MimicAgent: Learning Quadruped Skills via Text-to-Trajectory Generation

Lucky Kant Nayak[*], **Narayanan Palghat Parameswaran**[*], **Neehar Peri, Deva Ramanan**
Carnegie Mellon University

Abstract: We present MimicAgent, a text-to-trajectory generation framework for learning dynamic quadruped skills. Although reward shaping is extensively used when training quadruped policies, navigating the resulting reward landscape is notoriously difficult, requiring hours of "graduate student descent". Eureka attempts to automate reward design with LLMs, but we find that it struggles to generalize across diverse skills and morphologies. Motivated by the success of example-guided RL for humanoids, we revisit skill learning from demonstrations for quadrupeds. Unlike humanoids, which can exploit large-scale motion capture datasets for learning, quadrupeds lack such reference motion data. We make the observation that manually keyframing quadruped reference motions can be more intuitive than reward shaping; in particular, we find that rather coarse and even dynamically-infeasible motions can still be effective reference targets for example-guided RL. However, manual keyframing is still too cumbersome to create large-scale skill libraries. To address this challenge, we propose an LLM-based pipeline that generates kinematically feasible quadruped trajectories for diverse skills. Although these trajectories are not dynamically feasible, we show that they are sufficient to train successful policies. Across all evaluated skills, human raters often prefer policies generated by MimicAgent over those produced by Eureka. Please see the supplement for videos of sim-to-real results.

Keywords: Agentic Trajectory Generation, LLM-based Code Synthesis

1 Introduction

In recent years, learning-based locomotion policies have demonstrated impressive agility and robustness across a wide range of behaviors [1, 2] and terrains [3]. However, training such RL policies typically relies on carefully tuned rewards. For complex and highly dynamic skills, designing bespoke reward functions is notoriously difficult and often requires extensive trial-and-error [4, 5, 6]. This challenge has motivated recent efforts to eliminate manual reward engineering entirely.

Automating Reward Design with LLMs. Recent work has explored using LLMs to automate reward design [7, 8, 9, 10]. For example, Eureka [8] leverages coding LLMs to generate and iteratively refine reward functions from natural language task descriptions. While promising, these methods still depend on hand-engineered success metrics to evaluate candidate rewards and often struggle to generalize across diverse skills and morphologies. In practice, generated reward functions are brittle and often don't accurately follow natural language guidance, particularly for underexplored robot embodiments such as wheeled quadrupeds (Table 1).

Learning from Examples. An alternative to reward-centric RL is learning from demonstrations. Example-guided RL has been a key enabler for recent progress in humanoid locomotion, where motion capture datasets [11, 12, 13, 14, 15, 16, 17, 18], or manually annotated keyframes [19, 20, 21] provide rich supervision. However, quadrupeds lack such large-scale, publicly available datasets. Recent works [22, 23] curate libraries of simple quadruped motion patterns like walking, trotting, and

[*]Equal Contribution

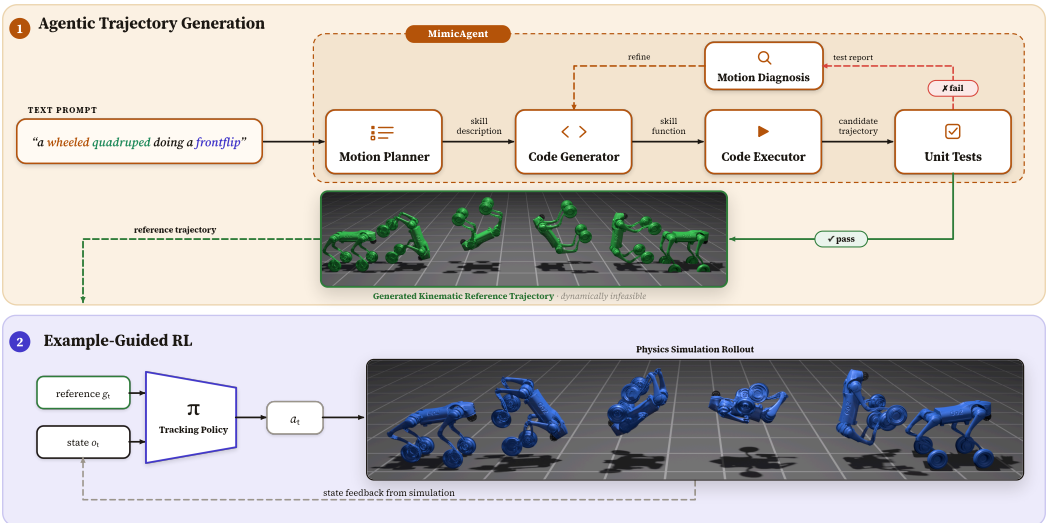


Figure 1: **Overview.** MimicAgent uses LLM coding agents within a recursive self-improvement loop to generate coarse reference trajectories (shown in green) from text prompts (Step 1). Our agentic pipeline converts short text prompts into detailed skill descriptions, translates these into executable functions that describe base motion and foot trajectories over time, and outputs reference trajectories. We show that these coarse (and even dynamically-*in*feasible) motions are effective reference targets for example-guided RL, and that generating plausible trajectories with an LLM is often easier than designing robust reward functions for the same behaviors (Step 2).

galloping. However, such efforts are orders of magnitude smaller and less diverse than comparable humanoid datasets. This data scarcity limits the applicability of demonstration-based methods despite their strong empirical performance.

Agentic Trajectory Generation. To address this gap, we propose MimicAgent, an LLM-based agentic pipeline that directly generates text-guided example trajectories for policy learning. *Our key observation is that it is far easier for a human – and by association, an LLM – to generate reference motions than to generate reward functions.* Given a natural language skill description, MimicAgent produces executable code defining base motion and foot trajectories. This code is then executed in a MuJoCo engine using forward kinematics to produce kinematically feasible trajectories, without running a full physics simulator. Although the resulting trajectories are not dynamically feasible, they provide structured, task-relevant supervision that can be consumed by a downstream imitation-based RL pipeline like DeepMimic [19].

Contributions. We present three major contributions.

1. *MimicAgent Framework.* We introduce an LLM-based text-to-trajectory framework specifically designed for learning dynamic quadruped skills. Further, we show that this framework extends to diverse morphologies in the supplement.

2. *Learning from Dynamically Infeasible Reference Trajectories.* We demonstrate that although MimicAgent generates dynamically infeasible trajectories, these reference trajectories are sufficient for training successful sim-to-real policies.

3. *Comparing Reference Trajectory Synthesis vs. Reward Shaping.* Our experiments highlight the limitations of prior LLM-based approaches that rely on reward shaping, and demonstrate the effectiveness of using LLMs for trajectory synthesis instead.

2 Related Works

Quadrupedal Locomotion has matured significantly in recent years [4, 3, 5, 2, 1, 24]. Early work primarily focused on training legged robots to execute complex contact sequences to achieve different locomotion gaits. A common approach is to use model predictive control (MPC) to track predefined contact patterns [25, 26], enabling canonical gaits such as trotting [27], pacing [28], bounding [29], and galloping [30], as well as non-conventional gaits with customized contact timings [31, 32]. Despite their success, MPC-based approaches require hand-designed reference trajectories and incur high computational costs, limiting behavioral diversity. Recent work extends these ideas to agile bipedal quadrupedal motions [33] and whole-body loco-manipulation [34, 35]. In contrast, our approach leverages example-guided reinforcement learning, which can be more intuitive than designing foot contact sequences.

Text-to-Robot Control is an active area of research for both quadrupeds [36, 37, 38] and humanoids [39, 40, 9, 41]. Early methods relied on structured text templates or NLP tools like parse trees to extract constraints and generate robot trajectories [42, 43, 44]. Other approaches used representation learning to train language-conditioned policies that map free-form instructions directly to actions [45, 46, 47, 48]. However, such approaches require large annotated datasets with paired text and robot control, which is difficult to collect for diverse locomotion behaviors. More recent methods prompt LLMs to generate robot code, bridging the gap between language and motor control via intermediate representations like high-level plans, primitive skills, or trajectories [49, 50, 51, 52, 53]. Inspired by prior work, we leverage LLMs to directly generate reference trajectories for imitation learning.

3 MimicAgent: Learning Dynamic Skills via Trajectory Synthesis

MimicAgent aims to automate motion imitation learning by replacing manual reward engineering with an agentic motion synthesis pipeline. Given a natural language description of a quadruped skill (e.g. `wheeled quadruped doing a frontflip`), MimicAgent generates reference motion trajectories that can be directly used by imitation-based RL algorithms. Rather than navigating a complex and brittle reward landscape, MimicAgent leverages the observation that generating a plausible reference trajectory is often significantly easier than designing a robust reward function for the same behavior.

Agentic Motion Synthesis Pipeline. MimicAgent is an iterative pipeline that decomposes motion synthesis into *motion planning, code generation, execution*, and *refinement* (Algorithm 1).

The input to the system is a natural language skill description, which is processed by the motion planner agent to produce a detailed motion plan encoding the temporal and kinematic structure of the skill. Each motion plan describes the overall motion intent and stability strategy, the decomposition into sub-phases with associated leg contact states, coordination patterns across leg groups, base linear and angular velocity commands in the body frame, and qualitative leg trajectories relative to the body. It also provides handoff notes for the code generation agent for controller implementation.

Each skill is parameterized by a normalized phase variable $\phi \in [0, 1]$, mapping motions of arbitrary duration into a unified phase-indexed form. A reference trajectory is defined as a sequence of keyframes sampled at discrete phase steps $\phi_t = t/T$, denoted as $\boldsymbol{\tau} = \{s_0, s_1, \ldots, s_T\}$. At each phase step t, the keyframe state is: $s_t = [\mathbf{p}_t^{\text{base}}, \boldsymbol{\alpha}_t^{\text{base}}, \mathbf{q}_t]$, where $\mathbf{p}_t^{\text{base}} \in \mathbb{R}^3$ represents the robot base 3D position, $\boldsymbol{\alpha}_t^{\text{base}} \in \mathbb{R}^4$ is the base orientation represented as a unit quaternion, and $\mathbf{q}_t \in \mathbb{R}^n$ is the vector of n joint rotations. This representation enables a compact and interpretable parameterization capable of expressing a wide range of locomotion behaviors.

Code Synthesis and Trajectory Execution. Given the motion plan, a coding agent synthesizes an executable program that defines the temporal functions and sequence for base and foot trajectories. This generated code is executed via a forward kinematics engine within a MuJoCo simulator. Importantly, we do not perform any physics simulation. As a result, the synthesized trajectories are

Algorithm 1 MimicAgent Trajectory Generation

```
def mimic_agent(skill_description, N=4):
    iters = 0; success = False; state = [skill_description]
    motion_plan = motion_planner(skill_description)
    state.append(motion_plan)
    skill_code = coder_agent(motion_plan)
    state.append(skill_code)
    while not success and iters < N:
        trajectory = code_executor(skill_code)
        success, result = unit_test(trajectory)
        state.append(result)
        if success:
            return trajectory
        llm_feedback = motion_diagnosis(state)
        state.append(llm_feedback)
        skill_code = code_modifier(skill_code, state, feedback=None)
        iters += 1
    return success
```

kinematically consistent but generally violate dynamic constraints. MimicAgent explicitly embraces this tradeoff, delegating dynamic feasibility to the reinforcement learning stage.

Task-Agnostic Unit Tests. To validate generated trajectories without introducing task-specific biases, MimicAgent employs a set of *task-agnostic unit tests*. Rather than optimizing task rewards, MimicAgent filters generated motions using task-agnostic constraints, enabling scalable reference trajectory generation without manually specifying task-specific success criteria. These tests are designed to reject physically implausible or degenerate trajectories while remaining broadly applicable across skills. Each unit test computes a scalar reward, and the aggregate score determines whether a trajectory passes validation. The unit tests include:

- *Base Height Envelope.* The base height is constrained to remain within a reasonable envelope relative to the ground. Heights below 0.1 meters or those exceeding twice the standing base height are penalized. This test returns a binary reward of $+1$ or -1.
- *Joint Violation Penalty.* Joint limit violations are penalized proportional to their frequency. The reward is computed as $r_{\text{joint}} = -1 \times N_{\text{violated}}$, where N_{violated} denotes the number of joints exceeding their limits.
- *Ground Penetration.* Foot-ground penetration is penalized based on severity. Trajectories in which all four feet penetrate the ground incur a large penalty, while partial penetration results in a smaller penalty. The reward is bounded between $+1$ and -1.
- *Feet Air-Time Constraint.* For non-aerial skills, at least one foot must remain in contact with the ground for half of the motion cycle. This constraint prevents degenerate motions that unrealistically leave the ground for extended durations.

Importantly, these unit tests are skill-agnostic and do not encode task success, performance objectives, or stylistic preferences. They serve purely as structural validity checks, ensuring that generated trajectories are suitable as imitation references. In contrast, Eureka [8] relies heavily on carefully designed task-specific success functions to generate reward signals. Defining such success criteria is a time-intensive process and is often a significant bottleneck for learning large skill libraries. Motivated by this limitation, MimicAgent adopts a task-agnostic validation strategy that avoids task-specific success criteria entirely.

Iterative Refinement. If a trajectory fails validation, a diagnostic agent analyzes the failure in the context of the full system state, including the skill description, motion plan, generated code, and unit test outcomes. The diagnostic agent produces natural language feedback identifying likely failure modes. A code modification agent then revises the trajectory-generating program accordingly.

This process iterates until a trajectory passes all unit tests or a fixed iteration budget is reached. Valid trajectories are returned for downstream imitation learning. The code modification agent can *optionally* also take human feedback to improve code generation quality.

By shifting the focus from reward optimization to reference trajectory synthesis, MimicAgent avoids the brittle and time-consuming process of manual reward engineering. Unlike LLM-based reward design methods that require repeated policy training and hand-crafted evaluation criteria, MimicAgent generates supervision directly in trajectory space. This makes the framework efficient, interpretable, and naturally extensible to diverse quadruped skills.

4 Experiments

In this section, we briefly describe our evaluation metrics, provide an empirical analysis of our baselines, and ablate the annotation time and impact of reference motion quality between human annotated keyframes and our agentic trajectory synthesis pipeline.

