INTEREVOLVE: TEST-TIME EVOLUTION OF REWARD PROGRAMS FOR HUMANOID LOCO-MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We study test-time evolution for humanoid loco-manipulation: solving tasks that a controller was never trained for by repurposing its existing skills, improving from its own attempts, and retaining what it learns, without retraining. Our key insight is that *a broad controller already holds much of the competence a new task needs*, and that this competence becomes accessible through an interface between planning and control that is expressive enough to specify contact-rich, multi-stage interactions, yet executable and measurable enough that execution feedback can guide planning from experience. InterEvolve realizes this interface with two components. First, we develop an object-aware forward-backward (FB) behavioral foundation model, whose object residuals on a frozen body prior turn a new reward about the body or objects into loco-manipulation behavior at test time. Second, we specify tasks as *reward programs*: staged rewards with completion conditions and tunable constants. A large language model (LLM) agent revises the program structure in context, drawing on execution feedback and a skill library of verified programs, while a numerical optimizer tunes its constants. With every candidate verified across parallel simulation scenarios, the program explores new ways to induce, repurpose, and compose the controller's existing motor competence for the task at hand, and thus improves over iterations. Experiments show that human-designed rewards leave much of the FB model's loco-manipulation competence untapped, whereas the programs InterEvolve evolves release it, sometimes through novel strategies. It further produces behaviors for diverse tasks, complex scenes, and long-horizon compositions in simulation, and evolved skills run autonomously on a physical Unitree G1 from egocentric onboard perception.

1 INTRODUCTION

Learned humanoid controllers now provide broad whole-body motion for real-robot control (Luo et al., 2026), as well as contact-rich loco-manipulation (He et al., 2024), trained from human data. Yet a humanoid in an open-ended environment will meet tasks and scenes that were not enumerated during training. Consider a controller that has learned to push, lift, and carry boxes from human demonstrations. Asked to tip a box onto another face, it has no demonstration of this behavior, although the motions it needs lie within what it can already do. We study *test-time evolution*, in which a humanoid finds such uses of its existing skills, improves from its own attempts within a budget of trials, and keeps what it learns, so that a later composite task, such as carrying a box, placing it, and then kicking it, can build on earlier solutions. What evolves is the task specification, meaning which objectives, stages, and weights to use, together with the experience retained across tasks, while the controller stays fixed.

Achieving this capability requires a way to state a new task that is expressive enough for contact-rich, multi-stage interaction, yet cheap to revise after each attempt. Common task interfaces for humanoid control, such as motion references (Yang et al., 2025), goal states (He et al., 2026), and skill labels (Wang et al., 2026b), request desired motions, target states, or predefined behaviors, so a new objective such as tipping an object must first be turned into a reference or a new skill. Rewards can state such objectives directly, as their role in shaping contact-rich behavior during training shows (Xu et al., 2025), and they also measure which part of an attempt fell short. Realizing a new reward, however, conventionally requires training a policy for it, so every revision costs a reinforcement-learning run and iterating over many candidates becomes slow (Ma et al., 2024a).

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.

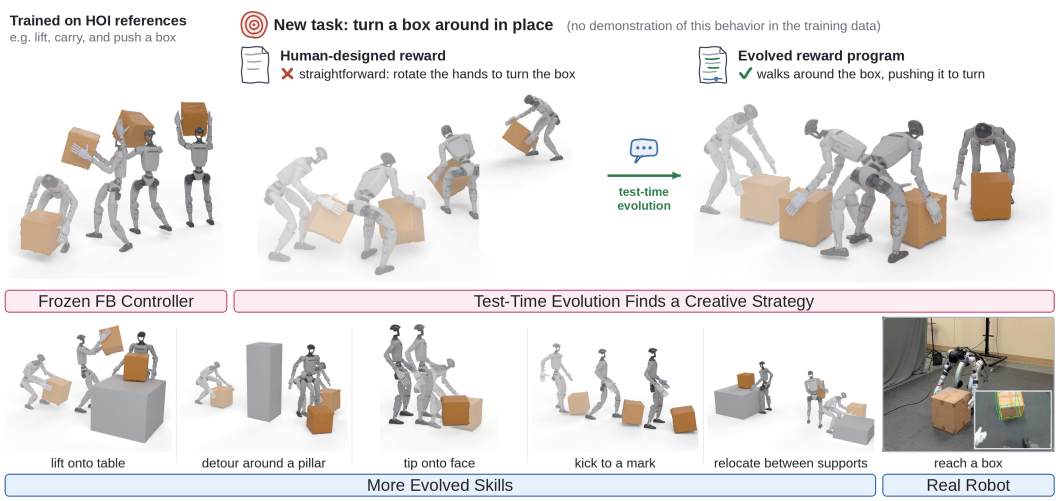


Figure 1: **InterEvolve adapts a frozen whole-body controller with novel use.** The controller trained on human-object interaction (HOI), such as lifting and pushing a box, is asked to turn a box around in place, a behavior absent from its training data (Sec. D.7). A human-designed reward tries to hack this behavior but fails. The reward program that InterEvolve evolves instead walks the robot around the box and pushes it to turn, with the controller unchanged. **Bottom:** InterEvolve performs more evolved skills in simulation and on a real Unitree G1. *Videos are on supplementary webpage*.

We introduce *InterEvolve*, which builds this interface from reward programs and a behavioral foundation model that executes them without training (Fig. 1). A *reward program* is a sequence of stages, each with a reward and a completion condition, plus tunable weights and thresholds. To execute a program, we use a forward-backward (FB) behavioral foundation model (Touati & Ollivier, 2021; Tirinzoni et al., 2025; Li et al., 2026), which maps a reward to a latent prompt that elicits the corresponding behavior from one fixed policy. Existing humanoid FB models, however, observe only the body, so rewards that differ only in the object's goal can collapse into one behavior. We extend the model in two ways: we attach trainable object residuals that read object features to its frozen body networks, and we train these residuals on large-scale human-object interaction (HOI) data.

With execution in place, the central challenge becomes reward formulation itself. It is especially pronounced in loco-manipulation, where a naively handcrafted reward can fail even when the controller contains the relevant motor capabilities. InterEvolve therefore lets a large language model (LLM) agent revise reward programs *in context*: with its weights fixed, the agent combines task intent and execution feedback from simulation to refine the rewards and their composition. Instead of collecting references and training a policy for each new behavior, InterEvolve spends compute proposing, tuning, and verifying reward programs. This mirrors test-time scaling in language models, where more inference compute yields better solutions (Snell et al., 2025; Brown et al., 2024).

We organize this evolution around two complementary objectives: improving the reward program for the current task and retaining what is learned to guide future tasks. The two parts of a program must be searched together: its *structure*, meaning which rewards and stages it uses, and its *constants*, meaning the weights and thresholds. A promising structure can still fail under poor weights, so the agent revises the structure while an inner search with the covariance matrix adaptation evolution strategy (CMA-ES) (Hansen & Ostermeier, 2001) tunes the constants of each proposal. Parallel simulation supplies batches of execution evidence, so each revision is guided by a comparison over many rollouts. For future tasks, InterEvolve summarizes successful reward programs as a *skill library*. Later tasks can retrieve these experiences without rediscovering them from scratch.

Our contributions are threefold. First, a framework for self-evolving humanoid loco-manipulation that shifts task-specific compute from training to test-time search: an LLM agent learns in context to write and adapt reward programs that a fixed, reusable controller executes for new tasks and scenes. Second, an object-aware FB behavioral foundation model that translates reward objectives into whole-body interaction without per-task policy optimization. Third, an evaluation of how reward design and accumulated experience affect reference tracking and goal-conditioned tasks, show-

ing that success grows with search. We further show unseen tasks and long-horizon compositions in simulation and fully autonomous deployment on a physical Unitree G1 from onboard perception.

2 Related Work

Humanoid loco-manipulation and task interfaces. Humanoid control has grown reusable, from simulated characters (Peng et al., 2018; 2022) to real-robot whole-body policies learned from teleoperation and human motion (Fu et al., 2024; He et al., 2024; Ji et al., 2024; Chen et al., 2025; Yin et al., 2026; Liao et al., 2026; Ze et al., 2025; Luo et al., 2026). Loco-manipulation policies learn grasping and contact-rich interaction from references (Luo et al., 2024; Xu et al., 2025; Tessler et al., 2025; Wang et al., 2025b; Xu et al., 2026b) and transfer to real robots (Liu et al., 2025; Li et al., 2024; Sun et al., 2025; Yang et al., 2025; Zhao et al., 2025; Fu et al., 2026; Wang et al., 2025a; He et al., 2026). Others condition control on multimodal prompts (Kareer et al., 2025; Xue et al., 2025; Ding et al., 2025; Deng et al., 2026; Kalaria et al., 2025; Jiang et al., 2026; Xie et al., 2026), sequences existing skills with planners (Yuan et al., 2025; Wen et al., 2025; Ren et al., 2026; Sun et al., 2026; Xiao et al., 2024; Tevet et al., 2025), or imitates video-imagined interactions (Chen et al., 2026). InterEvolve instead states the task as a reward program, which says which contacts and stage transitions matter, and revises it after each attempt.

Behavioral foundation models and reward inference. Successor features decouple occupancy from reward (Dayan, 1993; Barreto et al., 2017), and FB representations factorize the successor measure so a backward map projects any reward into a latent prompt (Touati & Ollivier, 2021; Touati et al., 2023). Humanoid behavioral foundation models thus run one policy for new rewards, goals, or references without task-specific training (Tirinzoni et al., 2025; Li et al., 2026). Follow-ups refines the representation (Cetin et al., 2025; Bagatella et al., 2026), searches the latent space (Sikchi et al., 2025b), infers tasks online (Rupf et al., 2025; Bagot et al., 2026), grounds language (Sikchi et al., 2025a), or trains a per-task encoder (Vainshtein et al., 2025). These models observe only the body, so rewards that differ only in what happens to an object collapse into one behavior. Our object-aware FB model removes this limit.

Reward design and agents that learn from execution. Language models write reward functions from task descriptions (Xie et al., 2024), refine them from simulator feedback (Ma et al., 2024a;b), and search over reward designs (Zhang et al., 2025; Gao et al., 2025; Lee et al., 2026), including for humanoid locomotion (Wu et al., 2025), but each candidate reward is realized by training a policy. Language to Rewards optimizes LLM-written rewards with model predictive control instead (Yu et al., 2023; Liang et al., 2024). MotionDisco evolves humanoid motions with an LLM and trajectory optimization before training trackers for them (Taouil et al., 2026), and ROSETTA builds multi-stage reward programs from language preferences (Srivastava et al., 2026). Coding agents likewise refine executable plans from feedback and retain reusable experience (Liang et al., 2023; Zha et al., 2024; Zhou et al., 2024; Lu et al., 2026; Elmaaroufi et al., 2026; Xiao et al., 2026; Wang et al., 2026a). In InterEvolve, each candidate reward costs a batch of rollouts on a fixed controller rather than an RL run, and verified programs form a skill library for later tasks.

Test-time compute and search. Language models improve with more inference compute through repeated sampling, step-level search, and long reasoning (Snell et al., 2025; Brown et al., 2024; Guo et al., 2025), and program-search systems evolve code against an automatic evaluator (Romera-Paredes et al., 2024; Novikov et al., 2025). In control, test-time search usually plans actions with a dynamics model (Hansen et al., 2024) or steers a pretrained humanoid policy (Zhang et al., 2026; Seo et al., 2026; Cao et al., 2026). InterEvolve searches over task specifications instead, keeping the controller fixed and verifying proposed reward programs in parallel rollouts.

3 Method

InterEvolve adapts a fixed whole-body controller to new loco-manipulation tasks by searching over reward programs at test time, and keeps verified programs in a skill library for later tasks to build on (Fig. 2). Sec. 3.1 formalizes this search problem and its budget. Sec. 3.2 defines reward programs, the editable task strategies that the search operates on. Sec. 3.3 shows how an object-aware forward-backward (FB) behavioral foundation model executes a reward program without retraining. Sec. 3.4

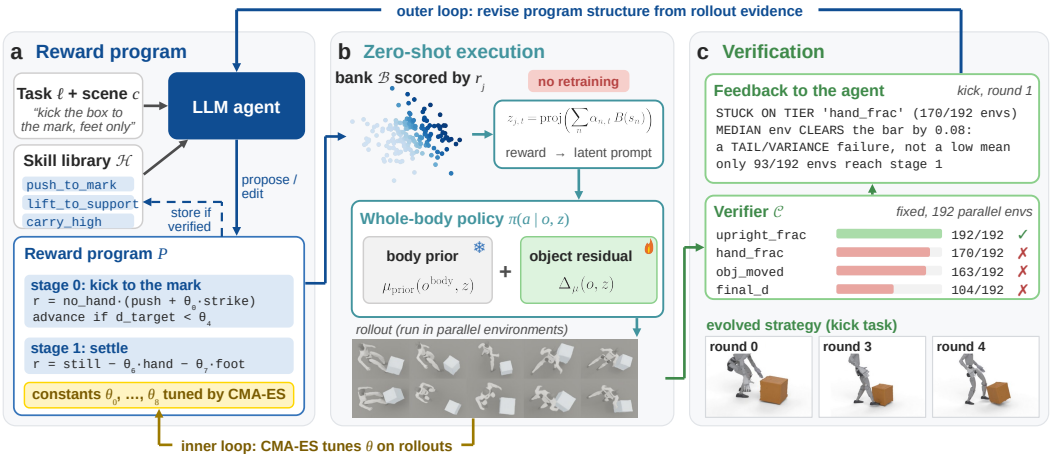


Figure 2: **InterEvolve overview.** (a) Given a task and scene, an LLM agent writes a staged reward program, drawing on verified programs in the skill library. (b) The forward-backward model scores the active stage's reward over a fixed state bank and projects it into a latent prompt, which a frozen body prior and a pretrained object residual execute without retraining. (c) A fixed verifier scores parallel rollouts, and criterion-level feedback drives the next structural revision (outer loop), while CMA-ES tunes the program constants (inner loop).

presents the evolution: an LLM agent revises program structure, CMA-ES (Hansen & Ostermeier, 2001) calibrates program constants, and the skill library carries verified programs to later tasks.

