MIMUS: POST-HOC REFINEMENT OF FROZEN REWARDS VIA RESIDUAL EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning systems often rely on rewards that are costly to redesign and may not fully capture task objectives. Inspired by inheritance and variation in natural evolution, we introduce MIMUS, a post-hoc reward refinement framework that freezes an existing base reward and evolves a compact, low-rank additive neural residual. The residual is initialized to output zero, ensuring that evolution starts exactly from the base reward. Before evolution, a Signal Gate checks whether candidate reward comparisons reveal a reliable optimization signal amid policy-training noise. The residual is subsequently evolved using Covariance Matrix Adaptation Evolution Strategy (CMA-ES), guided by teaching fitness computed from the native returns or task-aligned metrics of policies trained under candidate rewards. This approach enables reward optimization without residual supervision or differentiation through policy training. Beyond the evolutionary search, long-horizon audits guide reward selection for final policy training. We evaluate MIMUS on four tasks in Gymnax and MJX with handcrafted, LLM-generated, and learned neural base rewards. Across these tasks and reward sources, MIMUS improves downstream policy performance while leaving the base rewards unchanged, demonstrating that residual evolution offers a practical complement to reward design.

1 INTRODUCTION

Reinforcement learning (RL) has become a key driver of progress in Physical AI and robotics: coupled with high-fidelity simulation and sim-to-real transfer, it has enabled scalable high-dimensional control of quadruped and humanoid robots and is increasingly used to post-train robot foundation models from interaction experience (Hwangbo et al., 2019; Radosavovic et al., 2023; Physical Intelligence et al., 2025; Li et al., 2026). Yet the reward functions that drive RL remain heavily dependent on expert design and iterative tuning; even tasks with seemingly simple objectives often require staged rewards, curricula, and behavioral constraints to elicit the intended behavior (He et al., 2025; Xue et al., 2025). More fundamentally, a scalar reward often fails to fully express rich physical intent, leaving exploitable loopholes that allow a policy to obtain high return while deviating from the desired behavior—the familiar problems of reward misspecification and reward hacking (Amodei et al., 2016; Pan et al., 2022; Skalse et al., 2025). As simulators, policy optimizers, and robot foundation models continue to scale, reward design is therefore becoming a central bottleneck to further progress in robotics and Physical AI.

Automated reward design mitigates this bottleneck by treating the reward function itself as an optimization variable. Existing approaches optimize or select rewards within human-specified spaces (Sorg et al., 2010; Chiang et al., 2019; Faust et al., 2019; Zhang et al., 2025), use coding large language models to search for reward programs (Xie et al., 2024; Ma et al., 2024a), or learn neural rewards through meta-optimization (Zheng et al., 2018; Stadie et al., 2020; Lu et al., 2025). Yet many RL systems already possess a validated reward designed by experts, generated by an LLM, or learned by another method; re-optimizing it in its entirety enlarges the search space and may disrupt useful task structure. Prior work has either preserved a reward prior while learning additive corrections from human or VLM preferences (Cao et al., 2025; Hatgis-Kessell et al., 2026; Ni et al., 2026), or evolved neural proxies that augment or replace native rewards using policy-level performance (Ferreira et al., 2022). We consider a practical post-hoc setting in which the base reward remains frozen, no residual labels are available, and a candidate correction can be judged only by its

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.

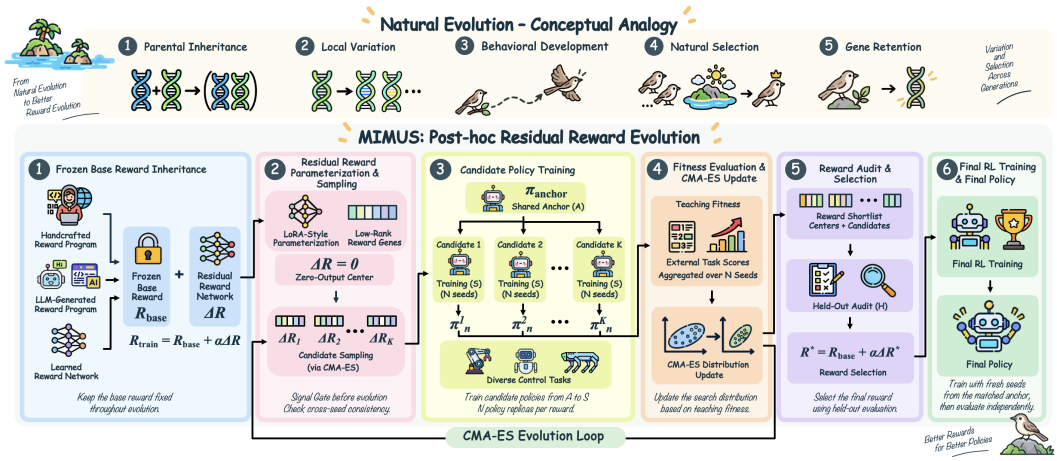

Figure 1: **Overview of MIMUS.** Top: a conceptual analogy to natural evolution. Bottom: MIMUS performs post-hoc reward refinement by evolving a low-rank neural residual on top of a frozen base reward, guided by downstream policy performance.

downstream teaching outcome after training a policy with it. **Can an existing reward be improved solely from the policy behavior it ultimately teaches?**

MIMUS originates from an intuition about natural evolution: inherited biological structure does not prescribe every action, but provides priors that make locomotion rapidly learnable; similarly, a good reward does not specify a policy, but guides trial and error toward useful behavior. Evolution preserves most inherited structure, introduces only limited and structured variation, and retains it according to fitness. Accordingly, MIMUS casts reward refinement as inheritance, variation, and selection: it freezes an existing reward, evolves only a residual in a compact low-rank reward-gene space, and selects variations by the policy phenotypes and downstream fitness they produce through RL. As a tribute to evolutionary thought, we name our framework MIMUS after *Mimus*, the genus of Galápagos mockingbirds whose island-specific variation helped inspire Darwin's early thinking about the evolution of species.

Building on this perspective, we introduce **MIMUS, a framework for post-hoc refinement of frozen rewards via residual evolution**. As illustrated in Figure 1, MIMUS places no restriction on the representation of the base reward: it may be an executable program written by experts or an LLM, or a learned neural reward model. MIMUS adds only a neural residual with a compact LoRA-style low-rank parameterization (Hu et al., 2022) and initializes it to produce exactly zero output, ensuring that evolution begins from the frozen base reward. At each generation, an evolution strategy produces candidate variations in the reward-gene space, each of which independently trains a policy from the same anchor. The resulting policy behavior serves as its phenotype and is evaluated using an environment-native return or a task-level objective. These evaluations guide the search toward residuals that teach better policies, improving downstream teaching outcomes while preserving the base reward.

Applying this idea to practical RL systems introduces two key challenges. First, a candidate residual can be evaluated only after a costly and stochastic reinforcement learning run, making an apparent single-run improvement difficult to distinguish from favorable training noise. Second, directly searching over a complete neural reward leads to a high-dimensional and expensive outer-loop optimization problem. MIMUS controls this search by freezing the base reward, initializing the residual at an exact zero-output center, and evolving only compact low-rank reward genes. It further separates reward-induced improvement from policy-training noise through a common policy anchor, multi-seed evaluation, signal validation, and long-horizon audits. We evaluate the method on JAX-native benchmarks in Gymnax and accelerated MuJoCo Playground tasks with MJX (Lange, 2022; Zakka et al., 2025). The frozen bases include manually written reward programs, LLM-generated rewards, and learned neural rewards. Our results show that **MIMUS can improve the policies taught by frozen rewards**, with benefits that depend on the base reward and training budget, and

long-horizon audits distinguish sustained reward quality from optimistic search-stage scores. Our main contributions are:

- **Representation-agnostic problem setting.** We formalize post-hoc refinement of frozen rewards, in which an existing reward of arbitrary form remains unchanged and is improved solely through the policy it ultimately teaches, without direct residual labels.
- **Post-hoc frozen-reward refinement.** We introduce MIMUS, which freezes an existing base reward and evolves an additive, low-rank neural residual whose zero-output initialization makes the initial combined reward exactly match the base reward. Candidate rewards are evaluated through policies trained from a shared anchor, without residual supervision or differentiation through policy training.
- **Generalization across reward representations.** We validate MIMUS on four Gymnax/MJX tasks and on handcrafted, learned neural, and LLM-generated base rewards, showing that each can be further evolved; multi-seed evaluation, Signal Gate, and long-horizon audits support reliable refinement.

2 Related Work

Search within predefined reward spaces. Early reward-design and AutoRL methods optimize parameterized rewards by the downstream performance of the resulting policy (Sorg et al., 2010; Chiang et al., 2019; Faust et al., 2019), and ORSO allocates training online to select among a finite set of candidate shaping rewards (Zhang et al., 2025). These methods automate reward tuning or selection but remain bounded by human-specified terms or candidates; MIMUS instead fixes an available base reward and searches a separate neural residual, allowing corrections beyond the original parameterization.

LLM-based reward program synthesis. Large language models expand reward search from numerical parameters to executable programs. Text2Reward generates dense reward code from task descriptions (Xie et al., 2024); Eureka combines coding LLMs, reward reflection, and evolutionary search (Ma et al., 2024a); DrEureka extends this paradigm toward robotic sim-to-real transfer (Ma et al., 2024b). MIMUS is complementary: it takes an LLM-generated program as a frozen base and refines it without modifying its internal logic.

Learning and optimizing neural rewards. Neural rewards remove the need for a fixed symbolic basis: prior work learns them by differentiating through policy updates or by formulating reward learning as a bi-level problem (Zheng et al., 2018; Stadie et al., 2020; Lu et al., 2025). Most closely related in its evaluation signal, Reward Networks use an outer-loop evolution strategy to optimize neural proxies that augment or replace native rewards, scoring each proxy by the agent trained under it (Ferreira et al., 2022). MIMUS shares the principle of judging rewards by the policies they teach, but treats an arbitrary working reward as immutable and evolves only a zero-initialized low-rank residual, so a reward learned by this family can serve as its frozen base. MIMUS does not differentiate through RL; shared anchors, multi-seed evaluation, and long-horizon audits instead make this search reliable under costly, stochastic RL training.

Residual reward refinement. Residual methods preserve existing reward knowledge through additive corrections: Residual Reward Models learn a residual from human preferences, Preference-Based Reward Repair learns transition-dependent corrections for misspecified proxies, and CoRe combines a formal reward with a VLM-preference residual (Cao et al., 2025; Hatgis-Kessell et al., 2026; Ni et al., 2026). MIMUS considers a different supervision regime: neither preferences nor residual targets are available, and each correction is evaluated only by the task-level fitness of the policy it trains. It therefore combines conservative residual refinement with outcome-driven black-box optimization, identifying the missing correction of a frozen base from policy-teaching outcomes.

3 Problem Formulation

Frozen-reward refinement. We consider a finite-horizon Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \rho_0, \gamma, T_{\text{ep}})$ with states $\mathcal{S}$, actions $\mathcal{A}$, dynamics P, initial-state distribution ρ_0, discount γ, and horizon T_{ep}. The system is equipped with an existing base reward $R_{\text{base}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. We

assume nothing about how this reward is represented or obtained: it may be an executable program written by a human or an LLM, or a learned neural reward model, and it is treated as immutable. We augment it with a learnable residual ΔR_θ and define the candidate training reward as

$$R_\theta(s_t, a_t, s_{t+1}) = R_{\text{base}}(s_t, a_t, s_{t+1}) + \alpha\, \Delta R_\theta(s_t, a_t, s_{t+1}), \tag{1}$$

where θ parameterizes the residual and $\alpha \geq 0$ controls its scale. Post-hoc refinement constrains all optimization to ΔR_θ and leaves R_{base} unchanged.

Policy-teaching objective. A candidate reward's quality is defined by the policy it teaches rather than by agreement with reward labels. Let Train_{RL} denote a fixed stochastic RL training procedure with a prescribed initialization or continuation protocol and budget, and let ξ collect its randomness in initialization, data collection, and optimization. Training under R_θ produces

$$\pi_\theta^\xi = \text{Train}_{\text{RL}}(\mathcal{M}, R_\theta; \xi). \tag{2}$$

We evaluate the resulting policy with a task-level objective $\mathcal{F}(\pi)$ such as an environment-native return or a predefined behavioral metric; the evaluator runs only after policy training and provides no target values or preference labels for the residual. The post-hoc reward-refinement problem is therefore

$$\theta^\star \in \arg\max_{\theta \in \Theta} J(\theta), \qquad J(\theta) = \mathbb{E}_\xi\left[\mathcal{F}\left(\pi_\theta^\xi\right)\right], \tag{3}$$

where $J(\theta)$ measures the expected downstream teaching outcome of the corrected reward. The objective is observed only by completing stochastic RL training runs, with neither residual supervision nor gradients through Train_{RL}; the problem is therefore nested and noisy black-box optimization. Refinement is judged by improvement over the frozen base reward across independent training runs.

