LocoMimic: Learning Locomotion

Vyvaswath Kalva
Department of Mechanical Engineering
Carnegie Mellon University
vkalva@andrew.cmu.edu

Dinesh Varun shankar Kandiyappan
Department of Mechanical Engineering
Carnegie Mellon University
dkandiya@andrew.cmu.edu

Abstract: Humanoid controllers trained with hand-written reward functions often exhibit behaviors that do not match human gait patterns, resulting in severe motion artifacts and unrealistic postures. We present a deep reinforcement learning framework for learning humanoid locomotion from retargeted motion clips on a Unitree G1 robot. The reward function measures reference tracking fidelity at every step, combining body position, orientation, and velocity terms drawn from the BeyondMimic framework. We compare three algorithms under this setting: [1] Proximal Policy Optimization (PPO), an on-policy method widely used in robotics; [2] Soft Actor-Critic (SAC), an off-policy method; and [3] TD-MPC2, a model-based method that performs trajectory optimization in the latent space of a learned world model. Our evaluations reveal that PPO is the most robust baseline, successfully producing smoother motions. Conversely, TD-MPC2 struggles to maintain balance in this high-dimensional continuous control task, experiencing early terminations and failing to track the reference motion. To address training instabilities in SAC, we propose MeanSAC, which replaces the clipped double-Q minimum with an averaged target and eliminates mid-training reward collapses. We further extend the framework to a 354-muscle musculoskeletal humanoid, demonstrating that the motion-imitation pipeline generalizes beyond rigid-body robots. Videos are available on our project website [4].

Keywords: Deep Reinforcement Learning, PPO, SAC, TD-MPC2

1 Introduction

Reinforcement Learning (RL) enables humanoid locomotion controllers to learn agile and robust behavior directly from simulations. Recent advances in RL algorithms like PPO [1] and SAC [2] have enabled stable learning for legged locomotion. With appropriate rewards and tuning, RL-trained controllers can perform diverse locomotion behaviors on various ground terrains. However, these methods often rely on hand-tuned rewards. Because humanoid locomotion is a high-dimensional control problem that requires human-like agility with limited DOFs, simple reward tasks, such as velocity tracking to make the humanoid walk at a target velocity while constraining the behavior to a specific gait frequency, often lead to severe motion artifacts and unrealistic postures. To address this, it is important to utilize a human motion prior, which ensures that the locomotion controller learns human-like motion from motion datasets without producing unnatural behaviors. While DeepMimic-style reward formulations have produced agile and natural skills by tracking reference data, we specifically utilize BeyondMimic, a variant of DeepMimic, to learn humanoid locomotion controllers from retargeted motion data; this framework has demonstrated a simple yet robust approach to learning sim-to-real policies.

Environment

Deep reinforcement learning requires millions of environment interactions to learn effective control policies, making data collection a major computational bottleneck. While simulation engines like NVIDIA Isaac Gym offer massive hardware parallelization that can reduce training times to a mat-

ter of minutes [5], they often incur significant computational resources. Consequently, we utilize `mjlab` [6], a lightweight framework for GPU-accelerated robot learning. `mjlab` adopts Isaac Lab's manager-based design allowing users to compose self-contained building blocks for observations, rewards, events, and commands and pairs it with MuJoCo Warp to enable massive environment parallelization, overcoming the limitations of standard CPU-based MuJoCo. Specifically, we base our environments on Unitree's `unitree_rl_mjlab` [7], a manufacturer-specific pipeline designed to facilitate the direct sim-to-real deployment of trained policies onto physical hardware. **Robot Model**: The Unitree G1 is a humanoid robot with 29 actuated joints covering the legs, waist, and arms. The MuJoCo model has 36 generalized position coordinates (7 for the floating base plus 29 joint angles) and 35 generalized velocity coordinates. The robot stands approximately 0.8 meters tall at the pelvis in its default configuration. Actuators are controlled via position targets passed to an internal PD controller. The observation and action spaces of the robot are detailed in Appendix A

Rewards

The reward function measures how closely the robot tracks the reference motion at each timestep. Our formulation is based on `mjlab`'s reimplementation of BeyondMimic [8]. The tracking components are split into global root tracking (position and orientation) and relative body tracking (position, orientation, linear velocity, and angular velocity). Each tracking term measures the mean squared error across the target bodies $\mathcal{B}$, shaped by a Gaussian kernel. To ensure the learned policies are smooth, we also include regularization penalties for high-frequency action rates, joint limit violations, and self-collisions. The complete list of reward terms and their respective weights are summarized in Appendix B. Episodes terminate early if the root height deviates from reference by more than the configured threshold, or if the quaternion dot product between the robot and reference root orientations falls below the configured threshold, indicating the robot has fallen beyond recovery.

