TRACE-RL: TRAJECTORY-GUIDED REWARD ADAPTATION AND COLLABORATIVE EVOLUTION FOR ROBOTIC SKILL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning robotic skills through reinforcement learning offers a promising path toward embodied intelligence, yet designing rewards that continually and reliably guide high-dimensional robotic control remains difficult. Large language models can reduce this burden by generating executable dense reward programs. Existing LLM-based methods, however, generally revise reward guidance at outer-loop boundaries while keeping each candidate reward fixed during the corresponding policy-training cycle. This design becomes restrictive as an improving policy reaches new contact configurations and near-success behaviors that a fixed reward may no longer guide adequately. We introduce **T**rajectory-Guided **R**eward **A**daptation and **C**ollaborative **E**volution for **R**obotic Skill **L**earning (**TRACE-RL**), a framework that maintains effective reward guidance as the policy-induced trajectory distribution evolves. In the inner loop, TRACE-RL combines an LLM-generated task-prior reward with a trajectory-calibrated reward learned using an executable trajectory-scoring function, and uses an evaluation function to support reliability-aware fusion. In the outer loop, the LLM jointly refines the three executable reward, score, and evaluation functions using structured policy-training feedback, while validated reward-model and policy states are retained across successive outer iterations. Across eighteen robotic tasks spanning locomotion, whole-body control, and dexterous manipulation, TRACE-RL achieves the best or tied-best mean maximum training score on every task, with ablation studies supporting the effectiveness of its key components.

1 INTRODUCTION

Reinforcement learning has achieved strong performance in locomotion and dexterous manipulation, yet its success depends critically on reward signals that remain effective throughout policy learning. Designing such rewards is difficult: sparse task rewards provide little guidance before successful behavior is observed (Andrychowicz et al., 2017; Pathak et al., 2017), whereas manually engineered dense rewards require substantial expertise in robot dynamics, contact geometry, and task progress. Even carefully designed rewards can contain misaligned terms, induce unintended shortcuts, or become brittle when the policy visits state regimes not anticipated during reward design (Ng et al., 1999; Hadfield-Menell et al., 2017; Amodei et al., 2016; Booth et al., 2023; Knox et al., 2023).

Recent large language models offer a promising way to reduce this burden (Kwon et al., 2023). Given task descriptions and environment code, LLMs can generate executable programmatic reward functions that encode distances, postures, contacts, and task-progress terms (Srivastava et al., 2026). Eureka (Ma et al., 2024) shows that reward code can be iteratively refined using policy-training feedback, while subsequent methods extend this process through reward-policy co-evolution (Huang et al., 2025), structured reward search (Hazra et al., 2025; Gao et al., 2025), and task-relevant exploration (Chang et al., 2026). These advances make dense reward design more accessible by reducing reliance on manual engineering, but they largely retain a stage-wise abstraction: a programmatic reward function is proposed, a policy is optimized under it, and the resulting task score is used to revise the reward function only in a later outer iteration.

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

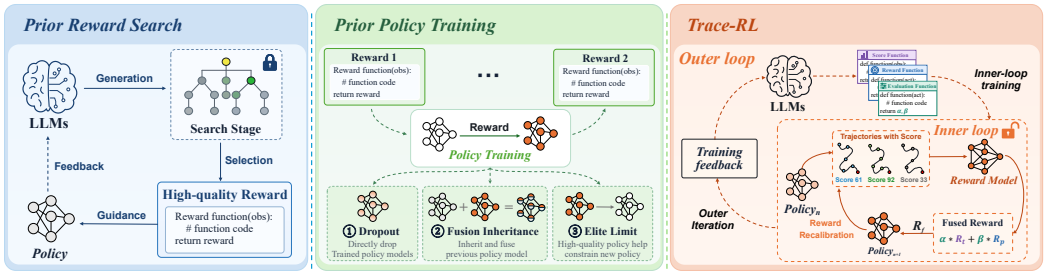

Figure 1: Comparison of prior LLM-based reward search and policy-transfer schemes with TRACE-RL, which recalibrates reward guidance from current-policy trajectories using supervision from executable score functions and retains reward-model and policy states across iterations.

Existing methods primarily improve the outer-loop search for programmatic rewards. Eureka revises reward programs using policy-training feedback (Ma et al., 2024); REvolve diversifies reward search through population-based evolution, while RF-Agent organizes candidate generation and refinement through language-agent tree search (Hazra et al., 2025; Gao et al., 2025). R* separates reward-structure evolution from parameter alignment (Li et al., 2025a). ORSO integrates reward selection with policy optimization, whereas LaRes combines adaptive reward search, experience relabeling, and constrained policy transfer (Zhang et al., 2024; Li et al., 2025b). Other methods explore reward–policy co-evolution and task-relevant exploration (Huang et al., 2025; Chang et al., 2026). These approaches remain primarily program-centric: they adapt reward guidance by selecting or revising reward programs rather than recalibrating rewards from newly collected trajectories during policy training, which may limit adaptation to newly encountered states and behaviors. Policy-level task scores summarize candidate performance but reveal little about which contact transitions or near-success behaviors remain insufficiently distinguished by the current reward function. A reward program that separates coarse failures early in training may thus become less informative as the policy reaches finer-grained behavioral regions; continued optimization of an imperfect fixed reward proxy can degrade performance on the underlying objective (Gao et al., 2023). As Figure 1 illustrates, the challenge is not only to find a strong reward program, but also to maintain and refine reward guidance as the policy-induced trajectory distribution evolves.

To address these limitations, we introduce **Trajectory-Guided Reward Adaptation and Collaborative Evolution for Robotic Skill Learning (TRACE-RL)**. TRACE-RL maintains reward guidance as policy optimization shifts the visited behavior distribution. In the inner loop, an LLM-generated Task-Prior Reward Function provides broad task guidance, while an Executable Trajectory-Scoring Function labels pairs of current-policy trajectories to train a trajectory-calibrated reward. A Fusion Evaluation Function provides auxiliary assessments for fusing the two reward signals with held-out task-score rankings, preserving the prior under weak score-derived supervision and increasing the calibrated reward's influence when it reliably distinguishes new behaviors. TRACE-RL monitors PPO progress and, upon a plateau, collects fresh on-policy trajectories, refreshes score-derived labels and reward calibration, and continues optimization under the updated fused reward. Across outer iterations, structured training feedback jointly revises the reward, score, and evaluation functions; validated reward-model and policy states are transferred to sustain learning across revisions. Unlike ROSKA, PoRSE, and LaRes, which emphasize reward search, exploration bonuses, policy-side transfer, or relabeled experience (Huang et al., 2025; Chang et al., 2026; Li et al., 2025b), TRACE-RL continually updates reward representation from score-derived supervision on fresh trajectory pairs. It therefore combines explicit task priors with behavior-dependent refinement as the policy enters new behavioral regions.

Our contributions are threefold: (1) We propose an inner-loop reward-adaptation mechanism that recalibrates a learned reward using score-derived pairwise supervision on current-policy trajectories and combines it with an LLM-generated task prior through reliability-aware fusion. (2) We introduce a collaborative outer loop that jointly revises the reward, score, and evaluation functions while transferring the paired reward-model and policy states of the global-best candidate across iterations. (3) We evaluate TRACE-RL on eighteen robotic tasks spanning locomotion and dexterous manipulation, achieving the best or tied-best mean MTS among the compared methods on every task, with ablation studies examining the contributions of each component.

2 Related Work

LLM-Based Reward Design. Before the advent of LLMs, automated reward-design methods explored reward-function search, including genetic-programming approaches with manually specified operators (Niekum et al., 2010). Recent LLM-based methods generate interpretable reward specifications from task descriptions and environment information (Srivastava et al., 2026). Language-to-Rewards (Yu et al., 2023) and Text2Reward (Xie et al., 2024) generate robot reward specifications or executable code, while Self-Alignment (Zeng et al., 2024) studies LLM-based reward learning for robot skills. Eureka (Ma et al., 2024) iteratively refines reward candidates using policy-training feedback; REvolve (Hazra et al., 2025), R* (Li et al., 2025a), and RF-Agent (Gao et al., 2025) extend reward search through population evolution, parameter alignment, and tree search. Pairwise trajectory comparisons are also used to learn reward signals from behavioral preferences (Christiano et al., 2017; Lee et al., 2021; Metcalf et al., 2022; Hu et al., 2024). Unlike prior methods that optimize a single reward program, TRACE-RL preserves the task prior and complements it with a separately learned trajectory-calibrated reward based on score-derived pairwise supervision over current-policy trajectories to capture behavioral distinctions emerging during policy learning, thereby maintaining informative reward guidance as the visited behavior distribution changes.

Continual Reward-Policy Co-Learning. Continual policy learning under changing reward specifications aims to preserve useful policy knowledge while avoiding unstable updates (Khetarpal et al., 2022; Wolczyk et al., 2021). ROSKA (Huang et al., 2025) co-evolves reward and policy populations through policy fusion and selection. ORSO (Zhang et al., 2024) combines online reward selection with policy optimization. PoRSE (Chang et al., 2026) transfers prior policy knowledge while refining reward shaping and task-relevant exploration, whereas LaRes (Li et al., 2025b) reuses relabeled experience and constrains policy transfer during reward search. RIME further shows that online reward-model learning is sensitive to noisy comparison feedback and benefits from robust data selection and a stable transition to online training (Cheng et al., 2024). In preference-based alignment, MetaRM shows that a reward model trained on an early policy distribution can lose discriminative ability as the policy output distribution shifts (Dou et al., 2025; Ackermann et al., 2025; Eisenstein et al., 2023). In contrast, TRACE-RL continually trains a trajectory-calibrated reward model on newly collected policy data, providing updated reward guidance that supports policy continuation while transferring validated reward-model and policy states across iterations.

3 Preliminaries

Reinforcement Learning for Robotic Skill Acquisition. We formulate robotic skill learning as a discounted Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, P is the transition kernel, r is the transition reward function, and $\gamma \in (0, 1)$ is the discount factor. At time step t, the robot observes $s_t \in \mathcal{S}$, samples an action $a_t \sim \pi_\theta(\cdot \mid s_t)$, transitions to $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, and receives $r_t = r(s_t, a_t, s_{t+1}, d_t)$, where d_t denotes the termination signal. For an episode horizon H, policy optimization seeks parameters that maximize the expected discounted return:

$$J(\theta; r) = \mathbb{E}_{\tau \sim p_{\pi_\theta}} \left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1}, d_t) \right], \tag{1}$$

where p_{π_θ} denotes the trajectory distribution induced by π_θ and $\tau = (s_0, a_0, \ldots, s_H)$. We optimize this objective using PPO (Schulman et al., 2017) to update the policy from collected rollout data. In TRACE-RL, r is instantiated by the policy-training reward R_{PPO} defined in Section 4.2, which combines the fused dense reward R_{f} with the task-specific environment reward R_{s}.

4 Method

4.1 Overview

TRACE-RL maintains reward guidance as policy learning expands the visited behavior distribution (Figure 2). We refer to the three executable components as the reward function, score function, and

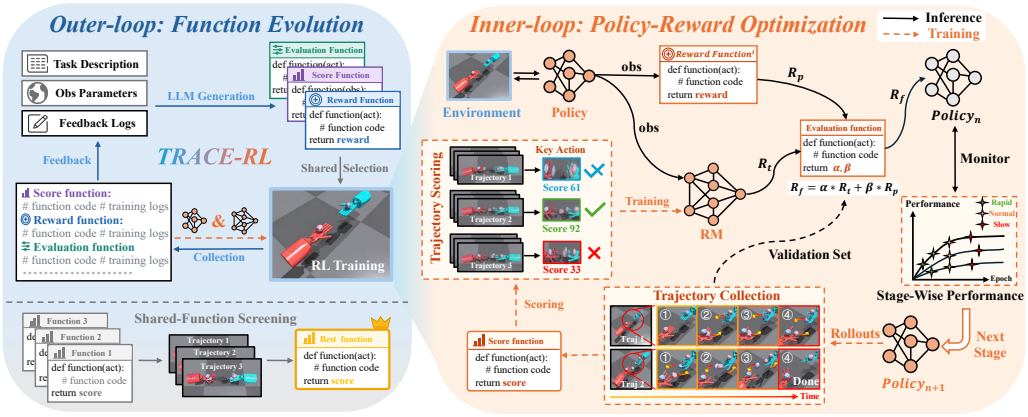

Figure 2: In the outer loop, TRACE-RL uses structured training feedback to jointly revise the reward, score, and evaluation functions. In the inner loop, current-policy trajectories periodically update the reward model, whose output is fused with the task-prior reward to optimize the policy.

evaluation function, respectively. At each outer iteration, the LLM generates group-specific reward functions and candidate pools of score functions and evaluation functions. Shared screening selects one score function and one evaluation function, which are paired with each group-specific reward function. Within every group, the inner loop alternates policy optimization and reward calibration: the score function labels pairs of on-policy trajectories to train a trajectory-calibrated reward model, and the evaluation function provides auxiliary assessments for determining the fusion weights. The policy is optimized under the fused reward. When task progress plateaus, fresh trajectories update the reward model, thereby refreshing the calibrated reward for the next stage. After inner-loop training, TRACE-RL selects the best candidate and reports final performance, while the statistics form structured feedback for joint LLM revision of the three functions. The best reward and policy models are transferred across iterations to support continual learning. Solid and dashed arrows in Figure 2 denote inference and training flows, respectively. The remainder of this section presents inner-loop trajectory-guided reward adaptation (Section 4.2) and outer-loop collaborative function evolution and continual reward–policy learning (Section 4.3).

4.2 Inner-Loop Trajectory-Guided Reward Adaptation

4.2.1 Programmatic Task-Prior and Trajectory-Calibrated Rewards

Each RL training group is represented by a candidate triple $\mathcal{G}_i^k = (\mathcal{R}_i^k, S^{k,*}, E^{k,*})$, where $\mathcal{R}_i^k$ is the group-specific reward function, while $S^{k,*}$ and $E^{k,*}$ are the shared score function and evaluation function selected for outer iteration k, respectively. For candidate $\mathcal{G}_i^k$, the LLM generates the executable reward function $\mathcal{R}_i^k$ from the task description and environment code. This reward function encodes task structure through dense progress, geometric, and stability terms, and produces the programmatic task-prior reward for a transition (s_t, a_t, s_{t+1}, d_t).

$$R_{\text{prior}} = \mathcal{R}_i^k(s_t, a_t, s_{t+1}, d_t). \tag{2}$$

In parallel, the selected score function $S^{k,*}$ evaluates pairs of on-policy trajectory segments. Its outputs are converted into reproducible score-derived soft pairwise targets, from which a reward model f_ϕ is trained with pairwise supervision. The transition-level output of this model defines the trajectory-calibrated reward

$$R_{\text{traj}} = f_\phi(s_t, a_t, s_{t+1}, d_t). \tag{3}$$

During reward-model training, f_ϕ aggregates its transition-level outputs within each segment into a segment score, and each pairwise target supervises the relative ordering of the two segment scores. Thus, R_{prior} supplies explicit task knowledge and R_{traj} captures behavior-quality distinctions learned from trajectories currently visited by the policy; neither reward replaces the other during policy learning, thereby preserving complementary sources of reward guidance.

4.2.2 ADAPTIVE REWARD FUSION

TRACE-RL optimizes the policy using two complementary rewards throughout the inner loop. We first normalize the two reward signals independently. We denote the resulting normalized trajectory-calibrated reward by R_{t} and the task-prior reward by R_{p}. Within each reward group, we estimate the reliability of both rewards on candidate-specific validation trajectory pairs reserved from that group's own rollout data. Environment task-score rankings on these pairs provide the direct reliability assessment. The evaluation function $E^{k,*}$ operates on the same candidate-specific validation pairs and provides relative-quality assessments for R_{t} and R_{p}.

Specifically, q_{t}^{T} denotes the ranking agreement of R_{t} with the held-out task-score rankings, whereas q_{t}^{E} denotes its agreement with the assessments produced by $E^{k,*}$. The corresponding quantities for R_{p} are denoted by q_{p}^{T} and q_{p}^{E}. The trajectory-calibrated reward coefficient is computed by aggregating these assessments:

$$\alpha = \mathcal{A}\big(\mathcal{Q}\big(q_{\mathrm{t}}^{T}, q_{\mathrm{t}}^{E}\big), \mathcal{Q}\big(q_{\mathrm{p}}^{T}, q_{\mathrm{p}}^{E}\big)\big), \tag{4}$$

where $\mathcal{Q}$ combines the available task-score and evaluation-function assessments, and $\mathcal{A}$ assigns a larger α when R_{t} receives stronger relative support. The coefficient of the programmatic task-prior reward is $\beta = 1 - \alpha$. The fused dense reward is therefore

$$R_{\mathrm{f}} = \alpha \cdot R_{\mathrm{t}} + \beta \cdot R_{\mathrm{p}}. \tag{5}$$

In addition to the fused reward R_{f}, TRACE-RL uses a task-specific environment reward R_{s} to provide direct task-progress or success-oriented feedback. Depending on the task, R_{s} may correspond to a sparse success signal or a continuous environment progress signal, as detailed in Table 16. This auxiliary reward is combined with R_{f} only when constructing the final reward supplied to PPO. Its effective coefficient is bounded above by 0.05 to limit its influence on the overall training signal. We denote the resulting policy-training reward by R_{PPO}.