Metrics. We quantitatively compare manual keyframing, Eureka [8], and MimicAgent with a 57 person user study conducted through Cint. We include additional details about the user study in the supplement. Participants were presented with pairs of textual task descriptions and videos of trained policies corresponding to that task and were asked to rate how well the video of policy rollouts follows the task description on a 5-point Likert scale. Videos were presented in a randomized order to eliminate ordering bias and participants were not informed which method was used to generate each video. In addition, we compute the mean squared error between the reference trajectories and the trajectories produced by the learned policy for both human annotated keyframes and MimicAgent's keyframes to measure how faithfully the learned policy follows the demonstration. This allows us to measure the quality and dynamic feasibility of the reference motion.

Baselines. We compare MimicAgent with policies trained by Eureka [8] and human annotated keyframes. Eureka frames reward design as a program synthesis problem: given a textual skill description and access to environment code (e.g., observations, episode termination conditions, and success function), an LLM generates candidate reward functions. Multiple reward functions are sampled per iteration, trained in parallel, and evaluated using predefined success criteria, with an evolutionary search loop selecting and refining the best-performing rewards. Next, we also evaluate DeepMimic [19] policies learned from human annotated keyframes. To simplify the annotation process, we created a custom annotation tool with Claude Sonnet 4.5 to control foot positions and root body pose using click-and-drag mechanics for coarse control and sliders for fine-grained control (see supplement for details). We evaluate all baselines on seven skills, including Go2 Trot, Go2 Bounding, Go2-W Front Flip (FF), Go2-W Side Flip (SF), Go2-W Aerial Crossover (AC), Go2-W Reverberating Yaw Pulse (RYP), and the Go2-W Crab Diagonal Scuttle (CDS). We describe each skill in the supplement.

Comparison with State-of-the-Art. We conduct a user study (n = 57) comparing policies generated from human annotated keyframes, Eureka, and MimicAgent. Participants consistently prefer MimicAgent policies for Trotting (+2.0 vs. Eureka), Side Flip (+3.5 vs. Eureka), Front Flip (+3.5 vs. Eureka),

Table 1: **Comparison to State-of-the-Art.** We conduct a user study (n=57) to compare policies generated by human-annotated keyframes, Eureka, and MimicAgent. We find that users prefer MimicAgent policies for Trotting, Side Flip, Front Flip, Aerial Crossover, and Reverberating Yaw Pulse. Users prefer policies generated by human-annotated keyframes for Bounding, though all policies receive low ratings. Lastly, users prefer Eureka for Crab Diagonal Scuttle. Higher is better.

Robot	**Go2**		**Go2-W**				
Method ↓ / Skill →	**Trot**	**Bound**	**Side Flip**	**Front Flip**	**Aerial Crossover**	**CDS**	**RYP**
Keyframe Tool	2.9 ± 1.4	$\mathbf{3.5 \pm 1.4}$	3.5 ± 1.2	4.2 ± 1.0	2.3 ± 1.3	2.5 ± 1.2	1.8 ± 1.2
Eureka	2.6 ± 1.5	2.8 ± 1.4	1.4 ± 1.1	1.1 ± 0.3	1.9 ± 1.1	2.1 ± 1.1	$\mathbf{4.3 \pm 1.4}$
MimicAgent	$\mathbf{4.4 \pm 1.0}$	3.2 ± 1.4	$\mathbf{4.9 \pm 0.3}$	$\mathbf{4.6 \pm 0.8}$	$\mathbf{4.2 \pm 1.0}$	$\mathbf{4.5 \pm 0.5}$	2.6 ± 1.4

Table 2: **Comparison of Time-to-Annotation.** We evaluate the time to annotation (in minutes) between human-annotated keyframes and MimicAgent keyframes. Notably, MimicAgent is consistently faster, particularly for complex skills like crab diagonal scuttle (CDS) and reverberating yaw pulse (RYP). Lower is better.

Robot	**Go2**		**Go2-W**				
Method ↓ / Skill →	**Trot**	**Bound**	**Side Flip**	**Front Flip**	**Aerial Crossover**	**CDS**	**RYP**
Keyframe Tool	2.6	4.5	4.5	4.4	7.6	14.5	12.4
MimicAgent	**1.1**	**1.2**	**2.9**	**3.2**	**3.6**	**1.6**	**1.8**

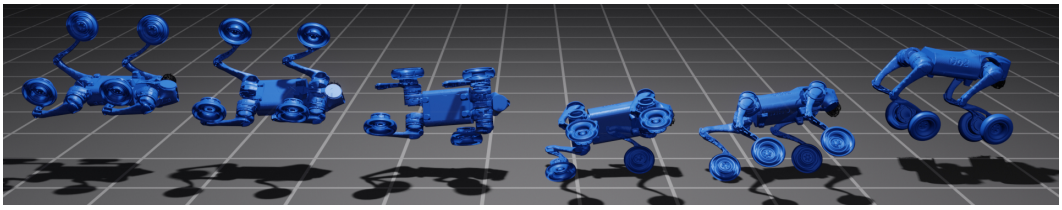
Table 3: **Comparison with Reference Motion.** We evaluate DeepMimic policies trained in IsaacGym with human-annotated keyframes and MimicAgent keyframes against their respective reference trajectories. Across all skills and metrics, MimicAgent policies achieve lower mean squared error on both pose-level errors (root and body position and rotation) and motion-level errors (linear and angular velocities).

	Robot	**Go2**		**Go2-W**				
Method	**Metric ↓ / Skill →**	**Trot**	**Bound**	**SF**	**FF**	**AC**	**CDS**	**RYP**
Keyframe Tool	E_{RP}	0.170	0.076	0.199	0.099	0.020	0.044	0.041
	E_{RR}	0.177	0.208	0.238	0.208	0.025	0.201	0.142
	E_{JR}	0.360	0.320	0.484	**0.507**	0.396	0.033	0.025
	E_{JV}	4.313	3.868	8.380	15.183	6.823	6.171	4.721
	E_{LV}	0.251	0.214	0.548	0.287	**0.035**	0.261	0.126
	E_{AV}	0.918	0.517	1.883	**1.126**	0.233	0.797	1.172
MimicAgent	E_{RP}	**0.052**	**0.049**	**0.077**	**0.089**	**0.013**	**0.035**	**0.013**
	E_{RR}	**0.046**	**0.114**	**0.207**	**0.160**	**0.024**	**0.032**	**0.089**
	E_{JR}	**0.109**	**0.224**	**0.382**	0.528	**0.373**	**0.009**	**0.018**
	E_{JV}	**0.646**	**1.580**	**5.299**	**12.871**	**4.761**	**1.609**	**2.348**
	E_{LV}	**0.093**	**0.136**	**0.295**	**0.264**	0.044	**0.124**	**0.047**
	E_{AV}	**0.176**	**0.353**	**1.869**	1.697	**0.211**	**0.264**	**0.455**

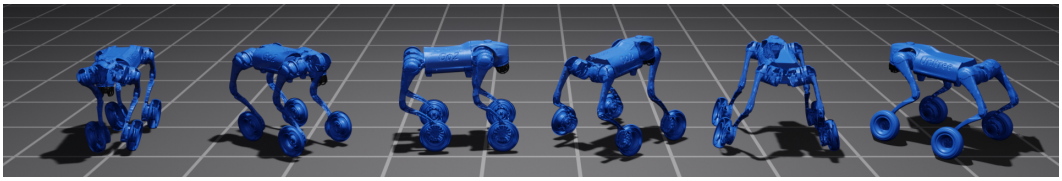
Aerial Crossover (+2.3 vs. Eureka), and Reverberating Yaw Pulse (+2.4 vs. Eureka). For Bounding, users favor policies generated from human-annotated keyframes, although all approaches received relatively low ratings. Eureka achieves the highest score on Crab Diagonal Scuttle (+1.7 vs. MimicAgent). Notably, Eureka still relies on carefully specified task-specific success metrics and exhibits limited robustness when scaling to diverse, stylistically different skills. Although MimicAgent does not require task-specific unit tests, we hand-engineer success criteria for Eureka's policies to give it a better chance of succeeding. Despite this, performance degrades for many wheeled quadruped skills (UniTree Go2-W), likely due to the increased complexity of the joint space. We provide videos for all policies used in this user study in the supplement.

Time-to-Annotation. We compare the time required for generating human-annotated keyframes and MimicAgent generated keyframes in Table 2. We find that MimicAgent consistently requires less time than manually annotated keyframes. The first five skills (trot, bounding, side flip, front flip, aerial crossover) are intentionally limited to relatively simple motions (e.g., unidirectional velocity or in-place yaw). In contrast, crab diagonal scuttle and reverberating yaw pulse represent more complex skills that incorporate asymmetric leg movements coupled with base rotations, which are difficult and time-consuming to keyframe manually. We posit that as skill complexity increases, human annotation time will grow significantly faster than MimicAgents skill generation time.

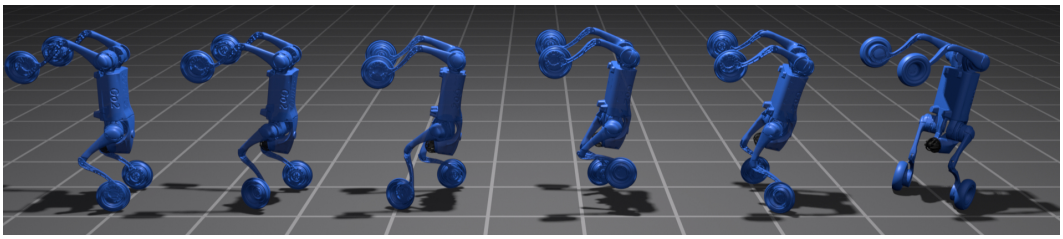
Comparison with Reference Motion. We compare DeepMimic policies trained on reference trajectories generated with our manual keyframing tool (133 frames) and MimicAgent (133 frames) in Table 3. Policies trained with MimicAgent reference trajectories consistently achieved lower tracking error across almost all skills and evaluation metrics, suggesting that the original reference trajectories were smoother and more dynamically feasible than human-annotated trajectories. We track errors for root position in the global frame (E_{rp}, m), root rotation (E_{rr}, rad), joint rotation (E_{jr}, rad), joint



(a) Backward lateral locomotion with cartwheel-style leg rotations and circular trajectories.



(b) Legs traces a diamond-shaped pattern on the ground.



(c) Front legs push-glide-push sequence for skating forward.

Figure 2: **Qualitative Skill Synthesis Results.** We present representative skills synthesized by MimicAgent. Each subfigure shows a different skill (with the initial prompt in the subcaption); please see the supplement for trajectory rollouts.

Figure 3: **Sim-to-Real Deployment.** We successfully deploy a two-legged handstand walking policy, generated via MimicAgent, directly onto the Go2-W quadruped. Despite being trained entirely in simulation, the policy achieves a stable forward walking gait at 0.4 m/s. See the supplementary material for full details regarding sim-to-real transfer and hardware deployment.

velocity (E_{jv}, rad/s), root linear velocity (E_{lv}, m/s), and root angular velocity (E_{av}, rad/s). This improvement is most pronounced for highly dynamic skills such as side flips, front flips, and aerial crossovers, where policies trained on manually keyframed trajectories exhibit significantly larger velocity and angular velocity errors. In contrast, policies trained on MimicAgent references result in improved motion timing and smoother rotational dynamics during policy learning. Even for simpler skills such as trotting and bounding, MimicAgent yields lower global pose error, reflecting higher reference motion quality. These results indicate that improved reference trajectory quality directly reduces downstream imitation error, demonstrating the effectiveness of MimicAgent as a scalable alternative to manual keyframing for imitation-based RL.

Qualitative Results. We visualize simulated policies trained with MimicAgent reference trajectories in Figure 2 and Figure 5. In Figure 2, we find that MimicAgent can successfully learn dynamic skills in simulation (more videos in the supplement). We highlight that MimicAgent can generate (to the best of our knowledge) never-before-seen skating behavior. Although the reference trajectory does not capture the nuances of skating on underactuated wheels while balancing on two legs, training

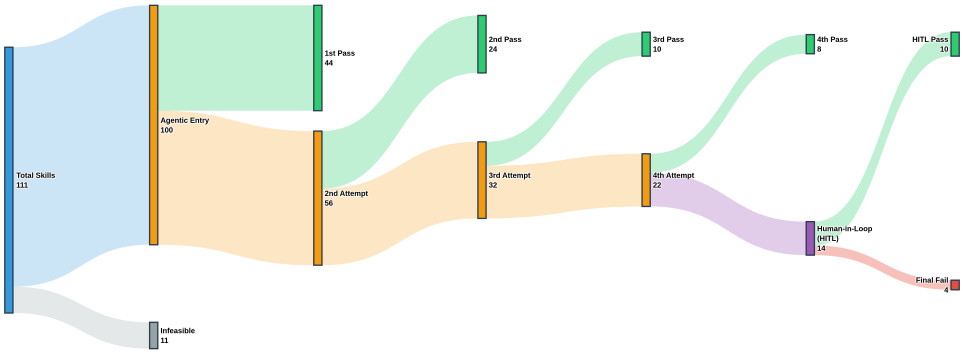


Figure 4: **Automatic Skill Generation.** We use an LLM to propose diverse locomotion skills according to our taxonomy (see supplement). Given an initial set of 111 candidate skills, Mimic Agent generates 86 successes after four refinement stages. Adding in optional human-in-the-loop feedback adds 10 more for a total of 96. A skill is considered successful if it passes all unit tests.

a policy with DeepMimic *does* capture such subtle dynamics. In Figure 5, we show that walking on two legs is actually transferreable to a real robot. We stress that the original reference trajectory generated by MimicAgent almost certainly is not stable, yet the learned RL policy is able to maintain balance while walking on challenging outdoor terrain.

Automatic Skill Generation. In addition to manually prompting MimicAgent, we can leverage LLMs to automatically generate large-scale skill libraries. Figure 4 visualizes the success rates of this automated skill generation process. We first prompt an LLM to propose candidate skills based on our taxonomy (see supplement), resulting in 111 total skills. Of these, 11 are deemed infeasible, while 78 successfully converge after four iterations. We further show that incorporating non-expert human feedback improves performance, increasing the number of successful skills to 96. Notably, this feedback only provides high-level guidance rather than specific code modifications or hyperparameter tuning. Success rate is determined by the number of skills that pass all unit tests.

