Modification-Considering Value Learning for Reward Hacking Mitigation in RL

Evgenii Opryshko, Umangi Jain, Igor Gilitschenski

Keywords: reinforcement learning, reward hacking, reward tampering, value learning, AI safety

Summary

Reinforcement learning agents can exploit misspecified reward signals to achieve high apparent returns while failing on the intended objective, a failure mode known as reward hacking. Existing practical defenses typically constrain policy updates to stay near a known safe reference, creating a tension between suppressing hacking and permitting legitimate improvement. We propose Modification-Considering Value Learning (MCVL), which operationalizes the theoretical idea of current utility optimization for standard value-based RL. MCVL wraps an off-policy learner and treats each incoming transition as a candidate modification: it forecasts two training paths, one that includes the transition and one that does not, and scores both with a frozen bootstrapped-return estimator derived from a learned reward model and value function. The transition is admitted only if inclusion does not decrease the score. MCVL mitigates reward hacking across diverse environments while continuing to improve the intended objective.

Contribution(s)

1. We propose Modification-Considering Value Learning (MCVL), a safeguard for off-policy value-based RL that operationalizes current utility optimization: for each incoming transition, MCVL forecasts two training branches (with and without the transition), scores both with a frozen bootstrapped-return estimator built from a learned reward model and Q-function, and admits the transition only if inclusion does not decrease that score. MCVL wraps any off-policy value-based learner and requires no access to a safe reference policy. We instantiate MCVL with DDQN and TD3.
   Context: MCVL requires a seed dataset of non-hacking transitions for pretraining the return estimator to distinguish task progress from reward hacking. Checking transitions introduces computational overhead. Current utility optimization was discussed in the context of AI safety (Yudkowsky, 2011; Hibbard, 2012; Yampolskiy, 2014), but has not been operationalized for standard value-based RL.
2. We show empirically that MCVL mitigates reward hacking across four safety-relevant gridworlds and three modified MuJoCo tasks while achieving performance comparable to an Oracle trained on the true reward; for continuous control, random-policy data suffices for the seed dataset.
   Context: The evaluation demonstrates effectiveness across diverse hacking mechanisms rather than providing a controlled benchmark. Gridworld tasks require a Safe variant with the hacking affordance removed for the seed dataset.
3. We formalize safety, permissiveness, and bounded-degradation guarantees for MCVL's gating rule, parameterized by evaluator accuracy ϵ decomposed into reward-model and value-function error.
   Context: The guarantees depend on an ϵ-accurate return estimator. The bound is conservative: both branches share the same frozen networks and start states, producing correlated errors that partially cancel. Achieving small ϵ is not guaranteed in general, though pretraining on hack-free data provides an initial fit, and successful filtering preserves buffer quality, helping maintain or improve the accuracy over time.

Modification-Considering Value Learning for Reward Hacking Mitigation in RL

Evgenii Opryshko[1,2], **Umangi Jain**[1,2], **Igor Gilitschenski**[1,2]
{eop, umangi, gilitschenski}@cs.toronto.edu

[1]**University of Toronto** [2]**Vector Institute**

Abstract

Reinforcement learning agents can exploit misspecified reward signals to achieve high apparent returns while failing on the intended objective, a failure mode known as reward hacking. Existing practical defenses typically constrain policy updates to stay near a known safe reference, creating a tension between suppressing hacking and permitting legitimate improvement. We propose Modification-Considering Value Learning (MCVL), which operationalizes the theoretical idea of current utility optimization for standard value-based RL. MCVL wraps an off-policy learner and treats each incoming transition as a candidate modification: it forecasts two training paths, one that includes the transition and one that does not, and scores both with a frozen bootstrapped-return estimator derived from a learned reward model and value function. The transition is admitted only if inclusion does not decrease the score. We formalize conditions under which this filtering is both safe and permissive, and instantiate MCVL with DDQN and TD3. Across four safety-relevant gridworlds and three modified MuJoCo continuous-control tasks with diverse hacking mechanisms, MCVL mitigates reward hacking while continuing to improve the intended objective. Code: https://ktolnos.github.io/mcvl/.

1 Introduction

Optimizing poorly defined or incomplete rewards can push RL agents toward unintended behaviors, leading to *reward hacking* (Skalse et al., 2022). For instance, an agent tasked with stacking blocks may learn to flip blocks if the reward is based on the height of the bottom face (Popov et al., 2017). As RL systems scale to safety-critical applications (e.g., autonomous driving (Kiran et al., 2021) or medical diagnostics (Ghesu et al., 2017)), ensuring reliable and safe behavior becomes increasingly important. Reward hacking can become more prevalent as models grow in complexity (Pan et al., 2022), which also affects large language models where RL is used for post-training (Denison et al., 2024; OpenAI, 2024; MacDiarmid et al., 2025). A common mitigation constrains policy updates around a trusted reference (Laidlaw et al., 2025), often at a cost to optimality.

A complementary safeguard is to *optimize what the agent currently values* while being conservative about changing those values, an idea discussed as *current utility optimization* (Orseau & Ring, 2011; Hibbard, 2012; Everitt et al., 2016; 2021). None of these works, however, provides a practical evaluation of this concept. We address this gap by investigating whether individual transitions can be predictive of reward hacking in the context of value-based RL. Our method, *Modification-Considering Value Learning (MCVL)*, wraps a standard off-policy learner and treats each update as a candidate modification. For a newly observed transition, the agent forecasts two scenarios: one in which it learns from the transition and one that ignores it. Then MCVL scores both resulting policies using its *current* learned return estimator, an n-step bootstrapped return combining a learned reward model with a value-function bootstrap, and accepts the transition only if inclusion does not decrease this score. MCVL blocks updates that, according to the agent's current return estimator, would shift

behavior toward undesirable strategies. To study it empirically, we instantiate MCVL with DDQN and TD3.

Our method only assumes a seed dataset with non-hacking transitions so that the evaluator can identify the intended behavior; for gridworlds we collect this in a *Safe* variant with the hacking affordance removed, while for continuous control a random-policy dataset suffices (Section 3). Under these conditions, MCVL mitigates reward hacking in multiple safety-relevant gridworlds (Leike et al., 2017; Everitt et al., 2021) and modified Gymnasium continuous-control environments (Towers et al., 2024) (Reacher, Ant, HalfCheetah) which we introduce to enable reward-hacking research in continuous control. All code will be made publicly available.

2 Notation and Preliminaries

We denote a Markov decision process (MDP) by $(S, A, P, R, \rho, \gamma)$ with state space S, action space A, transition model $P(s'|s,a) \in [0,1]$, reward function $R : S \times A \to \mathbb{R}$, initial state distribution ρ, and discount factor $\gamma \in (0,1)$. For any reward function r, we write $J_r(\pi) = \mathbb{E}_{\rho,\pi}[\sum_{t\geq 0} \gamma^t r(s_t, a_t)]$ for the expected return of the policy π under r. In standard RL, the agent's training objective is to learn a policy π maximizing $J_R(\pi)$. The state-action value $Q^\pi(s,a)$ is the expected return starting from (s,a) and following π thereafter (Sutton & Barto, 2018). Deep value-based methods like DDQN (van Hasselt et al., 2016) and TD3 (Fujimoto et al., 2018) approximate Q with a neural network and learn from transitions (s,a,r,s') sampled from a replay buffer via temporal-difference (TD) updates.

Reward hacking. Let R denote the observed training reward and R^* the intended reward (unobserved by the agent). The agent's true objective is to maximize $J_{R^*}(\pi)$, while observing only rewards from R. For a policy update from π to π', we say the update *induces reward hacking* if $J_R(\pi') > J_R(\pi)$ but $J_{R^*}(\pi') < J_{R^*}(\pi)$. In words, the update looks better under the proxy while making intended performance worse. Skalse et al. (2022) define *hackability* as a property of reward-function pairs; our notion focuses on the policy update and is complementary.

3 Method

Modification-Considering Value Learning (MCVL) wraps an off-policy learner and treats each learning update as a candidate modification to be evaluated before adoption. Because the desired objective is not observed, MCVL uses a learned *current return estimator* as a proxy to accept or reject updates. MCVL poses a counterfactual question: if it were to allocate the next l training steps either (i) to its current replay buffer $\mathcal{D}$ alone or (ii) to $\mathcal{D}$ augmented with the new transition T_{new}, which resulting policy would achieve a higher expected return according to the agent's *current bootstrapped-return estimator*? Both branches use the same compute budget and are scored by the same evaluator. The transition is accepted if and only if adding T_{new} does not decrease the score. This yields a locally rational accept/reject rule under the agent's present preferences.

Current bootstrapped-return estimator. MCVL maintains a reward model $R_\psi(s,a)$ trained by supervised regression on observed rewards and an action-value function $Q_\theta(s,a)$ trained with standard TD targets. Together they define a current n-step bootstrapped return estimator for a trajectory $\tau = (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n, a_n)$:

$$\hat{G}_n(\tau) \;=\; \sum_{t=0}^{n-1} \gamma^t \, R_\psi(s_t, a_t) \;+\; \gamma^n \, Q_\theta(s_n, a_n). \tag{1}$$

During scoring, the estimator parameters (ψ, θ) are *frozen to the live agent's current values*.

