AGENTIC ROBOT CREATION THROUGH GENERATION, VERIFICATION, AND REVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

Can an AI agent turn high-level user intent into a functioning robot and teach it how to behave? We study this question from two complementary aspects: embodiment and behavior creation. For embodiment creation, we investigate how well current multimodal models capture mechanical requirements, and introduce computational metrics for evaluating generated designs before fabrication. For behavior creation, we present **RoboForge**, an agentic framework that turns task specifications into executable policies through motion generation, policy learning, and execution-grounded revision. Given a robot embodiment and task, the agent generates a reference motion, configures the learning objective, and trains a policy. Crucially, motion remains in the loop: rollout feedback can revise the reference motion, reward, or both. We evaluate a generated embodiment across multiple MLLMs and behavior creation across four diverse robots against reward-refinement and one-shot motion baselines. Our computational metrics expose recurring mechanical failures in generated designs, while behavior experiments show that execution-grounded refinement improves task-level behavior and execution quality. Finally, we fabricate an agent-generated Gorilla and directly deploy learned behaviors on the physical robot.

1 INTRODUCTION

"Design is not just what it looks like and feels like; design is how it works." — Steve Jobs

Small expressive robots such as Microduck and recent character robots (Grandia et al., 2025; Müller et al., 2026) suggest a future in which people can personalize not only how a robot looks, but also how it moves and behaves. Yet creating such a robot today still demands substantial expertise. A high-level concept must be translated into a mechanical structure, suitable joints and actuators, a valid simulation model, desired motions, and ultimately a control policy capable of executing them. These stages typically require mechanical designers, animators, and control engineers to iterate together. With increasingly capable multimodal foundation models, we ask: **can robot creation become an agentic process, where users describe what they want and agents carry out much of the engineering required to make it physically realizable?**

Recent work has begun to automate individual stages of this process. Foundation models have been used to generate robot morphologies and articulated designs (Ringel et al., 2025; Sontakke et al., 2026; Wang et al., 2025). In parallel, LLMs can synthesize and iteratively refine rewards for reinforcement learning (Ma et al., 2024b; Lee et al., 2026), while MimicAgent generates reference motions for downstream policy learning (Nayak et al., 2026). Together, these advances suggest that much of the robot-creation pipeline may become amenable to agentic automation. Connecting these pieces, however, reveals a fundamental gap: **a robot that looks plausible is not necessarily buildable, and a motion that looks plausible is not necessarily executable**.

We study this gap from two aspects of robot creation. For **embodiment creation**, rather than introducing another morphology generator, we ask how well current multimodal large language models (MLLMs) understand the mechanical requirements that matter before fabrication. A visually convincing CAD model may still contain disconnected structures, insufficient actuator clearance, self-collisions, or unsuitable contact geometry. We therefore identify a set of mechanically relevant requirements and introduce multi-faceted computational metrics to quantify them before fabrica-

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.

tion. Using these metrics, we systematically compare different MLLMs, characterize the mechanical knowledge they already possess, and expose recurring failure modes in their generated designs. Importantly, we have fabricated an agent-generated robot *Gorilla* and demonstrate that designs passing these checks can be assembled and operated as physical systems.

For **behavior creation**, physical feasibility is harder to assess from generation alone. A motion may appear natural but remain incompatible with a particular embodiment because of its timing, pose, contact sequence or joint excursion. Such incompatibilities often become apparent only when a policy attempts to execute the motion. This observation motivates **RoboForge**, an agentic framework for turning desired behaviors into executable robot policies through **generation, physical execution, and revision**. Given a robot embodiment and a task description, the agent generates a reference motion and a learning objective, trains a policy, and inspects the resulting rollouts. Execution therefore serves as a physical verification signal rather than merely the endpoint of learning. When the policy fails, the agent reasons about whether the failure stems from *what behavior is being requested*—such as motion timing, target pose, contact sequence, or trajectory—or from *how the behavior is being learned*—such as reward terms and their relative weights. It then revises the corresponding component and retrains the policy. This closes a motion–policy feedback loop in which learning outcomes expose deficiencies in generated motions, while generated motions continue to provide structured guidance for learning.

We evaluate behavior creation across four diverse embodiments, including existing platforms such as Unitree Go2 and Microduck as well as agent-generated robots, and compare RoboForge with representative reward-refinement and one-shot motion-generation approaches. RoboForge leads to policies that are better aligned with the intended task, as evidenced by both user preference studies and complementary quantitative metrics. Besides, we analyze the intermediate failures and revisions throughout the process to understand how execution feedback drives improvement. We further deploy the resulting policies on the physical robot we fabricate, demonstrating behaviors that transfer from high-level user intent to real-world execution.

2 Related Work

Automated Robot Generation. Automating robot design has progressed from morphology search and modular optimization toward agent-driven generation. Prior methods translate high-level specifications into robot morphologies through language-guided search or generative design (Ma et al., 2024a; Ringel et al., 2025; Song et al., 2025; Qiao et al., 2025). Recent methods further generate articulated robot and object models from text, images, or their combinations (Chen et al., 2026; Sontakke et al., 2026; Wang et al., 2025), while morphology–control co-design couples embodiment optimization with downstream control performance (Qiu et al., 2024; Fang et al., 2025).

Policy Learning Guided by Reward and Motion. Multimodal large language models (MLLMs) are increasingly used to reduce manual specification in robot learning. One line of work synthesizes reward functions from textual task descriptions (Kwon et al., 2023; Yu et al., 2023; Xie et al., 2024; Ma et al., 2024b). Subsequent methods refine rewards using human feedback, training statistics, or execution feedback (Hazra et al., 2025; Li et al., 2024; Sun et al., 2025; Gao et al., 2026). A complementary line uses reference motion to specify desired behavior. Classical motion-imitation and motion-prior methods enable complex locomotion and whole-body control (Peng et al., 2018; 2021; Escontrela et al., 2022; Peng et al., 2022), and recent systems scale this paradigm to large motion datasets and expressive humanoid skills (Luo et al., 2023; Fu et al., 2024; He et al., 2024; Liao et al., 2026). More recently, MimicAgent uses MLLMs to generate task-specific reference motions for downstream RL (Nayak et al., 2026), but without policy feedback.

Existing approaches typically improve either the learning objective or the reference behavior in isolation: reward-generation methods refine *how* a policy is optimized, while motion-generation methods refine *what* behavior the policy is asked to reproduce. In contrast, our framework uses policy rollouts to diagnose whether failure originates from the reference motion, the reward specification, or both, and revises the corresponding component accordingly. This creates a closed motion–policy loop in which execution feedback jointly shapes the behavioral target and its optimization guidance.

3 EMBODIMENT CREATION

Given a reference image and a set of task descriptions, we use GPT-6 OpenAI (2026) to generate a robot embodiment. The reference image specifies the desired coarse morphology, while the task descriptions provide functional cues about the motions the robot should support. We additionally provide CAD assets and specifications for candidate actuators, together with official assembly examples. In practice, we find that such assembly examples are important: they help the model infer how motors, links, fasteners, and supporting structures should be connected, rather than merely placing components in visually plausible configurations. The generated embodiment $\mathcal{D}$ contains three consistent representations: (1) editable mechanical parts in STEP format for inspection and assembly, (2) an STL representation for fabrication, and (3) a URDF model for simulation and control.

3.1 PRE-FABRICATION FEASIBILITY METRICS

Fabricating every MLLM-generated design is prohibitively expensive and provides feedback only after substantial time and material have been invested. We therefore introduce a set of computational checks that evaluate whether a generated embodiment is likely to be manufacturable, assemblable, and capable of supporting the intended behaviors before fabrication.

We group the checks into three categories according to the type of evidence they require:

1) Design and Assembly checks whether parts can be manufactured and fitted together. It verifies valid single-solid CAD parts of positive volume, watertight meshes positioned on the print bed, and matching volumes in source and exported models. It checks whether each part fits the printer's build volume in its printing orientation. At mating interfaces, it examines mounting holes, tool openings, and support material; worst-case tolerance checks verify that clearance remains under dimensional variation. Assembly checks also examine insertion and tool-access paths, all fastener interfaces and traceable purchased-part specifications, and unwanted intersections in the complete assembly, including fasteners and cable envelopes.

2) Mechanical and Actuation Feasibility assess mechanical margins under the frozen material, load, and routing conventions. Local structural screening compares specified sections under recorded loads with an assumed allowable stress; load-path checks cover load-bearing parts and connections under defined load cases. Mass checks reconcile material estimates, link masses, and CAD and simulation totals. Fastener length chains compare selected screws with measured grip and nominal interface depths. Trajectory-limit margins check actual joint angles against existing limits; cable checks examine full-range clearance, bending, and twisting of the fixed routing candidate. Structural outcomes are conditional calculations, not measured strength; routing failures concern this candidate alone.

3) Motion and Task Execution evaluate whether the complete embodiment can realize the behaviors specified by the user. For each requested task, we construct a task-relevant reference motion by the agent and evaluate the robot along the corresponding trajectory. Checks include consistency of the simulated joint and inertial model, collision-free motion in the original CAD geometry, tracking error in simulation, actuator torque and saturation, joint-limit violations, and clearance margins under tracking error and fabrication tolerance.

Appendix I.2 provides each check's criteria and implementation details.

3.2 EVALUATION

We evaluate one gorilla embodiment generated by each of GPT-6 Astra, GPT-5.6 Sol, GPT-5.6 Terra and GPT-5.5. All models receive the same morphology reference, task descriptions, actuator candidates, and assembly examples. We evaluate each design using the proposed pre-fabrication metrics and the same two task motions: *airborne forward swinging* and *raised-hand waving*.

This comparison allows us to study not only whether a model produces a visually plausible robot, but also where its design fails mechanically. In Fig. 1, we observe qualitatively different failure modes across models, including missing mounting features, self-intersections between links, and insufficient clearance around actuators. Table 1 reports the number of checks passed by each model. GPT6-Astra shows substantially stronger mechanical understanding than the other GPT variants. We then manually correct the remaining design issues and fabricate the embodiment via 3D printing.

Table 1: Pre-fabrication feasibility of robot embodiments generated by GPT variants. Each entry x/y denotes the number of passed checks among the y checks in that category. Appendix I.2 provides the complete check definitions and per-design verdicts.

Check category	GPT6-Astra	GPT5.6-Sol	GPT5.6-Terra	GPT 5.5
Design and Assembly	6/10	4/10	4/10	3/10
Mechanical and Actuation Feasibility	4/6	0/6	1/6	0/6
Motion and Task Execution	5/7	1/7	1/7	1/7

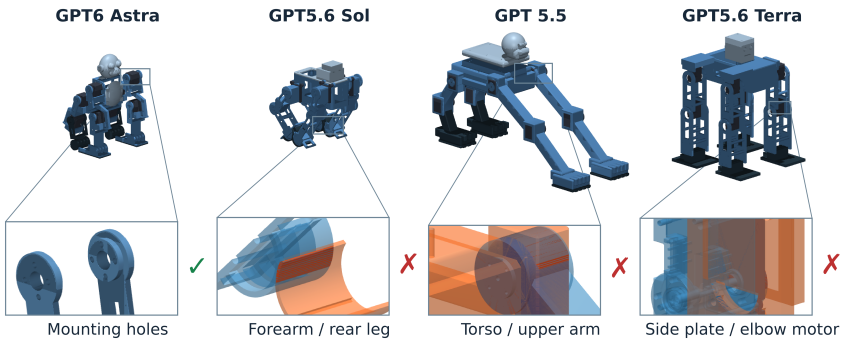


Figure 1: Generated robots (top) and zoomed details (bottom). GPT6-Astra shows the correct mounting-hole layout. The other views expose intersections between the forearm and rear leg in GPT5.6 Sol, the torso and upper arm in GPT 5.5, and the sideplate and elbow motor in GPT5.6 Terra. Surrounding parts are hidden or made transparent to reveal the interfaces.