Analysis of Failure Cases. Policies trained for jumping and midair 180° turns failed to learn the intended behaviors. Instead, the robot exploited the task by completing the turn on the ground, likely due to incorrect or near-static foot trajectories. We posit that this is due to the feet air-time constraint preventing proper lift-off. These issues align with recurring failure patterns observed in practice, where agents become trapped in error loops (e.g., alternating between ground penetration and air penalties). This suggests a need for more specialized tool calls and verification checks, such as explicit validation of foot placement and contact conditions.

Limitations and Future Work. While task-agnostic unit tests provide baseline coverage, we observe clear qualitative gains from human-in-the-loop feedback. This suggests that skill-specific feedback could further reduce failure rates. Future work should use LLMs to automatically generate skill-specific unit tests and provide targeted feedback during trajectory synthesis.

5 Conclusion

In this paper, we present MimicAgent, an LLM-based agentic pipeline for quadruped trajectory generation. By leveraging coding agents to synthesize coarse yet kinematically feasible reference motions, MimicAgent sidesteps the brittleness and manual effort of reward shaping while avoiding the need for large-scale motion capture datasets. Our results show that, despite being dynamically infeasible, these automatically generated trajectories are sufficient for training robust and expressive behaviors across a diverse set of skills. Human evaluations consistently favor policies trained with MimicAgent over those produced by Eureka, highlighting our method's improved motion quality and language following abilities.

References

[1] S. Mahankali, C.-C. Lee, G. B. Margolis, Z.-W. Hong, and P. Agrawal. Maximizing quadruped velocity by minimizing energy. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11467–11473, 2024. doi:10.1109/ICRA57147.2024.10609983.

[2] G. B. Margolis and P. Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

[3] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[4] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[5] I. Nahrendra, B. Yu, and H. Myung. Dreamwaq: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning. *arXiv preprint arXiv:2301.10602*, 2023.

[6] J. Long, Z. Wang, Q. Li, J. Gao, L. Cao, and J. Pang. Hybrid internal model: Learning agile legged locomotion with simulated robot response. *arXiv preprint arXiv:2312.11460*, 2023.

[7] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

[8] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[9] J. Cui, T. Liu, Z. Meng, J. Yu, R. Song, W. Zhang, Y. Zhu, and S. Huang. Grove: A generalized reward for learning open-vocabulary physical skill. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 15781–15790, 2025.

[10] N. Turcato, M. Iovino, A. Synodinos, A. Dalla Libera, R. Carli, and P. Falco. Towards autonomous reinforcement learning for real-world robotic manipulation with large language models. *IEEE Robotics and Automation Letters*, 2025.

[11] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pages 5442–5451, Oct. 2019.

[12] A. Allshire, H. Choi, J. Zhang, D. McAllister, A. Zhang, C. M. Kim, T. Darrell, P. Abbeel, J. Malik, and A. Kanazawa. Visual imitation enables contextual humanoid control. *arXiv preprint arXiv:2505.03729*, 2025.

[13] S. Xu, H. Y. Ling, Y.-X. Wang, and L.-Y. Gui. Intermimic: Towards universal whole-body control for physics-based human-object interactions. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 12266–12277, 2025.

[14] T. He, Z. Luo, W. Xiao, C. Zhang, K. Kitani, C. Liu, and G. Shi. Learning human-to-humanoid real-time whole-body teleoperation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8944–8951. IEEE, 2024.

[15] M. Ji, X. Peng, F. Liu, J. Li, G. Yang, X. Cheng, and X. Wang. Exbody2: Advanced expressive humanoid whole-body control. *arXiv preprint arXiv:2412.13196*, 2024.

[16] R. Yu, H. Park, and J. Lee. Human dynamics from monocular video with dynamic camera movements. *ACM Trans. Graph.*, 40(6), Dec. 2021. ISSN 0730-0301. doi:10.1145/3478513.3480504. URL `https://doi.org/10.1145/3478513.3480504`.

[17] Z. Luo, J. Cao, J. Merel, A. Winkler, J. Huang, K. Kitani, and W. Xu. Universal humanoid motion representations for physics-based control. *arXiv preprint arXiv:2310.04582*, 2023.

[18] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions On Graphics (TOG)*, 37(6):1–14, 2018.

[19] X. B. Peng, P. Abbeel, S. Levine, and M. Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

[20] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021.

[21] Z. Zhang, S. Bashkirov, D. Yang, M. Taylor, and X. B. Peng. Add: Physics-based motion imitation with adversarial differential discriminators. *arXiv preprint arXiv:2505.04961*, 2025.

[22] G. Bellegarda, M. Shafiee, and A. Ijspeert. Allgaits: Learning all quadruped gaits and transitions. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15929–15935. IEEE, 2025.

[23] S. Tirumala, A. Sagi, K. Paigwar, A. Joglekar, S. Bhatnagar, A. Ghosal, B. Amrutur, and S. Kolathaya. Gait library synthesis for quadruped robots via augmented random search. *arXiv preprint arXiv:1912.12907*, 2019.

[24] L. Smith, J. C. Kew, T. Li, L. Luu, X. B. Peng, S. Ha, J. Tan, and S. Levine. Learning and adapting agile locomotion skills by transferring experience. *arXiv preprint arXiv:2304.09834*, 2023.

[25] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter. Feedback mpc for torque-controlled legged robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4730–4737. IEEE, 2019.

[26] O. Villarreal, V. Barasuol, P. M. Wensing, D. G. Caldwell, and C. Semini. Mpc-based controller with terrain insight for dynamic legged locomotion. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2436–2442. IEEE, 2020.

[27] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018. doi:10.1109/IROS.2018.8594448.

[28] M. H. Raibert. Trotting, pacing and bounding by a quadruped robot. *Journal of Biomechanics*, 23:79–98, 1990. ISSN 0021-9290. doi:https://doi.org/10.1016/0021-9290(90)90043-3. URL https://www.sciencedirect.com/science/article/pii/0021929090900433. International Society of Biomechanics.

[29] P. Eckert, A. Sprwitz, H. Witte, and A. J. Ijspeert. Comparing the effect of different spine and leg designs for a small bounding quadruped robot. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3128–3133, 2015. doi:10.1109/ICRA.2015.7139629.

[30] D. Marhefka, D. Orin, J. Schmiedeler, and K. Waldron. Intelligent control of quadruped gallops. *IEEE/ASME Transactions on Mechatronics*, 8(4):446–456, 2003. doi:10.1109/TMECH.2003.820001.

[31] H. Li, T. Zhang, W. Yu, and P. M. Wensing. Versatile real-time motion synthesis via kino-dynamic mpc with hybrid-systems ddp. *arXiv preprint arXiv:2209.14138*, 2022.

[32] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018. doi:10.1109/LRA.2018.2798285.

[33] Y. Li, J. Li, W. Fu, and Y. Wu. Learning agile bipedal motions on a quadrupedal robot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9735–9742. IEEE, 2024.

[34] R.-Z. Qiu, Y. Song, X. Peng, S. A. Suryadevara, G. Yang, M. Liu, M. Ji, C. Jia, R. Yang, X. Zou, et al. Wildlma: Long horizon loco-manipulation in the wild. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10011–10019. IEEE, 2025.

[35] Y. Ouyang, J. Li, Y. Li, Z. Li, C. Yu, K. Sreenath, and Y. Wu. Long-horizon locomotion and manipulation on a quadrupedal robot with large language models. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11157–11164. IEEE, 2025.

[36] Y. Tang, W. Yu, J. Tan, H. Zen, A. Faust, and T. Harada. Saytap: Language to quadrupedal locomotion. *arXiv preprint arXiv:2306.07580*, 2023.

[37] K. Zhou, Y. Mu, H. Song, Y. Zeng, P. Wu, H. Gao, and C. Liu. Adaptive interactive navigation of quadruped robots using large language models. *arXiv preprint arXiv:2503.22942*, 2025.

[38] P. Jian, X. Wei, Y. Liu, S. A. Moore, M. M. Zavlanos, and B. Chen. Lapp: Large language model feedback for preference-driven reinforcement learning. *arXiv preprint arXiv:2504.15472*, 2025.

[39] J. Cui, T. Liu, N. Liu, Y. Yang, Y. Zhu, and S. Huang. Anyskill: Learning open-vocabulary physical skill for interactive agents. In *Conference on Computer Vision and Pattern Recognition(CVPR)*, 2024.

[40] C. Tessler, Y. Kasten, Y. Guo, S. Mannor, G. Chechik, and X. B. Peng. Calm: Conditional adversarial latent models for directable virtual characters. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701597. doi:10.1145/3588432.3591541. URL https://doi.org/10.1145/3588432.3591541.

[41] A. Tirinzoni, A. Touati, J. Farebrother, M. Guzek, A. Kanervisto, Y. Xu, A. Lazaric, and M. Pirotta. Zero-shot whole-body humanoid control via behavioral foundation models. *arXiv preprint arXiv:2504.11054*, 2025.

[42] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008. doi:10.1163/156855308X344864.

[43] J. Y. Chai, Q. Gao, L. She, S. Yang, S. Saba-Sadiya, and G. Xu. Language to action: towards interactive task learning with physical agents. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 29. AAAI Press, 2018. ISBN 9780999241127.

[44] T. M. Howard, S. Tellex, and N. Roy. A natural language planner interface for mobile manipulators. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6652–6659, 2014. doi:10.1109/ICRA.2014.6907841.

[45] S. Stepputtis, J. Campbell, M. Phielipp, S. Lee, C. Baral, and H. Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.

[46] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[47] O. Mees, L. Hermann, and W. Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters*, 7(4):11205–11212, 2022.

[48] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR, 2022.

[49] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor. Chatgpt for robotics: Design principles and model abilities. *Ieee Access*, 12:55682–55696, 2024.

[50] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, 2023.

[51] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[52] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[53] A. Bucker, L. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti. Latte: Language trajectory transformer. *arXiv preprint arXiv:2208.02918*, 2022.

[54] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In G. Gordon, D. Dunson, and M. Dudk, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/ross11a.html.

[55] C. Guo, S. Zou, X. Zuo, S. Wang, W. Ji, X. Li, and L. Cheng. Generating diverse and natural 3d human motions from text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5152–5161, June 2022.

[56] M. Petrovich, M. J. Black, and G. Varol. Tmr: Text-to-motion retrieval using contrastive 3d human motion synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9488–9497, October 2023.

A MimicAgent Implementation Details

MimicAgent orchestrates many task-specific sub-agents to generate kinematically feasible quadruped reference trajectories. For parity with Eureka, all MimicAgent sub-agents use Claude Sonnet 4.5. We implement MimicAgent using LangGraph, a graph-based framework that represents an agentic system as a directed graph in which nodes correspond to agents or deterministic processing steps, edges define control flow and conditional transitions, state is a shared, and persistent objects are passed between nodes. Unlike linear prompting pipelines, LangGraph enables cyclic execution (e.g., refinement loops), conditional branching based on evaluation outcomes, and the accumulation of long-horizon context without prompt bloat. This structure is essential for MimicAgent, as trajectory synthesis requires iterative correction based on downstream validation.

To generate a diverse set of candidate skills, we employ a dedicated skill proposal agent. Given an initial prompt, the agent generates K skill descriptions per iteration. To prevent duplication and encourage diversity, all previously generated skills are appended to the prompt context in subsequent iterations. The agent is instructed to propose novel behaviors that differ in contact patterns, base motion (translation, rotation, or vertical motion), limb coordination, and stylistic or agility characteristics. This process is repeated until a target number of unique skills is reached. Once generated, each skill description is processed independently by MimicAgent. After generating skill descriptions, MimicAgent processes each description using a skill filter node, motion planner agent, coder agent, code execution agent, task-agnostic unit test node, motion failure diagnosis agent, and a code rewrite agent. A shared state object accumulates all intermediate artifacts, enabling coherent multi-step refinement. We describe each component below.

First, the skill filter node performs lightweight semantic validation of the natural-language skill description. Its purpose is to reject skills that are fundamentally underspecified, contradictory, or kinematically impossible even under idealized assumptions. Filtering at this stage prevents downstream agents from wasting tokens on faulty specifications. The motion planner agent then converts a skill description into a structured phase-based motion plan. Its prompt enforces a strict planning-only role, and explicitly instructs the LLM to avoid code generation, equations, or hyperparameter selection. All motion is described using a single cyclic phase variable $\phi \in [0, 1]$. The output is a single JSON object with a fixed schema, including, phase decomposition into sub-phases, per-phase contact expectations, base velocity and angular rate intent, qualitative leg trajectory descriptions, and declarative constraints aligned with unit tests. Importantly, the planner specifies intent rather than implementation, enabling multiple realizations of the same skill structure.

Next, the coder agent translates the structured motion plan into executable Python code that implements a subclass of `BaseMotionGenerator`. The agent must precisely adhere to the provided base class interfaces and architectural patterns, and is explicitly prohibited from modifying base class APIs, inventing new abstractions, and emitting partial code or placeholders. All base motion is implemented via velocity and angular rate commands, while all leg trajectories are generated as phase-continuous functions in the body frame. Few-shot reference implementations (e.g., a trot gait) are provided to demonstrate correct phase handling, swing-stance decomposition, and trajectory shaping. These references guide code structure without constraining motion semantics. After the coder agent implements executable Python code, the code executor agent validates it by executing the code in the MuJoCo simulator and provides a binary signal based on actual runtime behavior. Unlike the planning or critique agents, the code executor agent does not analyze or modify code. If the simulator terminates successfully without runtime errors, the agent records a success flag in the shared state. If execution fails, the agent captures and propagates the standard error output, exception trace, and return code. These signals are passed verbatim to downstream agents for critique or refinement.

Generated trajectories are evaluated using a suite of task-agnostic unit tests that detect structural issues without encoding task success. These tests identify issues such as ground penetration, joint limit violations, excessive air time, and unsafe base height envelopes. Each test produces a scalar score, and trajectories must exceed a minimum threshold to pass validation. If validation fails, the

motion failure diagnosis agent analyzes the trajectory using the current skill implementation, unit test feedback, and original skill description. Importantly, this agent does not modify code, and is instead instructed to produce a structured diagnostic report identifying the most likely root cause of each failure.