4.2.3 STAGE-WISE POLICY IMPROVEMENT

The inner loop is organized as successive policy-learning stages rather than a single fixed training process. TRACE-RL monitors recent task-score improvement during training. After a minimum training budget has been consumed, TRACE-RL begins the next stage when the improvement in the best task score over a recent monitoring window falls below an adaptive plateau threshold estimated from earlier training progress. The framework then collects fresh current-policy trajectories, reconstructs informative trajectory pairs, and updates the trajectory-calibrated reward model. The updated reward model refreshes R_{t}, R_{f}, and the final PPO reward. PPO subsequently continues from the preceding policy state without reinitializing the policy between successive stages. This collect–calibrate–optimize cycle provides updated behavioral distinctions when the current reward guidance no longer supports effective improvement.

4.3 OUTER-LOOP COLLABORATIVE FUNCTION EVOLUTION

4.3.1 SHARED FUNCTION SELECTION

At each outer iteration, the LLM produces N candidates for each of the reward, score, and evaluation functions. The reward functions remain group-specific: candidate $\mathcal{R}_i^k$ is paired with its own policy rollout, prior-reward returns, and reward-model training data. This preserves the behavioral and optimization consequences of each proposed reward function. After each reward group has collected its initial current-policy rollouts, the score-function and evaluation-function candidates are screened independently on a common screening probe set. This set is constructed once from the initial rollout pool of a fixed anchor candidate and provides matched behavioral evidence for all function candidates. It is constructed separately from the candidate-specific validation pairs subsequently used for reliability-aware fusion. For a function pool F^k, the selected function is determined by

$$F^{k,*} = \mathcal{S}\big(F^k, \{A_{\mathrm{p},j}^k\}_{j=1}^N, \{A_{\mathrm{r},j}^k\}_{j=1}^N, \{A_{\mathrm{h},j}^k\}_{j=1}^N\big), \tag{6}$$

where $A_{\mathrm{p},j}^k$, $A_{\mathrm{r},j}^k$, and $A_{\mathrm{h},j}^k$ denote the task-score-aligned pairwise accuracy, Spearman rank agreement, and hard-pair accuracy of function candidate j, respectively. The screening operator $\mathcal{S}$ first

ranks candidates by $A^k_{\mathrm{p},j}$, then uses $A^k_{\mathrm{r},j}$ to distinguish candidates tied on the first criterion. The selected functions are then shared across all reward groups within the same outer iteration, providing consistent score-derived pairwise supervision and auxiliary reliability assessment while each reward function retains its own policy rollouts and reward-model data.

4.3.2 Collaborative Function Evolution

After an inner loop finishes, TRACE-RL summarizes each candidate using its task-score trace, held-out pairwise ranking agreement, reward-model diagnostics, evaluation-function assessments, fusion behavior, and execution records. The best valid candidate from outer iteration k is compared with the retained global-best candidate, and the higher-scoring one is retained as the single global-best reference for both the next LLM revision and cross-iteration state transfer. We denote this global-best reference after iteration k by $\mathcal{C}^k_{\mathrm{gb}}$. For the next outer iteration, the LLM jointly evolves the three executable function types according to

$$G^{k+1} = \mathcal{L}_{\mathrm{LLM}}\left(\mathcal{R}^k_{\mathrm{gb}}, S^k_{\mathrm{gb}}, E^k_{\mathrm{gb}}, T^k_{\mathrm{gb}}\right), \tag{7}$$

where $\mathcal{C}^k_{\mathrm{gb}}$ is the single global-best reference retained after outer iteration k. The symbols $\mathcal{R}^k_{\mathrm{gb}}$, S^k_{gb}, and E^k_{gb} denote the reward, score, and evaluation functions attached to this reference, which may have been produced in an earlier iteration. T^k_{gb} denotes its structured revision-feedback package for guiding subsequent revisions of all three functions. It contains the reference candidate's task-score trace, held-out pairwise ranking agreement, reward-model diagnostics, evaluation-function assessments, fusion behavior, and execution records, together with comparative summaries of the remaining candidates. G^{k+1} denotes the complete collection of function candidates produced for outer iteration $k+1$, including the group-specific reward-function candidates and the candidate pools of score and evaluation functions, while $\mathcal{L}_{\mathrm{LLM}}$ denotes the LLM-driven evolution process. The task and environment context is treated as fixed conditioning information implicit in $\mathcal{L}_{\mathrm{LLM}}$. Jointly evolving the three function types from their executable code and corresponding training evidence keeps task guidance, score-derived pairwise supervision, and fusion assessment aligned with the policy behaviors represented by the global-best reference.

4.3.3 Continual Reward and Policy Learning

Directly continuing policy training across different task-prior rewards can be unstable because each outer-loop revision changes the optimization objective and may disrupt previously acquired behaviors. TRACE-RL uses the trajectory-calibrated reward model as a bridging mechanism between successive reward specifications. At the beginning of a new outer iteration, the reward model and policy associated with the single global-best reference are retained as a paired initialization for subsequent training and incrementally updated using newly collected trajectories under the revised score function. Consequently, changes in score supervision refine an already trained reward representation rather than abruptly replacing the reward signal used for policy optimization. The recalibrated reward is then fused with the revised programmatic prior, providing a smoother transition between successive training objectives. The same global-best policy can therefore continue under progressively adapted reward guidance, preserving previously learned behavioral knowledge while learning behaviors required by the revised reward specification. By using this retained reward model to bridge successive reward specifications, TRACE-RL allows the policy to inherit all learned parameters and continue optimization across outer iterations without lossy modification, thereby maximizing the retention of previously acquired experience and knowledge.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate TRACE-RL on 18 GPU-accelerated robotic skill-learning tasks spanning locomotion, whole-body control, and dexterous manipulation (Makoviychuk et al., 2021; Chen et al., 2024). Detailed task definitions and their associated evaluation criteria are provided in Appendix A.4.1 and Table 16. Following prior reward-design studies (Ma et al., 2024; Huang et al., 2025; Chang et al., 2026), we report the maximum training score (MTS) for all tasks and the Human Normalized Score (HNS) for tasks with valid human-reward references.

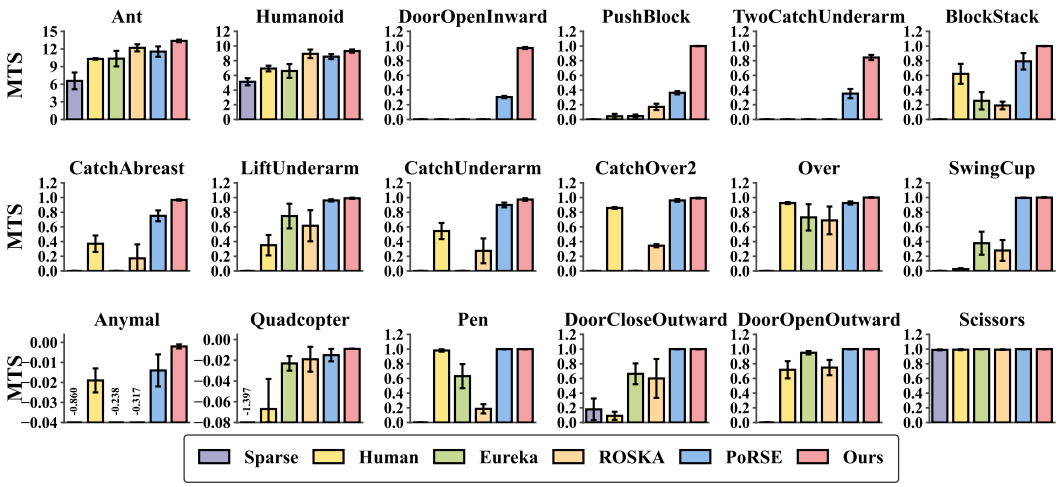

Figure 3: Maximum training score comparison across 18 robotic tasks under the same training budget. TRACE-RL achieves the best result on 14 tasks and ties for the best on the remaining four.

Training Settings. All methods are evaluated under the same task configurations, training budgets, and evaluation seed sets. Each result is aggregated over 5 independent seeds, with MTS computed separately for each run. Tables 3–5 in the appendix report the main benchmark means and standard deviations; the ablation tables report mean MTS and relative gaps. For LLM-based methods, we align the number of reward candidates, outer iterations, and policy-training budget. TRACE-RL uses GPT-5.6-Terra to generate 6 candidates at each outer iteration and runs for 5 outer iterations. Each candidate receives a policy training budget of 3000 epochs per outer iteration. Complete PPO, LLM, and task-specific hyperparameters are provided in Appendix A.4.

Baselines. We compare TRACE-RL with sparse environment rewards and official human-engineered dense rewards from Isaac Gym and Bi-DexHands (Makoviychuk et al., 2021; Chen et al., 2024); Eureka (Ma et al., 2024), which iteratively searches executable reward programs using policy-training feedback; ROSKA (Huang et al., 2025), which co-evolves reward and policy candidates using policy performance for candidate selection; and PoRSE (Chang et al., 2026), which augments LLM-generated task rewards with exploration bonuses.

Metrics. Our primary metric is maximum training score (MTS), defined as the highest environment task score observed during a training run. Unlike candidate-specific reward returns, the task score uses a fixed environment-defined metric shared by all methods, enabling direct comparison across rewards with different scales and compositions. We compute MTS separately for each random seed within the prescribed training budget and report the mean and standard deviation across five seeds for the main benchmarks. MTS therefore reflects the best task performance obtained by each method under the same optimization budget.

5.2 Main Results

MTS Result Comparison. Figure 3 compares MTS of TRACE-RL and prior methods under the same training budget. TRACE-RL achieves the best or tied-best result on all 18 tasks, strictly outperforming the strongest prior baseline on 14 tasks. Its advantage spans locomotion, whole-body control, and dexterous manipulation. On Ant and Humanoid, TRACE-RL improves the strongest previous scores from 12.192 and 8.941 to 13.372 and 9.320, respectively. The gains are especially pronounced on sparse, contact-intensive tasks: TRACE-RL reaches 1.000 on BlockStack and PushBlock, compared with prior best scores of 0.792 and 0.363, and improves DoorOpen Inward and TwoCatchUnderarm from 0.305 and 0.352 to 0.973 and 0.844. Overall, TRACE-RL attains the maximum score of 1.000 on eight tasks and ties the best prior performance on Pen, DoorClose Outward, DoorOpen Outward, and Scissors. These results indicate that combining programmatic task guidance with trajectory-calibrated refinement provides effective reward signals across diverse control settings, particularly for fine-grained contact and near-success behaviors.

Inner-loop Adaptation Efficiency. Figure 4 evaluates TRACE-RL's inner-loop adaptation using a fixed LLM-generated function group under the same candidate count and policy-training budget as one outer iteration. Without additional outer-loop function generation, the first iteration surpasses the strongest prior results on CatchAbreast, PushBlock, and CatchUnderarm, while remaining close on CatchOver2 and LiftUnderarm. Its gaps from full TRACE-RL across the five tasks are approximately 16%, 4%, 4%, 5%, and 6%, respectively. The advantage is most evident on PushBlock, where the first-round result is only 4% below full TRACE-RL, compared with 64% for the strongest prior baseline. These results show that inner-loop reward adaptation extracts substantial gains from a single outer-loop function-generation round, reducing reliance on repeated function redesign.

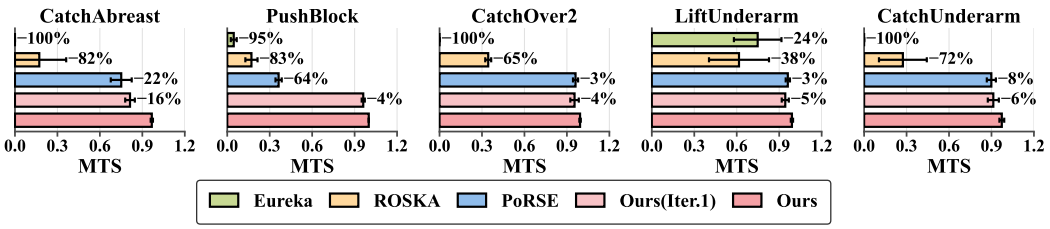

Figure 4: Inner-loop effectiveness on five representative tasks using only the initial LLM-generated function group. TRACE-RL (Iter. 1) approaches its final performance without additional outer-loop reward redesign; percentages denote relative gaps to the full TRACE-RL results.

Outer-loop Iterative Improvement. Figure 5 shows the cumulative best MTS over five outer-loop iterations on four representative tasks, with the mean cumulative best MTS increasing after every iteration. From Iter. 1 to Iter. 5, Ant increases from 10.166 to 13.372 and Humanoid from 6.658 to 9.320, corresponding to gains of 31.5% and 40.0%. Larger relative improvements occur on contact-intensive tasks: TwoCatchUnderarm rises from 0.529 to 0.844 and DoorOpen Inward from 0.632 to 0.973, yielding gains of 59.5% and 54.0%. Both contact-intensive tasks also improve in the final iteration, showing that the gains extend beyond the early rounds. This pattern is consistent with feedback-driven function revision and continued policy learning providing further opportunities to refine task guidance as new behaviors emerge over successive rounds of training.

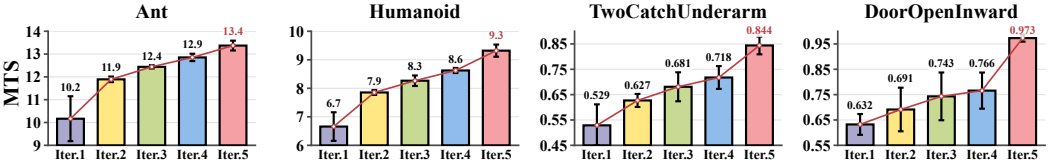

Figure 5: Evolution of the cumulative best maximum training score (MTS) over five outer-loop iterations on four representative tasks, with earlier best scores carried forward.

5.3 Ablation Experiments and Result Analysis

Ablation Setup. We conduct ablations on three tasks spanning different control regimes and difficulty levels. Table 1 evaluates reward guidance and cross-iteration inheritance by removing R_{p}, R_{t}, or R_{s}, replacing the evaluation function EF with equal weighting, and disabling policy- or reward-model inheritance. In the R_{p} ablation, shared score- and evaluation-function screening is also disabled to avoid introducing an implicit task prior. Table 2 evaluates transfer design by reinitializing the reward-model head, retaining only the first backbone layer, or adopting ROSKA-, PoRSE-, and LaRes-style policy transfer. All other task settings remain unchanged.

Reward Guidance and Cross-Iteration Inheritance. Table 1 evaluates the three reward signals, reliability-aware fusion, and cross-iteration inheritance. Every ablation reduces performance on all three tasks. On TwoCatchUnderarm, removing R_{t} causes the largest reward-component drop of 19.31%, compared with 10.78% and 12.44% for removing R_{p} and R_{s}, confirming their complementary roles in structural guidance, behavior discrimination, and task-completion feedback. Replacing reliability-aware fusion with equal weighting reduces performance by 8.50% on BlockStack and 10.90% on TwoCatchUnderarm. Disabling policy- or reward-model inheritance further produces

Table 1: Ablation of reward guidance and cross-iteration state inheritance on three representative tasks. Gap denotes the relative MTS decrease from the full TRACE-RL configuration.

Method	Ant		BlockStack		TwoCatchUnderarm	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL w/o R_{p}	12.523	6.35%↓	0.921	7.90%↓	0.753	10.78%↓
TRACE-RL w/o R_{t}	12.816	4.16%↓	0.931	6.90%↓	0.681	19.31%↓
TRACE-RL w/o R_{s}	12.707	4.97%↓	0.934	6.60%↓	0.739	12.44%↓
TRACE-RL w/o EF	12.551	6.14%↓	0.915	8.50%↓	0.752	10.90%↓
TRACE-RL w/o PM_{inherit}	11.945	10.67%↓	0.894	10.60%↓	0.706	16.35%↓
TRACE-RL w/o RM_{inherit}	12.043	9.94%↓	0.913	8.70%↓	0.721	14.57%↓
TRACE-RL	**13.372**		**1.000**		**0.844**	

drops of 10.60%–16.35% and 8.70%–14.57%. These results demonstrate the importance of both adaptive reward guidance and retaining learned policy and reward knowledge across iterations.

Table 2: Ablation of reward model inheritance and policy transfer strategies on three representative tasks. Gap denotes the relative MTS decrease from the full TRACE-RL configuration.