2 Method

We define our locomotion policy as a whole-body tracking task, where the controller is trained to replicate a reference motion clip in a physics-based simulation. For all experiments, we utilize the retargeted motion data from the LAFAN1 dataset [9]. Specifically, we focus on the `walk1_subject1` motion clip in our experiments.

We evaluate and compare three distinct reinforcement learning algorithms: Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Temporal Difference Learning for Model Predictive Control - 2 (TD-MPC2). We keep the reference motion, reward structure, and simulation environment constant across these trials.

2.1 Proximal Policy Optimization

We use Proximal Policy Optimization (PPO) [1] as our on-policy baseline for motion imitation. PPO optimizes the policy using finite on-policy rollout batches while constraining each update to remain close to the previous policy. Training is executed using `mjlab`'s GPU parallelization. At each iteration, we collect short on-policy rollouts (24 steps per environment), estimate advantages using generalized advantage estimation (GAE) [10], and update the networks for 5 epochs over 4 shuffled minibatches. The PPO surrogate objective is defined as:

$$L^{\text{clip}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}t\right)\right], \tag{1}$$

where $r_t(\theta) = \frac{\pi\theta(a_t|s_t)}{\pi\theta_{\text{old}}(a_t|s_t)}$ is the probability ratio, $\hat{A}t$ is the estimated advantage, and $\epsilon = 0.2$ is the clipping parameter. To ensure stable convergence, we utilize a clipped value loss, gradient norm clipping, and an adaptive learning rate schedule that scales the step size to maintain a target KL-divergence of 0.01. The final objective is:

$$L(\theta) = -L^{\text{clip}}(\theta) + c_v L_V(\theta) - c_e \mathcal{H}(\pi\theta), \tag{2}$$

where $L_V(\theta)$ is the critic squared-error loss and $\mathcal{H}(\pi_\theta)$ is the policy entropy bonus. Full PPO implementation details are provided in Appendix C.

2.2 Soft Actor-Critic

Soft Actor-Critic (SAC) [2] is an off-policy actor-critic algorithm that maximizes a trade-off between expected return and policy entropy. The policy is a tanh-squashed Gaussian with state-dependent mean and variance. SAC maintains two Q-functions and takes their minimum to reduce overestimation bias. The temperature α is automatically tuned to maintain a target entropy.. Being off-policy, SAC stores all past transitions in a replay buffer, allowing experience to be reused across many gradient steps. This makes SAC substantially more sample efficient than on-policy methods such as PPO, which throw away the data after a policy update step. Full implementation details and hyperparameters are provided in Appendix D.

2.3 Temporal Difference Learning for Model Predictive Control

To evaluate a model-based method against our model-free baselines, we implement a single-task adaptation of TD-MPC2 [3]. Rather than explicitly modeling the forward dynamics of the high-dimensional humanoid state space, TD-MPC2 learns an implicit, control-centric world model and performs local trajectory optimization within a learned latent space. Following the original work, our world model uses an *encoder, latent dynamics model, reward predictor, Q ensemble, and policy prior.* Our value function is represented by an ensemble of five Q-heads, and we add a separate termination head to model early episode cutoffs caused by falls. The encoder, dynamics, reward model, and Q ensemble are jointly trained with

$$L(\theta) = c_1 L_{\text{consist}} + c_2 L_{\text{reward}} + c_3 L_{\text{value}} + c_4 L_{\text{term}}, \tag{3}$$

where L_{consist} is a stop-gradient consistency loss between predicted latent states and target latents produced by the current encoder, L_{reward} and L_{value} are soft cross-entropy losses over symlog-transformed targets, and L_{term} is a binary cross-entropy loss for the termination. The policy prior is trained with a maximum-entropy objective over imagined rollout. During inference, TD-MPC2 performs MPPI planning in latent space. A fraction of the sampled action sequences comes directly from the policy prior to warm-start the planner, while the remaining sequences are sampled from a Gaussian whose mean is warm-started from the previous decision step.

3 Results and Analysis

We present the results of all methods evaluated in this report using the same underlying G1 tracking environment and reward function. The reward terms were inherited from the BeyondMimic[8] and kept fixed across the random policy, PPO, SAC, TD-MPC2. Since these rewards were optimized for an on-policy approach, other algorithms may not achieve the same level of performance, a phenomenon previously observed in the literature [11].

