A BIRD'S-EYE VIEW OF ITERATIVE REWARD DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing effective reward functions in RL typically requires substantial expertise and trial and error. Recent work automates this process with LLM-based systems that generate and iteratively improve reward code using policy feedback. However, these methods are often hard to compare because they differ in implementation details, feedback assumptions, and evaluation environments. To address this, we introduce a **B**enchmark for **I**terative **R**eward **D**esign (**BIRD**) that expresses existing methods in a unified configuration and evaluation space. This lets us compare algorithms directly, ablate individual design choices, and prototype new components under matched feedback conditions and policy-training budgets. Across MuJoCo, Meta-World, Assistax, and HumanoidBench, we identify a small set of simple design choices that consistently improve performance. Combining these choices yields significantly better performance than the evaluated methods from prior work. Our results highlight the strength of simple baselines and motivate further study of when additional algorithmic complexity improves iterative reward design.

1 INTRODUCTION

A central difficulty in reinforcement learning (RL) is designing reward functions that support efficient learning while remaining aligned with designer intent. At one extreme, a reward may be easy to optimize but poorly aligned: shaping terms and proxy rewards can improve sample efficiency but may bias the objective (Ng et al., 1999; Pan et al., 2022) or induce reward hacking (Amodei et al., 2016; Skalse et al., 2022). At the other extreme, a reward may be well aligned but difficult to optimize: binary completion flags in robotics, for example, can be too sparse to support efficient policy learning without heuristics (Andrychowicz et al., 2017). Navigating this tradeoff often requires trial and error, which is unsurprisingly both laborious and error-prone even for expert designers (Booth et al., 2023).

This process, known as *iterative reward design* (IRD), follows a basic loop: propose a reward, train a policy, evaluate its behavior, revise the reward, and repeat as needed (Figure 1).[1] To reduce the burden on human designers, methods like Eureka (Ma et al., 2024a) and Text2Reward (Xie et al., 2024) have proposed using LLMs to automate reward generation and revision. These rewards typically far surpass human-written ones and have thus inspired a large body of work on LLM-assisted IRD (Hazra et al., 2025; Gao et al., 2025; Ma et al., 2025; Lee et al., 2026; Jaswal et al., 2026).

The growing literature offers a broad menu for generating, evaluating, and updating reward functions, but provides limited guidance about which design choices work reliably. Within papers, benchmarks and ablations typically test design choices within particular algorithms and environments, leaving unclear whether those choices also improve other algorithms, remain effective in other environments, or retain their benefits when combined. Across papers, differences in feedback assumptions, harness implementations, model backbones, and policy-training procedures make it difficult to compare methods fairly or isolate the effects of individual choices. Moreover, higher benchmark performance does not by itself establish better alignment with the intended objectives. These limitations leave four central questions unresolved: (1) *Which IRD methods perform best across tasks?*; (2) *Which design choices drive performance?*; (3) *Can useful design choices be combined into better methods?*; and (4) *Does better performance mean better alignment?*

To address these questions, we develop a **B**enchmark for **I**terative **R**eward **D**esign (**BIRD**), built around a shared, configurable implementation of the IRD loop. BIRD represents each method as a configuration specifying how to generate candidate rewards, train and evaluate policies, and select and

[1] Sometimes humorously referred to as "graduate student descent."

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.

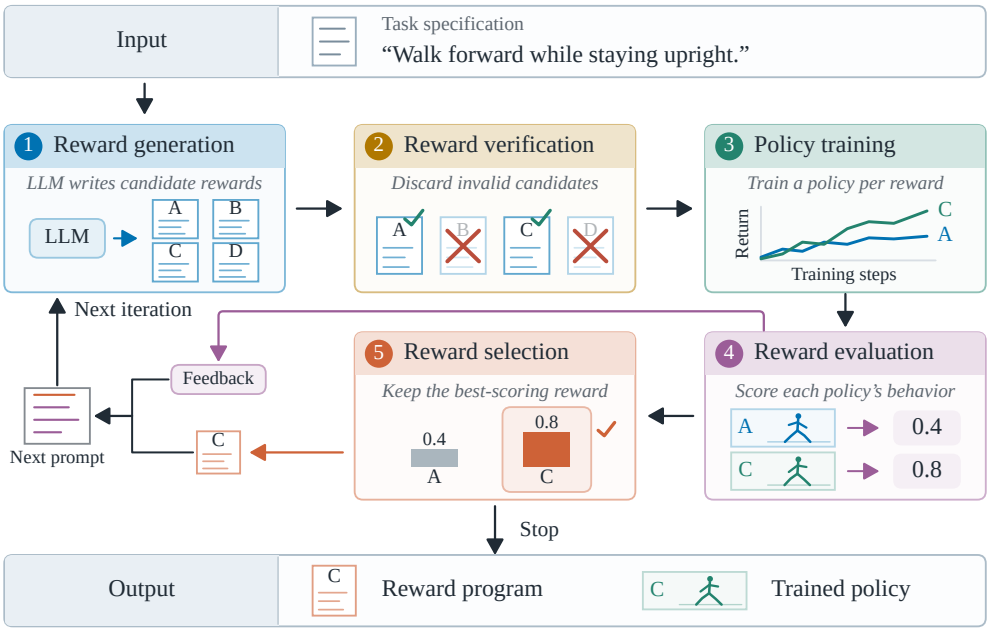


Figure 1: Most iterative reward design (IRD) methods follow this basic loop. BIRD uses this shared structure to represent methods in a common format, making them easier to compare.

revise rewards. When methods share a design choice, they reuse its implementation. Researchers can therefore reproduce existing methods, modify individual choices, or combine choices from different methods within the same framework. BIRD pairs this implementation with a common evaluation protocol, enabling comparisons under matched feedback conditions and policy-training budgets.

Our contributions are fourfold.

1. We introduce BIRD (Section 3), a unified framework for implementing, comparing, ablating, and developing IRD methods, with implementations for 14 published methods (Appendix C) and evaluation procedures for MuJoCo (Todorov et al., 2012), Meta-World (Yu et al., 2019), Assistax (Hinckeldey et al., 2025), and HumanoidBench (Sferrazza et al., 2024). Code and data will be released.
2. We compare 6 existing methods (Section 4.1) and ablate 30 design choices (Section 4.2) under matched experimental conditions. We find that many design choices, both from the literature and our own proposals, produce small or inconclusive effects when deployed across new tasks and algorithmic stacks, and that only a small set of simple design choices clearly helps. This suggests that intuition-driven design choices often fail to transfer across tasks and methods, and that systematic empirical benchmarking is a necessary complement.
3. Using BIRD, we develop two novel IRD algorithms ERA-U and ERA-S (unsupervised and supervised variants) that achieve significantly better performance over established baselines in their respective categories on the environment suites (Section 4.3).
4. To preliminarily assess IRD alignment, we conduct a small human study ($n = 2$). Consistent with prior work on specification gaming (Krakovna et al., 2020), we find that rewards designed by IRD methods can induce policies that score highly on benchmarks through unnatural or even harmful behavior (Section 4.4).

2 Preliminaries

We assume most readers have a passing familiarity with RL and so omit the usual MDP and policy definition regurgitation; interested readers and those new to the field can consult Sutton & Barto (2018). Instead, we focus on *what* iterative reward design is and *how* the process works.[2]

[2]We have written a "truly marvelous [prior work], which this margin is too narrow to contain" (Appendix B).

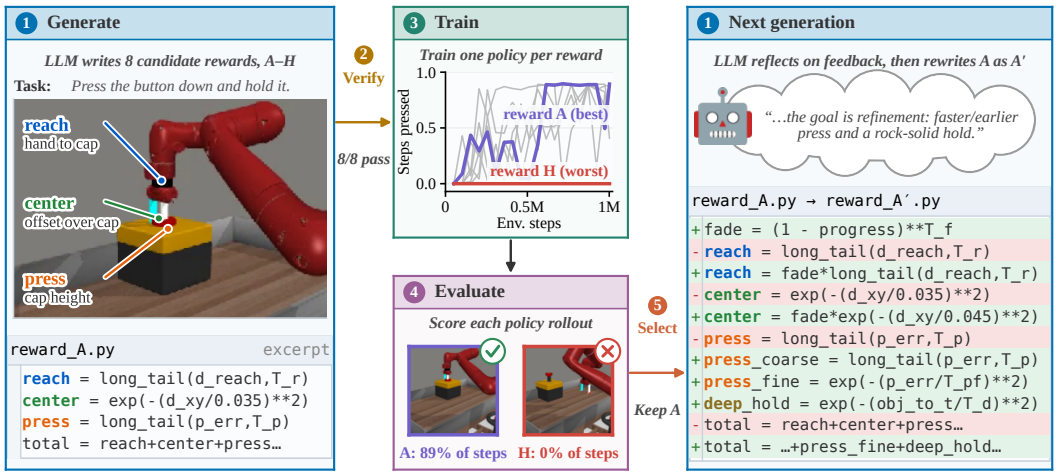


Figure 2: **One iteration of Eureka on Meta-World's button-press task.** The LLM generates eight reward functions, all of which pass verification. Each reward is used to train a PPO policy for 1M environment steps. Reward A produces the best policy (89% task success), while H produces the worst (0%). Eureka greedily retains A and provides the LLM with feedback on how its reward components (e.g., `reach`, `press`, and `center`) evolve during training. The LLM uses this feedback to revise A into A'. All results are from an actual run.

> *Iterative reward design (IRD) is the process of designing rewards by trial and error based on policy feedback, often using LLMs or VLMs.*

Before walking through the stages, we strongly encourage readers to skim the terminology table in Appendix A, which clarifies frequently reused terms such as "design choice" and "reference reward." As illustrated in Figure 1, IRD algorithms can typically be decomposed into the following stages.

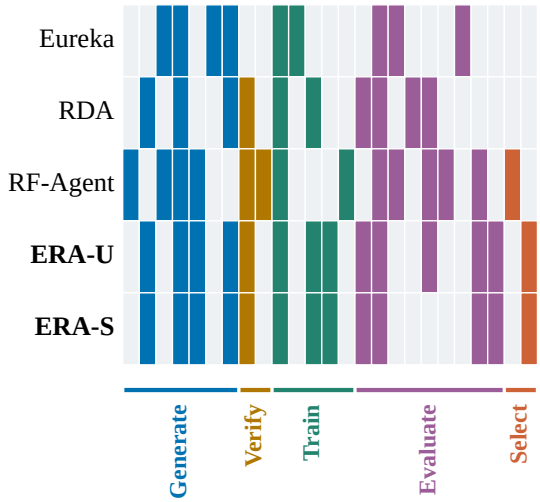
1. Reward generation. The process starts with an LLM or VLM proposing candidate rewards from a task description and information about the environment, often its source code. Any reward shipped with the environment (its *reference reward*) is withheld, so the model designs rewards from scratch. Rewards are usually represented as Python functions (see Figure 2). On the first iteration, the model works only from what it is given about the task; on later iterations, it can also draw on feedback from earlier attempts.

2. Reward verification. Training a policy under every generated reward can get expensive, so some methods first screen candidates to decide which are worth training on. Checking for coding errors is particularly useful when using weaker models. Other checks try to estimate reward quality before training a policy. CARD, for example, retains a buffer of trajectories from earlier iterations and validates candidate rewards by checking whether they rank successful trajectories above failed ones (Sun et al., 2025).

3. Policy training. The surviving candidates then move on to policy training, which is essentially vanilla RL. One or more policies are trained under each reward function. The design choices at this stage are fairly standard: which RL algorithm to use (e.g., PPO or SAC), how many environment steps to train for, and whether to train from scratch or warm-start from an earlier policy.

4. Reward evaluation. In the next stage, methods determine how well each reward worked. Here methods split into two kinds. *Supervised* IRD methods use the benchmark score, which grades trained policies against the task (Appendix A); Eureka, for example, combines it with statistics about reward components over the course of training (Ma et al., 2024a). *Unsupervised* methods never access the benchmark score: RDA, for example, replaces it with a VLM that rates task completion from image frames sampled uniformly across a policy rollout (Lee et al., 2026).