Method	Ant		BlockStack		TwoCatchUnderarm	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL w/o RM_{head}	12.792	4.34%↓	0.941	5.90%↓	0.680	19.43%↓
TRACE-RL w/o RM_{back}	12.930	3.31%↓	0.946	5.40%↓	0.788	6.64%↓
TRACE-RL w/ PM_{ROSKA}	11.495	14.04%↓	0.872	12.80%↓	0.757	10.31%↓
TRACE-RL w/ PM_{PoRSE}	10.917	18.36%↓	0.885	11.50%↓	0.713	15.52%↓
TRACE-RL w/ PM_{LaRes}	11.248	15.88%↓	0.862	13.80%↓	0.755	10.55%↓
TRACE-RL	**13.372**		**1.000**		**0.844**	

Reward-Model and Policy-Transfer Design. Table 2 evaluates reward-model inheritance and policy transfer across outer iterations. On TwoCatchUnderarm, reinitializing the reward-model head reduces performance by 19.43%, while retaining only the first backbone layer causes a 6.64% drop. Both variants also reduce Ant and BlockStack performance by 3.31%–5.90%, supporting complete reward-model inheritance rather than partial parameter reuse. Replacing TRACE-RL's policy transfer with ROSKA-, PoRSE-, or LaRes-style alternatives produces drops of 10.31%–18.36% across the three tasks evaluated in this study, with none consistently matching the full method under the tested settings. These results confirm that complete reward-model inheritance and policy transfer are important for preserving progress under evolving reward guidance.

6 Conclusion

We introduced TRACE-RL to maintain reward guidance as policy behavior evolves. Its inner loop learns a trajectory-calibrated reward from score-derived pairwise supervision on current-policy trajectories and fuses it with a programmatic task prior using reliability-aware weights. The outer loop jointly revises reward, score, and evaluation functions, retaining the reward model to bridge reward revisions and support full policy inheritance. TRACE-RL achieves the best or tied-best MTS on all 18 robotic tasks, with ablations validating key components. These results demonstrate the benefits of adapting reward guidance to the evolving trajectory distribution across diverse control regimes. More broadly, TRACE-RL provides a practical approach to long-horizon reward-policy learning without discarding previously acquired policy knowledge.

REPRODUCIBILITY STATEMENT

TRACE-RL is evaluated in a reproducible simulation setting using the Isaac Gym benchmark suite across locomotion, whole-body control, and dexterous manipulation tasks. The main paper specifies the reward-guidance mechanism, the outer-loop reward-refinement procedure, the trajectory collection process, the inner-loop policy-optimization schedule, and the evaluation metric. The appendix reports the complete task list, implementation details, ablation configurations, normalization definitions, and additional results to facilitate independent examination of the experimental protocol. For the main benchmark results, each task is run with five independent random seeds. We report the mean and standard deviation of the maximum training score (MTS), where the maximum is taken over the recorded task-score history for each run. The experimental protocol also fixes the reward-function interface, trajectory processing rules, training budget, evaluation procedure, and seed-based aggregation, so that the reported comparisons do not depend on undocumented manual choices.

ETHICS STATEMENT

This work studies reward-guided reinforcement learning entirely in simulated robotics environments and does not involve human participants, animal experiments, personally identifiable information, or sensitive personal data. The benchmark tasks and evaluation procedures are used for algorithmic research and do not make decisions about real individuals. We are not aware of any additional ethical, privacy, safety, or legal concerns specific to the experiments reported here. We nevertheless follow the ICLR Code of Ethics and take responsibility for the reliability, validity, and responsible presentation of the reported results, including transparent reporting of experimental settings.

AI USE STATEMENT

We used large language models to assist with manuscript preparation, including language polishing, translation, and revisions to improve the clarity and readability of the text and experimental-result discussions. The authors reviewed and edited the AI-assisted text to preserve the intended meaning. Separately, as part of the TRACE-RL research workflow, we used generative language models, including GPT-5.6-Terra, to generate and refine candidate programmatic reward, trajectory-scoring, and evaluation functions, as described in the method and experimental sections. Model-generated functions were checked for interface compatibility, numerical validity, and training-time behavior during the experimental workflow. Experimental results, analyses, figures, and scientific claims were reviewed by the authors. Language models were not assigned authorship or responsibility for the work. The authors remain fully responsible for the final algorithms, code, data processing, experimental design, interpretations, and all text and artifacts in this submission.

REFERENCES

Johannes Ackermann, Takashi Ishida, and Masashi Sugiyama. Off-policy corrected reward modeling for reinforcement learning from human feedback. In *Proceedings of the Second Conference on Language Modeling*, 2025.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5048–5058, 2017.

Serena Booth, W. Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 5920–5929, 2023. doi: 10.1609/aaai.v37i5.25733.

Yanbin Chang, Junfan Lin, Jie Jiang, Runhao Zeng, Changxin Huang, and Jianqiang Li. Master skill learning with policy-grounded synergy of llm-based reward shaping and exploring. In *International Conference on Learning Representations*, 2026.

Yuanpei Chen, Yiran Geng, Fangwei Zhong, Jiaming Ji, Jiechuang Jiang, Zongqing Lu, Hao Dong, and Yaodong Yang. Bi-dexhands: Towards human-level bimanual dexterous manipulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):2804–2818, 2024.

Jie Cheng, Gang Xiong, Xingyuan Dai, Qinghai Miao, Yisheng Lv, and Fei-Yue Wang. RIME: Robust preference-based reinforcement learning with noisy preferences. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 8229–8247, 2024.

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Shihan Dou, Yan Liu, Enyu Zhou, Songyang Gao, Tianlong Li, Limao Xiong, Xin Zhao, Haoxiang Jia, Junjie Ye, Rui Zheng, et al. Alleviating shifted distribution in human preference alignment through meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 23805–23813, 2025.

Jacob Eisenstein, Jonathan Berant, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alexander Nicholas D'Amour, Krishnamurthy Dj Dvijotham, Katherine Heller, Stephen Robert Pfohl, and Deepak Ramachandran. Reward model underspecification in language model alignment. In *NeurIPS 2023 Workshop on Distribution Shifts*, 2023.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866, 2023.

Ning Gao, Xiuhui Zhang, Xingyu Jiang, Mukang You, Mohan Zhang, and Yue Deng. RF-Agent: Automated reward function design via language agent tree search. In *Advances in Neural Information Processing Systems*, 2025.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. REvolve: Reward evolution with large language models using human feedback. In *International Conference on Learning Representations*, 2025.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Xiao Hu, Jianxiong Li, Xianyuan Zhan, Qing-Shan Jia, and Ya-Qin Zhang. Query-policy misalignment in preference-based reinforcement learning. In *International Conference on Learning Representations*, 2024.

Changxin Huang, Yanbin Chang, Junfan Lin, Junyang Liang, Runhao Zeng, and Jianqiang Li. Efficient language-instructed skill acquisition via reward-policy co-evolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022. doi: 10.1613/JAIR.1.13673.

W. Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis)design for autonomous driving. *Artificial Intelligence*, 316:103829, 2023. doi: 10.1016/j.artint.2022.103829.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=10uNUgI5Kl.

Kimin Lee, Laura M. Smith, and Pieter Abbeel. PEBBLE: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. In *International Conference on Machine Learning*, pp. 6152–6163, 2021.

Pengyi Li, Jianye Hao, Hongyao Tang, Yifu Yuan, Jinbin Qiao, Zibin Dong, and Yan Zheng. R*: Efficient reward design via reward structure evolution and parameter alignment optimization with large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025a.

Pengyi Li, Hongyao Tang, Jinbin Qiao, Yan Zheng, and Jianye Hao. Lares: Evolutionary reinforcement learning with llm-based adaptive reward search. In *Advances in Neural Information Processing Systems*, 2025b.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Katherine Metcalf, Miguel Sarabia, and Barry-John Theobald. Rewards encoding environment dynamics improves preference-based reinforcement learning. *arXiv preprint arXiv:2211.06527*, 2022.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, 1999.

Hexian Ni, Tao Lu, and Yinghao Cai. CoRe: Combined rewards with vision-language model feedback for preference-aligned reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2607.01721.

Scott Niekum, Andrew G. Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2778–2787, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sanjana Srivastava, Kangrui Wang, Yung-Chieh Chan, Tianyuan Dai, Manling Li, Ruohan Zhang, Mengdi Xu, Jiajun Wu, and Li Fei-Fei. ROSETTA: Constructing code-based reward from unconstrained language preference. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Ig6goVdtjb.

Ruiqi Wang, Dezhong Zhao, Ziqin Yuan, Ike Obi, and Byung-Cheol Min. PrefCLM: Enhancing preference-based reinforcement learning with crowdsourced large language models. *IEEE Robotics and Automation Letters*, 10(3):2486–2493, 2025. doi: 10.1109/LRA.2025.3528663.

Maciej Wolczyk, Michal Zajac, Razvan Pascanu, Lukasz Kucinski, and Piotr Milos. Continual world: A robotic benchmark for continual reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 28496–28510, 2021.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, 2024.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 374–404, 2023.

Yuwei Zeng, Yao Mu, and Lin Shao. Learning reward for robot skills using large language models via self-alignment. In *International Conference on Machine Learning*, pp. 58366–58386, 2024.

Chen Bo Calvin Zhang, Zhang-Wei Hong, Aldo Pacchiano, and Pulkit Agrawal. Orso: Accelerating reward design via online reward selection and policy optimization. *arXiv preprint arXiv:2410.13837*, 2024.

A Additional Details

A.1 Detailed Experimental Results

Detailed MTS Results. Tables 3–5 report the complete numerical MTS results as the mean and standard deviation over five seeds. MTS measures peak performance during training, not an independent evaluation of the returned policy. TRACE-RL achieves the best or tied-best mean MTS among the compared methods on all 18 tasks, with strict improvements on 14 tasks and ties on four. On the locomotion and whole-body control tasks, TRACE-RL reaches 13.372 ± 0.216 on Ant and 9.320 ± 0.213 on Humanoid, improving upon the strongest prior results of 12.192 ± 0.607 and 8.941 ± 0.580, respectively. It also obtains the highest scores on Anymal and Quadcopter, improving the strongest prior means from -0.014 and -0.015 to -0.002 and -0.009, where less negative values indicate better task performance. These results indicate that the advantage of TRACE-RL is maintained across tasks with substantially different score ranges and control dynamics.

The improvements are particularly pronounced on sparse and contact-intensive manipulation tasks under matched PPO-training budgets and the shared evaluation protocol. TRACE-RL raises the strongest prior scores from 0.792 to 1.000 on BlockStack, from 0.363 to 1.000 on PushBlock, from 0.305 to 0.973 on DoorOpen Inward, and from 0.352 to 0.844 on TwoCatchUnderarm. Consistent gains are also observed on Over (0.926 to 1.000), CatchOver2 (0.961 to 0.993), CatchAbreast (0.751 to 0.967), LiftUnderarm (0.961 to 0.990), and CatchUnderarm (0.899 to 0.973). TRACE-RL attains the maximum score of 1.000 on eight tasks and matches the strongest prior result on Pen, DoorClose Outward, DoorOpen Outward, and Scissors. Moreover, all nonsaturated TRACE-RL results on bounded manipulation tasks have standard deviations no greater than 0.035, indicating consistent performance across seeds. Collectively, the detailed results show that TRACE-RL provides both broad improvements across control domains and substantial gains on tasks where progress depends on distinguishing fine-grained contact transitions and near-success trajectories.

For each task with a valid human-reward reference, we compute the Human Normalized Score as

$$\mathrm{HNS} = \frac{\mathrm{MTS}_{\mathrm{method}} - \mathrm{MTS}_{\mathrm{sparse}}}{\mathrm{MTS}_{\mathrm{human}} - \mathrm{MTS}_{\mathrm{sparse}}}. \tag{8}$$

Here, $\mathrm{MTS}_{\mathrm{method}}$, $\mathrm{MTS}_{\mathrm{sparse}}$, and $\mathrm{MTS}_{\mathrm{human}}$ denote the corresponding mean MTS values. Thus, HNS values of 0 and 1 represent the sparse and human-reward baselines, respectively.

Table 3: Detailed MTS results on Ant, Humanoid, BlockStack, PushBlock, Over, and CatchOver2. Values are the mean and standard deviation over five seeds, with the best results shown in bold.

Method	Ant	Humanoid	Block Stack	Push Block	Over	Catch Over2
Sparse	6.565±1.429	5.130±0.486	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000
Human	10.304±0.122	6.933±0.386	0.621±0.136	0.043±0.033	0.925±0.013	0.858±0.013
Eureka	10.353±1.321	6.602±0.945	0.254±0.118	0.046±0.021	0.730±0.179	0.000±0.000
ROSKA	12.192±0.607	8.941±0.580	0.190±0.052	0.171±0.043	0.689±0.189	0.344±0.021
PoRSE	11.554±0.880	8.559±0.340	0.792±0.112	0.363±0.022	0.926±0.021	0.961±0.018
Ours	**13.372±0.216**	**9.320±0.213**	**1.000±0.000**	**1.000±0.000**	**1.000±0.000**	**0.993±0.005**

HNS Result Comparison. Figure 6 reports HNS results for 15 tasks with valid human-reward references; an additional task label, DoorClose Outward, is retained without HNS results because its human-reward reference is invalid. TRACE-RL achieves the best or tied-best HNS on every reported task, strictly outperforming prior methods on 12 tasks and tying on the remaining three. It also exceeds the human-reward baseline on all reported tasks, showing that its improvements remain consistent after normalization across different task-score scales while retaining the performance ordering within each task. For instance, TRACE-RL reaches 1.821 on Ant and 2.324 on Humanoid, compared with the strongest prior results of 1.505 and 2.114. Larger gains appear on PushBlock, where HNS increases from 8.442 to 23.256. It further matches the strongest results on Pen, DoorOpen Outward, and Scissors. Overall, these results demonstrate that TRACE-RL consistently improves upon both sparse and human-designed reward references.

Table 4: Detailed MTS results on CatchAbreast, LiftUnderarm, CatchUnderarm, TwoCatchUnderarm, Pen, and DoorOpen Inward. MTS measures the peak score recorded during training. Values are the mean and standard deviation over five seeds, with the best results shown in bold.

Method	Catch Abreast	Lift Underarm	Catch Underarm	TwoCatch Underarm	Pen	DoorOpen Inward
Sparse	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000	0.000±0.000
Human	0.371±0.112	0.351±0.139	0.544±0.110	0.000±0.000	0.981±0.016	0.000±0.000
Eureka	0.000±0.000	0.748±0.168	0.000±0.000	0.000±0.000	0.631±0.164	0.000±0.000
ROSKA	0.172±0.190	0.616±0.212	0.274±0.169	0.000±0.000	0.186±0.063	0.000±0.000
PoRSE	0.751±0.074	0.961±0.015	0.899±0.032	0.352±0.063	**1.000±0.000**	0.305±0.015
Ours	**0.967±0.008**	**0.990±0.008**	**0.973±0.017**	**0.844±0.035**	**1.000±0.000**	**0.973±0.014**

Table 5: Detailed MTS results on DoorClose Outward, DoorOpen Outward, Scissors, SwingCup, Anymal, and Quadcopter. MTS measures the peak score recorded during training. Values are the mean and standard deviation over five seeds, with the best results shown in bold.

Method	DoorClose Outward	DoorOpen Outward	Scissors	Swing Cup	Anymal	Quadcopter
Sparse	0.179±0.148	0.000±0.000	0.989±0.006	0.000±0.000	-0.860±0.018	-1.397±0.083
Human	0.091±0.055	0.718±0.117	0.992±0.007	0.025±0.013	-0.019±0.006	-0.067±0.029
Eureka	0.663±0.142	0.950±0.023	**1.000±0.000**	0.378±0.157	-0.238±0.031	-0.023±0.007
ROSKA	0.600±0.265	0.748±0.104	0.992±0.002	0.279±0.142	-0.317±0.140	-0.019±0.012
PoRSE	**1.000±0.000**	**1.000±0.000**	**1.000±0.000**	0.996±0.002	-0.014±0.008	-0.015±0.006
Ours	**1.000±0.000**	**1.000±0.000**	**1.000±0.000**	**1.000±0.000**	**-0.002±0.001**	**-0.009±0.000**

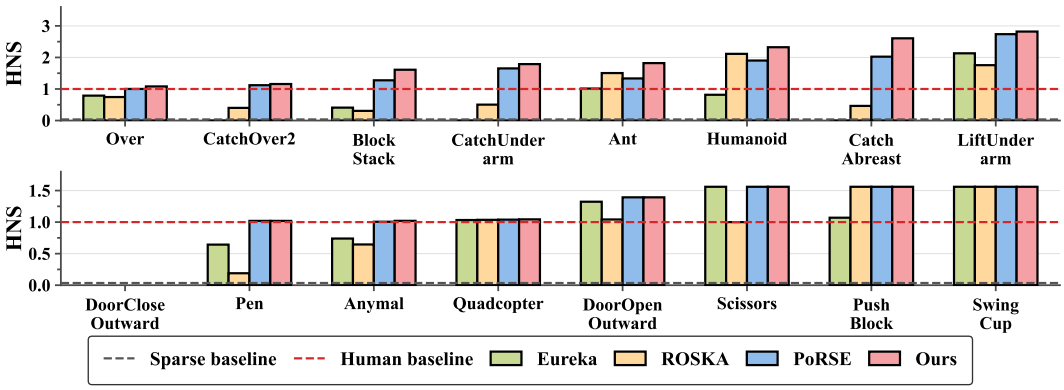

Figure 6: Human-normalized score (HNS) comparison across 15 robotic tasks with valid human-reward references. DoorClose Outward is additionally shown as a task label without HNS results because it lacks a valid human-reward reference. Gray and red dashed lines denote the sparse and human-reward baselines, respectively; TRACE-RL exceeds the human reference on every task.