3.1 Problem Setup

A task is a language request ℓ and a scene context c, such as object poses and obstacles. In Fig. 2, ℓ asks the robot to kick a box to a mark with its feet only. A fixed pretrained controller $\pi(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{z})$ maps policy observation $\boldsymbol{o}_t$ and latent prompt $\boldsymbol{z}$ to action $\boldsymbol{a}_t$. The agent links them through a reward program P of staged rewards (Sec. 3.2), whose active-stage reward becomes $\boldsymbol{z}$ (Sec. 3.3). Executing P from a scenario, a randomized initial condition, yields a trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_T)$ over the privileged body-object state $\boldsymbol{s}_t$ that rewards and the verifier read. Success is judged by a verifier $\mathcal{C} = \{C_i\}_{i=1}^{N_C}$ whose criteria $C_i(\tau)$ check task outcomes and physical constraints, such as staying upright and keeping the hands off the box (Fig. 2c). For language-specified tasks, an LLM-assisted parser instantiates $\mathcal{C}$ from ℓ and c once, and $\mathcal{C}$ then stays fixed, so the agent cannot ease a task by rewriting its evaluator. A request thus goes through $(\ell, c) \to P \to \boldsymbol{z} \to \tau \to \mathcal{C}(\tau)$, and we cast adaptation as a search over P alone: within a budget of K rounds, find a program whose trajectories satisfy every criterion in $\mathcal{C}$. The search keeps a *current best program*, the strongest one $\mathcal{C}$ has confirmed so far, and each round tries to replace it (Sec. 3.4). Across tasks, a skill library $\mathcal{H}$ stores each task's final program and verifier results, so the agent can start a new task from earlier solutions, such as the push, lift, and carry programs in Fig. 2a.

3.2 Reward Programs as Editable Task Strategies

The search needs a strategy representation that can specify multi-stage, contact-rich interaction and can be edited in response to execution evidence. A reward program $P = \big(\boldsymbol{\theta}, \{(r_j, g_j)\}_{j=1}^{J}\big)$ consists of an ordered list of J stages and a vector of tunable constants $\boldsymbol{\theta}$. Each stage j is one phase of interaction, such as orienting toward an object, acquiring contact, transporting, or releasing. As shown in Figure 2, it holds reward code r_j, which scores body-object states through declared features, and a completion condition g_j, which reads live rollout context such as object pose, contact status, and stage time and returns whether the stage is complete. Execution stays in stage j until g_j holds and then advances to stage $j+1$, and the last stage runs until the episode ends. The constants $\boldsymbol{\theta}$ are the numbers that r_j and g_j read, such as reward weights, tolerances, kernel widths, and stage thresholds.

This representation gives the agent explicit handles for evolution: what each stage rewards (r_j), when execution moves to the next stage (g_j), and the weights and thresholds that calibrate both

($\boldsymbol{\theta}$). The controller, in turn, realizes each stage with the motor behaviors learned in pretraining. To switch from lifting an object to sliding it, for example, the agent edits the object-motion rewards and stage conditions instead of specifying a new joint trajectory. The representation also separates the *structure* of a program, namely its stages, reward terms, and completion conditions, from its constants $\boldsymbol{\theta}$, so the agent revises the structure while a numerical optimizer tunes $\boldsymbol{\theta}$ (Sec. 3.4).

3.3 Executing reward programs without retraining

Evaluating each candidate program by training a policy for it would make search prohibitively slow. We instead execute programs with an FB behavioral foundation model (Touati & Ollivier, 2021; Tirinzoni et al., 2025; Li et al., 2026), which we call the motor model and whose actor is the controller π of Sec. 3.1. A latent prompt $\boldsymbol{z}$ indexes a policy, and forward and backward maps factorize the discounted future-state occupancy of that policy $M^{\pi_{\boldsymbol{z}}}(\mathrm{d}\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \approx F(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})^\top B(\boldsymbol{s}')\,\rho(\mathrm{d}\boldsymbol{s}')$. For any reward r, integrating it against this occupancy gives $Q_r^{\pi_{\boldsymbol{z}}}(\boldsymbol{s}, \boldsymbol{a}) \approx F(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})^\top \widetilde{\boldsymbol{z}}_r$ with $\widetilde{\boldsymbol{z}}_r = \mathbb{E}_\rho[r(\boldsymbol{s}')B(\boldsymbol{s}')]$. A new reward therefore requires only a new prompt for the same policy.

From stage reward to latent prompt. We estimate this prompt on a reward-inference bank $\mathcal{B} = \{\boldsymbol{s}_n\}_{n=1}^N$ of body-object states. The bank is sampled once, after motor-model training and before any search, from its multi-task, multi-object training replay, which already covers phases such as contact acquisition, transport, and release. For each bank state we cache $B(\boldsymbol{s}_n)$ and the features reward code reads; the bank stays fixed during search (Sec. B.1). Let $\hat{r}_{j,n,t}$ be the reward of stage j on bank state $\boldsymbol{s}_n$, evaluated with the live rollout context at time t (Fig. 2b). The prompt is

$$\boldsymbol{z}_{j,t} = \mathrm{proj}\left(\sum_{n=1}^{N} \alpha_{j,n,t}\, B(\boldsymbol{s}_n)\right), \qquad \alpha_{j,n,t} \propto \hat{r}_{j,n,t} \exp(\beta\, \hat{r}_{j,n,t}), \tag{1}$$

where the normalized weights $\alpha_{j,n,t}$ tilt the estimate toward high-reward states, e.g., a foot striking the box in stage 0 of the kick program. The rest of the bank keeps broad coverage, so other learned behaviors can still support the task (Sec. B). As stage rewards read live context, the prompt is recomputed at every control step and drives the controller $\pi(\boldsymbol{a}_t \mid \boldsymbol{o}_t, \boldsymbol{z}_{j,t})$.

Object-aware interaction representation. The pretrained FB model of BFM-Zero (Li et al., 2026) observes only the body. Rewards that differ only in where the object should go can therefore collapse into the same behavior. We give all three of its networks, the actor and the forward and backward maps, access to the object, while each keeps its pretrained body branch frozen (Fig. 2b). The actor keeps the local body observations of BFM-Zero and adds heading-frame object features: object position, orientation, linear and angular velocity, and distance-decayed vectors from body links to the nearest object surface (Xu et al., 2025). Its mean action adds a trainable object-conditioned residual to the frozen body prior: $\boldsymbol{\mu}_\pi(\boldsymbol{o}_t, \boldsymbol{z}_{j,t}) = \tanh\big(\boldsymbol{\mu}_{\mathrm{prior}}(\boldsymbol{o}_t^{\mathrm{body}}, \boldsymbol{z}_{j,t}) + \boldsymbol{\Delta}_\mu(\boldsymbol{o}_t, \boldsymbol{z}_{j,t})\big)$. where $\boldsymbol{o}_t^{\mathrm{body}}$ is the body-only part of $\boldsymbol{o}_t$. The forward and backward maps are extended in the same way, each adding a trainable object residual to a frozen body branch. We train the residual branches on human-object interaction data with the FB objective and a demonstration discriminator (Sec. A), so the motor model becomes object-aware while the frozen prior keeps its body behaviors.

3.4 Evolving reward programs at test time

InterEvolve searches over programs with two nested loops (Algorithm 1), because program structure and constants call for different searchers. An LLM agent is well suited to choosing a program's reward terms and stages, whereas a numerical optimizer is better at finding the constants that make a given structure work. The outer loop therefore lets the agent revise program structure, and the inner loop tunes $\boldsymbol{\theta}$ with CMA-ES (Hansen & Ostermeier, 2001). Both loops evaluate each candidate on a set of scenarios in parallel, so every decision rests on many rollouts.

Outer loop: structural revision. Each round starts with one prompt to the agent (Sec. C.2). The prompt contains the task request, scene context, and verifier $\mathcal{C}$, the current best program with its rollout feedback, and the skill library $\mathcal{H}$. The agent learns only from this context, as its weights stay fixed. It proposes several new programs. A validator discards programs that break the code rules, such as reading an undeclared feature, before any rollout. Keeping the current best in context encourages targeted repairs when a strategy is close, while allowing new structures when it is not.

Inner loop: numerical calibration. The same structure can fail under poor relative weighting, so each valid proposal is calibrated before judging. The inner loop freezes a program's stages and reward terms and searches only over $\boldsymbol{\theta}$ (e.g. the strike weight θ_0 and the stage-switch distance θ_4 in Fig. 2a), within agent-declared bounds. CMA-ES samples constants, the simulator evaluates each on the search scenarios in parallel, and the sampling distribution moves toward better-scoring constants.

Selection and feedback. Tuned candidates are compared with the current best program on the same search scenarios, so outcome differences reflect the programs rather than their initial conditions. The strongest candidate is then re-evaluated together with the current best, three times each. It replaces the current best only if its improvement under $\mathcal{C}$ exceeds the run-to-run evaluation noise. An accepted edit is one whose gain carries over to new initial conditions. The next prompt reports these outcomes per criterion and per stage. For round 1 of the kick task (Fig. 2c), it reports that the hand criterion still fails in a minority of environments although the median environment passes it, and that only 93 of 192 environments reach stage 1. Such reports locate the stage to repair, and typical repair targets are missing contact and premature stage advancement (Sec. C.2).

Budget and termination. The budget K counts revision rounds after the initial program. Each round makes one agent call and spends 192 tuning rollouts on every valid proposal plus 192 rollouts to confirm the top candidate, so its cost is bounded, and simulation dominates it (Sec. C.4). Search stops once the K rounds are spent, or earlier if the current best program satisfies every criterion in $\mathcal{C}$ on the confirmation scenarios. It returns the current best program, which is then added to $\mathcal{H}$.

Skill library. The skill library is a text document for summarizing tasks, placed in the agent's prompt every round. Each entry records a completed task: its scene, task text, verifier with the pass rate of each criterion, and the selected program with its tuned constants. With these entries in context, the agent can compose or adapt established programs instead of searching from scratch.

4 Experiments

This section verifies that our low-level controller already holds much of the competence a new task needs, and that test-time search over reward programs makes it accessible, through three questions. (**i**) *Competence:* can one object-aware motor model provide reusable whole-body interaction, and does better reward design improve tracking (Sec. 4.2)? (**ii**) *Access:* without reference motion, does execution-guided evolution reach behaviors fixed programs miss, and which search components produce the gain (Sec. 4.3)? (**iii**) *Accumulation:* does a skill library solve composite tasks whose contact modes it covers, and do selected programs transfer to new conditions and hardware (Sec. 4.4)?

4.1 Experimental setup

Data and embodiment. We pretrain the separated FB models on human-object interaction from OMOMO (Li et al., 2023) and GRAB (Taheri et al., 2020). OMOMO covers whole-body manipulation of large objects such as boxes and tables, GRAB whole-body grasping of small objects. We retarget both to the Unitree G1 (Unitree Robotics) with rubber hands and to the G1 with Inspire hands (Inspire Robots), adapting OmniRetarget (Yang et al., 2025) to the dexterous hands. Following ULTRA (He et al., 2026), we use the four box-like OMOMO objects: large box, plastic box, small box, and suitcase. For each object we hold out 50 clips for evaluation and train on the remaining 3,866; the held-out large-box clips form the tracking benchmark. The Inspire-hand G1 with GRAB shows qualitatively that InterEvolve extends to dexterous whole-body manipulation. DeepSeek-V4-Flash (Xu et al., 2026a) writes the reward programs. All controllers run in Isaac Lab (Mittal et al., 2025), and the transfer study replays selected programs in MuJoCo (Todorov et al., 2012) and on the real G1. For general-purpose tasks, we create a dedicated evaluation benchmark (Sec. D.3).

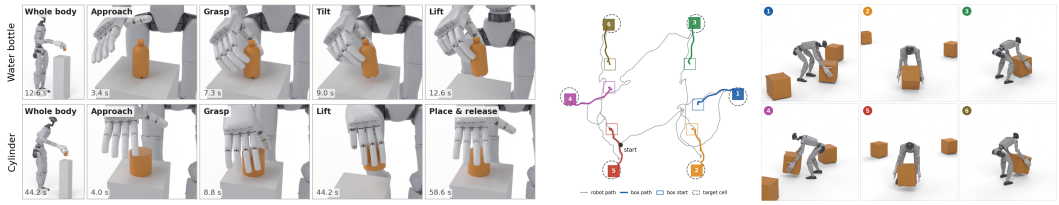
Metrics. For tracking, E_h and E_o are the mean body-joint and object-surface errors against the reference in cm, and the success rate (**SR**) is the fraction of clips tracked to the end without a fall, an object deviation above 0.5 m, or a lost required contact (Xu et al., 2025). For general-purpose tasks, SR requires every task criterion and physical constraint to hold in the same rollout, while earned tiers (**Earned**) measure the mean fraction of criteria passed. Standing still already passes about half of the criteria (Table 2), so SR is the more representative primary metric. Every reported metric is averaged over three independent evaluation runs. We report search cost as single-GPU wall time

Table 1: **Reference tracking** on held-out clips. An evolved reward lifts our success above the dedicated tracker, which keeps the lowest errors; bold marks the best reward-promptable controller. Errors in cm, SR in %. Evolving: with a reward the agent discovered for tracking.