4 MIMUS

MIMUS refines a frozen base reward by evolving a low-rank neural residual. The inner RL learner trains policies, while the outer covariance matrix adaptation evolution strategy (CMA-ES) (Hansen & Ostermeier, 2001) updates reward genes from their resulting fitness; a preliminary *Signal Gate* checks comparison reliability. Algorithm 1 in the appendix summarizes the complete procedure.

4.1 Low-Rank Residual Reward Genes

The residual network receives transition features, typically the concatenated current observation, clipped action, and next observation. A reward gene $\mathbf{z} \in \mathbb{R}^d$ contains the evolvable parameter entries; $\theta(\mathbf{z})$ reconstructs the full network parameters, including any fixed entries, and $R_{\mathbf{z}} := R_{\theta(\mathbf{z})}$. Inspired by LoRA (Hu et al., 2022), we factor each hidden-layer weight within the residual network as

$$\begin{aligned} &\mathbf{W}_\ell(\mathbf{z}) = \mathbf{A}_\ell(\mathbf{z})\mathbf{B}_\ell(\mathbf{z}), \quad \text{rank}(\mathbf{W}_\ell(\mathbf{z})) \leq r_\ell, \\ &\mathbf{A}_\ell(\mathbf{z}) \in \mathbb{R}^{d_\ell \times r_\ell}, \qquad\qquad \mathbf{B}_\ell(\mathbf{z}) \in \mathbb{R}^{r_\ell \times d_{\ell-1}}. \end{aligned} \tag{4}$$

Here d_0 is the input width, d_ℓ the hidden width, and r_ℓ the factorization rank. With small ranks, each matrix uses $r_\ell(d_{\ell-1} + d_\ell)$ parameters instead of $d_{\ell-1}d_\ell$; the factorization applies only to the residual.

The network uses $\tanh$ hidden activations and a dense scalar output head followed by $\tanh$, with fixed residual scale α. We randomly initialize the hidden factors and set the output-head weights and bias to zero, yielding

$$\Delta R_{\theta(\mathbf{z}_0)} \equiv 0, \qquad R_{\mathbf{z}_0} = R_{\text{base}}. \tag{5}$$

This **zero-output initialization** exactly preserves the base reward, while sampled perturbations can already produce nonzero corrections.

4.2 Budgeted Candidate Policy Training

We use PPO update indices $0 \leq A < S \leq H$ from training start: A is the shared anchor, S the selection endpoint, and H the long-horizon audit endpoint. Candidate policies are trained from A to S, and audited rewards from A to H.

For $A > 0$, every candidate starts from the same checkpoint trained under R_{base}; for $A = 0$, training starts from seed-matched initialization. Reusing the prefix avoids repeated early learning and reduces sensitivity to early randomness. S is typically before convergence, while H tests whether the refined advantage persists.

In generation g, each candidate reward is trained under N different policy-training seeds. Seed n under candidate k yields

$$\pi^{S}_{g,k,n} = \text{Train}^{S-A}_{\text{RL}}\Big(\mathcal{M}, R_{\mathbf{z}^{(k)}_g}; \mathcal{C}_A, \xi_{g,n}\Big), \tag{6}$$

where $\mathcal{C}_A$ specifies the shared checkpoint or $A = 0$ initialization, and $\xi_{g,n}$ controls training randomness. The N seed-specific evaluations are aggregated into the candidate fitness. Candidates share seed banks and restoration rules but maintain independent states; seed banks may stay fixed or refresh under a prespecified rule. Screening uses the same schedule as auditing through S.

4.3 Task-Aligned Teaching Fitness

Teaching fitness $\mathcal{F}(\pi)$ can be selected according to the task. The most direct choice is the environment's native episode return. For candidate k, each of its N training seeds yields a seed-specific fitness, and these values are aggregated into the candidate scalar used by CMA-ES; we use the mean by default.

For specific behavioral requirements, a useful template is *Gate Fitness*, which uses success rate to gate a task-dependent quality score:

$$\mathcal{F}_{\text{gate}}(\pi) = p_{\text{succ}}(\pi)\, \exp\left[-\sum_{j=1}^{M} c_j q_j(\pi)\right], \tag{7}$$

where p_{succ} is success rate and q_j are normalized nonnegative task costs, estimated from evaluation rollouts. The M components, their transformations, and weights $c_j \geq 0$ are task-specific; costs may measure completion time, control effort, or task-execution error. However, gating can amplify training noise, so the teaching fitness should be adapted to the specific task.

4.4 Signal Gate Before Evolution

Stochastic policy learning can obscure reward-induced differences. Before evolution, Signal Gate evaluates a fixed population around the initial center, including mirrored perturbations, across multiple seeds without updating CMA-ES, and checks whether candidate comparisons are reproducible.

Diagnostics cover residual scale and saturation, cross-seed rank agreement, mirrored-direction consistency, and candidate effects relative to candidate–seed interactions. They set the fitness, initial scale σ_0, population size K, policy-training seed count N, and training budget. More training seeds improve repeated measurement; a larger population explores more directions. The configuration is checked with fresh perturbations and seeds before the protocol is fixed. If a usable signal remains unavailable within the compute budget, we revisit the protocol or defer evolution.

4.5 CMA-ES Reward Evolution

After signal validation, CMA-ES maintains a mean $\mathbf{m}_g \in \mathbb{R}^d$, covariance shape matrix $\mathbf{C}_g \in \mathbb{R}^{d\times d}$, and global step size $\sigma_g > 0$ in the evolvable gene space. Starting from $\mathbf{m}_0 = \mathbf{z}_0$, it samples K candidates per generation:

$$\mathbf{z}^{(k)}_g = \mathbf{m}_g + \sigma_g \mathbf{C}^{1/2}_g \boldsymbol{\epsilon}^{(k)}_g, \qquad \boldsymbol{\epsilon}^{(k)}_g \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d). \tag{8}$$

The sampling covariance is $\sigma^2_g \mathbf{C}_g$, and $\mathbf{I}_d$ is the identity matrix. For each candidate, the N seed-specific teaching-fitness values are aggregated by the predeclared rule; the mean is the default. CMA-ES then ranks candidates and updates the mean, covariance shape, and step size; each generation restarts from the same anchor, and only reward genes evolve.

4.6 Long-Horizon Reward Audit and Selection

CMA-ES selects candidates using fitness measured from A to S. This short-horizon signal is efficient, but performance at S need not predict how a reward behaves after training to H. After evolution, we choose the final reward or rewards to audit and train them from A to H, comparing their performance at H with their S endpoint to detect long-horizon degradation.

The audit target is task-dependent. By default, we audit the final CMA-ES center reward $R_{\mathbf{m}_G}$. Alternatives include the best sampled candidate or a preferred nondominated reward selected from the empirical Pareto front of audited policy metrics. The selection metric and preference are fixed before auditing.

Let $\mathbf{z}^\star$ denote the selected gene. The retained reward is

$$R^\star = R_{\text{base}} + \alpha\, \Delta R_{\theta(\mathbf{z}^\star)}. \tag{9}$$

If the long-horizon audit shows no degradation, $R^\star$ is used as the final reward function for policy training and evaluation.

5 Experiments

We evaluate reward refinement through four stages: experimental setup, refinement of handcrafted rewards and longer-horizon audits (Figure 2, Table 1), component ablations (Figure 3), and refinement of learned and LLM-generated baselines (Figure 4).

5.1 Experimental Setup

Tasks and policy learning. We use Reacher-misc and DeepSea-bsuite from Gymnax and PointMass and FingerTurnEasy from MuJoCo Playground with MJX, covering reaching, sparse discrete exploration, acceleration control, and contact manipulation. Each candidate reward trains $N = 4$ policy-training seeds whose fitness values are aggregated into one candidate fitness. Reacher-misc and DeepSea-bsuite start at $A = 0$, and PointMass and FingerTurnEasy start from frozen checkpoints at $A = 3$. Selection uses $S = 200, 10, 10, 10$, with candidate policies independent across rewards and generations. All runs use NVIDIA RTX 4090 and RTX 5090 GPUs.

Reward parameterization and search. The main search uses rank-4 zero-output residuals with $\alpha = 1$. The residual network has two width-16 $\tanh$ hidden layers and a scalar output head, giving active gene dimensions of 313, 2,297, 280, and 344. CMA-ES uses $K = 64$ on Reacher-misc and $K = 32$ elsewhere, with generation budgets 50/20/50/50; figures show 0–49 (0–19 for DeepSea-bsuite). Search scales, seed banks and selection budgets are in Appendix C.

Evaluation metrics. The external fitness is fixed while the training reward evolves, and its components and weights are set before evolution: Reacher-misc combines success, distance, and control effort; PointMass combines stable arrival, time, and effort; DeepSea-bsuite uses success with a progress tie-break during selection; FingerTurnEasy combines occupancy with bounded distance and control-quality terms (Appendix B). Scores are compared only within tasks.

Signal Gate. Before evolution, a fixed-population Signal Gate around the zero-residual center, including mirrored perturbations, checks cross-seed reproducibility through split-half rank agreement, directional consistency, the candidate-to-interaction RMS ratio, and residual saturation, and fixes the fitness, σ_0, K, N, and budget. Passing the gate establishes that a local comparison signal is usable; final conclusions rest on the long-horizon audits. Details and exceptions are in Appendix D.

5.2 Refinement of Handcrafted Rewards

Here the frozen base reward is the task's native reward, and we test whether an evolved residual improves search-stage fitness and whether the selected reward's teaching quality persists at the long-horizon audit.

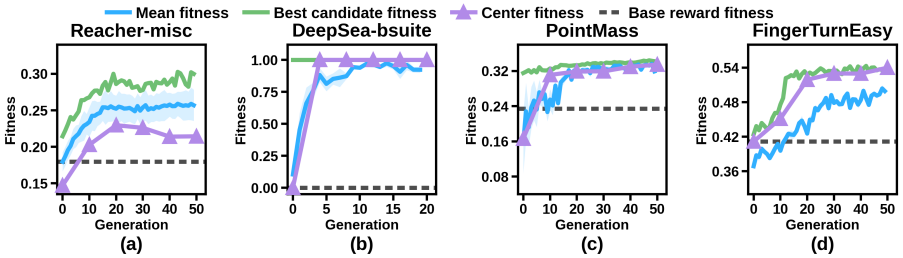

Figure 2: **Evolution with frozen handcrafted rewards.** Panels **a–d** show the four tasks. Blue/green/purple curves denote mean, best-candidate, and center-audit fitness; dashed lines are matched base rewards. Bands show SD across four training-seed means.

Improvement during evolution. Figure 2 tracks three quantities against each task's native reward: the mean fitness over the K candidates, the best-candidate fitness within each generation, and the fitness of the CMA-ES center reward gene evaluated by an independent center audit. The best-candidate curve stays above the native reward level throughout evolution, and both the population mean and the audited center end above it. Final mean fitness reaches 0.257 on Reacher-misc, 0.923 success on DeepSea-bsuite, 0.319 on PointMass, and 0.499 on FingerTurnEasy. On PointMass, the audited center improves from 0.166 to 0.334. Because the continuous-control scores measure execution quality, these gains are not merely higher accumulated training reward, and the center audit uses a policy-training seed bank distinct from evolution.

Table 1: Long-horizon reward audit (S and H are PPO update indices).

	Reacher-misc		DeepSea-bsuite		PointMass		FingerTurnEasy	
	S: 200	H: 500	S: 10	H: 100	S: 10	H: 20	S: 10	H: 43
Final center fitness	0.236	0.303	1.000	1.000	0.343	0.342	0.540	0.535
Best candidate fitness	0.234	0.273	1.000	1.000	0.344	0.333	0.543	0.526

Longer-horizon reward audit. We train every task to H, the policy-convergence update, and re-evaluate the final center and best-candidate rewards together with their S endpoints (Table 1), retraining both from the same task-specific anchor with $H = 500, 100, 20, 43$. This tests whether teaching fitness survives convergence.

Fitness changes only modestly from S to H, with some tasks improving and others decreasing slightly. Thus S is an efficient CMA-ES selection point that greatly reduces policy-training cost without large degradation at H.

5.3 Ablation Studies

Figure 3 compares four components under matched task-level populations and policy budgets. We report normalized AUC (trapezoidal area over the plotted generations divided by interval length), which measures average search performance rather than policy-learning speed; Appendix E gives full summaries, Table 7 lists the interventions, and Table 15 the width-64 gene dimensions.

