using the HumanoidBench tasks listed in Table 1, we train each agent for 1M steps on a nominal source task and then finetune it for 500k steps on each remaining target task, using 10 seeds per source-target pair. During source-task training, RCRL receives counterfactual supervision from the target rewards, whereas the baseline learns only from the nominal reward. We measure the transfer benefit as $T_{i,j} = \mathrm{AUC}_{i\to j}^{RCRL} - \mathrm{AUC}_{i\to j}^{Base}$. As shown in Figure 16, the individual effects vary across source-target pairs, ranging from -12% to 25% improvements. Interestingly, we observe substantial variation in transfer effectiveness across source-target task pairs, with some tasks serving as more effective transfer sources than others. In our setting, transfer difficulty is further influenced by differences in environment layout across tasks (e.g. empty scene in the walk task versus obstacles in pole task), since policies only receive information on the angular velocities and reward specification, and do not receive explicit information about environment variations. These results suggest that diverse reward supervision via RCRL can facilitate downstream adaptation, but understanding when and why these benefits emerge requires further investigation.

Takeaway

Diverse reward supervision via RCRL improves nominal task performance while enabling meaningful policy steerability. Exploring under alternative objectives can further improve auxiliary-task performance, but comes at the cost of nominal task efficiency.

5 Conclusions

We introduced RCRL, an off-policy method for training agents on multiple reward specifications while collecting experience under a nominal objective. The central idea is simple: when rewards are structured, replayed transitions can be reused to compute counterfactual rewards, providing additional supervision beyond the nominal reward. By conditioning the agent on the reward specification used in each update, RCRL allows this supervision to improve learning without collapsing distinct objectives into a single policy. Across various benchmarks, RCRL improves learning under the nominal reward and supports policy steerability at deployment. These results show that reward conditioning is useful not only for representing multiple objectives, but also as a practical mechanism for improving single-task learning. We discuss the limitations of our work in Appendix A

Acknowledgments

We gratefully acknowledge the Polish high-performance computing infrastructure, PLGrid (HPC Center: ACK Cyfronet AGH), for providing computational resources and support under grant no. PLG/2025/018597. We would also like to thank the Python [68], NumPy [31], Matplotlib [35], SciPy [69] and JAX [13] communities for developing tools that supported this work. Marek Cygan was partially supported by National Science Centre, Poland, under the grant 2024/54/E/ST6/00388. Pieter Abbeel holds concurrent appointments as a Professor at UC Berkeley and as an Amazon Scholar. This paper describes work performed at UC Berkeley and is not associated with Amazon.

References

[1] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International conference on machine learning*, pp. 11–20. PMLR, 2019.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

[4] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[5] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[6] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. In *ICLR 2021-Ninth International Conference on Learning Representations*, 2021.

[7] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594. PMLR, 2023.

[8] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[9] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pp. 501–510. PMLR, 2018.

[10] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 449–458. JMLR. org, 2017.

[11] Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. CrossQ: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *International conference on learning representations (ICLR)*, 2024.

[12] Diana Borsa, Andre Barreto, John Quan, Daniel J Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2019.

[13] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[15] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[16] Xinyue Chen, Che Wang, Zijian Zhou, and Keith W Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2020.

[17] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. *Advances in Neural Information Processing Systems*, 32, 2019.

[18] Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations*, 2022.

[19] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[20] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35:35603–35620, 2022.

[21] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taiga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. In *Forty-first International Conference on Machine Learning*, 2024.

[22] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Unsupervised zero-shot reinforcement learning via functional reward encodings. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 13927–13942, 2024.

[23] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

[24] Scott Fujimoto, Pierluca D'Oro, Amy Zhang, Yuandong Tian, and Michael Rabbat. Towards general-purpose model-free reinforcement learning. In *International Conference on Learning Representations*, 2025.

[25] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pp. 2170–2179. PMLR, 2019.

[26] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[27] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, 2017.

[28] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

[29] 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*, pp. 1861–1870. PMLR, 2018.

[30] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024.

[31] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[32] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning: Cf hayes et al. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

[33] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

[34] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. In *International Conference on Learning Representations*, 2021.

[35] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

[36] Tyler Ingebrand, Amy Zhang, and Ufuk Topcu. Zero-shot reinforcement learning via function encoders. In *International Conference on Machine Learning*, pp. 21007–21019. PMLR, 2024.

[37] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[38] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

[39] Chi Jin, Akshay Krishnamurthy, Max Simchowitz, and Tiancheng Yu. Reward-free exploration for reinforcement learning. In *International conference on machine learning*, pp. 4870–4879. PMLR, 2020.

[40] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pp. 1094–8, 1993.

[41] Junsu Kim, Seohong Park, and Sergey Levine. Unsupervised-to-online reinforcement learning. *URL https://arxiv. org/abs/2408.14785*, 2024.

[42] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[43] Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=4-k7kUavAj.

[44] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33:19884–19895, 2020.

[45] Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. In *International Conference on Machine Learning*, pp. 33352–33403. PMLR, 2025.

[46] Ilya Loshchilov, Cheng-Ping Hsieh, Simeng Sun, and Boris Ginsburg. ngpt: Normalized transformer with representation learning on the hypersphere. *arXiv preprint arXiv:2410.01131*, 2024.

[47] Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pp. 113038–113071, 2024.

[48] Michal Nauman, Marek Cygan, Carmelo Sferrazza, Aviral Kumar, and Pieter Abbeel. Bigger, regularized, categorical: High-capacity value functions are efficient multi-task learners. In *Proceedings of the 39th International Conference on Neural Information Processing Systems*, 2025.

[49] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pp. 16828–16847. PMLR, 2022.

[50] Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 23283–23295, 2021.

[51] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[52] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[53] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.

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

[55] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48: 67–113, 2013.

[56] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

[57] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[58] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2020.

[59] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pp. 8583–8592. PMLR, 2020.

[60] Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024.

[61] Harshit Sikchi, Andrea Tirinzoni, Ahmed Touati, Yingchen Xu, Anssi Kanervisto, Scott Niekum, Amy Zhang, Alessandro Lazaric, and Matteo Pirotta. Fast adaptation with behavioral foundation models. *arXiv preprint arXiv:2504.07896*, 2025.

[62] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[63] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[64] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[65] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

[66] Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems*, 34:13–23, 2021.

[67] Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? *arXiv preprint arXiv:2209.14935*, 2022.

[68] Guido Van Rossum, Fred L Drake, et al. *Python reference manual*, volume 111. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[69] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[70] Claas A Voelcker, Marcel Hussing, Eric Eaton, Amir-massoud Farahmand, and Igor Gilitschenski. Mad-td: Model-augmented data stabilizes high update ratio rl. *arXiv preprint arXiv:2410.08896*, 2024.

[71] Andrew J Wagenmaker, Yifang Chen, Max Simchowitz, Simon Du, and Kevin Jamieson. Reward-free rl is no harder than reward-aware rl in linear markov decision processes. In *International Conference on Machine Learning*, pp. 22430–22456. PMLR, 2022.

[72] Ruosong Wang, Simon S Du, Lin Yang, and Russ R Salakhutdinov. On reward-free reinforcement learning with linear function approximation. *Advances in neural information processing systems*, 33:17816–17826, 2020.

[73] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in neural information processing systems*, 32, 2019.

[74] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International conference on learning representations*, 2020.

[75] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[76] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

[77] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[78] Chongyi Zheng, Seohong Park, Sergey Levine, and Benjamin Eysenbach. Intention-conditioned flow occupancy models. In *International Conference on Learning Representations*, volume 2026, pp. 116781–116817, 2026.

[79] Zhiyuan Zhou, Andy Peng, Qiyang Li, Sergey Levine, and Aviral Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. *arXiv preprint arXiv:2412.07762*, 2024.

[80] Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021.

Impact Statement

This work contributes to improving the robustness, adaptability, and efficiency of reinforcement learning systems by enabling agents to learn and adapt to multiple reward specifications without additional environment interaction. We do not foresee any direct societal impacts specific to this work beyond those generally associated with advances in reinforcement learning research.