5. Reward selection. After curating information about each candidate, methods then select a winner or a set of elite candidates. A common design choice is to greedily keep the highest-scoring candidate, as in Eureka and RDA (Ma et al., 2024a; Lee et al., 2026). Other methods use populations or search



eureka.yaml

```yaml
generate:
  n_candidates: 8
  history_mode: last_iteration
verify:
  enabled: false
train:
  init: from_scratch
evaluate:
  fitness:
    source: native
select:
  rule: argmax_fitness
```

Figure 3: **BIRD represents IRD methods as configurations.** *Left:* Each row is a method and each column is a design choice key, grouped by the stages from Section 2. Gray cells represent design choices set by the BIRD default (e.g., `init=from_scratch`); colored cells are design choices set by the algorithm (e.g., Eureka's `n_candidates=8`). As we explain in Section 4.3, ERA-U and ERA-S use the RDA configuration as a backbone. *Right:* Each configuration is a short YAML file. An excerpt from Eureka's BIRD configuration.

trees to keep more candidates in play (Hazra et al., 2025; Gao et al., 2025). The comparison itself can also be based on preferences: REvolve uses human preferences (Hazra et al., 2025), while Gran Turismo[3] uses VLM preferences and then collects human feedback on the selected policy (Ma et al., 2025). These choices determine which rewards become the starting point for the next round of generation.

3 BIRD

As we explain in the introduction, IRD methods are hard to compare because they typically have different implementations, backbones, and assumptions. The key insight of BIRD is that these difficulties can be largely overcome by leveraging the shared structure of the IRD loop (Section 2).

Following Unifloral (Jackson et al., 2025), we represent IRD methods as "configurations." Each configuration specifies the choices made at each IRD stage: which feedback to collect, how to select candidates, and so on (Figure 3). When methods share a choice, they reuse its implementation. For example, Eureka and RDA use different feedback sources but make the same selection choice. Their BIRD configurations therefore specify different scoring signals but use the same ranking code. This helps prevent implementation differences (e.g., differences in tie-breaking logic) from becoming confounders.

BIRD currently represents 14 IRD methods (Appendix C). To the best of our knowledge, our implementations are correct. We verify our implementations against the original papers and reference source code; however, not all papers open-source their code, and some model backbones have been deprecated, making complete fidelity difficult, if not impossible.

4 Experimental Results

We use BIRD to investigate four key questions.

[3] Ma et al. (2025) do not name their method, so we refer to it by its environment: Gran Turismo.

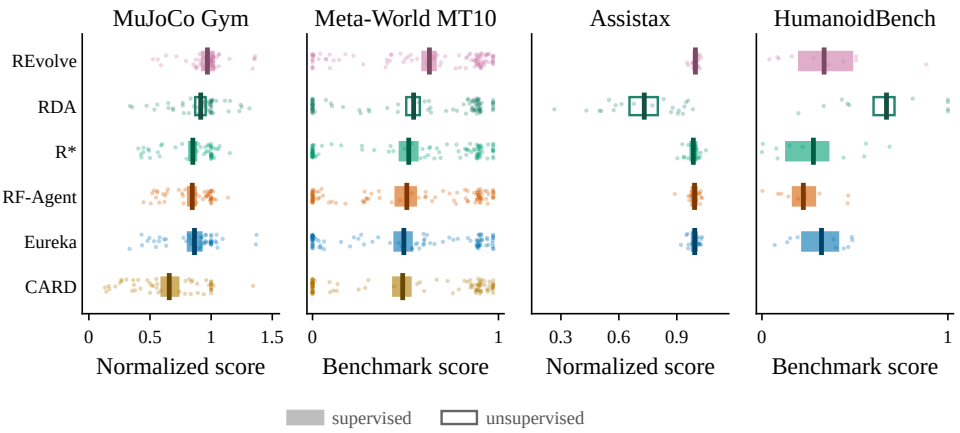


Figure 4: **Comparing IRD methods across four environment suites.** Each seed represents one run on an environment. Ticks show mean across tasks and seeds. Bars in this plot and all others are 95% BCa bootstrap confidence intervals. Experimental details are in Appendix I.

1. **Which methods work best?** We compare methods from prior work (Appendix C) across MuJoCo (Todorov et al., 2012), Meta-World MT10 (Yu et al., 2019), Assistax (Hinckeldey et al., 2025), and HumanoidBench (Sferrazza et al., 2024) under matched policy-training budgets.
2. **Why do those methods work?** We isolate individual design choices to distinguish broadly useful components from improvements that depend on a particular task or configuration.
3. **Can we combine useful choices?** We evaluate whether a small set of simple design choices provides an effective recipe for reward design.
4. **What does doing well mean?** We examine the alignment between benchmark scores and human judgments of whether the task is accomplished and how it is performed.

4.1 Comparing IRD Methods

We first use BIRD to compare 6 published IRD methods. We find that REvolve performs the strongest across all suites, only falling behind RDA in HumanoidBench; no other method consistently outperforms any other with non-overlapping CIs, suggesting these methods are surprisingly similar performance-wise. We provide per-task performance and compute efficiency plots in Appendices H.1 and H.2.

As we elaborate in Section 4.4, we find RDA—the only unsupervised method—performs more inconsistently on Assistax and HumanoidBench. We hypothesize that this reflects the difficulty of assessing task completion from visual feedback alone. For example, determining whether a button has been pressed is relatively straightforward, whereas determining whether a robot has successfully bathed a bedridden patient may require information that is difficult to infer purely visually.

We also observe diminishing returns from additional iterations. This is likely because most Python reward functions are represented as weighted sums and subsequent iterations mainly tune coefficients rather than introduce large structural changes (see excerpt in Figure 2).[4] For further evidence, see the EPIC reward distance (Gleave et al., 2021) analysis in Appendix F.

4.2 Ablating IRD Design Choices

The natural next direction after comparing IRD methods is investigating which design choices drive performance. To investigate, we ablate 30 choices drawn from existing algorithms and our own proposals. We select these manually, as exhaustively evaluating BIRD's more than 300 configurable choices is computationally infeasible.

[4]Some methods like RF-Agent develop countermeasures to avoid incremental iteration (Gao et al., 2025).

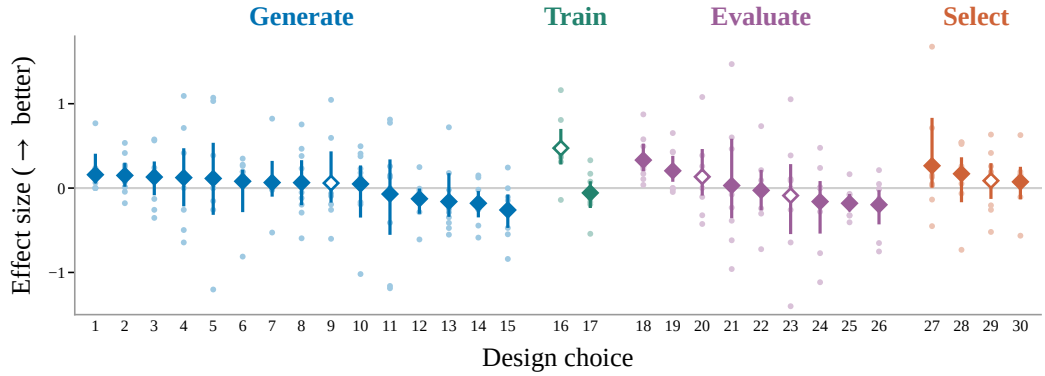


Figure 5: **Most design choices show no clear benefit.** Effects of enabling 30 design choices across applicable baselines (Eureka, RDA, CARD) and nine tasks. Hollow diamonds mark choices introduced in BIRD. Only four choices clearly help: (16) peak own-reward checkpoint, (18) summarized feedback, (19) task-success fitness, and (1) four-turn history. See Appendix E for all choices and experimental details.

We find that most choices do not consistently improve performance over the baseline, including many of our own proposals (Figure 5). Several of the strongest choices are simple, such as greedily retaining checkpoints and summarizing feedback. Together, these results suggest that additional algorithmic complexity does not reliably improve IRD.

The variability across tasks may partly explain why promising choices do not produce consistent gains. More broadly, the difficulty of manual reward design (Booth et al., 2023) gives reason to question whether human intuitions about reward revision reliably translate into effective automated methods.

4.3 A Simple Recipe for IRD

The ablations in Section 4.2 suggest that the choices that help reliably tend to be simple. We therefore asked whether stacking a few simple choices on an existing method is enough to outperform the methods we compared. Starting from the RDA configuration, we hill-climbed on four unsaturated

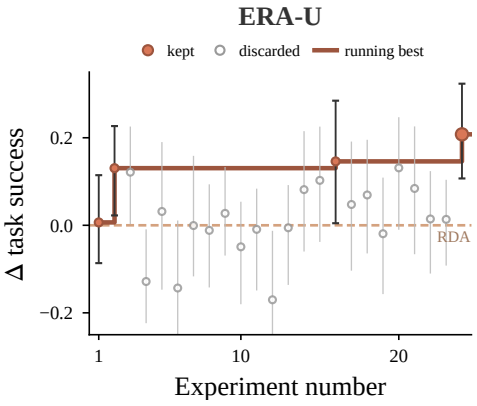


Figure 6: **Hill-climbing unsupervised IRD.** We tried 24 configurations on four unsaturated MT10 tasks and stopped at +0.21 benchmark score over RDA without trying to climb further. Bars: 95% BCa intervals over 3-7 seeds per experiment. Full details in Appendix I.5.

Table 1: **The ERA recipes** (U: ERA-U, S: ERA-S). ✔ added to RDA; ✓ inherited. ERA-S and ERA-U differ in exactly one key, the source of the signal used to rank candidates (shaded).

			Ours	
	Design choice	RDA	U	S
Gen.	Task decomposition	✓	✓	✓
	Design thought (RF-Agent)	–	✔	✔
Train	Warm start	✓	✓	✓
	Peak own-reward ckpt.	–	✔	✔
Eval.	Per-subtask feedback	✓	✓	✓
	Continuous score, ×2	–	✔	✔
	States shown to judge	–	✔	✔
	Selection: VLM completion rating	✓	✓	–
	Selection: reference reward	–	–	✔
Sel.	Greedy selection	✓	✓	✓
	Keep incumbent unless beaten	–	✔	✔

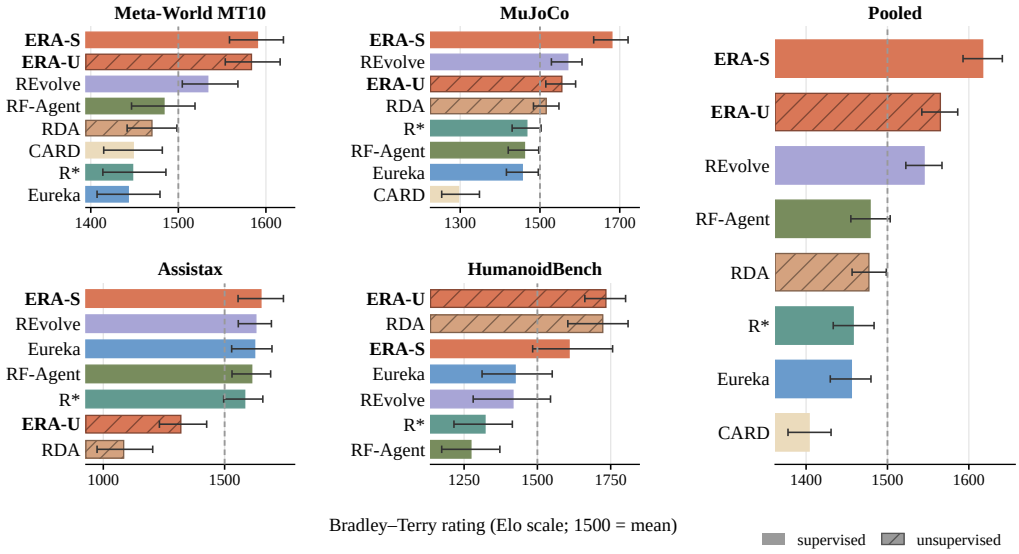


Figure 7: **ERA-S and ERA-U rank first and second overall.** Bradley–Terry ratings on the Elo scale (mean 1500, dashed line) for each benchmark and pooled over all 21 tasks. Within each task, every seed of one method is compared with every seed of another; the higher benchmark score wins and ties count as half a win. Intervals resample seeds with the task set held fixed. Hatched bars mark unsupervised methods. CARD is run only on MuJoCo and MT10.

Meta-World MT10 tasks (pick-place, push, peg-insert-side, and drawer-open), changing one design choice at a time. To keep the search cheap, most configurations ran at a reduced budget of 24–30 policy trainings per search. We kept the search deliberately small and stopped after 24 configurations, when the last one, re-run at the full budget of 40 trainings, led RDA by more than 0.2 in benchmark score (Figure 6). The result, ERA-U, adds five choices to RDA (Table 1), including the peak own-reward checkpoint that helped most clearly in Figure 5.