3.1 PPO

The PPO agent serves as our primary baseline, benefiting heavily from the parallelization capabilities of `mjlab`. The policy was trained for 30,000 iterations, which corresponds to 2.9 billion environment steps across 4,096 parallel environments.

As shown in the graphs, during the first few iterations, the mean episode length was merely 50 steps with heavily penalized returns. But as the training progressed, the policy completely solved the survival problem, achieving a maximum episode length of 500 steps at 3,000 iterations. Following this quick stabilization phase, the policy focused on high-fidelity motion tracking for the remaining 27,000 iterations. The mean returns started climbing suddenly, plateauing around 41 while the episode length remained pinned to 500 steps.

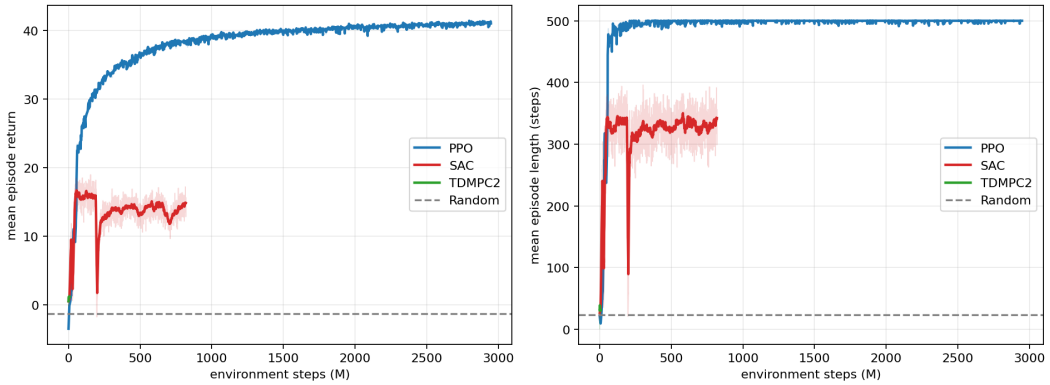

Figure 1: Training Curves for PPO, SAC and TDMPC2 for 3B Steps

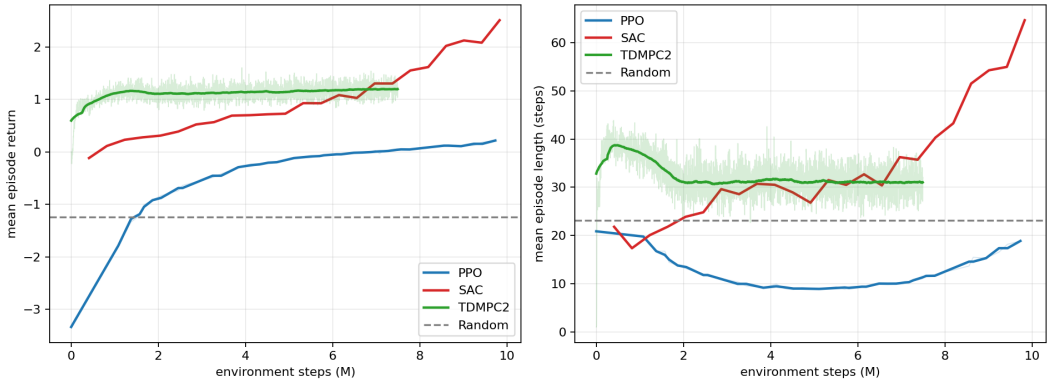

Figure 2: Training Curves for PPO, SAC and TDMPC2 for 10M Steps

Altogether, these results show that PPO converges to reliable long-horizon tracking behavior. For the remainder of the report, we therefore treat PPO as the strongest standard baseline and the main reference point for judging whether other methods can match full-horizon stability on this task. The evaluation results are provided in Table 1 and videos are available at our project website [4].

Table 1: Evaluation results for PPO

Metric	**Value**
Mean Return	46.061 ± 0.657
Mean Length	500.0 ± 0.0

3.2 SAC

We train SAC using the methodology described in Appendix D. We ablated our runs against gradient steps per iteration, entropy targets, the initialization of alpha, and the use of a distributional critic versus a scalar critic. Our best set of hyperparameter choices is presented in this section, while other findings are documented in the appendix.

We run our algorithm for 200,000 iterations, which corresponds to approximately 800 million environment steps, as we utilize 4096 parallel environments for rollout data collection. We observed that adaptive sampling negatively affects training performance; therefore, we only sample motion frames uniformly from the motion clip during training. However, our training runs suggest that the

Q-function estimates are occasionally highly inaccurate. This instability causes sudden drops in returns during training, which are visible in the learning curve. Despite these training difficulties, we successfully trained a policy that tracks the motion reference effectively. The evaluation results are provided in Table 2 and videos are available at our project website [4].