Detailed HNS Results. Tables 6–8 report the complete HNS results using identical reference rewards for all methods within each task. For each task, we first average the MTS values over the five seeds for each method and reference reward, then apply Equation 8 to compute HNS rather than averaging individual ratios. We report HNS only for tasks on which the human-designed reward outperforms the sparse reward, so that the normalization represents improvement relative to a meaningful human-reward reference. DoorClose Outward is excluded because its human-reward score is lower than its sparse-reward score, while TwoCatchUnderarm and DoorOpen Inward are excluded because the two reference scores are identical, yielding a zero denominator. This restriction avoids interpreting undefined or reversed reference scales as meaningful measures of relative performance. Among the remaining 15 tasks, TRACE-RL achieves the best or tied-best HNS on every task, with strict improvements on 12 tasks and ties on Pen, DoorOpen Outward, and Scissors. Moreover, all

reported TRACE-RL results exceed $\mathrm{HNS} = 1$, demonstrating consistent improvements over both the sparse and human-designed reward baselines.

The normalized results further reveal how the gains vary across task families. TRACE-RL improves the strongest prior HNS from 1.505 to 1.821 on Ant, from 2.114 to 2.324 on Humanoid, and from 1.275 to 1.610 on BlockStack. On contact-intensive manipulation tasks, it reaches 2.606 on CatchAbreast, 2.821 on LiftUnderarm, and 1.789 on CatchUnderarm, compared with prior best results of 2.024, 2.738, and 1.653, respectively. TRACE-RL also obtains the highest HNS on Over, CatchOver2, Anymal, and Quadcopter. Particularly large normalized values occur on PushBlock, SwingCup, and Scissors because the performance gaps between their sparse and human-designed rewards are small, reflecting the sensitivity of HNS to small denominators. For example, the human-reward MTS values on PushBlock and SwingCup are only 0.043 and 0.025 above their sparse baselines, causing the near-perfect TRACE-RL scores to correspond to HNS values of 23.256 and 40.000. These values should therefore be interpreted together with the raw MTS results rather than as direct multiplicative improvements in absolute task performance. Overall, the HNS results confirm that TRACE-RL's advantage is preserved after accounting for task-specific score scales, while the MTS tables provide the primary measure of absolute performance.

Table 6: Detailed HNS results on Ant, Humanoid, BlockStack, PushBlock, DoorClose Outward, and CatchOver2. N/A indicates unavailable normalization, and the best results are shown in bold.

Method	Ant	Humanoid	Block Stack	Push Block	DoorClose Outward	Catch Over2
Eureka	1.013	0.816	0.409	1.070	N/A	0.000
ROSKA	1.505	2.114	0.306	3.977	N/A	0.401
PoRSE	1.334	1.902	1.275	8.442	N/A	1.120
Ours	**1.821**	**2.324**	**1.610**	**23.256**	N/A	**1.157**

Table 7: Detailed HNS results on CatchAbreast, LiftUnderarm, CatchUnderarm, TwoCatchUnderarm, Pen, and DoorOpen Inward. N/A indicates unavailable normalization because human and sparse reference scores are identical, and the best results are shown in bold.

Method	Catch Abreast	Lift Underarm	Catch Underarm	TwoCatch Underarm	Pen	DoorOpen Inward
Eureka	0.000	2.131	0.000	N/A	0.643	N/A
ROSKA	0.464	1.755	0.504	N/A	0.190	N/A
PoRSE	2.024	2.738	1.653	N/A	**1.019**	N/A
Ours	**2.606**	**2.821**	**1.789**	N/A	**1.019**	N/A

Table 8: Detailed HNS results on Over, DoorOpen Outward, Scissors, SwingCup, Anymal, and Quadcopter. HNS uses task-specific sparse and human-designed reward references to normalize the mean MTS across seeds. The best results are shown in bold.

Method	Over	DoorOpen Outward	Scissors	Swing Cup	Anymal	Quadcopter
Eureka	0.789	1.323	**3.667**	15.120	0.740	1.033
ROSKA	0.745	1.042	1.000	11.160	0.646	1.036
PoRSE	1.001	**1.393**	**3.667**	39.840	1.006	1.039
Ours	**1.081**	**1.393**	**3.667**	**40.000**	**1.020**	**1.044**

A.2 Ablation Visualization

Figures 7 and 8 visualize the component and transfer-design ablations reported in the main text.

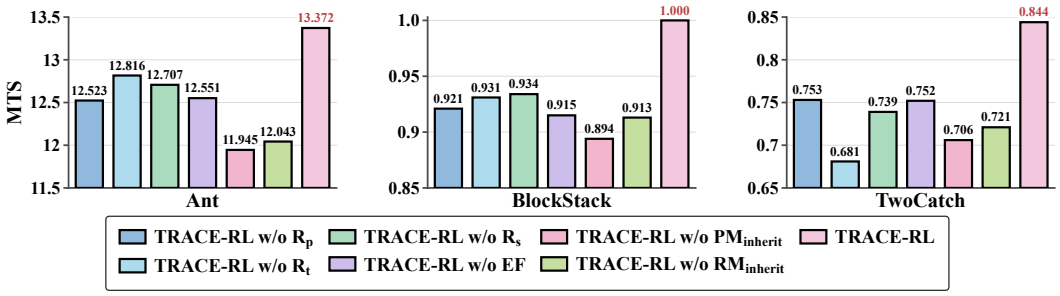

Figure 7: Component ablations of TRACE-RL on Ant, BlockStack, and TwoCatchUnderarm. Bars report mean MTS for reward-guidance and inheritance variants, with full TRACE-RL as the reference. The w/o R_{p} variant also removes shared function screening; w/o EF denotes fixed equal-weight fusion. Higher bars indicate better task performance.

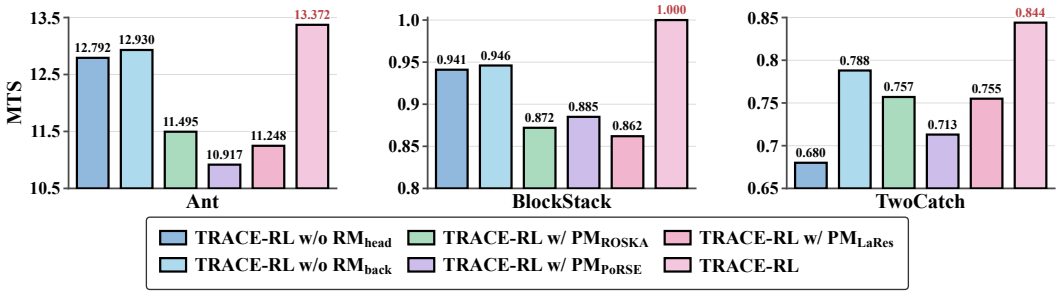

Figure 8: Ablations of reward-model inheritance and policy-transfer design in TRACE-RL on Ant, BlockStack, and TwoCatchUnderarm. Bars report MTS for variants that reinitialize the reward-model head, restrict reward-model backbone inheritance to the first layer, or replace TRACE-RL's policy-transfer design with the strategies used by ROSKA, PoRSE, and LaRes; the complete TRACE-RL configuration is shown as a reference. Higher bars indicate better task performance.

Component Contributions. Figure 7 examines reward guidance, reliability-aware fusion, and cross-iteration inheritance. All tested variants have lower mean MTS on all three tasks, with task-dependent effects. On TwoCatchUnderarm, removing the trajectory-calibrated reward R_{t} produces the largest reward-component drop, reducing MTS from 0.844 to 0.681, or 19.31%. Jointly removing the programmatic task prior R_{p} and shared screening, or removing the environment reward R_{s}, yields decreases of 10.78% and 12.44%, respectively. The joint prior/screening removal reduces performance on Ant and BlockStack by 6.35% and 7.90%, respectively; these changes do not isolate the contribution of R_{p}. Replacing reliability-aware fusion with fixed equal weights produces drops of 6.14%, 8.50%, and 10.90%, supporting adaptive fusion over this fixed-weight baseline. Cross-iteration inheritance is similarly important: removing policy inheritance decreases performance by 10.67%, 10.60%, and 16.35%, while removing reward-model inheritance results in drops of 9.94%, 8.70%, and 14.57%. These results show that TRACE-RL relies on both adaptive reward guidance and the preservation of learned policy and reward-model states during continued policy optimization across outer iterations, especially on the more demanding TwoCatchUnderarm task.

Reward-Model and Policy-Transfer Design. Figure 8 further evaluates the design of reward-model inheritance and policy transfer. Reinitializing the reward-model head or restricting backbone inheritance to the first layer consistently degrades performance. The reward head is particularly important on TwoCatchUnderarm: reinitializing it reduces MTS from 0.844 to 0.680, corresponding to a 19.43% drop, whereas restricting backbone inheritance to the first layer reduces MTS to 0.788, a 6.64% drop. On Ant and BlockStack, both variants produce smaller but consistent reductions of 3.31%–5.90%. This pattern suggests that the inherited backbone preserves reusable trajectory representations, while the task-facing reward head is especially important for retaining distinctions among precise multi-object and contact-sensitive behaviors. Replacing TRACE-RL's policy-transfer design with the strategies used by ROSKA, PoRSE, or LaRes (Huang et al., 2025; Chang et al., 2026; Li et al., 2025b) reduces performance by 10.31%–18.36% across the three tasks. Although the

ROSKA-style strategy performs best among these alternatives on Ant and TwoCatchUnderarm and the PoRSE-style strategy performs best on BlockStack, none consistently approaches the complete TRACE-RL configuration. The results support transferring the policy together with the calibrated reward model as a paired initialization for subsequent policy optimization.

A.3 Supplementary Ablation Studies

The seven supplementary ablations use a separate protocol on Ant, Humanoid, and Anymal, rather than the Ant, BlockStack, and TwoCatchUnderarm tasks used for the main-text ablations. Unless explicitly varied, this suite uses GPT-5.6-Terra, five outer iterations, and six candidates per iteration. Each candidate receives $1{,}000$ PPO epochs per outer iteration, rather than $3{,}000$, divided into three stages of 167, 333, and 500 epochs. Adaptive stage extension and candidate pruning are disabled to preserve this per-candidate budget; epoch-based calibration, warm-up, and monitoring settings are adjusted to the reduced budget, with at most 83 epochs per reward-model calibration stage. The shared baseline uses a fixed maximum trajectory length of $L_{\mathrm{traj}} = 200$. MTS is computed separately for each run and averaged over five independent seeds. Gap denotes the relative change in mean MTS from the full TRACE-RL baseline under this supplementary protocol, normalized by the absolute baseline MTS; arrows indicate decreases or increases. It is not computed relative to the main-text results. For Anymal, gaps are computed from the displayed three-decimal MTS values using the baseline -0.073 and reported to three decimal places. Within each study, settings other than the designated ablation factors are held fixed relative to this baseline. The seven studies examine: (i) sensitivity to trajectory length at a fixed number of collected trajectories; (ii) dependence on the backbone LLM used to generate functions; (iii) the contribution of shared screening for the score and evaluation functions, individually and jointly; (iv) sensitivity to the number of score-labeled trajectory pairs at a fixed trajectory length; (v) the respective roles of reward-model updating and trajectory-data refreshing in online calibration; (vi) the contribution of jointly evolving the reward, score, and evaluation functions compared with evolving restricted subsets; and (vii) the effect of candidate-pool size on outer-loop search. The final study holds the per-candidate training budget fixed, so its total search budget varies with the number of candidates.

A.3.1 Trajectory-Length Sensitivity

Under the supplementary protocol above, we vary the maximum trajectory-segment length over $L_{\mathrm{traj}} \in \{50, 100, 200, 400\}$, using 200 as the reference setting in Table 9. At each reward-calibration stage, the collected segment count and score-labeled trajectory-pair budget are fixed at $20{,}000$ and $40{,}000$, respectively; the task configurations, five-seed evaluation protocol, and per-candidate PPO training budget remain unchanged. Adaptive trajectory-length growth is disabled. The configured length is an upper bound: episodes may terminate earlier, and padded steps are excluded from trajectory scoring and reward-model training, so only observed transitions contribute to these computations. The aim is to assess performance sensitivity to the temporal context available for executable trajectory scoring and reward-model calibration: short segments may omit task-progress information, whereas long segments may provide richer context but make transition-level credit assignment from segment-level supervision more difficult. Mean MTS and the relative gaps to $L_{\mathrm{traj}} = 200$ quantify this sensitivity on Ant, Humanoid, and Anymal without assuming that longer trajectories necessarily improve performance. Because the number of segments is fixed, the total number of collected transitions can vary across lengths; this study therefore controls the segment count and PPO epoch budget, not the total trajectory-data volume.

Effect of Trajectory Length. Table 9 examines how the temporal context used for trajectory scoring and reward calibration affects policy performance with segment counts and pair budgets held constant. Shortening L_{traj} from 200 to 50 reduces MTS by 8.67%, 5.52%, and 10.959% on Ant, Humanoid, and Anymal, respectively. At $L_{\mathrm{traj}} = 100$, these gaps narrow to 7.91%, 4.58%, and 5.479%. Extending the length to 400 instead improves performance over the default by 4.05%, 5.31%, and 9.589%, yielding the best performance among the tested settings on all three tasks. Across the tested lengths of 50, 100, 200, and 400, mean MTS increases monotonically on all three tasks. For Anymal, MTS changes from the default -0.073 to -0.081, -0.077, and -0.066 at lengths 50, 100, and 400, respectively. The lower performance at both shorter lengths is consistent with shorter segments omitting useful task-progress information, while longer segments provide additional context for trajectory scoring. Extending the horizon from 200 to 400 improves performance

Table 9: Performance of TRACE-RL with different trajectory lengths. We report mean MTS across five random seeds. The default setting uses $L_{\text{traj}} = 200$. Arrows indicate decreases or increases relative to the default, normalized by the absolute baseline MTS.

Method	Ant		Humanoid		Anymal	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL ($L_{\text{traj}} = 50$)	10.163	8.67%↓	7.846	5.52%↓	-0.081	10.959%↓
TRACE-RL ($L_{\text{traj}} = 100$)	10.248	7.91%↓	7.924	4.58%↓	-0.077	5.479%↓
TRACE-RL ($L_{\text{traj}} = 400$)	11.579	4.05%↑	8.745	5.31%↑	-0.066	9.589%↑
TRACE-RL ($L_{\text{traj}} = 200$)	**11.128**		**8.304**		**-0.073**	

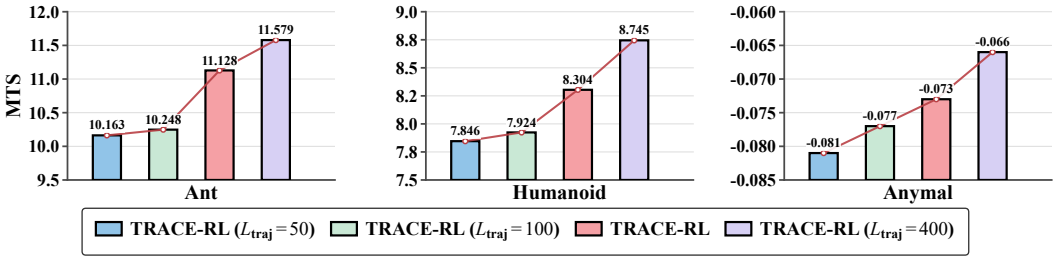

Figure 9: Effect of trajectory length on TRACE-RL performance across Ant, Humanoid, and Anymal. Bars report MTS for $L_{\text{traj}} \in \{50, 100, 200, 400\}$, where $L_{\text{traj}} = 200$ is the default. The red line connects the MTS values to highlight the trend; higher bars indicate better performance.

on all three tasks, although this does not directly establish whether transition-level credit assignment becomes easier or harder. As noted in the experimental setup, longer segments can also increase the collected data volume, so their gains cannot be attributed exclusively to temporal context. Overall, the default $L_{\text{traj}} = 200$ consistently outperforms the two shorter settings on all three tasks. Extending the maximum length to 400 provides further performance gains, but can also increase trajectory-data volume and reward-model processing requirements. These comparisons support $L_{\text{traj}} = 200$ as a practical default that balances the performance benefits of longer segments against the data and processing demands of further extending the trajectory horizon.

A.3.2 Backbone-LLM Sensitivity.