A Limitations

RCRL requires alternative rewards to be evaluable on replayed transitions and is limited by the behavioral coverage of nominal task data. We focus on structured rewards with explicit components, but the same idea could extend to learned reward families. Objectives requiring states or actions rarely visited by the nominal policy may be difficult to learn without additional exploration. As shown in our experiments, the choice of alternative rewards also matters, with some providing useful supervision, and others interfering with learning. Although RCRL is robust across several perturbation distributions and scales, selecting informative and compatible reward specifications remains an open problem. Finally, richer reward supervision may require sufficient model capacity. Our ablations show larger models benefit more from RCRL. In our experiments, the added computational cost is modest: updates are approximately 10% slower on fixed A100 hardware, while replay memory increases by only $O(k)$ float32 values per transition for k reward components.

B Background

The goal of RL is to learn a policy that maximizes expected return in a MDP [54] $(\mathcal{S}, \mathcal{A}, P, r, \rho, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(s'|s,a)$ the transition dynamics, ρ the initial state distribution, $r(s,a)$ the scalar reward, and $\gamma \in (0,1)$ the discount factor. A policy $\pi(a|s)$ is optimized to maximize expected discounted return. The value $V^\pi(s)$ and Q-value functions $Q^\pi(s,a)$ denote the expected return from a state and state-action pair. In many control tasks, desired behavior is specified through composite reward functions [63, 52], which combine multiple components such as task progress, control cost or smoothness. In such case, the scalar reward $r(s,a)$ is defined as:

$$r_\psi(s,a) = f\big(\psi, c_1(s,a), ..., c_k(s,a)\big), \quad \psi \in \Psi.$$

Above, $c_1, ..., c_k$ denote the k reward components, ψ denotes the specification , and Ψ denotes the considered specifications given a function class f. Single-task RL typically optimizes a fixed specification $\psi^\star$, whereas multi-objective and multi-task learning can be viewed as considering a set of reward specifications corresponding to different preferences, objectives, or tasks [55].

C Expanded Related Work

Auxiliary objectives. Auxiliary objectives have been used to improve representation learning and sample efficiency in RL. UNREAL [37] augments the main RL objective with auxiliary tasks such as pixel control and reward prediction, while methods such as DeepMDP [25] and SPR [58] use objectives over latent dynamics to improve data efficiency. RCRL similarly uses additional supervision to improve a primary task, but the additional objectives are alternative reward specifications on which the agent is conditioned. In consequence, diverse reward supervision can improve nominal task learning while also producing reward-specific behaviors that remain accessible at deployment.

Return and Command-Conditioned Policies. A related line of work conditions policies on desired outcomes, such as target returns, horizons, or behavioral commands. UDRL [57] trains policies to predict actions conditioned on commands specifying desired return and time horizon, while reward-conditioned policies and Decision Transformer condition behavior on desired return-to-go [42, 15]. These methods enable behavior to be steered under a fixed reward function by changing the desired outcome provided to the policy. RCRL differs in that the conditioning variable is not a target return or command, but the reward specification itself. Thus, changing the conditioning in RCRL changes how rewards are computed, rather than specifying a desired level of return under a fixed reward.

Reward-free, unsupervised, and meta-RL. Reward-free RL aims to collect experience that supports adaptation to objectives revealed after exploration [39, 72, 71]. Related approaches learn

broadly reusable behaviors or representations through unsupervised pretraining [22, 36, 41], model-based exploration [59], or pretrained behavioral representations [61]. Meta-RL methods, such as VariBAD [80], instead adapt to latent tasks by inferring task information from interaction history. RCRL targets a different setting, focusing on the nominal objective and a structured family of alternative rewards known during training, and the reward specification is explicitly provided rather than inferred. Instead of optimizing for broad downstream adaptation, RCRL uses counterfactual reward supervision to improve online learning of the nominal task, with within-family steerability emerging as a secondary capability resulting from off-policy learning of diverse reward functions.

Domain randomization and data augmentation. Domain randomization and observation augmentation improve robustness by exposing agents to nuisance variation in dynamics or observations [65, 53, 44, 74]. These methods typically encourage invariance to the randomized factors. RCRL can instead be viewed as reward augmentation: the same experience is relabeled under different reward specifications, and the agent is explicitly conditioned on that variation. Rather than learning invariance, RCRL preserves reward-specific behavior while using the additional supervision to improve the nominal objective.

D Additional Considerations on Training Stability

Training across multiple reward specifications can make critic optimization harder when rewards differ substantially in scale or difficulty. With a standard MSE critic, larger reward scales induce larger TD errors and gradients, which can dominate learning of the shared value function [33, 48]. In our experiments, most reward specifications remain within a controlled range around the nominal reward, and we do not observe instability beyond that of the underlying backbone. Modern RL algorithms already include mechanisms that reduce this sensitivity. In particular, reward normalization and categorical value learning have been shown to stabilize learning across heterogeneous rewards [6, 10, 21, 48]. Backbones such as SIMBAV2 and BRC therefore require no additional stabilization when combined with RCRL. As discussed in Section 4.2, model capacity provides a complementary consideration. RCRL asks a shared critic to represent value functions for multiple reward specifications, and our ablations show substantially larger gains with larger networks. This suggests that sufficient critic capacity is important for exploiting the additional supervision provided by RCRL, consistent with related findings in multi-task RL [48].

E Implementation Details

In this section, we describe how RCRL is integrated into each baseline method, and detail the two proposed RCRL implementation procedures (PRC and ARC).

E.1 Algorithms

We briefly summarize the algorithmic integrations considered in this work. Unless otherwise stated, all architectural choices, optimization settings, and training procedures follow the corresponding baseline implementations, with modifications limited to the incorporation of reward conditioning.

SimbaV2+RCRL and BRC+RCRL. The integration of RCRL into SIMBAV2 and BRC follows the same high-level procedure, as both backbones use reward normalization and categorical value learning. Both algorithms normalize rewards and learn categorical value distributions over bounded return ranges $[V_{\min}, V_{\max}]$, which requires care when training with multiple rewards. In principle, one could design reward specifications whose resulting rewards are already normalized. Instead, we use a simpler strategy that avoids additional reward engineering: we maintain separate running normalization statistics for each reward specification used during training. Since normalization statistics are tracked per specification, we instantiate PRC using a finite set of reward specification. Specifically, at the beginning of training, we sample 1024 perturbation vectors using the log-uniform procedure described in Section 4, and keep this set fixed throughout training. ARC is finite by construction, since it uses the reward functions associated with the available auxiliary tasks. Because both PRC and ARC therefore operate over finite specification sets in these experiments, we use the embedding-based conditioning strategy described in Section 3. Each reward specification is assigned a learnable embedding, which is trained jointly with the critic through the critic objective and used to

condition both the critic and policy. For BRC, we reuse its existing task-conditioning machinery, treating the active reward specification as the conditioning identity.

DrQV2+RCRL. In contrast to SIMBAV2 and BRC, DRQV2 uses a standard MSE critic loss and does not employ reward normalization, which makes the integration of RCRL simpler. We therefore apply reward-conditioned training directly with the MSE critic loss, without adding normalization or distributional value learning. This allows us to evaluate whether RCRL remains stable without the stabilization mechanisms used by the other base algorithms. In this setting, reward specifications are sampled from a continuous space, and the corresponding perturbation vectors are used directly for conditioning by concatenating them to the agent input, without a separate embedding module.

E.2 Conditioning

In this subsection, we describe the implementation of the two strategies used to generate the set of alternative reward specifications in RCRL. We focus on how the set Ψ is instantiated in practice, how reward specifications are sampled during training, and how conditioning information is incorporated.