To obtain a supervised recipe, we did no further search. ERA-S is ERA-U with a single key flipped: the one that sets the ranking signal, from the VLM completion rating to the reference reward (shaded rows of Table 1).

Although ERA-U was hill-climbed on only four MT10 tasks and ERA-S was not tuned at all, both recipes transfer well. Under our matched protocol, they take first and second place in the pooled Bradley–Terry ranking (Figure 7): ERA-S outperforms REvolve, the strongest supervised baseline, and ERA-U outperforms RDA, its unsupervised parent, both with non-overlapping CIs. The ranking holds when the four development tasks are excluded, and when all seven tasks used at any point during development are excluded (Appendix K). ERA-U, which never sees the benchmark signal, also ranks above every supervised method except REvolve. As we discuss in Section 5, we present ERA-U and ERA-S not as methods to adopt in their own right (we expect further search to surpass them substantially), but as evidence that simple choices and a small hill-climb go a long way in IRD.

4.4 Beyond Benchmark Scores

Higher benchmark scores are only useful if they reflect the behavior the designer intended. To check this, two authors rated policy rollouts for *completion* (was the task achieved?) and *naturalness* (was it achieved the way the rater expected?). Details of the rating process can be found in Appendix J. Although this is admittedly a small study, our claims are a negative result about low naturalness that we expect the majority of people would agree with. We find that none of the evaluated methods, including ERA-U and ERA-S, achieve consistent success and naturalness on Assistax and HumanoidBench despite achieving some success according to reference rewards. We formulate our hypotheses about IRD failure modes below.

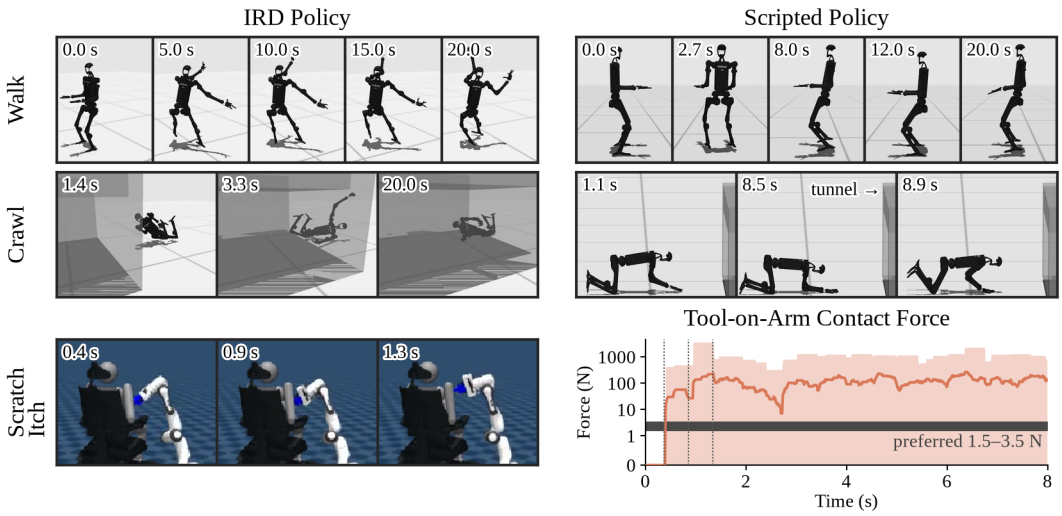


Figure 8: **Examples of high success, low naturalness.** All rollouts achieve $\geq 75\%$ reference reward. *Top left:* IRD policies on HumanoidBench flail frenetically on walk and crawl. *Top right:* scripted policies earn comparable reference reward with even stranger behavior: hopping facing backwards, rocking in place. *Bottom:* The IRD scratch-itch policy scores well on the reference reward, but poorly on Assistax's human preference reward: at one point applying over 3 kN of force to the patient's arm, enough to cause severe pain (ISO, 2016).

Visual verification is unreliable. Human completion ratings track benchmark scores closely on Meta-World ($\rho = 0.78$) but only weakly on Assistax (0.36) and HumanoidBench (0.26; Table 2). In these domains, success depends on quantities that are hard to see, such as contact force when scratching an arm, or hard to judge visually, such as forward velocity in a poor gait. We hypothesize that judgments humans cannot make accurately from video alone will also be hard for VLM judges. Therefore, we propose that when VLM judges are used to inform the reward, it is crucial that they are provided with sufficient information to judge task success (e.g. access to velocity if task is about speed).

Table 2: **Humans agree on completion more than on naturalness.** Gym MuJoCo clips were rated for completion only, and HumanoidBench naturalness ratings are almost all at the minimum, so that cell is not reported, as no informative correlation can be calculated.

	Compl.	Natur.
Rater A vs. B	0.83	0.54
Human vs. benchmark	0.58	0.47
MT10	0.78	0.71
Gym MuJoCo	0.57	–
Assistax	0.36	0.29
HumanoidBench	0.26	–

Naturalness is hard to agree on. Alignment requires both that the task is done and that it is done as humans intend. We find that correlation between the two reviewers is higher on task completion ($\rho = 0.83$) than on naturalness ($\rho = 0.54$). We also find that reference rewards track naturalness ratings less well than they track completion ratings (0.47, Table 13) showing that naturalness is less closely tied to the reference reward. However, naturalness ratings on HumanoidBench are almost unanimous: 70 of 73 ratings gave a naturalness score of 0 and every one of the 73 was given with high confidence. On the manipulation domains the same question is far less settled, with high confidence on 60% of Meta-World ratings and 47% of Assistax ratings. This asymmetry matters for anyone proposing visual assessments of trajectories as a training signal: it is reliable where the embodiment makes failure legible, and noisy everywhere else.

Reference rewards are hackable. We find, consistent with prior work on specification gaming (Krakovna et al., 2020), that HumanoidBench's walk and crawl reference rewards can be maximized without walking or crawling as a human would. Of 67 learned policies, 9 exceed 75% reference reward, and raters judged all 9 unnatural (naturalness ≤ 2). This could be an artifact of RL finding strange policies; we rule this out with two scripted policies that reach comparable reference reward by hopping facing backward or rocking in place (Figure 8). We elaborate on the implications of this in the discussion.

5 Discussion

Two small bitter lessons for reward design. Sutton (2019) famously argued that general-purpose methods that scale with compute tend, in the long run, to outperform approaches built around human intuition. RL reward design offers two modest parallels. First, LLM-based search can produce rewards that outperform those written by hand (Ma et al., 2024a). Second, a wide search can help improve the reward-design procedure itself. Several choices we expected to help did not: tapering the number of candidates per round, for example, made little difference at an equal budget (Appendix G). Yet a small, semi-manual hill-climb over simple design choices produced ERA-U and ERA-S (Section 4.3). These findings suggest that empirical search deserves a larger role in designing IRD methods, especially as stronger models and automated experimentation make that search more effective.

Alignment. Better search, however, only helps to the extent that its objective captures what we want. Higher benchmark scores did not reliably correspond to the behavior we intended (Section 4.4). Task descriptions usually specify what to achieve while leaving much of how to achieve it unstated. We hypothesize that LLM designers, as aggressive reward seekers (Pan et al., 2022; 2024), will widen rather than close the gap between score and naturalness as IRD methods become stronger unless naturalness becomes part of the objective or feedback.

Scalable oversight. Models can reduce the human effort of evaluating behavior, but relying on their judgments to guide further optimization risks propagating errors that those same evaluators may not detect. In our study, a VLM judge scored clips whose camera views obscured task-relevant outcomes; human inspection exposed these problems and prompted re-rendering. We also encountered inaccurate task descriptions, illustrating that both the objective and the evidence used to assess it can be unreliable. These are problems that any IRD system would need to address: automated changes should not be accepted as improvements merely because they satisfy an evaluator whose inputs or judgments are flawed. Our observations therefore caution against trusting current model-based feedback as the sole oversight mechanism for autonomous improvement. Human review helped correct specific problems here; scaling this process requires systematic checks of whether evaluators can observe the relevant outcomes and whether their judgments track the intended goals.

The future of IRD. As models become more capable, we suspect that new paradigms of IRD will become possible. VLMs may be able to emit reward values directly without needing to represent them programmatically; they may also natively be able to reliably control embodiments, or write Python programs that do so.

6 Conclusion

We introduced BIRD, a framework that expresses published IRD methods as configurations of one loop, so that they can be compared, ablated, and recombined under matched feedback conditions and policy-training budgets. Our experiments suggest two takeaways. First, simple design choices were the most robust: in our matched setting, the benefits of many design choices, including many of our own proposals, did not carry over consistently across tasks and methods, which may reflect how strongly those benefits depend on the setting, while a few simple choices helped consistently. Second, search over such choices is productive: a small hill-climb starting from RDA and drawing on components from prior methods, such as RF-Agent's design thought, produced the two strongest methods we evaluated, and we expect further search to improve on them. Our human study, though small, also suggests a limit to this approach: hill-climbing on benchmark scores improves performance but not necessarily the intended behavior, and encoding naturalness and human intent in the objective remains a hard, open problem for the field. We describe limitations in Appendix D.

REFERENCES

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. *Advances in Neural Information Processing Systems (NeurIPS)*, 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. *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Arpad E. Elo. *The rating of chessplayers, past and present*. Arco Publishing, New York, 1978.

Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving rewards to automate reinforcement learning. *ICML Workshop on Automated Machine Learning (AutoML)*, 2019.

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

Adam Gleave, Michael Dennis, Shane Legg, Stuart Russell, and Jan Leike. Quantifying differences in reward functions. *International Conference on Learning Representations (ICLR)*, 2021.

Andrew F. Hayes and Klaus Krippendorff. Answering the call for a standard reliability measure for coding data. *Communication Methods and Measures*, 1(1):77–89, 2007.

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

Leonard Hinckeldey, Elliot Fosong, Elle Miller, Rimvydas Rubavicius, Trevor McInroe, Patricia Wollstadt, Christiane B. Wiebel-Herboth, Subramanian Ramamoorthy, and Stefano V. Albrecht. Assistax: A hardware-accelerated reinforcement learning benchmark for assistive robotics. *RLC Workshop on Coordination and Cooperation in Multi-Agent Reinforcement Learning (CoCoMARL)*, 2025.

Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with MuJoCo. *arXiv preprint arXiv:2212.00541*, 2022.

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

ISO. ISO/TS 15066:2016 – robots and robotic devices – collaborative robots, 2016. Technical Specification. Annex A, Table A.2: biomechanical limits for quasi-static and transient contact.

Matthew Thomas Jackson, Uljad Berdica, Jarek Liesen, Shimon Whiteson, and Jakob Nicolaus Foerster. A clean slate for offline reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Akshat Singh Jaswal, Ashish Baghel, and Paras Chopra. Discovering reinforcement learning interfaces with large language models. *Reinforcement Learning Conference (RLC)*, 2026.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. *European Conference on Machine Learning (ECML)*, 2006.

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

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. *arXiv preprint arXiv:2502.15280*, 2025.

Hojoon Lee, Ajay Subramanian, Ben Abbatematteo, Vijay Veerabadran, Pedro Matias, Karl Ridgeway, and Nitin Kamra. RDA: Reward design agent for reinforcement learning. *Reinforcement Learning Conference (RLC)*, 2026.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022.

Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. Auto MC-Reward: Automated dense reward design with large language models for Minecraft. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

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. *International Conference on Machine Learning (ICML)*, 2025a.

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

Michel Ma, Takuma Seno, Kaushik Subramanian, Peter R. Wurman, Peter Stone, and Craig Sherstan. Automated reward design for Gran Turismo. *NeurIPS Workshop on Bridging Language, Agent, and World Models (LAW)*, 2025.

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. *International Conference on Learning Representations (ICLR)*, 2024a.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language model guided sim-to-real transfer. *Robotics: Science and Systems (RSS)*, 2024b.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. *International Conference on Machine Learning (ICML)*, 1999.

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.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *International Conference on Learning Representations (ICLR)*, 2022.

Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. In *International Conference on Machine Learning*, 2024.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL https://jmlr.org/papers/v22/20-1364.html.

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

Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. FastTD3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.

Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoid-Bench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *Robotics: Science and Systems (RSS)*, 2024.