Under the supplementary protocol, we replace the default GPT-5.6-Terra backbone with alternative LLMs while retaining the task descriptions, executable function interfaces, prompt structure, and screening rules. Each backbone is responsible for generating and revising all three function types: the programmatic reward, trajectory score, and evaluation functions, thereby affecting reward guidance, supervision, and reliability assessment. The reward-model architecture, five outer iterations, six candidates per iteration, per-candidate PPO budget, and five-seed evaluation protocol remain unchanged on Ant, Humanoid, and Anymal. Table 10 compares mean MTS and relative gaps to the GPT-5.6-Terra baseline. This study asks how strongly TRACE-RL depends on the backbone used for executable function generation and whether its learning procedure remains effective with alternative backbones. It evaluates the downstream effect of changing the function-generation and revision model while keeping the underlying learning procedure otherwise unchanged, rather than assuming that general-purpose model capability directly predicts policy performance.

Effect of the Backbone LLM. Table 10 examines how the backbone used for executable function generation and revision affects policy performance under the same supplementary protocol. Replacing the default GPT-5.6-Terra with GPT-5.4-Mini reduces MTS by 2.91%, 3.32%, and 6.849% on Ant, Humanoid, and Anymal, respectively. With GPT-5.6-Luna, these gaps narrow to 1.93%, 2.18%, and 4.110%. GPT-5.6-Sol instead improves performance over the default by 2.23%, 2.96%, and 5.479%, achieving the highest mean MTS among the tested backbones on all three tasks. The consistent ordering across tasks indicates that backbone selection affects downstream policy per-

Table 10: Performance of TRACE-RL with different backbone LLMs. We report mean MTS across five random seeds. All task and training settings are kept unchanged.

Method	Ant		Humanoid		Anymal	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL(GPT-5.4-Mini)	10.804	2.91%↓	8.028	3.32%↓	-0.078	6.849%↓
TRACE-RL(GPT-5.6-Luna)	10.913	1.93%↓	8.123	2.18%↓	-0.076	4.110%↓
TRACE-RL(GPT-5.6-Sol)	11.376	2.23%↑	8.550	2.96%↑	-0.069	5.479%↑
TRACE-RL	**11.128**		**8.304**		**-0.073**	

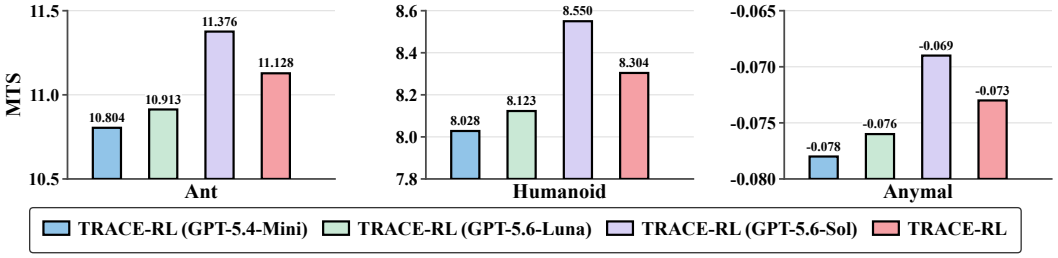

Figure 10: Effect of the backbone LLM on TRACE-RL performance across Ant, Humanoid, and Anymal. All task and training settings are kept unchanged. Bars report MTS obtained with GPT-5.4-Mini, GPT-5.6-Luna, GPT-5.6-Sol, and the default TRACE-RL backbone.

formance even when the prompts, function interfaces, screening rules, and policy-training budgets remain unchanged. This pattern is consistent with differences in the generated executable functions and their feedback-driven revisions, rather than changes to the underlying reward-calibration or policy-learning procedure. Overall, the default GPT-5.6-Terra configuration consistently outperforms the Mini and Luna alternatives, supporting its effectiveness as the function-generation backbone in TRACE-RL. The additional gains with GPT-5.6-Sol are consistent with stronger task-grounded code-generation and revision capabilities providing more effective function candidates for policy learning. Together, these results support the framework's use of LLM-based function generation and revision, with an effective default backbone and the flexibility to obtain further policy gains through backbone upgrades without changing the underlying learning procedure.

Table 11: Ablation of shared screening for the score function S and evaluation function E. Unif. abbreviates unification. Without unification, each reward group uses its own generated function instead of the shared winner.

Method	Ant		Humanoid		Anymal	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL w/o S unif.	10.766	3.25%↓	7.996	3.71%↓	-0.078	6.849%↓
TRACE-RL w/o E unif.	10.912	1.94%↓	8.119	2.23%↓	-0.076	4.110%↓
TRACE-RL w/o $(S+E)$ unif.	10.512	5.54%↓	7.796	6.12%↓	-0.079	8.219%↓
TRACE-RL	**11.128**		**8.304**		**-0.073**	

A.3.3 Shared Function Screening.

Table 11 compares the default shared screening of the score function S and evaluation function E with three variants that disable S unification, E unification, or both, with unification abbreviated as unif. in the table. When unification is disabled for a function type, each reward group uses its own generated function instead of the common screened winner; the function remains active and continues providing supervision or auxiliary reliability assessments. In the single-function variants, the

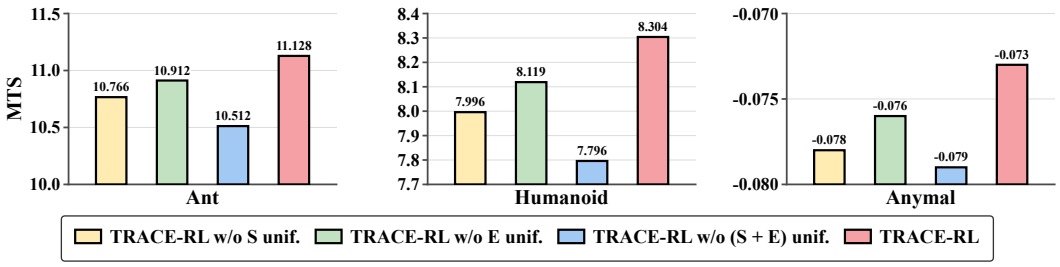

Figure 11: Effect of shared function screening in TRACE-RL across Ant, Humanoid, and Anymal. Bars report MTS for variants without score-function unification, without evaluation-function unification, without both shared-screening mechanisms, and with the full TRACE-RL configuration. Higher bars indicate better task performance.

other function type still undergoes shared screening. Reward groups retain their own policy rollouts and reward-model data, and the supplementary training and evaluation budgets remain unchanged. This experiment asks whether selecting and sharing score and evaluation functions provides consistent supervision and reliability assessment across competing reward groups, and whether these roles contribute individually or jointly. The variants assess the combined effect of common-probe selection and cross-group sharing, rather than isolating screening quality from sharing itself.

Effect of Shared Function Screening. Table 11 examines how shared screening of the score and evaluation functions affects policy performance. Disabling score-function unification reduces MTS by 3.25%, 3.71%, and 6.849% on Ant, Humanoid, and Anymal, respectively. Disabling evaluation-function unification produces smaller decreases of 1.94%, 2.23%, and 4.110%. Disabling both yields the largest reduction on each task, with gaps of 5.54%, 6.12%, and 8.219%, reducing MTS from the default 11.128, 8.304, and -0.073 to 10.512, 7.796, and -0.079, respectively. Anymal shows the largest percentage decrease relative to its own baseline in every ablation. The consistently larger losses from disabling score-function unification are consistent with the direct role of S in generating pairwise supervision for reward-model learning, whereas E provides auxiliary assessments for reward fusion. Selecting a common score function may reduce variation in supervision quality across reward groups, while selecting a common evaluation function may provide a more consistent basis for reliability assessment. Since both functions remain active in the ablated variants, the observed differences concern how they are selected and shared, rather than whether they are present. Overall, the default configuration that jointly screens and shares S and E achieves the highest mean MTS on all three tasks, outperforming both single-function variants and the variant without either form of unification. These results support retaining the complete shared-screening configuration to provide a common basis for score-derived supervision and reliability assessment across reward groups, with consistent policy-performance advantages under the tested protocol.

Table 12: Performance of TRACE-RL with different numbers of score-labeled trajectory pairs at a fixed trajectory length. The default uses $N_{\text{pair}} = 40{,}000$.

Method	**Ant**		**Humanoid**		**Anymal**	
	MTS	**Gap**	**MTS**	**Gap**	**MTS**	**Gap**
TRACE-RL ($N_{\text{pair}} = 10{,}000$)	10.552	5.18%↓	7.950	4.26%↓	-0.079	8.219%↓
TRACE-RL ($N_{\text{pair}} = 20{,}000$)	10.675	4.07%↓	8.026	3.35%↓	-0.078	6.849%↓
TRACE-RL ($N_{\text{pair}} = 60{,}000$)	11.377	2.24%↑	8.521	2.61%↑	-0.073	0.000%
TRACE-RL	**11.128**		**8.304**		**-0.073**	

A.3.4 Trajectory-Pair Supervision Budget.

At a fixed maximum trajectory length of $L_{\text{traj}} = 200$, we vary the number of score-labeled trajectory pairs per calibration stage over $N_{\text{pair}} \in \{10{,}000, 20{,}000, 40{,}000, 60{,}000\}$, with $40{,}000$ as the reference setting in Table 12. Each stage retains the same collection budget of $20{,}000$ trajectory

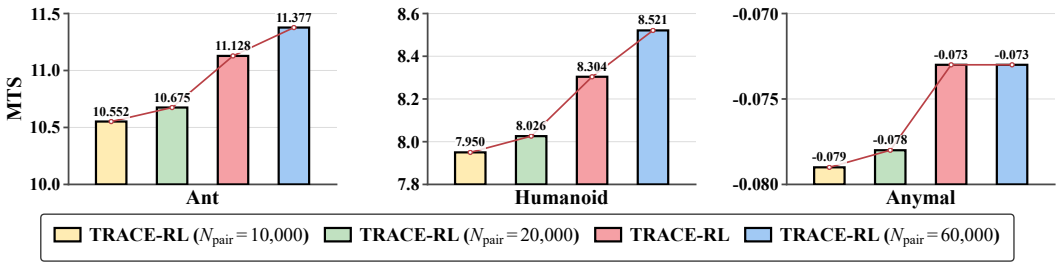

Figure 12: Effect of the score-labeled trajectory-pair budget on TRACE-RL at a fixed trajectory length. Bars show MTS on Ant, Humanoid, and Anymal for $N_{\text{pair}} \in \{10{,}000, 20{,}000, 40{,}000, 60{,}000\}$ (default: 40,000).

segments and a candidate-pair pool of 60,000; the pair-selection procedure and training–validation split ratio are unchanged. The remaining settings follow the supplementary protocol. This design varies the amount of score-derived pairwise supervision without simultaneously changing the trajectory horizon or candidate-pair pool size. The study asks whether smaller labeled datasets provide sufficient supervision for trajectory-reward calibration and whether increasing the pair budget yields further policy gains or diminishing returns. MTS and its relative gap measure the resulting policy-performance sensitivity; the experiment does not impose equal reward-model optimization work across different dataset sizes, with more pairs potentially requiring additional optimization updates.

Effect of Trajectory-Pair Supervision Budget. Table 12 examines how the number of score-labeled trajectory pairs affects policy performance at a fixed trajectory length. Reducing N_{pair} from 40,000 to 10,000 lowers MTS by 5.18%, 4.26%, and 8.219% on Ant, Humanoid, and Anymal, respectively. At $N_{\text{pair}} = 20{,}000$, these gaps narrow to 4.07%, 3.35%, and 6.849%. Increasing the budget to 60,000 instead improves Ant and Humanoid by 2.24% and 2.61%, yielding MTS values of 11.377 and 8.521, while Anymal shows no reported improvement over the default -0.073. Within the tested range, larger pair budgets therefore produce nondecreasing mean performance on all three tasks, although their benefits beyond the default depend on the task. Anymal exhibits the largest relative losses at the two reduced budgets but no additional gain at 60,000, suggesting that its supervision requirements are largely met by the default setting under this protocol. This trend is consistent with larger labeled pair sets providing more behavioral distinctions for reward calibration, whereas smaller sets may offer insufficient coverage of the available trajectory relationships. Because reward-model optimization work is not matched across dataset sizes, the observed gains reflect the supervision-budget configuration as a whole rather than data quantity alone. Overall, the default $N_{\text{pair}} = 40{,}000$ consistently outperforms the two smaller budgets and remains close to the performance obtained with 60,000 pairs. Increasing the default budget by 50% yields additional gains of 2.24% and 2.61% on Ant and Humanoid, respectively, with no improvement on Anymal at the reported precision. These results support 40,000 pairs as an effective default that balances policy performance and supervision requirements, avoiding the larger losses of reduced pair budgets while using fewer labeled pairs than the largest tested setting.

A.3.5 Online Trajectory-Reward Calibration.

Table 13 examines the roles of reward-model updating, trajectory-data refreshing, and collection-policy adaptation during the inner loop. All variants retain the initial calibration procedure in each outer iteration and differ only in subsequent stages. Fixed Collection Policy, denoted Fixed π_{collect} in the table, preserves a separate copy of the policy used for first-stage trajectory collection and uses this frozen copy to collect new reward-calibration trajectories in later stages, while continuing pairwise score labeling and reward-model fitting. The collection-policy copy is renewed at the start of each outer iteration; the PPO policy remains trainable and collects its own on-policy rollouts as usual. Relative to the default, this variant retains the per-stage trajectory-segment and score-labeled pair budgets, trajectory-length setting, and reward-model training settings, changing only the policy used to collect calibration data. Fixed $\mathcal{D}_\tau$ instead reuses the first-stage trajectory pool while continuing pairwise score labeling and reward-model fitting. No Recalibration (No Recalib. in the table) retains the first-stage trajectory pool, score-derived pairwise labels, and reward model without refreshing any of them. All variants follow the supplementary protocol's stage-wise PPO epoch bud-

Table 13: Ablation of online trajectory-reward calibration in TRACE-RL. We vary reward-model updating, trajectory-data refreshing, and the collection policy.

Method	Ant		Humanoid		Anymal	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL (No Recalib.)	9.948	10.60%↓	7.544	9.15%↓	-0.082	12.329%↓
TRACE-RL (Fixed $\mathcal{D}_\tau$)	10.239	7.99%↓	7.744	6.74%↓	-0.080	9.589%↓
TRACE-RL (Fixed π_{collect})	10.499	5.65%↓	7.900	4.87%↓	-0.078	6.849%↓
TRACE-RL	**11.128**		**8.304**		**-0.073**	

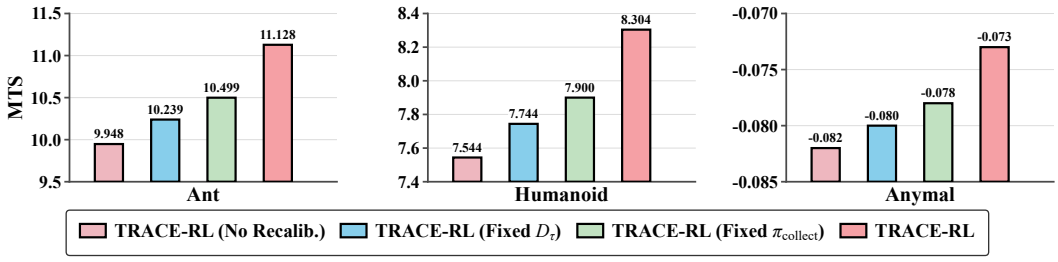

Figure 13: Ablation of online trajectory-reward calibration in TRACE-RL across Ant, Humanoid, and Anymal. Bars report MTS for variants without online recalibration, with a fixed trajectory dataset D_τ, with a fixed collection policy π_{collect}, and with the full TRACE-RL configuration.

get, outer-loop function revision, and state-transfer procedure. These controls ask whether continued fitting on reused data is sufficient, whether collecting from the evolving policy improves calibration beyond obtaining new samples from a fixed policy, and whether initial calibration alone suffices for subsequent policy learning. The comparison concerns calibration within an outer iteration, not removal of the learned reward or outer-loop revision.

Table 14: Ablation of collaborative evolution over the reward function R, score function S, and evaluation function E. The default jointly evolves all three function types.