4 RoboForge

We propose **RoboForge** (Figure 2), an agentic framework that learns executable robot behaviors from a given embodiment D and task specification T. RoboForge generates a reference motion M, configures a task-specific reward r, and trains a policy π, then uses policy execution to refine M, r, or both while keeping D and T fixed.

4.1 Reference Motion Generation and Policy Learning

Given the robot model D and task specification T, the agent instantiates the requested behavior through a robot-specific procedural motion specification. Rather than directly predicting joint trajectories, the agent specifies a root-pose trajectory $\mathbf{B}(t)$, end-effector targets $\mathbf{E}(t)$, and explicit trajectories $\mathbf{q}_{\mathcal{K}}(t)$ for selected task-relevant joints $\mathcal{K}$. Together, these components specify the whole-body motion, limb targets, and task-specific joint behavior.

The limb joints $\mathcal{I}$ that are not explicitly prescribed are then solved through inverse kinematics at each frame:

$$\mathbf{q}^*_{\mathcal{I}}(t) = \arg\min_{\mathbf{q}_{\mathcal{I}}} \|\mathbf{e}(\mathbf{q}_{\mathcal{I}}; \mathbf{B}(t), \mathbf{E}(t), \mathbf{q}_{\mathcal{K}}(t))\|^2, \qquad \mathbf{q}_{\min,\mathcal{I}} \le \mathbf{q}_{\mathcal{I}} \le \mathbf{q}_{\max,\mathcal{I}}, \tag{1}$$

where $\mathcal{I}$ denotes the limb joints being optimized, $\mathbf{B}(t)$ is the prescribed root pose, $\mathbf{E}(t)$ denotes the target end-effector states, and $\mathbf{q}_{\mathcal{K}}(t)$ is held fixed during IK. The residual $\mathbf{e}$ measures the error between the resulting and target end-effector states. Combining the prescribed and IK-solved joint trajectories over all frames yields the joint-position trajectory $\mathbf{Q} \in \mathbb{R}^{N\times J}$, with joint velocities $\dot{\mathbf{Q}} \in \mathbb{R}^{N\times J}$ obtained by temporal differentiation. Forward kinematics further provides the corresponding world-frame body states, including body positions, orientations, linear velocities, and angular velocities. Together, these trajectories form a candidate reference motion $\widetilde{M}$.

The complete trajectory is checked for joint-position and velocity limits, self-collision, and consistency with the requested behavior, temporal structure, and style. Violations trigger revision of the procedural specification and re-solving. The accepted motion is denoted by M.

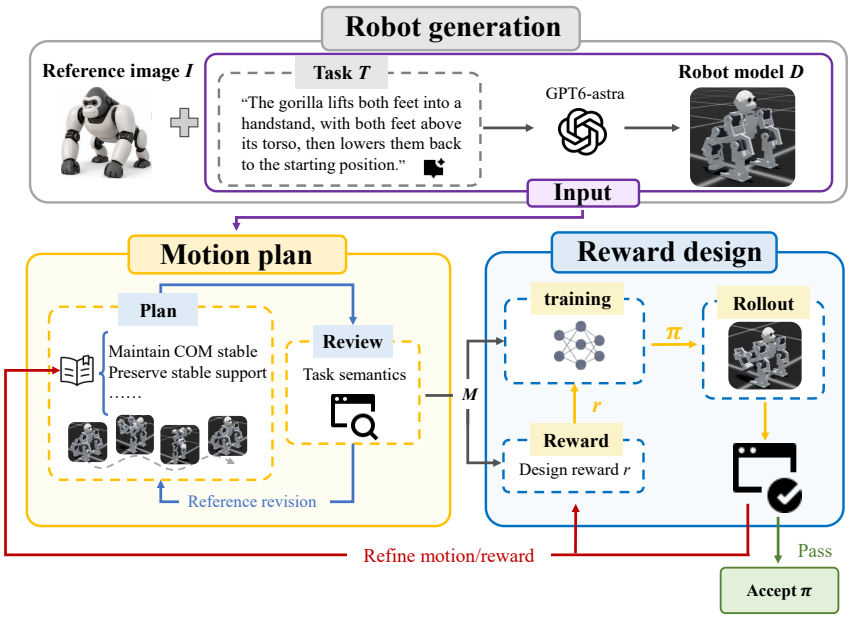


Figure 2: **Overview of RoboForge: motion-guided, execution-grounded policy learning.** Given a robot embodiment and task, the agent generates and validates a reference motion, configures the learning objective, and trains a policy. Execution feedback then refines the motion, reward, or both while keeping the embodiment and task fixed.

Using T and M, the agent configures an initial reward r from a DeepMimic(Peng et al., 2018)-style reference-tracking library of composable tracking terms, for example, joint-pose, joint-velocity, and body-motion tracking terms, and trains a policy π through reinforcement learning. Here, M specifies a concrete realization of the behavior, while r provides the optimization guidance for learning it. Together, they form the initial learning guidance, whose effectiveness is subsequently evaluated through policy rollouts.

4.2 Execution-Grounded Motion and Reward Refinement

A semantically valid and kinematically feasible reference does not necessarily provide effective learning guidance, as valid realizations can differ substantially in posture, timing, support, whole-body coordination, and learnability. We therefore evaluate policies through simulation rollouts for task-semantic compliance and execution quality, including motion quality, support, and balance.

We denote the initial motion, reward, and policy by $M_0 = M$, $r_0 = r$, and $\pi_0 = \pi$. At refinement round k, M_k and r_k are used to train π_k. If the trial budget is exhausted, π_k is returned without further review; otherwise, its rollout is evaluated against the same criteria. If successful, π_k is returned; otherwise, the agent revises M_k, r_k, or both to obtain (M_{k+1}, r_{k+1}).

4.3 Learnability-Aware Execution Diagnosis and Revision

For each rollout that enters refinement, the agent localizes the failure across four levels: the intended task, the reference motion M_k and body trajectory, the policy command $\mathbf{q}_{\text{cmd}}(t)$, and the realized motion $\mathbf{q}_{\text{actual}}(t)$ together with contact states. This helps distinguish limitations of the current motion realization from failures introduced during policy learning and execution. Tracking discrepancies are treated as diagnostic evidence rather than the evaluation objective.

The agent diagnoses whether the failure stems from M_k, considering motion structure, geometric and joint margins, timing, support transitions, and whole-body coordination, or from the optimization guidance in r_k. Motion-related failures lead to revisions of posture, trajectory, timing, support coordination, or phase structure while preserving task semantics and successful behavior; reward-related failures lead to adding, removing, or reweighting reward terms. Both may be revised when

Table 2: User-study ratings across four robot embodiments ($n = 57$; ↑). RoboForge achieves the highest overall rating and the highest rating on three of four embodiments.

Method	Go2	MicroDuck	Ostrich	Gorilla	Overall
Eureka-style	2.79	2.01	2.00	1.59	2.10
One-shot Motion	3.93	3.30	**4.04**	3.14	3.60
RoboForge (Ours)	**4.10**	**4.03**	3.99	**3.77**	**3.97**

Table 3: Reference-independent execution-quality metrics (↓). RoboForge achieves the lowest mean value for every metric.

Method	Yaw Err. (rad)	Foot-slip (m/s)	Action Var. (rad^2)	Joint Acc. (rad/s^2)
Eureka-style	0.0345	0.121	0.0830	114
One-shot Motion	0.0584	0.0716	0.00665	31.7
RoboForge (Ours)	**0.00836**	**0.0131**	**0.00290**	**22.0**

their effects interact. Each revision specifies the suspected cause, proposed change, behavior to preserve, and expected evidence in the next rollout. Section 5.4 evaluates this structured procedure.

5 Experiments

This section addresses three central questions: **Q1: Does execution-grounded motion-reward refinement improve policy learning? Q2: What does keeping motion in the loop change? Q3: Can GPT recover the refinement strategy without explicit guidance?** Together, these experiments evaluate the effectiveness of execution-grounded motion–reward refinement, examine how keeping motion in the loop allows execution feedback to reshape the concrete realization of a task toward more learnable behaviors, and test whether the GPT agent can independently recover and execute the proposed refinement strategy.

5.1 Experimental Setup

Tasks and robots. We evaluate four embodiments: Unitree Go2, MicroDuck, and GPT-generated Ostrich and Gorilla, with the Gorilla additionally fabricated and evaluated on hardware. We use five natural-language tasks for each of the first three robots and four for Gorilla, yielding 19 tasks in total. The tasks span locomotion, stylized behaviors, and whole-body skills with diverse coordination and contact patterns, while admitting multiple realizations consistent with the same task semantics. Robot and task details are provided in Appendix A and C, respectively.

Baselines. We compare **Eureka-style** and **One-shot Motion** with **RoboForge**. Eureka-style, inspired by Eureka (Ma et al., 2024b), refines rewards from policy execution without reference motions. One-shot Motion follows a MimicAgent-style approach (Nayak et al., 2026), using the same validated motion-generation pipeline as RoboForge but fixing the reference once policy learning begins. RoboForge instead refines the motion, reward, or both from execution feedback, with at most three training trials per task. All methods share the same PPO architecture, simulator configuration, and GPT-6 Astra agent (OpenAI, 2026). Further details are provided in Appendix B and D.

Metrics. We evaluate final policy behavior using metrics independent of each method's reference motion and reward formulation, since neither reference-tracking errors nor optimized reward values are directly comparable across methods. Task-level behavior is assessed through a visual **user study**, where raters score compliance with the natural-language task and execution quality. We also report four reference-independent quantitative metrics. For walking tasks, which require straight-line locomotion without a target speed, **Yaw Err.** measures directional compliance and **Foot-slip** characterizes contact stability. Across all tasks, Action Variation (**Action Var.**) and **Joint Acc.** measure smoothness at the command and realized-motion levels, respectively. Detailed definitions are provided in Appendix F.

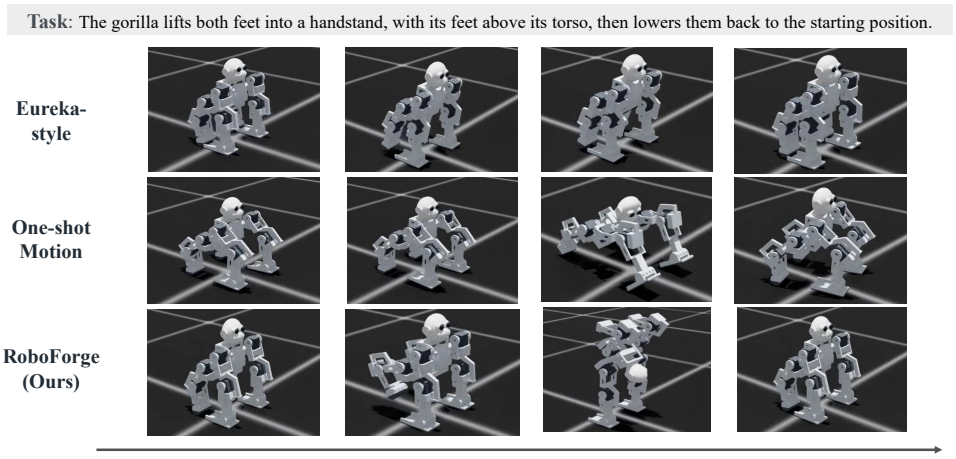


Figure 3: Qualitative comparison of learned policy executions on the gorilla handstand task. Eureka-style fails after an initial rear-leg lift, while One-shot loses balance with outward leg rotation. RoboForge folds the legs inward before inversion, enabling a balanced transition and successful completion.

Table 4: Comparison with unguided GPT. GPT spontaneously introduces and refines reference motions, while our structured pipeline achieves higher user ratings and better execution metrics. *Motion generation is prescribed in our pipeline.