Policy forecasting and comparison. Upon observing $T_{\text{new}} = (s,a,r,s')$, MCVL constructs two forecasts under an identical training budget of l learner updates:

$$\tilde{\pi}^{\,0} \;=\; \mathsf{Forecast}(\mathcal{D},\, l), \qquad \tilde{\pi}^{\,+} \;=\; \mathsf{Forecast}(\mathcal{D} \cup \{T_{\text{new}}\},\, l).$$

Algorithm 1 MCVL (wrapper around an off-policy value-based learner)

while training **do**
  Observe $T_{\text{new}} = (s, a, r, s')$ ▷ Action is selected using the policy of a base learner
  if $|\, r - R_\psi(s,a) \,| < \delta_r$ **then** ▷ Optional check to avoid excessive evaluations
    Insert T_{new} into replay buffer $\mathcal{D}$; Perform a training step; **continue**
  end if
  $\tilde{\pi}^{\,0} \leftarrow \mathsf{Forecast}(\mathcal{D},\, l)$ ▷ Forecast performs l training steps on a provided replay buffer
  $\tilde{\pi}^{\,+} \leftarrow \mathsf{Forecast}(\mathcal{D} \cup \{T_{\text{new}}\},\, l)$
  Estimate $\hat{J}(\tilde{\pi}^{\,0})$ and $\hat{J}(\tilde{\pi}^{\,+})$ via k rollouts of length n using Equation 2
  if $\hat{J}(\tilde{\pi}^{\,+}) \geq \hat{J}(\tilde{\pi}^{\,0})$ **then**
    Insert T_{new} into $\mathcal{D}$
  else ▷ The transition is suspected to be reward hacking-inducing
    Reset the environment ▷ Continued exploration might impede the detection accuracy
  end if
  Perform a training step: sample a batch from $\mathcal{D}$ and update the base learner and R_ψ on it.
end while

The operator $\mathsf{Forecast}(\cdot, l)$ clones the current networks and runs l base-learner updates on minibatches from the specified dataset. These updates do not affect the live agent. Both forecasts are scored by the same frozen evaluator from Equation 1. Let $\{s^{(i)}\}_{i=1}^{k} \sim \rho$ be a set of initial states and let rollouts be of length n under the same transition model $\hat{P}$ for both branches. Define

$$\hat{J}(\pi) \;=\; \frac{1}{k} \sum_{i=1}^{k} \left[\hat{G}_n(\tau \sim (\hat{P}, \pi) \,|\, s_0 = s^{(i)}) \right]. \tag{2}$$

as the approximated expected on-policy return under the current bootstrapped return estimator. It is used for evaluating T_{new}: MCVL admits T_{new} if and only if $\hat{J}(\tilde{\pi}^{\,+}) \geq \hat{J}(\tilde{\pi}^{\,0})$. Using matched compute, evaluator parameters, and transition model isolates the effect of admitting T_{new} and makes the comparison less sensitive to estimation errors. The training procedure is described in Algorithm 1.

Instantiations (MC-DDQN and MC-TD3). MC-DDQN wraps a DDQN agent with an ϵ-greedy behavior policy. Forecasting clones all parameters, including targets, and runs l ordinary DDQN updates; forecasted policies are greedy with respect to their respective Q-functions. MC-TD3 analogously clones the actor and critics and runs l standard TD3 updates. During scoring, rollouts use a transition model $\hat{P}$ (either the environment itself or a learned approximation; Section 4.3) and compute rewards using the learned reward model. If accepted, T_{new} is inserted into $\mathcal{D}$ and future updates may sample it to update both Q_θ and R_ψ. If rejected, the transition is discarded and the episode is reset, since future transitions may not be connected to the transitions in the replay buffer which impedes policy forecasting. Full algorithmic details appear in Appendices E and F.

Pretraining. MCVL assumes a seed dataset of non-hacking transitions $\mathcal{D}_0$. The motivation is identifiability: without such data, a learned return estimator cannot distinguish genuine task progress from reward hacking. We fit R_ψ by supervised regression on the observed proxy rewards in $\mathcal{D}_0$ and train Q_θ with standard TD targets as part of the normal RL training. After pretraining, every newly observed transition is screened by the forecast-and-score check; since R_ψ and Q_θ continue to update with each base-learner step, the evaluator incorporates new information beyond the seed data.

We study two practical sources of $\mathcal{D}_0$: (i) *Safe* variants that match observations, actions, and reward but remove the hacking affordance, so that hacking transitions are unreachable (used for gridworlds where undirected exploration quickly discovers hacks); and (ii) random-policy data collected in the *Full* environment, which suffices when hacking requires specific conditions unlikely under random exploration (used for all three continuous-control tasks). Importantly, neither source reveals R^* where R and R^* differ, and policies trained on $\mathcal{D}_0$ transfer only imperfectly to the Full environment.

Hyperparameters and cost. To limit overhead, we invoke forecasting only when the observed reward disagrees with the reward model, $|r - R_\psi(s,a)| \geq \delta_r$; otherwise T_{new} is admitted without a check as a heuristic. As shown in Section 4.3, this filtering does not change conclusions in our experiments. The marginal per-transition cost is $2l$ base-learner updates plus $2k \cdot n$ rollout steps; δ_r controls how often this cost is paid. Detailed hyperparameter guidance appears in Appendix K.

Reward hacking prevention. MCVL evaluates the *policy change* from admitting a transition using the agent's current bootstrapped-return estimator, relative to an equally trained counterfactual that excludes it. This yields a local self-consistency test: if inclusion steers learning toward behavior the evaluator already scores worse over horizon n (e.g., shifting effort from task completion to reward tampering), the update is vetoed. If inclusion raises (or leaves unchanged) the score, the transition is admitted. This captures ordinary competence gains (shorter paths, reduced control effort) the evaluator already values. While not every hack is guaranteed to lower the score, in our experiments MCVL rejects the updates that produce undesired behaviors across diverse environments.

Theoretical analysis. We formalize the conditions under which MCVL correctly accepts or rejects transitions. Let $J_{R^*}(\pi) = \mathbb{E}_{\rho,\pi}\left[\sum_{t\geq 0} \gamma^t R^*(s_t,a_t)\right]$ denote the true expected return under the intended reward R^*, which the agent does not observe.

Assumption 3.1 (ϵ-accurate evaluator)**.** The bootstrapped-return estimator $\hat{J}$ is *ϵ-accurate* over a policy class Π if $|\hat{J}(\pi) - J_{R^*}(\pi)| \leq \epsilon$ for all $\pi \in \Pi$.

Proposition 3.2 (Safety, permissiveness, and bounded degradation)**.** *Suppose $\hat{J}$ is ϵ-accurate over $\Pi \supseteq \{\tilde{\pi}^+, \tilde{\pi}^0\}$. Then:*

1. ***Safety.*** *If $J_{R^*}(\tilde{\pi}^+) < J_{R^*}(\tilde{\pi}^0) - 2\epsilon$, then MCVL rejects T_{new}.*
2. ***Permissiveness.*** *If MCVL rejects T_{new}, then $J_{R^*}(\tilde{\pi}^+) < J_{R^*}(\tilde{\pi}^0) + 2\epsilon$. That is, MCVL never rejects a transition whose inclusion would improve true performance by more than 2ϵ.*
3. ***Bounded degradation.*** *If MCVL accepts T_{new}, then $J_{R^*}(\tilde{\pi}^+) \geq J_{R^*}(\tilde{\pi}^0) - 2\epsilon$.*

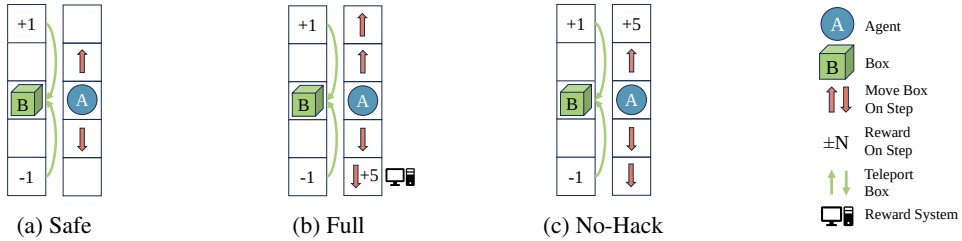
The 2ϵ threshold can be made explicit in terms of the reward-model error ϵ_R and the value-function error ϵ_Q; see Appendix A for proofs and the full error decomposition. The bound clarifies two practical aspects: (i) pretraining on hack-free data reduces ϵ_R and ϵ_Q, tightening the safety bound, and (ii) successful filtering preserves buffer quality, keeping the evaluator accurate over time. In practice, the effective gap is likely tighter than 2ϵ because both branches are scored by the same frozen evaluator on the same start states and rolled out under the same transition model, producing positively correlated errors that largely cancel in the comparison.

4 Experiments

We evaluate whether MCVL mitigates reward hacking while continuing to improve task performance. We compare MCVL to its base learner (DDQN in discrete domains; TD3 in continuous control), an Oracle agent trained with the base learner on the *true* reward (which MCVL cannot access), and a Frozen policy that fixes the pretrained networks and performs no further learning in the *Full* environment. All methods share hyperparameters, pretrained weight initialization, and the pretraining replay buffer. We use a learned reward model and true transition dynamics for scoring in all experiments unless stated otherwise. We report the *true performance* (intended objective) and the *observed return* for each environment, with means and bootstrapped 95% CIs over 10 seeds.