Satinder Singh, Richard L. Lewis, and Andrew G. Barto. Where do rewards come from? *Annual Conference of the Cognitive Science Society (CogSci)*, 2009.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Jonathan Sorg, Richard L. Lewis, and Satinder Singh. Reward design via online gradient ascent. *Advances in Neural Information Processing Systems (NeurIPS)*, 2010.

Shengjie Sun, Runze Liu, Jiafei Lyu, Jing-Wen Yang, Liangpeng Zhang, and Xiu Li. A large language model-driven reward design framework via dynamic feedback for reinforcement learning. *Knowledge-Based Systems*, 326:114065, 2025.

Richard S. Sutton. The bitter lesson, 2019. Incomplete Ideas (blog).

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

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

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. *International Conference on Learning Representations (ICLR)*, 2024.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019.

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. *Conference on Robot Learning (CoRL)*, 2023.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

A TERMINOLOGY

Table 3: Terminology used throughout the paper. Examples illustrate reward design for a cube-manipulation task.

Term	Definition	Example
Task objective	The intended behavior the policy should achieve.	Pick up a cube and hold it at the target height for two seconds.
Task description	The natural-language statement of the task objective given to the generator, judges, and raters.	"Lift the cube and hold it at the green marker."
Reward function	The function providing the learning signal during policy training. A *candidate reward* is one proposed during search.	$R = R_{\text{reach}} + R_{\text{lift}} + R_{\text{hold}}$, rewarding reaching, lifting, and maintaining the hold.
Reference reward	The reward function distributed with the environment. It is withheld from the generator.	The benchmark's hand-designed dense reward for the cube task.
Benchmark score (fitness)	The value a benchmark assigns a trained policy with respect to the task objective; formally, the fitness F. Reward design seeks to maximize $\mathbb{E}[F(\mathcal{A}_M(R))]$.	The rate of successful two-second holds: eight successes in ten evaluation episodes give $F = 0.8$.
Feedback	Information used to guide reward search, including benchmark scores, model or human judgments, and training statistics.	A benchmark score of 0.4, or "The cube reaches the target height but is immediately dropped. Increase the reward for maintaining the hold."
Supervised / unsupervised IRD	Whether a method uses the benchmark score as feedback during search (supervised) or never sees it (unsupervised).	Eureka selects the reward whose policy has the highest success rate (supervised); RDA asks a VLM to judge rollout frames (unsupervised).
Design choice	A configurable aspect of the IRD procedure.	Policy initialization: `from_scratch` or `warm_start`.
Key	Each design choice is represented as a *key* in a method's configuration YAML file.	The default BIRD algorithm sets `checkpoint_selection=final`.
Configuration	The complete set of selected settings for the design choices that specifies an IRD procedure.	See Figure 3.
Policy-training budget	The amount of policy training allocated to reward search, specified per training run and across the whole search.	5 iterations $\times$ 10 candidates per iteration $\times$ 1M environment steps per policy-training run $=$ 50M environment steps total.
Completion rating	A human or model judgment of how fully the displayed behavior achieves the task objective.	"5/5: the cube is held at the target height for two seconds."
Naturalness rating	A human judgment of how closely the displayed behavior matches their expectations about how the task objective should be achieved.	"2/5: the hold succeeds, but the arm repeatedly jerks."

B PRIOR WORK

Reward optimality. The history of iterative reward design began with the observation that not all rewards are created equal. Singh et al. (2009) explored this point by distinguishing between the

reward the agent learns to maximize and the *fitness* by which the designer judges the usefulness of that reward. Intuitively, the designer cares about what the agent learns to do, rather than the return it earns under a particular reward. For example, a grasping reward might combine gripper proximity, contact, and object height, but the designer judges its usefulness by how reliably the trained agent picks up an object. Formally, this is written as a *reward design problem* (RDP)

$$R^\star \in \arg\max_{R \in \mathcal{R}} \mathbb{E}[F(\mathcal{A}_M(R))]\,, \tag{1}$$

where $\mathcal{R}$ is the space of candidate rewards, $\mathcal{A}_M$ is a learning procedure that trains a policy in environment M under reward R, and F evaluates that policy.[5]

Early automated search. While Singh et al. (2009) formulated the optimal reward problem, they remained agnostic about how best to solve it. Their experiments used exhaustive searches, motivating subsequent work on finding useful rewards more efficiently. Evolutionary approaches searched over symbolic reward programs, while gradient-based approaches exploited an agent's planning structure to improve its reward online (Niekum et al., 2010; Sorg et al., 2010). Reward optimization subsequently developed into methods for learning intrinsic rewards alongside policy learning (Zheng et al., 2018) and large-scale tuning of predefined reward components, as in AutoRL (Faust et al., 2019). These developments made reward search more practical, but faced different constraints: flexible program search required many evaluations, while more structured approaches depended on a specified reward family or access to the agent's optimization process.

LLM-guided reward design. The advent of coding LLMs offered a new way to guide reward search using task descriptions, environment code, and natural-language feedback. Rather than relying only on generic search operators, coding models could use their programming knowledge to propose and revise programs—a possibility explored more broadly in evolutionary search by Lehman et al. (2022). Eureka (Ma et al., 2024a) applied this idea to reward design, retaining an evolutionary loop in which candidate rewards are evaluated through policy training and revised using feedback on the results. Text2Reward (Xie et al., 2024) similarly generated reward programs and supported iterative refinement through human feedback. These methods automated much of the manual design loop, but still had to decide which candidate rewards were worth training policies under and how to judge the behavior they produced.

Extensions to the loop. With this loop in place, recent work asks how to make better use of each policy-training run. Broader search histories and trajectory-based screening offer ways to revisit promising directions and reject unsuitable rewards before training again (Hazra et al., 2025; Gao et al., 2025; Sun et al., 2025). The reward is also only one part of the learning setup: extending search to policy observations and training distributions allows these choices to be designed alongside it (Jaswal et al., 2026; Ma et al., 2024b). Across these extensions, however, better search still requires a way to judge behavioral improvement. This has motivated VLM-based judgments of policy rollouts as an alternative to benchmark-score feedback, allowing task descriptions to guide candidate comparisons and subsequent revisions (Lee et al., 2026; Ma et al., 2025).

We select the methods represented in BIRD to cover these different search procedures, design spaces, and feedback assumptions. Figure 3 summarizes their configuration differences; Appendix C describes each represented variant and distinguishes implementation coverage from the methods evaluated in our experiments.

C Methods in BIRD

BIRD implements 14 published IRD methods (Table 4). We briefly describe each below.[6]

[5] We use the formulation and terminology from Ma et al. (2024a).

[6] Two early methods, the reward search of Singh et al. (2009) and Language to Rewards (Yu et al., 2023), rely on exhaustive search and model-predictive control. We include them in simplified form to show that BIRD's design space extends beyond LLM-written rewards trained with RL.

Table 4: **Methods in BIRD.** Feedback: S = score, C = per-component reward statistics, T = trajectories, N = text critique. Optimal Reward Search and Language to Rewards are omitted (no policy-training feedback loop). [a]LLM-written critics refit reward parameters. [b]An LLM rates reward designs as a search prior. [c]The original paper's own automated variant (REvolve: the benchmark score as its hand-designed fitness; Gran Turismo: an LLM in place of human feedback). [d]Shown to the generator, but never used to rank. [e]Interactive variant.

		Evaluator					
Method	Ranking signal	Bench.	VLM	Human	LLM	Feedback	Selection
Supervised							
Eureka	benchmark	✓	–	–	–	S C	argmax
DrEureka	benchmark	✓	–	–	–	S C	argmax
ROSKA	benchmark	✓	–	–	–	S C	elitist
R*	benchmark	✓	–	–	✓[a]	S C	argmax
RF-Agent	benchmark	✓	–	–	✓[b]	S C	UCT tree search
LaRes	benchmark	✓	–	–	–	S	elitist
LIMEN	benchmark	✓	–	–	–	S	MAP-Elites
REvolve[c]	benchmark	✓	–	–	–	S C	island EA
CARD	none	✓[d]	–	–	–	S C T	none
Unsupervised							
RDA	VLM score	–	✓	–	–	S N	argmax
Gran Turismo[c]	LLM preference	–	✓	–	✓	C N	Bradley–Terry
Text2Reward[e]	none	–	–	✓	–	N	none

Optimal Reward Search Singh et al. (2009) argue that the best reward for a learning agent need not be the fitness its designer cares about, and define the optimal reward as the one under which the agent accumulates the most fitness. They find it by exhaustive search. Each candidate reward assigns a value from a discretized range to each combination of a few state features, such as the agent's hunger and thirst. A Q-learning agent (Watkins & Dayan, 1992) is trained under each candidate, and the reward whose agent accumulates the most fitness over its lifetime is kept.

Language to Rewards Yu et al. (2023) argue that LLMs struggle to output low-level robot actions directly, and use reward functions as the interface between language and control. A "motion descriptor" rewrites the user's instruction as a templated description of the desired motion, and a "reward coder" turns it into code that sets the weights of predefined reward terms. MuJoCo MPC (Howell et al., 2022) then optimizes the reward online, so no policy is trained, and a person types corrections for the next round.

Eureka Ma et al. (2024a) cast reward design as evolutionary search over reward code. Each iteration, an LLM samples a batch of reward programs from the task description and the environment's source code. A policy is trained under each, and the reward with the best task score is kept. A "reward reflection" summarizing each reward component's values over training then seeds the next iteration.

Text2Reward Xie et al. (2024) generate dense reward code from a natural-language instruction, showing the LLM a hand-written Pythonic abstraction of the environment rather than its source. Execution errors are returned to the LLM until the code runs, and a few-shot variant retrieves an expert-written reward from a related task. An interactive variant revises the reward using a person's free-text feedback on rollout videos.

DrEureka Ma et al. (2024b) extend Eureka to sim-to-real transfer. The reward search adds a safety instruction asking for stable, smooth motion. After the search, a reward-aware physics prior finds the range of each physics parameter over which the policy still succeeds, and the LLM writes domain-randomization configurations within those ranges. Policies trained under them are compared on the real robot.

REvolve Hazra et al. (2025) target tasks such as driving, where good behavior is hard to quantify, and argue that refining a single winner, as Eureka does, discards useful candidates. REvolve runs an island-model evolutionary search: each offspring is produced by mutation or crossover of parents sampled by fitness from one island, and it joins that island only if its fitness is at least the island

mean. Fitness is an Elo rating (Elo, 1978) over human preferences between rollout videos, and human critiques are added to the prompt.

CARD Sun et al. (2025) aim to reduce the cost of iterative reward design in human feedback, LLM queries, and RL training. A Coder writes one reward per round, and an Evaluator reports training curves, success rates, and example rollouts back to it. Their Trajectory Preference Evaluation checks whether a new reward scores stored successful trajectories above failed ones, and a reward that fails this check is not trained.

RF-Agent Gao et al. (2025) argue that greedy and evolutionary searches make poor use of past feedback, and cast reward design as Monte Carlo tree search over trained rewards. Each round descends the tree by UCT (Kocsis & Szepesvári, 2006) to a leaf and expands it with LLM actions such as structure mutation, weight mutation, and crossover with elite rewards. The best task score is backed up the tree, and an LLM's rating of each reward's design serves as an exploration prior.

R* Li et al. (2025a) extend Eureka in two ways. First, a quarter of each batch comes from crossover that inserts a reward component from one archived reward into another, without an LLM call. Second, LLM-written critic programs compare trajectory segments, and each reward's numeric parameters are refit to these preferences with a Bradley–Terry loss (Bradley & Terry, 1952).

ROSKA Huang et al. (2025) argue that training each candidate's policy from scratch, as Eureka does, is inefficient. Each new candidate instead starts from a blend of the previous best policy and a random initialization, with the blend ratio chosen by Bayesian optimization (Snoek et al., 2012) over short training probes. The search also keeps sampling rewards until one beats the incumbent.

LaRes Li et al. (2025b) target sample efficiency by pairing each of a population of LLM-written rewards with its own off-policy agent. All agents share one replay buffer, relabeled under every member's reward, and Thompson sampling (Thompson, 1933) decides which agent collects data. Periodically, the LLM rewrites the less successful rewards from the best one, and their agents restart from the best agent.

Gran Turismo Ma et al. (2025) design rewards for a racing agent without access to a fitness function during search. An LLM writes reward components from a description of the state and action space, and a filter trains only the candidates most consistent with earlier preferences. A VLM describes paired clips of the trained agents, an LLM picks the better agent in each pair, and Bradley–Terry ranking selects the winner, on which a person gives feedback.