Controller	Evolving	$E_h \downarrow$	$E_o \downarrow$	SR↑
ULTRA	✗	**15.68**	**21.45**	66
BFM-Zero	✗	28.36	62.33	8
InterEvolve	✗	27.20	30.80	60
InterEvolve	✓	19.80	24.92	**72**

Table 2: **Goal-conditioned tasks**, averaged over eight task families. Evolution more than doubles the success of the best calibrated fixed program. Tune: CMA-ES calibration. Evolving: execution-guided revision. SR and earned tiers in %, tokens in millions (one family). whereas InterEvolve runs five rounds and calibrates several proposals per round, hence its higher GPU-h.

Program	Tune	Evolving	SR↑	Earned↑	GPU-h↓	Tokens↓
Inaction	–	–	0.0	49.1	n/a	n/a
Human	✗	✗	8.0	65.2	n/a	n/a
Agent	✗	✗	32.2	73.3	n/a	0.03
Human	✓	✗	18.0	75.2	0.6	n/a
Agent	✓	✗	34.6	82.5	0.6	0.03
InterEvolve	✓	✓	**86.5**	**95.6**	2.1	0.23



(a) Dexterous manipulation of small objects

(b) Six boxes arranged in one take

Figure 3: **Dexterous hands and long-horizon composition. (a)** The G1 with Inspire hands, tracking a reference with the tracking reward calibrated by InterEvolve. **Left**: whole-body view at the moment of lift; **Right**: close-ups of the manipulation phases. **(b)** Arranging six boxes into a ring in one take. **Left**: top-down trace of the robot (gray) and each box from its start (outlined square) to its target cell (dashed circle); numbers give the push order. **Right**: the robot pushing each box.

(GPU-h) and total LLM tokens (prompt + output, millions) for one complete search on one task family. Fixed external criteria score each general-purpose task; details are in Sec. D.

Baselines. For tracking, we adapt the tracking policy of ULTRA (He et al., 2026), a generalist trained with motion imitation to follow loco-manipulation references, which we train on the same data as our motor model, and with BFM-Zero (Li et al., 2026), a pretrained body-only FB model. For general-purpose tasks, no public method applies directly to our setting under complex scene context, so we compare InterEvolve with fixed programs on the same controller: a human-written program and the agent's initial program, each executed directly or after CMA-ES calibration.

4.2 Reusable control for reference tracking

Object-aware FB can match a dedicated tracker. Table 1 tests whether one controller can mimic body motion and object interaction. ULTRA, an RL tracker trained for tracking alone, has the lowest errors. Our motor model, which also serves reward programs, approaches its success rate, whereas the body-only BFM-Zero loses the object on most clips, so object-aware pretraining supplies the interaction competence that reward programs repurpose. Without the link-to-surface vectors or the object state, the controller still follows the body but rarely succeeds (Table 6), and the same network trained from scratch almost never works, whereas residuals on the frozen body prior improve with width (Table 7), so the prior carries body skills and the residual learns interaction.

Evolving rewards unlock the potential for motion tracking. *Reward design matters* even in tracking. Our actor observes the body and object in a local frame, so world-frame drift from the reference is invisible to it. The agent finds a reward that supplies this signal: an object-anchored drift correction blended with the reference prompt (Sec. B.2). It reduces both errors, lifts success above ULTRA, and nearly halves the final drift of root and object (Fig. 5). The same procedure also tracks dexterous whole-body manipulation of small objects with Inspire hands (Fig. 3(a)).

Table 3: **Ablation of the search design.** Stages and numerical calibration are most important designs. Each row disables one component (✗) of the full system; definitions are in Sec. D.5.

Search components					Results			
CMA-ES tuning	Targeted edits	Multi-scenario	Scene context	Multi-stage	SR (%)↑	Earned (%)↑	GPU-h ↓	Tokens (M)↓
✓	✓	✓	✓	✓	**86.5**	**95.6**	2.1	0.23
✗	✓	✓	✓	✓	51.6	89.9	1.7	0.27
✓	✗	✓	✓	✓	68.9	92.3	2.7	0.20
✓	✓	✗	✓	✓	68.4	89.9	**1.0**	**0.17**
✓	✓	✓	✗	✓	78.7	93.2	2.0	0.21
✓	✓	✓	✓	✗	44.7	86.9	2.1	0.19

4.3 Reward-program evolution for general-purpose tasks

We next remove the reference motion and specify only task outcomes and constraints. The benchmark spans eight task families (Sec. D.3): pushing to a mark, carrying at several heights, tipping or reorienting an object, feet-only interaction, lifting onto a support, and pushing through gates or around obstacles. Success depends on contact mode, obstacle layout, and when to switch phases. Each run starts from the task text and the observed scene, and only the reward program changes.

Formulation, not hyperparameters, limits a written reward. Table 2 separates three effects. A reward written once is not enough: even the agent's initial program, which outperforms the human-designed one, fails in most episodes. CMA-ES calibration of the weights alone brings limited gains, since better weights cannot repair an objective with the wrong structure. InterEvolve more than doubles the success of the best calibrated program, and because both fixed programs are calibrated, this gap comes from structural changes: what is rewarded and how objectives are staged.

Evolution helps most where fixed programs fail. Calibrated fixed programs solve pushing to a mark but succeed in at most a quarter of the episodes when carrying at chest height, tipping onto a new face, or pushing through a gate or around an obstacle (Table 13). Evolution lifts each of these families above 70% success. Kicking to a mark remains the hardest, since kicks are rare in the training data and need strong whole-body coordination from the reward.

Staging and calibration matter most. Every search component contributes (Table 3; Sec. D.5 defines each variant). Forcing a single stage causes the largest drop, so contact-rich interaction needs a task decomposed into phases with their own objectives. Removing numerical tuning is nearly as harmful, since a program can only be judged once its constants are calibrated. Dropping targeted edits, so that the agent rewrites each program from scratch, or multi-scenario evaluation, so that each candidate runs from a single start, costs a similar amount: repairing a working program is more reliable than regenerating it, and varied starts tell a robust edit from a lucky one. Scene context, which describes obstacles and supports, has the smallest effect but still helps.

Success grows with test-time search, though not monotonically. Table 4 tracks the current best program over rounds, with the controller weights fixed. The first round brings most of the gain by removing the initial program's coarse failures, and later rounds resolve the remaining criteria. Earned tiers rise monotonically, whereas full success dips from round 2 to round 3, since it requires all criteria in the same rollout and fixing one criterion can briefly break another. Tokens grow linearly with rounds, but simulation dominates the cost, with minutes per round in the language model against tens of minutes of rollouts (Sec. C.4), so a faster simulator such as mjlab (Zakka et al., 2026) in place of Isaac Lab (Mittal et al., 2025) could shorten search substantially. Figure 7 shows that revised rewards change the contact strategy itself.

Qualitative results and real deployment from onboard perception. Figures 8 and 9 show goal-conditioned skills evolved in simulation, in which acquisition, transport, placement, and ground-level handling arise from reward programs run by the shared controller. Figure 4 deploys programs evolved for three tasks on a physical Unitree G1. The robot detects the box with its egocentric camera, and FoundationPose (Wen et al., 2024) estimates its 6-D pose, which supplies the controller's object features. Each program is evolved and verified in simulation, then run on the robot.

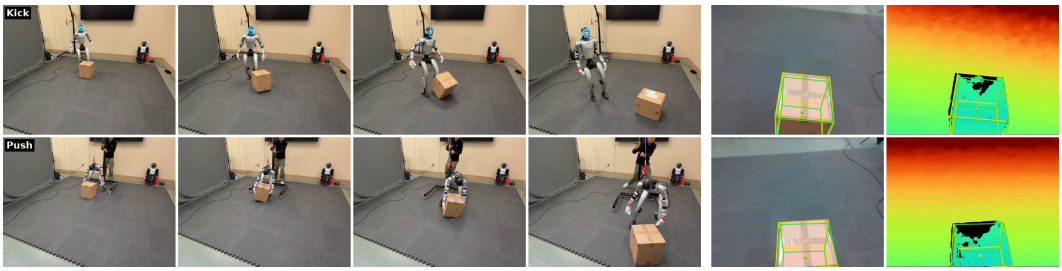


Figure 4: **Autonomous** execution of real-world G1 with **onboard camera** and task intent of kicking a box (top) and pushing a box (bottom). Each row runs a program evolved in simulation. The first four columns show the run from a third-person view, left to right in time. The last two columns are the robot's onboard color and depth images as it approaches the box, overlaid with the FoundationPose estimate (yellow) and the filtered pose that the controller receives (green).

Table 4: **Ablation on search rounds.** Most of the gain comes in round 1; earned tiers rise monotonically. The final row repeats Table 2.

Round	SR↑	Earned↑	Tokens
Initial	32.2	73.3	0.03
1	75.0	91.7	0.08
2	76.8	92.6	0.13
3	73.6	93.1	0.18
4 (final)	**86.5**	**95.6**	0.23

Table 5: **Reuse on composite tasks** unseen by every library, under one target-task budget. The full library solves the most episodes. Carry-only keeps the three carrying programs. Tokens (millions, mean over the tasks) exclude library construction. Every condition searches for three rounds.

Library	Relocate	Stack	Carry-place-kick	Tokens↓
Full	**8/10**	**4/10**	**5/10**	0.16
None	0/10	0/10	1/10	0.13
Carry-only	6/10	1/10	1/10	0.15

4.4 Reuse and Composition

Composite tasks need a library that covers their contact modes. Search experience is valuable only if later tasks can reuse it. Table 5 evaluates three composite tasks that each combine previously searched objectives in one episode: relocating an object between supports, stacking one box on another, and carrying, placing, then kicking an object. We compare the full skill library with no library and a carry-only library. With the full library, the agent solves most relocation episodes and about half of the stacking and carry-place-kick episodes, but almost none without a library. The carry-only library recovers most of relocation but helps little on stacking or kicking. Reuse therefore depends on covering the contact modes a task needs, not on having any stored experience.

Search adapts stored programs. A finer ablation on carry, place, and kick shows that stored programs run without search almost never solve the task (Table 15), so the gain comes from search adapting them rather than from retrieval. Broader coverage still helps.

Searched skills chain into long-horizon tasks. Figure 3(b) chains searched programs to arrange six scattered boxes into a ring. An external planner orders the boxes, and each leg runs a walking program to the box followed by the searched pushing program. One controller executes all twelve phases from a single reset, so errors accumulate across legs instead of being reset away. In the take shown, all six boxes end within 11 cm of their cells and no placed box is disturbed by later legs. On a Unitree G1, the same composition chains basic skills in sequence.

5 Discussion and Conclusion

InterEvolve suggests a different division of labor for humanoid intelligence. A behavioral foundation model learns how to move once, and task knowledge lives outside its weights, in reward programs that a language model can read, edit, and test by execution. Reasoning and control thus improve on separate timescales: the controller with more interaction data, the agent with more test-time search and a growing program library, and neither requires retraining the other. Physical execution grounds the agent's reasoning, since it revises what it asks of the controller from observed outcomes rather than assumptions. We see this as a step toward humanoids that, like language models at inference time, improve by testing more, and accumulate skills as inspectable, reusable programs.

AI USE STATEMENT

A language model writes and revises reward programs in our method (Section 3.4). AI assistants aided reward-skill and experiment-harness development, job monitoring, literature discovery and consistency checks. The authors verified all AI-assisted content and are responsible for the paper.

REFERENCES

Marco Bagatella, Matteo Pirotta, Ahmed Touati, Alessandro Lazaric, and Andrea Tirinzoni. TD-JEPA: Latent-predictive representations for zero-shot reinforcement learning. In *ICLR*, 2026.

Louis Bagot, Mathieu Lefort, and Laëtitia Matignon. Exploration and online transfer with behavioral foundation models. *arXiv preprint arXiv:2606.29980*, 2026.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *NeurIPS*, 2017.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Jianuo Cao, Yuxin Chen, Yuzhen Song, Masayoshi Tomizuka, Chenran Li, and Thomas Tian. TEXEDO: Test time scaling for controller-aware language-conditioned humanoid motion generation. *arXiv preprint arXiv:2606.22998*, 2026.

Edoardo Cetin, Ahmed Touati, and Yann Ollivier. Finer behavioral foundation models via auto-regressive features and advantage weighting. *Reinforcement Learning Journal*, 6, 2025.

Jiahe Chen, ZiRui Wang, Feiyu Jia, Xiao Chen, Xiaojie Niu, Weishuai Zeng, Tianfan Xue, Xiaowei Zhou, Jiangmiao Pang, and Jingbo Wang. Imagine2Real: Towards zero-shot humanoid-object interaction via video generative priors. *arXiv preprint arXiv:2605.22272*, 2026.

Zixuan Chen, Mazeyu Ji, Xuxin Cheng, Xuanbin Peng, Xue Bin Peng, and Xiaolong Wang. GMT: General motion tracking for humanoid whole-body control. *arXiv preprint arXiv:2506.14770*, 2025.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4), 1993.

Zekai Deng, Ye Shi, Kaiyang Ji, Lan Xu, Shaoli Huang, and Jingya Wang. Human-object interaction via automatically designed VLM-guided motion policy. In *ICLR*, 2026.

Pengxiang Ding, Jianfei Ma, Xinyang Tong, Binghong Zou, Xinxin Luo, Yiguo Fan, Ting Wang, Hongchao Lu, Panzhong Mo, Jinxin Liu, Yuefan Wang, Huaicheng Zhou, Wenshuo Feng, Jiacheng Liu, Siteng Huang, and Donglin Wang. Humanoid-VLA: Towards universal humanoid control with visual integration. *arXiv preprint arXiv:2502.14795*, 2025.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026.

Yuhui Fu, Feiyang Xie, Chaoyi Xu, Jing Xiong, Haoqi Yuan, and Zongqing Lu. DemoHLM: From one demonstration to generalizable humanoid loco-manipulation. *IEEE Robotics and Automation Letters*, 11(4):4393–4400, 2026.