Perturbed Reward Conditioning (PRC). For PRC, the set Ψ is defined implicitly as a continuous family of reward specifications obtained by multiplicatively perturbing the nominal reward coefficients. For SIMBAV2 and BRC, we first sample a finite collection of 1024 perturbation vectors from the continuous PRC distribution. During training, alternative specifications are sampled independently for each transition from this fixed collection. For DRQV2, perturbations are instead sampled directly from the continuous distribution during training. We present an implementation of PRC in Figure 12.

```python
import numpy as np
import gymnasium as gym

def sample_perturbations(number_of_reward_components, number_of_external_rewards=1024, spread=4.0):
    lower = np.random.uniform(1.0 / spread, 1.0, size=(number_of_external_rewards // 2, number_of_reward_components))
    upper = np.random.uniform(1.0, spread, size=(number_of_external_rewards // 2, number_of_reward_components))
    perturbations = np.vstack([lower, upper])
    np.random.shuffle(perturbations)
    return perturbations


class MujocoWrapper(gym.Wrapper):

    def __init__(self, env, seed, env_name):
        super().__init__(env)
        self.env.reset(seed=seed)
        self.env_name = env_name
        np.random.seed(seed)
        number_of_reward_components = 2 if env_name == 'HalfCheetah-v4' else 3
        self.perturbations = sample_perturbations(number_of_reward_components)

    def step(self, action):
        state, reward, terminated, truncated, infos = self.env.step(action)
        if self.env_name == 'Humanoid-v4':
            reward_healthy = infos['reward_alive']
            reward_forward = infos['forward_reward']
            reward_control = infos['reward_quadctrl']
            extra_rewards = [reward_healthy, reward_forward, reward_control]
        if self.env_name == 'Ant-v4':
            reward_healthy = infos['reward_survive']
            reward_forward = infos['reward_forward']
            reward_control = infos['reward_ctrl']
            extra_rewards = [reward_healthy, reward_forward, reward_control]
        if self.env_name == 'HalfCheetah-v4':
            reward_forward = infos['reward_run']
            reward_control = infos['reward_ctrl']
            extra_rewards = [reward_forward, reward_control]
        if self.env_name == 'Hopper-v4' or self.env_name == 'Walker2d-v4':
            reward_healthy, reward_forward, reward_control = self._get_hopper_walker_rewards(action, infos)
            extra_rewards = [reward_healthy, reward_forward, reward_control]
        extra_rewards = np.stack(extra_rewards)
        extra_rewards = self.weights @ extra_rewards
        infos['extra_rewards'] = extra_rewards
        return state, reward, terminated, truncated, infos

    def _get_hopper_walker_rewards(self, action, infos):
        env = self.env
        while hasattr(env, "env"):
            env = env.env
        reward_forward = env._forward_reward_weight * infos['x_velocity']
        reward_healthy = env.healthy_reward
        reward_control = -env.control_cost(action)
        return reward_healthy, reward_forward, reward_control
```

Figure 12: ***Code example.*** We present the code used to sample perturbations vector Δ, as well as an OpenAI Gym wrapper used to calculate alternative reward functions in our single-task experiments.

In practice, the benchmarks used in our experiments (i.e. DM Control, OpenAI Gym, and HumanoidBench) employ reward functions that are either linear combinations of reward components or multiplicative (or exponentiated) compositions. For linear reward functions of the form $r_\psi(s,a) = \sum_{i=1}^{k} \psi_i \, c_i(s,a)$ the perturbations are applied directly to the linear coefficients ψ_i. For multiplicative reward functions of the form $r_\psi(s,a) = \prod_{i=1}^{k} c_i(s,a)^{\psi_i}$ the perturbations are applied to the corresponding exponents. This unified specification allows the same perturbation mechanism to be applied consistently across different reward structures.

Auxiliary Reward Conditioning (ARC). For ARC, the set Ψ is instantiated explicitly as a finite collection of task-specific reward functions defined within each benchmark. Specifically, auxiliary task rewards are constructed by evaluating the reward functions of other tasks that share the same agent embodiment on the collected state-action trajectories. For example, in the DeepMind Control Suite dog benchmarks, tasks such as `dog-run`, `dog-walk`, and `dog-trot` share an identical reward structure that differs only in the target velocity of the dog. We therefore implement auxiliary reward conditioning by evaluating all reward functions on every transition collected under the nominal task, yielding multiple reward signals corresponding to different target speeds. We show our implementation of this procedure in Figure 13.

```python
import numpy as np
import gymnasium as gym
import dm_control.utils.rewards as utils

class DogWrapper(gym.Wrapper):
    def __init__(self, env, seed=0):
        super().__init__(env)
        self.move_speeds = [0, 1, 4, 9]
        self.min_upright = np.cos(np.deg2rad(30))
        self.env.reset(seed=seed)
        self._init_consts()

    def reset(self, **kw):
        obs, info = self.env.reset(**kw)
        self._init_consts()
        return obs, info

    def step(self, action):
        obs, r, term, trunc, info = self.env.step(action)
        torso_pelvis_height, upright, touch_sensors, com_forward_velocity = self._sensors()
        info["extra_rewards"] = self._stand_reward(torso_pelvis_height, upright, touch_sensors) * self._move_reward(com_forward_velocity)
        return obs, r, term, trunc, info

    def _init_consts(self):
        physics = self.env.unwrapped._env.physics
        self.stand_h = physics.torso_pelvis_height() * 0.9
        self.weight = physics.named.model.body_subtreemass["torso"] * -physics.model.opt.gravity[2]

    def _sensors(self):
        physics = self.env.unwrapped._env.physics
        return physics.torso_pelvis_height(), physics.upright(), physics.touch_sensors().sum(), physics.com_forward_velocity()

    def _stand_reward(self, torso_pelvis_height, upright, touch_sensors):
        torso  = utils.tolerance(torso_pelvis_height[0], (self.stand_h[0], np.inf), margin=self.stand_h[0])
        pelvis = utils.tolerance(torso_pelvis_height[1], (self.stand_h[1], np.inf), margin=self.stand_h[1])
        upright = utils.tolerance(upright, (self.min_upright, np.inf), margin=self.min_upright + 1, sigmoid="linear", value_at_margin=0)
        touch = utils.tolerance(touch_sensors, (self.weight, np.inf), margin=self.weight, sigmoid="linear", value_at_margin=0.9)
        return torso * pelvis * upright * touch

    def _move_reward(self, v):
        r = []
        for s in self.move_speeds:
            if s < 1:
                r.append(1.0)
            else:
                r.append(utils.tolerance(v, (s, 2 * s), margin=max(1.0, s), sigmoid="linear", value_at_margin=0))
        return (4 * np.stack(r) + 1) / 5
```

Figure 13: ***Code example.*** We present the code used to calculate auxiliary task rewards for the DeepMind Control `dog` tasks. We use this wrapper in our multi-task experiments.

In contrast, in the HumanoidBench benchmark, reward functions are often environment-specific rather than task-specific. For instance, tasks such as `stair` and `slide` use identical reward definitions but differ only in the environment layout (e.g. a staircase versus a sliding surface). Moreover, in these tasks the state representation consists solely of proprioceptive features (e.g. joint positions and velocities) and does not encode information about external objects in the environment. As a result, evaluating certain task reward functions on trajectories collected in other environments can yield identical reward signals. To avoid redundancy, we construct Ψ by retaining only reward functions that produce distinct reward outputs on the collected data, discarding any auxiliary reward functions that are equivalent to others. Consequently, the number of distinct auxiliary rewards may be smaller than the number of tasks in the benchmark. We list the exact rewards used in each benchmark in Table 1.

Table 1: ***ARC rewards.*** Tasks in parentheses share identical reward functions and are treated as a single reward specification.

DMC Dogs	HumanoidBench	HumanoidBench Hard
`stand`	`crawl`	`crawl`
`walk`	`stand`	`stand`
`trot`	`walk`	`walk`
`run`	`run`	`run`
	(`stair`, `slide`)	(`stair`, `slide`)
	`pole`	`pole`
	`hurdle`	`hurdle`
		(`balance_simple`, `balance_hard`)
		(`sit_simple`, `sit_hard`)

F Experimental Details

Here, we discuss the details relevant to reproduction of our experiments. All experiments were run using NVIDIA A100 GPU with 40GB of RAM and 8 CPU cores of AMD EPYC 7742.