Lastly, the code rewrite agent revises the trajectory-generating code based on unit test feedback and diagnostic analysis. Its prompt enforces a minimal-change philosophy: only components responsible for reported failures may be modified. Structural failures (e.g., joint violations) are addressed first by adjusting amplitudes, phase timing, or trajectory envelopes. Quality failures (e.g., velocity spikes) are addressed through smoothing and phase-continuous reshaping without flattening the motion or altering its defining characteristics. The agent outputs a single, complete Python file, closing the agentic loop. This execution loop repeats until either the generated trajectory passes all unit tests, or a fixed iteration budget is reached. All intermediate artifacts are retained in the shared state, enabling consistent multi-step reasoning.

B DeepMimic Training and Sim-to-Real Transfer

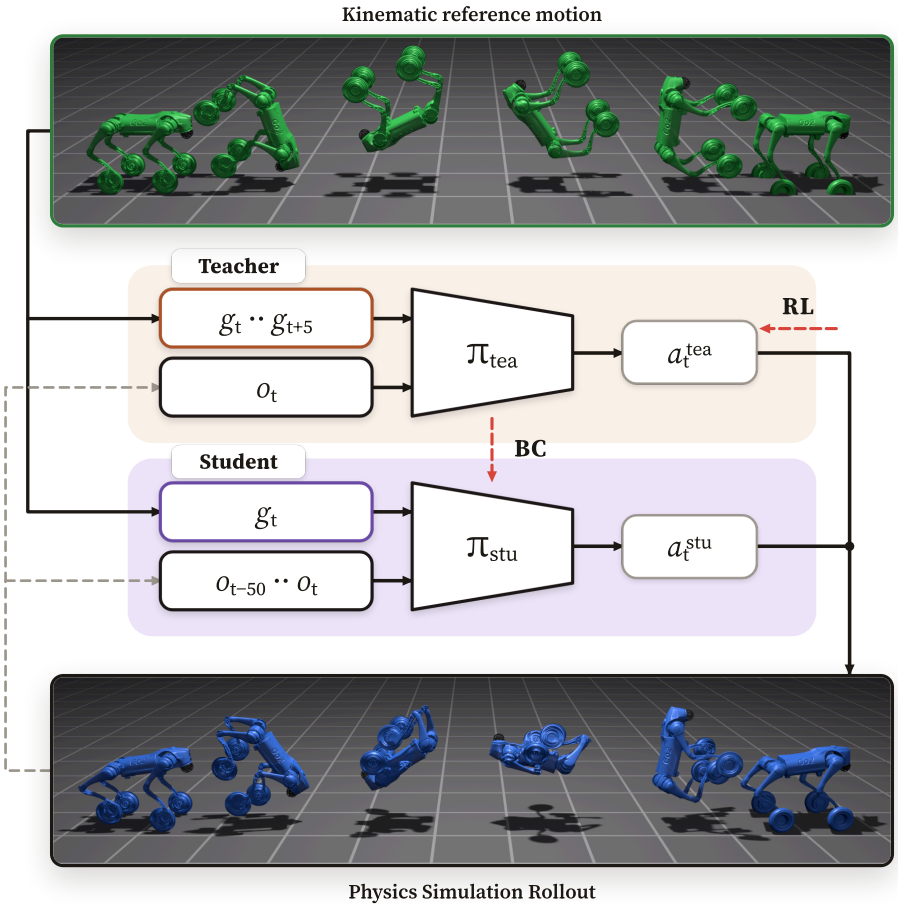


Figure 5: **Sim-to-Real Training.** We use the MimicAgent generated kinematic trajectory for both training in simulation and also during hardware deployment. We successfully deploy the policy using standard teacher-student policy distillation with DAgger.

Problem Formulation. We cast motion imitation as a goal-conditioned reinforcement learning problem DeepMimic [19] and adopt the teacher–student paradigm of HOVER [14] . A privileged *teacher* (oracle) policy is first trained with PPO using full simulator state, and then distilled into

a deployment-ready *student* policy whose observations are restricted to quantities a real Go2W can reconstruct on-board from its IMU and joint encoders. At each control step t (50 Hz), a policy $\pi(a_t \mid s_t^p, s_t^g)$ maps a proprioceptive observation s_t^p and a goal observation s_t^g to an action $a_t \in [-1, 1]^{16}$ for the 16-DoF Go2W (12 leg joints and 4 wheels). The action specifies per-joint motor torques, applied at the 200 Hz physics rate (control decimation 4).

Observations. we hat reference (goal) quantities and split each observation into a proprioceptive block s_t^p and a goal block s_t^g. The teacher is *yaw-aware* (sim-only) and the student is *yaw-invariant* (deployable); the two therefore differ only in their self-orientation and reference-orientation channels. The proprioceptive blocks are

$$s_t^{p\text{-teacher}} = \big[\, \mathbf{q}_t^{\text{root}},\ \boldsymbol{\omega}_t^{\text{root}},\ \boldsymbol{\theta}_t,\ \dot{\boldsymbol{\theta}}_t,\ a_{t-1} \,\big], \tag{1}$$

$$s_t^{p\text{-student}} = \big[\, \mathbf{g}_t,\ \boldsymbol{\omega}_t^{\text{root}},\ \boldsymbol{\theta}_t,\ \dot{\boldsymbol{\theta}}_t,\ a_{t-1} \,\big], \tag{2}$$

where $\mathbf{q}_t^{\text{root}}$ is the root orientation quaternion (yaw-aware, sim-only), $\mathbf{g}_t$ is the body-frame gravity direction (projected gravity, which an IMU measures drift-free and which is yaw-invariant), $\boldsymbol{\omega}_t^{\text{root}}$ is the base angular velocity, $\boldsymbol{\theta}_t, \dot{\boldsymbol{\theta}}_t$ are the joint positions and velocities, and a_{t-1} is the previous action. The goal blocks expose the reference targets:

$$s_t^{g\text{-teacher}} = \big[\, \Delta\mathbf{p}_t^{\text{root}},\ \hat{\mathbf{q}}_t^{\text{root}},\ \hat{\mathbf{v}}_t,\ \hat{\boldsymbol{\omega}}_t,\ \hat{\boldsymbol{\theta}}_t,\ \hat{\dot{\boldsymbol{\theta}}}_t,\ \Delta\mathbf{f}_t,\ \phi_t \,\big], \tag{3}$$

$$s_t^{g\text{-student}} = \big[\, \hat{\mathbf{g}}_t,\ \hat{\mathbf{v}}_t,\ \hat{\boldsymbol{\omega}}_t,\ \hat{\boldsymbol{\theta}}_t,\ \hat{\dot{\boldsymbol{\theta}}}_t,\ \Delta\mathbf{f}_t,\ \phi_t \,\big], \tag{4}$$

where $\Delta\mathbf{p}_t^{\text{root}}$ is the body-frame root-position error to the reference, $\hat{\mathbf{v}}_t, \hat{\boldsymbol{\omega}}_t$ are the reference velocities, $\Delta\mathbf{f}_t$ is the body-frame end-effector tracking error, and $\phi_t \in [0, 1]$ is the cyclic motion phase. The student swaps the teacher's absolute-orientation channels for the single yaw-free reference gravity $\hat{\mathbf{g}}_t$, so its entire observation is reconstructable on hardware and free of IMU yaw drift; the remaining velocity, joint, and phase targets are shared by both policies.

The teacher additionally receives a privileged block s_t^{priv} comprising ground-truth base velocity and height, link positions, contact flags, a short future-reference lookahead, and the per-environment domain-randomization parameters. The oracle is a *symmetric* actor–critic: both networks consume $[\, s_t^{p\text{-teacher}}, s_t^{g\text{-teacher}}, s_t^{\text{priv}} \,]$. The student actor is restricted to the deployable channels plus a 50-step history H_t of its proprioception, which a 1D-CNN encoder compresses into a compact latent z_t for implicit system identification (RMA-style [4]); its critic reuses the teacher's privileged observation, yielding an *asymmetric* actor–critic. All channels are enumerated in Table 6.

Teacher–Student Distillation. The oracle is optimized with PPO (Table 5). Once converged, it supervises the student via DAgger [54]: the student rolls out $\pi_{\text{student}}(a_t \mid s_t^{p\text{-student}}, s_t^{g\text{-student}})$ while the oracle $\pi_{\text{teacher}}(\hat{a}_t \mid s_t^{p\text{-teacher}}, s_t^{g\text{-teacher}})$ is queried on the matched privileged state to produce target actions $\hat{a}_t$, and the student is trained by behavior cloning ($\|a_t - \hat{a}_t\|^2$) on its deployable observation. This transfers the oracle's behavior into a policy whose inputs are all reconstructable on the real robot.

Reward. The reward is a weighted sum of imitation terms, each an exponential tracking kernel $\exp(-k\, e)$ and hardware-aware regularization terms:

$$r_t = \underbrace{\sum_{m \in \mathcal{I}} w_m \exp\big(-k_m\, e_t^m\big)}_{\text{imitation}} + \underbrace{\sum_{n \in \mathcal{R}} w_n\, c_t^n}_{\text{regularization}}, \tag{5}$$

where each imitation error e_t^m is the squared deviation between the simulated and reference quantity for $m \in \{$joint position, joint velocity, root position, root rotation, root linear velocity, root angular velocity, end-effector$\}$. Body rotation uses the geodesic angle $\theta_t = 2 \arccos\big(|\mathbf{q}_t^{\text{root}} \cdot \hat{\mathbf{q}}_t^{\text{root}}|\big)$, and the end-effector term is root-anchored in world axes, so the body cannot satisfy it without actually rotating. The regularization terms c_t^n : gated by a master switch kept on for all hardware policies : penalize action rate, joint velocity/acceleration, and torque, hinge violations of the joint-position/velocity/torque limits (active only above 95% of each cap), and self-collisions. All weights w_m, w_n and kernel scales k_m are given in Table 4.

The teacher is trained with PPO and the student with DAgger (Table 5); both use $32,768$ parallel environments on 8x RTX 3090 GPU, with the domain randomization and observation noise of Table 7 for robust sim-to-real transfer.

Table 4: **Motion-imitation and regularization rewards.** Each imitation term is a tracking kernel $\exp(-k\,e)$ with weight w and scale k; each regularization term is a penalty c with a negative weight w. Errors are squared deviations from the reference unless noted.

Term	**Form**	**Weight** w	**Scale** k	**Notes**
Imitation (tracking kernels)				
Joint position	$\exp(-k\sum_j \lVert\boldsymbol{\theta}_j - \hat{\boldsymbol{\theta}}_j\rVert^2)$	0.50	0.25	tracked joints
Joint velocity	$\exp(-k\sum_j \lVert\dot{\boldsymbol{\theta}}_j - \hat{\dot{\boldsymbol{\theta}}}_j\rVert^2)$	0.10	0.01	tracked joints
Root position	$\exp(-k\lVert\mathbf{p}^{\text{root}} - \hat{\mathbf{p}}^{\text{root}}\rVert^2)$	0.15	5.0	world frame
Root rotation	$\exp(-k\,\theta_t^2)$	0.15	4.0	θ_t geodesic angle
Root linear vel.	$\exp(-k\lVert\dot{\mathbf{p}}^{\text{root}} - \hat{\dot{\mathbf{p}}}^{\text{root}}\rVert^2)$	0.50	4.0	world frame
Root angular vel.	$\exp(-k\lVert\boldsymbol{\omega}^{\text{root}} - \hat{\boldsymbol{\omega}}^{\text{root}}\rVert^2)$	0.15	1.0	world frame
End-effector (feet)	$\exp(-k\sum_i \lVert\mathbf{f}_i - \hat{\mathbf{f}}_i\rVert^2)$	2.50	40.0	world axes
Regularization (gated)				
Action rate	$\lVert a_t - a_{t-1}\rVert^2$	-1.0×10^{-2}	–	
Joint velocity	$\sum\lVert\dot{\boldsymbol{\theta}}\rVert^2$	-1.0×10^{-4}	–	
Joint acceleration	$\sum\lVert\ddot{\boldsymbol{\theta}}\rVert^2$	-5.0×10^{-8}	–	
Torque	$\sum\lVert\boldsymbol{\tau}\rVert^2$	-1.0×10^{-5}	–	
Torque-limit hinge	$\sum\text{relu}(\lvert\boldsymbol{\tau}\rvert - 0.95\,\boldsymbol{\tau}_{\max})$	-1.0	–	
Joint-vel.-limit hinge	$\sum\text{relu}(\lvert\dot{\boldsymbol{\theta}}\rvert - 0.95\,\dot{\boldsymbol{\theta}}_{\max})$	-2.0	–	
Joint-pos.-limit hinge	$\sum\text{relu}(\text{overflow of } 0.95 \text{ range})$	-5.0	–	
Collision	$\sum\mathbf{1}[\lVert\mathbf{f}_c\rVert > 0.1]$	-1.0	–	penalized links

Table 5: **Teacher (PPO) and student (DAgger) training hyperparameters.** Both share the rollout and optimization backbone; the student adds the history encoder and the distillation schedule. "–" denotes not applicable.

Hyperparameter	**Teacher (PPO)**	**Student (DAgger)**
Parallel environments	4096	4096
Rollout length (steps/env)	24	24
Learning epochs	5	10
Mini-batches	4	4
Discount γ	0.99	0.99
GAE λ	0.95	0.95
Clip ϵ	0.2	0.2
Entropy coefficient	0.0	0.0
Value-loss coefficient	0.5	–
Desired KL (adaptive)	0.01	adaptive
Learning rate	3×10^{-4}	3×10^{-4}
Max gradient norm	1.0	1.0
Init. action std (fixed)	0.1	0.1
Actor hidden dims	$[512, 512, 256, 128]$	$[512, 256, 128]$
Critic hidden dims	$[512, 512, 256, 128]$	$[512, 512, 256, 128]$
Activation / LayerNorm	SiLU / yes	SiLU / yes
Iterations	5000	3000
History encoder	–	1D-CNN $\rightarrow z_t \in \mathbb{R}^{32}$

Table 6: **Observation channels for the student and teacher policies.** The student actor uses only hardware-reconstructable (yaw-invariant) channels plus a short proprioceptive history encoded by a 1D-CNN; the teacher (oracle) and the student's critic share the privileged observation.