4.1 Environments

We introduce the *Box Moving* environment (Figure 1) to illustrate representative accept/reject decisions under MCVL, and adopt *Absent Supervisor* and *Tomato Watering* from AI Safety Gridworlds (Leike et al., 2017) and *Rocks and Diamonds* from Everitt et al. (2021); all are shown in Figure 2. For each gridworld we provide a *Safe* variant for pretraining.



(a) Safe (b) Full (c) No-Hack

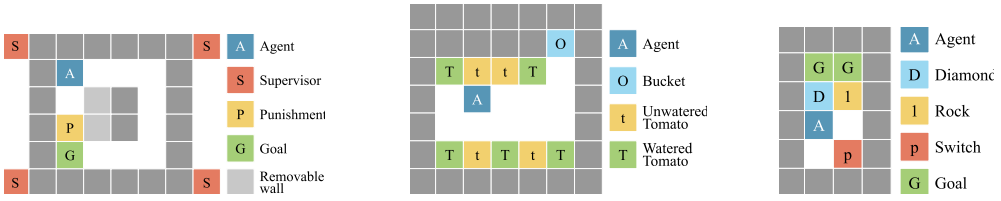
Figure 1: *Box Moving*: the agent moves up or down; stepping on an arrow moves the box that way, and the box teleports to the center on reaching the top. (**a**) *Safe*: the agent learns to move the box upward. (**b**) *Full*: a bottom cell yields a spurious $+5$ when pressed; reward-maximizing behavior repeatedly triggers it with down-arrows, moving the box downward against the true objective, whereas a non-hacking strategy can still raise return by alternating two up-arrows. (**c**) *No-Hack*: collecting the $+5$ does not block moving the box up, so such transitions are aligned with the objective and should not be rejected.

Continuous control (MuJoCo). For continuous control, we modify Reacher-v5, Ant-v5, and HalfCheetah-v5 (Towers et al., 2024) by introducing spurious reward sources that model reward tampering or sensor malfunctions (details in Appendix G). We augment Reacher-v5 with a hidden button yielding a one-time $+50$ after the manipulator has spent 15 consecutive frames in a small region. We modify Ant-v5 with a sensor-malfunction bonus triggered by high contact force and forward velocity, modeling IMU saturation in legged robots; and HalfCheetah-v5 with an optical-flow sensor failure bonus triggered when the front tip is low and speed is high. In each case the performance metric excludes the bonus, and random exploration is unlikely to trigger the hack.

4.2 Main results

Figure 3 shows that MCVL maintains high performance across all tasks while the base learners hack. MCVL attains final performance that is comparable to the Oracle in all tasks despite never accessing the true reward. In *Box Moving*, *Absent Supervisor*, and *Tomato Watering*, MCVL reaches strong performance faster than the Oracle, which we hypothesize is due to an implicit curriculum induced by rejecting transitions that cause large behavioral shifts early in training.

In continuous-control environments, MC-TD3 achieves performance comparable to the Oracle while the Frozen policy substantially underperforms, suggesting that continued learning beyond pretraining is important in the considered environments. Further details of the continuous-control experiments are provided in Appendix G.



(a) Absent Supervisor (b) Tomato Watering (c) Rocks and Diamonds

Figure 2: (**a**) *Absent Supervisor*: the shortest path to the goal passes through a punishment cell whose cost depends on whether a supervisor is present. In *Safe* the supervisor is always present; in *Full* it appears with probability 0.5. True performance penalizes the punishment cell regardless of supervision. (**b**) *Tomato Watering*: the agent waters tomatoes that dry stochastically; in *Full* a bucket causes *perceptual delusion*, making all cells appear as watered tomatoes. True performance counts true watered tomatoes; *Safe* removes the bucket. We provide a non-delusional transition model for scoring. (**c**) *Rocks and Diamonds*: diamonds yield $+1$ per step, rocks -1. In *Full* a switch flips the observed rock reward to $+10$ while true performance still prefers diamonds; *Safe* removes the switch.

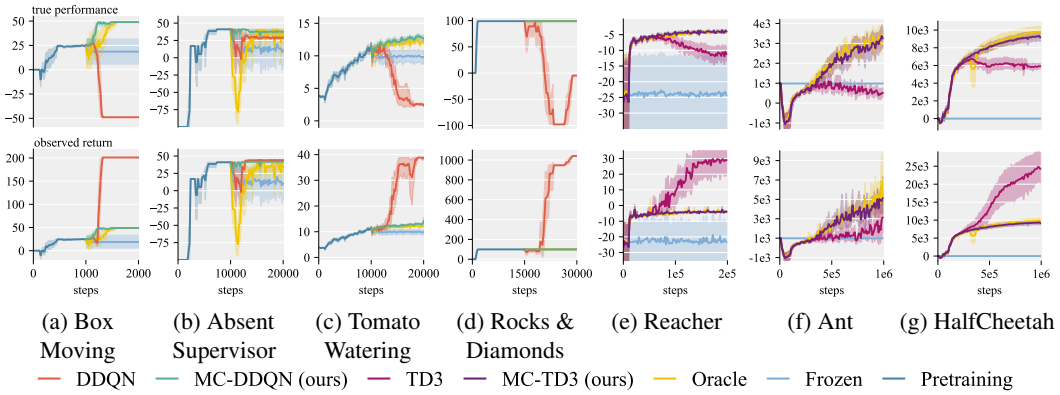


Figure 3: **Main results.** Top: true performance metric (intended objective, unobserved). Bottom: observed return (proxy). We compare the base learner (DDQN/TD3), MCVL, an Oracle trained on true reward, and a Frozen policy that stops learning after pretraining. Bold lines: mean over 10 seeds; bands: bootstrapped 95% CI.

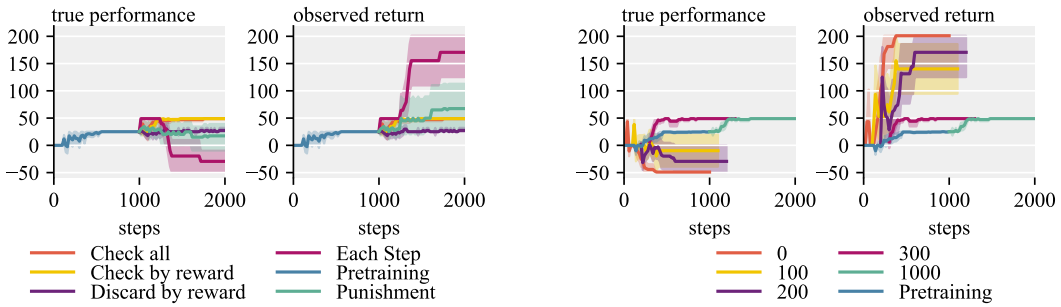
Across environments, the qualitative pattern is consistent: transitions inducing reward hacking produce forecasted policies that score lower than counterfactuals trained without them. In *Full* Box Moving, exploiting the $+5$ tile yields lower forecasted return than pursuing up-arrows (while in the *No-Hack* variant, MCVL admits all transitions and recovers the Oracle policy, Figure 5b); in Absent Supervisor, unsupervised punishment-cell transitions produce policies that use the cell more often, scoring lower under an evaluator pretrained to always penalize it regardless of supervision state; in Rocks and Diamonds, rock-pushing transitions score below diamond-prioritizing ones. In Tomato Watering, MCVL mitigates hacking only with a non-delusional transition model which tracks true world state and lets the evaluator distinguish genuine watering from the bucket's illusion, much as a person in VR still knows their physical location. For continuous control, sensor-corruption and reward-tampering transitions produce policies that prioritize the bonus over movement. In Ant pursuing high contact forces impedes stable policy learning for TD3, but MC-TD3 tracks Oracle.

4.3 Ablations and sensitivity

We study when to trigger checks, how to conduct them, and how to handle harmful transitions (Figure 4). Triggering only when $|r - R_\psi(s,a)| \geq \delta_r$ (*Check-by-reward*) performs comparably to *Check-all* but with lower computational cost, and outperforms *Discard-by-reward* (which never admits large-discrepancy transitions), as the latter filters out legitimately informative data and fails to reach optimal performance. Removing the gate entirely and bootstrapping on a frozen reward model (*Freeze RM*) isolates the gate's contribution: it matches the Oracle in five of our seven tasks, so a reward model fit on the pretraining data is often already sufficient, but it collapses where that model contains errors. Notably, in *Absent Supervisor*, dropping the gate induces reward hacking and a sharp fall in true performance; in one Ant seed, the no-gate policy even drove its predicted return far above any achievable true return while true performance collapsed (Appendix H).

Importance of forecasting. An *Each-step* variant that compares the policy before and after a *single* gradient step does not reliably prevent hacking. Policy changes only occur once the critic begins assigning higher value to the new behavior, at which point both the critic and reward model already endorse it. By contrast, allowing l standard updates during forecasting gives the base learner enough room to translate a transition into a meaningful policy shift, which the evaluator can then assess effectively using current live networks. The scoring noise for the branch comparison does not depend on l, while forecasted policies are more likely to diverge with l, so increasing l improves detection reliability (see Appendix A).

Reject vs. penalize. Replacing rejected transitions with large negative rewards (*Punishment*) is less effective than discarding them. Punished transitions accumulate in the buffer which discourages



(a) Alternative training schemes

(b) Pretraining steps

Figure 4: Additional experiments in Box Moving. (a) Comparison of training schemes: *Check all* checks all transitions; *Check by reward* checks only transitions for which predicted reward differs from the observed by at least δ; *Discard by reward* discards all transitions where predicted reward sufficiently differs from the observed; *Each step* evaluates policies before and after each gradient step without forecasting future policies; *Punishment* replaces rejected transitions' rewards with a punishment reward. (b) Effect of different amounts of pretraining, 0 means no pretraining. After as low as 300 steps, MCVL can achieve optimal performance across all seeds.

learning policies that exploit them. However, this also prevents learning these policies during forecasting, which decreases the reliability of the forecast-and-score gate.