Zipeng Fu, Qingqing Zhao, Qi Wu, Gordon Wetzstein, and Chelsea Finn. HumanPlus: Humanoid shadowing and imitation from humans. In *CoRL*, 2024.

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

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

Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *ICLR*, 2024.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 2001.

Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris M. Kitani, Changliu Liu, and Guanya Shi. OmniH2O: Universal and dexterous human-to-humanoid whole-body teleoperation and learning. In *CoRL*, 2024.

Xialin He, Sirui Xu, Xinyao Li, Runpei Dong, Liuyu Bian, Yu-Xiong Wang, and Liang-Yan Gui. ULTRA: Unified multimodal control for autonomous humanoid whole-body loco-manipulation. *arXiv preprint arXiv:2603.03279*, 2026.

Inspire Robots. The Dexterous Hands. URL https://en.inspire-robots.com/product-category/the-dexterous-hands/.

Mazeyu Ji, Xuanbin Peng, Fangchen Liu, Jialong Li, Ge Yang, Xuxin Cheng, and Xiaolong Wang. ExBody2: Advanced expressive humanoid whole-body control. *arXiv preprint arXiv:2412.13196*, 2024.

Haoran Jiang, Jin Chen, Qingwen Bu, Li Chen, Modi Shi, Yanjie Zhang, Delong Li, Chuanzhe Suo, Chuang Wang, Zhihui Peng, and Hongyang Li. WholeBodyVLA: Towards unified latent VLA for whole-body loco-manipulation control. In *ICLR*, 2026.

Dvij Kalaria, Sudarshan S. Harithas, Pushkal Katara, Sangkyung Kwak, Sarthak Bhagat, Shankar Sastry, Srinath Sridhar, Sai Vemprala, Ashish Kapoor, and Jonathan Chung-Kuan Huang. DreamControl: Human-inspired whole-body humanoid control for scene interaction via guided diffusion. *arXiv preprint arXiv:2509.14353*, 2025.

Simar Kareer, Dhruv Patel, Ryan Punamiya, Pranay Mathur, Shuo Cheng, Chen Wang, Judy Hoffman, and Danfei Xu. EgoMimic: Scaling imitation learning via egocentric video. In *ICRA*, 2025.

Hojoon Lee, Ajay Subramanian, Ben Abbatematteo, Vijay Veerabadran, Pedro Matias, Karl Ridgeway, and Nitin Kamra. RDA: Reward design agent for reinforcement learning. *arXiv preprint arXiv:2606.01672*, 2026.

Jiaman Li, Jiajun Wu, and C. Karen Liu. Object motion guided human motion synthesis. *ACM Transactions on Graphics*, 2023.

Jinhan Li, Yifeng Zhu, Yuqi Xie, Zhenyu Jiang, Mingyo Seo, Georgios Pavlakos, and Yuke Zhu. OKAMI: Teaching humanoid robots manipulation skills through single video imitation. In *CoRL*, 2024.

Yitang Li, Zhengyi Luo, Tonghe Zhang, Cunxi Dai, Anssi Kanervisto, Andrea Tirinzoni, Haoyang Weng, Kris M. Kitani, Mateusz Guzek, Ahmed Touati, Alessandro Lazaric, Matteo Pirotta, and Guanya Shi. BFM-Zero: A promptable behavioral foundation model for humanoid control using unsupervised reinforcement learning. In *ICLR*, 2026.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *ICRA*, 2023.

Jacky Liang, Fei Xia, Wenhao Yu, Andy Zeng, Montserrat Gonzalez Arenas, Maria Attarian, Maria Bauza, Matthew Bennice, Alex Bewley, Adil Dostmohamed, et al. Learning to learn faster from human feedback with language model predictive control. *arXiv preprint arXiv:2402.11450*, 2024.

Qiayuan Liao, Takara E Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C Karen Liu. Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion. *Science Robotics*, 11(117):eadx8924, 2026.

Fukang Liu, Zhaoyuan Gu, Yilin Cai, Ziyi Zhou, Hyunyoung Jung, Jaehwi Jang, Shijie Zhao, Sehoon Ha, Yue Chen, Danfei Xu, and Ye Zhao. Opt2Skill: Imitating dynamically-feasible whole-body trajectories for versatile humanoid loco-manipulation. *IEEE Robotics and Automation Letters*, 2025.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Zhengyi Luo, Jinkun Cao, Sammy Christen, Alexander Winkler, Kris M. Kitani, and Weipeng Xu. OmniGrasp: Grasping diverse objects with simulated humanoids. In *NeurIPS*, 2024.

Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Fernando Castañeda, Sirui Chen, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, Jinhyung Park, David Sami, Zi Wang, Xingye Da, Runyu Ding, Cyrus Hogg, Lina Song, Edy Lim, Eugene Jeong, Tairan He, Haoru Xue, Wenli Xiao, Simon Yuen, Jan Kautz, Yan Chang, Umar Iqbal, Linxi Jim Fan, and Yuke Zhu. SONIC: Supersizing motion tracking for natural humanoid whole-body control. *Science Robotics*, 11(117): eaed4592, 2026. doi: 10.1126/scirobotics.aed4592.

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 *ICLR*, 2024a.

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 *RSS*, 2024b.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Munoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

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

Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. ASE: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Transactions on Graphics*, 2022.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Peng Ren, Haoyang Ge, Chuan Qi, Cong Huang, Hong Li, Jiang Zhao, Pei Chi, and Kai Chen. CyboWaiter: A physical agentic framework for humanoid whole-body locomotion-manipulation. *arXiv preprint arXiv:2603.10675*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, et al. Mathematical discoveries from program search with large language models. *Nature*, 2024.

Thomas Rupf, Marco Bagatella, Marin Vlastelica, and Andreas Krause. Optimistic task inference for behavior foundation models. *arXiv preprint arXiv:2510.20264*, 2025.

Yunsoo Seo, Sol Choi, Euncheol Im, Myo Taeg Lim, and Yisoo Lee. RGB: RL guided whole-body MPPI for humanoid control. *arXiv preprint arXiv:2606.25123*, 2026.

Harshit Sikchi, Siddhant Agarwal, Pranaya Jajoo, Samyak Parajuli, Caleb Chuck, Max Rudolph, Peter Stone, Amy Zhang, and Scott Niekum. RLZero: Direct policy inference from language without in-domain supervision. In *NeurIPS*, 2025a.

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

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *ICLR*, 2025.

Sanjana Srivastava, Kangrui Wang, Yung-Chieh Chan, Tianyuan Dai, Manling Li, Ruohan Zhang, Mengdi Xu, Jiajun Wu, and Li Fei-Fei. ROSETTA: Constructing code-based reward from unconstrained language preference. In *ICLR*, 2026.

Wandong Sun, Luying Feng, Baoshi Cao, Yang Liu, Yaochu Jin, and Zongwu Xie. ULC: A unified and fine-grained controller for humanoid loco-manipulation. *arXiv preprint arXiv:2507.06905*, 2025.

Ziyi Sun, Jingwen Chen, Yuxi Wang, Xiuze Xia, Long Cheng, Zhaoxiang Zhang, and Yujun Dong. CHOREO: Every humanoid skill as a trajectory. *arXiv preprint arXiv:2609.22274*, 2026.

Omid Taheri, Nima Ghorbani, Michael J. Black, and Dimitrios Tzionas. GRAB: A dataset of whole-body human grasping of objects. In *ECCV*, 2020.

Ilyass Taouil, Michal Ciebelski, Shafeef Omar, Haizhou Zhao, Angela Dai, Aaron M. Johnson, and Majid Khadiv. MotionDisco: Motion discovery for extreme humanoid loco-manipulation. *arXiv preprint arXiv:2606.06139*, 2026.

Chen Tessler, Yifeng Jiang, Erwin Coumans, Zhengyi Luo, Xue Bin Peng, and Gal Chechik. Maskedmanipulator: Versatile whole-body control for loco-manipulation. In *SIGGRAPH Asia*, 2025.

Guy Tevet, Sigal Raab, Setareh Cohan, Daniele Reda, Zhengyi Luo, Xue Bin Peng, Amit Bermano, and Michiel Van de Panne. CLoSD: Closing the loop between simulation and diffusion for multi-task character control. In *ICLR*, 2025.

Andrea Tirinzoni, Ahmed Touati, Jesse Farebrother, Mateusz Guzek, Anssi Kanervisto, Yingchen Xu, Alessandro Lazaric, and Matteo Pirotta. Zero-shot whole-body humanoid control via behavioral foundation models. In *ICLR*, 2025.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. In *NeurIPS*, 2021.

Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? In *ICLR*, 2023.

Unitree Robotics. Unitree G1. URL https://www.unitree.com/g1/.

Ron Vainshtein, Zohar Rimon, Shie Mannor, and Chen Tessler. Task Tokens: A flexible approach to adapting behavior foundation models. *arXiv preprint arXiv:2503.22886*, 2025.

Huayi Wang, Wentao Zhang, Runyi Yu, Tao Huang, Junli Ren, Feiyu Jia, Zirui Wang, Xiaojie Niu, Xiao Chen, Jiahe Chen, et al. PhysHSI: Towards a real-world generalizable and natural humanoid-scene interaction system. *arXiv preprint arXiv:2510.11072*, 2025a.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Yiqun Zhang, Zihan Wang, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution. *arXiv preprint arXiv:2608.11350*, 2026a.

Yinhuai Wang, Qihan Zhao, Runyi Yu, Hok Wai Tsui, Ailing Zeng, Jing Lin, Zhengyi Luo, Jiwen Yu, Xiu Li, Qifeng Chen, Jian Zhang, Lei Zhang, and Ping Tan. SkillMimic: Learning basketball interaction skills from demonstrations. In *CVPR*, 2025b.

Yinhuai Wang, Qihan Zhao, Yuen Fui Lau, Runyi Yu, Hok Wai Tsui, Qifeng Chen, Jingbo Wang, Jiangmiao Pang, and Ping Tan. HumanX: Toward agile and generalizable humanoid interaction skills from human videos. *arXiv preprint arXiv:2602.02473*, 2026b.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *CVPR*, 2024.

Congcong Wen, Geeta Chandra Raju Bethala, Yu Hao, Niraj Pudasaini, Hao Huang, Shuaihang Yuan, Baoru Huang, Anh Nguyen, Mengyu Wang, Anthony Tzes, and Yi Fang. Humanoid agent via embodied chain-of-action reasoning with multimodal foundation models for zero-shot loco-manipulation. *arXiv preprint arXiv:2504.09532*, 2025.

Zhenwei Wu, Jinxiong Lu, Yuxiao Chen, Yunxin Liu, Yueting Zhuang, and Luhui Hu. STRIDE: Automating reward design, deep reinforcement learning training and feedback optimization in humanoid robotics locomotion. *arXiv preprint arXiv:2502.04692*, 2025.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

Zeqi Xiao, Tai Wang, Jingbo Wang, Jinkun Cao, Wenwei Zhang, Bo Dai, Dahua Lin, and Jiangmiao Pang. Unified human-scene interaction via prompted chain-of-contacts. In *ICLR*, 2024.

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 *ICLR*, 2024.

Tianyi Xie, Haotian Zhang, Jinhyung Park, Zi Wang, Bowen Wen, Jiefeng Li, Xueting Li, Qingwei Ben, Haoyang Weng, Yufei Ye, David Minor, Tingwu Wang, Chenfanfu Jiang, Sanja Fidler, Jan Kautz, Linxi Fan, Yuke Zhu, Zhengyi Luo, Umar Iqbal, and Ye Yuan. GRAIL: Generating humanoid loco-manipulation from 3D assets and video priors. *arXiv preprint arXiv:2606.05160*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

Sirui Xu, Hung Yu Ling, Yu-Xiong Wang, and Liang-Yan Gui. InterMimic: Towards universal whole-body control for physics-based human-object interactions. In *CVPR*, 2025.

Sirui Xu, Samuel Schulter, Morteza Ziyadi, Xialin He, Xiaohan Fei, Yu-Xiong Wang, and Liang-Yan Gui. InterPrior: Scaling generative control for physics-based human-object interactions. *arXiv preprint arXiv:2602.06035*, 2026b.

Haoru Xue, Xiaoyu Huang, Dantong Niu, Qiayuan Liao, Thomas Kragerud, Jan Tommy Gravdahl, Xue Bin Peng, Guanya Shi, Trevor Darrell, Koushil Sreenath, and S. Shankar Sastry. LeVERB: Humanoid whole-body control with latent vision-language instruction. *arXiv preprint arXiv:2506.13751*, 2025.

Lujie Yang, Xiaoyu Huang, Zhen Wu, Angjoo Kanazawa, Pieter Abbeel, Carmelo Sferrazza, C. Karen Liu, Rocky Duan, and Guanya Shi. OmniRetarget: Interaction-preserving data generation for humanoid whole-body loco-manipulation and scene interaction. *arXiv preprint arXiv:2509.26633*, 2025.

Kangning Yin, Weishuai Zeng, Ke Fan, Minyue Dai, Zirui Wang, Qiang Zhang, Zheng Tian, Jingbo Wang, Jiangmiao Pang, and Weinan Zhang. UniTracker: Learning universal whole-body motion tracker for humanoid robots. *IEEE Robotics and Automation Letters*, 2026.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *CoRL*, 2023.

Haoqi Yuan, Yu Bai, Yuhui Fu, Bohan Zhou, Yicheng Feng, Xinrun Xu, Yi Zhan, Börje F. Karlsson, and Zongqing Lu. Being-0: A humanoid robotic agent with vision-language models and modular skills. *arXiv preprint arXiv:2503.12533*, 2025.

Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. mjlab: A lightweight framework for gpu-accelerated robot learning. *arXiv preprint arXiv:2601.22074*, 2026.