F.1 Benchmarks

In the single-task setting, where RCRL augments SIMBAV2, we adopt a subset tasks used in Lee et al. [45]. In the multi-task setting, we use the benchmarks proposed in Nauman et al. [48].

DMC Dogs
`dog-stand, dog-walk, dog-trot, dog-run`

OpenAI Gym
`HalfCheetah-v4, Humanoid-v4, Hopper-v4, Walker2d-v4, Ant-v4`

HumanoidBench
`h1-walk, h1-stand, h1-run, h1-stair, h1-crawl, h1-hurdle, h1-slide, h1-pole, h1-maze, h1-reach, h1-sit_simple, h1-sit_hard, h1-balance_simple, h1-balance_hard`

HumanoidBench MT
`h1-walk, h1-stand, h1-run, h1-stair, h1-crawl, h1-pole, h1-slide, h1-hurdle, h1-maze`

HumanoidBench Hard MT
`h1hands-walk, h1hands-stand, h1hands-run, h1hands-stair, h1hands-crawl, h1hands-pole, h1hands-slide, h1hands-hurdle, h1hands-maze, h1hands-sit_simple, h1hands-sit_hard, h1hands-balance_simple, h1hands-balance_hard, h1hands-reach, h1hands-spoon, h1hands-window, h1hands-insert_small, h1hands-insert_normal, h1hands-bookshelf_simple, h1hands-bookshelf_hard`

DMC Vision
`acrobot-swingup, cartpole-swingup_sparse, cheetah-run, hopper-hop, reacher-easy, reacher-hard, quadruped-run, quadruped-walk, walker-run`

Table 2: ***Benchmark summary.*** Checkmarks indicate the environment suites included in each benchmark.

Environment suite	Tasks	SimbaV2	BRC	DrQV2
DMC Dogs	4	✓	✓	–
OpenAI Gym	5	✓	–	–
HumanoidBench	14	✓	–	–
HumanoidBench MT	9	–	✓	–
HumanoidBench Hard MT	20	–	✓	–
DMC Vision	9	–	–	✓
Total tasks	–	23	33	9

F.2 Additional Details on Experiments

Here, we provide additional details on the experiments presented in this paper. All figures show 95% bootstrapped confidence intervals calculated using the methodology presented in Agarwal et al. [3].

F.2.1 RCRL Improves Learning Efficiency

First, we describe the experiments presented in Section 4.1. All experiments in this section follow the experimental settings, benchmarks, and evaluation protocols established in prior work.

Single-task RL. We leverage the experimental setup proposed in SIMBAV2 manuscript [45]. We use a subset of 3 out of 4 benchmarks: 14 tasks from HumanoidBench, 5 tasks from OpenAi Gym, and 4 tasks from DeepMind Control. We list the specific tasks in Appendix F.1. We implement RCRL via perturbed reward augmentation as described in Section 3 and Appendix E. We run 10 seeds for SIMBAV2 and SIMBAV2+RCRL, basing on the official implementation provided in `https://github.com/DAVIAN-Robotics/SimbaV2`.

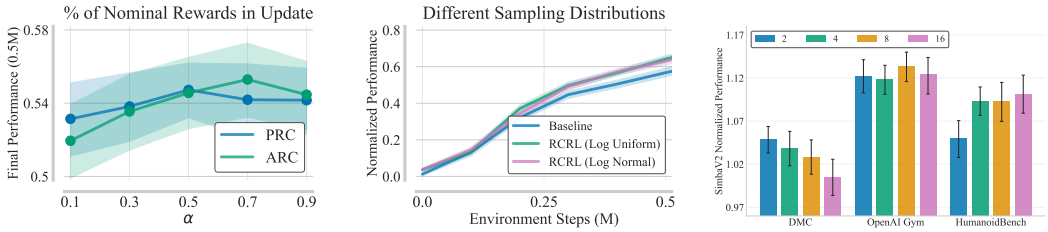


Figure 14: ***RCRL sensitivity to counterfactual supervision.*** All experiments use 500k steps and 10 seeds. (**Left**). Sensitivity to the conditioning probability α, which controls the fraction of updates performed under the nominal reward. (**Middle**) Comparison of perturbation sampling distributions for PRC on 14 HumanoidBench tasks. We compare the default log-uniform to a log-normal centered around the nominal specification. (**Right**) using the same 14 tasks, we vary the spread of the log-uniform distribution used to sample perturbations spread $\in \{2, 4, 8, 16\}$ while keeping all other hyperparameters fixed.

Multi-task RL. For multi-task experiments, we adopt the experimental setup from BRC [48]. We evaluate on three benchmarks: DMC Dogs (4 tasks), HumanoidBench (9 tasks), and HumanoidBench Hard (20 tasks). We list all tasks in Appendix F.1. In this setting, we implement auxiliary reward conditioning as described in Section 3 and Appendix E, conditioning each task on the reward functions of all other non-redundant rewards sharing the same embodiment. We run 5 seeds per benchmark and report results following the original BRC evaluation protocol and build on the official implementation provided in `https://github.com/naumix/BiggerRegularizedCategorical`.

Vision-based RL. For vision-based single-task experiments, we follow the experimental setup introduced in DrQV2 [75]. We evaluate on the DMC-medium benchmark and exclude two tasks that use a single reward components (`finger-turn_easy` and `finger-turn_hard`), as they are incompatible with our reward specification strategy. We run 5 random seeds for each method and list the evaluated tasks in Appendix F.1. RCRL is implemented using perturbed reward conditioning as described in Section 3 and Appendix E, with conditioning performed via a standardized perturbation vector concatenated to the state. We build on the implementation provided in `https://github.com/sukhijab/maxinforl_jax`.

F.2.2 What Makes RCRL Effective?

Here, we provide additional details on the ablation experiments presented in Section 4.2. All experiments use 10 random seeds unless directly stated differently.

Table 3: ***Effect of batch size.*** Comparison of SimbaV2 and SimbaV2+RCRL on 14 HumanoidBench tasks after 500k environment steps, averaged over 10 seeds. Gain denotes the relative improvement of RCRL over SimbaV2.

Total batch size	SimbaV2	SimbaV2+RCRL	RCRL Gain
256	0.33 ± 0.003	0.36 ± 0.006	$+9.1\%$
512	0.32 ± 0.006	0.37 ± 0.004	$+15.6\%$

Reward supervision and coverage. We use the BRC backbone on the four DMC Dogs tasks, training for 1M environment steps per task with 5 seeds. We compare four configurations: Vanilla ST, ST+RCRL, ST+Coverage, and Full MT. For ST+Coverage, we progressively introduce replay transitions collected by single-task agents on the remaining three tasks, following the tandem RL setup [50]. This provides $4\times$ more transitions while retaining the nominal reward objective.

Reward representation. We evaluate four conditioning strategies using the ARC variant of SimbaV2+RCRL: no conditioning, where the agent learns a single policy for all reward functions; fixed preference vectors using one-hot encoding; learned reward embeddings following BRC [48]; and separate reward-specific heads. We evaluate all variants on 18 tasks from HumanoidBench and DMC, training for 1M environment steps with the same reward specifications and training protocol.

Reward supervision and exploration. We compare SimbaV2, RCRL, and RCRL+Explore using both PRC and ARC on the same 18 HumanoidBench and DMC tasks, training each configuration for 500k environment steps. RCRL+Explore samples reward specifications for environment interaction

from the same distribution used during training, whereas standard RCRL always interacts under the nominal specification. All other settings are kept fixed.

Sensitivity to counterfactual supervision. We evaluate the nominal reward sampling probability α using SIMBAV2+RCRL on the same 18 HumanoidBench and DMC tasks, training for 500k steps. We consider $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, varying the proportion of nominal and counterfactual updates while keeping the reward sampling distribution fixed. The results are presented in Figure 14.