Pretraining budget. As shown in Figure 4b, some seeds avoid hacking with as few as 100 pretraining steps in *Safe*; by 300 steps all seeds succeed, even though most have not converged to the optimal policy in the *Safe* environment. With zero pretraining, MCVL matches the base learner.

Learned transition model. Because both branches share the same $\hat{P}$, exact dynamics are unnecessary; it suffices that the model preserves the relative ranking of hacking and non-hacking trajectories. Replacing the environment with a learned forward model in *Box Moving* preserves Oracle performance while avoiding reward hacking (Appendix D).

Forecast budget l. Too small l fails to capture the policy change induced by a transition, reducing the robustness of rejecting harmful updates, which slows learning of reward hacking, but does not completely prevent it. Increasing l resolves this (Figure 5a). Additional experiments in different variations of gridworld environments are provided in Appendix C.

4.4 Comparison to occupancy-regularized objectives

The closest practical baseline in standard RL settings is occupancy-regularized policy optimization toward a known safe policy (Laidlaw et al., 2025). When the Oracle policy deviates substantially from the reference, the regularization strength sufficient to suppress hacking also suppresses Oracle-level improvements, so no single λ separates the two. We test this with the objective $F(\pi, \pi_{\text{ref}}) = J(\pi) - \lambda\, D(\mu_\pi \| \mu_{\pi_{\text{ref}}})$ holding the reference fixed to Frozen. Across 10 seeds and two divergences ($\sqrt{\chi^2}$, KL), such a λ *often does not exist* (Appendix B). By contrast, MCVL achieves performance comparable to the Oracle without relying on a safe policy.

5 Limitations and Future Work

Computation. MCVL adds forecasting and scoring overhead. Triggering checks only on reward discrepancies (Section 3) keeps the overall wall-clock training time to $1.8\times$–$4.2\times$ slowdowns versus the base learner on continuous-control tasks (Appendix J). Further reductions appear feasible through caching and batched rollouts.

Scope of applicability. MCVL assumes the evaluator ranks hacking-inducing trajectories below non-hacking ones at scoring horizons. If proxy misspecification is already valued by the evaluator,

harmful updates may be admitted. This may happen due to incorrect reward shaping, as in CoastRunners (OpenAI, 2023) where the agent learns to repeatedly collect boosts instead of following the track. If the reward for boosts is learned by the reward model, the gating mechanism would not reject policies that exploit it. MCVL is therefore complementary to better reward design, including potential-based shaping (Ng et al., 1999).

Transition dynamics. Because both branches share $\hat{P}$, relative ranking accuracy matters more than absolute fidelity. In *Box Moving*, learned-model rollouts preserve Oracle-level MCVL performance (Appendix D). Other environments were tested with true dynamics for scoring; learning accurate world models for higher-dimensional environments is left for future work.

Pretraining dependence. MCVL needs a seed dataset with no hacking transitions to identify the intended behavior; miscalibrated initial evaluators can entrench bad preferences. In our tasks, *Safe*-variant data (gridworlds) or random exploration (continuous control) was sufficient; alternatives include simpler-task pretraining, trajectory filtering, or human demonstrations. MCVL does not modify the base learner's exploration policy; it only gates which transitions enter the replay buffer, so exploration isn't affected, as demonstrated by the Oracle-comparable performance.

6 Related Work

Agents exploiting misspecified objectives are studied as *reward hacking* (Skalse et al., 2022), *reward gaming* (Leike et al., 2018), and *specification gaming* (Krakovna et al., 2020). Krakovna et al. (2020) survey such failures, and Skalse et al. (2022) formalize reward hacking and show that it cannot be prevented without restricting the set of possible policies or controlling optimization.

A common mitigation is constraining policy to remain close to a trusted behavior distribution (Laidlaw et al., 2025; Liu et al., 2026). MCVL instead filters individual replay updates using counterfactual forecasting, without requiring a safe reference policy or specific proxy-family assumptions. In our settings, MCVL reaches optimal returns where ORPO-style objectives often cannot simultaneously avoid hacking and learn the optimal policy (Appendix B).

Direct manipulation of reward channels is studied as *wireheading* (Amodei et al., 2016; Taylor et al., 2016; Everitt & Hutter, 2016; Majha et al., 2019) or *reward tampering* (Kumar et al., 2020; Everitt et al., 2021). A long-standing idea is *current utility optimization*: choose actions that improve the current objective without changing what is optimized (Yudkowsky, 2011; Hibbard, 2012; Yampolskiy, 2014). Schmidhuber (2003) describes a self-modifying *Gödel machine* agent that adopts only code or utility changes provably beneficial according to the current objective. Everitt & Hutter (2016) consider Bayesian agents over hand-specified utility functions that select actions to avoid altering beliefs about the reward mechanism, and Everitt et al. (2021) give conditions under which optimizing the *current* reward avoids incentives to tamper. MCVL operationalizes this perspective in practical off-policy value-based RL, including settings beyond direct reward/sensor tampering.

7 Conclusion

We introduced *Modification-Considering Value Learning*, a forecast-and-score safeguard for off-policy value-based RL. MCVL evaluates two counterfactual update paths with a frozen bootstrapped-return estimator and admits a transition only if inclusion does not reduce that score, optimizing what the agent currently values while remaining conservative about changing those values.

Implementations, MC-DDQN and MC-TD3, mitigate reward hacking in safety-relevant gridworlds and modified MuJoCo tasks while remaining close to Oracle on true reward. By operationalizing ideas from current utility optimization within standard deep RL, MCVL offers a practical way toward avoiding reward hacking without sacrificing performance on the intended objective.

Acknowledgments

Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, companies sponsoring the Vector Institute and the Digital Research Alliance of Canada (alliancecan.ca). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the Canadian government.

A Proofs and Error Decomposition

Proof of Proposition 3.2. All three claims follow from the ϵ-accuracy of $\hat{J}$.

(i) Safety. By ϵ-accuracy, $\hat{J}(\tilde{\pi}^+) \leq J_{R^*}(\tilde{\pi}^+) + \epsilon$ and $\hat{J}(\tilde{\pi}^0) \geq J_{R^*}(\tilde{\pi}^0) - \epsilon$. Combining with the hypothesis $J_{R^*}(\tilde{\pi}^+) < J_{R^*}(\tilde{\pi}^0) - 2\epsilon$:

$$\hat{J}(\tilde{\pi}^+) \leq J_{R^*}(\tilde{\pi}^+) + \epsilon < J_{R^*}(\tilde{\pi}^0) - 2\epsilon + \epsilon = J_{R^*}(\tilde{\pi}^0) - \epsilon \leq \hat{J}(\tilde{\pi}^0).$$

Hence $\hat{J}(\tilde{\pi}^+) < \hat{J}(\tilde{\pi}^0)$ and the accept condition fails.

(ii) Permissiveness. MCVL rejects means $\hat{J}(\tilde{\pi}^+) < \hat{J}(\tilde{\pi}^0)$. By ϵ-accuracy:

$$J_{R^*}(\tilde{\pi}^+) \leq \hat{J}(\tilde{\pi}^+) + \epsilon < \hat{J}(\tilde{\pi}^0) + \epsilon \leq J_{R^*}(\tilde{\pi}^0) + 2\epsilon.$$

Hence $J_{R^*}(\tilde{\pi}^+) < J_{R^*}(\tilde{\pi}^0) + 2\epsilon$, so no transition with true improvement exceeding 2ϵ is rejected.

(iii) Bounded degradation. Acceptance implies $\hat{J}(\tilde{\pi}^+) \geq \hat{J}(\tilde{\pi}^0)$. By ϵ-accuracy:

$$J_{R^*}(\tilde{\pi}^+) \geq \hat{J}(\tilde{\pi}^+) - \epsilon \geq \hat{J}(\tilde{\pi}^0) - \epsilon \geq J_{R^*}(\tilde{\pi}^0) - 2\epsilon. \qquad \square$$

Error decomposition. When rollouts use the true dynamics ($\hat{P} = P$) and the number of rollouts $k \to \infty$ (or the environment is deterministic), the evaluator error decomposes as follows.