Yanjie Ze, Zixuan Chen, Joao Pedro Araujo, Zi-Ang Cao, Xue Bin Peng, Jiajun Wu, and C. Karen Liu. TWIST: Teleoperated whole-body imitation system. In *CoRL*, 2025.

Lihan Zha, Yuchen Cui, Li-Heng Lin, Minae Kwon, Montserrat Gonzalez Arenas, Andy Zeng, Fei Xia, and Dorsa Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In *ICRA*, 2024.

Chen Bo Calvin Zhang, Zhang-Wei Hong, Aldo Pacchiano, and Pulkit Agrawal. ORSO: Accelerating reward design via online reward selection and policy optimization. In *ICLR*, 2025.

John Z. Zhang, Maks Sorokin, Jan Brüdigam, Brandon Hung, Stephen Phillips, Dmitry Yershov, Farzad Niroui, Tong Zhao, Leonor Fermoselle, Xinghao Zhu, Chao Cao, Duy Ta, Tao Pang, Jiuguang Wang, Preston Culbertson, Zachary Manchester, and Simon Le Cléac'h. Sumo: Dynamic and generalizable whole-body loco-manipulation. *arXiv preprint arXiv:2604.08508*, 2026.

Siheng Zhao, Yanjie Ze, Yue Wang, C. Karen Liu, Pieter Abbeel, Guanya Shi, and Rocky Duan. ResMimic: From general motion tracking to humanoid whole-body loco-manipulation via residual learning. *arXiv preprint arXiv:2510.05070*, 2025.

Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *arXiv preprint arXiv:2306.12156*, 2023.

Zhiyuan Zhou, Pranav Atreya, Abraham Lee, Homer Rich Walke, Oier Mees, and Sergey Levine. Autonomous improvement of instruction following skills via foundation models. In *CoRL*, 2024.

A MOTOR MODEL

This section details the object-aware motor model of Sec. 3.3, which is pretrained once before any reward-program search and stays fixed during search.

A.1 INTERACTION FEATURES

For the G1 with rubber hands, the 105-dimensional interaction encoding tells the controller where the object is and how close each body link is to its surface. It concatenates a 15-dimensional object state with a 90-dimensional link term: one 3-D displacement from each of 30 body links to the object surface. The object state holds position, a six-dimensional orientation, and linear and angular velocity. Horizontal position is relative to the robot root and height stays in world coordinates, while orientation and velocities use the heading frame. For each link, the displacement to the nearest sampled object-surface point is normalized and weighted by an exponential of its distance. These features encode proximity rather than contact force. The Inspire model keeps the same object state and adds the 24 Inspire finger links (12 per hand) to the link geometry, 54 links in total.

A.2 RESIDUAL ARCHITECTURE AND TRAINING OBJECTIVE

The actor, F, and B each keep their body branch frozen and train only their object residual (Eq. 3.3) on the human-object interaction data of Sec. 4.1, so that training starts from the pretrained body prior.

Actor objective. FB learning estimates discounted occupancies from replay transitions, a latent-conditioned discriminator keeps motion close to the demonstrations, and an auxiliary critic carries stability regularizers (Tirinzoni et al., 2025; Li et al., 2026). The actor maximizes the FB task value $F(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})^\top \boldsymbol{z}$ together with the discriminator critic Q_D and the auxiliary critic Q_A:

$$\mathcal{L}_\pi = -\mathbb{E}\big[F(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})^\top \boldsymbol{z} + \lambda_D Q_D(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z}) + \lambda_A Q_A(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})\big], \qquad \boldsymbol{a} = \pi(\boldsymbol{o}, \boldsymbol{z}). \tag{2}$$

The effective weights λ_D and λ_A are nominal coefficients multiplied by the detached mean absolute FB value, so regularization scales with the learned task-value estimates.

Imitation and auxiliary rewards. The imitation reward behind Q_D is the log-odds of the discriminator probability $D(\boldsymbol{s}, \boldsymbol{z})$,

$$r_D(\boldsymbol{s}, \boldsymbol{z}) = \log D(\boldsymbol{s}, \boldsymbol{z}) - \log(1 - D(\boldsymbol{s}, \boldsymbol{z})), \tag{3}$$

with probabilities clipped for numerical stability. Expert latents are backward embeddings of demonstration windows, and the discriminator is trained as a conditional classifier with gradient regularization. Auxiliary rewards penalize undesirable motion and physical violations, and Q_A is trained on their normalized return. Both scalar critics and the FB learner use target networks.

A.3 EMBODIMENT AND HAND CONTROL

The actor produces actions for 29 body joints, which are scaled into joint-position targets for low-level control. In actuated-hand embodiments, a separate finger controller supplies finger actuation. For the Inspire hands on the G1, a low-dimensional grasp interface replaces free exploration of all hand actuation: a per-hand scalar that the policy outputs next to its 29 body actions, from the same observation and prompt $\boldsymbol{z}$, is expanded to authored finger targets through software mimic coupling.

A.4 MOTOR-MODEL DIAGNOSTICS

Two ablations test the choices of Appendices A.1–A.2. Each variant is retrained on one GPU with a smaller training budget than the main model and evaluated with the tracking protocol of Sec. D.2. The main-model rows therefore serve as a reference rather than a matched control.

Interaction inputs. Table 6 removes the object state or the link geometry from the 105-dimensional encoding, which changes only what the controller observes while the verifier's state stays intact. Without link geometry, body error is the lowest of all variants, yet object error stays at the level of the body-only BFM-Zero in Table 1. Without object state, object error is lower than without geometry, but success still falls to about a third of the full model's.

Table 6: **Interaction observations.** Ablations remove the object state or the link geometry. ΔSR is relative to the full 105-dimensional encoding, which is the main model of Table 1 (without replanning).

Interaction encoding	Dim.	$E_h \downarrow$	$E_o \downarrow$	SR↑	ΔSR
Object state + link geometry	105	27.20	30.80	60	n/a
No object state	90	28.57	49.20	20	−40
No link geometry	15	25.05	60.83	12	−48

Residual capacity. Table 7 varies the residual branch from small MLPs to full width, with shared training data and body prior. The from-scratch model is trained on the same interaction data without the body prior.

Table 7: **Residual capacity.** MLP width×depth of the residual branch; the full-width residual mirrors the prior's module architectures and is the main model of Table 1 (without replanning). Parameter counts include all trainable modules. Errors and SR follow Table 1.

Model	Trainable parameters	$E_h \downarrow$	$E_o \downarrow$	SR↑
From scratch	843 M	60.96	78.21	2
Residual on the body prior				
MLP 256×2	14.1 M	36.19	50.81	28
MLP 512 × 2	30.5 M	31.51	42.58	30
MLP 1024 × 2	79.5 M	29.22	37.42	46
Full-width residual	852.0 M	27.20	30.80	60

B Reward Inference

Every prompt the motor model executes is computed through B, either from a reward scored over the bank or directly from reference states.

B.1 Uniform and reward-tilted aggregation

The FB factorization gives an approximate value for any integrable state reward:

$$Q_r^{\pi_{\boldsymbol{z}}}(\boldsymbol{s}, \boldsymbol{a}) = \int r(\boldsymbol{s}')\, M^{\pi_{\boldsymbol{z}}}(\mathrm{d}\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) \approx F(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})^\top \mathbb{E}_{\boldsymbol{s}' \sim \rho}[r(\boldsymbol{s}')B(\boldsymbol{s}')]. \tag{4}$$

The expectation is the prompt $\widetilde{\boldsymbol{z}}_r$ of Sec. 3.3, which we estimate on the reward-inference bank $\mathcal{B}$.

Bank construction. We build $\mathcal{B}$ once, after motor-model pretraining and before agent search. Its body-object states come from the source distribution used to learn the FB representation: replay from multi-task, multi-object loco-manipulation training. We add no separate rollouts: the replay already contains the reference-following rollouts collected during training, which cover interaction phases such as contact acquisition, transport, and release. For each state we cache $B(\boldsymbol{s}_n)$ and the reward-feature tensors that programs read (namespace `F` in Sec. C.1). The bank stays fixed during search. Target-task rollouts update the skill library $\mathcal{H}$ but add no states to $\mathcal{B}$, and only the bank diagnostics of Sec. B.3 change it.

Uniform aggregation. Averaging $r(\boldsymbol{s}_n)B(\boldsymbol{s}_n)$ over $\mathcal{B}$ estimates the expectation in Eq. 4. The projection proj then rescales the nonzero result to the sphere radius used in pretraining, block by block when the latent has separately normalized blocks. Scaling the whole reward by a positive constant therefore leaves the prompt unchanged, so only relative reward weights matter.

Table 8: **Drift-correction ablation.** Tracking protocol of Table 1; E_h and E_o in cm. The anchor is the quantity steered toward the reference, k_p is the correction gain, and the shaded row is the configuration used in Table 1.

Anchor	Addition	k_p	SR (%)↑	E_h ↓	E_o ↓
None	–	–	60	27.20	30.80
Anchor					
Root	–	2	68	19.56	25.27
Object	–	3	**72**	19.80	24.92
Root and object	–	2, 3	68	**18.24**	**24.10**
Correction gain k_p					
Root	–	1	62	19.76	25.40
Root	–	3	64	19.44	24.94
Object	–	2	64	20.57	25.77
Object	–	4	68	19.99	24.69
Object	–	5	70	19.77	24.84
Design choices					
Root	Update every 2 steps	2	66	19.55	25.23
Root	Kernel width 0.5 m/s	2	64	18.82	24.43
Object	Kernel width 0.5 m/s	3	68	19.57	25.05
Additions to the object anchor					
Object	Derivative term, $k_d = 2$	3	62	20.33	25.56
Object	Derivative term, $k_d = 5$	3	66	20.38	25.26
Object	Object-yaw feedback, gain 0.5	3	70	20.74	25.69
Object	Object-yaw feedback, gain 1.5	3	66	22.12	27.00
Object	Threshold gain, $\gamma = 1$	3	66	19.85	24.72
Object	Threshold gain, $\gamma = 3$	3	68	19.96	25.08

Reward-tilted aggregation. Reward tilting shifts the estimate toward high-reward bank states:

$$w_n = \frac{\exp(\beta r_n)}{\sum_{k=1}^{N} \exp(\beta r_k)}, \qquad \boldsymbol{z}_r = \mathrm{proj}\left(\sum_{n=1}^{N} w_n r_n B(\boldsymbol{s}_n)\right), \quad r_n = r(\boldsymbol{s}_n), \quad \beta \geq 0. \tag{5}$$

With the stage rewards $r_n = \hat{r}_{j,n,t}$, Eq. 5 gives the prompt $\boldsymbol{z}_{j,t}$ of Eq. 1 with $\alpha_{j,n,t} \propto w_n r_n$, because proj removes the overall scale. At $\beta = 0$ it reduces to uniform aggregation, and larger β concentrates the estimate on high-reward bank states. The tilt is therefore a biased heuristic. Reward scale and β jointly set its concentration, and a sharply concentrated weighting makes $\boldsymbol{z}_r$ depend on a few bank states.

B.2 Object-anchored drift correction

The actor observes body and object only in local frames (Sec. A.1), so it cannot see how far execution has drifted from the reference in the world frame. We supply this signal through reward inference, reusing the tracking checkpoint, reference, and bank without training or search. The world-frame position error of an anchor (the root, the object, or both) gives a proportional velocity correction with gain k_p. The correction is bounded, added to the anchor's reference velocity, and expressed in the live heading frame. A Gaussian reward scores each bank state's anchor velocity against this command. We center the reward over $\mathcal{B}$ and infer a correction latent, which is normalized separately, blended with the native reference latent $\boldsymbol{z}_t^{\mathrm{ref}}$, the projected mean of B over a short window of upcoming reference states, by a relative coefficient, and projected back onto the latent sphere. An optional threshold gain γ scales the gain to $k_p\,(1 + \gamma\,(\|\Delta \boldsymbol{p}_t\|/0.5)^2)$, raising it as the offset $\Delta \boldsymbol{p}_t$ approaches the 0.5 m termination distance.

Anchor, gain, and additions. Table 8 shows that every variant improves on tracking without correction. Each anchor has its own best gain, $k_p = 2$ for the root and $k_p = 3$ for the object, because moving the object requires moving the body first. Anchoring both root and object gives the lowest errors but a lower SR than the plain object anchor, which Table 1 therefore uses.

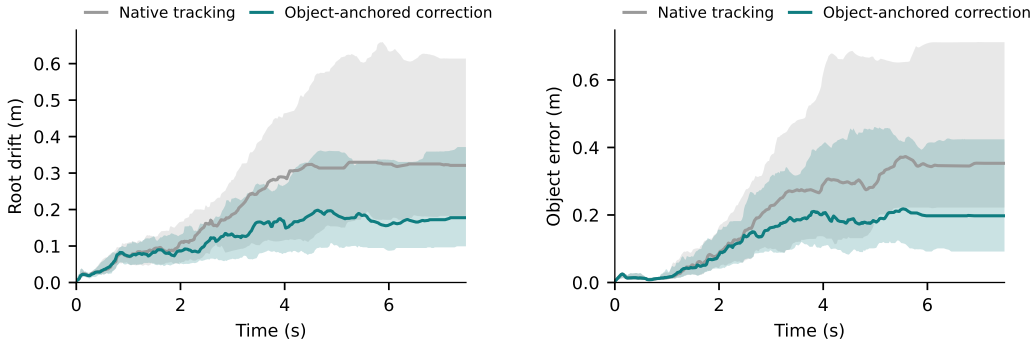


Figure 5: **World-frame drift over time.** Horizontal distance between the simulated and reference root (left) and object (right) for native tracking and the object-anchored drift correction, on the 50 largebox clips of Table 1. Lines show the median and bands the 25–75% range over all 50 motions; each motion is counted to the end of its clip, and after its clip ends its final value is held. Without correction the median root drift grows to 0.29 m at 4 s and 0.32 m by the end, against 0.16 m and 0.18 m with the correction, which levels off after about 4 s. The two curves coincide for the first second because the correction has a 0.10 m deadband.