Model capacity. We compare SIMBAV2 and SIMBAV2+RCRL on the original 23-task benchmark spanning HumanoidBench (14 tasks), OpenAI Gym (5 tasks), and DMC (4 tasks), training for 1M environment steps. We use network widths of 128 (0.3M parameters) and 512 (2.5M parameters), with the latter corresponding to the default SIMBAV2 architecture.

F.2.3 RCRL and Reward Compatibility

All experiments in Section 4.3 use the ARC variant of SIMBAV2+RCRL on the 8 HumanoidBench tasks listed in Table 1, training for 1M environment steps with 10 random seeds. Since `stair` and `slide` share the same reward function, we consider 7 distinct reward specifications, yielding 48 nominal-counterfactual reward pairs. For the leave-one-out analysis, we compare RCRL trained with the full reward set against variants with one counterfactual reward removed, keeping the training setup and nominal sampling probability fixed. To study reward similarity, we compute Pearson correlations between nominal and counterfactual reward signals evaluated on the same transitions. For gradient compatibility, we compute critic gradients under both rewards on shared minibatches and measure the frequency of conflicting updates following Yu et al. [77]. Finally, to study gradient consistency, we compare conflict frequencies under vanilla and RCRL training, calculating the average gradient conflict over all 20 checkpoints and 10 random seeds, considering both nominal-counterfactual gradients and nominal gradients computed from independently sampled minibatches.

F.2.4 Steerability Beyond the Nominal Objective

Finally, we describe experiments presented in Section 4.4.

Zero-shot steerability. We evaluate zero-shot adaptation on three continuous control tasks from the DeepMind Control Suite: `cheetah-run`, `hopper-hop`, and `humanoid-walk`. For each task, we run 10 random seeds. All experiments use the same hyperparameter configurations as in the nominal reward experiments, and we evaluate SIMBAV2+RCRL, vanilla SIMBAV2, and RCRL+Explore (a variant where counterfactual rewards are also used to guide exploration; in practice this implies an algorithm that works similarly to MTRL/MORL, but with reward relabeling). For each task, we define a set of transformed reward functions that promote specific behavioral variations.

```python
import numpy as np
import dm_control.utils.rewards as utils

def _get_extra_rewards_humanoid():
    action_penalty_weights = np.array([0.0625, 0.125, 0.25, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0])
    weights = np.concatenate((action_penalty_weights[:, None], np.ones((action_penalty_weights.shape[0], 3))), 1)
    physics = self.env.unwrapped._env.physics
    standing = utils.tolerance(physics.head_height(), bounds=(1.4, float('inf')), margin=0.35)
    upright = utils.tolerance(physics.torso_upright(), bounds=(0.9, float('inf')), sigmoid='linear', margin=1.9, value_at_margin=0)
    small_control = utils.tolerance(physics.control(), margin=1, value_at_margin=0, sigmoid='quadratic').mean()
    small_control = 0.1 + 0.9 * small_control
    speed = np.linalg.norm(np.copy(physics.center_of_mass_velocity())[[0, 1]])
    move = utils.tolerance(np.linalg.norm(speed, bounds=(self._move_speed, float('inf')), margin=self._move_speed, value_at_margin=0, sigmoid='linear')
    move = (5*move + 1) / 6
    reward_components = np.hstack((small_control, standing, upright, move))
    log_reward_components = np.log(reward_components + 1e-8)
    log_extra_rewards = weights @ log_reward_components
    extra_rewards = np.exp(log_extra_rewards)
    return extra_rewards

def _get_extra_rewards_cheetah():
    physics = self.env.unwrapped._env.physics
    extra_rewards = []
    for target_speed in range(13):
        reward_augment = utils.tolerance(physics.speed(), bounds=(target_speed, target_speed), margin=5.0, value_at_margin=0, sigmoid='linear')
        extra_rewards.append(reward_augment)
    extra_rewards = np.stack(extra_rewards)
    return extra_rewards


def _get_extra_rewards_hopper():
    physics = self.env.unwrapped._env.physics
    standing = utils.tolerance(physics.height(), (self._STAND_HEIGHT, 2))
    other = utils.tolerance(physics.speed(), bounds=(2.0, float('inf')), margin=1.0, value_at_margin=0.5, sigmoid='linear')
    extra_rewards = []
    for target_height in [0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7]:
        standing_augment = utils.tolerance(physics.height(), bounds=(target_height, target_height), margin=0.3, value_at_margin=0, sigmoid='linear')
        extra_rewards.append(standing_augment*other)
    extra_rewards = np.stack(extra_rewards)
    return extra_rewards
```

Figure 15: ***Code example.*** We present the code used to calculate auxiliary rewards in the zero-shot steerability experiments.

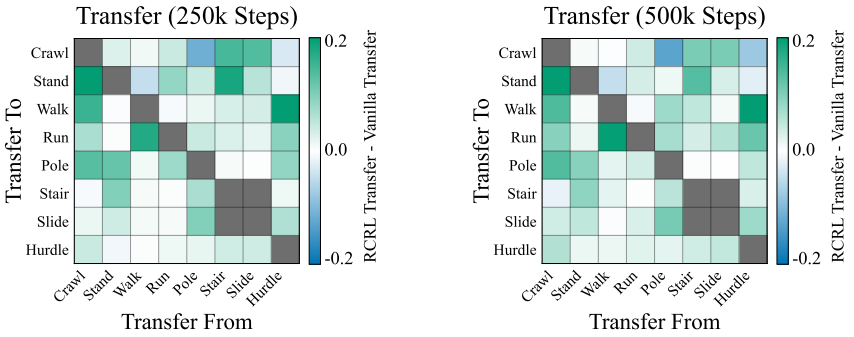


Figure 16: ***Transfer to alternative objectives.*** We compare the transfer performance of SIMBAV2+RCRL against vanilla SIMBAV2 across 54 source-target pairs from HumanoidBench. The heatmaps show differences in finetuning AUC after 250k and 500k target-task steps, with positive values indicating improvements from RCRL. RCRL improves transfer on 74% of pairs, with mean AUC gains of 5%. Results are averaged over 10 random seeds per pair.

For the standard RCRL agent, these transformed rewards are used only for reward conditioning and off-policy updates; the agent does not collect additional experience or explore under these alternative rewards, and all environment interaction is performed exclusively under the nominal reward. After 1M environment steps of training for SIMBAV2 and SIMBAV2+RCRL, and 3M environment steps for RCRL+Explore, we evaluate all agents under each of the considered reward functions, reporting the mean return over 10 rollout episodes per reward specification. For RCRL+Explore, we resample the objective used for exploration after every episode. Zero-shot evaluation is performed by changing only the reward conditioning provided to the policy at test time, without updating any network parameters. Specifically, for `cheetah-run`, we construct alternative rewards that are maximized when the agent runs at specific target speeds, with desired speeds given by $[0.0, 1.0, 2.0, \ldots, 12.0]$, with maximal recorded speed in single-task training of around 11. The nominal reward in this task encourages high forward velocity without specifying a target speed. For `hopper-hop`, auxiliary rewards are defined to favor different target standing heights, with desired heights $[0.30, 0.35, \ldots, 0.70]$. The nominal reward softly enforces a minimum height above 0.6 but does not prefer any particular height beyond this threshold. Finally, for `humanoid-walk`, we vary the relative importance of the control cost term by scaling its coefficient with multipliers $[0.0625, 0.125, 0.25, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0]$.

Multi-objective and reward-free methods. We compare single-task SIMBAV2, RCRL, PCC [73], USFA [12], and FB [67] on all 10 DMC `walker`, `dog`, and `humanoid` tasks. We use a budget of 500k environment interactions and 5 random seeds, keeping training settings consistent where possible. We report AUC under both nominal and alternative reward specifications. Since FB is typically trained with substantially more offline data, we evaluate it initialized with replay collected during an RCRL training run from the corresponding seed.

Table 4: ***Comparison with multi-objective and reward-free methods.*** We report performance aggregated across Walker, Dog, and Humanoid tasks. Each entry shows the AUC on the nominal / alternative objectives.