Table 2: Evaluation results for Vanilla SAC

Metric	**Value**
Mean Return	13.300 ± 29.033
Mean Length	237.2 ± 493.0

Although the controller has learned to walk, it fails to maintain balance if the robot is initialized at the first frame of the reference motion. This failure mode likely occurs because the controller encountered significantly fewer standing poses than moving poses during training. Consequently, the policy produces sub-optimal control signals in these specific states, leading to a fall.

Furthermore, although the control policy was trained for 800 million environment steps, it did not appear to converge to an optimal policy (see mean episode length). There are several potential reasons for this. First, as previously noted, the rewards were specifically tuned for PPO, which may cause SAC to converge toward a local optimum. Additionally, the decision to disable adaptive sampling for SAC may have led to slower convergence compared to the results seen with PPO in the BeyondMimic framework [8].

In order to address the instability observed during training, we modify both the architecture and the update procedure of vanilla SAC. While there is currently limited research addressing these stability issues in the context of whole-body tracking, some recent work has begun to explore efficient off-policy learning for such tasks [5]. Building on these insights, our specific improvements aim to stabilize learning in high-dimensional humanoid locomotion and are detailed in our first modification, Section 4.1

3.3 TD-MPC2

In stark contrast to PPO's sudden convergence to the maximum episode horizon, training for TD-MPC2 saturated very early. During initialization, the method experienced the same near-immediate failures, with episode lengths of zero. As optimization proceeded, the world model captured enough local dynamics to produce short periods of balance, causing episode lengths to rise and mean return to become positive. However, unlike PPO, the improvement saturated early. The episode length plateaued at ~ 40 and mean reward at ~ 1.3. We tuned the model by relaxing self-collision penalties, introducing regularization, and increasing MPC horizon lengths, but those changes barely improved performance. This suggests that the learned world model and short-horizon planner were sufficient for stabilization but inadequate for maintaining accurate whole-body reference tracking across long horizons.

We hypothesize that, while model-free algorithms like PPO can directly optimize the reward by iteratively shifting the policy distribution over dense on-policy rollouts, TD-MPC2 relies on explicit reward regression R_θ and latent dynamics unrolling. Regressing the multi-term Gaussian tracking reward from a compressed latent space z introduces approximation errors. Over the planning horizon, these errors can diverge from the true physical reward landscape, causing the MPPI planner to optimize a miscalibrated objective. Future work could address these limitations by bootstrapping the replay buffer with successful tracking trajectories generated by a pre-trained PPO policy. By seeding the buffer with expert demonstrations, the world model could learn accurate representations of high-reward states prior to MPPI exploration, avoiding the initial data-collection failures and breaking the negative feedback cycle.

3.4 Comparison of the Three Algorithms

As evidenced by the training curves and evaluation results, PPO outperformed both SAC and TD-MPC2 in our experimental suite. Although SAC is theoretically more sample efficient, it converged to a lower mean training return than PPO at 800M steps. Visual analysis of the generated gait patterns reveals that PPO produces qualitatively smoother and more accurate tracking compared to SAC. We attribute this primarily to the fact that the reward formulation was highly tuned for PPO. Despite our efforts to stabilize Vanilla SAC through architectural interventions, the performance gains remained marginal. This suggests that the inherent instability of critic learning in high-dimensional locomotion tasks and reward signal not optimized for off-policy algorithms, limits the algorithm's effectiveness in this specific task. We hypothesize that a reward design specifically tailored to SAC could further bridge this performance gap, as noted in [12].

TDMPC2 did not perform competitively in the whole-body tracking task within the scope of these experiments. The computational overhead of the MPC-based planning, combined with a lower number of parallel environments compared to the PPO and SAC runs, resulted in an exceedingly high wall-clock time (exceeding 20 hours per ablation). While TDMPC2 has demonstrated success in learning simplified walking controllers, its application to complex, high-DoF whole-body tracking remains an open challenge. We believe that with significantly more compute time and a scaling strategy that better leverages the massive parallelization available in mjlab, TDMPC2 could eventually converge toward an optimal world model and tracking policy.

4 Modifications

4.1 Updates to Soft Actor Critic to Stabilize Training

Vanilla SAC, as noted in Section 2, suffers from significant training instabilities in high-dimensional humanoid tasks. While prior work often utilizes layer normalization in the critic architecture to stabilize learning, we found that the standard "clipped double-Q" (minimum Q) approach was counterproductive to performance, a phenomenon also observed in [5]. Consequently, we modified the update rule to use average Q-values instead of the minimum. We refer to this implementation as MeanSAC.