RDA Lee et al. (2026) observe that coarse numeric feedback such as success rate produces policies that reach the goal while ignoring the instruction. A VLM first decomposes the instruction into subtasks. It then scores each candidate's rollout frames against each subtask, and the reward with the best mean score is kept. Reflection revises both the reward and the subtask list, and policies can warm-start from the previous best.

LIMEN Jaswal et al. (2026) note that LLM reward design assumes a fixed observation space, and instead search over the whole RL interface. Each candidate is a program that outputs both an observation map and a reward. A MAP-Elites archive (Mouret & Clune, 2015), binned by observation size and program complexity, keeps the best interfaces and supplies parents for the next candidate.

D LIMITATIONS

We observe six main limitations. First, human ratings come from two authors and may not represent broader human preferences. Second, we evaluate only a subset of IRD methods, excluding specialized approaches such as LIMEN (iterative reward and observation design) (Jaswal et al., 2026) and Auto-MC (Li et al., 2024) (IRD for Minecraft), which may offer additional effective design choices. Third, we do not study computational efficiency or ways to reduce the cost of IRD. Fourth, ERA-U and ERA-S come from a deliberately small search and are not attempts at a state-of-the-art method. Fifth, we did not systematically vary the generator or judge model, so how our conclusions depend on model choice is untested. It is also possible that some task information has leaked into the VLMs, as the benchmarks are publicly available. Finally, our evaluation covers robotic control only, leaving other RL domains untested.

E DESIGN CHOICES IN THE ABLATION STUDY

Table 5 lists the 30 design choices evaluated in Figure 5, using the same numbering and grouping by stage of the iterative reward design procedure. A dagger (†) marks a choice introduced in BIRD. A double dagger (‡) marks an intervention that changes two or three coupled configuration settings; its measured effect therefore applies to the combined intervention.

We evaluate the effect of switching each choice on in the applicable baseline configurations (Eureka, RDA, and CARD), excluding configurations in which the choice is inactive. The study covers nine tasks: six from Meta-World MT10 (peg-insert-side, pick-place, push, reach, window-close, window-open) and three from Gym MuJoCo (Hopper, Reacher, Swimmer), chosen so that the set spans manipulation and locomotion and includes the four MT10 tasks on which the baselines are not saturated (Appendix I.5). Each task is run with five seeds per experimental cell. Effects are expressed as Glass's Δ: the change in benchmark score divided by the baseline methods' standard deviation on the corresponding task. Effects are pooled over applicable baselines and tasks. In Figure 5, dots show individual tasks and diamonds show means across tasks with 95% BCa intervals computed over tasks; hollow diamonds identify the choices marked with † below.

Table 5: Design choices evaluated in the ablation study. Descriptions refer to the enabled setting. Numbering matches Figure 5. † indicates design choice introduced in BIRD, ‡ indicates multiple key flips to facilitate a design choice

No.	Design choice	Enabled setting
Generate		
1	Four-turn history	Use a history of four turns as generation context.
2	Behavioral analysis in prompt	Include behavioral analysis in the generation prompt.
3	Thought + parent + alignment‡	Enable design thought, parent conditioning, and alignment together.
4	Global-best parent	Use the highest-fitness reward found so far as the parent for generation.
5	Co-evolve subtasks	Update the subtask decomposition alongside the reward.
6	Prompt reset each iteration	Reset the prompt at each iteration.
7	Design thought	Generate a description of the reward's design intent alongside the reward code.
8	Reward-engineering tips	Include reward-engineering guidance in the generation prompt.
9	Tapered candidate schedule†‡	Generate fewer candidates in later search rounds.
10	Thought + parent's thought‡	Enable design thought and include the parent's design thought as context.
11	Task decomposition in prompt	Include a decomposition of the task into subtasks in the generation prompt.
12	Dialogue history	Include dialogue history as generation context.
13	Environment source code	Include the environment's source code in the generation prompt.
14	Reward code as assistant turn	Represent reward code as an assistant turn in the conversation.
15	Per-component output + feedback‡	Enable per-component reward output together with per-component feedback.
Train		
16	Peak own-reward checkpoint†	Evaluate the policy checkpoint with the highest return under the candidate's own reward.
17	Warm start from best policy‡	Initialize policy training from the best policy found so far.
Evaluate		
18	Summarized feedback	Provide evaluation feedback in summarized form.
19	Task-success fitness	Use task success as the candidate's fitness.
20	Fitness scalar in feedback†	Include the scalar fitness value in feedback to the generator.
21	Maximum fitness over training	Use the maximum fitness observed during policy training.
22	Aliased metric name	Present the evaluation metric under an alias.

Continued on next page

Table 5 (continued)

No.	Design choice	Enabled setting
23	Repeated VLM judging†	Evaluate policy behavior through repeated calls to a vision-language model judge.
24	Curve-similarity report	Include a curve-similarity report in the evaluation feedback.
25	Numeric reflection	Include numeric reflection in the evaluation feedback.
26	Per-component feedback	Provide feedback for individual reward components.
Select		
27	Return global best	Return the highest-fitness reward found over the full search.
28	Population of 16 survivors‡	Retain a population of 16 surviving candidates.
29	Select across all rounds†	Select candidates from across all search rounds.
30	Island EA (two demes)‡	Use island-model evolutionary search with two demes.

F Reward Distance Across Iterations

To measure how the reward itself changes as IRD iterates, we compute the EPIC distance (Gleave et al., 2021) between each iteration's best reward and the reference reward. EPIC canonicalizes both rewards to remove potential shaping and compares them by Pearson distance over a distribution of transitions: 0 means the two rewards induce the same optimal policies, and unrelated rewards score about 0.7. We sample 1,024 transitions from rollouts of a competent policy (Meta-World's scripted policies on MT10; PPO trained on the reference reward on MuJoCo), so the distance reflects states a successful policy visits, with 128 samples for canonicalization. We evaluate Eureka, RDA, CARD and Gran Turismo with ten seeds and five iterations on all ten MT10 tasks and on the four Gym MuJoCo tasks whose reference reward is published (half-cheetah, hopper, reacher and swimmer). In these runs, policy training for all four methods used early stopping based on the benchmark score; this affects which reward is retained at each iteration but not how EPIC is computed.

Every method is far closer to the reference reward than chance. On MT10, Eureka, RDA and CARD sit at about 0.2 and Gran Turismo at about 0.3, against 0.65 for random rewards; on MuJoCo the methods lie between 0.3 and 0.45, against 0.74 (Figure 9).

Iteration does not move the reward toward the benchmark's. On MT10, every method's mean distance rises slightly over five iterations, by at most 0.06; the distance fell significantly from the first iteration to the last in 1 of 40 method–task combinations and rose in 10 (sign test, $p < 0.05$). On MuJoCo, Eureka and RDA drift away ($+0.06$ and $+0.08$), Gran Turismo is flat, and only CARD moves closer (-0.10), driven by reacher and swimmer, where its first-iteration rewards started near chance. Closeness to the reference reward also does not predict benchmark score: the pooled Spearman correlation between EPIC distance and within-task benchmark-score rank is -0.08 on MT10 and $+0.08$ on MuJoCo, though it reaches ± 0.4 on individual tasks. The search therefore improves the trained policy without recovering the reference reward.

G Candidate Allocation Isoquant Sweep

A reward search spends a fixed number of candidates C, which can be arranged as K candidates per iteration over $N = C/K$ iterations. $K = C$ is a single zero-shot round with no feedback; $K = 1$ is C sequential refinements of one reward. We ran Eureka on all ten Meta-World MT10 tasks with $K \in \{1, 2, 4, 8, 16, 80\}$ at $KN = 80$, and with schedules that vary K across rounds at the same total. Each candidate trains a PPO policy for 1M environment steps and is scored by the best task success over its training checkpoints; a run's value at budget C is the best such score among its first C candidates. Smaller budgets are read as prefixes of the same runs (e.g. the first 16 iterations of a $K = 1$ run), which is valid because the number of iterations changes nothing else in these configurations.

Wide versus deep. At $C = 16$ (Figure 10), seven of the ten tasks are flat across K. On drawer-open, peg-insert and pick-place, a single round of 16 candidates ($K = 16$) does worst and more iterations do better, so feedback helps where the task is not already solved; push shows the opposite trend, within seed variation.

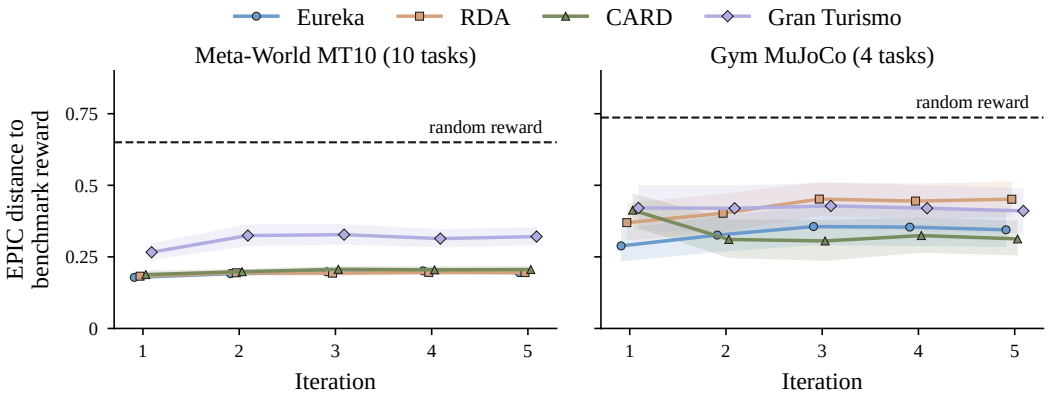


Figure 9: **Iteration does not move rewards toward the reference reward.** EPIC distance from each iteration's best reward to the reference reward, over transitions from a competent policy. Lines show means over tasks and ten seeds; bands indicate 95% bootstrap confidence intervals over seeds with the task set held fixed. The dashed line is the median distance of random rewards, averaged over tasks.

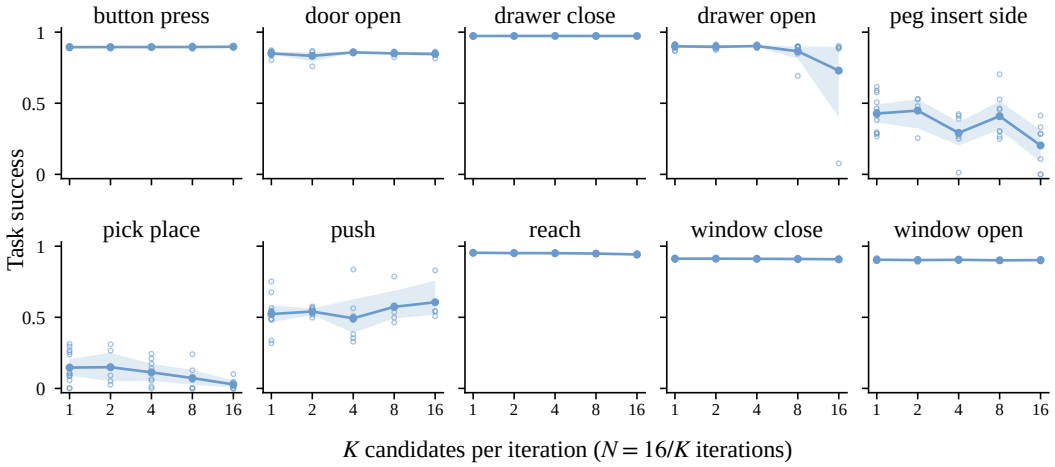


Figure 10: **Arranging a fixed budget of 16 candidates (Eureka).** Each panel is one MT10 task; K candidates per iteration trade against $N = 16/K$ iterations, so $K = 16$ is a single round. Lines show means over seeds (4–15 per point, median 8); hollow dots show individual seeds; bands indicate 95% bootstrap confidence intervals over seeds. The y-axis is the best task success among the first 16 candidates.

Tapered schedules. Table 6 compares two schedules that vary K across five rounds, Taper (40, 20, 10, 5, 5) and Ramp (5, 5, 10, 20, 40), with constant schedules that spend the same 80 candidates. Against $K = 8$ and $K = 80$, neither schedule differs from the constant arm by more than 0.02 in task success on average, and Taper and Ramp are indistinguishable from each other (-0.01, 95% CI $[-0.03, +0.01]$). Both lead $K = 16$, but so do the other constant arms: $K = 16$ is the weakest constant arm on these seeds. At a fixed budget, varying the number of candidates per round therefore makes little difference.