	Agent Behavior		Final Policy Quality				
Method	Motion Gen.	Motion Refine	User Rating ↑	Yaw Err. ↓	Foot-slip ↓	Action Var. ↓	Joint Acc. ↓
GPT	16/19	7/19	3.61	0.0172	0.0419	0.0179	29.2
RoboForge (Ours)	19/19*	11/19	**3.97**	**0.00836**	**0.0131**	**0.00290**	**22.0**

5.2 Q1: Does Execution-Grounded Motion-Reward Refinement Improve Policy Learning?

Table 2 compares RoboForge with two complementary alternatives. Eureka-style retains reward refinement in the execution loop but provides no explicit motion guidance; RoboForge improves the overall rating from 2.10 to 3.97, indicating the value of adding structured motion guidance. One-shot Motion provides such guidance but fixes the accepted reference once policy learning begins. Relative to this baseline, RoboForge improves the overall score from 3.60 to 3.97 and achieves the highest rating on three of four embodiments, with only a slight decrease on Ostrich. Since One-shot Motion corresponds to the first policy-training trial of our pipeline, this comparison shows the benefit of keeping the initially generated learning guidance revisable after observing policy execution. Together, the two comparisons highlight complementary roles: motion provides an explicit behavioral realization, while execution-grounded refinement allows that realization and its learning objective to adapt when execution exposes their limitations.

The user-study results are supported by the reference-independent quantitative metrics in Table 3, where our method performs best on all four measures. Lower Yaw Err. and Foot-slip indicate more accurate directional control and cleaner support contacts, while lower Action Var. and Joint Acc. indicate smoother commands and realized motion. These results show that the user-rated improvements are accompanied by measurable gains in execution quality.

Figure 3 illustrates this process on the Gorilla handstand task. Eureka-style briefly lifts the rear legs but fails to complete the motion, while One-shot Motion rotates the lower legs outward and rapidly loses balance. Our method instead revises the realization by folding the lower legs inward before rotating the body upward, producing a more compact transition that completes the handstand. This comparison illustrates a key advantage of execution-grounded motion–reward refinement: rather

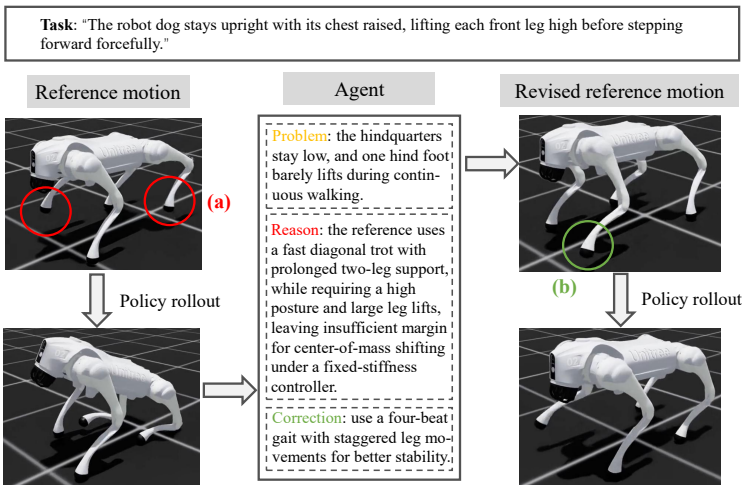


Figure 4: Motion refinement preserves task semantics while improving gait realization. The initial gait has two airborne feet (a), whereas the refined gait uses single-foot swing (b) with coordinated body shift, yielding a more stable four-beat gait.

than committing to an initially generated realization, the learning guidance can adapt after observing policy failure, allowing the resulting policy to adopt a realization that is better aligned with successful execution.

5.3 Q2: What Does Keeping Motion in the Loop Change?

Keeping motion in the loop allows the agent to revise *how* a task is realized while preserving *what* it requires. As shown in Figure 4, both references satisfy the requested high-posture, high-stepping walk, but the initial two-beat diagonal trot is difficult to reproduce, with rollouts showing rear-leg support collapse and insufficient hind-foot clearance. The agent attributes this difficulty to limited center-of-mass transfer under two-leg support, high posture, and large leg excursions, and revises the motion to a four-beat gait with staggered limb swings and coordinated body shifts. This preserves task semantics while providing a more learnable realization, showing that execution feedback can revise a valid but difficult-to-realize reference rather than only the learning objective.

Across the 19 evaluated tasks, the framework revises the generated motion in 11 tasks, with 15 revisions in total. This prevalence shows that execution frequently exposes limitations missed during motion generation, making refinement a recurring step in bridging generation and execution.

5.4 Q3: Can GPT Discover the Refinement Strategy Without Explicit Guidance?

To test whether the refinement strategy emerges without explicit guidance, we remove the structured procedure and ask GPT to solve each task through reinforcement learning using only the shared policy architecture and training interface. Without instructions to generate or revise reference motions, GPT introduces motion guidance in 16 of 19 tasks and refines it in 7. This shows that motion-guided learning and execution-driven motion revision can emerge independently in the unguided GPT agent.

Discovering the strategy does not guarantee effective execution. Table 4 shows that our structured pipeline achieves higher user ratings and better results on all four reference-independent metrics, with especially large reductions in foot slip and action variation. This indicates that explicit guidance helps translate execution failures into more effective motion and reward revisions, rather than merely prompting refinement itself.

Figure 5 illustrates this distinction. Both approaches observe foot sliding and increase swing-foot clearance, but differ in how they revise the motion. Unguided GPT primarily corrects the visible gait deficiency, whereas our framework also revises body–support coordination by shifting the body

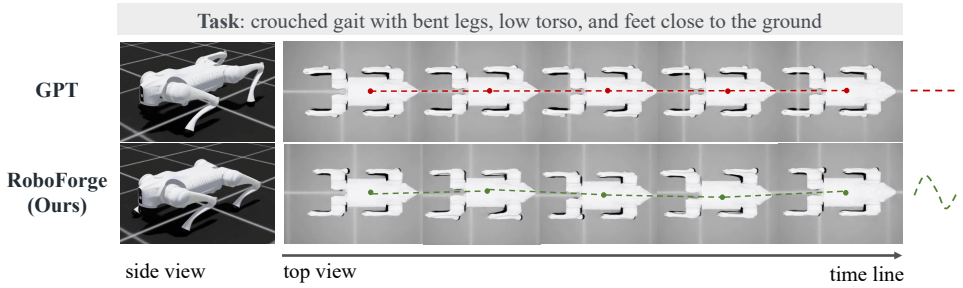


Figure 5: Refined motions from GPT and RoboForge. Both increase swing-foot clearance, but differ in whole-body coordination. CoM trajectories show that GPT remains nearly straight, whereas RoboForge shifts laterally toward the supporting legs for coordinated weight transfer.

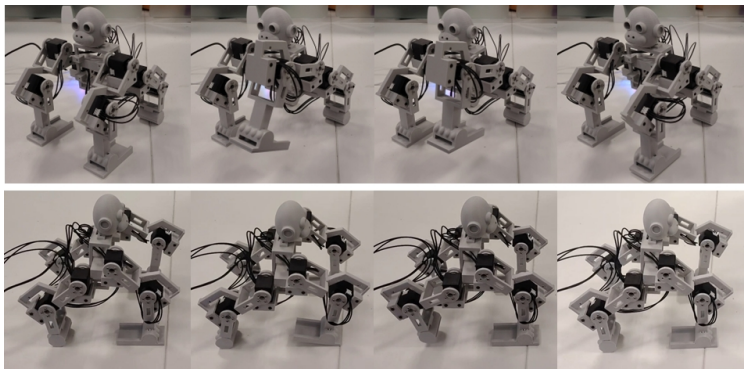


Figure 6: Physical realization of the generated Gorilla and its learned behaviors. Top: hand waving. Bottom: walking. Frames are ordered from left to right.

toward the stance legs during swing while preserving their support trajectories in world coordinates. The resulting motion therefore improves not only gait appearance, but also the whole-body realization presented for policy learning. Further results are provided in Appendix H.

6 Real-World Demonstration

We validate RoboForge on hardware using the agent-generated 8-DoF Gorilla, which we 3D-print and assemble (Appendices A.3 and A.4). We directly deploy the simulation-trained walking and hand-waving policies: walking requires repeated hand–foot support transfer, while hand waving maintains support through the remaining limbs (Figure 6). These demonstrations extend the behavior-learning results from simulation to physical execution on the fabricated robot.

7 Conclusion

We studied agentic robot creation from two complementary aspects: embodiment and behavior creation. For embodiment creation, our computational metrics expose mechanically relevant failures in MLLM-generated designs before fabrication. For behavior creation, we presented **RoboForge**, which learns executable policies through motion generation and execution-grounded refinement. Across four embodiments, RoboForge improves task-level behavior and execution quality, while direct deployment on the fabricated Gorilla demonstrates learned behaviors on hardware. Together, these results highlight the value of physical verification for generated embodiments and adaptive motion guidance for behavior learning.

AI USE STATEMENT

Generative AI tools were used during this work to assist with research ideation and execution, to draft portions of the manuscript, and to aid with language editing and polishing. AI systems also constitute part of the proposed research framework and experimental methodology, including robot generation and execution-grounded motion and reward refinement; these uses are described explicitly in the corresponding method and experimental sections. All research decisions, experimental design, implementation, analysis, verification of results, and final manuscript content were reviewed and approved by the authors.

ETHICS STATEMENT

Our user study involved human participants. All participants were informed of the purpose and procedure of the study and provided informed consent before participation. Participation was voluntary, and participants could withdraw at any time. We did not collect personally identifiable or sensitive information, and all reported results were aggregated and anonymized. The study was conducted in accordance with our institution's requirements for research involving human participants.

REPRODUCIBILITY STATEMENT

We provide the key implementation and evaluation details used in our experiments in the Appendix, including the policy-training setup, baseline configurations, metric definitions, and user-study protocol. Each refinement trial retrains the policy from scratch without reusing previous checkpoints. Additional experimental settings are reported where necessary to interpret and reproduce the presented results.

REFERENCES

Gang Chen, Moji Shi, Yu Xing, Marija Popović, Javier Alonso-Mora, Lei Zhang, and Jiangmiao Pang. Automatic personalized limbed robot design from media inputs. *npj Robotics*, 2026.

Alejandro Escontrela, Xue Bin Peng, Wenhao Yu, Tingnan Zhang, Atil Iscen, Ken Goldberg, and Pieter Abbeel. Adversarial motion priors make good substitutes for complex reward functions. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 25–32. IEEE, 2022.

Jiawei Fang, Yuxuan Sun, Chengtian Ma, Qiuyu Lu, and Lining Yao. Robomore: Llm-based robot co-design via joint optimization of morphology and reward. *arXiv preprint arXiv:2506.00276*, 2025.

Zipeng Fu, Qingqing Zhao, Qi Wu, Gordon Wetzstein, and Chelsea Finn. Humanplus: Humanoid shadowing and imitation from humans. *arXiv preprint arXiv:2406.10454*, 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. *Advances in Neural Information Processing Systems*, 38:172532–172577, 2026.

Ruben Grandia, Espen Knoop, Michael A Hopkins, Georg Wiedebach, Jared Bishop, Steven Pickles, David Müller, and Moritz Bächer. Design and control of a bipedal robotic character. *arXiv preprint arXiv:2501.05204*, 2025.

Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. Revolve: Reward evolution with large language models using human feedback. In *International Conference on Learning Representations*, volume 2025, pp. 101949–101990, 2025.

Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Omnih2o: Universal and dexterous human-to-humanoid whole-body teleoperation and learning. *arXiv preprint arXiv:2406.08858*, 2024.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.

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.

Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. Auto mc-reward: Automated dense reward design with large language models for minecraft. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16426–16435. IEEE, 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.

Zhengyi Luo, Jinkun Cao, Kris Kitani, Weipeng Xu, et al. Perpetual humanoid control for real-time simulated avatars. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10895–10904, 2023.