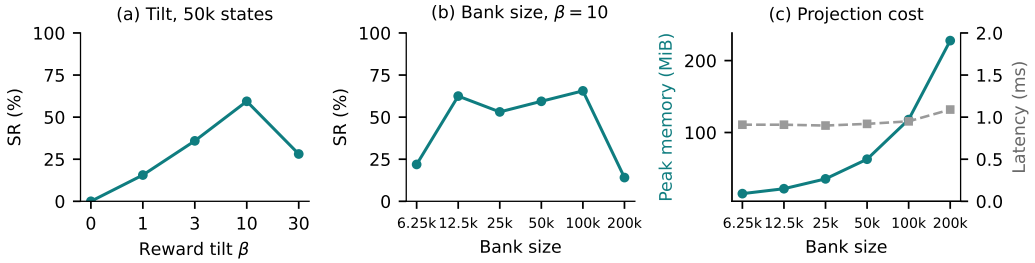


Figure 6: **Reward tilt, bank size, and cost on the feet-only task.** (a) SR against the reward tilt β with 50k states. (b) SR against bank size at $\beta = 10$. (c) Peak memory (solid) and projection latency (dashed) against bank size.

B.3 Bank size, reward tilt, and cost

The bank determines which behaviors a reward can reach and how much each projection costs. Bank studies therefore change only $\mathcal{B}$, keeping the controller, reward program, skill library, and test scenarios fixed.

We run these studies on the feet-only task, whose target behavior is rare in the replay. Every bank is therefore a random draw from a pool that over-samples states showing this behavior, and count comparisons draw nested banks of different sizes from it. Tilt comparisons share the bank, reward, and checkpoint, vary β with reward scale fixed, and run without agent search. Table 9 pairs success with N_{90}, the number of states that carry 90% of the weight, and with projection latency and peak memory. Latency covers repeated reward projection on identical hardware and excludes one-time bank construction and feature caching.

Where the weight goes. N_{90} explains both failure modes summarized in Sec. 4.3. Uniform weights spread over the whole bank, and $\beta = 30$ puts nearly all weight on three states. At $\beta = 10$ the weight moves with the context: at most frames it sits on about ten states, and during contact it briefly spreads over thousands. The 200k bank fails because its tilted weight concentrates on an order of magnitude fewer states than the 100k bank. Peak memory grows linearly with bank size (Fig. 6c).

Table 9: **Reward-inference bank diagnostics on the feet-only task.** SR is the task success rate of the fixed program; N_{90} is the number of bank states that carry 90% of the weight at a frame, averaged over the frames of a rollout and reported as the median over the 64 test scenarios. Latency is per reward projection; peak memory covers resident bank features and projection workspace.

β	State count	SR (%)↑	N_{90}	Projection latency (ms)↓	Peak memory (MiB)↓
Reward tilt					
0 (uniform)	50k	0.0	45,000	–	–
1	50k	15.6	28,674	–	–
3	50k	35.9	10,655	–	–
10	50k	**59.4**	2,077	0.92	63
30	50k	28.1	3	–	–
Bank size					
10	6.25k	21.9	293	0.91	15
10	12.5k	62.5	346	0.91	22
10	25k	53.1	1,441	0.90	36
10	100k	**65.6**	2,815	0.95	118
10	200k	14.1	170	1.09	228

Algorithm 1: Reward-program evolution at test time.

Input: task ℓ, scene c, verifier $\mathcal{C}$, motor model π with bank $\mathcal{B}$, skill library $\mathcal{H}$, round budget K.

1: The agent writes an initial program from ℓ, c, $\mathcal{C}$, and $\mathcal{H}$; evaluate it and make it the current best.
2: **for** $k = 1, \ldots, K$ **do**
3: Prompt the agent with ℓ, c, $\mathcal{C}$, $\mathcal{H}$, and the current best program with its rollout feedback.
4: Parse and validate the proposed programs; discard invalid candidates.
5: *Inner loop:* tune the constants $\boldsymbol{\theta}$ of each valid program with CMA-ES on the search scenarios.
6: Compare the tuned candidates with the current best on the same search scenarios.
7: Re-evaluate the top candidate and the current best on the confirmation scenarios; the candidate becomes the current best if its gain exceeds evaluation noise.
8: Summarize criterion- and stage-level feedback.
9: **if** the current best satisfies every criterion in $\mathcal{C}$ **then break**
10: **end for**
11: Add the current best program and its verifier results to $\mathcal{H}$.

Output: the current best program.

C Reward Programs and the Evolution Loop

This section makes the programs and the search loop of Algorithm 1 concrete.

C.1 Program semantics

A program $P = \big(\boldsymbol{\theta}, \{(r_j, g_j)\}_{j=1}^{J}\big)$ exposes each part of a strategy as a separately editable component (Table 10): tunable constants and, per stage, reward code, a completion condition, and optional local memory. Completion conditions make temporal structure explicit: a stage can persist until g_j holds, enforce a dwell time, time out, or return to an earlier stage.

Table 10: **Editable components of a reward program.** Each edit is judged by its executed outcome.

Component	Symbol	Role	Example edit
Reward code	r_j	Desired body-object interaction	Upward → horizontal object motion
Constants	$\boldsymbol{\theta}$	Scale and target of an objective	Target height, contact offset
Stages	j	Intermediate objectives	Split acquisition from transport
Completion	g_j	Completing or switching a stage	Require support before transport
Stage memory	–	Context captured at runtime	Displacement since stage entry

Bank features and live context. Program code reads bank features and live context from separate namespaces, so that a reward means the same thing over the bank as in the rollout that executes its latent. `F` holds bank features, each a tensor over all N bank states, so one call to `reward` returns

the stage rewards $\hat{r}_{j,n,t}$ of Eq. 1 for the whole bank at once. `C` holds live context: scalars read from the current frame, such as the object-target distance, and the unit direction `C["v_dir"]` from the object toward its target. The split is needed because a live measurement is not a property of the bank states. A current contact report, for instance, says nothing about contact forces in a bank that stores only geometry. `W` exposes the constants $\boldsymbol{\theta}$, namely relative reward weights and thresholds or kernel widths, and expressions can read only this declared interface.

A complete program. The selected program for `lift_rel` (place the box on a 0.65 m table and let go) has two stages and seven constants. In code, r_j is the function `reward` and g_j is `subgoal`:

```
name    : lift_carry_release_2stage
weights : [1.460, 0.440, 0.212, 0.203, 1.578, 0.310, 1.961]
           lift   drive  damp   arrive release still  settle

stage 0  # carry, and place
  def reward(F, C, W):
      high  =  W[0] * sigmoid((F["obj_z"] - 0.85)/0.05) * F["hand_contact"]
      drive =  W[1] * (F["obj_vel"] * C["v_dir"]).sum(-1)
      damp  = -W[2] * F["vel_norm"]
      return high + drive + damp
  def subgoal(C, W):
      return C["d_target"] < W[3]

stage 1  # release
  def reward(F, C, W):
      return ( W[4] * (1.0 - F["hand_contact"])
             - W[5] * F["vel_norm"]
             - W[6] * norm(F["root_vel_local"]) )
  def subgoal(C, W):
      return None
```

Stage 0 ends once the live object-target distance `C["d_target"]` falls below `W[3]`, and stage 1 then rewards release.

C.2 Agent context

Each round, the agent receives one plain-text prompt. Every task and run uses the same template: the fixed parts below are quoted from it, lightly abridged, and ⟨*placeholders*⟩ mark what each run fills in. The parts appear in the prompt in the order shown, and the full template and the final skill library are included as text files in the supplementary material.

Role and mechanism. The prompt first explains how a program becomes behavior, so that the agent writes rewards for the bank rather than for a live robot.

Prompt: role

You design a **staged** reward that teaches a **frozen** humanoid policy a manipulation skill. A skill is a sequence of stages. Each stage is a single behavior latent z for the frozen policy, obtained by scoring the policy's ∼50k-state behavior buffer with a reward and taking a softmax-weighted mean. Each stage has `reward(F, C, W)`, a tensor of shape $(N,)$ over the N buffer states (higher = pick it), and `transition(C, W)`, a bool; when true, advance to the next stage (last stage: null). `W` is a numeric weight vector; CMA-ES tunes `W` afterwards.

Feature vocabulary. Programs can read only these names. `F` is evaluated over the bank and `C` on the live frame (Sec. C.1).

Prompt: feature vocabulary (excerpt)

F[...] are per-buffer-state features (tensors of shape $(N,)$ unless noted), egocentric:

`F["obj_z"]`	object height (m)
`F["obj_vel"]` $(N, 3)$	object velocity, heading frame
`F["vel_norm"]`	object speed
`F["hand_contact"]`	$\exp(-5\, d_{\text{hand,object}})$; high when a hand is on the object
`F["foot_contact"]`	foot-object contact
`F["root_vel_local"]` $(N, 3)$	robot root velocity, heading frame
`F["robot_yaw_rate"]`	robot turning rate about the vertical (signed)

C[...] are live per-frame scalars (Python floats unless noted):

`C["d_target"]`	object-to-target distance (m)
`C["v_dir"]` $(3,)$	unit direction toward the target, heading frame
`C["face"]`	how much the robot already faces the target: $+1$ facing, 0 sideways, -1 behind
`C["obj_z"]`, `C["rest_z"]`	live and resting object height
`C["d_hand"]`	hand-to-object distance (m)
`C["progress"]`	episode progress, 0 to 1

⟨*further features, and a variant for tasks without a target position*⟩

Code rules. Programs that violate these rules are rejected by the validator before any rollout.

Prompt: code rules (excerpt)

The reward must reference `F[...]`. Keep 1–3 stages and few weights. `torch` and `math` are available.

- `reward(F,C,W)` must return a tensor of shape $(N,)$ built from `F[...]`, never a Python scalar. `C[...]` entries are Python floats: never call `torch.exp` or `torch.sigmoid` on them.
- `C["v_dir"]` is a length-3 tensor; use `C["v_dir"].unsqueeze(0)`.
- `transition(C,W)` returns a Python bool from `C` and `W`, e.g. `return C["d_target"] < W[3]`; it must not read `F[...]`. The last stage's transition is null.

Task and verifier. The task text is followed by the verifier, one row per criterion with its definition, so feedback can refer to criteria by name.

Prompt: task and criteria

TASK: ⟨*task request and scene description*⟩

What each tier measures. These are definitions, not instructions: deciding which reward term satisfies each one is your job, and the right term differs by task.

⟨*metric*⟩ ⟨*op*⟩ ⟨*threshold*⟩ ⟨*definition*⟩, for example:

`final_hand_d > 0.35` hand-to-object distance on the final frame, in metres; the only metric that separates "placed it and let go" from "carried it to the mark and froze there still holding it".

`final_d < 0.3` 3-D distance from the object's last frame to the target, in metres.

Search phase. One instruction per round sets how far a proposal may depart from the current best program.

Prompt: phase instruction (one per round)

Explore (early rounds): vary the decomposition the most: try 1 vs. 2 vs. 3 stages, different transition conditions, and different sub-goals per stage.
Refine (middle rounds): keep the winning decomposition, but vary the reward terms inside each stage: which features, which combinations, add, drop, or reweight a term.
Tune (late rounds): keep the winning structure almost fixed and propose small variations; CMA-ES fine-tunes the numbers.

Reference programs with per-criterion evidence. Each carried program is listed with how many of the 192 confirmation rollouts pass each criterion of $\mathcal{C}$, followed by the full source of the three strongest programs:

Prompt: reference programs

Reference structures so far, each showing how far up the criteria it climbed and where it is stuck. Beat the leader: graft the mechanism that got the furthest-climbing one past its criteria, then fix its stuck criterion.
– ⟨*program*⟩: tiers=⟨*m*⟩/⟨*n*⟩ stuck@⟨*criterion*⟩ (⟨*k*⟩/192 envs clear it, median ⟨*value*⟩)
⟨*criterion*⟩ ⟨*k*⟩/192 PASS/FAIL ⟨*criterion*⟩ ⟨*k*⟩/192 PASS/FAIL …
Full source of the strongest references (graft from these, do not re-derive): ⟨*three programs*⟩

Which reference leads on which criterion. Because z is inferred per stage, a stage that earns a criterion can be transplanted, and the prompt says where each criterion is earned:

Prompt: which reference leads

⟨*program A*⟩ leads on: ⟨*criterion*⟩=⟨*pass rate*⟩, … ⟨*program B*⟩ leads on: ⟨*criterion*⟩=⟨*pass rate*⟩, …
No single reference leads everywhere. A stage that achieves a tier can be carried over as a stage: take the stage that earns one tier from one reference and the stage that earns another from a second reference, and write the transition between them. At least one of your proposals **must** be such a merge, quoting the two sources.

Rollout feedback. The failing criterion is reported with its margin and spread, followed by a stage-resolved trace and the problems read off it:

Prompt: rollout feedback

round ⟨*r*⟩: best '⟨*program*⟩' passes ⟨*m*⟩/⟨*n*⟩ tiers (⟨*J*⟩ stages). **Stuck on tier** '⟨*criterion*⟩' (⟨*k*⟩ of 192 envs clear it). The median env is ⟨*value*⟩, which ⟨*clears | misses*⟩ the bar by ⟨*margin*⟩, so this is a ⟨*tail/variance | low-mean*⟩ failure. … Find the stage that was supposed to move '⟨*criterion*⟩' and did not, and change only that stage.
Stage trace (median over envs):
stage ⟨*j*⟩: ⟨*k*⟩/192 envs, dwell ⟨*s*⟩ s, obj_z ⟨*a*⟩→⟨*b*⟩, d_target ⟨*a*⟩→⟨*b*⟩, grip ⟨*a*⟩→⟨*b*⟩
Stage problems: ⟨*e.g. stage 0 lasted only 0.66 s of 14.0 s: its transition fires almost immediately*⟩

Output format. The agent returns a JSON list of programs (Sec. C.1). Each stage's `transition` field holds g_j, and `w_init`, `w_lo`, and `w_hi` give initial values and bounds for $\boldsymbol{\theta}$:

Prompt: request and output format

Propose ⟨K⟩ diverse staged structures as the JSON list. Return strict JSON, a list of structures, each:
`{"name": ..., "w_names": [...], "w_init": [...], "w_lo": [...], "w_hi": [...], "stages": [{"reward": "def reward(F,C,W): ...", "transition": "def transition(C,W): ..."}, ...]}`
The last stage's transition is null. Make the structures diverse (different numbers of stages, terms, and transitions).