H Per-Task Method Comparison Results

We provide full per-task results below.

Table 6: **Tapered schedules match constant ones (Eureka, 80 candidates).** Mean paired difference in best task success, schedule minus constant arm, paired on task and seed, with 95% bootstrap confidence intervals over seeds (task set fixed). The unsaturated tasks are push, pick-place and peg-insert. Each Taper row pools 46 pairs over all ten tasks and 12 over the unsaturated three; Ramp, 43 and 12; the constant rows, 52 and 12.

schedule	vs. constant	all 10 tasks	unsaturated 3
Taper (40, 20, 10, 5, 5)	$K = 8$	$+0.01$ $[-0.02, +0.03]$	$+0.02$ $[-0.07, +0.10]$
	$K = 16$	$+0.02$ $[+0.00, +0.03]$	$+0.06$ $[+0.01, +0.11]$
	$K = 80$	$+0.01$ $[-0.01, +0.02]$	$+0.02$ $[-0.05, +0.09]$
Ramp (5, 5, 10, 20, 40)	$K = 8$	$+0.02$ $[-0.01, +0.04]$	$+0.06$ $[-0.02, +0.13]$
	$K = 16$	$+0.03$ $[+0.01, +0.05]$	$+0.10$ $[+0.03, +0.16]$
	$K = 80$	$+0.02$ $[+0.00, +0.03]$	$+0.06$ $[+0.01, +0.11]$
Constant $K = 8$ vs. $K = 16$		$+0.01$ $[-0.01, +0.03]$	$+0.05$ $[-0.04, +0.12]$
Constant $K = 80$ vs. $K = 16$		$+0.01$ $[-0.00, +0.02]$	$+0.04$ $[-0.02, +0.10]$

H.1 Performance

Figure 11 shows every method on each of the 21 tasks. No single method is best everywhere. Excluding InvertedPendulum, where every method reaches the maximum score, ERA-S has the highest mean on 9 of the remaining 20 tasks and ERA-U on 3, the two tie on HumanoidBench walk, and four baselines are highest on the other 7 (RDA, REvolve, and RF-Agent on two each, and Eureka on one).

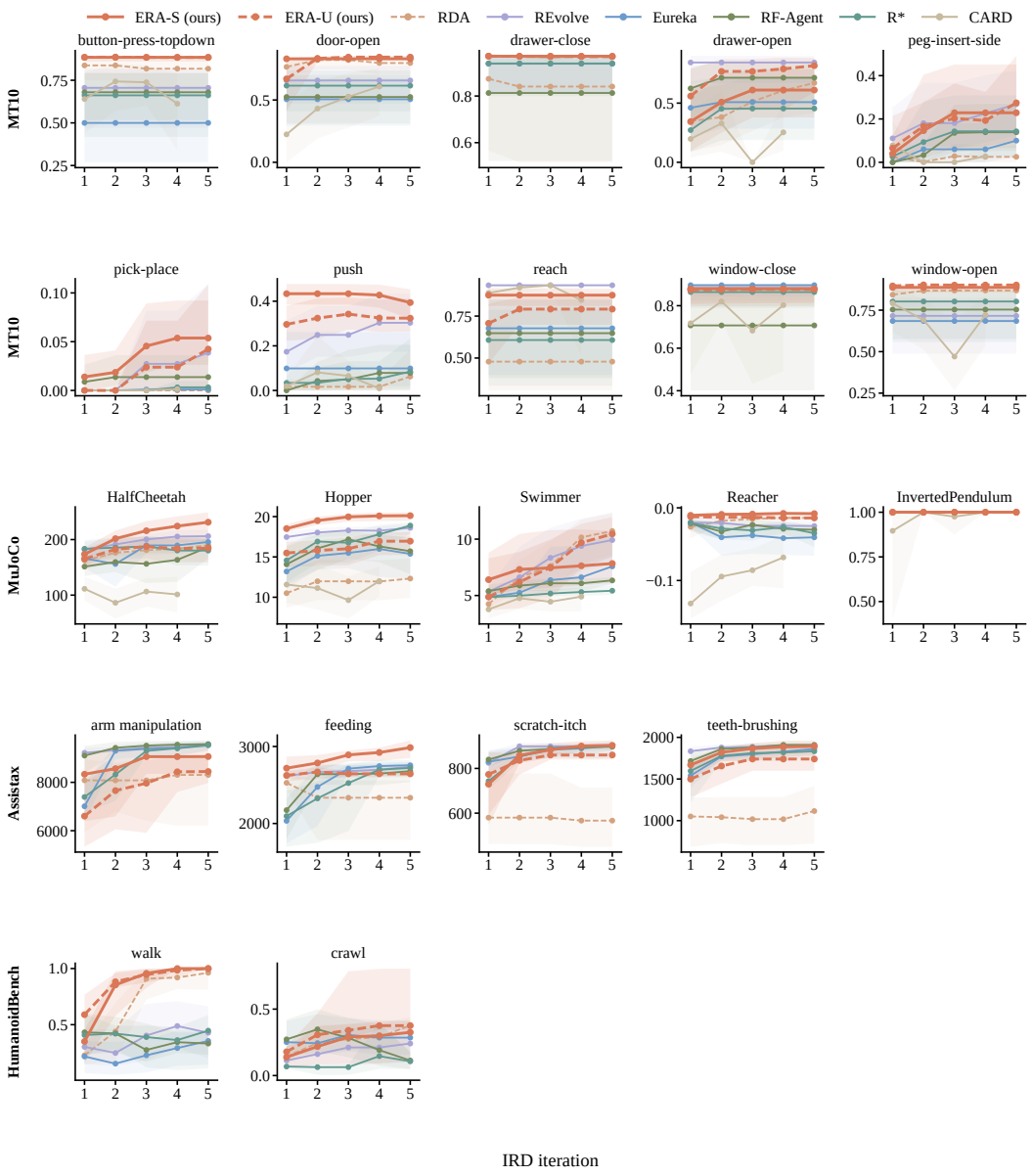


Figure 11: **Per-task results.** Benchmark score of the policy each search would return after each IRD iteration, in each task's own units: Meta-World MT10 (rows 1–2), MuJoCo (row 3), Assistax (row 4), and HumanoidBench (row 5). Lines show means over seeds (10 per task on MT10 and MuJoCo, 5 on Assistax and HumanoidBench) and shaded bands 95% BCa bootstrap intervals. Dashed lines mark unsupervised methods. CARD, shown on MT10 and MuJoCo, trains policies in its first four iterations only.

H.2 Compute Efficiency

All evaluated methods except for CARD use near-identical environment step training budgets. CARD uses approximately 10× fewer environment steps and, as would be expected, performs worse. However, the difference in performance is not as significant as would be expected given the discrepancy in compute, suggesting that there are diminishing returns for widening the reward search space within iterations. This paper does not optimize for compute efficiency; we expect that there is a lot of potential for further improvements.

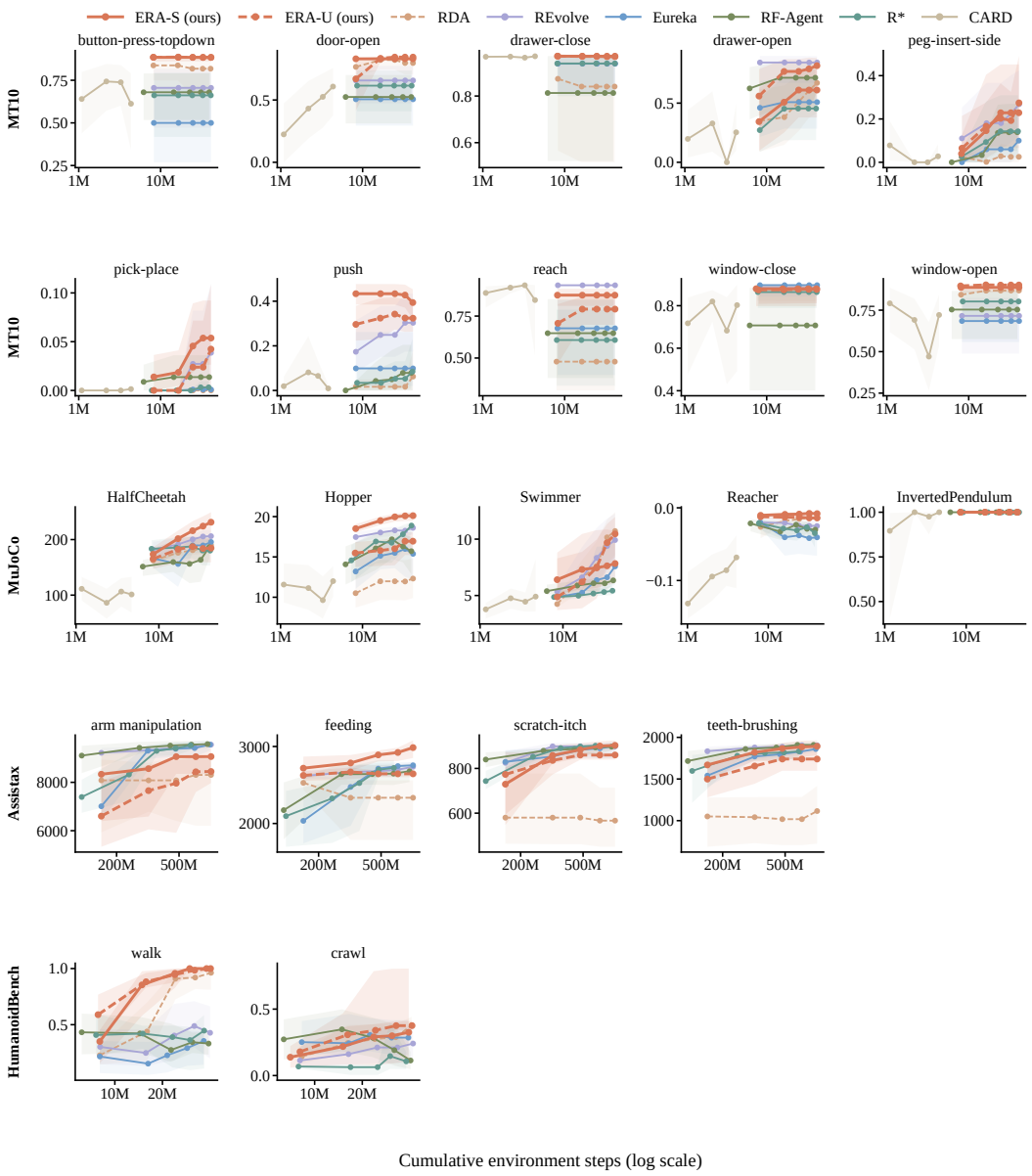


Figure 12: **Performance against environment steps.** The per-task curves of Figure 11, plotted against cumulative environment steps (log scale, mean over seeds). CARD trains one candidate reward per iteration and the other methods train eight, so CARD finishes with about one-tenth of their environment steps.

I Experimental Details of IRD Runs

I.1 Benchmarks and Tasks

We evaluate IRD methods on 21 tasks from four benchmarks (Table 7). Each task is specified once, in a task file that fixes the environment, its version, the episode horizon, the task description shown to the generator and the judges, and the *reference reward*: the quantity we report for a trained policy (Appendix A).

Table 7: **Benchmarks and tasks.** The IRD comparison uses 21 tasks: 10 from Meta-World MT10, 5 from Gym MuJoCo, 4 from Assistax and 2 from HumanoidBench. Horizon is the episode length in environment steps; seeds is the number of independent reward searches per method and task. †Bed bathing is not in the IRD comparison, because its episodes can end early; it appears only in the human study (Section 4.4).