Frozen base reward prior. Figure 3a–d tests whether preserving the frozen base reward is useful: it removes the base reward and uses the evolved reward network as the entire training reward. This lowers AUC on all tasks. Endpoint fitness also drops on Reacher-misc (0.257 versus 0.012), PointMass (0.319 versus 0.303), and FingerTurnEasy (0.499 versus 0.403), most clearly on the first and third. DeepSea-bsuite is the exception: it is near saturation and seed-sensitive, so network-only evolution can slightly exceed final success (0.992 versus 0.923) despite lower AUC. Preserving the base prior still gives better average search progress.

Outer optimizer. Figure 3e–h replaces CMA-ES with OpenAI-ES (Salimans et al., 2017), keeping the reward representation and search budgets fixed. CMA-ES adapts covariance, making evolu-

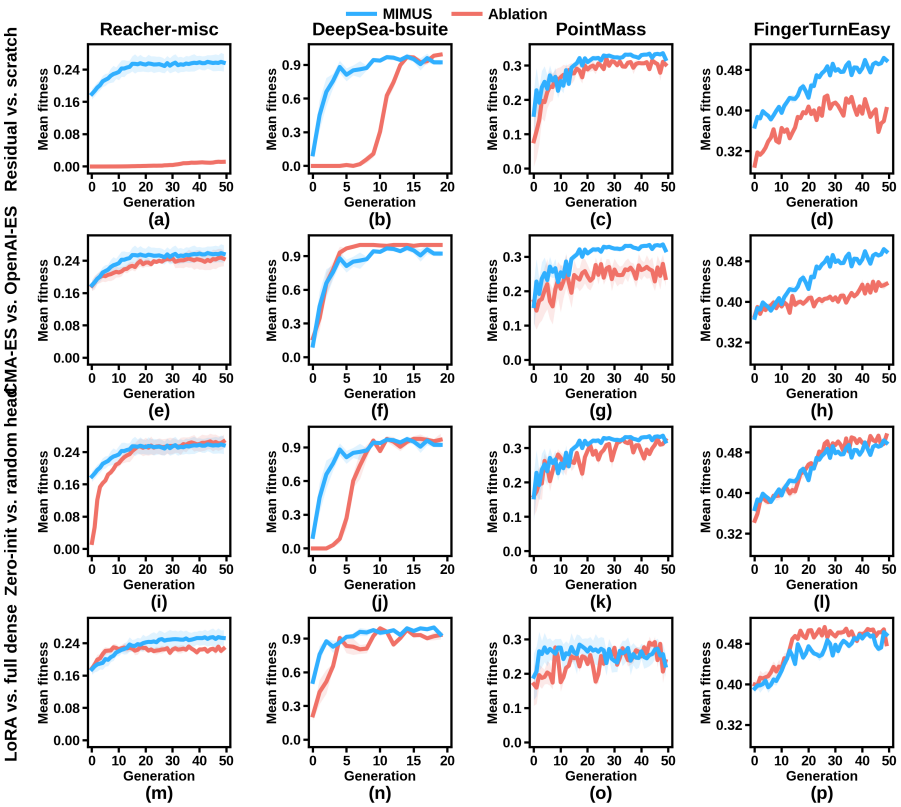

Figure 3: **Component comparisons.** Columns are tasks; rows compare residual/scratch rewards, CMA-ES/OpenAI-ES, zero/random heads, and rank-4/dense representations. Curves are population means with SD across four training-seed means.

tion directional and suited to lower-dimensional gene spaces. It has higher AUC on Reacher-misc, PointMass, and FingerTurnEasy with the same endpoint ordering. DeepSea-bsuite has a much larger reward-gene space; OpenAI-ES is slightly better there in AUC (0.907 versus 0.850) and final success (1.000 versus 0.923), but the gap is small. We therefore use CMA-ES by default and OpenAI-ES for particularly large gene spaces.

Zero-output initialization. Figure 3i–l compares zero-output initialization with a random output head while retaining the base reward. Zero initialization exactly recovers the base reward at the initial center, whereas random heads add an unvalidated correction. Final means are similar, with random heads slightly higher, but zero initialization has higher AUC on Reacher-misc (0.245 versus 0.233), DeepSea-bsuite (0.850 versus 0.661), and PointMass (0.301 versus 0.278), indicating faster average improvement; FingerTurnEasy is the only exception (0.456 versus 0.451). Zero output thus provides exact initial identity and better search performance on most tasks.

Low-rank representation. Figure 3m–p widens both hidden layers to 64 and compares rank-4 factors with dense matrices of the same width. Rank-4 reduces the gene from 5,441 to 1,033 serialized coordinates on Reacher-misc (5.3×) and from 37,185 to 3,017 on DeepSea-bsuite (12.3×); FingerTurnEasy uses 1,065 versus 5,953 (Table 15). The smaller gene shrinks the CMA-ES covariance state and reduces storage/search cost; DeepSea-bsuite's dense covariance exceeded memory, so both arms use diagonal covariance. Low rank raises both AUC and endpoint on Reacher-misc and DeepSea-bsuite, improves PointMass AUC (0.259 versus 0.236) with a small endpoint gap, and raises FingerTurnEasy endpoint (0.498 versus 0.479) with slightly lower AUC (0.459 versus 0.480). Every task thus improves on at least one metric.

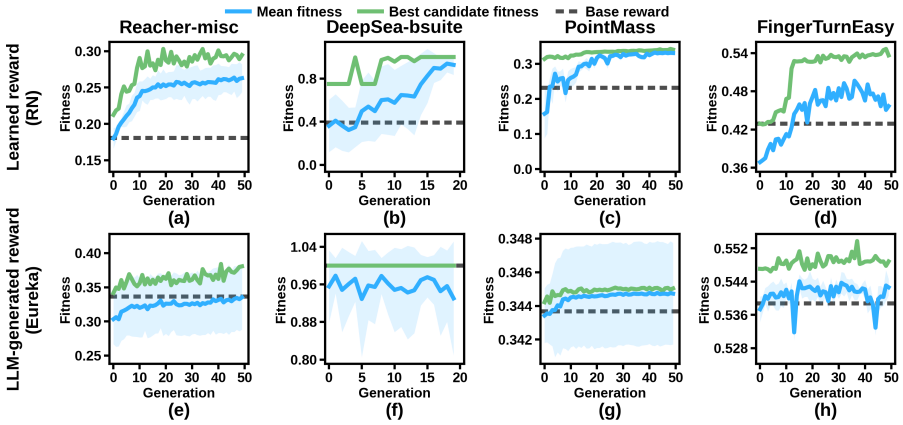

Figure 4: **Refining learned RN and LLM-generated Eureka rewards.** Top/bottom rows are RN/Eureka; columns are tasks. Blue/green curves denote population mean/best-candidate fitness; dashed lines are matched base reward, and bands show SD across four training-seed means.

5.4 Baseline

We use Reward Networks (RN) (Ferreira et al., 2022) and Eureka (Ma et al., 2024a) as frozen bases, producing RN+MIMUS and Eureka+MIMUS. MIMUS changes neither the RN parameters nor the Eureka program. RN uses a width-128 state-potential network with its wrapper and scaling; Eureka uses `DeepSeek-V4.1-Flash` for five iterations of 16 generated programs, and Appendices C.5 and C.6 record the construction settings. Reported numbers are the population mean and the best candidate of the final generation.

Learned neural rewards. RN+MIMUS exceeds the RN reference on both statistics in every task: Reacher-misc $0.181 \rightarrow 0.263/0.295$; DeepSea-bsuite $0.393 \rightarrow 0.927/1.000$; PointMass $0.232 \rightarrow 0.331/0.340$; FingerTurnEasy $0.429 \rightarrow 0.457/0.536$.

LLM-generated rewards. Eureka+MIMUS raises the best-candidate fitness above the Eureka reference on all four tasks: Reacher-misc $0.337 \rightarrow 0.381$, PointMass $0.3437 \rightarrow 0.3450$, FingerTurnEasy $0.539 \rightarrow 0.549$, and DeepSea-bsuite ties the saturated reference at 1.000. The population mean also rises on PointMass ($0.3437 \rightarrow 0.3447$) and FingerTurnEasy ($0.539 \rightarrow 0.543$), and stays level on Reacher-misc (0.335 versus 0.337); on DeepSea-bsuite the reference is already saturated. Every Eureka base therefore remains a starting point that MIMUS continues to refine, with the improvement appearing most clearly in the best candidate.

6 Conclusion

To address the difficulty of designing effective rewards for reinforcement learning, we introduced MIMUS, a post-hoc refinement framework that freezes an existing base reward and evolves a low-rank neural residual initialized to zero output. A Signal Gate assesses whether candidate comparisons reveal a reliable optimization signal amid policy-training noise, and CMA-ES evolves residuals using teaching fitness derived from downstream policy outcomes. Experiments on four Gymnax and MJX tasks show that refining handcrafted, LLM-generated, and learned neural rewards improves downstream policy performance. The A/S/H protocol reduces policy-training costs during evolution, while long-horizon audits show that the gains largely persist to the target training horizon. Ablations clarify the roles of the base reward prior, outer optimizer, zero-output initialization, and low-rank parameterization. These findings demonstrate the value of post-hoc refinement: rewards produced by different design methods can serve as starting points for further policy improvement while their original formulations remain intact. A remaining limitation is that high policy-training noise can obscure reward-induced performance differences, preventing MIMUS from identifying a reliable optimization direction. Future work will investigate more noise-robust reward evaluation and extend the framework to more complex robotic reinforcement learning tasks.

AI USE STATEMENT

This work used generative AI tools to assist with code development and debugging, with the orchestration and monitoring of training runs, with the generation of figures, and with the drafting and editing of manuscript text. All AI-assisted code was reviewed and validated by the authors through the experiments reported here, and every figure and numerical result was regenerated from the underlying run artifacts, manifests and evaluation summaries. Generative AI tools were not used to select which results to report, and no AI tool is an author of this work. Separately, one of the frozen base rewards studied in Section 5.4 is itself a reward program generated by a large language model as part of the experimental protocol; that reward is a studied object rather than an assistant in the writing process. The authors take full responsibility for the content of this work, including all text, claims and artifacts.

ETHICS STATEMENT

This work studies automated reward refinement for reinforcement learning and is evaluated primarily in simulated environments. The experiments do not involve human subjects, personal data, or sensitive information. Automated reward optimization may produce unintended behaviors when the specified objective does not fully capture the desired behavior. We therefore emphasize explicit evaluation criteria, safety-related constraints, and independent evaluation on fresh seeds. Real-world deployment of such methods should include additional safety validation and human oversight.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the environments, reward formulations, optimization procedures, policy training configurations, and evaluation protocols in the main paper and supplementary material. All experiments are conducted with multiple random seeds, and the corresponding hyperparameters and implementation details are reported. We plan to release the source code, configuration files, and evaluation scripts upon publication to facilitate reproduction of our results.

REFERENCES

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016. URL https://arxiv.org/abs/1606.06565.

Chenyang Cao, Miguel Rogel-García, Mohamed Nabail, Xueqian Wang, and Nicholas Rhinehart. Residual reward models for preference-based reinforcement learning, 2025. URL https://arxiv.org/abs/2507.00611.

Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl, 2019. URL https://arxiv.org/abs/1809.10124.

Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving rewards to automate reinforcement learning, 2019. URL https://arxiv.org/abs/1905.07628.

Fabio Ferreira, Thomas Nierhoff, Andreas Sälinger, and Frank Hutter. Learning synthetic environments and reward networks for reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=C1_esHN6AVn.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

Stephane Hatgis-Kessell, Logan Mondal Bhamidipaty, and Emma Brunskill. Repairing reward functions with feedback to mitigate reward hacking, 2026. URL https://arxiv.org/abs/2510.13036.

Tairan He, Zi Wang, Haoru Xue, Qingwei Ben, Zhengyi Luo, Wenli Xiao, Ye Yuan, Xingye Da, Fernando Castañeda, Shankar Sastry, Changliu Liu, Guanya Shi, Linxi Fan, and Yuke Zhu. Viral: Visual sim-to-real at scale for humanoid loco-manipulation, 2025. URL https://arxiv.org/abs/2511.15200.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), January 2019. ISSN 2470-9476. doi: 10.1126/scirobotics.aau5872. URL http://dx.doi.org/10.1126/scirobotics.aau5872.

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL https://github.com/RobertTLange/gymnax.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Yang Zhaohui, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=TQhSodCM4r.