Weicheng Ma, Luyang Zhao, Chun-Yi She, Yitao Jiang, Alan Sun, Bo Zhu, Devin Balkcom, and Soroush Vosoughi. On the exploration of lm-based soft modular robot design. *arXiv preprint arXiv:2411.00345*, 2024a.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pp. 26516–26560, 2024b.

David Müller, Espen Knoop, Dario Mylonopoulos, Agon Serifi, Michael A Hopkins, Ruben Grandia, and Moritz Bächer. Olaf: Bringing an animated character to life in the physical world. *IEEE Robotics and Automation Letters*, 2026.

Lucky Kant Nayak, Narayanan Palghat Parameswaran, Neehar Peri, and Deva Ramanan. Mimicagent: Quadruped skills via prompt-to-trajectory generation, 2026. URL `https://arxiv.org/abs/2609.24145`.

OpenAI. GPT-6 Astra: A New Generation of Intelligence, 2026. URL `https://openai.com/index/gpt-6-astra/`.

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, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

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 (TOG)*, 41(4):1–17, 2022.

Yanyuan Qiao, Kieran Gilday, Yutong Xie, and Josie Hughes. Lang2morph: Language-driven morphological design of robotic hands. *arXiv preprint arXiv:2509.18937*, 2025.

Kevin Qiu, Władysław Pałucki, Krzysztof Ciebiera, Paweł Fijałkowski, Marek Cygan, and Łukasz Kuciński. Robomorph: Evolving robot morphology using large language models. *arXiv preprint arXiv:2407.08626*, 2024.

Ryan P Ringel, Zachary S Charlick, Jiaxun Liu, Boxi Xia, and Boyuan Chen. Text2robot: Evolutionary robot design from text descriptions. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5789–5797. IEEE, 2025.

Junru Song, Yang Yang, Huan Xiao, Wei Peng, Wen Yao, and Feifei Wang. Laser: Towards diversified and generalizable robot design with large language models. In *International Conference on Learning Representations*, volume 2025, pp. 58460–58489, 2025.

Nitish Sontakke, K Niranjan Kumar, and Sehoon Ha. Robotdesigngpt: Automated robot design synthesis using vision language models. *arXiv preprint arXiv:2601.11801*, 2026.

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

Jiawei Wang, Dingyou Wang, Jiaming Hu, Qixuan Zhang, Jingyi Yu, and Lan Xu. Kinematify: Open-vocabulary synthesis of high-dof articulated objects. *arXiv preprint arXiv:2511.01294*, 2025.

Tianbao Xie, Siheng Zhao, Chen Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 35663–35699, 2024.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

A Robot Embodiments

A.1 Embodiment Overview

We evaluate our framework on four robot embodiments: the Unitree Go2, MicroDuck, and two robots generated by GPT-6 Astra, Ostrich and Gorilla. We particularly highlight Gorilla as a physically realized generated embodiment. Its construction explicitly accounts for the real motor model, self-collisions that arise during target-task execution, and manufacturability, enabling the resulting design to be directly fabricated and assembled. Figure 7 shows the GPT-generated Ostrich and Gorilla. Once an embodiment is constructed, its structure and joint definitions remain fixed throughout the subsequent motion–RL loop; refinement operates only on the learning guidance for that embodiment.



(a) Ostrich (b) Gorilla

Figure 7: GPT-generated robot embodiments used in our experiments. (a) Ostrich. (b) Gorilla, whose design is additionally validated through physical fabrication and assembly.

A.2 Ostrich Kinematic Tree

Ostrich has 15 actuated degrees of freedom, each driven by a single motor. Its kinematic tree consists of 18 links connected by 17 joints, including 15 revolute joints and two fixed joints. The torso (`base_link`) forms the root, from which four branches extend to the neck and head, the left leg, the right leg, and the tail.

Neck and head: six motors. The neck and head form a six-DoF serial chain. Five pitch joints connect the torso to four successive neck links and then to the head mount, followed by a yaw joint connecting the head mount to the head. This branch therefore contains six links beyond the torso: four neck links, one head mount, and one head link.

Left and right legs: four motors each. Each leg branches from the torso through four actuated joints: hip roll, hip pitch, knee pitch, and ankle pitch. The corresponding link chain consists of the hip-roll link, thigh, shin, ankle, and foot. The foot is rigidly attached to the ankle through a fixed joint and therefore introduces no additional actuated degree of freedom. Together, the two legs account for eight motors.

Tail: one motor. A single pitch joint connects the torso directly to the tail link. Overall, the 15 motors are distributed as six across the neck and head, four on each leg, and one on the tail.

A.3 Gorilla Kinematic Tree

Gorilla has eight actuated degrees of freedom, each driven by a single motor. Its kinematic tree consists of nine links connected by eight revolute joints. The torso (`base`), which also incorporates

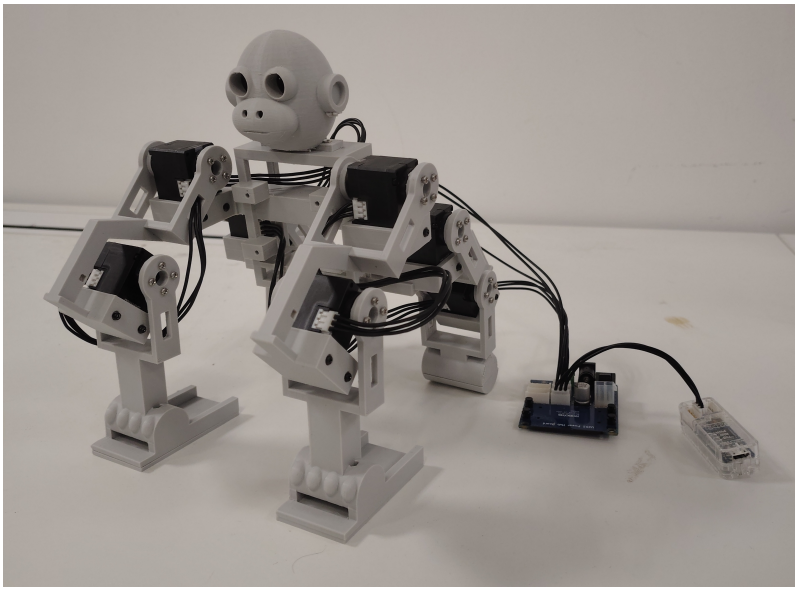


Figure 8: Fabricated Gorilla embodiment.

the head, forms the root of four branches corresponding to the left arm, right arm, left leg, and right leg.

Left and right arms: two motors each. Each arm contains two pitch joints. A shoulder-pitch joint connects the torso to the upper arm, followed by an elbow-pitch joint connecting the upper arm to the lower arm. The hand is integrated into the lower-arm link and therefore does not introduce a separate wrist joint or actuated degree of freedom. Each arm consequently contains two links and two actuated joints.

Left and right legs: two motors each. Each leg similarly contains two pitch joints. A hip-pitch joint connects the torso to the upper leg, followed by a knee-pitch joint connecting the upper leg to the lower leg. The foot is integrated into the lower-leg link and therefore introduces no separate ankle joint or actuated degree of freedom. Each leg consequently contains two links and two actuated joints.

All eight actuated joints are pitch joints, yielding two motors per limb and eight motors in total.

A.4 Gorilla Hardware Fabrication

We fabricated the agent-generated Gorilla from its verified CAD design. The structural components were 3D printed in PLA using a Bambu Lab X1E printer with a layer height of 0.2 mm and an infill density of 15%. Figure 8 shows the resulting physical embodiment.

The electrical and actuation hardware comprises eight DYNAMIXEL XL330-M288-T actuators, a U2D2 communication interface, a U2D2 Power Hub Board Set, an external power supply, and a host PC. The U2D2 bridges the PC's USB connection to the actuators' TTL serial bus, allowing the PC to send joint commands and receive actuator feedback using DYNAMIXEL Protocol 2.0. The Power Hub Board distributes power from the external supply to the actuators.

B Simulation and Reinforcement Learning Setup

This section specifies the common simulation environment, actor architecture, observation interface, and PPO configuration used as the reinforcement-learning template. Unless otherwise noted, the settings below define the common starting configuration, with task-specific settings specified by

each method when required. Task objectives and refinement procedures are described in Appendix D and C.1.

B.1 Simulation and Control Settings

Experiments use Isaac Lab 2.1 with Isaac Sim 4.5. The scene contains a flat ground plane at $z = 0$, with static and dynamic friction coefficients of 1.0 and restitution 0. Each robot is controlled at 50 Hz across all four methods, corresponding to one policy update every 0.02 s. Joint-position targets are held between policy updates while the simulator advances the robot dynamics.

Table 5 lists the physics timestep Δt_{sim} and control decimation d, i.e., the number of physics steps per policy update. These satisfy $f_\pi = 1/(\Delta t_{\text{sim}} d) = 50$ Hz. Go2, MicroDuck, and Ostrich use a 0.005 s physics timestep with $d = 4$, whereas Gorilla uses a 0.002 s timestep with $d = 10$.

Table 5: Simulation timing shared across methods. All robots are controlled at a policy frequency of 50 Hz.

Robot	Physics step (s)	Decimation	Policy (Hz)
Go2, MicroDuck, Ostrich	0.005	4	50
Gorilla	0.002	10	50

B.2 Noise and Initialization

All methods use the same nominal home configuration for each robot and are equally permitted to specify a task-dependent initial pose when appropriate for the target behavior.

By default, the motion-tracking configurations used by One-shot Motion and RoboForge add no observation noise to base velocities, projected gravity, joint positions, joint velocities, or observation history. Joint-position reset perturbations and reset-velocity scaling are initialized to zero, contact-material parameters are fixed, and no center-of-mass offsets or external pushes are applied.

The GPT and Eureka-style templates likewise disable observation corruption by default. Task-specific implementations may specify reset states, episode lengths, termination conditions, and physical perturbations. The base environment uses a default episode length of 10 s.

B.3 Actor Network and Observation Template

All methods use a feedforward actor with three hidden layers of widths 512, 256, and 128. Each hidden layer uses an ELU activation, followed by a linear output layer. Given an observation vector $o_t \in \mathbb{R}^{d_o}$, the actor produces an action mean in $\mathbb{R}^{d_a}$ through

$$d_o \longrightarrow 512; (\text{ELU}) \longrightarrow 256; (\text{ELU}) \longrightarrow 128; (\text{ELU}) \longrightarrow d_a. \tag{2}$$

The base observation concatenates the velocity command $\mathbf{c}_t = [v_x^{\text{cmd}}, v_y^{\text{cmd}}, \omega_z^{\text{cmd}}]$, body-frame projected gravity $\mathbf{g}_t^b$, joint positions relative to the nominal configuration, joint velocities, and the previous raw action:

$$o_t = [\mathbf{c}_t, \mathbf{g}_t^b, \mathbf{q}_t - \mathbf{q}_0, \dot{\mathbf{q}}_t, \mathbf{a}_{t-1}], \qquad d_o = 6 + 2J + d_a, \tag{3}$$

where J denotes the number of joints and $\mathbf{q}_0$ the nominal joint configuration used by the task interface. Joint positions and velocities are expressed in rad and rad/s, respectively, while the command specifies body-frame planar velocity in m/s and yaw rate in rad/s. The base interface uses the current observation without a history window. The actor observation may be extended only with signals available to the policy at inference time, while retaining consistent feature definitions, joint mappings, and temporal interpretation between training and inference. Additional observations may be used by training-only components, such as the critic or, where applicable, a discriminator, but are not provided to the actor.

The PPO runner applies empirical observation normalization,

$$\hat{o}_t = \frac{o_t - \boldsymbol{\mu}_o}{\boldsymbol{\sigma}_o + 10^{-2}}, \tag{4}$$

where the running mean and standard deviation are estimated during training. Statistics stop updating after 10^8 observed samples and are stored with the policy for inference.

During training, the policy samples raw actions from a diagonal Gaussian whose mean is produced by the actor. Each action dimension has a learnable, state-independent standard deviation initialized to 1.0. At inference, the policy uses the action mean. Task-specific action mappings convert raw actions into valid joint-position targets using the corresponding scaling, offsets, and joint limits.