Benchmark	Tasks	Horizon	Benchmark score	Reference reward	Seeds
Meta-World MT10, v3 (Yu et al., 2019)	button-press-topdown, door-open, drawer-close, drawer-open, peg-insert-side, pick-place, push, reach, window-close, window-open	500	Fraction of evaluation episodes in which Meta-World's own success flag is set on at least one step	Meta-World's shipped dense reward: shaped reach, grasp and place terms, scaled to $[0, 10]$ per step	10
Gym MuJoCo, v5 (Todorov et al., 2012)	HalfCheetah, Hopper, Swimmer, InvertedPendulum, Reacher	1000[a]	A physical quantity per task (forward displacement; time balanced; final distance to target), normalized so that a uniform-random policy scores 0 and the best method's mean scores 1	Gymnasium's shipped reward: forward velocity less a control cost (Hopper adds an alive bonus); $+1$ per upright step (InvertedPendulum); negative distance to target less a control cost (Reacher)	10
Assistax (Hinckeldey et al., 2025)	arm manipulation, feeding, scratch itch, teeth brushing, bed bathing†	1000	Fraction of steps on which the tool is at its target and a contact-force condition holds (our metric, defined per task in the text)	Assistax's shipped dense reward, transcribed term by term: distance-to-target shaping with orientation, speed and contact-force terms, and a control cost	5
HumanoidBench (Sferrazza et al., 2024)	walk, crawl	1000	Mean forward speed as a fraction of the 1 m/s target, clipped to $[0, 1]$; for crawl, multiplied by the fraction of steps spent inside the tunnel	HumanoidBench's shipped reward: posture terms (upright and standing for walk; low body height and in-tunnel for crawl) times velocity toward 1 m/s times small control effort; a return of 700 counts as success	5

I.2 Methods compared

Section 4.1 compares six published methods and our two recipes (Table 8). Every arm is the BIRD configuration described in Appendix C, run under one execution protocol: eight candidates per iteration, five iterations, and 1M environment steps per policy training (excluding Assistax which runs at 20M environment steps because of JAX acceleration), so a search trains 40 policies (excluding CARD). The RL optimizer is common per benchmark (Appendix I.3) rather than each paper's own. `claude-opus-5` writes every reward and, for RDA and ERA-U, judges the rollouts;

Table 8: The compared arms. K candidates per iteration, N iterations (published $\rightarrow$ run here); trainings are policies trained per search. The ranking signal picks the candidate carried forward; unsupervised arms never see the reference reward or the benchmark score during search. [a]CARD ranks nothing; its trajectory screen reads Meta-World's success flag and passes every reward elsewhere. [b]The return of the reference reward on MuJoCo, Assistax and HumanoidBench; on Meta-World, which ships a success flag, the flag is used instead, and it coincides with the benchmark score there.

method	K	N	trainings	ranking signal
supervised				
Eureka (Ma et al., 2024a)	$16 \rightarrow 8$	5	40	reference reward[b]
RF-Agent (Gao et al., 2025)	8	$11 \rightarrow 5$	40	reference reward[b]
REvolve (Hazra et al., 2025)	$16 \rightarrow 8$	$7 \rightarrow 5$	40	reference reward[b]
R* (Li et al., 2025a)	$16 \rightarrow 8$	5	40	reference reward[b]
CARD (Sun et al., 2025)	1	$3 \rightarrow 5$	4	none[a]
ERA-S (ours)	8	5	40	reference reward[b]
unsupervised				
RDA (Lee et al., 2026)	$4 \rightarrow 8$	5	40	VLM completion rating
ERA-U (ours)	8	5	40	VLM completion rating

`claude-sonnet-5` serves the secondary evaluator roles, such as RF-Agent's self-scoring of candidate code.

REvolve's island count. REvolve's paper configuration runs thirteen islands over a population of sixteen. At eight candidates per iteration, thirteen islands leave most islands with a single member. We run three islands, which keeps every island populated at this width.

I.3 Policy training

Policy training is not modified by IRD methods tested here, so we hold it fixed and choose it for the benchmark rather than for any method: every arm's candidates train under the same RL optimizer, hyperparameters, step budget and evaluation cadence on a given benchmark.

Meta-World and MuJoCo: PPO on CPU. We use Stable-Baselines3 PPO (Raffin et al., 2021; Schulman et al., 2017) at its library defaults. This setup is deliberately modest. A stronger learner solves several MT10 tasks under almost any reasonable reward, which hides differences between rewards; PPO at this budget leaves room for the reward to matter.

HumanoidBench: FastTD3 on GPU. HumanoidBench trains with FastTD3 (Seo et al., 2025) using SimbaV2 hyperspherical networks (Lee et al., 2025) at FastTD3's defaults.

Assistax: upstream's IPPO in JAX. Assistax trains with the benchmark's own learner rather than ours: the independent-PPO (IPPO) trainer released with Assistax (Hinckeldey et al., 2025), run end to end on the GPU in JAX at upstream's tuned settings. On Assistax, the whole training runs as one compiled JAX program, so for every method the policy is evaluated only once, at the end, so checkpoint-based choices (the peak own-reward checkpoint of ERA-U and ERA-S, and the maximum-over-training signals of Eureka, RF-Agent and R*) have no effect there.

Evaluation during training. Every training is paused at up to 20 evenly spaced points to evaluate the current policy; each checkpoint records the candidate's own return, the reference reward's return and the benchmark score.

Warm starts. RDA, ERA-U and ERA-S initialize each candidate's policy from the best policy found so far in the search, as RDA specifies; every other arm trains from scratch. Warm starting shares parameters, not rewards: the candidate is still trained and evaluated under its own reward alone.

I.4 Reward search protocol

A search is one run of the loop of Section 3: five iterations, eight candidate rewards per iteration, one policy trained per candidate under the benchmark's learner (Appendix I.3), and one reward returned

Table 9: Per-arm search settings, as resolved in the reported runs. Repairs are LLM retries allowed on failed reward candidate code. Return is the reward the search hands to the final retrain.

method	environment shown as	prompt memory	repairs	return
Eureka	full source	last iteration, 2 turns	0	global best
RF-Agent	full source	tree path, 4 ancestors	8	global best
REvolve	Pythonic abstraction	archive	0	archive best
R*	full source	last iteration	0	global best
CARD	Pythonic abstraction + template	full dialogue	10	final incumbent
RDA	full source	rebuilt, 4 turns	0	global best
ERA-U/S	full source	rebuilt, 4 turns	0	final incumbent

at the end. This subsection fixes what every arm shares and what each arm sets; Table 9 lists the per-arm settings.

Models. Every reward is written by `claude-opus-5`. The same model is the vision-language judge for RDA, ERA-U and ERA-S. `claude-sonnet-5` fills the secondary evaluator roles: the text analysis behind per-subtask feedback, and RF-Agent's rating of its own candidate code.

Verification. Every candidate passes the same checks before training: the code parses, defines a reward function of the expected signature and uses no forbidden symbol, then runs for one environment step and returns finite values of the right shape, under a 60-second limit. What happens on failure is each method's own choice.

Feedback and judging. After training, each candidate is described back to the generator. Supervised arms receive the ranking signal (Appendix I.2) together with statistics specified by their algorithm. The unsupervised arms are judged from video. Each trained policy is rolled out three times; twenty frames, evenly spaced over the rollout, are shown to the judge.

I.5 The hill-climb behind ERA

Setup. The climb started from RDA and ranked candidates only on the VLM judge: no configuration saw the benchmark score or the reference reward during search. We scored configurations by their mean task success on four MT10 tasks (pick-place, push, peg-insert-side and drawer-open), the only MT10 tasks on which the published methods were not saturated, with three to seven seeds each. To keep the climb cheap, configurations 2–23 ran with 24–30 policy trainings per search, most with early stopping on the candidate's own reward, and so used 8–25M environment steps (Table 10). Only configuration 1 and ERA-U (configuration 24) ran the full 40 trainings of Appendix I.2.

Demonstrations. Configurations 2–12 and 14–21 include a verifier that uses rollouts of Meta-World's scripted expert policy. Before training, it checks that a candidate reward pays the expert more than a random policy, and the rewards it assigns to these rollouts are shown to the judge and the generator. These configurations therefore use demonstrations, which RDA does not. The verifier rejected none of roughly 600 candidates, and we removed it in configuration 22, so ERA-U uses no demonstrations.

The climb. We ran four waves, each changing one design choice, or one coupled group of choices, on a base taken from the previous wave. We chose bases by judgment rather than by a fixed rule, since the intervals of the leading configurations overlap. From the RF-Agent wave (14–21) we kept design thought (20) rather than elite crossover (16), because the tree-based configurations produced collapsed seeds on peg-insert-side and drawer-open. We then removed the verifier (22) and added the peak own-reward checkpoint at the full budget, which gave ERA-U (24). We stopped there, without a second climb, without climbing ERA-S or any other method, and without varying the models.

I.6 Compute

Every search in Sections 4.1 and 4.3 trains 40 policies. Table 11 gives the cost of one search and of each benchmark's full set, measured where we have the run records and estimated from measured per-training rates otherwise.

Table 10: The 24 experiments of Figure 6, by experiment number. Δ is the configuration's task success (best checkpoint of the returned reward) relative to RDA, averaged over pick-place, push, peg-insert-side and drawer-open. Intervals are 95% BCa over seeds. †Fewer completed seeds on some tasks. • marks correspond the filled markers of the figure.

#	configuration	change tested	seeds	steps (M)	Δ vs. RDA [95% CI]
1	v1	Continuous $[0, 1]$ judge score; return the end of the chain; tapered candidate count at 40 trainings	$5^{\dagger}$	41.3	$+0.01\,[-0.09, +0.11]$ •
2	v2_verify	Demonstration-based verifier before training; early stopping on own reward; 24 trainings	3	10.7	$+0.13\,[+0.02, +0.23]$ •
3	v2_verify_wide	Wide first round, short tail	3	12.1	$+0.12\,[+0.00, +0.23]$
4	v2_verify_short	Two rounds only, 16 then 8 candidates	3	11.9	$-0.13\,[-0.22, -0.01]$
5	v2_verify_adapt	Verifier sets how many passers to train	3	8.0	$+0.03\,[-0.15, +0.19]$
6	v2_verify_kl	KL penalty toward the demonstration policy	3	10.6	$-0.14\,[-0.26, +0.01]$
7	v2_verify_noes	Verifier without early stopping	3	24.8	$0.00\,[-0.12, +0.16]$
8	v2_verify_es8	Gentler early-stopping rule	$5^{\dagger}$	19.4	$-0.01\,[-0.14, +0.09]$
9	v2_verify_es8_constant	Control: constant width instead of the taper	5	19.5	$+0.03\,[-0.07, +0.13]$
10	v2_verify_es8_ramp	Control: the taper reversed	5	19.7	$-0.05\,[-0.18, +0.05]$
11	v2_verify_es8_distinct	Diversity by distinct per-sample prompts	$4^{\dagger}$	19.5	$-0.01\,[-0.15, +0.08]$
12	v2_verify_es8_personas	Diversity by system-prompt personas	$3^{\dagger}$	20.1	$-0.17\,[-0.31, -0.01]$
13	v2_noverify_es8	Control: verifier removed	$5^{\dagger}$	19.8	$-0.01\,[-0.14, +0.09]$
14	v3_tree	RF-Agent's search tree as persistent memory	5	9.3	$+0.08\,[-0.06, +0.21]$
15	v3_all	All five RF-Agent mechanisms together	5	9.6	$+0.10\,[-0.04, +0.22]$
16	v3_elite	Crossover with an elite set of trained rewards	5	9.6	$+0.15\,[+0.00, +0.28]$ •
17	v3_actions	RF-Agent's distinct mutation actions	5	9.6	$+0.05\,[-0.10, +0.19]$
18	v3_puct	PUCT child selection in place of UCT	5	10.4	$+0.07\,[-0.06, +0.19]$
19	v3_selfverify	LLM self-verification score as a UCT prior	5	10.2	$-0.02\,[-0.16, +0.11]$
20	v3_thought	Design thought before the code, carried in the prompt	5	11.4	$+0.13\,[-0.01, +0.25]$
21	v3_thought_elite	Design thought and elite crossover together	5	9.6	$+0.08\,[-0.07, +0.23]$
22	v4	Verifier removed from v3_thought	5	13.5	$+0.01\,[-0.11, +0.12]$
23	v4_tree	Design thought on the tree-and-actions arm, verifier removed	5	13.2	$+0.01\,[-0.09, +0.10]$
24	**ERA-U**	v4_peak_noes40: peak own-reward checkpoint; early stopping off; Eureka's 40 trainings (main-body study)	7	40.0	$\mathbf{+0.21}\,[+0.11, +0.32]$ •

Table 11: Compute per benchmark. Set totals cover the searches in Section 4.1 and Figure 7; development runs, pilots and the hill-climb of Section 4.3 are excluded.

benchmark	learner and hardware	searches	set total
Meta-World MT10	PPO (SB3), 8 vCPUs of a 128-vCPU n2d	800	≈4,200 CPU-cell-h
Gym MuJoCo	PPO (SB3), 8 vCPUs of a 128-vCPU n2d	400	≈1,400 CPU-cell-h
Assistax	IPPO, 1 H200 per training	214	≈1,600 GPU-h
HumanoidBench	FastTD3 + SimbaV2, 1 H200 per training	93	≈2,200 GPU-h

Table 12: Rater reliability. Within-rater figures use unannounced repeat showings; between-rater figures use the clips both raters saw. α is Krippendorff's coefficient with an ordinal metric (Hayes & Krippendorff, 2007), which measures absolute agreement and therefore charges a constant difference in scale use as disagreement; Spearman's ρ is invariant to it. Intervals are percentile bootstrap, 2000 resamples.

comparison	question	pairs	α	ρ	exact match	mean gap (scale steps)
within rater	completion	19	0.95	0.91	84%	0.16
within rater	naturalness	16	0.85	0.71	62%	0.38
between raters	completion	24	0.74	0.83	42%	0.58
between raters	naturalness	17	0.57	0.54	53%	0.49

Language-model usage. Across the approximately 1500 IRD runs for the main results, the generator and judge together made approximately 200k model calls and consumed about 2.2 billion tokens.