Method	Ant		Humanoid		Anymal	
	MTS	Gap	MTS	Gap	MTS	Gap
TRACE-RL (evolve R only)	10.583	4.90%↓	7.941	4.37%↓	-0.077	5.479%↓
TRACE-RL (evolve $R+S$)	10.488	5.75%↓	7.788	6.21%↓	-0.079	8.219%↓
TRACE-RL (evolve $R+E$)	10.766	3.25%↓	8.010	3.54%↓	-0.078	6.849%↓
TRACE-RL	**11.128**		**8.304**		**-0.073**	

Effect of Online Trajectory-Reward Calibration. Table 13 examines whether effective reward adaptation requires continued reward-model fitting and trajectories that track the evolving policy. Disabling recalibration after the initial stage reduces MTS by 10.60%, 9.15%, and 12.329% on Ant, Humanoid, and Anymal, respectively, yielding the largest performance losses among the tested variants. Continuing reward-model fitting on the fixed first-stage trajectory pool narrows these gaps to 7.99%, 6.74%, and 9.589%. Collecting new trajectories from a fixed collection policy further reduces the gaps to 5.65%, 4.87%, and 6.849%, but still underperforms the complete framework on all three tasks. This consistent ordering suggests that continued fitting can recover part of the performance lost without recalibration, while additional samples provide further benefits beyond repeatedly using the initial trajectory pool. However, the remaining deficits under Fixed π_{collect} indicate that obtaining new samples from an unchanged policy does not capture the full benefit of collecting from the evolving policy. Since this variant retains the default trajectory-segment and pair-labeling budgets, its lower performance is consistent with the importance of incorporating su-

pervision from newly visited policy behaviors, rather than merely collecting additional samples. As all variants preserve the initial calibration and outer-loop revision procedures, these results support continued calibration within each outer iteration rather than relying on its initial reward model alone. Overall, the default configuration, which combines continued reward-model fitting with trajectory refreshes from the evolving policy, achieves the highest mean MTS on all three tasks. Its consistent advantage over initial calibration alone, repeated fitting on a fixed pool, and new sampling from a fixed policy supports retaining the complete online-calibration procedure. These results favor the framework's coordinated update of calibration data and reward guidance as policy behavior changes.

A.3.6 COLLABORATIVE FUNCTION EVOLUTION.

Table 14 compares joint outer-loop evolution of the reward function R, score function S, and evaluation function E with three restricted variants: evolving R only, $R + S$, or $R + E$. All variants generate the complete function group in the first outer iteration. From the second iteration onward, functions excluded from evolution retain the executable code associated with the first iteration's selected global-best candidate, while the designated function types continue to be revised. The retained functions remain operational rather than being removed. Inner-loop reward-model calibration, paired policy and reward-model state transfer, candidate count, and per-candidate training budget follow the supplementary protocol. This experiment asks whether revising the task-prior reward alone is sufficient, or whether score-derived pairwise supervision and auxiliary reliability assessment also need to evolve as policy behavior changes. Comparing the two-function variants with full evolution further examines the contribution of adapting both supporting function types together.

Effect of Collaborative Function Evolution. Table 14 examines whether adapting the task-prior reward alone is sufficient, or whether the score and evaluation functions also benefit from coordinated outer-loop revision. Evolving only R reduces MTS by 4.90%, 4.37%, and 5.479% on Ant, Humanoid, and Anymal, respectively, relative to full TRACE-RL. Evolving $R + S$ yields larger reductions of 5.75%, 6.21%, and 8.219%, whereas evolving $R + E$ produces reductions of 3.25%, 3.54%, and 6.849%. Joint evolution of all three functions therefore achieves the highest mean MTS on every tested task. However, revising an additional function does not necessarily improve performance: $R + S$ underperforms R-only evolution on all three tasks, while $R + E$ improves upon it on Ant and Humanoid but not on Anymal. This pattern suggests that the benefits of function revision depend on which supporting functions are updated together, rather than simply on the number of functions being revised. As described in the experimental setup, functions excluded from evolution remain operational, and inner-loop reward calibration and paired state transfer are retained. The experiment thus evaluates restrictions on outer-loop adaptation rather than the removal of reward-learning or reliability-assessment components. The results are consistent with the importance of keeping task guidance, score-derived pairwise supervision, and auxiliary reliability assessment aligned as policy behavior changes, although policy scores alone do not identify the specific mechanism underlying the differences between restricted variants. Overall, jointly evolving R, S, and E yields the highest mean MTS on all three tasks, whereas neither reward-only evolution nor either two-function variant matches the complete framework. This consistent advantage supports the default strategy of revising the complete function group, allowing task-prior rewards, score-derived supervision, and reliability assessment to adapt together as policy behavior changes.

A.3.7 CANDIDATE-POOL SIZE.

Table 15 varies the number of candidates per outer iteration, N_{cand}, using six candidates as the reference configuration. This setting controls both the number of reward groups trained and the sizes of the score-function and evaluation-function candidate pools. Each variant retains five outer iterations, $1{,}000$ PPO epochs per reward candidate per iteration, and the same trajectory-collection, trajectory-pair labeling, and reward-calibration settings per group. The task set and five-seed evaluation protocol remain unchanged. The experiment asks how the breadth of function search affects the best policy obtained, whether smaller pools retain sufficient candidate diversity, and whether expanding the pool yields further gains beyond the default setting. Because the per-candidate budget is fixed, larger pools require more aggregate policy training; this is a search-scale sensitivity study, not an equal-total-compute comparison. Mean MTS and relative gaps are referenced to the six-candidate supplementary baseline.

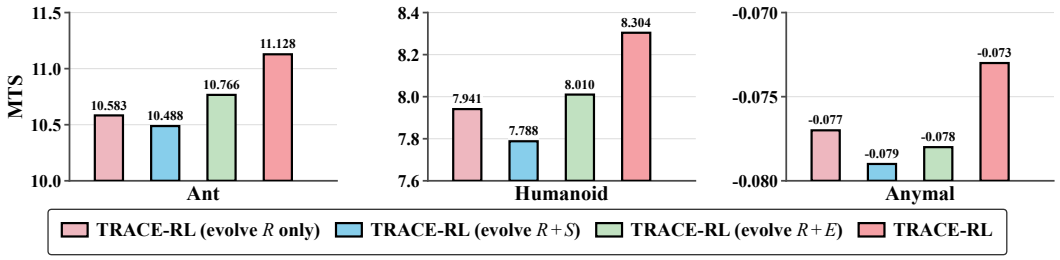

Figure 14: Ablation of collaborative evolution over the reward function R, score function S, and evaluation function E. Bars report MTS when evolving only R, evolving R together with S, evolving R together with E, and using the full TRACE-RL configuration.

Table 15: Performance of TRACE-RL with different numbers of candidates per outer iteration. The default setting uses $N_{\text{cand}} = 6$. Arrows indicate decreases or increases relative to the default, normalized by the absolute baseline MTS.

Method	**Ant**		**Humanoid**		**Anymal**	
	MTS	**Gap**	**MTS**	**Gap**	**MTS**	**Gap**
TRACE-RL ($N_{\text{cand}} = 2$)	9.789	12.03%↓	7.403	10.85%↓	-0.083	13.699%↓
TRACE-RL ($N_{\text{cand}} = 4$)	10.328	7.19%↓	7.767	6.47%↓	-0.079	8.219%↓
TRACE-RL ($N_{\text{cand}} = 8$)	11.436	2.77%↑	8.555	3.02%↑	-0.070	4.110%↑
TRACE-RL ($N_{\text{cand}} = 6$)	**11.128**		**8.304**		**-0.073**	

Effect of Candidate-Pool Size. Table 15 examines how the breadth of outer-loop function search affects policy performance. Reducing N_{cand} from the default 6 to 2 lowers MTS by 12.03%, 10.85%, and 13.699% on Ant, Humanoid, and Anymal, respectively. With $N_{\text{cand}} = 4$, these gaps narrow to 7.19%, 6.47%, and 8.219%. Increasing the candidate count to 8 instead improves performance over the default by 2.77%, 3.02%, and 4.110%, yielding the highest mean MTS among the tested configurations on all three tasks. This pattern is consistent with broader search providing more opportunities to identify effective reward programs and suitable score and evaluation functions, whereas smaller pools restrict the alternatives available for selection. However, candidate diversity is not measured directly, and the setting jointly changes the reward, score, and evaluation pools, so the results do not isolate the contribution of expanding each function pool. As specified in the experimental setup, the training budget per candidate remains fixed; larger pools therefore require more aggregate policy training, and their gains should not be interpreted as improvements under an equal total computational budget. Overall, the default $N_{\text{cand}} = 6$ consistently outperforms the smaller pools across all three tasks. Expanding to eight candidates yields further gains of 2.77%–4.110%, but requires more aggregate policy training under the fixed per-candidate budget. Within the range of candidate counts evaluated here, these comparisons support six candidates as a practical default for reward search and screening of score and evaluation functions, balancing policy performance against the training requirements of broader search.

A.4 Detailed Experimental Setup

A.4.1 Benchmark Overview

We summarize the 18 GPU-accelerated benchmark environments in Figure 16 and Table 16. The table reports the observation and action dimensions, task descriptions, environment task-score definitions, and success criteria used for evaluation.

A.4.2 Task Environments and Success Criteria

Table 16 summarizes the 18 task environments used in our experiments. We report the observation and action dimensions exposed to the policy, together with the environment task score or success

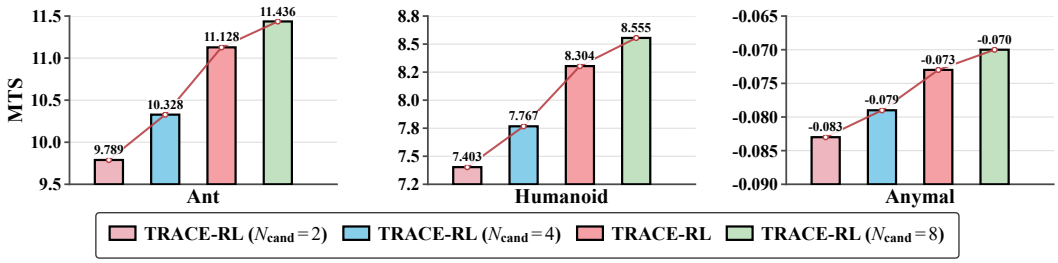

Figure 15: Effect of candidate-pool size on TRACE-RL performance across Ant, Humanoid, and Anymal. The setting jointly controls the number of reward groups and the sizes of the score-function and evaluation-function candidate pools. Bars report MTS under $N_{\text{cand}} \in \{2, 4, 6, 8\}$, where the default setting uses $N_{\text{cand}} = 6$.

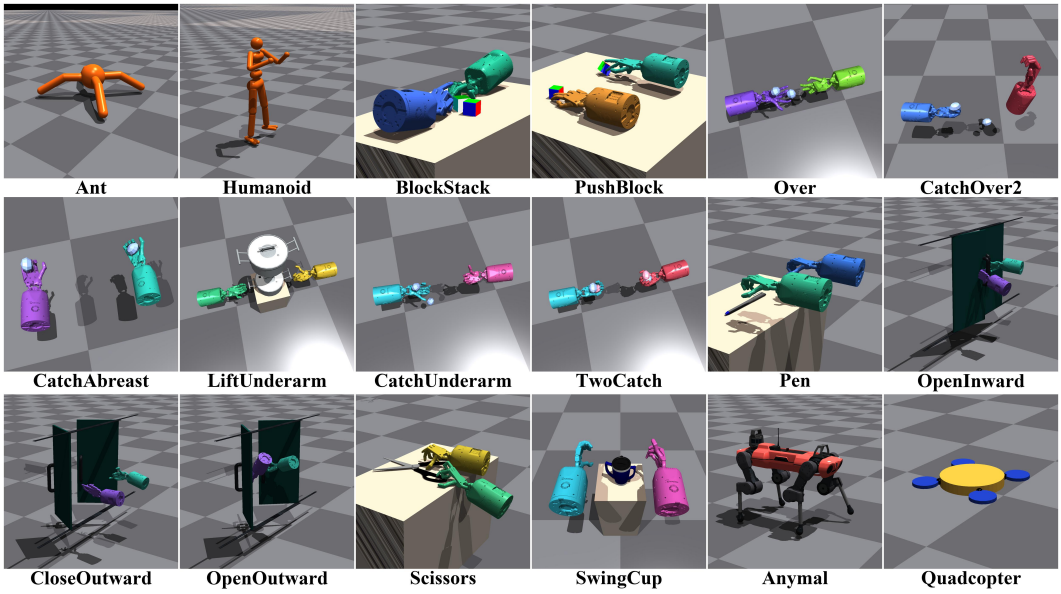

Figure 16: Representative views of the 18 robotic skill-learning benchmarks used in our experiments, covering locomotion, whole-body control, and dexterous manipulation tasks.

condition used for evaluation. For the manipulation tasks, distance thresholds are expressed in meters and rotation thresholds in radians to clarify their physical interpretation.

All compared methods use the same simulator task definition, PPO implementation, policy-training budget, and declared seed set for each benchmark. For information parity, all LLM-based methods receive the same task description, environment interface, and environment-defined task-score or success specification. This task-score specification is part of the declared task semantics rather than a hidden evaluation oracle; numerical outcomes are not shared across methods or independent runs, but candidate feedback may be aggregated within a search run according to the algorithm.

TRACE-RL uses the environment-defined task metric through explicitly separated interfaces. Its symbolic definition is included in the prompts for generating the three executable function types and contains task semantics but no measured trajectory outcomes. Task-score rankings on common screening probe pairs, constructed once from the initial rollout pool of a fixed anchor candidate, are used only to select the shared score and evaluation functions. Screening reuses this existing pool without additional trajectory collection. The anchor policy inherits the current global-best policy when available and is randomly initialized otherwise, including the first outer iteration and any iteration following only failed iterations. Separately, each reward candidate partitions pairs constructed from its own rollouts into reward-model training and validation sets. The candidate-specific validation rankings determine the direct reliability terms q_{t}^{T} and q_{p}^{T}, while evaluation-function assessments on the same validation pairs determine q_{t}^{E} and q_{p}^{E}. The screening probe pairs and candidate-specific

validation pairs are constructed separately and serve different roles; they are not treated as a single shared held-out set. Online task-score traces are used for plateau detection and structured feedback, whereas candidate-level aggregate task scores are reserved for outer-loop selection and MTS reporting. These aggregate scores do not supervise the reward-model objective.

Table 16: Benchmark task environments, interfaces, and evaluation criteria used in TRACE-RL. Observation and action dimensions describe the policy interface.

Benchmark Environments		
Environment (obs., act.)	**Task description**	**Environment task score / success criterion**
Ant (60, 8)	Run forward as quickly as possible.	`mean(potentials - prev_potentials)` (logged as `progress_reward`)
Humanoid (108, 21)	Run forward as quickly as possible with a humanoid body.	`mean(potentials - prev_potentials)` (logged as `progress_reward`)
Anymal (48, 12)	Track randomly sampled linear and yaw velocity commands with a quadruped.	`mean(-(linvel_error + angvel_error))`
Quadcopter (21, 12)	Reach and hover near a fixed target position.	`mean(-target_dist)`
BlockStack (428, 52)	Coordinate two Shadow Hands to stack two blocks into a stable tower.	`1[left_dist < 0.07 and right_dist < 0.07 and height_gap < 0.03]`
PushBlock (428, 52)	Use two Shadow Hands to push two blocks to their respective target positions.	`1[left_dist <= 0.1 and right_dist <= 0.1]`
Over (398, 40)	Throw and pass an object between two opposing fixed-base Shadow Hands.	`1[dist < 0.03]`
CatchOver2 (422, 52)	Perform an overhand-to-underarm object transfer between two Shadow Hands.	`1[dist < 0.03]`
CatchAbreast (422, 52)	Transfer an object between two side-by-side Shadow Hands.	`1[dist < 0.03]`
LiftUnderarm (417, 52)	Grasp a pot handle with two hands and lift it while maintaining horizontal stability.	`1[goal_dist < 0.05]`
CatchUnderarm (422, 52)	Transfer an object between two movable, upward-facing Shadow Hands.	`1[dist < 0.03]`
DoorOpenInward (417, 52)	Open a door inward through bimanual handle manipulation.	`1[door_handle_gap > 0.5]`
Pen (417, 52)	Open a pen cap using two Shadow Hands.	`1[5 * handle_dist > 1.5]`
TwoCatchUnderarm (446, 52)	Simultaneously transfer two objects to the opposite-hand targets.	`1[goal_dist + goal_another_dist < 0.06]`

Benchmark Environments (continued)		
Environment (obs., act.)	**Task description**	**Environment task score / success criterion**
DoorCloseOutward (417, 52)	Close a door outward through bimanual handle manipulation.	`1[door_handle_gap < 0.5]`
DoorOpenOutward (417, 52)	Open a door outward through bimanual handle manipulation.	`1[door_handle_gap > 0.5]`
Scissors (417, 52)	Coordinate two Shadow Hands to open a pair of scissors.	`1[dof_pos > -0.3]`
SwingCup (417, 52)	Hold a dual-handled cup with two hands and swing it to the target orientation.	`1[rot_dist < 0.785]`

A.4.3 Training, Evaluation, and Task-Score Protocol

Task-Score Availability and Roles. Environment dynamics remain fixed while the training reward is recalibrated between stages. The environment-defined task score is computed online after each rollout and is available throughout training. It is used for plateau detection, shared screening of score and evaluation functions, candidate-specific reliability assessment, outer-loop candidate selection, and MTS reporting. The aggregate task score is not directly used as the PPO reward or as a regression target for the reward model; the task-specific training reward R_{s} is defined separately. Thus, TRACE-RL assumes access to an environment-defined task metric during training, rather than treating it as a hidden post-training oracle.