C.3 Selection and the skill library

Accepting an edit on the scenarios that suggested it would reward lucky edits, so search and confirmation use separate scenario grids. Candidates are developed on a 16-scenario search grid, where the current best program and each candidate share initial conditions. A 64-scenario grid then re-evaluates the pair with three repeats per scenario, and the candidate replaces the current best only if its improvement exceeds the run-to-run evaluation noise. The final evaluation re-runs the selected program on a different set of 64 scenarios drawn from the same distribution. Both grids spread the box bearing evenly over $\pm 12°$ and its distance over ± 0.15 m around the nominal start, and every method is evaluated on the identical 64 scenarios.

The skill library $\mathcal{H}$ is a text document built from completed tasks. Each entry records the scene, the task text the agent saw, the verifier with the pass rate of each criterion, the measured success, and the selected program with its tuned constants. A closing section states the lessons: reward-design regularities shared across the stored programs. For example, every multi-stage program ends with a release stage that pays for $1-$contact and for object and robot stillness, and a lift term is always a product with hand contact, never a bare height term, which would select airborne states. The whole library, eight tasks in about 28 KB, is placed in the agent's prompt every round.

C.4 Search cost accounting

Search cost counts every executed candidate, including rejected ones. CMA-ES with G generations of λ_{ES} candidates on S scenarios costs $\lambda_{\mathrm{ES}} GS$ rollouts per program, before confirmation. With $\lambda_{\mathrm{ES}} = 4$, $G = 3$, and $S = 16$, this is 192 rollouts per program, and each confirmed candidate adds 64 scenarios $\times$ 3 repeats $=$ 192 rollouts. Every rollout runs the full episode (700 control steps at 50 Hz), so simulator cost is proportional to the rollout count. A five-round search on one task family executes 6.7–14.2 M environment steps, 9.5 M on average over the eight families, or about 1.9 M per round. Table 2 reports language-model tokens. The simulator pauses while the agent revises a program. The agent is DeepSeek-V4-Flash, queried through its API. A three-round run makes four to five calls and uses 0.13–0.18 M tokens. Each round spends 1–2 min in the language model and 16–34 min in simulation on one GPU, so simulation dominates wall-clock time.

The goal-conditioned experiments of Sec. 4.3 set budget $K = 4$, so each search runs five rounds including the initial program (Table 4). Ablations that skip tuning or use a single scenario spend the saved rollouts on more proposals, which keeps their simulation budget equal to that of the full system (Sec. D.5).

D Experimental Protocols and Additional Results

This section gives the protocol behind each experiment of Sec. 4, in the same order, after the scoring and replication rules they share.

D.1 Verifier scoring and replication

The verifier $\mathcal{C}$ is fixed apart from the candidate programs and scores every study independently of the reward being optimized. It records task outcomes, constraint satisfaction, and stage-resolved progress. Only $\mathcal{C}$ decides success, while program termination and reward value serve as diagnostics. Proximity, contact, lift, and sustained support are separate criteria.

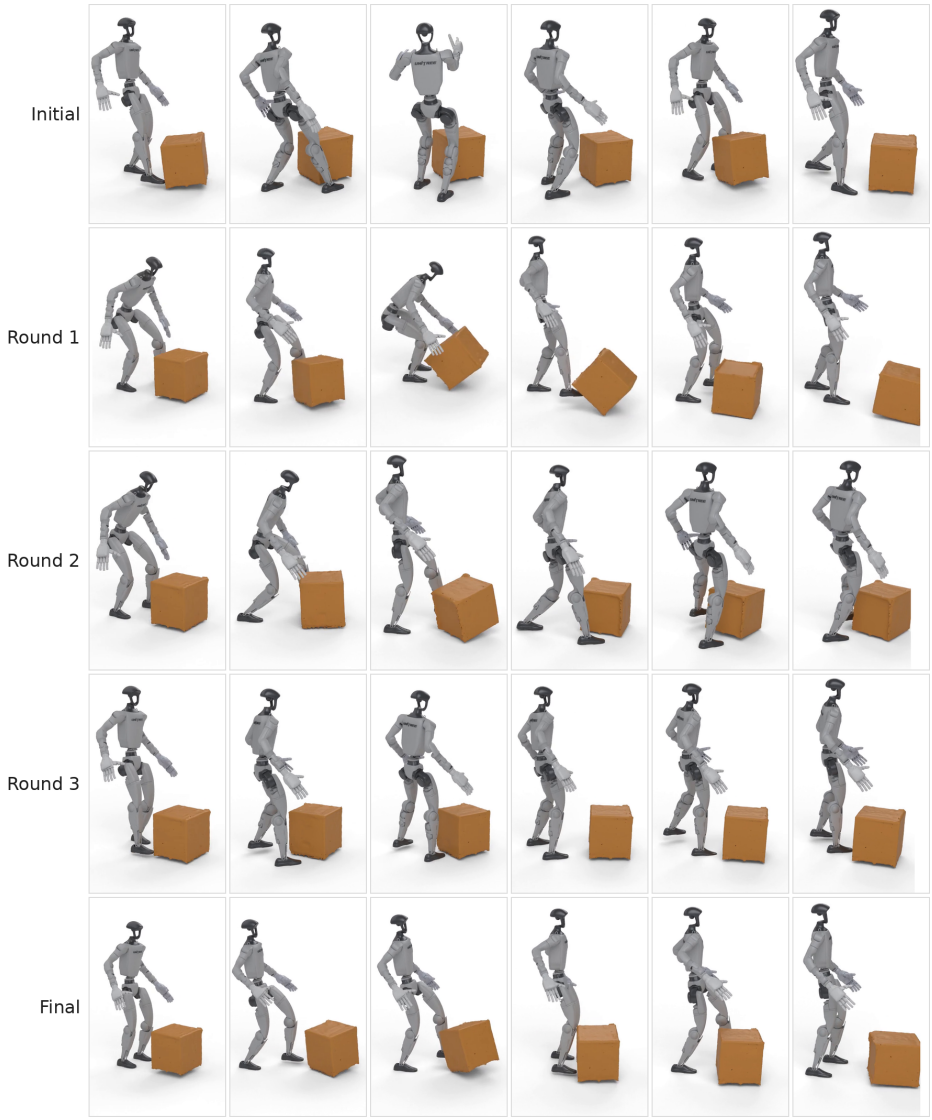


Figure 7: **Reward-program evolution on the kick task.** Each row shows the current best program after the initial proposal, rounds 1–3, and the final round (top to bottom), the kick programs evaluated in Table 4; frames run left to right. Revised objectives change the contact strategy across rounds.

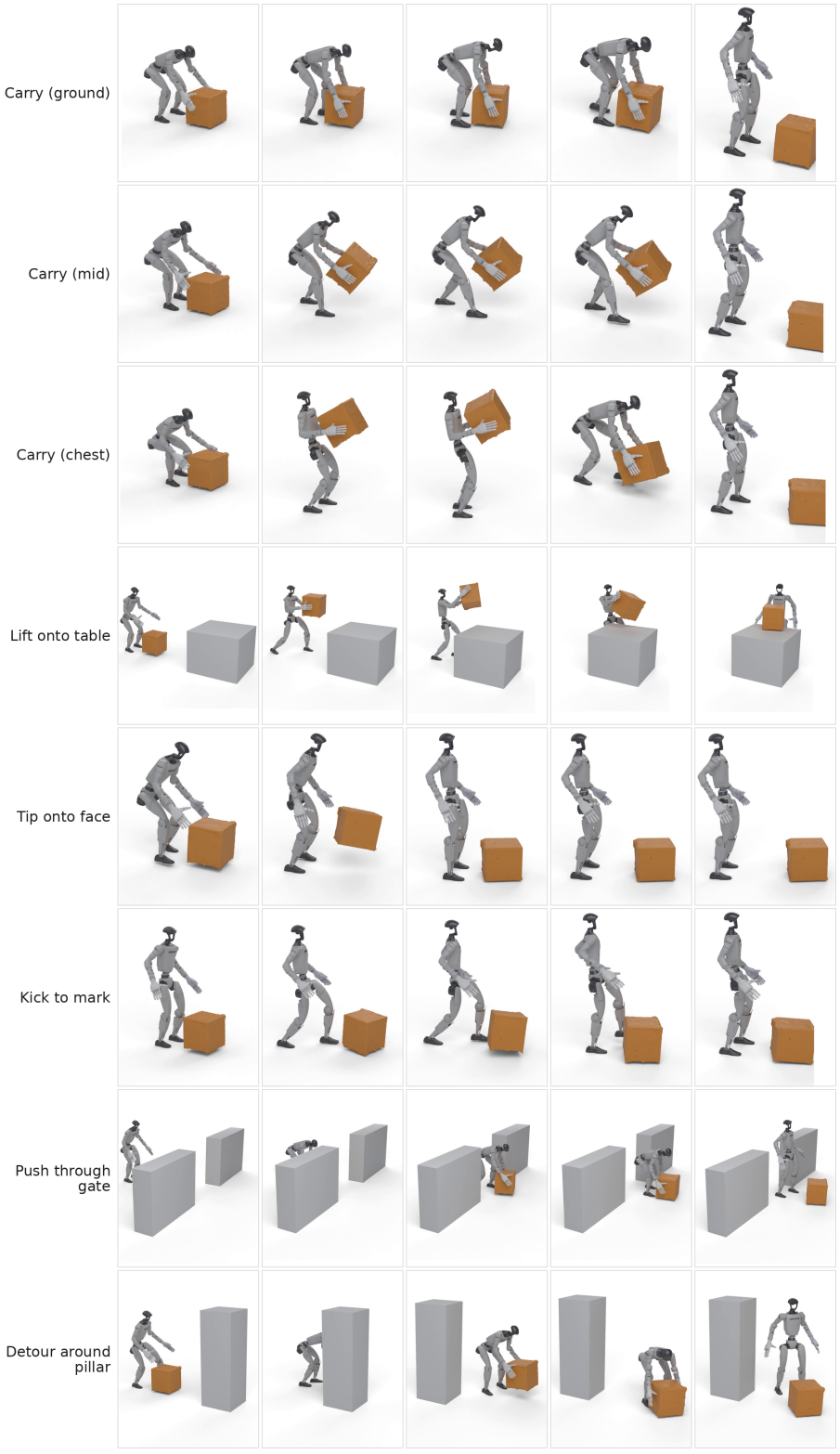


Figure 8: **The eight basic skills.** Final evolved program of each task family, frames left to right. Whole-body motion links object acquisition, transport, and placement across carrying, lifting, tipping, kicking, and obstacle-aware pushing.

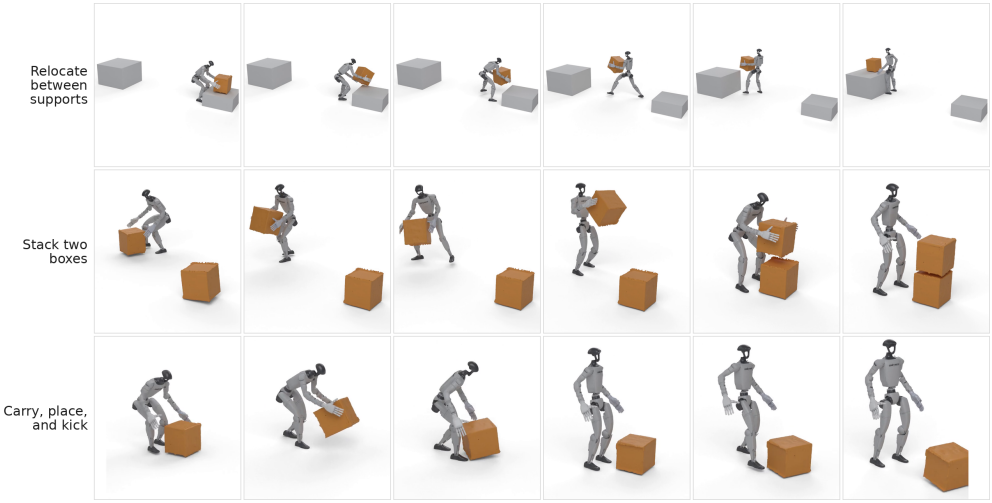


Figure 9: **Composite tasks.** The three composite tasks of Table 5: relocating a box between supports, stacking one box on another, and carrying, placing, then kicking a box. Frames run left to right.

Success and earned tiers. Success requires every criterion in $\mathcal{C}$ to hold in the same rollout, and earned tiers measure partial progress. Let $\mathcal{P}_k$ be the criteria a candidate passes on task k out of the $|\mathcal{C}_k|$ criteria of that task. The earned fraction is

$$E_k = |\mathcal{P}_k| \,/\, |\mathcal{C}_k|. \tag{6}$$

Some criteria, such as staying upright, are already met by doing nothing. E_k counts them like any other criterion, so we report an inaction reference that stands still (Table 2). The reference and all thresholds are fixed before any comparison. Aggregate success macro-averages task families after averaging test scenarios within each family, and aggregate earned tiers average E_k over task families.