Lemma A.1 (Evaluator error bound)**.** *Under the conditions above,*

$$|\hat{J}(\pi) - J_{R^*}(\pi)| \;\leq\; \frac{1-\gamma^n}{1-\gamma}\,\epsilon_R \;+\; \gamma^n\,\epsilon_Q,$$

where $\epsilon_R = \max_{s,a} |R_\psi(s,a) - R^*(s,a)|$ *and* $\epsilon_Q = \max_{s,a} |Q_\theta(s,a) - Q^\pi_{R^*}(s,a)|$*, with* $Q^\pi_{R^*}$ *denoting the true action-value function of* π *under* R^**.*

Proof. Under exact dynamics and expectations,

$$\hat{J}(\pi) - J_{R^*}(\pi) = \mathbb{E}_{\rho,\pi}\left[\sum_{t=0}^{n-1} \gamma^t \big(R_\psi(s_t,a_t) - R^*(s_t,a_t)\big) \;+\; \gamma^n \big(Q_\theta(s_n,a_n) - Q^\pi_{R^*}(s_n,a_n)\big)\right].$$

Taking absolute values and applying the triangle inequality,

$$|\hat{J}(\pi) - J_{R^*}(\pi)| \;\leq\; \sum_{t=0}^{n-1} \gamma^t\,\epsilon_R + \gamma^n\,\epsilon_Q \;=\; \frac{1-\gamma^n}{1-\gamma}\,\epsilon_R + \gamma^n\,\epsilon_Q. \qquad \square$$

Substituting into Proposition 3.2, MCVL rejects a hacking-inducing transition whenever the true return drop $\delta = J_{R^*}(\tilde{\pi}^0) - J_{R^*}(\tilde{\pi}^+)$ satisfies

$$\delta \;>\; 2\left(\frac{1-\gamma^n}{1-\gamma}\,\epsilon_R + \gamma^n\,\epsilon_Q\right). \tag{3}$$

This highlights the role of the scoring rollout steps n: each additional step replaces reliance on the bootstrap Q_θ with a direct reward-model observation. The bound $g(n) = \frac{1-\gamma^n}{1-\gamma}\,\epsilon_R + \gamma^n\,\epsilon_Q$ is monotonically decreasing in n when $\epsilon_Q > \frac{\epsilon_R}{1-\gamma}$, which we expect in practice since R_ψ directly observes the forecasted policy's behavior during rollouts while Q_θ was not trained on the forecasted policy. When $\hat{P}$ is a learned model, compounding prediction error provides a practical upper bound on useful n; in episodic tasks with a simulator, the episode length is a natural choice. Successful filtering keeps the buffer free of hacking transitions, helping maintain small ϵ_R and ϵ_Q over time.

Why forecasting improves detection. MCVL scores each branch with k rollouts of length n under the frozen evaluator. Because evaluator parameters are frozen and n is fixed, the variance of the per-rollout bootstrapped return is bounded by some σ^2 independently of the forecast budget l. Using paired rollouts with shared start states exploits positive covariance between the two branch estimates, so the variance of the estimator $\hat{J}_k(\tilde{\pi}^0) - \hat{J}_k(\tilde{\pi}^+)$ is at most $2\sigma^2/k$, independent of l.

The expected scoring gap $\Delta(l) = \hat{J}(\tilde{\pi}_l^0) - \hat{J}(\tilde{\pi}_l^+)$ (positive when the hacking branch scores lower) in general grows with l: each additional forecast step gives the base learner more room to translate the hacking transition into a policy change the evaluator can distinguish. So increasing the forecast budget improves detection without increasing scoring cost.

At $l = 1$ (the *Each-step* variant), a single gradient step barely changes the policy, especially before the critic has begun to assign high value to the hacking behavior, so $\Delta(1) \approx 0$ and the check is unreliable. Increasing l to the point where the hacking transition induces a meaningful policy divergence produces a gap the evaluator can reliably detect, explaining why the Each-step variant fails while moderate l succeeds (Figure 5a).

B Feasibility of Occupancy-Regularized Objectives

It would be trivial to show that regularizing to a safe policy either performs at the same level as the frozen safe policy (or reward hacks) by selecting a high (or low) regularization coefficient. Instead, we test whether an ORPO-style objective presented by (Laidlaw et al., 2025) could, *in principle*, select the desired behavior in our settings. For each gridworld environment we train DDQN Q-functions for *Frozen* (safe, post-pretraining), *Hacking* (trained on observed reward), and *Oracle* (trained on true reward). From these Q-functions, we derive stochastic policies via (i) softmax over Q-values and (ii) ϵ-greedy with $\epsilon = 0.05$. We estimate occupancy measures and empirical on-policy discounted episodic returns under observed reward J_π with 1000 rollouts. We compute ORPO objective $F(\pi, \pi_{\text{Frozen}}) = J(\pi) - \lambda D(\mu_\pi \| \mu_{\pi_{\text{Frozen}}})$ for $D \in \{\text{KL}, \sqrt{\chi^2}\}$. We record the fraction of seeds (out of 10) where some $\lambda > 0$ exists such that it satisfies *both* $F(\pi_{\text{Oracle}}, \pi_{\text{Frozen}}) > F(\pi_{\text{Frozen}}, \pi_{\text{Frozen}})$ and $F(\pi_{\text{Oracle}}, \pi_{\text{Frozen}}) > F(\pi_{\text{Hacking}}, \pi_{\text{Frozen}})$. Let J_O, J_F, J_H denote the observed returns of Oracle, Frozen, and Hacking policies; let $D_O = D(\mu_O \| \mu_F)$ and $D_H = D(\mu_H \| \mu_F)$. The first inequality gives $\lambda < \frac{J_O - J_F}{D_O}$ when $D_O > 0$ (and requires $J_O > J_F$ when $D_O = 0$). The second inequality is $\lambda(D_H - D_O) > J_H - J_O$, which yields three cases: if $D_H > D_O$, then $\lambda > \frac{J_H - J_O}{D_H - D_O}$; if $D_H < D_O$, then $\lambda < \frac{J_H - J_O}{D_H - D_O}$; if $D_H = D_O$, it requires $J_O > J_H$. A seed is feasible iff the resulting open interval intersects $(0, +\infty)$. Per-seed feasibility does not imply a single global λ across seeds. We present results in Table 1.

Policy	Divergence	Box Moving	Absent Supervisor	Tomato Watering	Rocks & Diamonds
Soft-Q	$\sqrt{\chi^2}$	0%	100%	0%	0%
Soft-Q	KL	0%	100%	0%	0%
ϵ-greedy	$\sqrt{\chi^2}$	60%	40%	30%	0%
ϵ-greedy	KL	40%	50%	0%	0%

Table 1: Percentage of seeds (of 10) where a regularization weight $\lambda > 0$ exists that ranks the Oracle policy above both Frozen and Hacking under an ORPO-like objective.

In many cases, no such λ exists, suggesting that occupancy regularization fails to suppress high-value hacks without also suppressing Oracle-like improvements. In contrast, MCVL achieves performance comparable to the Oracle across all these tasks.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *ArXiv preprint*, 2016.

Carson Denison, Monte MacDiarmid, Fazl Barez, David Duvenaud, Shauna Kravec, Samuel Marks, Nicholas Schiefer, Ryan Soklaski, Alex Tamkin, Jared Kaplan, et al. Sycophancy to subterfuge: Investigating reward-tampering in large language models. *ArXiv preprint*, 2024.

Tom Everitt and Marcus Hutter. Avoiding wireheading with value reinforcement learning. In *Artificial General Intelligence*, 2016.

Tom Everitt, Daniel Filan, Mayank Daswani, and Marcus Hutter. Self-modification of policy and utility function in rational agents. In *Artificial General Intelligence*, 2016.

Tom Everitt, Marcus Hutter, Ramana Kumar, and Victoria Krakovna. Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective. *Synthese*, (Suppl 27), 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, Proceedings of Machine Learning Research, 2018.

Florin-Cristian Ghesu, Bogdan Georgescu, Yefeng Zheng, Sasa Grbic, Andreas Maier, Joachim Hornegger, and Dorin Comaniciu. Multi-scale deep reinforcement learning for real-time 3d-landmark detection in ct scans. *IEEE transactions on pattern analysis and machine intelligence*, (1), 2017.

Bill Hibbard. Model-based utility functions. *Journal of Artificial General Intelligence*, (1), 2012.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, (274), 2022.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, (6), 2021.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of AI ingenuity. *DeepMind Blog*, 2020.

Ramana Kumar, Jonathan Uesato, Richard Ngo, Tom Everitt, Victoria Krakovna, and Shane Legg. REALab: An embedded perspective on tampering. *ArXiv preprint*, 2020.

Cassidy Laidlaw, Shivam Singhal, and Anca Dragan. Correlated proxies: A new definition and improved mitigation for reward hacking. In *ICLR*, 2025.

Jan Leike, Miljan Martic, Victoria Krakovna, Pedro A Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg. AI safety gridworlds. *ArXiv preprint*, 2017.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *ArXiv preprint*, 2018.

Zixuan Liu, Xiaolin Sun, and Zizhan Zheng. Robust optimization for mitigating reward hacking with correlated proxies. In *ICLR*, 2026.

Monte MacDiarmid, Benjamin Wright, Jonathan Uesato, Joe Benton, Jon Kutasov, Sara Price, Naia Bouscal, Sam Bowman, Trenton Bricken, Alex Cloud, Carson Denison, Johannes Gasteiger, Ryan Greenblatt, Jan Leike, Jack Lindsey, Vlad Mikulik, Ethan Perez, Alex Rodrigues, Drake Thomas, Albert Webson, Daniel Ziegler, and Evan Hubinger. Natural emergent misalignment from reward hacking in production RL. *ArXiv preprint*, 2025.

Arushi Majha, Sayan Sarkar, and Davide Zagami. Categorizing wireheading in partially embedded agents. *ArXiv preprint*, 2019.

A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

OpenAI. Faulty reward functions. https://openai.com/research/faulty-reward-functions, 2023. Accessed: 2024-04-10.

OpenAI. Openai o1 system card. https://openai.com/index/openai-o1-system-card/, 2024. Accessed: 2024-09-26.

Laurent Orseau and Mark Ring. Self-modification and mortality in artificial agents. In *Artificial General Intelligence*, 2011.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *ICLR*, 2022.

Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *ArXiv preprint*, 2017.