Renzhi Lu, Zonghe Shao, Yuemin Ding, Ruijuan Chen, Dongrui Wu, Housheng Su, Tao Yang, Fumin Zhang, Jun Wang, Yang Shi, et al. Discovery of the reward function for embodied reinforcement learning agents. *Nature Communications*, 16(1):11064, 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. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=IEduRUO55F.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language Model Guided Sim-To-Real Transfer. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024b. doi: 10.15607/RSS.2024.XX.094.

Hexian Ni, Tao Lu, and Yinghao Cai. Core: Combined rewards with vision-language model feedback for preference-aligned reinforcement learning, 2026. URL https://arxiv.org/abs/2607.01721.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models, 2022. URL https://arxiv.org/abs/2201.03544.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a vla that learns from experience, 2025. URL https://arxiv.org/abs/2511.14759.

Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Real-world humanoid locomotion with reinforcement learning, 2023. URL https://arxiv.org/abs/2303.03381.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. URL https://arxiv.org/abs/1703.03864.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking, 2025. URL https://arxiv.org/abs/2209.13085.

Jonathan Sorg, Richard L Lewis, and Satinder Singh. Reward design via online gradient ascent. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (eds.), *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/168908dd3227b8358eababa07fcaf091-Paper.pdf.

Bradly Stadie, Lunjun Zhang, and Jimmy Ba. Learning intrinsic rewards as a bi-level optimization problem. In Jonas Peters and David Sontag (eds.), *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pp. 111–120. PMLR, 03–06 Aug 2020. URL https://proceedings.mlr.press/v124/stadie20a.html.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=tUM39YTRxH.

Haoru Xue, Tairan He, Zi Wang, Qingwei Ben, Wenli Xiao, Zhengyi Luo, Xingye Da, Fernando Castañeda, Guanya Shi, Shankar Sastry, Linxi "Jim" Fan, and Yuke Zhu. Opening the sim-to-real door for humanoid pixel-to-action policy transfer, 2025. URL https://arxiv.org/abs/2512.01061.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. MuJoCo Playground, 2025. URL https://arxiv.org/abs/2502.08844.

Chen Bo Calvin Zhang, Zhang-Wei Hong, Aldo Pacchiano, and Pulkit Agrawal. Orso: Accelerating reward design via online reward selection and policy optimization. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 38981–39024, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/6155cfb4e650d41167848fe2ead30ab8-Paper-Conference.pdf.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/51de85ddd068f0bc787691d356176df9-Paper.pdf.

A Additional Method Details

A.1 Optimization Objects and Notation

There are three distinct objects in the nested optimization. The base reward R_{base} is a frozen artifact. The reward gene $\mathbf{z}$ determines the trainable residual through $\theta(\mathbf{z})$. The policy parameters are optimized by the inner RL learner under $R_{\text{base}} + \alpha\Delta R_{\theta(\mathbf{z})}$. Only $\mathbf{z}$ is passed to the outer optimizer. Policy parameters, value-function parameters, and normalization statistics stay outside the reward gene and are rebuilt for every candidate. Table 2 lists the symbols and units used throughout the appendix.

We use K for the candidate population size, N for the number of policy-training seeds per candidate, and G for the number of outer generations. The residual multiplier α changes the reward supplied to PPO. The CMA-ES step size σ_g changes the distribution of sampled genes. They act at different levels of the search. The implementation sometimes uses different letters for population size and policy-training seed count; throughout the manuscript the meanings above are fixed.

Table 2: Notation and units. A, S, and H count PPO updates from training start; transition counts are reported separately.

Symbol	Meaning
d	Number of active scalar coordinates in the reward gene.
K, N, G	Candidates per generation, policy-training seeds per candidate, and outer generations.
α, σ_0	Residual reward multiplier and initial coordinate-space search standard deviation.
A	Shared policy-anchor update; zero for training from scratch.
S, H	Absolute PPO-update indices for candidate selection and longer-horizon reward audit; candidate policies train from A to S, and audited rewards from A to H.
T_{ep}	Episode limit in environment control steps.

A.2 Reward-Gene Construction

The continuous-control residual takes transition features formed from the current observation, bounded action, and next observation. For discrete actions, an action encoding replaces the continuous action vector. These inputs are distinct from the normalized observations supplied to the policy: the reward-feature transformation is fixed by the reward specification. In particular, the transition residual is computed from the environment transition alone, so the external evaluation score stays outside the training reward.

For L hidden layers, write $\mathbf{h}_0 = \phi(s, a, s')$ and

$$\mathbf{h}_\ell = \tanh(\mathbf{A}_\ell \mathbf{B}_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_\ell)\,, \qquad \Delta R_{\theta(\mathbf{z})} = \tanh\big(\mathbf{w}_{\text{out}}^\top \mathbf{h}_L + b_{\text{out}}\big)\,. \tag{10}$$

The gene serializes the evolvable factors, hidden biases, and output-head entries in a fixed order. Inactive coordinates are reconstructed from their frozen values before evaluation. With all biases present and all entries active, the low-rank and dense parameter counts are

$$d_{\text{LR}} = \sum_{\ell=1}^{L} \big[r_\ell(d_{\ell-1} + d_\ell) + d_\ell\big] + d_L + 1, \tag{11}$$

$$d_{\text{dense}} = \sum_{\ell=1}^{L} \big[d_{\ell-1} d_\ell + d_\ell\big] + d_L + 1. \tag{12}$$

The active dimension subtracts any frozen entries from these counts. For example, the PointMass $10 \to 16 \to 16 \to 1$ rank-4 residual has 281 serialized parameters and 280 active coordinates because its output bias is frozen. Its same-width dense counterpart has 465 parameters before any coordinate freezing, so the compression ratio follows from the rank together with the frozen coordinates.

The low-rank factors parameterize the residual network, which is added to a frozen base reward. A programmatic base and a neural base use the same additive interface. The evolvable gene holds the

complete parameter set of that network: both factors of every low-rank layer, the hidden biases, and the output head.

A.3 Zero-Output Initialization and Its Scope

We initialize hidden factors to fixed random values and set the output head to zero. Equation 10 then gives $\Delta R_{\theta(\mathbf{z}_0)} = 0$ for every input, and hence $R_{\mathbf{z}_0} = R_{\text{base}}$. The identity holds at the reward-function level, and reproducing the trajectory of a policy trained on the base reward additionally requires matched learner state, randomness, and numerical execution.

The output bound gives $|R_{\mathbf{z}} - R_{\text{base}}| \leq \alpha$ per transition for this direct residual, and small reward changes can alter exploration and the policy reached by PPO. The bound therefore describes the reward channel, while the learning dynamics determine the policy outcome. Zero-output initialization fixes the exact starting point: the center reproduces the base reward, and the perturbed candidates of the first generation explore nonzero residuals.

A.4 Budgets, State Restoration, and Audits

We follow Section 4.2: A, S, and H are absolute PPO-update indices from training start, with $A < S \leq H$. The anchor is at A, candidate fitness is measured at S, and the retained reward is audited at H. Candidate policies are trained from A to S, and audited rewards from A to H. The episode horizon T_{ep} is separate. Historical manifests may record environment transitions instead of updates; we convert them using the task-specific transitions per update and check whether each value denotes an endpoint or a duration.

For the main PointMass and FingerTurnEasy continuation runs, one PPO update contains 983,040 environment transitions. The anchor is at $A = 3$ (2,949,120 transitions); candidates are trained from A to $S = 10$ (9,830,400 transitions). The interval from A to S contains seven updates, or 6,881,280 transitions. Later audits of retained rewards are separate evaluations that keep the selection endpoint fixed.

An anchor protocol specifies policy and critic parameters, normalization state, optimizer and schedule handling, and environment-reset semantics. Loading the full state and deliberately restarting an optimizer are different continuation protocols. The same rule applies to every candidate and to the matched base reward control. Evaluation freezes observation-normalization statistics and excludes the evolved residual from the external task score.

Algorithm 1 gives the intended reward-selection procedure. The experiment report distinguishes which evaluations provide independent validation and which reuse the selection seed bank for a horizon diagnostic. Repeating an evaluator with the same policy and random seed checks numerical repeatability and adds no new policy-training seed. An audit used to select a generation or candidate is validation evidence, and a final test is independent when its seeds are unused for those choices.

B Tasks and Reward Specifications

The four-task study covers continuous reaching, discrete exploration, acceleration control, and contact-based manipulation. Table 3 summarizes the observation, action, transition-feature, and episode dimensions together with the fitness function of each task. Task definitions and external evaluators remain fixed within each comparison. Changing the source of the base reward changes the inner training reward while the external task score keeps its meaning. The implemented fitness functions are specified below; Equation 7 is the template that each task instantiates.

B.1 Reacher-misc

Reacher-misc uses the two-link reaching task in Gymnax, with eight observation entries, two bounded continuous actions, and 100-step episodes. Success means reaching within distance 0.2 of the goal at least once during an episode. Let p be the fraction of successful episodes, $\bar{d}_{\min}$ the average closest goal distance over episodes, and $\bar{u}$ the episode-averaged sum of squared bounded

Algorithm 1 MIMUS: residual evolution and reward selection

Require: Task $\mathcal{M}$, frozen base R_{base}, residual scale α, anchor protocol $\mathcal{C}_A$, absolute update indices $A < S \leq H$.
Require: Population K, policy-training seeds N, generations G, initial scale σ_0, fitness and aggregation rule, seed and audit protocols.
Ensure: Retained reward $R^\star$.
Initialize $\mathbf{z}_0$ with zero residual output; validate the comparison protocol using Signal Gate.
If no usable signal is found, defer evolution and **return** R_{base}.
Fix the protocol; initialize $(\mathbf{m}_0, \mathbf{C}_0, \sigma_0) = (\mathbf{z}_0, \mathbf{I}_d, \sigma_0)$ and audit archive $\mathcal{Q} = \varnothing$.
for $g = 0, \ldots, G-1$ **do**
  Sample genes $\mathbf{z}_g^{(1:K)}$ from the current search distribution.
  for $k = 1, \ldots, K$ **do**
    for $n = 1, \ldots, N$ **do**
      Restore an independent learner under $\mathcal{C}_A$ with seed-specific randomness $\xi_{g,n}$.
      Train with $R_{\text{base}} + \alpha \Delta R_{\theta(\mathbf{z}_g^{(k)})}$ and evaluate at S.
    end for
    Aggregate the N seed-specific evaluations into selection fitness $\widehat{f}_g^{(k)}$.
  end for
  Update $(\mathbf{m}_{g+1}, \mathbf{C}_{g+1}, \sigma_{g+1})$ after all candidates finish.
  Save genes, selection records, and optimizer/random-number state.
  if an audit is scheduled or $g = G-1$ **then**
    Shortlist $\mathbf{m}_{g+1}$ and any candidates allowed by the selection protocol.
    Train the selected reward from $\mathcal{C}_A$ to H with the audit seeds, compare its fitness at H against its value at S to check long-horizon degradation, and store evaluations in $\mathcal{Q}$.
  end if
end for
Select $\mathbf{z}^\star$ under the declared rule; retain $\mathbf{z}_0$ if improvement is unsupported.
Fix $R^\star = R_{\text{base}} + \alpha \Delta R_{\theta(\mathbf{z}^\star)}$ before final fresh-seed testing.
return $R^\star$.

Table 3: Task interfaces. Reward features concatenate observation, action encoding, and next observation.

Task	Obs.	Actions	Features	T_{ep}	Fitness
Reacher-misc	8	2	18	100	Equation 13
DeepSea-bsuite	256	2 (discrete)	514	16	Equation 14
PointMass	4	2	10	1,000	Equation 15
FingerTurnEasy	12	2	26	1,000	Equation 17

action components. The implemented gate is

$$F_{\text{R}} = p \exp\left[-\left(\frac{\bar{d}_{\min}}{0.2} \right)^2 - 0.25 \frac{\bar{u}}{1.0} \right]. \tag{13}$$

Distances and action costs include failed episodes. Squaring the aggregate closest distance differs from averaging squared distances; Equation 13 uses the former, matching the evaluator. Action cost serves as a control-effort proxy, and native episode return is reported separately. The residual receives 18 transition features: two eight-dimensional observations and the two-dimensional action.

B.2 DeepSea-bsuite

DeepSea-bsuite uses the Gymnax DeepSea task on a grid of size 16, with deterministic dynamics and a fixed action map in the reported configuration. Its flattened observation has 256 entries, and the two actions are encoded with two entries for the residual. Concatenating current observation, action encoding, and next observation gives 514 reward features. Evaluation uses the policy's highest-probability action and frozen normalization statistics.