B.4 PPO Optimization Defaults

The shared RL template uses RSL-RL PPO with $4{,}096$ parallel environments and collects 24 policy steps per environment before each optimization phase. Each iteration therefore contains $98{,}304$ transitions and spans 0.48 s of simulated control per environment at 50 Hz. The rollout batch is divided into four minibatches of $24{,}576$ transitions and optimized for five epochs, resulting in 20 optimizer updates per iteration.

Table 6: Default PPO and rollout configuration of the shared RL template.

Parameter	Default value
Parallel environments	$4{,}096$
Policy steps per environment per iteration	24
Transitions per iteration	$98{,}304$
Optimization epochs per iteration	5
Minibatches per epoch	4
Transitions per minibatch	$24{,}576$
Optimizer	Adam
Initial learning rate	10^{-3}
Learning-rate schedule	Adaptive to KL divergence
Target KL divergence	0.01
Discount factor γ	0.99
GAE parameter λ	0.95
PPO clipping parameter	0.2
Entropy coefficient	0.005
Value-loss coefficient	1.0
Clipped value loss	Enabled
Maximum gradient norm	1.0
Advantage normalization	Full rollout batch
Observation normalization	Empirical running statistics
Initial action standard deviation	1.0

Advantages are computed using generalized advantage estimation and normalized over the full rollout batch. The adaptive learning-rate schedule divides the learning rate by 1.5 when the mean KL divergence exceeds 0.02 and multiplies it by 1.5 when the divergence lies in $(0, 0.005)$, with the learning rate bounded to $[10^{-5}, 10^{-2}]$.

The default random seed is 0 and can be specified at launch. Checkpoints are saved every 500 iterations and once at the end of training. For GPT, One-shot Motion, and RoboForge, training uses a cumulative budget of up to $3{,}000$ iterations per trial; Eureka-style uses $2{,}250$ iterations per candidate. Resumed runs restore the policy, optimizer, observation-normalization statistics, and completed iteration count before continuing within the remaining trial budget. Trial counts and per-task budgets are specified in Appendix D.4.

C Task Diversity and Motion Realizations

C.1 Task Coverage

We evaluate 19 natural-language tasks across four robot embodiments, including seven walking tasks and twelve skills. The task set spans locomotion, stylized behavior, and whole-body coordination, with substantial variation in both the body parts involved and the required contact patterns. Table 7 lists the complete task inventory. Stylistic requirements appear throughout both walking and skill tasks; for example, locomotion may specify not only forward movement but also a recognizable posture or manner of execution.

Table 7: Task inventory across four embodiments: seven walking tasks and twelve whole-body skills.

Robot	Walking tasks	Whole-body skills	Total
Go2	High-step walk; low walk	Bow; look around; kick	5
MicroDuck	Fast walk; waddling walk	Bow; look around; kick	5
Ostrich	Happy walk; sad walk	Nod; stretch; squat and rise	5
Gorilla	Alternating walk	Handstand; stretch; wave	4
Total	7 walking tasks	12 skills	19

The required contact patterns range from posture changes under stationary support to alternating stepping, transient single-foot support, and hand-supported inversion. Successful execution therefore depends on coordinating expressive motion with the limbs responsible for support and balance. Task completion cannot in general be characterized by locomotion progress alone, motivating evaluation at the level of the requested behavior and its execution quality.

Importantly, the task instruction constrains the behavior without uniquely determining its motion realization. It may specify a recognizable style, action order, or required support phase, while leaving quantities such as motion amplitude, timing, posture, and inter-limb coordination underdetermined. This flexibility is what enables execution-grounded refinement: the motion–RL loop can modify how a task is physically realized when the current reference is difficult to learn or execute, while preserving the task semantics specified by the user.

C.2 Stylized Locomotion

Ostrich's two walking tasks illustrate how different motion realizations can share the same underlying locomotion objective (Figure 9). As with all walking tasks in our evaluation, both require straight-line forward locomotion. The happy walk calls for an upright body, light alternating steps, and accompanying head motion, whereas the sad walk requires a lowered head while maintaining alternating forward progression. Their distinction therefore lies not in the locomotion direction, but in how head–neck posture and stepping are coordinated to express the requested style.

C.3 Whole-Body Coordination

MicroDuck's kick illustrates the phased coordination required by a whole-body skill (Figure 10). The requested behavior begins by looking downward and shifting the body toward one side, creating support for lifting the opposite foot. The lifted foot then performs a short forward kick before retracting as the robot returns to a stable two-foot stance. The task semantics therefore include not only the kick itself, but also the preparation, support transition, and recovery required to realize the complete behavior.

D Method Implementation

All agent roles across the compared methods use GPT-6 Astra with reasoning effort set to `high`. The methods receive the same robot assets and natural-language task specifications, while differing

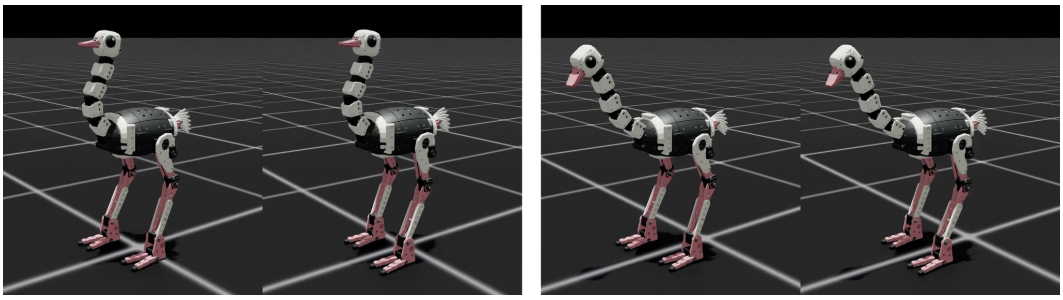


Figure 9: Generated stylized locomotion motions for Ostrich. Left: *happy walking*; right: *sad walking*. Both motions are generated by our method and share the same forward-locomotion objective, while differing in head–neck posture and its coordination with the stepping pattern.

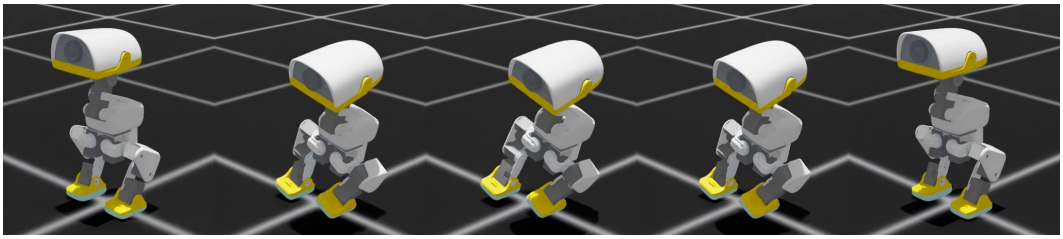


Figure 10: Generated motion for MicroDuck's kick task produced by our method. The sequence coordinates gaze, lateral body shift, the kicking leg, and the supporting leg across preparation, single-foot support, and recovery.

in the learning scaffold they are given and in how policy execution is used to construct or revise learning guidance.

D.1 Unguided GPT and Our Motion–RL Loop

GPT and RoboForge use the same training budget of up to three trials per task. Each trial runs 4,096 parallel environments and is limited to a cumulative budget of 3,000 PPO iterations; resumed training within a trial consumes the remaining budget of that trial. Evaluation uses the terminal policy from the last completed trial.

GPT. The unguided GPT baseline is provided with the shared actor template, the robot and task specification, and general RL tools for constructing rewards, training policies, and evaluating execution. It is not given a prescribed procedure for reference-motion generation, motion refinement, or learnability-aware diagnosis, nor is it provided with the reference-tracking guidance used by RoboForge. Within the same training budget, the agent determines how to construct the learning objective and organize its training process. No motion-based strategy is prohibited: the agent may generate, use, or revise a reference motion if it chooses to do so. This setting examines whether a general-purpose agent independently discovers and executes a motion-based refinement strategy from task execution feedback.

RoboForge. RoboForge is provided with a DeepMimic-style reference-conditioned RL scaffold and explicit guidance for generating, validating, and iteratively refining the learning guidance. Before policy training, the generated reference motion is checked for kinematic validity and task-semantic consistency. After training, policy execution is evaluated against the requested behavior and used to diagnose deficiencies in posture, timing, support transitions, and whole-body coordination. The resulting feedback can revise the reference motion, the reward, or both, while the robot embodiment and task semantics remain fixed.

This comparison therefore tests whether the execution-grounded motion–reward refinement strategy emerges autonomously from a general RL interface or benefits from being explicitly instantiated

as part of the agentic learning procedure. The full refinement procedure is described in the main Method section, while Appendix H analyzes the strategies observed in the unguided GPT baseline.

D.2 MimicAgent-style One-shot Motion

One-shot Motion. Our One-shot Motion baseline follows the generate–validate–train structure of MimicAgent (Nayak et al., 2026). A reference motion is first generated and validated, and the resulting motion is then used to guide imitation-based reinforcement learning. MimicAgent itself allows trajectory generation to iterate in response to task-agnostic validity tests before policy training. Accordingly, "one-shot" refers to the absence of execution-driven motion refinement after policy learning begins, rather than to generating the reference motion only once; the motion itself may be iteratively revised during generation and validation. In our implementation, both the accepted reference motion and the initial reward remain fixed throughout policy training.

To provide the one-shot baseline with the same initial learning guidance as RoboForge, we use the same reference-generation procedure, semantic checks, and initial reward. In addition to kinematic and physical validity, the generated motion is checked against the requested task semantics, style, and required action phases before training. The two methods therefore begin from the same reference and reward; they differ in whether these quantities remain fixed after observing policy execution.

Concretely, One-shot Motion is the terminal policy obtained from the first formal trial of RoboForge, including any continuation within that trial. The trial uses $4{,}096$ parallel environments and a cumulative budget of at most $3{,}000$ PPO iterations. Once this trial is complete, the reference motion and reward are fixed, and no execution-grounded revision is performed. Ours may instead use subsequent trials to revise the motion, reward, or both according to policy execution feedback.

This comparison focuses on the effect of keeping the generated learning guidance open to revision after policy execution. Each refinement trial retrains the policy from scratch without reusing checkpoints from previous trials, so improvements across rounds cannot be attributed to warm-starting or accumulated policy optimization. One-shot Motion trains once with the accepted reference, whereas RoboForge retrains under the same initialization protocol after each revision. If RoboForge terminates after the first trial, the two methods coincide.

D.3 Eureka-style Reward Refinement

Eureka-style. Our Eureka-style baseline instantiates the reward-search and reward-reflection procedure of Eureka (Ma et al., 2024b) within our common RL interface. For each robot and task, the agent first constructs the task environment and an evaluator. Before reward search begins, the observation interface, action mapping, reset and termination rules, PPO configuration, and evaluation interface are fixed. Reward design is then the component revised throughout the search. The agent may modify reward terms, functional forms, weights, and temperature parameters, but does not use reference motions or framewise tracking objectives. This provides an iterative reward-only counterpart to RoboForge, where execution feedback can revise the learning objective but not a motion reference.

We run two reward-search rounds with two candidates per round. Each candidate is trained from scratch using $4{,}096$ parallel environments for $2{,}250$ PPO iterations. The two candidates within a round are proposed before either training result is observed. After training, GPT receives the resulting reward-component statistics and task-evaluation feedback, compares the candidates, and determines how to construct the next round of reward programs. The second round therefore builds on the reward code and execution feedback from the first round, while policy parameters, optimizer state, and observation-normalization statistics are reinitialized for every candidate. Candidate comparison and reward revision are performed by the agent rather than by a hand-crafted selection rule.