J Human Annotation Study

J.0.1 Evaluation Setup

We collected human ratings of 257 unique episode clips across 22 tasks in Meta-World MT10, HumanoidBench, Assistax and Gym MuJoCo. Every clip is a single training rollout of a learned policy produced by one of the IRD methods under comparison, and carries the benchmark score measured on that same filmed episode. Clips were selected to cover a range of benchmark scores within each task. Two authors provided 300 usable annotations (282 first ratings and 18 repeat showings; a further 13 records were flagged by the rater as unjudgeable and excluded). Raters saw a clip and its task description, rated completion and naturalness on five-point scales, and reported confidence in each on a three-point scale. Clips were presented in randomized order across tasks and domains, and their source was hidden. Raters were blind to a clip's source but not to the study's purpose. Both raters are authors of this paper, so they are more familiar with these tasks than a typical member of the population. Analysis of the rater reliability is shown in Table 12. This shows that the raters were consistent when considering the continuous nature of the rating scales, but have lower agreement when considering ratings as ordinal categories. Figure 13 shows the interface. Raters can pause a clip and see its progress; they cannot rewind, and clips loop, which encourages repeated viewing rather than scrubbing.

J.1 Clips and sample composition

The rated sample comprises 257 unique episode clips across 21 tasks, as shown in Figure 14.

J.2 Additional results analysis

Agreement varies sharply by domain (Table 13). Agreement is highest in MT10 tasks where the tasks are more simple and easier to judge visually while Gym Mujoco is harder due to the difficulty of visually evaluating metrics such as velocity. On Assistax, human completion ratings are higher than benchmark scores: the arm reaches the person and holds a plausible pose, and whether the tool

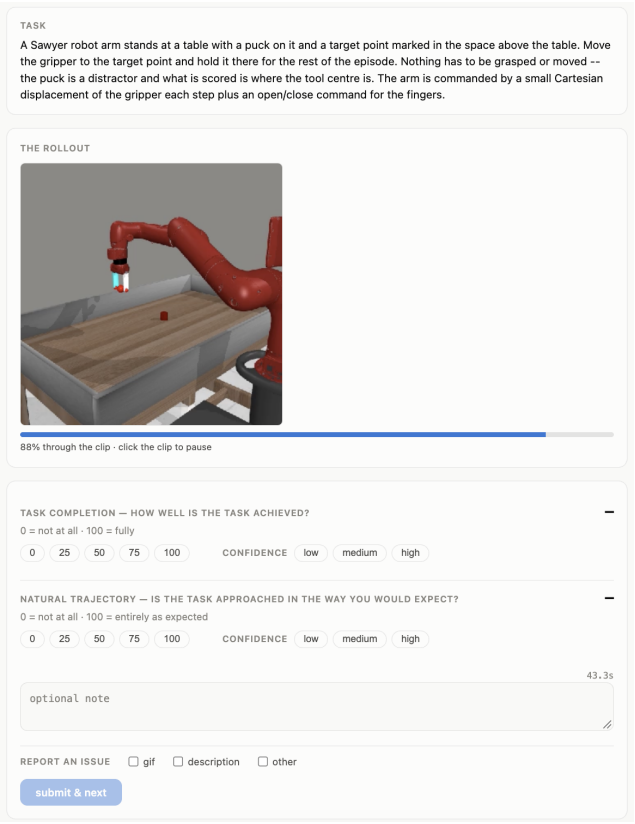


Figure 13: Screenshot of the annotation UI showing an MT10 task

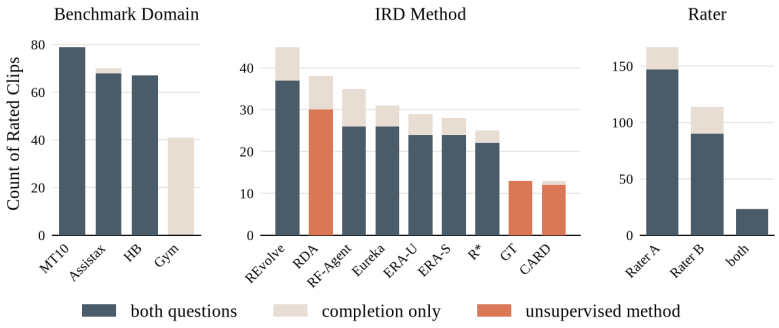


Figure 14: Breakdown of the composition of annotation data.

Table 13: Rank agreement between human completion ratings and the benchmark score, by benchmark domain. The benchmark score is the environment's own measure on the filmed episode, rescaled to $[0, 1]$; on HumanoidBench it is the shipped HumanoidBench reward of the filmed episode over the 700-return success bar, clipped. Intervals are 95% percentile bootstrap over 2000 resamples of tasks. †HumanoidBench has only two tasks, too few to resample, so its interval resamples clips and is correspondingly optimistic. The two naturalness rows pool the clips on which the second question was asked; Gym MuJoCo is completion-only by design.

domain	clips	tasks	mean benchmark	mean human	ρ	95% CI
Meta-World MT10	79	10	0.67	0.69	$+0.78$	$[+0.60, +0.89]$
Gym MuJoCo	41	5	0.70	0.64	$+0.57$	$[+0.38, +0.72]$
Assistax	70	5	0.24	0.45	$+0.36$	$[0.00, +0.76]$
HumanoidBench†	67	2	0.33	0.10	$+0.26$	$[+0.04, +0.47]$
all domains	257	22	0.47	0.46	$+0.58$	$[+0.37, +0.76]$
naturalness, pooled						
vs. benchmark score	214	17			$+0.47$	$[+0.17, +0.70]$
vs. human completion	214	17			$+0.83$	$[+0.73, +0.87]$

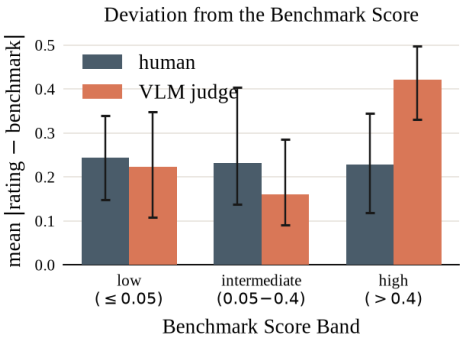


Figure 15: Mean absolute deviation between completion rating and benchmark score, by benchmark-score band, for human raters and the VLM judge (Opus 5). Lower is better.

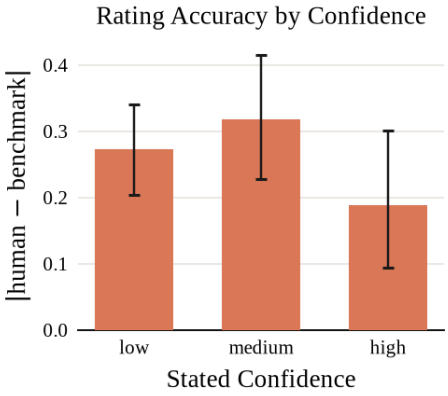


Figure 16: Rating accuracy by stated confidence: mean |human completion − benchmark score| per rating, with 95% percentile bootstrap confidence intervals. Lower is better.

contacts the target is often not resolvable at the available camera distance. The opposite is true for HumanoidBench where the reference reward pays for upright, forward-moving behavior that raters do not regard as walking or crawling.

Human completion ratings track the benchmark score about equally well at every level of performance: their mean deviation stays near 0.23 in all three bands (Figure 15). The VLM judge (Opus 5) matches humans on low-scoring policies and is slightly closer on intermediate ones. On high-scoring policies, however, its deviation almost doubles to about 0.42. So the judge is least reliable where it matters most: telling a good policy from a very good one. For unsupervised IRD methods, this means the ranking signal gets weaker as the trained policies get better, which could put a ceiling on the performance they can reach.

Stated confidence is at best a partial guide to accuracy. On average, high-confidence ratings are closest to the benchmark score (mean deviation about 0.19). The relationship is not monotonic, though: low-confidence ratings (0.27) are slightly more accurate than medium-confidence ones (0.32; Figure 16). All of these differences are tentative. The confidence intervals overlap substantially: the high-confidence interval (roughly 0.09–0.30) covers most of the low-confidence one, so no pairwise difference is clearly resolved at this sample size. With that caveat, the pattern still matters for human

review as an oversight mechanism. Reviewing only the low-confidence ratings would not work, because they are no worse than the medium-confidence ones. The data point, weakly, to a different rule: high-confidence ratings may be trusted somewhat more, so review effort is better spent on every rating the rater did not mark as high-confidence than on the low-confidence tail alone.

J.3 Statistical analysis

Cross-measure agreement is summarized by Spearman's ρ over clip-level scores, a clip's human score being the mean of its usable ratings. Confidence intervals are percentile bootstrap intervals from 2000 resamples of tasks, since clips within a task share an environment, a metric and often a search run; HumanoidBench contributes only two task clusters, so its intervals resample clips and are reported as such. Records carrying a rater's issue flag (13) and repeat showings (18) are excluded from the primary analyses and used only for the reliability statistics. The pooled comparison uses 257 clips across 22 tasks; the naturalness analyses use the 214 clips on which the second question was asked. High completion and low naturalness denote normalized clip means of at least 0.75 and at most 0.25 respectively: 75 clips are high-completion and 11 of those (15%) are also low-naturalness. Reliability coefficients use Krippendorff's α with an ordinal metric, which accommodates unordered pairs and the unequal numbers of repeat showings per rater.

K Bradley–Terry Rankings Without the Development Tasks

To verify our results, we also include the Bradley–Terry ranking on the tasks excluding the development tasks (Figure 17). We find that ERA-S and ERA-U still rank first and second, as in Section 4.3.

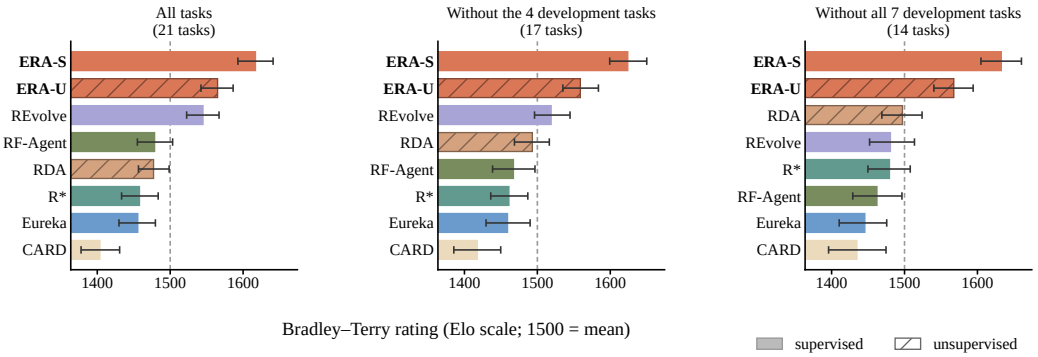


Figure 17: **Holding out the development tasks.** Pooled Bradley–Terry ratings over all 21 tasks (left), without the four MT10 tasks ERA-U was developed on (drawer-open, peg-insert-side, pick-place, push; middle), and without all seven tasks used at any point during development, which adds MT10 reach, HalfCheetah, and Swimmer (right).

L AI Disclosure

We used coding agents (mainly `claude-opus-5` and `claude-fable-5`) to help implement methods, polish writing, and brainstorm ideas. We also used them to help propose and refine hypotheses, design and provide feedback on research methodology and experiments, support data analysis, and interpret results. We have reviewed all AI-assisted work: AI-generated code was reviewed by the authors, and all cited literature was checked against the original papers. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.