Jürgen Schmidhuber. Gödel machines: self-referential universal problem solvers making provably optimal self-improvements. *arXiv preprint cs/0309048*, 2003.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *NeurIPS*, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

Jessica Taylor, Eliezer Yudkowsky, Patrick LaVictoire, and Andrew Critch. Alignment for advanced machine learning systems. *Ethics of Artificial Intelligence*, 2016.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *ArXiv preprint*, 2024.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

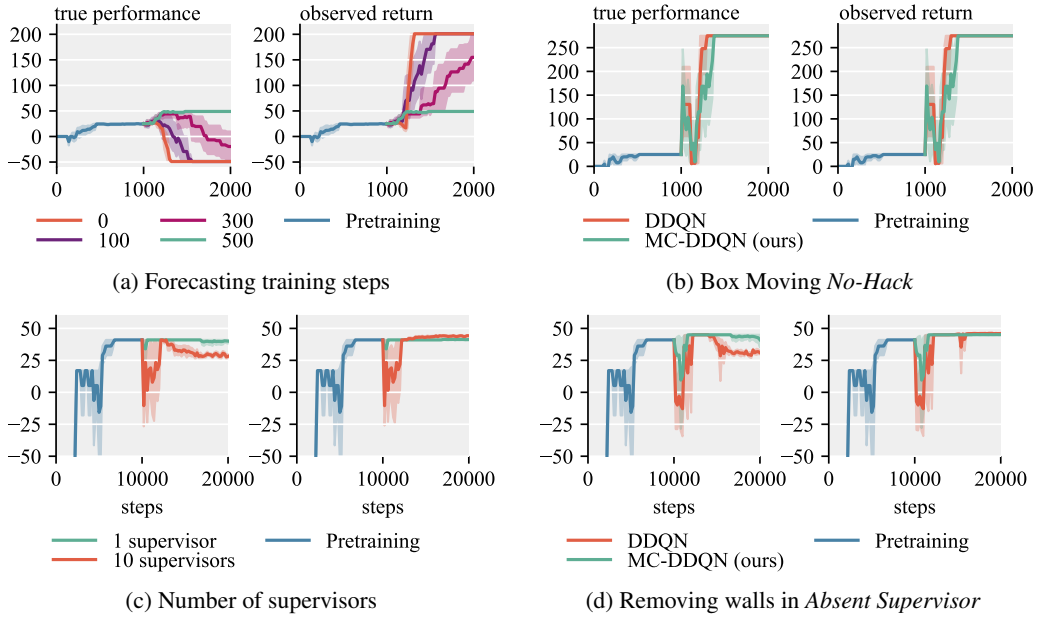
Roman V Yampolskiy. Utility function security in artificially intelligent agents. *Journal of Experimental & Theoretical Artificial Intelligence*, (3), 2014.

Eliezer Yudkowsky. Complex value systems in friendly ai. In *Artificial General Intelligence: 4th International Conference, AGI 2011, Mountain View, CA, USA, August 3-6, 2011. Proceedings 4*. Springer, 2011.

Supplementary Materials

The following content was not necessarily subject to peer review.

C Additional Experiments



(a) Forecasting training steps

(b) Box Moving *No-Hack*

(c) Number of supervisors

(d) Removing walls in *Absent Supervisor*

Figure 5: (a) Sensitivity to forecasting training steps l in Box Moving. (b) Results in the *No-Hack* version of Box Moving. (c) Varying the number of supervisors in Absent Supervisor. (d) A variant of Absent Supervisor where a shorter path becomes available in *Full*.

In Figure 5a, we investigated the number of forecasting training steps l needed to avoid undesired behavior in Box Moving. With an insufficient number of training steps, certain undesired transitions are not rejected, yet our algorithm still slows down the learning of reward hacking behavior.

In Figure 5b, we examine the behavior of MC-DDQN in the *No-Hack* version of *Box Moving* (Figure 1). In this version, the agent receives a +5 reward on the top cell which does not interfere with moving the box upward. As anticipated, in this scenario our agent does not reject transitions and learns the optimal policy.

We also conducted experiments in *Absent Supervisor*, varying the number of supervisors. In Figure 5c, increasing the number of supervisors from 1 to 10 leads to less consistent detection of transitions that induce reward hacking, despite the change being purely visual. Qualitative analysis revealed that our neural networks struggled to adapt to this distribution shift (all gridworld environments are encoded as multi-hot vectors), resulting in predicted rewards deviating significantly from the ground truth.

Furthermore, we explored the impact of removing two walls from *Absent Supervisor* after training in *Safe*. Without these two walls, a shorter path to the goal is available that bypasses the punishment cell, although going through the punishment cell remains faster. In Figure 5d, it is evident that while our algorithm can learn a better policy that avoids the punishment cell, the rejection of reward hacking transitions becomes less reliable. This decline is attributed to the increased distribution shift between *Safe* and *Full*.

D Learned Transition Model

Scoring uses a transition model $\hat{P}$ solely to *compare* two candidate policies under a frozen evaluator; exact dynamics are unnecessary as long as the evaluator continues to rank hacking trajectories below non-hacking ones. To verify that a learned model suffices, we train a forward dynamics model and use it in place of the environment during scoring rollouts. The model is a two-hidden-layer MLP with 128 units per layer, ReLU activations, and layer normalization. It takes the concatenation of the current observation and a one-hot encoded action as input and predicts the next observation. The model is trained with MSE loss using Adam (learning rate 10^{-2}) on 50 episodes of random exploration data (1000 gradient steps, batch size 256) and frozen during deployment. We run MC-DDQN in *Box Moving* with the same hyperparameters as in the main experiments, replacing only the transition source. As shown in Figure 6, MC-DDQN with the learned $\hat{P}$ avoids reward hacking and reaches Oracle performance, matching the result obtained with the true environment. This supports the claim that approximate dynamics suffice for reliable gating.

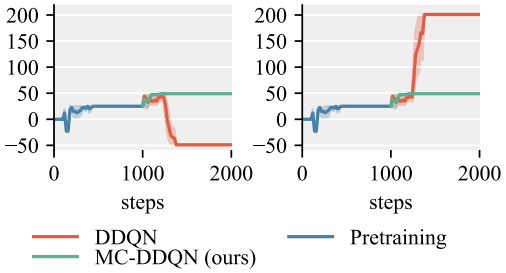


Figure 6: MC-DDQN with a learned transition model in *Box Moving*.

E Implementation Details of MC-DDQN

Algorithm 2 Policy Forecasting

Input: Set of transitions T, replay buffer D, current Q-network parameters θ, training steps l
Output: Forecasted policy π_f

$\theta_f \leftarrow \textsc{Copy}(\theta)$ ▷ Copy current Q-network parameters
for training step $t = 1$ to l **do**
  Sample random mini-batch B of transitions from D
  $\theta_f \leftarrow \textsc{TrainDDQN}(\theta_f, B \cup T)$ ▷ T is added to each batch for deterministic environments
end for
return $\pi_f(s) = \arg\max_a Q_{\theta_f}(s, a)$ ▷ Return forecasted policy

Algorithm 3 Scoring

Input: Policy π, transition model $\hat{P}$, return estimator parameters θ and ψ, initial states ρ, rollout steps n, number of rollouts k
Output: Estimated bootstrapped return of the policy π

for rollout $r = 1$ to k **do**
  $g \leftarrow 0$ ▷ Initialize return for this rollout
  $s_0 \sim \rho$ ▷ Sample an initial state
  $a_0 \leftarrow \pi(s_0)$ ▷ Get action from policy
  for step $t = 0$ to $n - 1$ **do**
    $g \leftarrow g + \gamma^t R_\psi(s_t, a_t)$ ▷ Accumulate predicted reward
    $s_{t+1} \sim \hat{P}(s_t, a_t)$ ▷ Sample next state from transition model
    $a_{t+1} \leftarrow \pi(s_{t+1})$ ▷ Get action from policy
  end for
  $g \leftarrow g + \gamma^n Q_\theta(s_n, a_n)$ ▷ Add final Q-value
end for
return $\frac{1}{k} \sum_{r=1}^{k} g$ ▷ Return average return over rollouts

Algorithm 4 Modification-Considering Double Deep Q-learning (MC-DDQN)

Input: Pretrained return estimator parameters θ and ψ, replay buffer D, transition model $\hat{P}$, initial states ρ, rollout steps n, number of rollouts k, forecasting train steps l, threshold δ_r.
Output: Trained Q-network and reward model

Initialize state s_0
for time step $t = 0$ to end of training **do**
  $a_t \leftarrow \epsilon$-GREEDY($\arg\max_a Q_\theta(s_t, a)$)
  Execute action a_t, observe reward r_t, and transition to state s_{t+1}
  $T_t \leftarrow (s_t, a_t, r_t, s_{t+1})$
  if $|r_t - R_\psi(s_t, a_t)| < \delta_r$ **then**
    $accept \leftarrow$ **True**
  else
    $\tilde{\pi}^+ \leftarrow$ FORECAST($\{T_t\}, D, \theta, l$) ▷ Forecast a policy with new transition
    $\tilde{\pi}^0 \leftarrow$ FORECAST($\{\}, D, \theta, l$) ▷ Forecast a policy without new transition
    $J_{\tilde{\pi}^+} \leftarrow$ SCORE($\tilde{\pi}^+, \hat{P}, \theta, \psi, \rho, n, k$) ▷ Estimate n-step bootstrapped return for $\tilde{\pi}^+$
    $J_{\tilde{\pi}^0} \leftarrow$ SCORE($\tilde{\pi}^0, \hat{P}, \theta, \psi, \rho, n, k$) ▷ Estimate n-step bootstrapped return for $\tilde{\pi}^0$
    $accept \leftarrow (J_{\tilde{\pi}^+} \geq J_{\tilde{\pi}^0})$ ▷ Accept if $\tilde{\pi}^+$ is not worse by current estimator
  end if
  if $accept$ **then**
    Store transition T_t in D
  else
    Reset the environment ▷ Without T_t in D future forcasted policies might fail to learn.
  end if
  Sample random mini-batch B of transitions from D
  $\theta \leftarrow$ TRAINDDQN(θ, B) ▷ Update Q-network
  $\psi \leftarrow$ TRAIN(ψ, B) ▷ Update reward model using L_2 loss
  $s_t \leftarrow s_{t+1}$
end for