	Single-task	RCRL	PCC	USFA	FB Online	FB
Walker	790 / 240	**828** / 633	807 / **797**	635 / 588	81 / 84	642 / 392
Dog	420 / 162	**441** / **324**	238 / 230	48 / 49	18 / 18	95 / 80
Humanoid	99 / 48	**154** / 130	138 / **136**	3 / 4	2 / 2	6 / 5

Transfer to alternative objectives. We evaluate performance under alternative reward specifications through finetuning experiments on a subset of 8 tasks from the HumanoidBench benchmark that share identical state and action space structures. The tasks are listed in Table 1. For each task, we first train an agent for 1M environment steps under the nominal reward specification using auxiliary reward conditioning. During this phase, the agent is exposed to auxiliary rewards corresponding to the remaining tasks in the set. To evaluate transfer, we initialize a new task by loading the actor and critic parameters from the pretrained source agent, while reinitializing the optimizer state. We then finetune the transferred agent for an additional $500k$ environment steps under the target task reward. Following prior work on online finetuning [7, 79], initial exploration in the target environment is performed by sampling actions from the transferred policy rather than using random exploration. For

reward-conditioned agents, transfer is implemented by switching the reward embedding used for conditioning from the source task to the embedding corresponding to the target task. This procedure results in 56 source-target runs, with 10 seeds per pair. However, since `stair` and `slide` share the same reward function and differ only in environment layout, the two corresponding transfer directions do not involve a change in reward specification. We therefore exclude these two pairs from the aggregate transfer statistics, leaving 54 source-target pairs with distinct reward specifications. This protocol is applied to both SIMBAV2+RCRL and the vanilla SIMBAV2 baseline.

F.3 Normalization

Here, we describe the normalization of the evaluation score used throughout the paper. We follow the procedures adopted in prior work [45, 48]. For all benchmarks, we use the following normalization:

$$score^{norm} = \frac{score - score^{random}}{score^{optimal} - score^{random}}, \tag{1}$$

where $score^{norm}$ denotes the normalized evaluation returns, $score$ denotes unnormalized evaluation returns, $score^{optimal}$ denotes the returns achieved by an optimal policy as defined in previous works, and $score^{random}$ denotes the returns achieved by a random policy.

Table 5: Random and optimal scores used for normalization.

Benchmark	Task	Random Score	Optimal Score
DeepMind Control	dog-stand	0.000	1000
DeepMind Control	dog-walk	0.000	1000
DeepMind Control	dog-trot	0.000	1000
DeepMind Control	dog-run	0.000	1000
OpenAI Gym	Ant-v4	-70.288	3942
OpenAI Gym	HalfCheetah-v4	-289.415	10574
OpenAI Gym	Hopper-v4	18.791	3226
OpenAI Gym	Humanoid-v4	120.423	5165
OpenAI Gym	Walker2d-v4	2.791	3946
HumanoidBench	balance_hard-v0	10.032	800
HumanoidBench	balance_simple-v0	10.170	800
HumanoidBench	basketball-v0	8.979	1200
HumanoidBench	bookshelf_hard-v0	14.848	2000
HumanoidBench	bookshelf_simple-v0	16.777	2000
HumanoidBench	hurdle-v0	2.371	700
HumanoidBench	insert_normal-v0	1.673	350
HumanoidBench	insert_small-v0	1.653	350
HumanoidBench	maze-v0	106.233	1200
HumanoidBench	pole-v0	19.721	700
HumanoidBench	reach-v0	-5024	12000
HumanoidBench	run-v0	1.927	700
HumanoidBench	sit_hard-v0	2.477	750
HumanoidBench	sit_simple-v0	10.768	750
HumanoidBench	slide-v0	3.142	700
HumanoidBench	spoon-v0	4.661	650
HumanoidBench	stair-v0	3.161	700
HumanoidBench	stand-v0	11.973	800
HumanoidBench	walk-v0	2.505	700
HumanoidBench	window-v0	2.713	650

F.4 Hyperparameters

We detail the hyperparameters used in our experiments in Tables 6, 7 and 8. As noted in Section 4, we adopt the configurations of the baseline methods to which RCRL is applied. A detailed discussion of PRC and ARC is provided in Appendix E. Other hyperparameters used in SIMBAV2, BRC, DRQV2 are in Lee et al. [45], Nauman et al. [48], and Yarats et al. [75] respectively.

Table 6: SimbaV2+RCRL

Hyperparameter	Value
UTD	2
Action repeat	2
Discount rate	Heuristic
Num atoms	101
$V_{\min}$	-5
$V_{\max}$	5
Buffer size	1e6
Actor architecture	SimbaV2
Actor depth	1
Actor width	128
Critic architecture	SimbaV2
Critic depth	2
Critic width	512
Total batch	512
α	0.5
Conditioning	PRC

Table 7: BRC+RCRL

Hyperparameter	Value
UTD	2
Action repeat	1
Discount rate	0.99
Num atoms	101
$V_{\min}$	-10
$V_{\max}$	10
Buffer size per task	1e6
Actor architecture	BroNet
Actor depth	1
Actor width	256
Critic architecture	BroNet
Critic depth	2
Critic width	4096
Total batch	2048
α	0.5
Conditioning	ARC

Table 8: DrQV2+RCRL

Hyperparameter	Value
UTD	1
Action repeat	2
Discount rate	0.99
TD(N)	3
Actor update delay	2
Exploration σ	Schedule
Buffer size	1e6
Actor architecture	MLP
Actor depth	2
Actor width	256
Critic architecture	MLP
Critic depth	2
Critic width	256
Total batch	512
α	0.5
Conditioning	PRC

F.5 Additional Description of Key Figures

All plots report mean performance across multiple random seeds. Confidence intervals are computed using nonparametric bootstrapping with RLiable [3] and correspond to 95% confidence intervals.

Figures 1 & 3. Scores are first normalized within each benchmark following the standard practices described in Appendix F.3. We then aggregate results across benchmarks by flattening normalized scores across tasks and random seeds. For example, if one benchmark contains 3 tasks and another contains 5 tasks, and each task is evaluated with 10 seeds, statistics are computed over an array of shape $[8, 10]$. In both Figures, the performance at each evaluation step is computed by averaging returns over 10 evaluation rollouts and seeds. We use 10 seeds for single-task, 5 seeds for multi-task, 5 seeds for vision-based, and 10 seeds for steerability experiments.

Figure 8. This figure presents the reward compatibility analysis across 8 HumanoidBench tasks, comprising 48 nominal-counterfactual reward pairs. All experiments use 1M environment steps and 10 random seeds. The left panel shows the contribution of individual counterfactual rewards to nominal task learning, measured by the difference in AUC between full RCRL and leave-one-out variants ($C_{i,j}$). The middle panel relates these contributions to the correlation between nominal and counterfactual reward signals, while the right panel examines their relationship with critic gradient conflict frequency calculated according to Yu et al. [77] methodology.

Figure 10. This figure reports zero-shot adaptation results on three DeepMind Control tasks: `cheetah-run`, `hopper-hop`, and `humanoid-walk`. The vanilla single-task agent is trained exclusively under the nominal reward. The SIMBAV2+RCRL agent collects experience only under the nominal reward but learns additional reward functions off-policy through reward conditioning. In contrast, the final agent collects experience and learns under all reward functions using relabeling, and has an expanded data budget. All methods are evaluated without further training by varying only the reward specification provided to the policy at test time. For each task, the results are averaged over 10 random seeds, with evaluation returns computed as the mean over 10 rollout episodes per reward specification. For each task, the x-axis corresponds to the target behavioral characteristic specified by the reward specification (target speed for `cheetah-run`, target height for `hopper-hop`, and control cost multiplier for `humanoid-walk`). The top row shows the observed behavioral characteristic achieved by the policy at deployment (e.g. actual running speed or standing height), while the bottom row reports the corresponding return under the specified reward function.