We adapt Eureka's reward-refinement procedure to the same agent, simulator, and RL stack used by the other methods. Whereas the original Eureka experiments use GPT-4-0314, pre-existing benchmark environments and fitness functions, IsaacGym, and substantially broader reward search, our implementation uses GPT-6 Astra, Isaac Lab, RSL-RL, and a two-round search with two candidates per round. The search breadth is chosen to match the formal training allocation used by GPT and

RoboForge while preserving Eureka's core loop of reward generation, policy training, execution feedback, and reward revision.

D.4 Training Budgets and Final-Policy Evaluation

All methods use the actor architecture and robot-specific simulation timing described in Appendix B, with 24 policy steps collected per environment per PPO iteration. We control training resources by matching the maximum formal PPO allocation of GPT, RoboForge, and Eureka-style:

$$3 \times 3{,}000 = 2 \times 2 \times 2{,}250 = 9{,}000 \quad \text{PPO iterations per task.} \tag{5}$$

With $4{,}096$ parallel environments, each method therefore receives at most $884{,}736{,}000$ environment transitions per task. One-shot Motion consists only of the first training trial and is limited to $3{,}000$ PPO iterations, or $294{,}912{,}000$ transitions. A method may consume less than its maximum allocation when training terminates before exhausting the budget.

For GPT and RoboForge, evaluation uses the terminal policy from the last completed formal trial, while One-shot Motion uses the terminal policy from the first trial. For Eureka-style, GPT compares the trained reward candidates using the feedback available within the reward-search procedure and determines which candidate to retain. We do not introduce an additional hand-crafted ranking or selection criterion for choosing among Eureka-style candidates.

All submitted policies are subsequently evaluated using the same external user-study rubric and reference-independent physical metrics. These final evaluation measurements are used only for cross-method comparison and do not feed back into policy or reward selection.

The original Eureka protocol performs additional multi-seed PPO retraining after reward search. In our comparison, the policy returned by the budgeted search is used directly for final evaluation, without additional post-search training. This keeps the formal PPO allocation comparable across GPT, Eureka-style, and RoboForge.

E User Study Protocol

Because the compared methods may construct different reference motions and learning objectives, method-specific quantities such as tracking errors or training rewards are not directly comparable across methods. We therefore evaluate the final learned policies under a common behavioral criterion: whether the executed behavior fulfills the natural-language instruction while maintaining satisfactory execution quality. The user study operationalizes this criterion through a shared rating rubric applied identically to all methods.

E.1 Participant Instructions and Rating Rubric

Before rating the policies, participants read a common set of instructions defining the evaluation criteria and the five-point rating scale. For each assigned task, participants are shown the natural-language task description together with four policy videos presented in randomized order. Each video is rated independently on an integer scale from 1 to 5; because the study evaluates each policy rather than requiring a ranking among methods, equal ratings are allowed.

The rubric asks participants to jointly consider two aspects of the final behavior. First, *task-semantic fulfillment* measures whether the policy realizes the behavior specified by the instruction, including the requested action, style, and temporal phases when applicable. Because a task may admit multiple valid motion realizations, participants are instructed not to prefer one realization merely for appearing closer to a particular motion once the task requirements are satisfied. Second, *execution quality* captures properties of the realized motion, including smoothness, stability, coordination, recovery, and recognizable stepping when relevant. These aspects are combined into a single holistic rating: once task semantics are satisfied, differences in rating should primarily reflect execution quality rather than preference for a particular realization. Table 8 summarizes the five rating anchors.

Table 8: Five-point rating rubric used in the user study.

Score	Description
1	The task is largely unfulfilled, or the motion has severe problems.
2	Few task requirements are fulfilled, or motion problems are substantial.
3	The task is partially or mostly fulfilled, with noticeable shortcomings.
4	The task is fulfilled with generally stable motion and minor problems.
5	The task is fulfilled with smooth, coordinated, and stable motion.

E.2 Video Presentation and Study Interface

The study contains 76 policy videos, corresponding to four methods (Eureka-style, One-shot motion, GPT and RoboForge) evaluated on each of the 19 tasks. All videos depict executions of the learned policies and are presented at 1280×720 resolution and 30 frames per second. Within each task, the camera configuration and playback rate are held fixed across methods, while each video preserves the duration of its corresponding policy rollout.

For each task, all four methods are presented with the same natural-language task description and are identified only by anonymous labels (A–D).

E.3 Participants, Task Assignment, and Response Filtering

The study includes 57 participants. Participants are assigned subsets of the 19 tasks, so the number of valid raters varies across tasks. Within each task, however, the same participants evaluate all four methods. This paired presentation keeps the rater population fixed within each method comparison and avoids confounding method differences with differences in task-specific rater composition.

Incomplete or otherwise excluded questionnaires are omitted from the analysis. The questionnaire system additionally applies a three-second minimum completion-time filter and rejects submissions below this threshold. All reported user-study statistics are computed only from the remaining valid responses.

F Metric Definitions and Aggregation

F.1 Human-Rating Statistics

For robot r, method m, and task t, let $\mathcal{R}_{rmt}$ index the valid individual ratings and let $n_{rmt} = |\mathcal{R}_{rmt}|$. Validity follows Appendix E.3: only submitted, non-excluded questionnaires are included. Counts refer to rating entries; one participant may contribute ratings to several tasks. The task-level mean and sample standard deviation are

$$\bar{s}_{rmt} = \frac{1}{n_{rmt}} \sum_{i \in \mathcal{R}_{rmt}} s_i, \qquad \sigma_{rmt} = \sqrt{\frac{1}{n_{rmt} - 1} \sum_{i \in \mathcal{R}_{rmt}} (s_i - \bar{s}_{rmt})^2}, \tag{6}$$

where the sample standard deviation is defined for $n_{rmt} > 1$.

For a task group g (walking or non-walking), let $\mathcal{T}_{rg}$ denote its tasks for robot r and let $n_{rmg} = \sum_{t \in \mathcal{T}_{rg}} n_{rmt}$. We pool all valid ratings within each robot, method, and group:

$$\bar{s}_{rmg} = \frac{1}{n_{rmg}} \sum_{t \in \mathcal{T}_{rg}} \sum_{i \in \mathcal{R}_{rmt}} s_i = \frac{\sum_{t \in \mathcal{T}_{rg}} n_{rmt} \bar{s}_{rmt}}{\sum_{t \in \mathcal{T}_{rg}} n_{rmt}}. \tag{7}$$

Thus, each task contributes in proportion to its number of valid ratings. The symbols n_W and n_S in the rating table denote n_{rmg} for walking and non-walking skill tasks, respectively.

The corresponding sample standard deviation is computed from the same pooled ratings:

$$\sigma_{rmg} = \sqrt{\frac{1}{n_{rmg} - 1} \sum_{t \in \mathcal{T}_{rg}} \sum_{i \in \mathcal{R}_{rmt}} (s_i - \bar{s}_{rmg})^2}. \tag{8}$$

It describes variation among individual ratings in that cell.

For the robot-level score, $\mathcal{T}_r$ includes all tasks of robot r:

$$\bar{s}_{rm} = \frac{\sum_{t \in \mathcal{T}_r} n_{rmt} \bar{s}_{rmt}}{\sum_{t \in \mathcal{T}_r} n_{rmt}}. \tag{9}$$

The Overall column in Appendix G uses this mean and the sample standard deviation of all ratings for that robot and method. Combining the walking and non-walking groups therefore weights their means by n_W and n_S.

The cross-robot Overall and User Rating columns in the main paper give equal weight to the four unrounded robot-level means:

$$\bar{s}_m = \frac{1}{4} \sum_r \bar{s}_{rm}. \tag{10}$$

This two-stage rule is shared by all user-rating comparisons: individual ratings have equal weight within each robot, and the four robots have equal weight in the cross-robot comparison. Means and sample standard deviations are reported to two decimal places. Boldface denotes the largest unrounded mean among the displayed methods, including ties, and does not indicate statistical significance.

F.2 Physical Metrics

Physical metrics are computed from the evaluation rollout of each method–task pair.

Let J denote the number of named joints recorded in the evaluation trace, including stationary joints, and let $a_t, q_t \in \mathbb{R}^J$ denote the commanded joint-position targets and measured joint positions, respectively, at sample t. Both are expressed in radians. Commands are evaluated after the task-specific action mapping, such that the metric operates on the joint targets actually sent to the controller. The initial record, at which no target has yet been sent, is excluded; N denotes the number of remaining valid samples. All finite differences use consecutive samples without padding or filtering. The recorded joint counts are $J = 12$ for Go2, 14 for MicroDuck, 15 for Ostrich, and 8 for Gorilla.

Yaw Err. For a walking trajectory with N samples, let ψ_t be the body yaw obtained from the recorded world-frame quaternion in (w, x, y, z) order. The requested straight-walking direction is world $+X$, with $\psi^\star = 0$. We compute

$$E_{\text{yaw}} = \frac{1}{N} \sum_{t=1}^{N} \left| \text{atan2}(\sin(\psi_t - \psi^\star), \cos(\psi_t - \psi^\star)) \right|. \tag{11}$$

The result is measured in radians and quantifies the mean absolute deviation from the desired heading.

Second-order Action Variation (Action Var.). Writing $\delta^2 a_t = a_t - 2a_{t-1} + a_{t-2}$ for the second difference of the commanded joint-position targets, we compute

$$E_{\text{action}} = \frac{1}{(N-2)J} \sum_{t=3}^{N} \|\delta^2 a_t\|_2^2. \tag{12}$$

This metric measures changes in successive command increments, with lower values indicating smoother command evolution. It is averaged over time and all recorded joints, including stationary joints. The result is in rad^2.

Joint acceleration RMS (Joint Acc.). With measurement interval Δt, we estimate

$$\hat{\ddot{q}}_t = \frac{q_t - 2q_{t-1} + q_{t-2}}{\Delta t^2}, \qquad E_{\text{acc}} = \sqrt{\frac{1}{(N-2)J} \sum_{t=3}^{N} \|\hat{\ddot{q}}_t\|_2^2}. \tag{13}$$

The result is in rad/s^2 and characterizes the magnitude of changes in joint velocity. Squared accelerations are averaged over time and joints before taking the square root. Acceleration is estimated from positions at the metric sampling interval.

Contact-conditioned slip RMS (Foot-slip). Let $v_{t,f}^{xy}$ be the world-horizontal velocity of support point f and $c_{t,f}$ its contact indicator. We compute

$$E_{\text{slip}} = \sqrt{\frac{\sum_{t,f} c_{t,f} \|v_{t,f}^{xy}\|_2^2}{\sum_{t,f} c_{t,f}}}. \tag{14}$$

The result is in m/s and measures horizontal support-point motion during detected contact. Gorilla includes both hands and feet as potential support points. Contact is detected when the recorded net contact-force magnitude on the corresponding body exceeds 10 N for Go2, 0.5 N for MicroDuck, or 0.3 N for Ostrich and Gorilla.

Go2 uses calf-attached foot-center proxies, MicroDuck uses configured ankle support points, Ostrich uses foot-link origins, and Gorilla uses the local hand and foot support points defined by the task. The recorded net body contact force may include self-contact.

F.3 Sampling and Evaluation Windows

The evaluation protocol specifies a control frequency of 50 Hz, corresponding to a control interval of 0.02 s. Metrics are computed over the complete recorded task window at the policy's native playback speed and task duration.

To preserve a common evaluation window, automatic environment resets are disabled during metric collection. Native termination events are still recorded, while the rollout continues until the end of the task window. This allows the physical metrics to characterize the full execution shown for evaluation, including behavior following an early failure.

F.4 Table Aggregation and Reporting

Each physical metric is first computed independently for each task. Task-level values are then averaged equally over the applicable tasks for each robot, and the resulting robot-level values are averaged equally across the four embodiments. Yaw error and contact-conditioned slip are reported over the seven walking tasks, whereas action variation and joint acceleration are computed over all 19 tasks.

Tasks are aggregated only after their individual metrics have been computed, so differences in task duration do not change their relative weight. For RMS metrics, the square root is taken at the task level before averaging across tasks and robots.