Distinguishing Task-Aligned Signals. The reported task score is the environment's online metric with task-specific aggregation; MTS takes its maximum within each training run, and results are then aggregated across seeds. The offline trajectory KPI is recomputed from recorded segments to provide reference rankings for function screening and the task-score channel of reliability estimation. For tasks such as TwoCatchUnderarm and BlockStack, this KPI augments success criteria with distance-based progress terms, so it is not simply the reported success rate evaluated offline. The LLM-generated score function S instead returns pairwise scoring margins that are converted into soft targets for reward-model training. The auxiliary reward R_{s} is constructed from per-environment success or progress signals, with a task-provided reward signal as a fallback; it is independently normalized and weighted before being added to R_{f} for PPO. Thus, the reported metric, offline KPI, and R_{s} reflect the same task objective but differ in aggregation and numerical construction; the generated score function supplies the distinct pairwise supervision.

A.4.4 Implementation Details and Hyperparameters

Unless otherwise specified, TRACE-RL generates six candidates for each executable function type at every outer iteration and performs five outer iterations, using the LLM configuration reported in the main experimental setup with a temperature of 1.0. Each candidate receives a total budget of 3000 PPO epochs per outer iteration. The reported default inner-loop allocation follows a progressive $500/1000/1500$-epoch schedule. Plateau-based stage adaptation may reallocate the remaining epochs across stages, including consuming the reserved budget of later stages, but never increases the total budget; consequently, a later trajectory-calibration stage may receive fewer epochs or no epochs if an earlier stage continues improving. We retain the task-specific PPO architectures and optimizer configurations provided by the corresponding Isaac Gym and Bi-DexHands environments (Makoviychuk et al., 2021; Chen et al., 2024), and use the same PPO configuration for all methods on a given task. At each reward-calibration stage, TRACE-RL collects up to $20{,}000$ trajectory segments with a maximum length of 200, constructs a pool of $60{,}000$ candidate pairs, and selects $40{,}000$ pairs through BALD-based acquisition (Houlsby et al., 2011); 15% of the selected pairs are reserved for validation. The trajectory-calibrated reward is represented by a transition-level MLP $f_\phi(s_t, a_t, s_{t+1}, d_t)$ with hidden dimensions $[256, 256, 128]$, LayerNorm, ReLU activations, and a dropout rate of 0.1. The reward model is optimized using Adam with a learning rate

of 3×10^{-4}, weight decay 10^{-4}, and training and validation batch sizes of 160. At each calibration stage, the reward model is trained for at most 250 epochs, with a minimum of 20 epochs and early-stopping patience values of 30 and 10 for the initial and subsequent stages, respectively. The task-score and evaluation-function assessments are weighted equally when both are available. The task-specific environment reward is added only when constructing the final PPO training reward. Its effective coefficient is bounded above by 0.05 and is automatically reduced when its normalized contribution would otherwise dominate the fused dense reward. Experiments are executed on a server equipped with two NVIDIA GeForce RTX 4090 GPUs with 24 GB of memory each.

A.4.5 Baselines and Ablation Controls

The comparison suite includes sparse environment rewards, official or human-engineered dense rewards, LLM reward-program generation, and reward–policy co-evolution baselines. All reported Sparse, Human, Eureka, ROSKA, and PoRSE results were obtained by rerunning the baselines under the shared task configurations, matched PPO-training budgets, and five-seed protocol described above; no baseline values were copied from the original papers. Our main-text ablations remove reward signals, replace reliability-aware fusion with fixed equal weighting, or disable state inheritance; the R_{p} ablation additionally disables shared score- and evaluation-function screening. The main-text ablations retain their baseline's task interface, PPO allocation, and evaluation rule; supplementary studies follow their separately specified protocol. Budget matching concerns PPO training, excluding additional trajectory collection and reward-model calibration, and does not imply equal total environment interactions or compute.

A.5 Task-Score Specification and Shared Information Access

Source and Objective Alignment. The environment-defined task score provides an explicit, computable specification of task progress or success and is distinct from the executable trajectory score function S generated by the LLM. As summarized in Table 16, these human-specified metrics use simple expressions over task-relevant environment quantities: forward progress for Ant and Humanoid, negative velocity-tracking error for Anymal, negative target distance for Quadcopter, and threshold-based success indicators for manipulation tasks. These definitions directly reflect the stated task objectives without prescribing a complete dense reward or an action sequence for solving the task. The environment task metric remains fixed while TRACE-RL adapts its reward guidance and executable functions. It therefore specifies the desired outcome, whereas the framework addresses how to guide policy learning toward that outcome.

Equal Access and Permitted Use. The environment-defined task score is a common source of task feedback available to all compared methods, rather than information reserved for TRACE-RL. Under the comparison protocol, each method is permitted to compute and use this metric on its own training rollouts through the same task definition and environment interface. Shared availability does not mean that measured outcomes are exchanged across methods or independent runs. TRACE-RL uses this feedback for progress monitoring, function screening, reliability assessment, and candidate selection, as detailed in Appendix A.4.3. These uses represent algorithmic choices about how to exploit commonly available information within the shared task setting, not exclusive access to an additional supervisory oracle. Other methods may use the same feedback according to their respective learning procedures; equal information access does not require identical information utilization. The task score is thus an explicitly available training signal, rather than a hidden test-only signal whose use would cross a training–test information boundary.

Human Specification and Automation. Human involvement in defining the task metric concerns specifying the desired objective and its computable criterion, rather than providing expert annotations for individual trajectory pairs or manually revising rewards throughout training. This specification is part of the shared task setup, not additional human supervision supplied exclusively to TRACE-RL. Given a task description and environment interface, LLMs could also assist in translating the intended criterion into executable metric code, subject to validation of semantic correctness, variable availability, and numerical behavior to ensure faithful implementation of the intended objective. Automating this process is complementary to our framework but is not evaluated or claimed as a contribution here. TRACE-RL instead focuses on using an available task metric to support

adaptive reward guidance as policy behavior evolves; the availability of a meaningful, computable task metric is an explicit assumption shared by the comparisons.

A.6 Algorithm Description

Algorithms 1 and 2 together summarize TRACE-RL. Algorithm 1 generates and trains candidates for one outer iteration; Algorithm 2 calls this procedure, handles failed iterations, and maintains the global-best state. At each outer iteration, the LLM proposes group-specific reward functions and candidate pools for score functions and evaluation functions, using the retained global-best candidate as the common reference when available. Each reward group first initializes or transfers its model states and collects its initial current-policy rollouts. Once all Round-1 collections are available, TRACE-RL reuses the initial rollout pool of a fixed anchor candidate as the shared screening trajectory source, without additional trajectory collection. One score function and one evaluation function are selected on the resulting shared trajectory pairs and then used by all reward groups during that outer iteration. Each reward group then partitions its own rollouts into reward-model training and validation pairs. The candidate-specific validation pairs, rather than the shared screening pairs, are used to estimate fusion reliability. The fused dense reward combines the programmatic task prior with the trajectory-calibrated reward, while the task-specific environment reward is added separately to form the final reward used by PPO. When policy improvement reaches a plateau, TRACE-RL updates the current-policy trajectories, score-derived pairwise labels, and reward model, and then refreshes the rewards used in the next stage. When at least one valid candidate completes the configured inner-loop procedure, the best valid candidate from the current iteration is compared with the retained global-best candidate, and the higher-scoring candidate becomes the single global reference for both the next LLM revision and state transfer. This shared reference links subsequent function revisions and model initialization to the same selected candidate and its accumulated training evidence. Reported MTS values instead cover the full prescribed training run, including the settling period excluded from candidate selection. The remaining diagnostics are retained as structured feedback for the next LLM revision. The two algorithms distinguish the fused dense reward $R_{\mathrm{f},i}$ from the final policy reward $R_{\mathrm{PPO},i}$. The score and evaluation functions affect reward-model calibration and the initial reliability assessment, but neither replaces the environment task score used for candidate selection. During a stage transition, the updated reward model refreshes $R_{\mathrm{t},i}$ and hence both rewards used by PPO. The operator $\mathrm{UpdateBest}$ retains the candidate state only when its selection score exceeds m_{gb}^{k-1}. This separation avoids a circular selection criterion while allowing reward guidance to adapt to newly visited policy behaviors.

A.6.1 Shared-Function Screening Details

In each outer iteration, every reward group first initializes or transfers its state and collects its initial current-policy rollouts. Once all Round-1 collections are available, TRACE-RL reuses the initial rollout pool D_a of a fixed anchor candidate ($a = 1$) as the shared screening source, setting $D_{\mathrm{scr}}^{k} \leftarrow D_a$ without additional trajectory collection. The anchor policy follows the same transfer-or-initialize procedure as the other candidates: it inherits the current global-best policy when available and is randomly initialized otherwise. The screening rollout source is used to construct a common probe set containing $20{,}000$ trajectory pairs with a fixed sampling seed. We first sample a preliminary pool 1.5 times this size and compute the absolute environment task-score difference for every pair. Using the 30th and 80th percentiles of these differences as cutoffs, the final probe set is stratified to contain 30% near-tie, 50% medium-gap, and 20% large-gap pairs. The same probe pairs and task-score differences are reused for every score-function and evaluation-function candidate in the corresponding pool, ensuring comparisons use the same behavioral evidence. For candidate j in outer iteration k, $A_{\mathrm{p},j}^{k}$ is the fraction of non-tied probe pairs for which the sign of the candidate's output agrees with the sign of the environment task-score difference. $A_{\mathrm{r},j}^{k}$ is the Spearman correlation between the candidate outputs and task-score differences over the same non-tied pairs. We define the large-gap subset (termed the hard-pair subset in the main text) as the non-tied pairs whose absolute task-score differences are at or above the 75th percentile among all nonzero differences in the final probe set; $A_{\mathrm{h},j}^{k}$ is the pairwise sign accuracy on this subset. Candidates are ranked lexicographically by $A_{\mathrm{p},j}^{k}$, $A_{\mathrm{r},j}^{k}$, and $A_{\mathrm{h},j}^{k}$, with exact ties resolved by their fixed candidate order. Screening is performed independently for the score-function and evaluation-function pools.

Algorithm 1 TRACE-RL: Candidate Generation and Training

Require: environment $\mathcal{E}$, LLM operator $\mathcal{L}_{\mathrm{LLM}}$, candidates N, outer-iteration index k, task context, global reference $\mathcal{C}_{\mathrm{gb}}^{k-1}$, feedback feedback^{k-1}

$\left(\{\mathcal{R}_i^k\}_{i=1}^N, F_S^k, F_E^k\right) \leftarrow \mathcal{L}_{\mathrm{LLM}}(\text{context}, \mathcal{C}_{\mathrm{gb}}^{k-1}, \text{feedback}^{k-1})$
for $i = 1, \ldots, N$ **do**
  $(\pi_i, f_{\phi_i}) \leftarrow \mathrm{TransferOrInitialize}(\mathcal{C}_{\mathrm{gb}}^{k-1})$
  $D_i \leftarrow \mathrm{Collect}(\mathcal{E}, \pi_i)$
end for
$a \leftarrow 1$ (fixed anchor candidate)
$D_{\mathrm{scr}}^k \leftarrow D_a$
$P_{\mathrm{scr}}^k \leftarrow \mathrm{ScreenPairs}(D_{\mathrm{scr}}^k)$
$S^{k,*} \leftarrow \mathrm{Screen}(F_S^k, P_{\mathrm{scr}}^k)$
$E^{k,*} \leftarrow \mathrm{Screen}(F_E^k, P_{\mathrm{scr}}^k)$
$G^k \leftarrow \{(\mathcal{R}_i^k, S^{k,*}, E^{k,*})\}_{i=1}^N$
for $i = 1, \ldots, N$ **do**
  $(P_i^{\mathrm{tr}}, P_i^{\mathrm{val}}) \leftarrow \mathrm{PairAndSplit}(D_i)$
  $Y_i \leftarrow S^{k,*}(P_i^{\mathrm{tr}})$
  $f_{\phi_i} \leftarrow \mathrm{Calibrate}(f_{\phi_i}, P_i^{\mathrm{tr}}, Y_i)$
  $\alpha_i \leftarrow \mathrm{FuseWeight}(f_{\phi_i}, \mathcal{R}_i^k, E^{k,*}, P_i^{\mathrm{val}})$
  $\beta_i \leftarrow 1 - \alpha_i$
  $R_{\mathrm{f},i} \leftarrow \alpha_i R_{\mathrm{t},i} + \beta_i R_{\mathrm{p},i}$
  $\bar{R}_{\mathrm{s},i} \leftarrow \mathrm{NormalizeClip}(R_{\mathrm{s},i})$
  $\lambda_{\mathrm{s},i} \leftarrow \mathrm{SparseWeight}(R_{\mathrm{f},i}, \bar{R}_{\mathrm{s},i})$
  $R_{\mathrm{PPO},i} \leftarrow R_{\mathrm{f},i} + \lambda_{\mathrm{s},i} \bar{R}_{\mathrm{s},i}$
  while the prescribed inner-loop budget remains **do**
    $\pi_i \leftarrow \mathrm{PPO}(\pi_i, R_{\mathrm{PPO},i})$
    if recent task-score improvement reaches a plateau **then**
      $D_i \leftarrow \mathrm{Collect}(\mathcal{E}, \pi_i)$
      $(P_i^{\mathrm{tr}}, P_i^{\mathrm{val}}) \leftarrow \mathrm{PairAndSplit}(D_i)$
      $Y_i \leftarrow S^{k,*}(P_i^{\mathrm{tr}})$
      $f_{\phi_i} \leftarrow \mathrm{Calibrate}(f_{\phi_i}, P_i^{\mathrm{tr}}, Y_i)$
      refresh $R_{\mathrm{t},i}$, $R_{\mathrm{f},i}$, $\lambda_{\mathrm{s},i}$, and $R_{\mathrm{PPO},i}$
      continue in the next stage
    end if
  end while
  record (V_i^k, M_i^k, Z_i^k): completion status, task score, pairwise ranking statistics, reward diagnostics, fusion behavior, and execution records
  $\mathcal{C}_i^k \leftarrow (\mathcal{R}_i^k, S^{k,*}, E^{k,*}, f_{\phi_i}, \pi_i)$
end for
Return: $\{(\mathcal{C}_i^k, V_i^k, M_i^k, Z_i^k)\}_{i=1}^N$

A.6.2 Reward-Model Objective

TRACE-RL generates score-derived pairwise supervision with an executable score function applied to current-policy trajectories and refreshes this supervision when the policy reaches new behavior regions. For each trajectory pair (ξ_n^a, ξ_n^b), the score function returns a signed pairwise scoring margin, whose sign indicates which segment ranks higher and whose magnitude expresses the strength of that ranking. We clip this margin to $[-20, 20]$ and convert it with a unit-temperature sigmoid into a soft supervision target, encoding preference strength rather than assigning hard labels. The reward model f_ϕ predicts transition-level rewards from the state, action, next state, and termination indicator, and sums them to obtain each segment score. This formulation uses comparisons between trajectory segments to supervise dense reward learning without requiring separate reward annotations for individual transitions. The model is trained with soft-label binary cross-entropy together with a scale regularizer of weight 0.1, which keeps the mean absolute transition reward near one and facilitates stable fusion with the programmatic task-prior reward. We disable the optional regression loss to the programmatic reward in all reported experiments, so that the calibrated reward model can learn behavior distinctions that are not already encoded by the task prior.

Algorithm 2 TRACE-RL: Candidate Selection and Failure Handling

Require: environment $\mathcal{E}$, LLM operator $\mathcal{L}_{\text{LLM}}$, task context, outer iterations K, candidates N

$(\mathcal{C}_{\text{gb}}^0, m_{\text{gb}}^0) \leftarrow (\emptyset, -\infty)$
feedback$^0 \leftarrow \emptyset$
for $k = 1, \ldots, K$ **do**
  run Algorithm 1 with $\mathcal{E}$, $\mathcal{L}_{\text{LLM}}$, N, k, task context, $\mathcal{C}_{\text{gb}}^{k-1}$, and feedback^{k-1}
  receive $\{(\mathcal{C}_i^k, V_i^k, M_i^k, Z_i^k)\}_{i=1}^N$
  if $V_i^k = 0$ for all $i = 1, \ldots, N$ **then**
    log the failed iteration
    if strict completion is required **then**
      terminate with failure
    end if
    $(\mathcal{C}_{\text{gb}}^k, m_{\text{gb}}^k) \leftarrow (\mathcal{C}_{\text{gb}}^{k-1}, m_{\text{gb}}^{k-1})$
    feedback$^k \leftarrow$ feedback^{k-1}
    continue to the next outer iteration
  end if
  $i_k^* \leftarrow \arg\max_{i:\, V_i^k = 1} M_i^k$
  $\mathcal{C}_k^* \leftarrow \mathcal{C}_{i_k^*}^k$
  $(\mathcal{C}_{\text{gb}}^k, m_{\text{gb}}^k) \leftarrow \text{UpdateBest}\left(\mathcal{C}_{\text{gb}}^{k-1}, m_{\text{gb}}^{k-1}, \mathcal{C}_k^*, M_{i_k^*}^k\right)$
  feedback$^k \leftarrow \text{Summarize}(\{Z_i^k\}_{i=1}^N, \mathcal{C}_{\text{gb}}^k)$
end for
if $\mathcal{C}_{\text{gb}}^K = \emptyset$ **then**
  Output: failure (no validated candidate)
else
  Output: global-best validated state $\mathcal{C}_{\text{gb}}^K$
end if

A.6.3 Reward Adaptation and Transfer Details

Reliability Estimation. Fusion reliability is estimated from large-gap held-out trajectory pairs, defined as those whose nonzero absolute environment task-score differences fall in the top quartile. For each dense reward, TRACE-RL measures the fraction of these pairs on which its cumulative reward ordering agrees with the environment task-score ordering. It computes a second agreement rate against the evaluation function on the same pairs for which that function returns a valid, nonzero difference. The two agreement rates are weighted equally when both are available. If the evaluation function is invalid, its assessment is omitted; if a reward produces a non-informative constant output, it receives the neutral reliability value 0.5. Spearman correlations are retained only as diagnostics and do not affect fusion. Because both agreement rates use the same held-out pairs and task specification, the evaluation-function result is treated as auxiliary consistency evidence rather than as an independent source of statistical evidence.