F.6 Baseline Algorithms

SimbaV2 [45] is a recent state-of-the-art off-policy actor-critic algorithm for continuous control that combines categorical value learning [10] with hyperspherical [46] value representations. It achieves strong performance and sample efficiency across a wide range of locomotion tasks and serves as

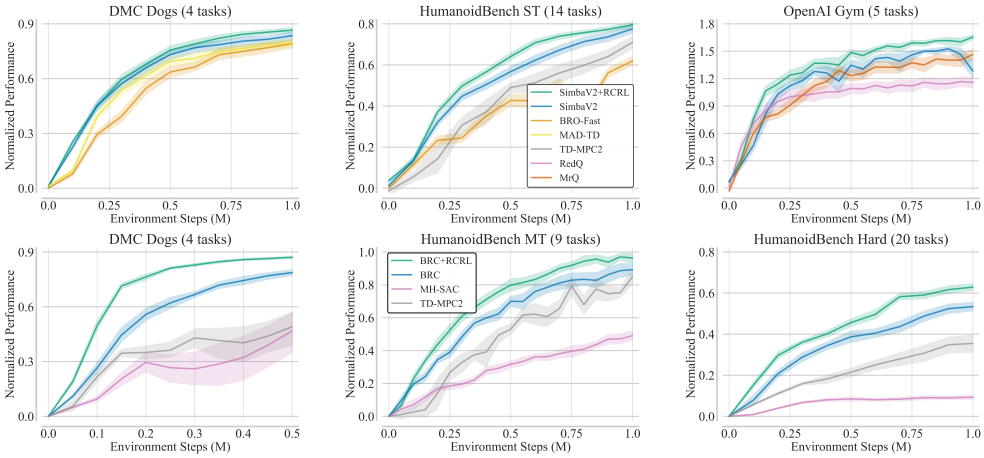


Figure 17: ***Sample efficiency of RCRL when evaluated under the nominal reward.*** Results for single-task (top row) and multi-task (bottom row) benchmarks corresponding to results presented in Figure 3.

our primary single-task RL baseline. We combine SIMBAV2 with RCRL as the backbone for both single-task and transfer experiments. We use the implementation and results provided in the source manuscript (`https://github.com/DAVIAN-Robotics/SimbaV2`).

BRC [48] is a single and multi-task RL algorithm designed for efficient learning. Its core design components are scaled value network, conditioning on task embeddings, and distributional value learning with reward normalization. We use BRC as the base model for evaluating RCRL in the multi-task settings. We use the implementation and results provided in the source manuscript (`https://github.com/naumix/BiggerRegularizedCategorical`).

DrQV2 [75] is a strong baseline for vision-based RL that combines data augmentation with off-policy actor-critic learning. It uses relatively simple architectures and optimization techniques compared to recent large-scale methods, making it a useful testbed for evaluating whether RCRL can be applied effectively in pixel-based control without relying on stabilization mechanisms. We use the implementation provided in `https://github.com/sukhijab/maxinforl_jax`.

BRO [47] is an off-policy reinforcement learning algorithm that demonstrates how scaling the value network can lead to substantial performance improvements in online RL. Its core contribution is a scaled critic architecture based on a normalized ResNet, which enables stable training of large value networks. We use the implementation and results provided in the source manuscript (`https://github.com/naumix/BiggerRegularizedOptimistic`).

MAD-TD [70] is an approach designed to stabilize training under high update-to-data ratios in off-policy RL [38, 34, 49, 18]. It mitigates instability and overestimation by augmenting real replay data with a small amount of model-based transitions. We use the results provided in Lee et al. [45].

TD-MPC2 [30] is a model-based RL algorithm that combines latent dynamics modeling with planning and policy learning and was shown to robustly perform across a variety of benchmarks, while using a single hyperparameter configuration. We use the results provided in Lee et al. [45].

REDQ [16] is an early approach designed to stabilize training under high update-to-data ratios in off-policy RL by using an ensemble of critics and randomized target selection during updates. It represents a strong and widely used baseline for continuous control in OpenAI gym tasks. We use the results provided in Lee et al. [45].

MrQ [24] is a model-free RL algorithm that was shown to perform well across both discrete and continuous control tasks using a unified hyperparameter configuration. It integrates model-based representations into a model-free learning process in the style of TD3 [23] to improve robustness across varied tasks. We use the results provided in Lee et al. [45].

MH-SAC [76] is a multi-head variant of Soft Actor-Critic [27, 28] designed for multi-task RL, where different tasks are represented via separate output heads while sharing lower-level representations. It provides an early and influential example of conditioning policies on task identity in multi-task RL. We use the implementation and results provided in Nauman et al. [48].

G Training Curves

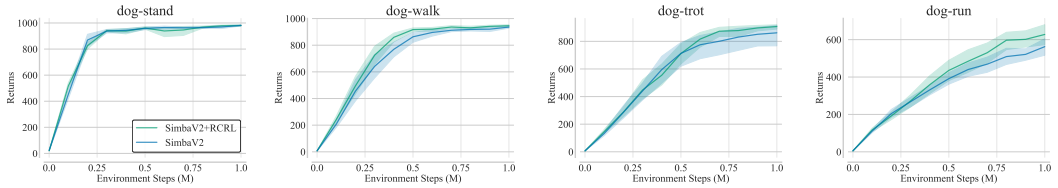


Figure 18: ***Single-task DeepMind Control performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 10 random seeds.

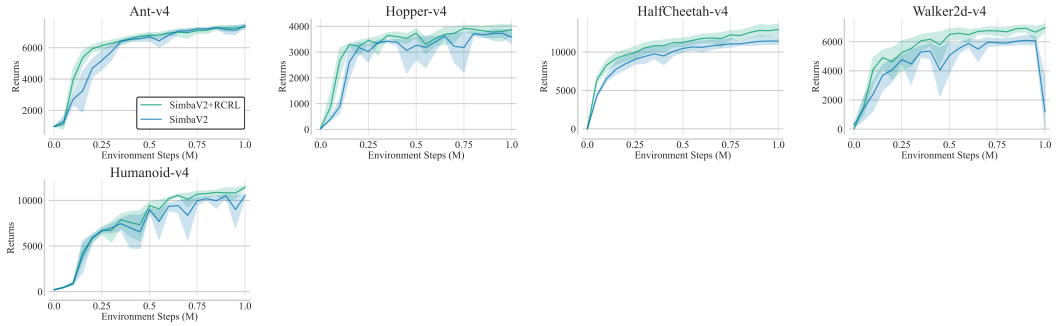


Figure 19: ***Single-task Gym performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 10 random seeds.

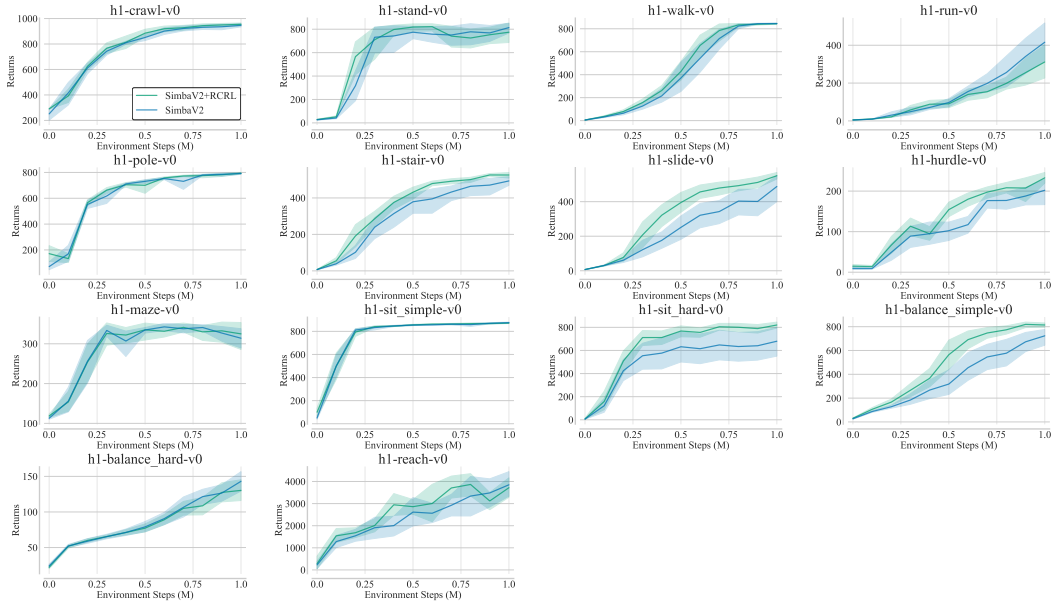


Figure 20: ***Single-task HumanoidBench performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 10 random seeds.