The resulting physical metrics characterize complementary properties of policy execution. Lower action second difference and joint acceleration indicate smoother commanded and realized joint trajectories, respectively, while yaw error measures heading deviation and contact-conditioned slip measures support-point motion during contact. These diagnostics are reported together with the human ratings, which assess whether the requested behavior is successfully realized.

Writing $\mathcal{T}_{rk}$ for the applicable tasks of robot r for metric k, and E_{rmtk} for a task-level measurement, the reported means are

$$\bar{E}_{rmk} = \frac{1}{|\mathcal{T}_{rk}|} \sum_{t \in \mathcal{T}_{rk}} E_{rmtk}, \qquad \bar{E}_{mk} = \frac{1}{4} \sum_{r} \bar{E}_{rmk}. \tag{15}$$

A dash denotes a metric that is not applicable to a given task; in particular, yaw and slip are evaluated only on walking tasks. All applicable rollouts are retained in the aggregation, including unsuccessful executions. Undefined measurements are reported as such rather than replaced with zero.

G Detailed Evaluation Results

This appendix provides a more detailed breakdown of the three-method comparison between Eureka-style, One-shot, and RoboForge. We first disaggregate the user ratings by robot and task group to identify where refinement changes the final behavior, and then report physical execution metrics by robot. Together, these results complement the aggregate results in the main paper by showing both where refinement yields substantial gains and where strong One-shot solutions remain competitive.

G.1 User Ratings by Robot and Task Group

Table 9 separates the user ratings into walking and non-walking tasks for each robot. Relative to One-shot, RoboForge obtains a higher overall mean rating on Go2, MicroDuck, and Gorilla, while One-shot remains slightly higher on Ostrich. The task-group breakdown further localizes these differences. On Go2, the overall gain is driven by walking behaviors, whose mean changes from 4.56 to 4.33. MicroDuck improves in both groups, from 2.82 to 4.00 for walking and from 3.60 to 4.05 for non-walking tasks. On Gorilla, walking remains comparable (3.71 versus 3.79), whereas the non-walking mean increases from 2.95 to 3.76. On Ostrich, the non-walking means are tied at 4.25, while the walking mean changes slightly from 3.71 to 3.57.

Importantly, One-shot is the first policy-training trial of our pipeline. This comparison therefore evaluates the benefit of continuing execution-grounded refinement from the same initial motion and reward. The clearest gains occur when the first-trial policy is relatively weak: MicroDuck improves in both task groups, Gorilla improves substantially on non-walking tasks, and Go2 walking also shows a marked gain. When the first trial already performs strongly, refinement changes the ratings only modestly. This pattern suggests that refinement is most useful when the initially generated guidance does not yet translate into satisfactory policy execution. Rather than requiring every additional trial to improve every task, the loop keeps the motion and reward revisable when execution reveals a mismatch between the generated guidance and what the policy can successfully realize.

Table 9: User ratings by robot and task group (↑), reported as mean ± sample standard deviation. n_W and n_S denote the numbers of valid individual ratings per method for walking and non-walking tasks, respectively, and are not counts of distinct participants. Bold indicates the highest unrounded mean among the displayed methods within each robot and column, including ties.

Robot	Method	n_W	Walking	n_S	Non-walking	Overall
Go2	Eureka-style	29	3.00 ± 1.54	43	2.65 ± 1.38	2.79 ± 1.44
	One-shot	29	3.00 ± 1.44	43	$\mathbf{4.56} \pm 0.63$	3.93 ± 1.28
	RoboForge (Ours)	29	$\mathbf{3.76} \pm 0.91$	43	4.33 ± 0.78	$\mathbf{4.10} \pm 0.87$
MicroDuck	Eureka-style	28	2.64 ± 1.34	43	1.60 ± 0.85	2.01 ± 1.18
	One-shot	28	2.82 ± 1.09	43	3.60 ± 1.29	3.30 ± 1.27
	RoboForge (Ours)	28	$\mathbf{4.00} \pm 1.02$	43	$\mathbf{4.05} \pm 0.92$	$\mathbf{4.03} \pm 0.96$
Ostrich	Eureka-style	28	2.07 ± 1.09	44	1.95 ± 1.12	2.00 ± 1.10
	One-shot	28	$\mathbf{3.71} \pm 0.85$	44	$\mathbf{4.25} \pm 0.75$	$\mathbf{4.04} \pm 0.83$
	RoboForge (Ours)	28	3.57 ± 0.96	44	$\mathbf{4.25} \pm 0.84$	3.99 ± 0.94
Gorilla	Eureka-style	14	1.14 ± 0.36	42	1.74 ± 0.80	1.59 ± 0.76
	One-shot	14	3.71 ± 0.61	42	2.95 ± 1.41	3.14 ± 1.30
	RoboForge (Ours)	14	$\mathbf{3.79} \pm 0.58$	42	$\mathbf{3.76} \pm 0.93$	$\mathbf{3.77} \pm 0.85$

G.2 Physical Execution Metrics by Robot

Table 10 reports the physical execution metrics under the same three-method comparison. RoboForge achieves the lowest Foot-slip on all four embodiments and the lowest Action Var. and Joint Acc. on Go2, MicroDuck, and Ostrich, while One-shot performs best on the latter two metrics for Gorilla. Directional accuracy is more embodiment dependent: Eureka-style achieves the lowest Yaw MAE on Go2, One-shot on Ostrich, and RoboForge on MicroDuck and Gorilla. Overall, RoboForge achieves the best value in 12 of the 16 robot–metric comparisons, with particularly consistent gains in support quality and execution smoothness.

This pattern complements the user-study results. The physical metrics characterize specific aspects of execution, including directional accuracy, command smoothness, realized joint motion, and contact slip, whereas the user ratings assess whether the policy realizes the requested behavior as a whole. A favorable value on an individual physical metric therefore does not necessarily imply successful task execution. All method–task outcomes are retained in the analysis, with each metric aggregated only over the tasks for which it is defined.

Table 10: Physical execution metrics by robot (↓). Yaw error and slip are averaged over walking tasks, whereas action variation and joint acceleration are averaged over all tasks. Values are rounded to three significant digits. Bold indicates the minimum among the displayed methods within each robot and metric, including ties. Action var. denotes second-order action variation normalized by the number of named joints.

Robot	Method	Yaw Err. (rad)	Action Var. (rad^2)	Joint Acc. (rad/s^2)	Foot-slip (m/s)
Go2	Eureka-style	**0.00807**	0.0531	72.7	0.148
	One-shot	0.063	0.00307	23.3	0.2
	RoboForge (Ours)	0.0135	**0.000174**	**13.9**	**0.0354**
MicroDuck	Eureka-style	0.0779	0.034	122	0.0564
	One-shot	0.139	0.0222	77.9	0.0674
	RoboForge (Ours)	**0.00435**	**0.0102**	**46.4**	**0.0104**
Ostrich	Eureka-style	0.032	0.0585	144	0.0582
	One-shot	**0.00656**	0.00106	20.7	0.00592
	RoboForge (Ours)	0.0121	**0.000712**	**16.9**	**0.00324**
Gorilla	Eureka-style	0.02	0.186	119	0.223
	One-shot	0.0255	**0.00023**	**4.88**	0.0133
	RoboForge (Ours)	**0.0035**	0.000482	10.8	**0.00353**

G.3 Types of Motion and Reward Revisions

The main-paper Q2 analysis records 15 motion revisions across 11 of the 19 evaluated tasks. These counts characterize how often motion revision is used. The observed changes fall into four recurring types: *posture and limb trajectory*, including body height, leg folding, and swing clearance; *timing and phase structure*, including swing duration and limb ordering; *support transfer*, including preservation of stance contacts and body shifts before limb lift; and *whole-body coordination*, which couples the moving limb with the torso and the remaining support limbs. A single revision may involve multiple types, so these categories are not treated as mutually exclusive frequency counts.

These revision types clarify what keeping motion in the loop contributes to the refinement process. Motion revision directly changes the behavioral realization used to guide policy learning—for example, its posture, timing, support pattern, or whole-body coordination—while reward revision changes the optimization objective used to learn that behavior. Both remain constrained by the original task semantics, and they can be applied jointly when execution feedback indicates that changes to both the behavioral guidance and the learning objective are needed.

H Unguided GPT Analysis

H.1 Experimental Setup

The agent is provided with the robot, task specification, and the same actor template available to our framework, but receives no explicit instruction to generate a reference motion or to follow the motion–reward refinement procedure. The agent is therefore left to determine how to formulate the learning problem, including whether to introduce a motion reference and whether to revise it in response to policy execution. This ablation evaluates the contribution of explicitly prescribing execution-grounded motion–reward refinement, while holding access to the underlying learning tools and actor template fixed. For evaluation, we use the policy from the agent's final formal trial, following the selection rule in Appendix D.

As reported in the main paper, unguided GPT introduces reference motions in 16 of 19 tasks and subsequently revises them in seven. Thus, motion guidance can emerge even without being explicitly prescribed. This ablation therefore tests a stronger question than whether GPT can discover motion guidance at all: whether it can use execution feedback to refine that guidance effectively. We evaluate the resulting policies using the same user-rating protocol and physical execution met-

Table 11: User ratings on walking and non-walking tasks (↑), reported as mean ± sample standard deviation. Bold indicates the higher mean between GPT and RoboForge within each robot and task group.

Robot	Method	Walking	Non-walking
Go2	GPT	3.14 ± 1.36	3.86 ± 1.15
	RoboForge (Ours)	$\mathbf{3.76} \pm 0.91$	$\mathbf{4.33} \pm 0.78$
MicroDuck	GPT	3.61 ± 1.13	3.91 ± 1.06
	RoboForge (Ours)	$\mathbf{4.00} \pm 1.02$	$\mathbf{4.05} \pm 0.92$
Ostrich	GPT	3.21 ± 1.10	4.05 ± 0.83
	RoboForge (Ours)	$\mathbf{3.57} \pm 0.96$	$\mathbf{4.25} \pm 0.84$
Gorilla	GPT	2.57 ± 0.85	3.64 ± 0.93
	RoboForge (Ours)	$\mathbf{3.79} \pm 0.58$	$\mathbf{3.76} \pm 0.93$

Table 12: Physical execution metrics of GPT and RoboForge across robot embodiments (↓). Yaw error and slip are averaged over walking tasks, whereas action variation and joint acceleration are averaged over all tasks. Bold indicates the lower value.

Robot	Method	Yaw Err. (rad)	Action Var. (rad^2)	Joint Acc. (rad/s^2)	Foot-slip (m/s)
Go2	GPT	**0.00746**	0.0698	68.9	0.0905
	RoboForge (Ours)	0.0135	**0.000174**	**13.9**	**0.0354**
MicroDuck	GPT	0.0133	**0.00106**	**22.3**	0.0241
	RoboForge (Ours)	**0.00435**	0.0102	46.4	**0.0104**
Ostrich	GPT	0.0143	**0.00054**	17.7	0.0355
	RoboForge (Ours)	**0.0121**	0.000712	**16.9**	**0.00324**
Gorilla	GPT	0.0337	**0.0000328**	**7.83**	0.0175
	RoboForge (Ours)	**0.0035**	0.000482	10.8	**0.00353**

rics as for RoboForge, separating the emergence of a refinement strategy from its effectiveness in downstream policy learning.

H.2 User Ratings on Walk and Non-walk Tasks

Table 11 disaggregates the user-study ratings for GPT and RoboForge into walking and non-walking tasks for each robot. RoboForge obtains a higher mean rating in both task groups across all four embodiments. This consistent pattern shows that the higher user ratings are not driven by a single task category, but are observed across both walking and non-walking behaviors. The scores follow the aggregation procedure defined in Appendix F.

H.3 Physical Metrics by Robot

Table 12 reports the physical execution metrics for GPT and RoboForge, aggregated by robot according to the procedure in Appendix F. Lower values indicate better execution for all reported metrics.