Block	Channel	Dim	Student actor	Teacher / critic
Proprio s_t^p	Projected gravity $\mathbf{g}_t$	3	✓	✗
	Root orientation $\mathbf{q}_t^{\text{root}}$	4	✗	✓
	Base angular velocity $\boldsymbol{\omega}_t^{\text{root}}$	3	✓	✓
	Joint positions $\boldsymbol{\theta}_t$	16	✓	✓
	Joint velocities $\dot{\boldsymbol{\theta}}_t$	16	✓	✓
	Previous action a_{t-1}	16	✓	✓
Goal s_t^g	Ref. projected gravity $\hat{\mathbf{g}}_t$	3	✓	✗
	Ref. root pos. diff (body) $\Delta\mathbf{p}_t^{\text{root}}$	3	✗	✓
	Ref. root orientation $\hat{\mathbf{q}}_t^{\text{root}}$	4	✗	✓
	Ref. linear velocity $\hat{\mathbf{v}}_t$	3	✓	✓
	Ref. angular velocity $\hat{\boldsymbol{\omega}}_t$	3	✓	✓
	Ref. joint positions $\hat{\boldsymbol{\theta}}_t$	16	✓	✓
	Ref. joint velocities $\hat{\dot{\boldsymbol{\theta}}}_t$	16	✓	✓
	End-effector diff (body) $\Delta\mathbf{f}_t$	12	✓	✓
	Motion phase ϕ_t	1	✓	✓
History	Proprio history H_t ($\rightarrow z_t$)	50×54	✓	✗
Privileged s_t^{priv}	Base linear velocity	3	✗	✓
	Base height	1	✗	✓
	Link positions (body)	12	✗	✓
	Root position (world)	3	✗	✓
	Contact flags	4	✗	✓
	Future-reference lookahead	125	✗	✓
	Domain-randomization parameters	24	✗	✓
Total			**2808**	**285**

Table 7: **Domain randomization and observation noise.** "$\mathcal{U}$" denotes a uniform range resampled per environment; white-noise terms are resampled every control step.

Category / Parameter	Range	Comment
Dynamics randomization		
Ground friction μ	$\mathcal{U}[0.25,\ 1.6]$	applied to all rigid shapes
Restitution	$\mathcal{U}[0.0,\ 0.5]$	
Added base mass	$\mathcal{U}[-2.0,\ 3.0]$ kg	trunk payload
CoM offset	$\mathcal{U}[-0.06,\ 0.06]$ m	per axis
Motor strength	$\mathcal{U}[0.65,\ 1.35]$	per-joint torque scale
Joint dry friction	$\mathcal{U}[0.0,\ 0.12]$ N m	Coulomb, per joint
Action delay	$\mathcal{U}\{0, \ldots, 3\}$ ticks	0–60 ms at 50 Hz
Disturbances		
Push interval	10 s	6-DoF velocity kick
Push linear vel. (xy / z)	±0.8 / ±0.3 m/s	
Push angular vel.	±0.3 rad/s	
Continuous force	$\mathcal{U}[-5,\ 5]$ N	prob. 0.005/step
Continuous torque	$\mathcal{U}[-0.25,\ 0.25]$ N m	prob. 0.005/step
Observation white noise		
Joint position / velocity	0.01 rad / 1.5 rad/s	encoder / finite-diff
Base angular velocity	0.2 rad/s	IMU gyro
Projected gravity	0.05	IMU accelerometer
Base linear velocity	0.1 m/s	critic only

Hardware Deployment. We deploy the distilled student policy zero-shot on a physical Unitree Go2W, with on-board inference closing the control loop at the same rate used in simulation. Because the student observation is restricted to quantities derivable from the on-board IMU and joint encoders (Table 6), the policy runs without any external motion capture or privileged state. The domain randomization and observation noise applied during training (Table 7) account for the residual sim-to-real gap. Sim-to-real rollout videos are provided in the project webpage as supplementary material.

C Baseline Implementation

We present implementation details for our baselines below.

Eureka uses Claude Sonnet 4.5 to design reward functions using evolutionary program synthesis. Eureka takes as input a natural language task description ℓ, the environment definition $\mathcal{M}$ expressed as executable source code exposing state and action variables, a black-box task fitness function F, and a reinforcement learning algorithm $\mathcal{A}$ used to optimize candidate rewards. The environment source code $\mathcal{M}$ is parsed to extract only the components that define observations, actions, and relevant state variables. This code, together with the task description ℓ and generic reward-design instructions, is provided as context to the LLM. No task-specific prompts or reward templates are used.

Given the contextualized prompt, the LLM generates a batch of K executable reward programs

$$\{R_1, R_2, \ldots, R_K\},$$

each implemented as a TorchScript-compatible function returning a scalar reward and a dictionary of interpretable reward components. Each reward candidate R_k is evaluated by training a policy

$$\pi_k = \mathcal{A}(\mathcal{M}, R_k)$$

using RL. The resulting policy is scored using the task fitness function $F(\pi_k)$, yielding a scalar fitness value s_k. The best-performing reward $R^* = \arg\max_k s_k$ is retained as the current best reward. If s^* exceeds the best score achieved so far, the best reward is updated. Training statistics, including success rates, episode lengths, and per-component reward magnitudes, are summarized into a textual reward reflection. This reflection provides diagnostic feedback describing which reward components are ineffective, poorly scaled, or dominant. The best reward R^* and its reflection are appended to the LLM context. The LLM is then prompted to generate a new batch of mutated reward candidates by modifying the structure, scaling, or functional forms of existing components, or by introducing new components. This defines one evolutionary iteration. The processes of sampling, evaluation, reflection, and mutation are repeated for N iterations, optionally with multiple random restarts to mitigate local optima. The final output is the reward program with the highest observed fitness across all iterations. The algorithm returns a fully executable, human-interpretable reward function that can be reused for policy training, curriculum learning, or further refinement with human feedback. Notably, Eureka relies on carefully specified task-specific success metrics and exhibits limited robustness when scaling to diverse, stylistically different skills. Although MimicAgent does not require task-specific unit tests, we hand-engineer success criteria for Eurekas policies to give it a better chance of succeeding. We used Eurekas default configuration, 5 evolutionary iterations with 16 reward samples each, where each policy was trained for 3,000 PPO iterations (6 hours on 8x RTX 3090s) and ran it once per task due to high token cost.

Keyframing Reference Motion Tool. To streamline the process of defining reference motions for quadruped robots, we developed a custom keyframe annotation tool (Figure 6) that visualizes a MuJoCo forward kinematics model of the Go2 robot. The interface allows users to "puppeteer" the agent and manually construct reference trajectories through a hybrid control scheme. It utilizes intuitive click-and-drag mechanics for coarse spatial adjustments alongside a comprehensive UI panel for fine-grained, slider-based manipulation of individual foot positions (X, Y, Z) and root body orientation (roll, pitch, yaw). The tool further accelerates the animation process with advanced features like leg mirroring, a dedicated timeline for keyframe management, and direct export capabilities to

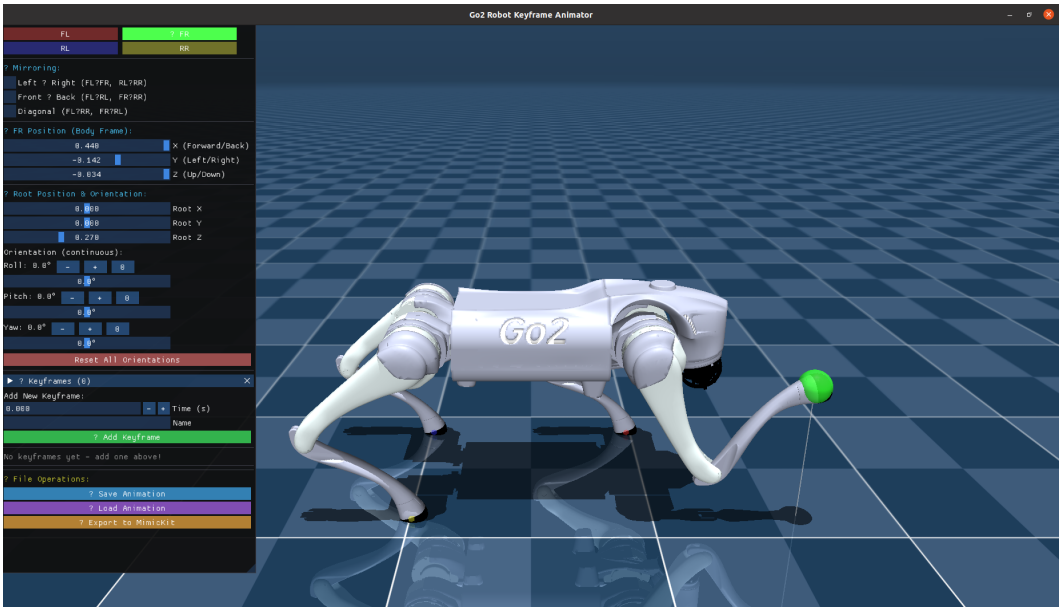


Figure 6: **Keyframing Reference Motion.** We visualize a MuJoCo forward kinematics model for *puppeteering* a quadruped by manually keyframing reference trajectories. We demonstrate that such an interface is easier for both humans and LLMs to tune, compared to standard workflows for reward shaping.

frameworks like MimicKit. Ultimately, this interactive design significantly lowers the barrier to entry, making trajectory tuning faster and more intuitive for both human operators and LLMs compared to traditional reward shaping workflows.

User Study. To evaluate qualitative text-policy alignment, we conducted a user survey of 57 non-expert participants through the Cint platform. The demographic profile reveals a relatively affluent, older, and highly educated cohort; of those who reported their education, all hold at least a college degree, including 20 with graduate degrees. Household incomes skew high, most commonly falling between $150,000 and $174,999, with several exceeding $200,000. Additionally, the group leans slightly male (22 male vs. 16 female) and is predominantly aged 60 and above, followed closely by the 30 to 44 age bracket. Participants awarded an overall average alignment rating of 3.31 out of 5 across all measured actions.

D Quadruped Taxonomy

We present a simple taxonomy of quadruped locomotion skills (Figure 7) that categorizes kinematic behaviors for wheeled quadruped robots according to the primary source of propulsion. The taxonomy consists of two main classes: *Active (Wheel-Driven)* and *Passive (Limb-Driven)* skills. Active skills are defined by wheel joint actuation as the dominant mechanism for locomotion; while leg joints may move freely and can exhibit large, coordinated motions for posture adjustment, stylization, or dynamic effects, they are not intended to contribute propulsive forces. This category includes planar rolling and wheel maneuvers like straight, curved, looping trajectories, pivots, and spins, wheel-assisted aerial behaviors in which wheels govern translational displacement while limbs shape the aerial pose, and hybrid stylized motions that combine rolling, rotation, or limb coordination to produce expressive, non-canonical trajectories. In contrast, passive skills rely exclusively on coordinated leg-joint actuation, with wheels remaining inactive such that propulsion emerges entirely from limb kinematics. Representative behaviors in this class include skating motions generated through lateral or diagonal leg sweeps, multi-limb undulation producing wave-like body or contact patterns, and

asymmetric motions characterized by non-canonical limb sequencing, diagonal pushes, or alternating support patterns that produce complex trajectories. We use this taxonomy to propose skills below.

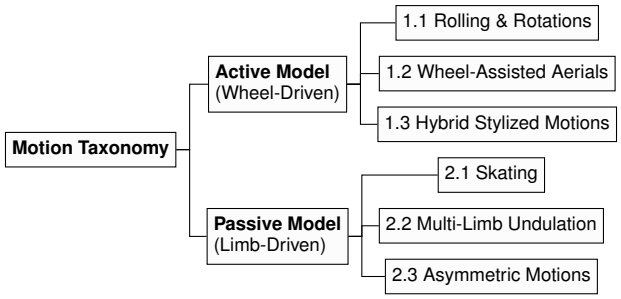


Figure 7: **Quadruped Taxonomy.** Kinematic locomotion skills categorized by primary source of propulsion: wheel-dominant (*Active*) vs. limb-dominant (*Passive*). Each skill is labeled with an ID (e.g., 1.1 for Rolling & Rotations) for easy reference.

E Skill Definitions

We evaluate the following skills in our user study.

- *UniTree Go2 Trot.* A symmetric diagonal gait where front-left and rear-right legs alternate with front-right and rear-left, maintaining continuous ground contact and a stable torso while tracking commanded forward velocity.
- *UniTree Go2 Bounding.* A dynamic gait where the front legs push off together followed by the rear legs, producing a brief aerial phase and pronounced vertical oscillation while maintaining forward propulsion.
- *UniTree Go2-W Front Flip.* An acrobatic maneuver where the robot generates strong forward pitch momentum, enters a fully airborne phase, rotates 360 about its lateral axis, and lands upright with controlled impact.
- *UniTree Go2-W Side Flip.* A lateral acrobatic motion where the robot initiates a rapid roll rotation about its longitudinal axis, becomes fully airborne, completes a full 360 side rotation, and recovers to a stable stance.
- *UniTree Go2-W Aerial Crossover.* While carving forward on its wheels, the robot lifts alternating diagonal leg pairs off the ground, executing rapid mid-air crossover wave motions.
- *UniTree Go2-W Reverberating Yaw Pulse.* An oscillatory motion with decreasing amplitude envelope, mimicking physical damping. Phase controls the amplitude decay and frequency of reverberating angular pulses. Unlike constant-rate spinning, this creates visually distinct accelerating-decelerating rhythm.
- *UniTree Go2-W Crab Diagonal Scuttle.* The robot scuttles diagonally with body axis perpendicular to travel vector. This decouples body orientation from travel direction, creating crab-like diagonal motion. Phase coordinates cross-body leg sweeps where front and rear act in opposition. The orientation remains perpendicular to the velocity vector.

F Comparison with Expert RL Controller

To evaluate whether MimicAgent policies achieve locomotion quality competitive with hand-tuned expert controllers, we compare against Walk-These-Ways (WTW) [2] on two canonical gaits: trot and bound. Because the two methods may produce gaits at different frequencies, we compare via Waveform Morphology Analysis, normalising each trajectory to a single gait cycle for frequency-invariant comparison.

Table 8: **Comparison with Walk-These-Ways (WTW).** Waveform Morphology Analysis enables frequency-invariant comparison of gait quality at a commanded velocity of 1.0 m/s. MimicAgent achieves 6× lower velocity error on trot and 30% lower on bound. For bound, MimicAgent spontaneously discovers an aerial flight phase (stance duty 39.4% vs. 47.7%).