The evaluator identifies success by native episode return exceeding 0.5. The primary reported metric is success rate p. For outer-loop ranking only, an all-failure evaluation receives a small progress

term:

$$F_{\mathrm{D}} = \begin{cases} p, & p > 0, \\ 10^{-3}\bar{q}, & p = 0, \end{cases} \tag{14}$$

where $\bar{q} \in [0, 1]$ is the average maximum normalized progress reached during an episode. The condition $p = 0$ applies to the evaluated candidate aggregate. The progress term distinguishes failed candidates, and curves explicitly labeled success rate report p; the failure tie-break applies only inside the outer-loop ranking.

B.3 POINTMASS

PointMass uses the acceleration-controlled PointMass task in the MJX environment of MuJoCo Playground, with a ten-dimensional transition input. The formal gate requires the point mass to remain within radius 0.0075 of its goal for 20 consecutive control steps during a 1,000-step episode. Let p denote the fraction satisfying this condition and $\bar{\tau}_{\mathrm{succ}}$ the mean time of the first qualifying hit among successful episodes. Let $\bar{u}$ be the recorded mean squared-action cost over the evaluated episodes. The time/effort gate is

$$F_{\mathrm{P}}^{\mathrm{gate}} = p \exp\Big[-\frac{\bar{\tau}_{\mathrm{succ}}}{250} - 0.25\frac{\bar{u}}{0.05}\Big]. \tag{15}$$

The implemented exponential argument is clipped to $[-60, 60]$ for numerical stability. Time is measured in control steps, and distance enters through the success predicate of this time/effort variant.

For an evaluation with no successful episodes, selection uses $10^{-3}\bar{q}$ instead of the zero gate, with per-episode progress

$$q_e = \mathrm{clip}\left(\frac{\max(d_{e,0} - d_{e,\min}, 0)}{d_{e,0}}, 0, 1\right). \tag{16}$$

The implementation assigns zero progress when the initial distance is at most 10^{-8}. This fallback is reported separately from success and from the gate in Equation 15. The evaluator also retains tail distance and target-occupancy diagnostics, allowing detection of policies that reach the target and subsequently drift away.

B.4 FINGERTURNEASY

FingerTurnEasy uses the FingerTurn task in the MJX environment of MuJoCo Playground, with a 12-dimensional observation, two actions, and 1,000-step episodes. The adapter retains the native physics and reward while exposing task metrics to the evaluator. Evaluation uses mean actions and frozen observation normalization. The residual input has 26 entries.

Here success is measured as a *time fraction*. Let p_{time} be the mean fraction of an episode spent satisfying the native target predicate, $\bar{d}_{\mathrm{abs}}$ the mean absolute target-distance metric, and $\bar{u}$ the mean control cost. The gate score is

$$F_{\mathrm{F}} = p_{\mathrm{time}} + 0.1\left\{\exp\left[-\left(\frac{\bar{d}_{\mathrm{abs}}}{0.25}\right)^2\right] + \exp\Big[-\frac{\bar{u}}{0.05}\Big]\right\}. \tag{17}$$

This additive score retains distance and control-quality information at low success fractions. Each quality term is bounded, and the total score lies in $[0, 1.2]$ for the configured nonnegative metrics, combining success occupancy with distance and control quality on one scale. We report native return and the score components alongside it. In a complete fixed-length episode, native return divided by episode length gives the success fraction used by this evaluator.

B.5 FROZEN REWARD SOURCES AND COMPOSITION

The handcrafted condition uses the task's native reward as its base. In the learned condition, RN denotes a frozen reward-network artifact produced by the baseline pipeline. In the programmatic condition, Eureka supplies a frozen executable reward. Their internal forms differ, and the residual is evaluated through a common scalar-reward interface. The base artifact includes the feature transformation and normalization needed to reproduce its output, and those transformations stay frozen during refinement.

The complete base expression matters. For example, a neural potential used together with native reward is a different base from the potential alone. The reward-network artifact, composition rule, and any shaping coefficient are therefore identified together. The label RN denotes the reward-network artifact of the baseline pipeline, and each baseline is attributed according to its producing implementation.

Freezing an artifact also freezes its selection: MIMUS keeps the LLM program and the RN parameters fixed while optimizing the residual. Policies trained on the base reward and on the refined reward use the same evaluator and matched training budget. If a continuation experiment uses an anchor obtained under a different reward source, we report that shared anchor and its origin, and the experiment measures refinement from that anchor.

B.6 Baseline Construction and Frozen Artifacts

We distinguish three budgets: constructing a base reward, training a policy under the frozen base, and refining that base with MIMUS. The original baseline search uses its own seeds and horizon, so the matched base reward control shares the refinement experiment's anchor, learner, selection horizon, and evaluator without the added residual.

Handcrafted rewards. These use native task rewards and skip the reward-search stage; policy training, anchor construction, and evaluation consume compute as usual. The native reward and the external scores specified above are separate quantities.

Learned and programmatic rewards. The RN and Eureka bases follow the construction procedures of Appendices C.5 and C.6, and each artifact is frozen before refinement. The frozen manifests record that test results were not used for base selection, which certifies the baseline pipeline's declared split, and the independence of a later MIMUS audit follows from its own seed records. Section C.6.1 reproduces the prompts behind the Eureka bases.

C Experimental Settings

C.1 Inner Policy Optimization

Each candidate is trained under N policy-training seeds. Seed-specific randomness controls policy initialization where applicable, rollout sampling, and minibatch ordering. Common seed banks make candidates comparable within a generation, and their learner states remain separate. A fixed bank reduces comparison variance, and policy training on fresh seeds assesses transfer to new randomness.

PPO settings are task-specific. The Gymnax implementation and the MJX implementation aligned with the official learner share the algorithm name while differing in normalization, rollout layout, and update semantics. The reward normalization and reward-scaling choices are especially consequential because they alter the effective residual seen by the learner. Within each comparison these choices are fixed across the base reward and its refined counterpart.

The PointMass configuration uses global observation statistics updated once per PPO update, Brax-style generalized advantage estimation, no value clipping, and no per-minibatch advantage normalization. Reward normalization is controlled by its own field, separate from the reward-clip value of 10, and Table 4 marks fields that an archived configuration omits as not serialized.

The inspected Gymnax policy implementation uses separate two-layer width-64 tanh actor and critic networks. Reacher-misc has a diagonal Gaussian policy with a learned state-independent log standard deviation; DeepSea-bsuite uses a categorical policy. These are the policy architectures; the reward networks are the width-16 residuals described above. The historical network definitions, including the specialized MJX continuation actor and critic, are released with the producing revision that identifies the official-aligned runs.

The archived Reacher-misc configuration uses 4,096 environments and 100 rollout steps per update, while DeepSea-bsuite uses 1,024 environments and 64 steps. Candidate budgets are taken from the executed update count, while the nominal maximum-training field controls the schedule. Reacher-

Table 4: Inner PPO parameters for handcrafted-reward refinement. FingerTurnEasy entries marked by the column dagger follow the current continuation implementation, whose archived outer manifest omits the PPO parameters; a historical-checkpoint verification accompanies the release.

Parameter	Reacher-misc	DeepSea-bsuite	PointMass	FingerTurnEasy†
Parallel environments	4,096	1,024	2,048	2,048
Rollout steps	100	64	30	30
Unrolls per update	1	1	16	16
Transitions per update	409,600	65,536	983,040	983,040
Minibatches / epochs	32 / 4	32 / 4	32 / 16	32 / 16
Learning rate	$3\,10^{-4}$	$3\,10^{-4}$	10^{-3}	10^{-3}
Learning-rate annealing	yes	yes	no	no
Discount γ	.99	.99	.995	.995
GAE parameter	.95	.95	.95	.95
Policy clipping	.2	.2	.3	.3
Value-loss coefficient	.5	.5	.5	.5
Entropy coefficient	0	.01	.01	.01
Gradient-norm limit	.5	.5	none	none
Observation clip	10	10	10	10
Reward normalization	yes	not serialized	no	no
Reward scaling	not serialized	not serialized	10	10

misc selects at $S = 200$; DeepSea-bsuite selects at $S = 10$ while recording candidate evaluations through update 25. The latter incurs the cost of 25 updates even though its ranking uses the earlier checkpoint. PointMass and FingerTurnEasy select at $S = 10$ from anchors at $A = 3$.

C.2 Outer Optimization and Residual Settings

The handcrafted Reacher-misc run uses $K = 64$ candidates and 50 generations. The handcrafted DeepSea-bsuite, PointMass, and FingerTurnEasy runs use $K = 32$, with 20, 50, and 50 generations, respectively. All four use $N = 4$ candidate-training seeds. The initial search scales recorded for these particular runs are 0.02, 0.025, 0.16, and 0.08, respectively. Each value is specific to the reported run and its network geometry.

Table 5: Main handcrafted-reward search configurations. G is the configured budget. All use rank-4, two width-16 hidden layers, and residual multiplier $\alpha = 1$.

Task	Input	Full / active d	K	N	G	σ_0
Reacher-misc	18	313 / 313	64	4	50	.02
DeepSea-bsuite	514	2,297 / 2,297	32	4	20	.025
PointMass	10	281 / 280	32	4	50	.16
FingerTurnEasy	26	345 / 344	32	4	50	.08

RN and Eureka refinement keeps the task-level populations, candidate seed counts, and generation budgets of Table 5; the Reacher-misc handcrafted and learned-base searches all configure 50 generations. Hidden-factor initialization is 0.5 for the main zero-output runs and 0.1 for the DeepSea-bsuite RN and Eureka refinements, and the per-arm records preserve that difference.

CMA-ES updates its covariance and step size in the active gene space, while the OpenAI-ES ablation replaces this update with mirrored Gaussian perturbations, rank-based utilities, and an Adam mean update at its recorded scale and learning rate. Optimizer comparisons account for evaluated policy-training seeds and environment transitions as well as generations, because a larger population changes both the data available to an update and the cost of a generation.

C.3 Seed Banks, Anchors, and Evaluation

Each task defines one seed bank that the whole study draws on, listed in Table 6. Every stage trains candidate policies with the task's candidate bank and scores them with the task's evaluation bank:

the base reward reference, the base + MIMUS search, the long-horizon audits, the ablations, and the RN/Eureka refinements all reuse these seeds, so the stages differ in their endpoints while the seeds stay fixed. Candidate evaluation uses 64 episodes per evaluation seed and center evaluation uses 128 on Reacher-misc and DeepSea-bsuite.

Table 6: Task-level seed banks. Every experiment on a task draws on the same banks, and — marks training from scratch.

Task	Anchor seed	Candidate training	Evaluation	Center reward	Outer seed
Reacher-misc	—	42–45	1001–1003	2001–2004	132
DeepSea-bsuite	—	42–45	1001–1003	2001–2005	132
PointMass	9	83–86	1001–1003	113–116	9001
FingerTurnEasy	1	101–104	41001–41003	101–104	9001

Within an evaluation, aggregation follows the declared rule, and for a nonlinear score F the two orders generally differ:

$$\frac{1}{N}\sum_{n=1}^{N} F(\mathbf{q}_n) \neq F\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{q}_n\right). \tag{18}$$

Conditional quantities such as success time are computed by pooling the contributing episodes, and per-seed diagnostics accompany the scalar that the optimizer update consumes.

C.4 Controlled Ablation Changes

The scratch-reward ablation removes the frozen base contribution and evolves the network reward alone. Its output-head initialization is recorded separately: a random nonzero head avoids an identically zero initial training reward, so this comparison intervenes on the reward objective together with its initialization. The random-head ablation instead retains the base reward while replacing zero-output initialization. It tests the starting point of residual search, not whether a base reward is available.

The optimizer ablation retains the reward representation and task protocol while replacing CMA-ES with OpenAI-ES. The representation ablation retains the chosen network width and compares low-rank factors with a dense matrix representation. Search scales can differ after function-space calibration. Where memory requires a separable covariance optimizer, both representation arms use that optimizer for the matched comparison. Partially completed runs are compared only over their common observed budget.

Table 7: Interventions in component ablations. Shared budgets refer to the corresponding task-level comparison, not to equal budgets across tasks.

Comparison	Changed quantities	Important qualification
Residual / scratch	Remove base reward; evolve network-only reward.	Reacher-misc, DeepSea-bsuite, and PointMass record scratch head scale .5, which changes the objective and its initialization together.
CMA / OpenAI-ES	Replace covariance adaptation with the recorded OpenAI-ES implementation.	Learning rate .01; task-specific noise scales below.
Zero / random head	Retain base; initialize output head randomly with scale .5.	The gene-spec zero-head label can precede a later head override.
Low-rank / dense	Change matrix parameterization at width 64; use separately recorded noise scales.	DeepSea-bsuite uses separable CMA in both arms, and active-bias conventions are recorded per arm.