Replication. Each variant is searched once per task family. Variants share task instances and evaluation scenarios. Search and test scenarios are disjoint, while confirmation re-evaluates candidates on the test scenarios with separate repeats (Sec. C.3).

D.2 Reference tracking

Every method shares the reference, initialization, horizon, and termination protocol, and E_h and E_o are computed over executed frames against the time-aligned reference. An episode terminates when the root drops below $0.15\,\mathrm{m}$, the mean object-surface error exceeds $0.5\,\mathrm{m}$, or an interaction-consistency check fails. SR counts every start, so a trajectory that terminates early lowers SR even when its errors are small.

ULTRA is trained on the same training clips and split as our motor model. BFM-Zero and our motor model each receive the native reference prompt $\boldsymbol{z}_t^{\mathrm{ref}}$, the projected mean of B over a short window of upcoming reference states, and the last row adds the drift correction of Sec. B.2 to the same checkpoint.

D.3 Goal-conditioned task families

Table 12 lists the eight families with the key constraint of each verifier, and Table 13 reports success per family. Tasks with intermediate contacts or scene constraints are scored on the full verifier instead of terminal position error, so a missed contact mode stays visible.

Rare behaviors need focused reward inference. How the prompt of Eq. 1 weighs bank states matters most for rare target behaviors such as kicking. Uniform averaging washes out the few relevant states, whereas sharp focus rests the prompt on a handful of them, so success peaks at a

Table 11: **Supplementary tracking metrics** on the clips of Table 1. Every metric is averaged over three evaluation runs of the 50 clips, with rates over clips rounded to integers, and the rows below the first block come from recordings in which every clip runs to its end without early termination. *Orientation error*: object orientation error, averaged over the whole clip. *Contact retention*: fraction of reference-contact frames in which the simulated hand is also in contact, with contact meaning a hand vertex within 3 cm of the object surface. *Slip*: wrist speed in the object frame during contact, in simulation and on the reference. *Penetration*, *object below ground*: frames deeper than 2 cm. *Falls*: pelvis below 0.35 m or tilted beyond 60°.

	ULTRA	BFM-Zero	InterEvolve	+ Evolving
SR (%) ↑	66	8	60	**72**
E_h (cm) ↓	**15.68**	28.36	27.20	19.80
E_o (cm) ↓	**21.45**	62.33	30.80	24.92
Orientation error (°) ↓	37.8	47.6	23.4	26.2
Contact retention (%) ↑	92	2	89	89
Slip, sim / ref. (cm/s)	27.7 / 19.4	20.0 / 26.4	20.0 / 20.8	19.6 / 20.5
Penetration (% frames) ↓	0.0	0.0	0.0	0.0
Object below ground (% frames) ↓	0.02	0.0	0.0	0.0
Joint-limit violations (% frames) ↓	0.60	0.03	0.00	0.05
Falls (% clips) ↓	34	0	0	0

Table 12: **Task families for program search.** The verifier scores outcomes and constraints independently of the agent-written reward.

Task	Interaction	Key constraint
Push to mark	Push object to a ground target	Keep it on the floor, stop near target
Carry at mid height	Acquire, raise, and carry	Hold a waist-relative height band
Carry at chest height	Acquire, raise, and carry	Hold a body-relative height
Tip onto a new face	Turn it over about a horizontal axis	Large tilt, left standing on the new face
Push through a gate	Push between two immovable walls	Hold a straight heading through the gap
Push around an obstacle	Push past a pillar in the way	Deviate, then recover the line
Kicking to mark	Move object with repeated foot contacts	No hand contact
Lift to support	Raise onto an elevated surface	Final height and placement

moderate tilt β, and bank size shows the same trade-off (Table 9, Fig. 6). Reward inference therefore needs *enough relevant states* and a *moderate focus*.

Table 13: **Success per task family (%).** CMA denotes CMA-ES. The macro-average weights families equally.

Task family	Human + CMA	Initial agent + CMA	InterEvolve
Push to mark	96.9	**100.0**	**100.0**
Carry at mid height	0.0	40.6	**100.0**
Carry at chest height	0.0	0.0	**100.0**
Tip onto a new face	0.0	0.0	**95.3**
Push through a gate	0.0	25.0	**73.4**
Push around an obstacle	0.0	4.7	**71.9**
Kicking to mark	17.2	37.5	**59.4**
Lift to support	29.7	68.8	**92.2**
Macro-average	18.0	34.6	**86.5**

D.4 FIXED-PROGRAM BASELINES

Human and initial agent programs are each evaluated before and after CMA-ES with the same feature access, parameter bounds, and rollout budget per program. Because reward-tilted inference couples reward scale with β (Sec. B.1), we hold β fixed when comparing reward formulations.

D.5 SEARCH-DESIGN ABLATIONS

Table 3 removes one of the five search components below at a time. Every variant runs on the same task families, controller, and verifier, and is scored on the program its selection rule returns.

CMA-ES tuning. In the full system, the inner loop tunes the constants $\boldsymbol{\theta}$ of every valid proposal with CMA-ES before the proposal is compared with the current best program (Sec. C.4). Without it, each proposal runs with the initial constants the agent wrote. This isolates how much of the gain comes from calibrating a given structure.

Targeted edits. In the full system, the agent sees the current best program and its feedback and is instructed to repair it: to change the stage or term that caused the failing criterion, or to merge stages from programs that lead on different criteria (Sec. C.2). The full-rewrite variant keeps the same context, feedback, acceptance rule, and budget, but asks for new programs written from scratch each round. This isolates the value of editing a working program over regenerating it.

Multi-scenario evaluation. In the full system, every candidate is evaluated on a grid of multiple search scenarios that vary the object's bearing and distance (Sec. C.3). The single-scenario variant evaluates each candidate on one scenario and spends the saved budget on more candidates. This isolates whether judging a candidate across varied starts helps select edits that generalize rather than edits that happen to work once.

Scene context. In the full system, the prompt describes the scene, such as object placement, supports, and obstacles, next to the task text. The variant gives the agent only the task text. The controller still observes the full scene, so this ablation affects planning but leaves feedback control intact.

Multi-stage programs. In the full system, a program may split the task into up to three stages with their own rewards and completion conditions. The single-stage variant must express the whole task with one reward from the same feature vocabulary under the same budget. This isolates the value of temporal composition.

D.6 SKILL-LIBRARY REUSE

Library experiments fix everything except the library content, including the controller, the bank, and the target-task budget. The source library is frozen before target search and excludes the target task instances, so target rollouts inform only within-task revision. Table 5 varies library content on composite tasks.

Composite tasks. Table 14 describes the three composite tasks of Table 5. Each combines several contact modes that appear separately among the eight task families of Table 12, in one episode and from one initial reset: carrying and placing on a support, lifting and lowering onto another object, and pushing followed by kicking. As in the other families, the verifier scores the outcome and its constraints independently of the agent-written reward, and success requires every criterion in the same rollout. None of the three tasks, and no program written for them, is in any source library.

Whether programs or lessons carry over. Table 15 extends the carry, place, and kick task of Table 5 with more library variants, all scored on the same 64 test scenarios. Lessons are the library's shared design notes without its programs summarized by human. Run as stored, without search, only the kicking program solves any scenario (6.3%), so the gain comes from search adapting stored programs rather than from copying one. After three search rounds, the full library reaches 60.9%, programs without lessons 48.4%, lessons without programs 21.9%, and neither 12.5%. Among program subsets, the three carrying programs reach 23.4% and every subset that contains the kicking program reaches 43.8–53.1%, all below the full library.

Table 14: **Composite tasks for library reuse.** Distances are measured from the box's starting position.

Task	Scene and interaction	Verifier criteria
Relocate between supports	The box starts on a 0.3 m support in front of the robot. The robot lifts it, turns around, carries it 2.9 m to a 0.6 m support behind it, sets it down, and lets go.	Stays upright; holds the box for a large part of the episode; lifts it above the destination height; ends within 0.3 m of the mark on the support; hands clear of the box at the end.
Stack two boxes	Box A is on the floor 0.5 m ahead; box B, of the same size and free to move, is 1.9 m to the left. The robot lifts A, carries it above B, lowers it onto B's top face, and lets go.	Stays upright; lifts A clear of B's top; sustained two-handed contact; A stays level; ends within 0.25 m of B's top (the target follows B if B is pushed); hands clear; A at rest.
Carry, place, and kick	The box is on the floor 0.5 m ahead. The robot pushes it about 1.0 m along the ground with its hands, lets go, and kicks it with its feet a further 1.5 m to a mark 3.0 m ahead.	Stays upright; box stays low (no lift); sustained hand contact during the push; at least one foot contact; box travels at least 1.0 m after the last hand contact; hands clear; ends within 0.3 m of the mark.

Table 15: **Library ablations on carry, place, and kick**, extending Table 5. Every row is scored on the same 64 test scenarios, none of which any search used; SR is the percentage of scenarios solved, where a scenario is solved when one of three rollouts meets every criterion. Left: each stored program run as written, without search. Right: three search rounds from each source library; the shaded row is the full library of Table 5.

Stored program, no search	SR (%)
Kicking to mark	6.3
Push to mark	0.0
Carry at mid height	0.0
Carry at chest height	0.0
Push around an obstacle	0.0
Push through a gate	0.0
Lift to support	0.0
Tip onto a new face	0.0

Source library	Programs	Lessons	SR (%)
Full	8	✓	**60.9**
Programs only	8	–	48.4
Lessons only	0	✓	21.9
None	0	–	12.5
Tip, lift, chest carry, kick	4	✓	50.0
Lift, kick	2	✓	53.1
Kick only	1	✓	43.8
Carry only (three carrying)	3	–	23.4

D.7 Coverage of the pretraining references

Fig. 10 compares evolved behaviors with the data the FB model was pretrained on. Carrying at ground height and pushing stay inside the reference cloud, whereas lifting onto a table, carrying at chest height, kicking, tipping, and carrying, placing, then kicking form their own clusters, so the frozen controller produces motions that no training reference contains. **Motions.** The references are the 729 OMOMO largebox training clips; the evolved side is 11 skills with 16 rollouts each, cut to the interaction window (from 0.5 s before the object first moves to 0.5 s after it last moves). **Descriptor.** Only the body is used, so object size and scene layout cannot separate the two sets. For each frame, the 29 link positions are taken relative to the pelvis and rotated into the robot's own heading, so walking direction and turning do not count as different motions. Each coordinate is resampled to 128 steps and summarized by its first 16 DCT coefficients, a low-frequency description of how the posture changes over the window. **Embedding.** Features are standardized with the reference statistics, reduced by PCA fit on the references (32 components, 91% of variance), and embedded jointly with UMAP. **Distance check.** Independently of the 2D layout, we measure each rollout's distance to its nearest reference clip in the PCA space and rank it against the distance of each reference clip to its nearest reference from a different recording. The median rollout ranks at the 76th percentile; lifting onto a table, carrying at chest height, and carrying, placing, then kicking rank at the 98th, and carrying at ground height and pushing around a pillar at the 18th and 9th.

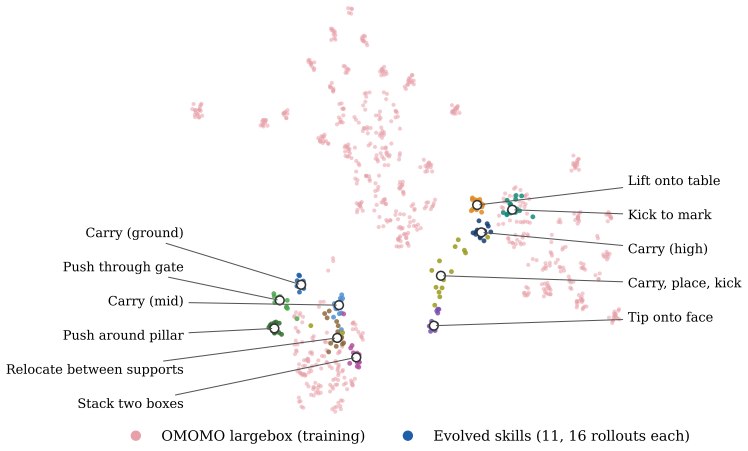


Figure 10: **Evolved skills versus pretraining references.** Each point is one motion: an OMOMO largebox training clip (pink) or one rollout of an evolved skill (colored, 16 per skill; circles mark skill centers). A motion is described by its body posture relative to the pelvis and heading, summarized over time, and embedded with UMAP.

D.8 Transfer, composition, and physical deployment

Perception. We deploy on a Unitree G1 whose policy commands the same 29 joints as in simulation at 50 Hz. An RGB-D camera on the torso streams color and depth to an off-board GPU workstation, which runs perception, state estimation, and the policy. FastSAM (Zhao et al., 2023) and CLIP (Radford et al., 2021) find the box in the image, and FoundationPose (Wen et al., 2024) registers a mesh of the box on this mask and tracks its 6-D pose frame by frame, discarding poses that disagree with the measured depth. Each pose is time-stamped at capture and mapped into the robot frame with the joint angles and leg odometry of that moment. A filter fuses these poses with box fits from a LiDAR on the torso and with floor and contact constraints, and uses the camera only while the box is free, since the arms occlude it during manipulation. From the filtered pose, the controller computes the same object features it observes in simulation.

Execution. The robot does not search. A program is evolved and verified in simulation, and the latent prompts it produces during a simulated rollout are replayed on the robot, one per control step, while the frozen controller closes the loop on proprioception and the estimated object pose. Execution starts once the object estimate is confident, and it pauses while the estimate is invalid.

E Limitations

Program search is bounded by the motor repertoire, the reward-inference bank, and the available measurements. It cannot elicit behavior the controller never learned. Each search also costs GPU-hours of simulation and LLM latency, which rules out real-time replanning during physical execution. Letting accumulated experience also update the controller is a next step.