Gait	Metric	MimicAgent	WTW (Expert)
Trot	Mean velocity (m/s)	**0.9999**	1.1959
	Velocity error (MAE)	**0.0351**	0.1959
	Periodic stability	**0.9575**	0.8646
	Velocity std dev	**0.0434**	0.0598
	Stance duty cycle (%)	49.8	47.0
	Cycle duration (frames)	10.0 ± 0.2	16.6 ± 0.5
Bound	Mean velocity (m/s)	**1.0053**	1.1930
	Velocity error (MAE)	**0.1495**	0.2139
	Periodic stability	**0.9587**	0.8625
	Velocity std dev	0.1841	**0.1608**
	Stance duty cycle (%)	39.4	47.7
	Cycle duration (frames)	9.8 ± 0.8	16.7 ± 0.5

Results are presented in Table 8. On trot, MimicAgent achieves near-perfect velocity tracking (0.9999 m/s on a 1.0 m/s command, 0.01% error) with 6× lower MAE than WTW. Periodic stability is also higher (0.958 vs. 0.865), and the stance duty cycle of 49.8% closely matches the canonical 50/50 trot definition. The cycle duration variance (±0.2 frames vs. ±0.5) further confirms a more metronomic gait. On bound, MimicAgent tracks the commanded velocity with 30% lower error while achieving higher periodic stability (0.959 vs. 0.863). Notably, MimicAgent spontaneously discovers a more dynamic gait topology: the 39.4% stance duty cycle indicates a genuine aerial flight phase, compared to WTW's more grounded 47.7%.

G Skill Diversity and Scalability

A key advantage of MimicAgent is the ability to scale to a large and diverse skill library without per-skill reward engineering. Using our quadruped taxonomy as a structured prior (Section **??**), we prompt an LLM to propose diverse locomotion skills, yielding 111 candidates. MimicAgent generates reference trajectories for all candidates: 15 are deemed infeasible, while 78 succeed after up to four iterative attempts. Incorporating human-in-the-loop feedback further improves performance, increasing the total number of successful skills to 96 (Figure 4).

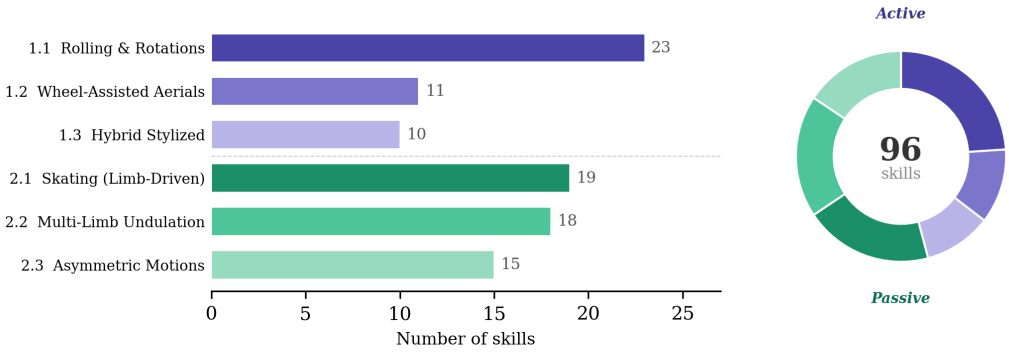


Figure 8: **Skill diversity across the quadruped taxonomy.** MimicAgent generates 96 skills distributed across all six taxonomy categories, covering both Active (wheel-driven) and Passive (limb-driven) motion regimes. The balanced spread demonstrates that the pipeline does not collapse onto a narrow subset of the motion space.

The distribution of generated skills across the taxonomy is visualised in Figure 8. MimicAgent produces a balanced spread: 24.0% Rolling & Rotations (1.1), 11.4% Wheel-Assisted Aerials (1.2), 10.4% Hybrid Stylized (1.3), 19.8% Skating (2.1), 18.8% Multi-Limb Undulation (2.2), and 15.6% Asymmetric Motions (2.3). No single category dominates, confirming that LLM-guided skill proposal combined with taxonomy conditioning avoids mode collapse. To verify physical feasibility, we select one representative skill from each taxonomy category and measure tracking performance against the reference trajectory (Table 9).

Table 9: **Tracking errors for representative skills across taxonomy categories.** Ground locomotion categories (1.1, 1.3, 2.1, 2.2) show low tracking error, confirming compatibility with physical constraints. The higher error for Wheel-Assisted Aerials (1.2) reflects the expected reference-to-physics gap for ballistic maneuvers, where the RL policy must deviate from the kinematic prior to achieve dynamically stable takeoff and landing. Lower is better (↓).

Metric	**Unit**	**1.1**	**1.2**	**1.3**	**2.1**	**2.2**
Root position	m	0.043	0.138	0.013	0.019	0.032
Root rotation	rad	0.082	0.306	0.085	0.017	0.049
Body rotation	rad	0.437	0.652	0.445	0.355	0.444
DOF velocity	rad/s	2.558	15.752	2.384	1.201	1.174
Root velocity	m/s	0.052	0.479	0.048	0.057	0.079
Root angular vel.	rad/s	0.207	2.780	0.459	0.186	0.231

Ground locomotion categories (1.1, 1.3, 2.1, 2.2) achieve excellent tracking fidelity, with root position errors below 5 cm and DOF velocity errors under 3 rad/s. Category 1.2 (Wheel-Assisted Aerials) exhibits notably higher errors : 13.8 cm root position and 15.75 rad/s DOF velocity : reflecting the inherent *reference-to-physics gap*: while the kinematic reference encodes the semantic intent for a flip or jump, the RL policy must deviate from the reference to find dynamically stable solutions for takeoff and landing. To further quantify pipeline reliability, we train policies for 53 of the 96 generated skills and manually inspect them. Of these, 50 (94.3%) correctly execute the semantic meaning of the seed prompt, indicating that MimicAgent reliably produces policies that capture the intended motion.

H Human Ratings vs. LLM-as-a-Judge

We also experimented with using LLMs to automatically evaluate policy quality, but found this approach significantly less reliable than human evaluation. As shown in Table 10, the LLM judge assigns identical or near-identical scores across methods on several skills: for example, all three methods receive a perfect score of 5 on *Bounding* and *Walking*, and broadly fails to distinguish between qualitatively different behaviors that human raters clearly differentiate, particularly on dynamic maneuvers such as flips and aerial crossovers. We hypothesize that this failure stems from a fundamental gap in the pretraining distribution of general-purpose LLMs: such models are unlikely to have encountered substantial data depicting *quadrupedal* robots executing acrobatic skills like side flips, front flips, or aerial crossovers. Without grounded priors for what a high-quality quadruped maneuver looks like, as opposed to a human gymnast or animated character, the LLM cannot meaningfully assess whether the observed motion is physically plausible, dynamically stable, or stylistically faithful to the reference. Human evaluation therefore remains necessary for reliable assessment of such embodiment-specific motor behaviors.

I Training Time and Token Analysis

In Table 11, we show that MimicAgent uses significantly fewer tokens than Eureka. We posit that this is likely because Eureka spends many tokens in its massively parallel reward exploration. In contrast, MimicAgent uses significantly fewer tokens due to the compactness of LangGraph's

Table 10: **LLM-as-a-Judge vs. Human-as-a-Judge.** The LLM judge fails to capture meaningful quality differences, assigning identical scores where human raters clearly distinguish methods. Human evaluation remains necessary for reliable assessment.

	LLM Judge			Human Judge		
Skill	**KF**	**Eureka**	**Ours**	**KF**	**Eureka**	**Ours**
Aerial Crossover	5	2	2	2.3	1.9	**4.2**
Bounding	5	5	5	**3.5**	2.8	3.2
Front Flip	5	5	3	4.2	1.1	**4.6**
Side Flip	5	1	1	3.5	1.4	**4.9**
Walking	5	5	5	2.9	2.6	**4.4**
CDS	5	2	2	2.5	2.1	**4.5**
RYP	5	5	5	1.8	**4.3**	2.6

state management. Somewhat surprisingly, training policies with Eureka is considerably faster than training DeepMimic policies with MimicAgent's reference trajectories. Specifically, DeepMimic requires at least 10,000 PPO iterations for both student and teacher policy training, taking roughly 12 hours (student + teacher) on 8x NVIDIA RTX 3090 to achieve satisfactory tracking performance. In contrast, Eureka, configured with 3,000 PPO iterations, 5 reward iterations, and 16 reward samples per iteration trains in approximately 6 hours.

Table 11: **Token Usage Between Baselines.** We compare the number of tokens used by both MimicAgent and Eureka below. Although MimicAgent iteratively refines generated trajectories and has multiple sub-agents, Eureka uses significantly more tokens across all five baseline tasks due to its parallel reward generation approach. Lower is better.

Robot	**Go2**		**Go2-W**				
Method ↓ / Skill →	**Walking**	**Bounding**	**Side Flip**	**Front Flip**	**AC**	**CDS**	**RYP**
Eureka	803K	917K	676K	682K	730K	658K	625K
MimicAgent	**14K**	**12K**	**13K**	**36K**	**16K**	**14K**	**16K**

J Transferability to Different Simulators

We train DeepMimic policies using only trajectory tracking rewards, without any domain randomization in Isaac Gym. The trained policies are then evaluated in both Isaac Gym and Newton. As shown in Table 12, we find that our DeepMimic policies achieve similar error rates in both simulators, suggesting that our trained policies are robust to simulator characteristics. Further, this suggests that such policies may transfer well to on-robot deployment in the real world.

Table 12: **Cross-Simulator Generalization.** Isaac Gym – trained policies evaluated in Newton. Policies achieve similar performance across simulators, highlighting robustness. Metrics averaged across all skills. Lower is better (↓).

Simulator	**Tracking Error ↓**					
	$E_{\mathbf{RP}}$	$E_{\mathbf{RR}}$	$E_{\mathbf{JR}}$	$E_{\mathbf{JV}}$	$E_{\mathbf{LV}}$	$E_{\mathbf{AV}}$
IsaacGym	0.0640	0.1331	0.3504	5.1658	0.1799	0.9936
Newton	0.0592	0.2182	0.4320	5.5619	0.1792	1.0971

K Impact of Unit Tests on Success Rate

We ablate the impact of task-agnostic unit tests on the overall success rate of trajectories generated by MimicAgent in Table 13. To isolate the contribution of different constraints, we define six progressive unit test configurations: no unit tests (C0), adding a base height limit (C1), adding joint limit checks (C2), adding a feet airtime penalty (C3), adding ground penetration checks (C4, default used in the main paper), and adding dynamically generated, skill-specific unit tests proposed by an LLM (C5). To evaluate these configurations, we randomly sample five skills from the 101 feasible skills generated by our skill proposal agent (Figure 4). For each skill and unit test configuration, we manually validate (e.g. pass / fail) whether the executed skill closely matches the text description. The final success metric for each configuration is the total success rate across the five evaluated skills.

Our results show that progressively adding unit tests improves the overall success rate, with C4 and C5 achieving the highest policy success rate of 80%. The dip observed at C3 can be explained by an interaction between constraints. Specifically, enforcing feet airtime in isolation successfully detects when all feet leave the ground, but compensates by driving the body downward to restore contact. Without a ground penetration constraint, this leads to feet clipping into the terrain. This issue is fully resolved at C4, where the ground penetration constraint is introduced alongside the airtime constraint, demonstrating that these two constraints are complementary and must be applied together to be effective. At C5, LLM-proposed tests introduce skill-specific constraints tailored to the nuances of each motion, resulting in a modest but consistent improvement in the quality of reference trajectories.

Table 13: **Unit Test Ablation.** Impact of unit tests on reference-trajectory and learned-policy quality. Checkmarks indicate active unit tests. Adding all unit tests yields the best performance.

Config	**Active Unit Tests**					**Human Score**	
	Height	**Joint**	**Airtime**	**Ground**	**LLM**	**Reference**	**Policy**
C0	×	×	×	×	×	1/5	1/5
C1	✓	×	×	×	×	3/5	2/5
C2	✓	✓	×	×	×	4/5	3/5
C3	✓	✓	✓	×	×	4/5	2/5
C4	✓	✓	✓	✓	×	**5/5**	**4/5**
C5	✓	✓	✓	✓	✓	**5/5**	**4/5**

L Generalization to Humanoid Robots

While MimicAgent is designed for quadruped skill learning, our approach is grounded in the example-guided RL literature that has driven recent progress in humanoid locomotion. This raises a natural question: *can MimicAgent generalize beyond quadrupeds to higher-dimensional humanoid morphologies?* We answer affirmatively, presenting results in two complementary settings. First, we apply MimicAgent to the standard DeepMimic Humanoid URDF across a diverse battery of skills, demonstrating that our text-to-trajectory pipeline produces stable, coherent policies spanning cyclic locomotion, dynamic acyclic maneuvers, and expressive stylized poses. Second, we provide an *absolute grounding* comparison against ground-truth motion capture on the SMPL humanoid, benchmarking MimicAgent-generated references directly against AMASS/HumanML3D motion capture data.

DeepMimic Humanoid URDF. We evaluate MimicAgent on the DeepMimic Humanoid URDF, training policies in IsaacGym from MimicAgent-generated reference trajectories. Unlike the quadruped setting, the humanoid presents substantially higher degrees of freedom and a less stable base, making both reference generation and policy learning more challenging. We evaluate seven skills that span the full difficulty spectrum, which we group into three regimes:

- *Cyclic locomotion* (*Running* at 2 m/s, *Walk Backward*): periodic gaits with distinct swing/stance phases and, for running, a flight phase. These represent the "well-behaved" end : dynamically benign and periodic.
- *Dynamic / athletic* (*Kicking*, *Ronaldo SIU Jump*): high-momentum, acyclic maneuvers. The *Ronaldo* skill is the stress test, combining a forward run, a vertical jump with a 180° aerial yaw rotation, the signature "SIU" flare, and a stable wide-stance celebratory hold.
- *Expressive / stylized* (*Dab*, *Gangnam Style*, *Usain Bolt*): pose-driven and rhythmic upper-body skills with minimal base translation, testing fine joint coordination rather than gross dynamics.

Across all seven skills, MimicAgent produces stable, coherent policies with low body-position error (≤ 0.10 m), indicating spatially faithful tracking regardless of skill type. The error structure cleanly separates the regimes (Table 14). *Expressive* poses such as *Usain Bolt* and *Dab* exhibit very low velocity residuals (root velocity 0.01–0.03 m/s, root angular velocity < 0.3 rad/s), as expected for near-static targets. *Cyclic* and *dynamic* skills show larger velocity and angular-velocity errors, concentrated in DOF velocity and root angular velocity, with the acyclic *Ronaldo* jump the most demanding (root rotation 0.42 rad, body rotation 0.52 rad).