F Implementation Details of MC-TD3

Our implementation is based on the implementation provided by Huang et al. (2022). The overall structure of the algorithm is consistent with MC-DDQN, described in Appendix E, with key differences outlined below. TD3 is an actor-critic algorithm, meaning that the parameters θ define both a policy (actor) and a Q-function (critic). In Algorithm 2 and Algorithm 4, calls to TRAINDDQN are replaced with TRAINTD3, which updates the actor and critic parameters θ as specified by Fujimoto et al. (2018). Additionally, in Algorithm 2, the returned policy $\pi_f(s)$ corresponds to the actor rather than $\arg\max_a Q_\theta(s, a)$, and in Algorithm 4 the action executed in the environment is also selected by the actor.

Forecast implementation detail. In Algorithm 2, the transition set T is added to each forecast minibatch ($B \cup T$) rather than relying on stochastic replay sampling from $D \cup T$. This is an implementation choice to reduce variance in the branch comparison by ensuring the checked transition is represented in the "with-T" branch under a fixed update budget. The theoretical analysis is unchanged, because it only assumes an abstract $\mathsf{Forecast}(\cdot, l)$ operator.

G Details of the Continuous-Control Experiments

All three continuous-control environments follow the standard procedure where the policy and critics are first trained on transitions collected by a random policy. No *Safe* variant is required because random exploration is unlikely to simultaneously satisfy the velocity and physical-configuration conditions that trigger hacking or keep the arm in place for multiple frames in Reacher. This

demonstrates that MCVL can function effectively without a separate safe environment whenever the hacking affordance is difficult to discover by chance.

Reacher. The rewards in the original Reacher-v5 environment are calculated as the sum of the negative distance to the target and the negative joint actuation strength. This reward structure encourages the robotic arm to reach the target while minimizing large, energy-intensive actions. The target's position is randomized at the start of each episode, and random noise is added to the joint rotations and velocities. Observations include the angles and angular velocities of each joint, the target's coordinates, and the difference between the target's coordinates and the coordinates of the arm's end. Actions consist of torques applied to the joints, and each episode is truncated after 50 steps.

We modified the environment by introducing a +50 reward when the arm's end remains within a small, fixed region for 15 consecutive steps. This region remains unchanged across episodes, simulating a scenario where the robot can tamper with its reward function by holding a button, but such behavior is difficult to discover. In our setup, a reward-tampering policy is highly unlikely to emerge through random actions and is typically discovered only when the target happens to be inside the reward-tampering region.

In our experiments, TD3 consistently discovers the hack when the goal in one of the episodes falls within the hacking region. MC-TD3 successfully filters out such transitions.

Ant. The standard Ant-v5 environment rewards forward movement (velocity in the x-direction), a healthy-posture bonus, and penalizes control effort and contact forces. We augment it with a sensor-malfunction model motivated by real-world IMU failures in legged robots: when the total contact-force norm exceeds a threshold (400) and the agent's forward velocity is above a minimum (3.5), the reward is augmented with a bonus equal to 10 times the contact force. This models the self-reinforcing loop in which aggressive gaits produce high ground-reaction forces that saturate accelerometers, causing inflated velocity readings that feed back into the reward. The performance metric is the reward *before* the sensor-corruption bonus is applied. Random exploration is unlikely to simultaneously achieve the required high velocity and high contact forces.

In our experiments TD3 learner with this bonus reward achieved lower evaluation return compared to Oracle that did not receive the bonus reward (Figure 3f). We hypothesize that pursuing transitions that achieved high contact forces in the past is brittle and doesn't lead to stable policies.

HalfCheetah. The standard HalfCheetah-v5 environment rewards forward velocity and penalizes control effort. We augment it with an optical-flow sensor failure model: when the front tip of the cheetah drops below a height threshold (-0.23) while the forward velocity exceeds a minimum (6), 1 reward bonus is added for each millimeter of the tip's negative height. This models the velocity over-estimation that occurs when a downward-facing optical-flow sensor operates too close to the ground, where proximity saturation and height-division error compound. The performance metric excludes this bonus. Random exploration rarely produces the combination of high forward speed and low front-tip height needed to trigger the hack.

In HalfCheetah, MC-TD3's mean falls slightly below the Oracle though within the 95% CI, likely because of occasional over-rejection of legitimate transitions or due to environment resets when hacking is discovered by exploration.

H Frozen Reward Model Baseline

To isolate the contribution of the forecast-and-score gate from that of merely having a learned reward model, we evaluate a *Freeze RM* baseline. Unlike the *Frozen* baseline of Figure 3, which stops policy and critic learning after pretraining, here only the reward model is frozen: the policy and critic continue to train, but bootstrap on the frozen pretrained reward model R_ψ instead of the observed proxy reward, and *every* transition is admitted (no gate). The reward model is frozen at the end of

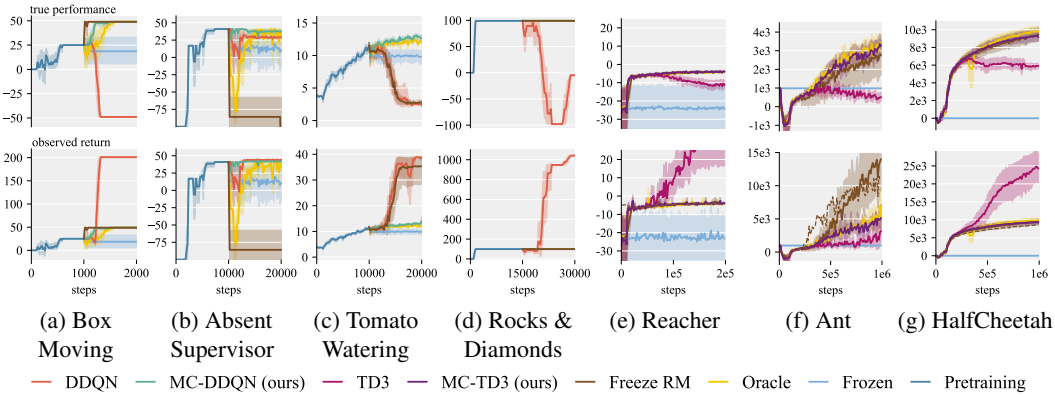


Figure 7: **Freeze RM baseline across all environments.** Top: true performance (unobserved objective); bottom: observed return (proxy). Same format as Figure 3, with the Freeze RM baseline (brown) added. The dashed brown line in the continuous-control panels is the reward model's own predicted return for Freeze RM (the quantity it maximizes). Bold lines: mean over 10 seeds; bands: bootstrapped 95% CI.

pretraining (for continuous control, at the switch from random exploration to the learned policy). All other components (pretrained initialization, replay buffer, and hyperparameters) are identical to MCVL (MC-DDQN in the gridworlds, MC-TD3 in continuous control). Freezing R_ψ is necessary because, without a gate, a live reward model would simply fit the hacking bonus and the baseline would collapse to the base learner.

Figure 7 reports Freeze RM against MCVL and the Oracle over 10 seeds per environment. On the *true* objective, Freeze RM matches the Oracle in five of the seven tasks (Box Moving, Rocks & Diamonds, Reacher, Ant, and HalfCheetah): freezing the reward model immediately after pretraining is, perhaps surprisingly, enough to recover Oracle-level performance. The two exceptions show that the gate can offer an advantage in some environments. In *Absent Supervisor*, Freeze RM's true performance collapses far below the Oracle (paired t-test vs. MC-DDQN, $p < 10^{-4}$; $9/10$ seeds fall below the worst Oracle seed), whereas MC-DDQN stays at Oracle level. The observed return also stays low, which indicates that the reward model is not accurately capturing the true reward. In *Tomato Watering*, Freeze RM exhibits the classic hacking signature, with the observed return inflating well above the achievable true return while true performance drops; however, MCVL's mitigation in this environment relies on a non-delusional transition model used by the gate, and since Freeze RM has no gate, the comparison conflates the gate with the world model and cannot be made fair.

Within continuous control, we observe a high proxy return for Freeze RM in the Ant environment. It is significantly higher than MC-TD3's (paired t-test, $p < 0.01$; $\approx 2.7\times$), and its reward-model-predicted return (dashed) tracks the inflated proxy rather than the true performance. In the single seed where this is most severe, the policy exploits errors in the frozen R_ψ, driving the predicted return far above any achievable true return while true performance collapses; because only one seed is affected, the effect on mean true performance is small and not significant. Freeze RM is also somewhat more failure-prone than MCVL, with more seeds whose converged true performance falls below the worst Oracle seed (Ant 3 vs. 2, HalfCheetah 3 vs. 1).