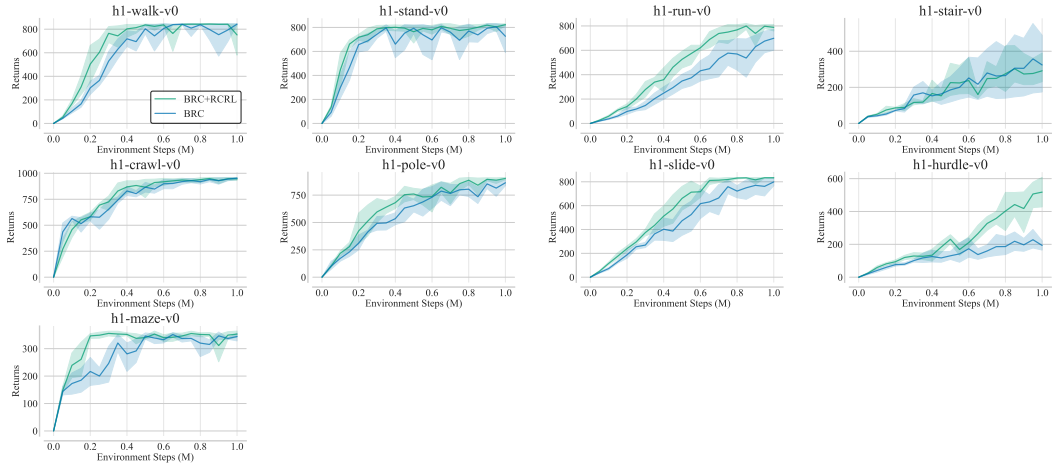


Figure 21: ***Multi-task HumanoidBench performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 5 random seeds.

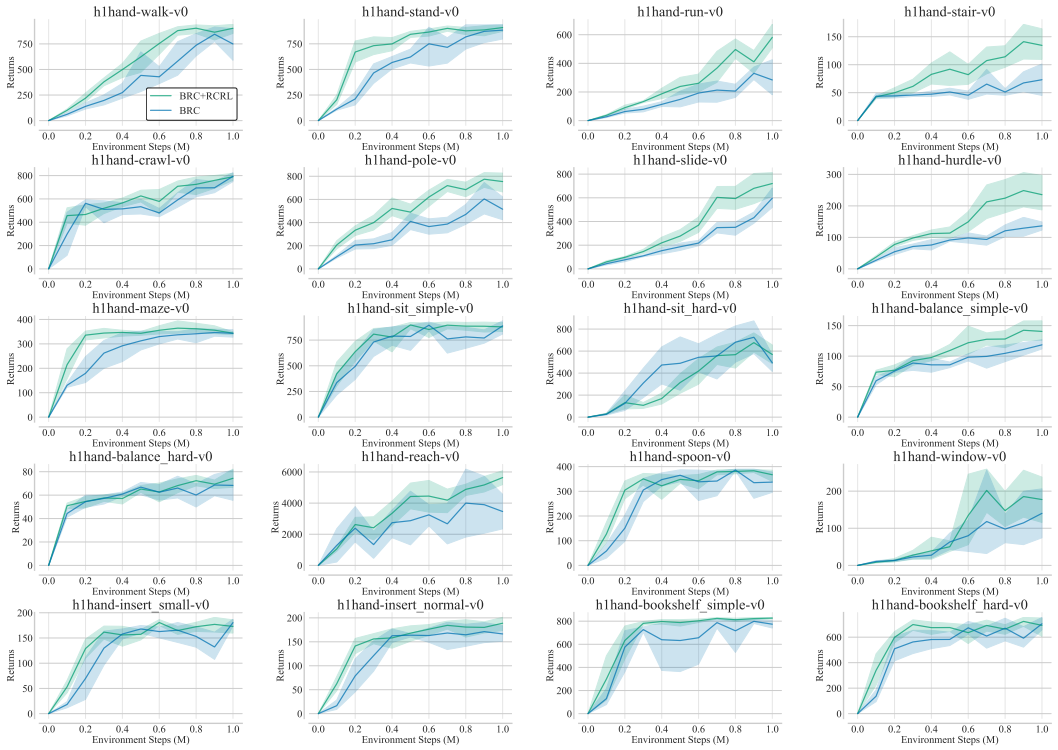


Figure 22: ***Multi-task HumanoidBench Hard performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 5 random seeds.

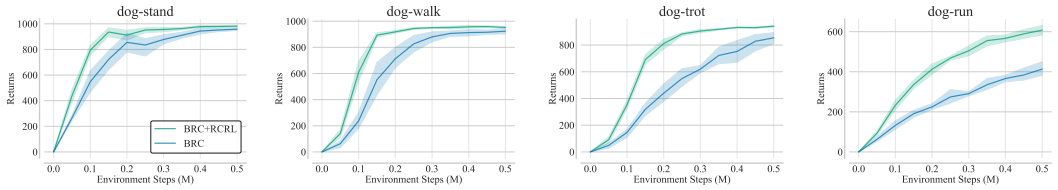


Figure 23: ***Multi-task DeepMind Control performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 5 random seeds.

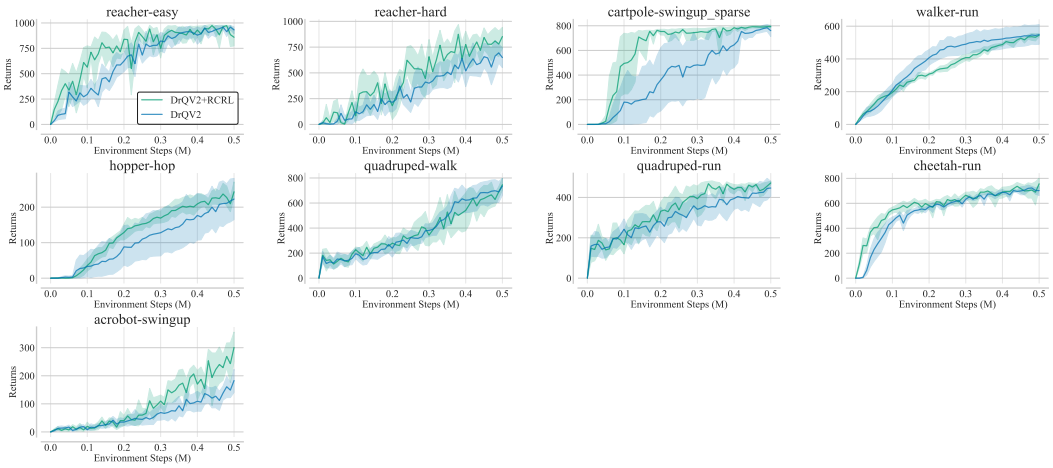


Figure 24: ***Vision-based DeepMind Control performance.*** We present mean and 95% confidence intervals calculated using bootstrapping. 5 random seeds.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope, as supported by Sections 3, 4 and A.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in the appendix A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include new theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Information on the experimental details are shared in Sections 4, E, and F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The paper builds on open-source code-bases listed in Appendix E and F. We plan to open-source full code and results soon.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Information on the experimental details are shared in Sections 4, E, and F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All results report 95% CI computed via stratified bootstrap using the RLiable [3].

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Computational resources are detailed in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We do not work on human subjects or sensitive data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses potential wider impact in the appendix.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No risk of misuse and no safeguards are required.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All benchmarks used are open-source and cited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not require IRB approval.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs use only for editing text.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.