The OpenAI-ES ablations use $(K, \sigma_0) = (64, .02)$ for Reacher-misc, $(32, .025)$ for DeepSea-bsuite, $(32, .16)$ for PointMass, and $(32, .08)$ for FingerTurnEasy. All four record learning rate .01. FingerTurnEasy explicitly records the `evosax_open_es` backend. Backend-specific utility normalization, Adam defaults, and sigma scheduling follow the producing implementation.

The original scratch comparison intervenes on the reward objective together with its initialization. For FingerTurnEasy, the compact scratch manifest records network-only reward and omits the head scale; that historical field stays open in the record.

C.5 RN CONFIGURATION

The archived RN producers identify mirrored evolution of a fully parameterized state-potential network. Each network has one hidden layer of width 128 and a scalar output, and its input is the current state observation, while the MIMUS residual uses transition features. Table 8 records the producing configurations; the worker count is a configuration field, and the total number of evaluated policies follows from how mirrored perturbations and training seeds are expanded.

Table 8: Archived RN construction settings, distinct from MIMUS residual settings.

Parameter	Reacher-misc	DeepSea-bsuite	PointMass	FingerTurnEasy
Input / hidden / output	8/128/1	256/128/1	4/128/1	12/128/1
Outer iterations	20	20	20	20
Configured workers	16	16	16	16
Perturbation std.	.1	.1	.1	.1
Step size	.5	.5	.5	.5
Candidate PPO updates	200	100	20	20
Recorded outer seed	132	132	9001	9001

The archived PointMass and FingerTurnEasy RN construction configurations use gradient-norm clipping at 0.5, unlike the unclipped continuation configuration reported for MIMUS. Reacher-misc RN construction records a PPO schedule ceiling of 81,920,000 transitions, while refinement records 204,800,000; the matched frozen-base versus refined-base comparison applies one learner configuration to both arms.

The scalar reward supplied to PPO follows from the architecture together with its wrapper. A direct network output, a potential difference $\gamma\Phi(s') - \Phi(s)$, and native reward plus either term are different constructions, so the RN artifact is stored together with the producing wrapper, terminal treatment, and scaling. Some compact Gymnax refinement configurations omit the frozen RN artifact identity, and their generic native-mode field leaves that identity open.

C.6 EUREKA CONFIGURATION

The four archived searches use the official DeepSeek endpoint with model identifier `DeepSeek-V4.1-Flash`, thinking enabled, reasoning effort `high`, and an unset temperature. Table 9 records the search configuration and Table 10 the training and evaluation budgets. Each run requests five iterations of 16 samples, i.e. 80 candidates, and validation and training outcomes are recorded per candidate. Policy training under a Eureka reward uses the inner learner of Table 4 with these budgets.

Table 9: Eureka search configuration. The single-backend-fingerprint requirement is enabled for Reacher-misc, DeepSea-bsuite, and FingerTurnEasy, and disabled for PointMass.

Setting	Value
Provider / model	DeepSeek official / `DeepSeek-V4.1-Flash`
Thinking mode / reasoning effort	enabled / `high`
Temperature	unset (serialized as null)
Iterations $\times$ samples	5×16 (80 requested candidates)
Sampling strategy	one independent request per candidate
Parallel workers / retries	16 / 2

Table 10: Eureka construction budgets and seeds. Episode counts are per evaluation seed. These differ from the subsequent MIMUS search settings.

Setting	Reacher-misc	DeepSea-bsuite	PointMass	FingerTurnEasy
Training updates	200	3	10	10
Selection train seeds	42–45	42–45	1–4	1–4
Selection eval seeds	1001–1003	1001	1001–1003	1001–1003
Selection episodes	64	128	64	64
Post-freeze train seeds	2001–2010	2001–2010	4001–4010	4001–4010
Post-freeze eval seeds	3001–3003	3001–3003	5001–5003	5001–5003
Post-freeze episodes	128	128	128	128

C.6.1 EUREKA PROMPTS

The Eureka baselines were generated with the prompts listed below. The shared system prompt is printed once and was paired with each task prompt. It shares its page with the preceding table, and each task prompt that follows keeps a page of its own.

Shared System Prompt

```
You are a reward engineer writing reward functions for reinforcement-
    learning tasks.
Your objective is to produce a training reward that helps a fixed PPO
    implementation learn the described task as effectively as possible.
The policy is evaluated with independent task metrics, not by the
    magnitude of your generated reward.
Return exactly one Python code block containing one function with this
     signature:

def compute_reward(observation, action, next_observation, done,
    task_features):
    ...
    return total_reward, component_dictionary

Requirements:
- Use only vectorized jax.numpy operations through the pre-defined
    name `jnp`; do not import anything.
- Allowed calls are: abs, clip, cos, exp, expm1, linalg.norm, log,
    log1p, maximum, mean, minimum, ones_like, power, sin, sqrt, square,
     sum, tanh, where, and zeros_like.
- Do not use Python loops, conditionals, comprehensions, helper
    functions, classes, file access, network access, or randomness.
- total_reward must be a finite float array with shape (num_envs,).
- component_dictionary must be non-empty and map short descriptive
    names to arrays of the same shape.
- Use only the five function arguments and the documented
    task_features keys.
- The native environment reward is unavailable and must not be
    reconstructed from hidden information.
- Use `jnp.where` for conditional numerical expressions.
```

Reacher-misc Task Prompt

```
Write a reward function for this task:
Control a two-link planar arm so that its fingertip reaches a randomly
     sampled target. The policy should reach the target reliably while
    avoiding unnecessarily large actions.

The preregistered primary selection metric is a task-aligned gated
    score computed by an independent deterministic evaluator from fully
     trained policies. Policies are ranked first by success rate,
    defined as the fraction of episodes whose fingertip comes within
    the 0.2 goal radius. Among successful policies, a smaller mean
    closest-approach distance is preferred; mean action energy is a
    mild secondary quality signal. Native episode return and the
    component statistics are secondary diagnostics and never affect
    candidate ranking.

Environment interface:
The observation has shape (num_envs, 8):
- observation[..., 0:2]: cosine of the two joint angles
- observation[..., 2:4]: sine of the two joint angles
- observation[..., 4:6]: two joint angular velocities
- observation[..., 6:8]: target x-y position
The action has shape (num_envs, 2), is clipped to [-1, 1], and
    represents joint torques.
next_observation has the same layout after the action is applied.
task_features contains only these vectorized arrays:
- goal_distance: Euclidean fingertip-to-target distance after the
    transition
- action_squared: sum of squared clipped actions
- success: 1 when goal_distance <= 0.2, otherwise 0
- position: negative goal_distance
- terminated: physical termination flag (always zero for this task)
- truncated: episode-horizon flag
The environment's native reward is intentionally unavailable to the
    generated function.

Return only the requested Python code block.
```

DeepSea-bsuite Task Prompt

```
Write a reward function for this task:
Navigate a deterministic deep-exploration grid. At every row the
    policy chooses action 0 or 1. Action 1 moves one column to the
    right and action 0 moves one column to the left, with boundary
    clipping. The episode lasts exactly 16 transitions. The task is
    solved only by reaching the far-right column at the final row.

The preregistered primary selection metric is full-path success rate,
    computed by an independent evaluator.
If every candidate has zero success, normalized maximum rightward
    progress is used only as a bounded tie-break.
Native episode return and action statistics are secondary diagnostics.

Environment interface:
The observation has shape (num_envs, 256).
It is the flattened 16 by 16 one-hot grid position in row-major order.
     Exactly one element is one before a terminal transition.
The action has shape (num_envs,), contains integer indices, and is
    clipped to {0, 1}.
next_observation has the same flattened layout after the action; the
    terminal observation may be all zeros.
task_features contains only these vectorized arrays:
- position: normalized post-transition column in [0, 1], where 1 is
    the far-right column
- action_squared: square of the executed action index
- terminated: episode-end flag after the sixteenth transition
- truncated: truncation flag, always zero
Neither the native reward nor the independent success metric is
    available to the generated function.

Return only the requested Python code block.
```

PointMass Task Prompt

```
Write a reward function for this task:
Control a two-dimensional point mass so that it reaches and remains
    near a fixed target at the origin. Actions are bounded two-
    dimensional motor controls. Episodes last 1000 physics transitions
    unless a non-finite state terminates them.

The preregistered primary selection metric is a task-aligned gated
    score computed by an independent deterministic evaluator from fully
     trained policies. Policies are ranked first by stable success rate
    : the point mass must reach the target region and remain there for
    a sustained hold. Among successful policies, a shorter time to the
    first sustained arrival is preferred, and lower action energy is a
    mild secondary quality signal. Native episode return, episode
    length, and near-target statistics are secondary diagnostics and
    never affect candidate ranking.

Environment interface:
The observation has shape (num_envs, 4) and contains [x_position,
    y_position, x_velocity, y_velocity].
The action has shape (num_envs, 2) and is clipped to [-1, 1].
next_observation has the same layout after the action.
task_features contains only these vectorized arrays:
- distance_to_target: post-transition Euclidean distance from the
    origin
- distance_progress: pre-transition distance minus post-transition
    distance
- speed: post-transition velocity norm
- radial_velocity: signed outward radial velocity; negative values
    move toward the target
- action_squared: sum of squared executed controls
- terminated: non-finite-state termination flag
- truncated: 1000-step time-limit flag
Neither the native reward nor its near-target/control components are
    available to the generated function.

Return only the requested Python code block.
```

FingerTurnEasy Task Prompt

```
Write a reward function for this task:
Control a two-link planar finger so that its fingertip enters and
    remains inside a small spherical target attached to a spinner.
    Actions are bounded two-dimensional joint controls. Episodes last
    1000 physics transitions.

The preregistered primary selection metric is a task-aligned gated
    score computed by an independent deterministic evaluator from fully
     trained policies. Policies are ranked first by the fraction of
    episode steps spent with the fingertip inside the target sphere.
    Policies with a higher inside-target fraction are preferred; among
    policies close in that primary signal, a smaller mean absolute
    target distance is preferred, and lower control effort is a mild
    secondary quality signal. Native episode return and the component
    statistics are secondary diagnostics and never affect candidate
    ranking.

Environment interface:
The observation has shape (num_envs, 12):
- bounded positions of the proximal body, distal body, and fingertip
- two hinge/root velocity coordinates
- two logarithmically scaled touch-sensor readings
- the target position relative to the spinner
- the signed fingertip-to-target-surface distance (negative means
    inside)
The action has shape (num_envs, 2), is clipped to [-1, 1], and
    represents joint controls.
next_observation has the same layout after the action.
task_features contains only these vectorized arrays:
- distance_to_target: signed post-transition fingertip-to-target-
    surface distance; negative means inside
- inside_target: 1.0 when the fingertip is inside the target sphere,
    otherwise 0.0
- action_squared: sum of squared executed controls
- touch_sum: sum of the two log-scaled touch sensors
- terminated: non-finite-state termination flag
- truncated: 1000-step time-limit flag
The native reward and the independent gate metric are not directly
    available to the generated function.


Return only the requested Python code block.
```

D Signal Validation and Search-Scale Calibration

D.1 Fixed-Population Signal Diagnostics

Signal Gate evaluates a fixed set of perturbed genes without updating the outer optimizer, and its outcome fixes the teaching fitness, the initial scale σ_0, the population size K, the seed count N, and the training budget before the search starts. Write Y_{kn} for the measured fitness of candidate k under training seed n. A diagnostic population includes mirrored pairs $\mathbf{z}_0 + \sigma\boldsymbol{\epsilon}_j$ and $\mathbf{z}_0 - \sigma\boldsymbol{\epsilon}_j$. Candidate identities and seed assignments remain fixed throughout this diagnostic, allowing reward-induced differences to be separated from a common easy or difficult training seed.

For a split of the training seeds into two groups, we compute candidate mean scores within each group and measure their rank agreement. Mirrored-pair differences $D_{jn} = Y_{j+,n} - Y_{j-,n}$ additionally test whether a perturbation direction has a consistent effect. In the PointMass diagnostic, the

directional-consistency statistic is

$$a_{\text{dir}} = \frac{1}{J}\sum_{j=1}^{J}\left|\frac{1}{N}\sum_{n=1}^{N}\text{sign}(D_{jn})\right|, \tag{19}$$

with zero differences contributing zero. The value is high when each pair favors a consistent sign across training seeds, and the subsequent search measures whether that direction stays beneficial across outer updates.