We attribute these elevated dynamic-error metrics primarily to the *reference-to-physics* gap: because MimicAgent trajectories are kinematically generated and not dynamically feasible, the policy must reconcile the kinematic prior with physical constraints. Importantly, we observe that the policy *improves* upon the reference : for *Ronaldo* it smooths the 180° turn and adjusts limb extension during landing, yielding a more natural and physically feasible behavior than the kinematic prior itself. The elevated error on the most dynamic skills therefore reflects this beneficial deviation rather than a tracking failure.

Table 14: **Quantitative Validation of Humanoid Policy Tracking (DeepMimic URDF).** Root- and joint-level error metrics across seven skills grouped into cyclic locomotion, dynamic/athletic, and expressive/stylized regimes. Pose-driven skills track with very low velocity residuals; dynamic acyclic skills (notably *Ronaldo*) show larger velocity and rotation errors reflecting the reference-to-physics gap, which the policy resolves by smoothing the kinematic prior into a physically consistent trajectory. Lower is better (↓).

Regime	Skill	E_{RP} [m] ↓	E_{RR} [rad] ↓	E_{JR} [rad] ↓	E_{JV} [rad/s] ↓	E_{LV} [m/s] ↓	E_{AV} [rad/s] ↓
Cyclic	Running (2 m/s)	0.080	0.082	0.107	2.317	0.093	0.904
	Walk Backward	0.062	0.111	0.142	9.097	0.217	6.171
Dynamic	Kicking	0.079	0.103	0.145	3.285	0.126	1.338
	Ronaldo (SIU Jump)	0.195	0.423	0.525	7.034	0.506	5.827
Expressive	Dab	0.069	0.039	0.088	3.042	0.031	0.298
	Gangnam Style	0.054	0.121	0.210	0.901	0.156	0.277
	Usain Bolt	0.139	0.055	0.072	0.402	0.013	0.061

These results demonstrate that our pipeline extends to higher-dimensional and more complex systems, producing stable policies across a broad range of skill types: from periodic gaits to acrobatic jumps to stylized celebratory poses. Further validation on more diverse skills and on real-world hardware remains an important direction for future work.

Absolute Grounding: Comparison with Motion Capture. A central concern with synthetic reference generation is whether MimicAgent trajectories can serve as a substitute for real, dynamically faithful motion capture data. To address this directly, we provide an *absolute grounding* comparison: for each of four skills (*Walk Sideways*, *Walk Backwards*, *Running*, and *Kicking*) we train two SMPL humanoid policies under identical settings : one on a ground-truth AMASS motion capture reference drawn from the HumanML3D dataset [55], and one on the corresponding MimicAgent-generated reference : and we compare each policy against its own reference. We report the Text-Motion Retrieval

(TMR) [56] score for both the learned policy and its reference, the fraction of reference quality the policy recovers (recovery %), mean per-joint position error (MPJPE, root-relative), acceleration error, policy jitter, and foot skating. The recovery percentage (policy TMR / reference TMR) is the key quantity: it measures how faithfully a policy reproduces its own reference, independent of the absolute quality of that reference.

Table 15 reports the comparison across four skills. The headline result holds along two axes. First, in *absolute* semantic quality, the MimicAgent-trained policy attains a higher policy TMR than the MoCap-trained policy on three of four skills (e.g. Kicking 0.746 vs. 0.491, Walk Backwards 0.815 vs. 0.735), with Running the lone exception (0.634 vs. 0.681). Second, in *recovery* : the fraction of its own reference a policy reproduces : the MimicAgent policies recover 100–108% (mean 102.6%) of their reference TMR, matching or slightly exceeding the kinematic prior, whereas the MoCap policies recover 66.8–101.6% (mean 86.7%), since the dynamically-real MoCap reference is a higher and harder bar to match exactly. Together these indicate that MimicAgent trajectories provide a dense, learnable reward signal sufficient for high-fidelity RL training on a high-DOF humanoid, and that the synthetic reference is *not* the bottleneck. As expected, the MoCap-trained policies retain an advantage in physical realism, exhibiting consistently lower foot-skating (0.17–0.38 m/s vs. 0.20–2.71 m/s) inherited from their dynamically faithful references, while the MimicAgent policies track their kinematic reference more tightly in absolute pose (MPJPE 10–50 mm vs. 55–67 mm). Both reference sources yield stable, trainable policies on all four skills.

Table 15: **Absolute Grounding (full metrics): MimicAgent-trained (M) vs. MoCap-trained (G) policies on the SMPL humanoid.** Each policy is evaluated against its own training reference. Recovery (%) = Policy TMR / Reference TMR. Better of (M, G) per skill in **bold** (Reference TMR is source context, not bolded). ↑/↓ indicate improvement direction.

[Lucky: TMR Scores NEEDS to be re-evaluated again same text for Mocap and Mimicagent traj for fair comparison]

		Walk Sideways		Walk Backwards		Running		Kicking	
Metric	**Dir.**	**M**	**G**	**M**	**G**	**M**	**G**	**M**	**G**
TMR (Policy)	↑	**0.694**	0.621	**0.815**	0.735	0.634	**0.681**	**0.746**	0.491
TMR (Reference)	↑	0.692	0.611	0.813	0.820	0.589	0.767	0.729	0.735
Recovery (%)	↑	100.3	**101.6**	**100.2**	89.7	**107.6**	88.7	**102.3**	66.8
MPJPE [mm]	↓	**12.57**	62.31	**10.54**	66.09	**49.92**	54.97	**14.29**	66.73
Accel. Err. [m/s^2]	↓	**8.55**	12.87	**8.57**	17.66	30.51	**27.99**	**2.41**	12.14
Jitter [m/s^3]	↓	**418.3**	669.1	**380.3**	793.4	1206.6	**1068.1**	**94.7**	634.6
Foot Skating [m/s]	↓	1.033	**0.246**	1.177	**0.376**	2.709	**0.295**	0.204	**0.169**

Robustness to Reference Trajectory Corruption. A core premise of MimicAgent is that *coarse*, even dynamically-infeasible reference trajectories are sufficient supervision for example-guided RL. We stress-test this premise directly by asking: how much does reference quality actually matter? We construct a controlled corruption ablation in which clean reference trajectories are progressively degraded with synthetic noise, and a separate tracking policy is trained on each corrupted reference. If MimicAgent policies remain trainable under heavy corruption, then the fidelity of the generated reference is not a bottleneck, and minor kinematic imperfections in LLM-generated motions are unlikely to harm downstream learning.

We apply nine compounding corruptions to each reference trajectory at three escalating severity levels (*Light*, *Medium*, *Hard*), summarized in Table 16. These span joint-angle and root-pose noise, foot-position noise, random frame drops with re-interpolation, temporal jitter, joint quantization, and sparse outlier spikes. The *Hard* setting is severe: 0.10 rad ($\approx 5.7°$) of joint noise, 5 cm of root-position noise, 20% frame drops, and 2%-per-frame outlier spikes of up to 0.5 rad ($\approx 29°$). For each of our five SMPL skills we train one policy per corruption level (plus an uncorrupted baseline) and report tracking error against the *clean* reference.

Table 17 reports tracking error against the clean reference for every (skill, corruption) pair. Two findings stand out. First, degradation is *graceful and monotonic*: there is no catastrophic collapse,

Table 16: **Reference corruption protocol.** Nine compounding corruptions applied at three escalating severity levels. Each level defines an independent training condition; a separate policy is trained per (skill, level) pair.

Corruption	Light	Medium	Hard
Joint angle noise (Gaussian std)	0.01 rad ($\sim$0.6°)	0.04 rad ($\sim$2.3°)	0.10 rad ($\sim$5.7°)
Root position noise (std)	3 mm	1.5 cm	5 cm
Root quaternion noise (std)	0.005	0.02	0.05
Foot position noise (std)	5 mm	2.5 cm	8 cm
Frame drops + interpolation	2%	8%	20%
Temporal jitter (dt std)	2%	6%	15%
Joint quantization step	off	0.02 rad	0.08 rad
Outlier spike probability	off	0.5%/frame	2%/frame
Outlier spike magnitude	—	0.2 rad ($\sim$11°)	0.5 rad ($\sim$29°)

no skill diverges, and every policy trains to completion even under the most aggressive corruption. Second, the degradation is highly uneven across metric types: *pose-level* errors (root and body position/rotation) are far more robust than *motion-level* errors (DOF, linear, and angular velocity). From clean to Hard corruption, pose errors grow only $\sim$5–6$\times$ while velocity errors grow $\sim$15–17$\times$. This asymmetry is expected: the dominant corruptions : frame drops, temporal jitter, and outlier spikes : inject high-frequency perturbations that primarily distort finite-difference velocities, whereas the integrated pose remains comparatively stable.

In absolute terms, even at Hard corruption the mean root-position error stays at $\sim$16 cm and body-position error at $\sim$12 cm, indicating that the policy still reproduces a spatially coherent motion. The clean (*None*) condition is informative on its own: all four skills are tracked tightly, with sub-centimeter to few-centimeter body-position error. For several skills (e.g., *Running*) certain pose metrics are even non-monotonic, with Light corruption occasionally matching or slightly improving on the clean baseline : consistent with mild noise acting as a regularizer rather than a disruption.

Learned Policy Improves Over Reference. A striking consequence of the corruption ablation is that the trained policy is often *more physically plausible than the reference it was trained on*. As corruption increases, the reference trajectory itself becomes visibly degraded : jittery, temporally inconsistent, and riddled with outlier spikes. This is directly visible in the metrics: the reference's jitter grows by over 35$\times$ from clean to Hard corruption (from 0.23 to 8.2×10^3 m/s^3) and its foot-skating more than doubles, while its T2M score falls from 0.71 to 0.53. The RL policy, however, is constrained by the physics simulator and cannot reproduce these dynamically-infeasible artifacts. Instead, it absorbs the high-level motion intent while discarding the high-frequency corruption.

The most telling signal is that the policy's jitter is *essentially corruption-invariant*: hovering around 1.4–1.7×10^3 m/s^3 regardless of how badly the reference is degraded : so that from Light corruption onward the policy is 2–6$\times$ *smoother* than its own reference. Likewise, the policy's foot-skating stays below its reference at every level, and its T2M score exceeds the reference's at every level, with the gap *widening* as corruption grows (from $+0.01$ clean to $+0.08$ at Medium). In other words, the imitation-learning stage acts as a dynamics-aware denoiser: even when handed a badly corrupted target, the policy recovers a clean, physically consistent execution of the intended skill. Table 18 quantifies this gap between reference and policy as a function of corruption severity.

Table 17: **Per-skill corruption ablation on the SMPL humanoid.** Tracking error against the clean reference for each skill across the four corruption conditions. Degradation is graceful across all skills; pose-level errors stay small in absolute terms even at Hard corruption. Lower is better (↓).

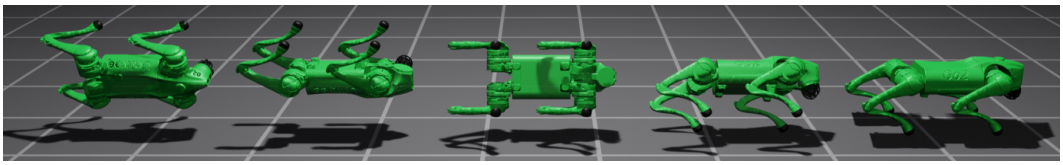
Skill	Metric ↓ / Corruption →	None	Light	Medium	Hard
Walk Side	E_{RP} [m]	**0.027**	0.095	0.177	0.161
	E_{RR} [rad]	**0.025**	0.080	0.182	0.259
	E_{JR} [rad]	**0.067**	0.208	0.317	0.437
	E_{JV} [rad/s]	**0.307**	2.286	2.951	4.325
	E_{LV} [m/s]	**0.061**	0.286	0.960	1.482
	E_{AV} [rad/s]	**0.113**	0.417	2.223	2.807
Walk Back	E_{RP} [m]	**0.032**	0.099	0.132	0.149
	E_{RR} [rad]	**0.021**	0.105	0.214	0.252
	E_{JR} [rad]	**0.047**	0.245	0.350	0.441
	E_{JV} [rad/s]	**0.246**	1.910	2.910	4.327
	E_{LV} [m/s]	**0.052**	0.365	0.911	1.474
	E_{AV} [rad/s]	**0.109**	0.547	1.828	2.624
Running	E_{RP} [m]	**0.060**	0.161	0.132	0.129
	E_{RR} [rad]	**0.094**	0.142	0.212	0.356
	E_{JR} [rad]	**0.190**	0.323	0.377	0.633
	E_{JV} [rad/s]	**0.746**	2.264	3.377	4.170
	E_{LV} [m/s]	**0.165**	0.553	1.017	1.545
	E_{AV} [rad/s]	**0.395**	1.585	2.820	3.706
Kick	E_{RP} [m]	**0.042**	0.088	0.137	0.223
	E_{RR} [rad]	**0.032**	0.087	0.152	0.255
	E_{JR} [rad]	**0.066**	0.233	0.296	0.501
	E_{JV} [rad/s]	**0.144**	2.166	3.125	4.692
	E_{LV} [m/s]	**0.062**	0.291	0.900	1.473
	E_{AV} [rad/s]	**0.103**	0.444	1.695	2.820

Table 18: **Reference vs. Policy under corruption (per skill).** For each skill and corruption level we report the intrinsic quality of the (corrupted) reference trajectory and of the policy trained on it. The policy's T2M score exceeds its reference at almost every level; its jitter is nearly corruption-invariant (so from Light onward it is far smoother than the reference, whose jitter explodes); and its foot-skating is lower than the reference at every single level. The imitation stage acts as a dynamics-aware denoiser. ↑ / ↓ indicate improvement direction; better of (Ref, Policy) per cell in **bold**.