Across the four embodiments and four reported metrics, RoboForge achieves the lower value in 10 of 16 robot–metric comparisons, including lower slip on all four embodiments. Although several individual metrics favor GPT, the overall pattern favors RoboForge across the measured execution properties. Together with the user ratings in Table 11, these results provide complementary evidence for the benefit of explicitly prescribed execution-grounded refinement across both task-level behavioral assessment and physical execution, rather than relying on any single metric.

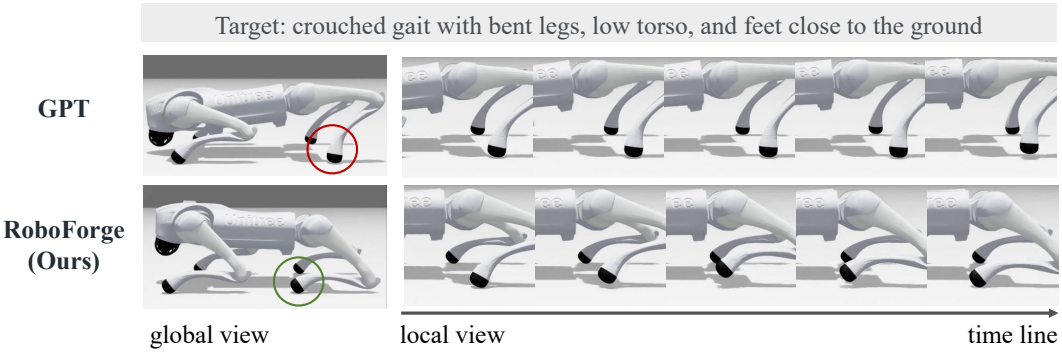


Figure 11: Final learned policy executions for the Go2 low-walk task. Figure 5 shows the corresponding reference-motion revisions, whereas this figure shows the final policies learned from those refined motions. RoboForge more clearly realizes the requested crouched gait, with distinct stepping, sustained low posture, and coordinated support transfer across the gait cycle.

H.4 From Motion Refinement to Policy Execution: A Low-Walk Case

The low-walk task provides a concrete case study connecting motion refinement to downstream policy learning. Figure 5 shows the corresponding reference-motion refinements. Both GPT and RoboForge identify foot sliding as a failure mode and increase swing-foot clearance, while our refinement further adjusts body motion and the supporting limbs to coordinate the swing-leg revision with posture maintenance and support transfer. The reference motions themselves, however, are intermediate learning guidance rather than the final output of the system. The central question is therefore whether these revisions provide more effective guidance for learning the requested behavior.

Figure 11 shows the corresponding final policy executions. Although both methods revise the reference in response to the same initial sliding failure, the learned policies realize the requested behavior differently. The policy trained with our refined motion more clearly combines distinct stepping with maintenance of the crouched posture and coordinated support transfer, whereas increasing swing-foot clearance alone does not fully resolve the whole-body gait. The two figures therefore connect the motion revision to its downstream consequence: our refinement does not merely produce a different reference trajectory, but produces guidance that leads to a more complete realization of the requested behavior by the learned policy.

This case illustrates the role of keeping motion in the execution-grounded refinement loop. Rollout feedback determines how the intermediate motion guidance should be revised, while the resulting policy execution determines whether that revision was useful for learning. Rather than treating the quality of the reference motion as an end in itself, our procedure treats motion as revisable learning guidance whose utility is evaluated through downstream policy behavior.

I Robot Generation Checks

I.1 Evaluation Setup

We evaluate four gorilla-inspired robots generated by GPT6 Astra, GPT5.6 Sol, GPT 5.5, and GPT5.6 Terra. The two tasks are forward swinging (`flat_palm_swing`) and hand waving (`flat_palm_wave`). GPT6 Astra executes the check programs for all four designs; the programs compute geometric, motion, and engineering measurements and determine the corresponding Pass/Fail outcomes.

The checks are grouped into **design and assembly**, **motion and task execution**, and **engineering**. The three groups contain 10, 7, and 6 independent checks, respectively. Each check is counted once and all counts are unweighted. *Static* checks use geometry or model properties, *reference* checks use the prescribed motion, and *execution* checks use recorded robot motion.

I.2 Check Definitions

Design and assembly. These checks progress from individual parts to the complete assembly. *Solid validity* verifies that each STEP part forms a valid single solid with positive volume. *Manufacturing mesh* checks that each printable mesh is closed, has positive volume, and is positioned correctly on the print bed. *Export volume fidelity* verifies that geometry is preserved from the source model to the STEP and STL exports. *BOM completeness* checks that printed parts, motors, files, and quantities are consistent. *Interface geometry* examines mounting holes, tool openings, and supporting material around the motor interface. *Print envelope* checks whether each part fits the available build volume in its printing orientation. *Fit tolerance* evaluates worst-case clearance at mating interfaces under dimensional variation. *Assembly and tool access* checks whether components can be inserted and tools can reach the required fasteners without interference. *Fastener interfaces* checks screw length, grip, engagement, bottoming, and head clearance at the mechanical connections. *Static assembly clearance* checks the neutral full assembly for unintended penetration between components.

Motion and task execution. These checks evaluate the articulated model and the robot behavior throughout the two tasks. *Mechanical model* verifies joint axes, order, limits, masses, inertias, the free base, and the eight actuated joints. *Forward swing* checks palm support, rear-foot lift, forward body motion, and stable landing. *Hand wave* checks left-hand lift, support from the other three pads, shoulder and elbow motion, and stable landing. *Motion interference* checks the CAD geometry for collisions through the recorded reference and executed poses. *Simulation tracking* measures joint and root-position errors between the executed and reference motions. *Actuator limits* checks peak torque, saturation, and joint-limit reaction torque. *Continuous clearance* checks whether positive geometric clearance remains throughout the motion after applying the specified manufacturing, assembly, calibration, tracking, elastic, thermal, model, and safety allowances.

Engineering checks. These checks evaluate load capacity and mechanical margins. *Local structural margin* evaluates critical sections under the loads recorded during motion. *Mass consistency* reconciles material estimates, link masses, and CAD and simulation totals. *Fastener length chain* verifies that the selected screws span the clamped parts, engage the threads, and avoid bottoming out. *Joint-limit margin* measures the distance between executed joint angles and the URDF travel limits. *Load-path strength* evaluates load-bearing parts and connections under support, landing, and emergency-stopping loads. *Cable routing* checks clearance, bending radius, and twist throughout the joint range.

I.3 Check Results

Table 13: Pass/Fail results for all robot-generation checks. The category counts in the main paper can be recomputed directly from this table.

ID	Check	GPT6 Astra	GPT5.6 Sol	GPT 5.5	GPT5.6 Terra
Design and assembly					
DA01	Solid validity	Pass	Pass	Fail	Pass
DA02	Manufacturing mesh	Pass	Fail	Fail	Fail
DA03	Export volume fidelity	Pass	Pass	Pass	Pass
DA04	BOM completeness	Pass	Pass	Pass	Pass
DA05	Interface geometry	Pass	Fail	Fail	Fail
DA06	Print envelope	Pass	Pass	Pass	Pass
A05	Fit tolerance	Fail	Fail	Fail	Fail
A06	Assembly and tool access	Fail	Fail	Fail	Fail
A07	Fastener interfaces	Fail	Fail	Fail	Fail
A09	Static assembly clearance	Fail	Fail	Fail	Fail
Motion and task execution					
DB01	Mechanical model	Pass	Pass	Pass	Pass
DB02	Forward swing	Pass	Fail	Fail	Fail

ID	Check	GPT6 Astra	GPT5.6 Sol	GPT 5.5	GPT5.6 Terra
DB03	Hand wave	Pass	Fail	Fail	Fail
DB04	Motion interference	Pass	Fail	Fail	Fail
DB05	Simulation tracking	Fail	Fail	Fail	Fail
DB06	Actuator limits	Pass	Fail	Fail	Fail
B05	Continuous clearance	Fail	Fail	Fail	Fail
Engineering checks					
DC01	Local structural margin	Pass	Fail	Fail	Fail
DC02	Mass consistency	Pass	Fail	Fail	Fail
DC03	Fastener length chain	Pass	Fail	Fail	Fail
DC04	Joint-limit margin	Pass	Fail	Fail	Pass
C02	Load-path strength	Fail	Fail	Fail	Fail
C05	Cable routing	Fail	Fail	Fail	Fail

I.4 Evaluation Parameters

Fit and assembly. Printed dimensions use a ± 0.2 mm size tolerance, hole position uses a 0.2 mm radial bound, purchased-part dimensions use ± 0.05 mm, and assembly position uses a 0.1 mm bound. For the M2 interfaces, the worst-case radial clearance is

$$\frac{d_h - 0.2 - d_s - 0.05}{2} - 0.2 - 0.1,$$

which must remain positive. Tool access is evaluated with a radius-2 mm, length-30 mm bit and a radius-6 mm, length-60 mm handle approaching coaxially with the screw.

Task completion. For forward swinging, both front palms must support the robot at normal force > 0.01 N each while both rear feet remain more than 2 mm above the ground for at least 0.5 s. The body must advance by at least 10 mm during the same interval. For hand waving, the left palm must remain more than 50 mm above the ground while the other three pads each provide normal support > 0.01 N for at least 1 s. The left shoulder and elbow excursions must each reach at least 0.1 rad. Both tasks require a final stable landing lasting at least 1 s, with four-pad support, upright body orientation, low linear and angular velocity, and no non-pad ground contact.

Tracking and actuation. Simulation tracking requires a maximum joint error $\leq 5^\circ$ and a maximum root-position error ≤ 5 mm over both motions. Actuator checks require peak torque ≤ 0.100001 N m, saturation fraction ≤ 0.001, and joint-limit reaction torque ≤ 0.001 N m.

Continuous clearance. The translational clearance budget includes 0.4 mm for manufacturing, 0.1 mm for assembly, 0.2 mm for elasticity, 0.05 mm for thermal effects, 0.05 mm for model error, and 0.2 mm for safety. Angular allowances are 0.5° for calibration and 5° for tracking. For a point at farthest radius r from a joint axis, an angular allowance θ contributes $2r\sin(\theta/2)$ to the displacement budget.

Structural loads. The local structural check uses an allowable stress of 8 MPa, a stress-concentration factor of 2, and an additional reserve factor of 3. The local margin is

$$\frac{8}{6\sigma_{\mathrm{eq}}},$$

and must be at least one. Load-path checks cover support, landing, and emergency stopping. Landing uses velocity 0.5 m/s and stopping distance 0.005 m, giving

$$F_{\mathrm{avg}} = m\left(g + \frac{v^2}{2s}\right).$$

Emergency stopping uses the recorded velocity over 0.1 s. The allowable stresses are 8 MPa for bulk material, 4 MPa transverse to print layers, and 4 MPa in connection shear.

Mass accounting. Mass accounting reconciles material estimates, link assignments, and whole-robot CAD and simulation masses. Material densities are 1.24, 1.27, and 1.21 g/cm^3 for PLA, PETG, and TPU, respectively. Each motor is modeled as 18 g, with 16.8 g assigned to the stator and 1.2 g to the rotor. The controller, battery, horn supports, and fasteners/wiring contribute 45 g, 54 g, 16 g, and 25 g, respectively.

Fastener length chains. For the horn connection, screw engagement must satisfy

$$L_h - g_h \in (0, 3]\ \mathrm{mm}.$$

For the front and rear motor-case connections, the engagement lengths are

$$L_c - g_c - 3.5 \quad \text{and} \quad L_c - g_c - 4.5,$$

respectively, and both must lie strictly between 0 and 15 mm.

Joint travel and cable routing. The minimum distance to the configured joint limits must remain at least 1° over all recorded frames of both tasks. Cable routing uses a 180 mm cable with a 1.5 mm envelope radius. The route must maintain a bend radius of at least 18 mm and a twist no greater than 90° per 100 mm.