Overall, the gate's necessity is environment-dependent. Where a reward model fit on the pretraining data already captures the true reward, bootstrapping on it without a gate suffices; where it does not, as in *Absent Supervisor*, removing the gate leads to reward hacking and collapse. We therefore expect the gate's contribution to grow in settings where a small pretraining set cannot capture the true reward.

I Qualitative Observations

During preliminary experiments, we encountered instances where the algorithm failed to reject transitions that induce reward hacking. Here we describe these occurrences and how they can be addressed.

Return estimation rollout steps. When using much smaller rollout steps n, we noticed that during evaluation of forecasted trajectories, the non-hacking policy sometimes needed to traverse several states with low rewards to reach a high-reward region. In such cases, the reward hacking policy, which remained stationary, had a higher estimated utility. Increasing n resolved this issue.

Forecasting without a counterfactual. Initially, we forecasted only one future policy by training with the checked transition added to each mini-batch, and compared the resulting policy to the current one. However, in some cases this led to situations where the copy learned better non-hacking behaviors than the current policy simply because it was trained for longer. The solution was to forecast two policies, one with the checked transition added to each mini-batch and one without.

Sensitivity to stochasticity. Evaluations in stochastic environments were noisy. To mitigate this, we compared the two policies starting from the same set of states and using the same random seeds of the transition model. We also kept the random seeds fixed while sampling mini-batches.

Handling rejected transitions. We observed that if a hacking-inducing transition was removed from the replay buffer and another such transition occurred in the same episode, the algorithm sometimes failed to detect it the second time because there was no set of transitions in the buffer connecting this second transition to the starting state. We therefore reset the environment on every rejection, as shown in Algorithm 1. In practical applications, it would be reasonable to assume that after detecting potential reward hacking, the agent would be returned to a safe state instead of continuing exploration. Alternatively, the learning can be just disabled until the end of the episode.

Irreversible changes. In *Rocks and Diamonds*, when comparing policies starting from the current state after the rock was pushed into the goal area, the comparison results were always the same, as it was impossible to move the rock out of the goal area. We addressed this by evaluating from the initial state of the environment. In cases where reset is not possible, the agent may store starting states in a buffer. This issue underscores the importance of future research into avoiding irreversible changes.

J Computational Requirements

All gridworld experiments were conducted on workstations equipped with Intel® Core™i9-13900K processors and NVIDIA® GeForce RTX™4090 GPUs. Continuous control experiments were conducted on NVIDIA® H100 MIG partitions (1/8 GPU, 8 vCPU cores). All experiments in the *Absent Supervisor* and *Tomato Watering* environments each required 12-14 GPU-hours, running 10 seeds in parallel. In *Rocks and Diamonds*, experiments took 1 GPU-day, while in *Box Moving* they required 2 hours each. For the continuous-control environments (1M training steps for Ant and HalfCheetah, 200k for Reacher), average per-seed wall-clock times on an H100 MIG partition were: Reacher TD3 75 min / MC-TD3 135 min ($1.8\times$), Ant TD3 68 min / MC-TD3 206 min ($3.0\times$), HalfCheetah TD3 65 min / MC-TD3 272 min ($4.2\times$). The variation in overhead across environments depends primarily on how frequently the reward-discrepancy check is triggered. In total, the main experiments described in Section 4 required approximately 20 GPU-days, including around 6 GPU-days for baselines.

K Hyperparameters of MC-DDQN

All hyperparameters are listed in Table 2. Our algorithm introduces several additional hyperparameters beyond those typically used by standard RL algorithms:

Table 2: Hyperparameters used for the experiments.

Hyperparameter Name	**Value**
Q_θ and R_ψ hidden layers	2
Q_θ and R_ψ hidden layer size	128
Q_θ and R_ψ activation function	ReLU
Q_θ and R_ψ optimizer	Adam
Q_θ learning rate	0.0001
R_ψ learning rate	0.01
Q_θ loss	SmoothL1
R_ψ loss	L_2
Batch size	32
Discount factor γ	0.95
Training steps on *Safe*	10000
Training steps on *Full*	10000
Replay buffer size	10000
Exploration steps	1000
Exploration ϵ_{start}	1.0
Exploration ϵ_{end}	0.05
Target network EMA coefficient	0.005
Forecasting training steps l	5000
Scoring rollout steps n	30
Number of scoring rollouts k	20
Predicted reward difference threshold δ_r	0.05
Add transitions from transition model	False

Reward model architecture and learning rate. Hyperparameters specify the architecture and learning rate of the reward model R_ψ. Since learning a reward model is a supervised learning task, these hyperparameters can be tuned on a dataset of transitions collected by any policy. The reward model architecture may be chosen to match the Q-function Q_θ.

Forecasting training steps l. This parameter describes the number of updates to the Q-function needed to predict the future policy based on a new transition. As shown in Figure 5a, this value must be sufficiently large to update the learned values and corresponding policy. It can be selected by artificially adding a transition that alters the optimal policy and observing the number of training steps required to learn the new policy.

Scoring rollout steps n. This parameter controls the length of the trajectories used to compare two forecasted policies. Longer trajectories provide more direct behavioral information about each policy and reduce reliance on the value-function bootstrap. The error decomposition in Appendix A confirms that, under expected conditions ($\epsilon_Q > \frac{\epsilon_R}{1-\gamma}$), the evaluator bound is monotonically decreasing in n, favoring longer rollouts. In episodic tasks with access to a simulator, a safe choice is the maximum episode length; in continuing tasks, a truncation horizon typically used in training may be suitable. When using a learned transition model, compounding prediction error provides a practical upper bound on useful n. Computational costs scale linearly in n and can be reduced by choosing a smaller value based on domain knowledge.

Number of scoring rollouts k. This parameter specifies the number of trajectories obtained by rolling out each forecasted policy for comparison. The required number depends on the stochasticity of the environment and policies. If both the policy and environment are deterministic, k can be set to 1. Otherwise, k can be selected using domain knowledge or by measuring how much rollouts are required to distinguish between multiple known policies with different behaviors.

Predicted reward difference threshold δ_r. This threshold defines the minimum difference between the predicted and observed rewards for a transition to trigger a check. As discussed in Section 4.3, this parameter is introduced only to reduce computations and can be set to 0. However, it can be adjusted based on domain knowledge to speed up training by minimizing unnecessary checks. The key requirement is that any reward hacking behavior must increase the reward by more than this threshold relative to the reward predicted by the reward model. In our gridworld and Reacher experiments, 0.05 performed well when rewards were normalized to $[-1, 1]$. For Ant and HalfCheetah, where the reward scale is larger, we use δ_r=2.0.

K.1 Environment-specific Parameters

The training steps in *Box Moving* were reduced to speed up training. *Tomato Watering* has many stochastic transitions because each tomato has a chance of drying out at each step. To increase the robustness of evaluations, we increased the number of scoring rollouts k. *Rocks and Diamonds* required more steps to converge to the optimal policy. Additionally, using the transition model to collect fresh data while forecasting in *Rocks and Diamonds* makes reward hacking detection more reliable. Each gridworld environment's rewards were scaled to $[-1, 1]$.

Table 3: Environment-specific hyperparameter overrides.

Hyperparameter Name	**Value**
Box Moving	
Training steps on *Safe*	1000
Training steps on *Full*	1000
Replay buffer size	1000
Exploration steps	100
Forecasting training steps l	500
Absent Supervisor	
Number of supervisors	1
Remove walls	False
Tomato Watering	
Number of scoring rollouts k	100
Rocks and Diamonds	
Training steps on *Safe*	15000
Training steps on *Full*	15000
Forecasting training steps l	7500
Add transitions from transition model	True

K.2 Hyperparameters of MC-TD3

Table 4: Hyperparameters used for the MC-TD3 experiments (Reacher, Ant, HalfCheetah).

Hyperparameter Name	**Value**
Actor, critic, and reward model hidden layers	2
Actor, critic, and reward model hidden layer size	256
Actor, critic, and reward model activation function	ReLU
Actor, critic, and reward model optimizer	Adam
Actor and critic learning rate	0.0003
R_ψ learning rate	0.003
Batch size	256
Discount factor γ	0.99
Training steps	200000
Replay buffer size	200000
Exploration steps	30000
Target networks EMA coefficient	0.005
Policy noise	0.01
Exploration noise	0.1
Policy update frequency	2
Forecasting training steps l	10000
Scoring rollout steps n	50
Number of scoring rollouts k	100
Predicted reward difference threshold δ_r	0.05

We did not perform extensive hyperparameter tuning; most hyperparameters are inherited from the implementation provided by Huang et al. (2022). The same MC-TD3 hyperparameters are used for all three continuous-control environments (Reacher, Ant, HalfCheetah), with the following environment-specific overrides:

Table 5: Environment-specific hyperparameter overrides for MC-TD3.

Hyperparameter Name	**Value**
Ant & HalfCheetah	
Training steps	1000000
Replay buffer size	1000000
Exploration steps	100000
Scoring rollout steps n	1000
Predicted reward difference threshold δ_r	2.0

Ant and HalfCheetah require more training steps (1M vs. 200k) because TD3 does not converge within 200k steps in these environments. The exploration budget is scaled proportionally. The reward difference threshold δ_r is increased from 0.05 to 2.0 to match the larger reward scale. The scoring rollout steps n is set to the full episode length (1000).