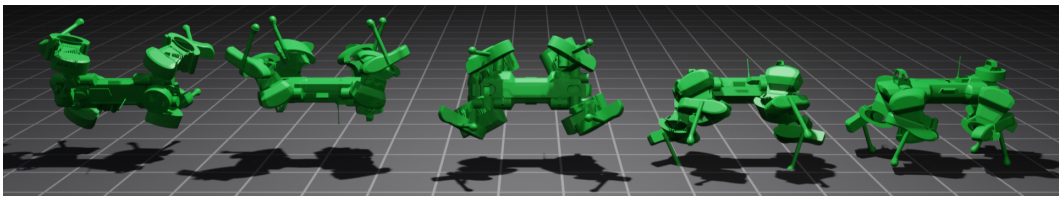
		None		Light		Medium		Hard	
Skill	**Metric**	**Ref.**	**Pol.**	**Ref.**	**Pol.**	**Ref.**	**Pol.**	**Ref.**	**Pol.**
Walk Side	T2M ↑	0.690	**0.694**	0.631	**0.696**	0.574	**0.693**	0.530	**0.577**
	Jitter [m/s^3] ↓	**48**	418	2207	**1633**	3746	**1660**	7693	**1551**
	Foot Skate [m/s] ↓	1.14	**1.03**	1.80	**1.08**	2.54	**0.96**	3.95	**0.97**
Walk Back	T2M ↑	0.797	**0.815**	0.650	**0.728**	0.586	**0.722**	**0.528**	0.506
	Jitter [m/s^3] ↓	**57**	380	4036	**1646**	3806	**1497**	13429	**1597**
	Foot Skate [m/s] ↓	2.49	**1.18**	2.48	**1.26**	2.74	**1.23**	5.91	**3.18**
Running	T2M ↑	0.591	**0.634**	0.582	**0.594**	0.580	**0.585**	0.569	**0.603**
	Jitter [m/s^3] ↓	**780**	1207	2386	**1818**	4547	**2244**	6669	**1551**
	Foot Skate [m/s] ↓	5.57	**2.71**	7.42	**1.93**	6.26	**1.33**	6.72	**2.22**
Kick	T2M ↑	**0.763**	0.746	0.634	**0.666**	0.527	**0.570**	0.482	**0.578**
	Jitter [m/s^3] ↓	**26**	95	4072	**1763**	8629	**1491**	4964	**1018**
	Foot Skate [m/s] ↓	0.25	**0.20**	2.07	**0.76**	3.96	**0.88**	3.17	**1.95**

M Additional Quadruped Morphologies

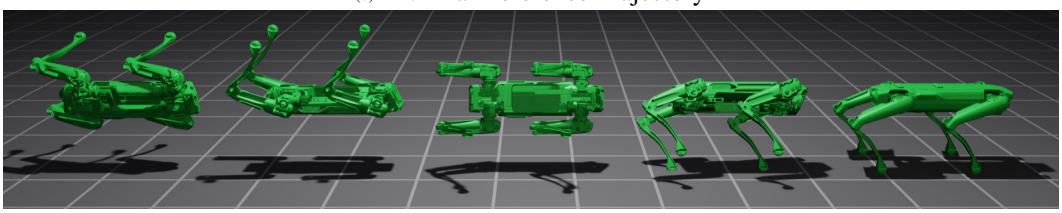
To demonstrate the generalization capabilities of our approach, we deploy the same LLM-generated skill code originally designed for the Go2 across different quadruped morphologies, *without any prompt modifications*. Since MimicAgent generates motion plans using a task-agnostic, phase-based formulation combined with body-frame kinematic commands, the resulting Python implementations remain independent of specific robot dimensions and mass properties. As shown in Figure 9, a single generated skill can therefore be applied directly to produce kinematically valid reference trajectories for a variety of distinct quadruped platforms.



(a) Go2 Reference Trajectory



(b) ANYmal Reference Trajectory



(c) Spot Reference Trajectory

Figure 9: **Morphology Generalization.** The same LLM-generated skill code is applied directly to (a) Go2, (b) ANYmal C, and (c) Spot *without any prompt modifications*. Each row shows keyframes from the resulting reference trajectories.

N Additional Supplementary Material

In addition to this document, we release the following supplementary material. (i) **Code:** the full MimicAgent pipeline, including all sub-agent prompts, the task-agnostic unit-test suite (ii) **Skill trajectory data:** the library of generated reference trajectories for all quadruped skills, in a format directly consumable by the training pipeline. (iii) **Project webpage:** a self-contained webpage, included in the supplementary zip — open `project-website/index.html` in a browser — with visualizations of every quadruped , including reference trajectories, policy rollouts in simulation, and hardware deployment results, also we include humanoid skills. We hope these resources facilitate reproduction of our results and future research on language-driven quadruped skill learning.

O Baseline Skills Prompts

We present the prompts used to generate the seven skills used in our user study below.

Skill Prompt (Bounding Gait).
Control the Go2 robot to execute a dynamic bounding gait while tracking commanded forward velocities in the x-direction. The robot alternates between synchronized front-leg contacts and synchronized rear-leg contacts, with ballistic flight phases between each contact where all feet leave the ground. The robot maintains a forward-facing orientation with minimal roll and yaw, allows natural pitch oscillation during flight and landing, enforces left–right force symmetry within each leg pair to prevent lateral drift, and avoids any four-foot stance phases. Bounding should generate sufficient vertical impulse for observable airborne time, maintain consistent body height, track the commanded forward velocity smoothly, and transition to a stable standing posture when commanded velocities are near zero. The gait remains rhythmic, energy-efficient, and robust to perturbations across varying speeds.

Skill Prompt (Front Flip).
To make the quadruped robot to execute a front flip (forward somersault) conditioned on the commanded forward velocity. When the commanded forward velocity vx is strictly positive, the robot should crouch to build energy, explosively launch upward while initiating forward pitch rotation, complete a full 360-degree rotation around its lateral (pitch) axis while airborne, and land stably on all four feet in an upright orientation with minimal post-landing motion. The flip should be performed approximately in place with minimal horizontal drift, the rotation should be clean without excessive roll or yaw deviation, and the robot must maintain sufficient height throughout the airborne phase to complete the rotation before touchdown. When the commanded forward velocity vx is zero or negative, the robot should remain idle in a stable standing posture with all feet on the ground, minimal base motion, low angular velocities, and low actuation effort. The robot should ignore lateral velocity and yaw commands, prioritizing explosive launch power, fast rotational velocity during flight, proper timing to achieve upright orientation before landing, and controlled impact absorption for stable recovery after touchdown.

Skill Prompt (Troting Gait).
To make the quadruped robot execute a stable and rhythmic trot gait while accurately tracking randomly sampled forward linear velocity commands along the x-axis. The robot should move using synchronized diagonal leg pairs that alternate between push-off and landing phases, producing brief and consistent aerial phases between contacts. Maintain a forward-facing torso with minimal roll, yaw and pitch of a trot, and generate steady,efficient forward propulsion in the commanded direction. The gait should remain robust to rapid velocity changes, minimize foot scuffing and lateral slip, recover gracefully from disturbances, and produce a fast, natural, and agile trotting motion suitable for dynamic quadrupedal locomotion.

Skill Prompt (Aerial Crossover Recovery).
When commanded with positive forward velocity (vx), the wheeled-legged robot should carve forward smoothly on its wheels while briefly lifting alternating diagonal leg pairs into the air. These rapid crossover wave motions of the legs occur without breaking wheel contact and are used to actively reshape the robots inertia and base orientation. The legs must re-enter contact in phase, allowing the robot to transition seamlessly back into a stable skating wave with increased forward speed and agile directional response. This skill treats leg motion as an inertial control surface rather than a support requirement, rewarding fast, rhythmic aerial leg alternation that enhances forward locomotion stability and agility. When commanded with zero or negative forward velocity (vx) and random vy or yaw rate, the robot should remain stationary in a stable, seated posture

Skill Prompt (Side Flip).
Control the Go2w robot to perform a dynamic side flip conditioned on the commanded forward velocity. When the commanded forward velocity vx is strictly positive, the robot should execute a single lateral side flip in place by rotating primarily about its body x-axis (roll), lifting all feet and wheels off the ground briefly, completing approximately one full rotation, and landing stably without falling. The robot should not translate significantly during the flip.
When the commanded forward velocity vx is zero or negative, the robot should remain idle in a stable, seated posture with minimal base motion, low joint velocities, and low actuation effort. The robot should ignore vy and yaw commands. Stability after landing, controlled angular motion, limited base translation, and low unnecessary torque are desired.

Skill Prompt (Crab Diagonal Scuttle).
Control the Go2w robot to scuttle diagonally while holding its body oriented perpendicular to the direction of travel, producing crab-like locomotion in which heading and travel direction differ. The robot should move along a diagonal forward-right vector by coordinating cross-body leg sweeps: the front legs sweep toward the rear while the rear legs sweep toward the front, acting in opposition to drive the body diagonally. The motion proceeds in repeated scuttle cycles—an initial sweep, a rapid leg reset, and a second sweep of larger amplitude—followed by a brief glide and stabilization. Throughout the motion the base must maintain its sideways orientation rather than turning to face the travel direction. Smooth diagonal displacement, consistent perpendicular body orientation, well-coordinated opposing leg sweeps, and stable balance during and after the scuttle are desired.

Skill Prompt (Reverberating Yaw Pulse).
Control the Go2w robot to rotate in place using a series of damped, oscillatory yaw pulses that gradually settle to a net heading change of approximately 45°. The robot should produce alternating-direction yaw impulses with a decreasing amplitude envelope: a sharp $\sim 60^\circ$ counterclockwise pulse, then a $\sim 40^\circ$ clockwise reversal, then $\sim 25^\circ$ counterclockwise, then a $\sim 15^\circ$ clockwise correction, and finally a small $\sim 5^\circ$ counterclockwise damping adjustment, ending rotated about 45° from the start. The legs act as tunable moment-generating elements, extending to create angular impulse during each pulse and retracting partially between pulses, with extension modulated by phase. The base should remain in place with minimal translation while the rotation reverberates. A distinct accelerating–decelerating rhythm, controlled decaying oscillation amplitude, accurate net heading, and a stable settled posture at the end are desired.

P MimicAgent Sub-Agent Prompts

Skill Generator Agent Prompt

SYSTEM: Skill Generator Agent

Task: Propose atomic motion skills for a quadruped / wheeled-quadruped robot (kinematic, MuJoCo feasible).

Constraints:

– No torques, forces, or numeric speeds.
– Describe only base and limb coordination.
– Skills must be kinematically feasible in MuJoCo.
– ...

Phase: Continuous $[0, 1]$, with multiple sub-phases and internal variation.

Wheel Abstraction: Active vs Passive conceptual interpretations.

Skill Mix:

– Locomotion: 50–60%
– Dynamic in-place: 20–30%
– Stylized / Hybrid: 10–20%
– ...

Output: JSON list of skills, each containing:

– `skill_id`
– `skill_name`
– `skill_intent`
– `skill_description`
– `think` (conceptual reasoning)
– ...

FEW-SHOT EXAMPLES: ...
Human Prompt: Generate exactly {skills_per_generation} skills. Follow the schema strictly. Return a JSON list only.

Motion Planner Agent Prompt

SYSTEM: Motion Planner Agent

Task: Take a high-level locomotion or agility skill and produce a structured, phase-based motion plan executable by a BaseMotionGenerator-style controller.

Constraints:

- Do NOT generate code or equations.
- Do generate execution logic, contact reasoning, coordination, and unit-test-compatible constraints.
- Base controller exposes ONLY:
  - `set_velocity_command`
  - `set_angular_velocity_command`
  - ...
- ...

Phase: Single variable $[0, 1]$, smooth, monotonic, cyclic. Motion decisions are functions of phase.

Output Format: Strict JSON with keys:

- `skill_id`
- `motion_plan`:
  - `summary`: ... .
- ...

Critical Rules: Do NOT invent APIs, rename keys, use absolute poses, or world-frame paths. Plan contact base motion leg motion. All constraints must map to unit tests.

Mental Model:

1. Visualize robot over one full cycle.
2. ...

FEW-SHOT EXAMPLES: ...
Human Prompt: Generate a JSON motion plan for a given skill. Follow schema strictly. Return JSON only.

Coder Agent Prompt

SYSTEM: Coder Agent

Role: Convert a structured, phase-based kinematic motion specification into executable Python code for MuJoCo animation and data generation.

Inputs Provided:

– Structured motion plan from Motion Planner Agent

– Base class definitions and utilities

– ...

Robot Model Assumptions:

– Legged or wheeled quadruped

– Purely kinematic motion

– ...

Core Responsibility:

– Interpret motion intent and phase structure

– Choose reasonable numeric parameters

– ...

Code Generation Constraints:

– Output exactly one Python file

– No dynamics, forces, or controllers

– No explanations or markdown

– ...

External Context:
Base classes, math utilities, and reference implementations are provided externally.
...

Human Prompt:

– Motion specification provided in structured JSON

– Skill class name: {`SKILL_NAME`}`_MotionGenerator`

– Leg names: FL, FR, RL, RR

– ...

BEGIN IMPLEMENTATION

Motion Failure Diagnosis Agent Prompt

SYSTEM: Motion Failure Diagnosis Agent

Role: Analyze failures in purely kinematic, phase-based quadruped / wheeled-quadruped motion skills.

You DO:

– Reason about failures and root causes

– Propose corrective strategies

You DO NOT:

– Write or modify code

– Suggest dynamics or control changes

Inputs:

– Current skill implementation

– Evaluation feedback

– Skill description

– ...

Execution Context:

– Purely kinematic

– Phase-driven motion

– MuJoCo for animation only

– ...

Failure Types:

– Structural failures (e.g., penetration, limits)

– Quality failures (e.g., spikes)

– ...

Output Format:

– Summary of failures

– Root cause analysis (per failure)

– Corrective strategies

– ...

Human Input:

{CURRENT_SKILL_CODE} ....

Code Rewrite Agent Prompt

SYSTEM: Code Rewrite Agent

Role: Rewrite an existing kinematic motion generator so that it passes automated evaluation while preserving the original skill intent.

You Receive:

– Current Python skill implementation
– Unit test feedback
– Skill description
– Motion failure diagnosis
– ...

Execution Context:

– Purely kinematic quadruped / wheeled-quadruped
– Phase-based, continuous motion
– MuJoCo used only for animation
– ...

Failure Modes Handled:

– Structural violations (hard constraints)
– Smoothness and quality violations
– ...

Semantic Constraints:

– Preserve defining characteristics of the skill
– Maintain phase ordering and intent
– ...

Hard Constraints:

– Output a single Python file only
– No explanations or markdown
– No new APIs or class changes
– ...

Human Input:

{CURRENT_SKILL_CODE} ...