Adaptive Fusion-Weight Update. The trajectory-calibrated reward coefficient α is determined from its reliability relative to that of the programmatic task-prior reward. Starting from the neutral value 0.5, TRACE-RL scales the normalized reliability difference by a gain of 2. When both rewards are close to chance-level agreement, the coefficient is pulled back toward 0.5. A further evidence gate suppresses an increase in α when the trajectory-calibrated reward differences have a standard deviation below 0.05 or correlate weakly with the programmatic reward differences, using 0.20 as the correlation reference. Each update combines 30% of the preceding coefficient with 70% of the new estimate to moderate abrupt changes in the reward mixture; the programmatic coefficient is its complement, $\beta = 1 - \alpha$. The initial coefficient is 0.5.

Task-Specific Environment Reward. The task-specific environment reward is normalized independently and clipped to $[0, 1]$ before being added to the fused dense reward to form the final PPO reward. Its coefficient is at most 0.05 and is reduced whenever the 90th-percentile magnitude of its contribution would exceed 7.5% of the corresponding fused dense-reward magnitude in the current vectorized PPO batch. If the environment-reward percentile is effectively zero, the upper-bound

coefficient is retained because the percentile-level contribution is already zero. This reward is added only after dense-reward fusion and therefore neither participates in estimating α nor changes the relative balance between R_{t} and R_{p}.

Stage-Transition Rule. The default PPO allocation contains three stages with base budgets of 500, 1000, and 1500 epochs, for a total of 3000 epochs. TRACE-RL measures recent progress as the difference between the best environment task score observed within the latest 300 epochs and the best score observed before that window. After the base budget of a nonfinal stage has been consumed, progress is checked every 100 epochs. Once three earlier measurements are available, their 75th percentile becomes the reference gain. The method enters the next stage when the current gain is less than 35% of that reference, making the threshold relative to previously observed improvements. If the reference gain is nonpositive, transition instead requires a nonpositive current gain. One qualifying check is sufficient. A stage that continues to improve may borrow unused epochs from later stages, including all of the remaining future-stage budget, while the total PPO budget remains unchanged; in this case, a later trajectory-calibration stage may receive no budget.

Candidate Selection Score. For candidate i in outer iteration k, the quantity M_i^k in Algorithm 2 is the highest task-specific environment score recorded across its completed inner-loop stages. The validity indicator V_i^k restricts selection to candidates that successfully complete the configured inner-loop procedure, and, when this set is nonempty, the valid candidate with the largest M_i^k is selected. For candidate selection only, the first 100 PPO epochs of an inherited policy are excluded as a settling period; reported MTS values include the full prescribed training run. Exact ties are resolved by candidate order. Pairwise ranking agreement, evaluation-function outputs, and reward-model diagnostics are retained for LLM feedback but do not enter the selection score.

Failed-Iteration Handling. If no candidate satisfies $V_i^k = 1$, TRACE-RL does not evaluate the empty-set $\arg\max$ or update the global best. By default, it logs the failed iteration, retains the previous global-best state and LLM feedback, and continues to the next outer iteration within the fixed K-iteration budget without adding extra iterations to compensate for failures. If no valid reference exists yet, subsequent candidates are initialized from scratch, and screening reuses the fixed anchor candidate's initial rollouts. If no valid candidate is found after all K iterations, the run terminates with an explicit failure instead of returning a policy. When strict completion is required, an iteration with no valid candidate instead terminates the run immediately.

State-Transfer Protocol. TRACE-RL retains the reward-model and policy states of the best valid candidate found up to each outer iteration. Every candidate in the next iteration starts from a copy of this paired state. Full inheritance denotes reuse of learned model states as initialization, not a guarantee against forgetting during subsequent optimization. Reward-model transfer restores all model parameters, observation-normalization statistics, and Adam optimizer state. Policy transfer restores the complete actor–critic parameters, normalization statistics, and PPO optimizer state, while resetting the epoch counter so that every candidate receives the prescribed training budget. If no valid reference candidate exists, both models are initialized from scratch.

A.7 MTS and Best-Policy Retention

Our evaluation targets the best task performance discovered within a prescribed training budget, and MTS directly matches this objective and the policy-checkpoint selection mechanism in TRACE-RL. During policy training, whenever the environment-defined task score exceeds the running best, the trainer saves the policy checkpoint associated with that epoch. The retained best checkpoint is updated upon further improvement and remains available when subsequent training scores decrease. Candidate selection and cross-iteration policy inheritance prioritize these best checkpoints, while the latest checkpoint is used to continue optimization within the inner loop. Thus, the policy retained by the algorithm is selected through a best-checkpoint procedure, rather than being determined solely by the final training epoch. This distinction makes MTS an appropriate primary criterion for assessing reward search: it measures the highest observed task performance within the available budget and is explicitly linked to checkpoint retention and reuse. Moreover, the score is defined by the environment and shared across methods, so comparisons do not depend on the scale or composition of their learned or programmatic rewards. Under the matched benchmark training budgets, we

compute MTS separately for each run and report its mean and standard deviation across five independent seeds. Together, these design choices align the reported metric with the goal of discovering and retaining high-performing policies, while avoiding dependence on the particular optimization state reached at the last epoch.

A.8 Computational Overhead and Budget Alignment

TRACE-RL controls its auxiliary computational cost through bounded trajectory collection and lightweight reward-model calibration. All methods use the same task-specific PPO configuration and policy-training budget; for LLM-based comparisons, we additionally align the LLM generation configuration, reward-candidate count, and number of outer iterations. At each calibration stage, the current policy is executed without policy-gradient updates to collect up to $20{,}000$ trajectory segments, each containing at most 200 steps, using $4{,}096$ parallel environments in our Isaac Gym implementation. In one representative Anymal run on our RTX 4090 server, collecting this pool required approximately 6.1 minutes initially and 1.3–1.9 minutes in subsequent stages. Shared function screening reuses the initial trajectory pool. The reward model is a compact transition-level MLP with hidden dimensions $[256, 256, 128]$, trained with early stopping and a maximum of 250 epochs per stage; calibration took approximately 1.6–4.6 minutes per stage in the same run. These timings illustrate the practical cost of the auxiliary stages and are task-dependent. LLM-generated functions are executed locally for trajectory scoring and evaluation, avoiding a separate remote LLM request for each trajectory. Together, these design choices limit the additional work required to adapt rewards to the current policy's behavior. This overhead is an explicit methodological cost of trajectory-guided reward adaptation, while the matched PPO budget controls the amount of policy optimization allocated to each method. Accordingly, our results establish performance gains under matched policy-training budgets; additional trajectory collection, reward calibration, and method-specific function generation are accounted for separately from that budget. Therefore, the additional overhead has a clearly specified budget upper bound, remains at the scale of minutes per calibration stage in our measured runs, and directly serves the core reward-adaptation mechanism rather than introducing unrelated computation. Given that this overhead is bounded, remains at the scale of minutes, and directly supports current-policy reward adaptation, we consider the additional computation and wall-clock time to be an acceptable and proportionate trade-off for the capability it provides.

A.9 Discussion about Closely Related Methods

We compare TRACE-RL with related methods in terms of supervision sources, optimized objects, reward-update timescales, and transferred states, highlighting how these choices shape the learning process. The following comparisons clarify methodological differences.

Reward-Program Search and Selection. **Eureka** iteratively revises executable reward programs using policy-training feedback, while keeping each candidate's reward code fixed during evaluation (Ma et al., 2024). **REvolve** evolves reward programs using human feedback (Hazra et al., 2025), while **RF-Agent** organizes candidate generation and refinement through language-agent tree search (Gao et al., 2025). $\mathbf{R}^*$ separates reward-structure evolution from parameter alignment (Li et al., 2025a), and **ORSO** allocates policy-training samples among candidate shaping rewards through online selection (Zhang et al., 2024). These mechanisms refine reward programs, optimize their parameters, or allocate training resources among candidates. Program-level refinement alone, however, does not provide a separate learned reward representation that continuously adapts within a fixed executable function group. **TRACE-RL** addresses this gap through score-supervised calibration of a transition-level reward model during candidate training. While the executable functions remain fixed, the score function supplies supervision from fresh current-policy trajectories to update the learned reward. The policy-training reward can therefore adapt to newly visited behaviors without waiting for the next program revision. The executable program supplies task structure, while the learned model incorporates behavioral evidence collected as the policy explores additional states. Taken together, existing reward-program search frameworks have generally emphasized program-level revision, whereas the combination of fixed executable functions with continuous, score-supervised calibration of a separate reward model appears less explored. In this work, **TRACE-RL** investigates this combination during candidate training. This design separates program

revision from reward-model adaptation, retaining explicit task guidance while allowing trajectory-driven refinement throughout policy learning.

Policy Transfer, Exploration, and Experience Reuse. **ROSKA** co-evolves reward and policy populations through policy fusion and selection (Huang et al., 2025). **PoRSE** combines policy transfer with task-relevant exploration bonuses (Chang et al., 2026), whereas **LaRes** uses experience relabeling and constrained policy transfer during reward search (Li et al., 2025b). These mechanisms emphasize the reuse of policies and experience, together with improved exploration. Preserving and updating a learned reward representation across executable-function revisions introduces an additional requirement beyond policy transfer. **TRACE-RL** addresses this requirement by jointly inheriting the reward-model and policy states of the same global-best candidate for subsequent outer iterations. Both transferred states originate from a common training history, providing a shared starting point for subsequent policy and reward learning. When the executable functions change, the inherited reward model continues learning under the revised score function, retaining its existing representation as the basis for recalibration. This paired transfer carries forward both the policy and the learned reward representation, allowing accumulated reward knowledge to incorporate updated score supervision. **TRACE-RL** explores the paired inheritance of reward-model and policy states from the same global-best candidate, together with score-based recalibration, across successive iterations of executable reward search.

Score-Based Supervision and Reward Calibration. **PEBBLE** learns rewards from teacher preferences and relabels replay-buffer experience, whereas **Self-Alignment** adjusts LLM-proposed reward parameters. **CoRe** learns residual rewards from video-level VLM feedback, and **PrefCLM** aggregates executable trajectory evaluators into preference labels (Lee et al., 2021; Zeng et al., 2024; Ni et al., 2026; Wang et al., 2025). These methods use generated functions as reward evaluators, synthetic teachers, or parameterized reward components to supervise reward learning or refine existing rewards. **TRACE-RL** uses an executable *score function* dedicated to persistent trajectory-reward calibration. The score function processes two fresh current-policy trajectories and produces a signed score margin, whose magnitude determines the strength of graded soft supervision. This supervision updates a separate transition-level reward model as the policy visits new behaviors. Candidate score functions are screened for alignment with environment task-score rankings on common probe pairs, and the selected function is shared across reward groups. A separate *evaluation function* assesses the reliability of programmatic and trajectory-calibrated rewards on held-out trajectory pairs to guide reward fusion. Trajectory scores thus guide the learned reward model, whereas evaluation outputs inform how reward sources contribute to subsequent policy optimization. In our formulation, the executable score function provides graded soft supervision for persistent trajectory-reward calibration. By assigning supervision generation to the score function and fusion assessment to the evaluation function, **TRACE-RL** supports adaptive reward guidance while retaining explicit task priors.

Summary of Methodological Novelty. TRACE-RL links two adaptation timescales through a persistent trajectory-calibrated reward model, combining behavioral feedback with executable function evolution. Within an outer iteration, fresh policy trajectories update this model under fixed executable functions to incorporate evidence from newly visited behavior regions. Across iterations, structured training feedback jointly revises the reward, score, and evaluation functions, while the same global-best candidate anchors function revision and paired reward-model and policy-state transfer. This coupling connects trajectory-level reward calibration to continued policy optimization as reward specifications change, without treating each revision as an isolated restart. Tables 1 and 2 examine the reward-guidance and state-transfer components under the reported benchmark protocol.

A.10 Prompt Templates

The following boxes summarize the invariant instructions in the three prompt templates; task-specific observation mappings, function signatures, category guidance, and metric definitions are inserted through the displayed placeholders at runtime. The placeholder `{task_score_reference}` contains only the task-specific metric definition and the observable-variable mapping required to implement it. It contains no trajectory-specific score, candidate-selection result, or reported MTS value. Although the same semantic reference is provided when generating the three executable function types, the resulting score and evaluation functions receive

only the trajectory observations and actions exposed by their runtime interfaces; they cannot query online PPO logs, candidate-level aggregate scores, or final reported results.

Programmatic Task-Prior Reward Function: System Prompt

You are a reward engineer designing an executable reward function for a reinforcement learning task. Your goal is to write a reward function that helps the agent learn the task described in the user message. Use only variables available in the exact reward-function signature: {task_reward_signature_string}. Treat the exact signature and the environment task-score reference in the user message as the source of truth. When the signature exposes a task-progress, success, reset, or potential-difference signal, use it as the primary objective. Observation-only terms may provide shaping, but must not replace the task objective. The environment task score specifies task semantics; do not assume access to any external task-score oracle that is not provided by the reward-function signature. Do not invent observation layouts, thresholds, or environment inputs. The function will be decorated with @torch.jit.script, so use TorchScript-compatible operations, Torch tensors rather than NumPy, and tensors on the same device as the inputs.

Programmatic Task-Prior Reward Function: User Prompt

Environment code and available reward-function inputs: {task_obs_code_string}
Task description: {task_description}
Environment task-score reference: {task_score_reference}
Write an executable programmatic task-prior reward function for this task. Use only the variables in the provided function signature and return a dense transition-level reward suitable for PPO optimization.

Executable Trajectory-Scoring Function: System Prompt

You write a NumPy executable trajectory-scoring function for reinforcement learning. It provides offline pairwise supervision for a trajectory-calibrated reward model; it is not a PPO step reward. Use one public function: compute_score(obs1, acts1, obs2, acts2). Rank the two trajectories within each pair primarily according to the environment task-score reference supplied in the user message. This reference is used only as an offline trajectory-level supervision target and must not be invoked as the PPO step reward. Apply a hard viability rule only when the task reference defines one; otherwise preserve the task-score ordering. Handle invalid or short arrays safely and return a bounded scalar logit. Do not use Torch, helper-function templates, invented observation fields, or unavailable environment state. {label_category_guidance} Required skeleton: {task_label_signature_string}

Executable Trajectory-Scoring Function: User Prompt

Environment observation code and available trajectory fields: {task_obs_code_string}
Task description: {task_description}
Offline environment task-score reference: {task_score_reference}
Write the Executable Trajectory-Scoring Function as compute_score. Include: 1) a short comment block mapping observation indices to physical meaning; and 2) pairwise ranking logic aligned with the provided offline task-score reference.
The output is used only to construct score-derived pairwise supervision for the trajectory-calibrated reward model.

Reliability-Aware Fusion Evaluation Function: System Prompt

You write a NumPy offline trajectory task-score evaluator for reliability-aware reward fusion. It does not generate reward-model training labels or a PPO step reward. For compatibility with the runtime interface, use one public function: compare_task_kpi(obs1, acts1, obs2, acts2). Compute the same task-score proxy for both trajectory segments, apply only a task-defined hard viability gate, and return a bounded difference between the two segment scores. The function is used only on held-out trajectory pairs to assess the relative reliability of the trajectory-calibrated reward and the programmatic task-prior reward. Do not use tanh, smoothness or posture tie-breakers, helper functions, Torch, invented observation fields, or training-label tie-break logic. {evaluator_category_guidance} Required skeleton: {task_evaluator_signature_string}

Reliability-Aware Fusion Evaluation Function: User Prompt

Environment observation index reference: {task_obs_code_string}
Task description: {task_description}
Held-out environment task-score reference: {task_score_reference}
Write the Reliability-Aware Fusion Evaluation Function as compare_task_kpi. Include: 1) a short comment block mapping observation indices to physical meaning; 2) a per-segment task-score proxy aligned with the provided reference; and 3) a hard viability gate only when it is explicitly defined by the task.
This function is evaluated only on held-out trajectory pairs to assess fusion reliability. Its output must not be used as a PPO step reward, and the function must not implement training-label tie-break logic.