We also decompose the score matrix descriptively:

$$Y_{kn} = \bar{Y} + a_k + b_n + e_{kn}, \quad a_k = \bar{Y}_{k\cdot} - \bar{Y}, \quad b_n = \bar{Y}_{\cdot n} - \bar{Y}. \tag{20}$$

The ratio of candidate-effect RMS to residual-interaction RMS measures whether candidate differences survive removal of a shared seed effect. This decomposition serves as a descriptive diagnostic of the score matrix. Evaluation randomness and other unmodeled effects also contribute to e_{kn}. More training seeds sharpen the estimate of candidate quality, while more candidates increase the number of tested directions.

Results. Table 11 reports the three admission statistics for the four native-reward bases together with the teaching fitness each diagnostic scores. All four tasks pass: the split-half rank agreement, the mirrored-direction agreement, and the candidate-to-interaction RMS ratio exceed their thresholds, so at the selection endpoint the fixed-population comparison resolves candidate differences above PPO-seed noise. PointMass additionally has an independently drawn confirmation run that reproduces the discovery statistics.

Table 11: Fixed-population Signal Gate on the native-reward bases. Thresholds are pre-registered (split-half $\rho \geq 0.60$, direction ≥ 0.625, candidate/interaction ≥ 1.0). The diagnostic population is $K = 8$ except PointMass with $K = 32$, and σ_0 is the perturbation scale the gate selects.

Task	Teaching fitness	σ_0	Split-half ρ	Direction	Cand./inter.
Reacher-misc	gate (success/distance/energy)	0.1	0.976	0.750	6.11
DeepSea-bsuite	gate (success rate)	0.025	0.905	0.938	1.16
PointMass (discovery)	native return (earlier protocol)	0.16	0.883	0.688	2.31
PointMass (confirmation)	native return (earlier protocol)	0.16	0.973	0.906	5.08
FingerTurnEasy	gate (occupancy/distance/effort)	0.08	0.639	0.750	1.31

The gate also selects σ_0, which matters most on Reacher-misc: at $\sigma \in \{0.0125, 0.02, 0.05\}$ at least one criterion fails (split-half -0.48, 0.10 and 0.48; direction 0.38, 0.38 and 0.63; candidate-to-interaction 0.42, 0.60 and 3.25), whereas $\sigma_0 = 0.1$ passes all three, so the table reports the smallest passing value. The reported Reacher search was run at $\sigma_0 = 0.02$, below the gate-selected value: this task needs a larger perturbation before the fixed-population comparison becomes reproducible.

D.2 Calibrating Perturbations in Reward Space

Equal coordinate-space σ values produce different reward perturbations across representations. For a low-rank layer, perturbing both factors gives

$$(\mathbf{A} + \sigma\mathbf{U})(\mathbf{B} + \sigma\mathbf{V}) - \mathbf{AB} = \sigma(\mathbf{UB} + \mathbf{AV}) + \sigma^2\mathbf{UV}. \tag{21}$$

The effect depends on factor initialization, rank, and width, and is further transformed by hidden and output nonlinearities. Dense coordinate perturbations follow a different geometry. We therefore characterize the induced residual on common replay transitions when comparing calibrated representations.

Let $r_{ki} = \Delta R_{\theta(\mathbf{z}_k)}(x_i)$ for replay transition x_i, and $\bar{r}_k$ its mean over M replay transitions. The state-dependent residual scale used in the PointMass calibration is

$$s_k = \left[\frac{1}{M}\sum_{i=1}^{M}(r_{ki} - \bar{r}_k)^2\right]^{1/2}, \qquad \eta = \frac{\alpha K^{-1}\sum_k s_k}{[M^{-1}\sum_i R_{\text{base}}(x_i)^2]^{1/2}}. \tag{22}$$

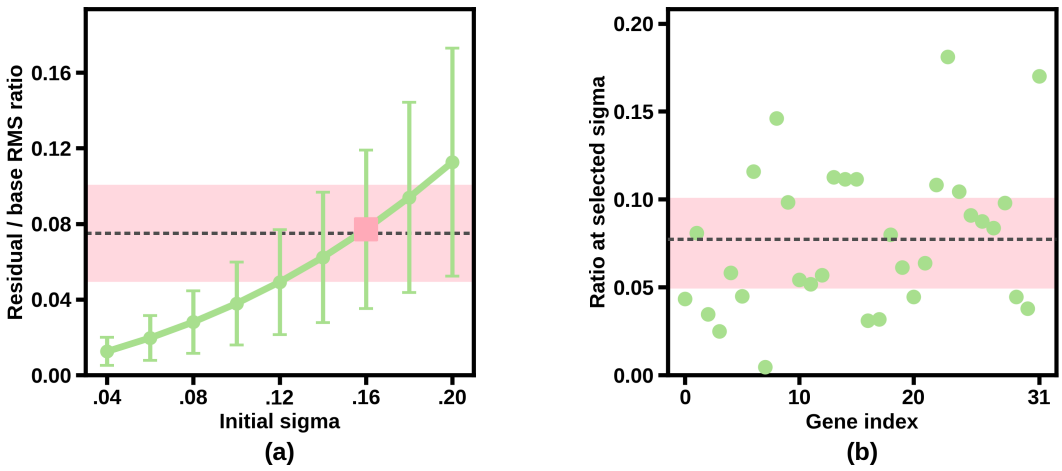

Figure 5: **Initial reward-space scale calibration for PointMass. a**, State-dependent residual RMS divided by base reward RMS, measured on 16,384 fixed replay transitions for all nine tested initial scales. Points and error bars show the mean and sample standard deviation across the 32 candidate genes. The shaded band is the admissible *mean* interval $[0.05, 0.10]$, and the dotted line is the target 0.075. The selected $\sigma_0 = 0.16$ (pink square) gives mean ratio 0.0771. **b**, All 32 individual ratios at that scale; the dashed line is their mean, and individual candidates spread around the admitted population mean. The archived saturation fraction is zero at every tested scale, and scale selection is fixed from replay measurements before any policy fitness is observed.

Subtracting the transition mean prevents a nearly constant residual from dominating this scale diagnostic, while the reward supplied to PPO keeps its original form. The implementation also records uncentered RMS, the fraction of energy attributable to the constant component, mirrored-pair differences, and output saturation. If the base RMS is zero, this normalization is undefined and a different declared reference is required.

The PointMass calibration admits a mean scale ratio in $[0.05, 0.10]$, targets 0.075, and limits the fraction of outputs with absolute value at least 0.95 to 0.01. Scale selection uses these replay measurements, so the sigma is fixed before any policy fitness is observed. Replay-scale matching equalizes the initial perturbation magnitude across representations, and each search then evolves its own scale.

Figure 5 makes the scale-selection criterion concrete. The individual-gene spread remains substantial at the selected scale, despite an admissible mean. The diagnostic fixes the perturbation magnitude that the calibrated runs start from.

D.3 Confirmation and Interpretation

Protocol exploration uses discovery measurements, followed by confirmation with a separate perturbation and seed configuration where that stage is available. PointMass is the only task with such a confirmation run; it reproduces the discovery statistics and raises the candidate-to-interaction ratio from 2.31 to 5.08. Its diagnostic scores are native episode returns, and the search itself ranks candidates with the time/effort gate, so Table 11 characterises the native-return protocol for that task. The other three tasks contribute one diagnostic each, scored with the fitness that ranks their candidates.

The PointMass RN-refinement manifest explicitly disables signal diagnostics and preflight under its unrestricted-control admission policy, so this control is recorded as an identified exception: the handcrafted-base protocol passed Signal Gate, and the RN arm ran without the preflight.

Passing Signal Gate establishes the feasibility of local comparisons, and the subsequent reward audits establish the outcome of the search. Candidate-by-seed interaction is measured after the reward changes, independently of the spread across the base reward runs. The unsuccessful Cartpole and WalkerWalk explorations delimit the tested protocols and budgets for these tasks.

E Additional Results and Analyses

E.1 Reported Objects and Their Sources

Four estimands appear in the paper, and each answers a different question. Table 12 defines each one, states how it is aggregated, and identifies the artifact it comes from. Population statistics describe search behaviour, a center audit evaluates one particular reward, and the base reference is the matched control that shares the refinement protocol without the added residual.

Table 12: Reported objects. K is the population size and N the number of policy-training seeds per candidate.

Object	Definition	Aggregation	Reported in
Population mean	Candidate fitness averaged over the K candidates of a generation	Mean over the N policy-training seeds	Blue curves of Figures 2–4
Best candidate	Largest measured candidate fitness within a generation	None; an optimistic maximum	Green curves of Figures 2–4
Center audit	The CMA-ES mean gene, retrained from the anchor and scored on independent seeds	Mean over the center-training seeds	Table 1, markers in Figure 2
Base reference	The frozen base reward trained under the matched protocol	Mean over the matched seed bank	Dashed lines of Figures 2–4

E.2 Search Trajectories

The curves of Figures 2–4 plot, for each generation g, the population mean $\bar{Y}_g$ and the best-candidate value $\max_k Y_{gk}$; the shaded band is the standard deviation of the N per-seed population means, with divisor N matching the plotting implementation. That band quantifies how much the population average moves when the policy-training seeds change, while independent outer repetitions and confidence intervals for the selected reward are reported separately. Table 13 summarises the same curves: the value of the population mean at generation 0, the final population mean with its seed spread, the largest best-candidate value reached, the budget-normalised AUC of Equation 23 computed on the mean curve over the displayed interval, and the gap between the final mean and the base. DeepSea-bsuite reaches the largest gain because its base reward is sparse, PointMass and FingerTurnEasy start from bases that already score 0.234 and 0.412, and on every task the AUC of the mean curve exceeds the base level from the first generations onward.

Table 13: Search-trajectory summary for the handcrafted-reward runs of Figure 2. Generations are the displayed interval; the final standard deviation is across the N policy-training seeds; AUC is the budget-normalised trapezoidal integral of the population-mean curve over that interval.

Task	Gens	Gen 0	Final mean	Final SD	Peak best	AUC	Base
Reacher-misc	0–49	0.180	0.257	0.021	0.303	0.245	0.180
DeepSea-bsuite	0–19	0.104	0.923	0.023	1.000	0.850	0.000
PointMass	0–49	0.157	0.319	0.006	0.344	0.301	0.234
FingerTurnEasy	0–49	0.369	0.499	0.002	0.543	0.451	0.412

E.3 Baseline Construction Trajectories

The frozen RN and Eureka bases are themselves the output of a search, so their quality is part of the comparison. Figure 6 shows those searches directly. The archived RN producers evolve a state-potential network for 20 outer iterations with 16 workers and report the population mean, the population best, and the center evaluated at each iteration; the artifact that the producing run selected and froze scores 0.750 on DeepSea-bsuite while its best sampled candidate reaches 1.000. The four Eureka searches run five iterations of 16 generated candidates, each trained and scored with the protocol of Appendix C.6; the figure plots every completed candidate, their per-iteration mean and best, and the frozen reward selected from the final population. On PointMass and FingerTurnEasy

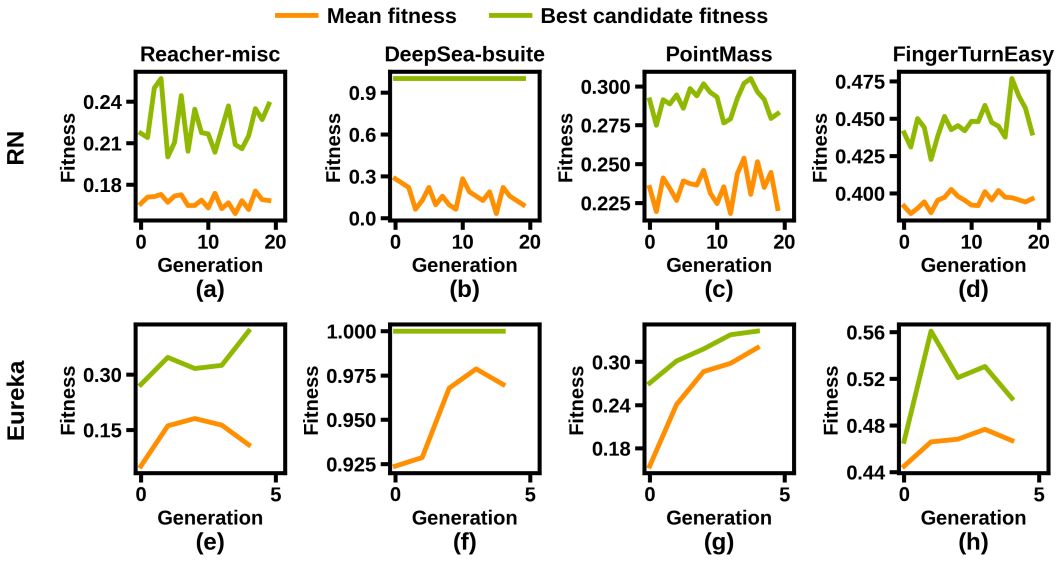

Figure 6: **Construction trajectories of the frozen RN and Eureka bases.** Rows are the two baseline families and columns the four tasks; every panel plots fitness against generation. Orange and green curves give the population mean and the best candidate of each iteration. The upper row covers the 20 outer iterations of the archived RN producers and the lower row the five iterations of the archived Eureka searches; panel letters (a)–(h) run left to right within each row.

the selected Eureka reward is the best candidate of the final iteration, whereas on Reacher-misc the frozen reward ties the search maximum of 0.418 reached in the same iteration. Both searches therefore improve substantially over their own first iterations, and the frozen bases refined in Section 5.4 are the artifacts that these multi-iteration searches selected.

E.4 Component and Representation Ablations

Four interventions share the task protocol of Figure 3: training under a network-only reward instead of the base plus residual, replacing covariance adaptation with OpenAI-ES, replacing the zero-output head with a random one, and replacing rank-4 factors with dense matrices at width 64. Each comparison keeps the population size, the policy-training seeds, the selection endpoint and the evaluator of its task, so every panel of Figure 3 isolates one intervention.

Area-under-curve comparisons use the same horizontal interval and definition within a task. For values f_j at observed budgets b_j, the budget-normalized trapezoidal summary is

$$\mathrm{AUC}_{\mathrm{norm}} = \frac{1}{b_m - b_0} \sum_{j=0}^{m-1} \frac{f_j + f_{j+1}}{2} (b_{j+1} - b_j). \tag{23}$$

The budget is measured in generations where population size and per-candidate training costs match, and otherwise in environment transitions or policy-training seeds. Missing generations are excluded, and each truncated run contributes the interval it actually covers.

Frozen base prior. Removing the prior lowers AUC on every task, and on Reacher-misc it collapses the curve (0.245 versus 0.004) with a final mean of 0.012 against 0.257. PointMass and FingerTurnEasy also end lower (0.319 versus 0.303 and 0.499 versus 0.403). DeepSea-bsuite is the exception in the endpoint column only: with a near-saturated base the network-only reward reaches 0.992, but its average progress is far lower (0.850 versus 0.423).

Outer optimizer. CMA-ES gives the higher AUC on Reacher-misc, PointMass and FingerTurnEasy, and the two optimizers rank consistently at the endpoint. DeepSea-bsuite is the opposite

Table 14: Search-curve summaries for the four comparisons of Figure 3. AUC is trapezoidal area divided by the observed generation interval (0–49 for Reacher-misc, PointMass and FingerTurnEasy; 0–19 for DeepSea-bsuite), and End is the final population mean within that interval. Each pair lists the reference configuration first (residual, CMA-ES, zero head, rank-4), then its alternative.

Task	Comparison	AUC ref.	AUC alt.	End ref.	End alt.
Reacher-misc	Residual / scratch	.245	.004	.257	.012
	CMA / OpenAI-ES	.245	.230	.257	.245
	Zero / random head	.245	.233	.257	.266
	Rank four / dense	.236	.222	.253	.227
DeepSea-bsuite	Residual / scratch	.850	.423	.923	.992
	CMA / OpenAI-ES	.850	.907	.923	1.000
	Zero / random head	.850	.661	.923	.969
	Rank four / dense	.924	.824	.938	.933
PointMass	Residual / scratch	.301	.278	.319	.303
	CMA / OpenAI-ES	.301	.238	.319	.239
	Zero / random head	.301	.278	.319	.324
	Rank four / dense	.259	.236	.224	.236
FingerTurnEasy	Residual / scratch	.451	.382	.499	.403
	CMA / OpenAI-ES	.451	.406	.499	.435
	Zero / random head	.451	.456	.499	.514
	Rank four / dense	.459	.480	.498	.479

case: its reward gene is an order of magnitude larger than the other tasks, and there OpenAI-ES reaches a higher AUC (0.907 versus 0.850) and endpoint (1.000 versus 0.923) with a small margin. Covariance adaptation therefore pays off on the smaller gene spaces, while a fixed Gaussian with rank-based utilities remains competitive when the search dimension is large.

Zero-output initialisation. Both heads end close together, with the random head slightly higher on all four tasks, but the zero head gives the higher AUC on Reacher-misc (0.245 versus 0.233), DeepSea-bsuite (0.850 versus 0.661) and PointMass (0.301 versus 0.278): starting exactly at the base reward reaches useful fitness sooner. FingerTurnEasy is the one task where the random head is ahead on both statistics (0.456 versus 0.451 AUC). The zero head's contribution is the exact identity at the centre: the search starts from the base reward itself, which is what turns into the faster early progress behind those AUC values.

Low-rank representation. At width 64 the rank-4 factors raise both AUC and endpoint on Reacher-misc and DeepSea-bsuite, raise AUC on PointMass, and raise the endpoint on FingerTurnEasy, so the low-rank gene improves on the dense network in at least one statistic on every task while using far fewer coordinates. The width-64 DeepSea-bsuite residual has 3,017 low-rank parameters against 37,185 dense parameters; a full covariance matrix over the dense coordinates would need approximately 5.15 GiB for one float32 matrix before optimizer intermediates, policies and simulator state, and the archived full-covariance dense attempt exhausted device memory. The matched width-64 comparison therefore uses separable CMA-ES for both representations, with an $O(d)$ diagonal covariance state, and its results are reported under that optimizer variant.

Table 15: Width-64 representation experiments. LR denotes rank-4 factorization; dimensions are serialized gene counts and the covariance column is the theoretical storage, not measured peak memory.

Task	LR d	Dense d	LR σ_0	Dense σ_0	Covariance
Reacher-misc	1,033	5,441	.021	.0342	full
DeepSea-bsuite	3,017	37,185	.0078	.0191	diagonal
PointMass	1,001	4,929	.08	.08	full
FingerTurnEasy	1,065	5,953	.04	.02	full

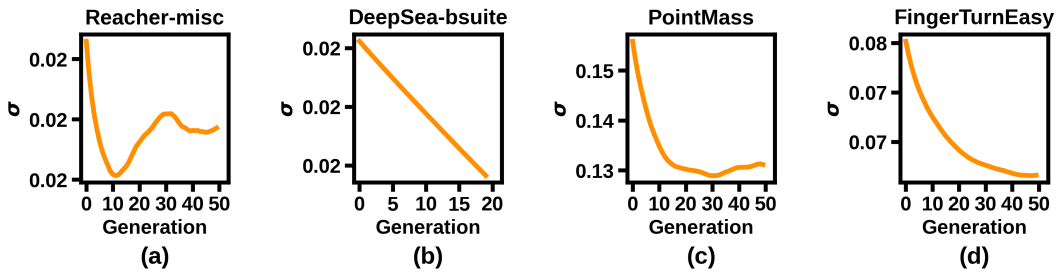

Figure 7: **CMA-ES step size over the handcrafted-reward searches.** Columns are the four tasks and the vertical axis is the step size σ, labelled in decimals on every panel; panels (a)–(d) run left to right. Each curve starts at the task's calibrated σ_0 (Table 5) and shows the adaptation performed by the outer optimizer.

Initial scales differ between the two arms because the two representations are matched in reward space (Appendix D). Table 15 reports serialized gene counts, and the active search dimension is one coordinate smaller whenever the output bias is frozen, as it is for PointMass and FingerTurnEasy. Runtime also includes policy training, and the mean, scale and eigenvectors of the search distribution carry information beyond the condition number, so we read these diagnostics together with the audited policy outcomes of Table 14.

E.5 Inner-Loop Training Diagnostics

Refinement changes the reward that PPO sees, so we also check that the refined reward trains as stably as the base. We use the FingerTurnEasy audit runs as the worked example, because their records serialize the inner-loop metrics; Table 16 reports the base reward and center reward arms at the short and long budgets. The two arms share the anchor, the evaluator, the seed bank and the budget, so only the training reward differs, and the same protocol applies to the other tasks' audits. In this example the center reward trains with a lower approximate KL and a lower clip fraction than the base at both budgets, and its policies end at markedly lower entropy: the refined reward concentrates the policy while keeping the updates stable. Reward normalisation is disabled in this configuration, which is why the recorded reward scale stays at one for both arms.

Table 16: PPO diagnostics of the FingerTurnEasy audit runs, the only task whose audit records serialize inner-loop metrics. Values are the final training metrics at $S = 10$ and $H = 43$ over the four policy-training seeds; Approx. KL and clip fraction are averages over the final update.

Arm	Budget	Episodic return	Approx. KL	Clip fraction	Entropy
Base reward	$S = 10$	298.7	0.037	0.104	4.00
Center reward	$S = 10$	326.5	0.017	0.083	1.94
Base reward	$H = 43$	425.0	0.015	0.055	8.62
Center reward	$H = 43$	410.7	0.017	0.058	2.69

E.6 Outer-Loop Optimizer Diagnostics

Figure 7 records the CMA-ES step size of the four handcrafted-reward searches. Every task starts from its calibrated σ_0 and decays it gradually as the covariance adapts, and the curves stay inside a narrow band without the step-size collapses or blow-ups that would indicate a failing search. The archived covariance is separable in the width-64 comparisons, whose condition number stays at approximately one throughout, so the step size is the informative outer-loop diagnostic; the reward-gene trajectories themselves are reported through the population curves of Figure 2.

F LIMITATIONS

MIMUS requires a measurable relationship between reward perturbations and downstream policy outcomes at an affordable evaluation budget. When candidate-by-seed interactions dominate that relationship, reliability comes from seed replication. Independent outer repetitions and fresh policy-training seeds cover the two aspects of reliability.

A task where the search signal is absent. WalkerWalk-MJX is an excluded locomotion task that shows the opposite regime. Table 17 reports its generation-0 fixed-population diagnostic: eight mirrored candidate genes, each trained with the task's policy-training seed bank under identical configurations and the same reward, and scored by the stable-walk gate. Four of the five criteria pass, and the mirrored-direction agreement is the single criterion that fails, so the run commits no outer update and stops with the decision `stop_outer_generation_signal_failed`. On the raw fitness scale the candidate main effect is 0.042 against a candidate-by-seed interaction of 0.101: different training seeds move the final fitness by more than the reward perturbation does, the external score stops pointing in a direction, and the run returns the frozen base reward as specified by Algorithm 1.

Table 17: Failed Signal Gate on WalkerWalk-MJX, a locomotion task excluded from the main experiments. The diagnostic fixes $K = 8$ mirrored candidate genes and $N = 8$ policy-training seeds, and scores every candidate with the stable-walk gate under the thresholds of Table 11.

Diagnostic	Value	Threshold	Result
Gate criteria			
Split-half rank agreement ρ	0.976	≥ 0.60	pass
Mirrored-direction agreement	0.438	≥ 0.625	**fail**
Candidate/interaction RMS (rank space)	1.03	≥ 1.0	pass
Raw fitness-scale decomposition			
Candidate main effect (RMS)	0.042	—	—
Candidate×seed interaction (RMS)	0.101	—	—
Main-effect/interaction ratio	0.42	—	—
Search outcome			
Committed outer updates	0	—	—
Decision	`stop_outer_generation_signal_failed`		

Lowering the inner-loop variance is the practical remedy: more policy-training seeds, a longer selection horizon, or a lower-variance RL algorithm. The MuJoCo Playground report, for example, documents that SAC attains markedly lower training variance than PPO on the same control tasks (Zakka et al., 2025). The task-level conclusion is therefore conditional on the learner: reward refinement needs an inner loop whose seed-to-seed variation is small relative to the effect of the reward itself.

Refinement is conditional on the base, anchor, learner, and budget. A reward that accelerates learning from one checkpoint may not improve training from scratch or remain beneficial after much longer training. The FingerTurnEasy horizon reversal is an observed instance of this budget dependence. Near-saturated bases leave little headroom under the chosen evaluator, so the procedure defaults to keeping the base reward.

Low-rank residuals restrict expressivity and introduce a factor-dependent search geometry. Their parameter savings can make covariance-based optimization more feasible, while dense residuals can match or exceed their fitness. The evidence supports measuring this tradeoff for the chosen task and architecture.

Finally, the external evaluator remains a specified proxy for task intent. Optimizing it can expose omissions in success definitions, effort penalties, or episode boundaries. Reporting native return and the fitness components keeps reward hacking and safety analysis in view. The current four-task evidence concerns simulated control; transfer to different dynamics, anchors, or hardware requires separate evaluation.