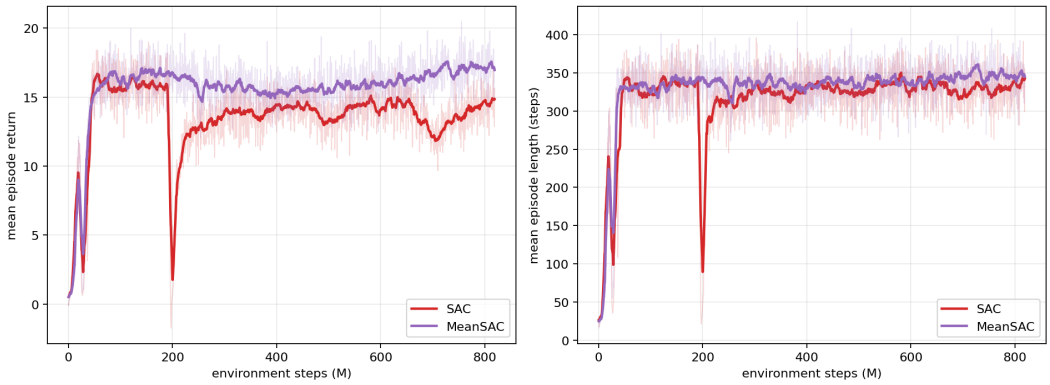

Figure 3: Training Curve Comparison between MeanSAC and Vanilla SAC

MeanSAC introduces two primary changes to the baseline SAC presented in Section 2. First, we add layer normalization to the critic architecture. Second, we utilize the mean of the two critic functions during updates rather than the minimum of the critic values. The combination of these changes has been shown to improve the performance of SAC in challenging tasks, such as whole-body tracking [5].

Table 3: Evaluation results for MeanSAC

Metric	Value
Mean Return	30.038 ± 60.537
Mean Length	490.1 ± 942.2

As observed in the learning curves and the evaluation results, MeanSAC performs significantly better than our vanilla SAC implementation. Specifically, the characteristic mid-training reward dips are eliminated. Furthermore, MeanSAC converges to a higher mean episode reward and achieves longer episode lengths. This indicates that using the mean Q-value provides a more stable target for the policy in this specific humanoid locomotion environment, leading to more stability compared to the vanilla implementation. However, we still observe a degree of sub-optimality in the policy, as it does not converge to high rewards within 800 million environment interactions. As previously noted, we attribute this behavior to the specific tuning of the reward functions for on-policy methods.

$$y = r + \gamma \left(\frac{Q_{\phi_1^-}(s', \tilde{a}') + Q_{\phi_2^-}(s', \tilde{a}')}{2} - \alpha \log \pi_\theta(\tilde{a}'|s') \right) \tag{4}$$

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\alpha \log \pi_\theta(a_t|s_t) - \frac{Q_{\phi_1}(s_t, a_t) + Q_{\phi_2}(s_t, a_t)}{2} \right] \tag{5}$$

4.2 Learning to walk using a musculoskeletal humanoid

A common research problem in biomechanics and robotics is understanding the underlying principles of human walking. As observed in our results, deep RL has demonstrated success in locomotion control for rigid-body humanoids, enabling them to walk, run, and produce natural behaviors. However, these humanoids do not accurately represent the human body, as they are controlled by electric motors rather than muscles.

To this end, we replace the G1 humanoid with a whole body musculoskeletal humanoid consisting of 354 muscles, with an observation dimension of 2418, which includes muscle states. We utilize the MuscleMimic [13] framework to train a PPO policy using MuJoCo as the physics engine. Because the musculoskeletal humanoid represents a significantly different embodiment compared to the G1, we employ different reward terms [13] scaled differently that are still rooted in the DeepMimic [14] and BeyondMimic [8] methodologies. We trained our policy for around 1 billion environment steps. While prior work suggests that over 10 billion steps are required to obtain a fully converged, optimal policy, we found that 1 billion steps were sufficient to produce a controller capable of walking and matching the reference motion effectively. The evaluation scores are provided in Table 4. The specific hyperparameter configuration is detailed in Table 13

Table 4: Evaluation results for Musculoskeletal Humanoid (PPO)

Metric	Value
Mean Return	405.073 ± 240.507
Mean Length	358.9 ± 213.7

Evaluation of the musculoskeletal policy reveals that while the controller can achieve a maximum episode length of 600 steps, the mean episode length converged to approximately 400 steps during the 1 billion environment interactions. Notably, this convergence occurred significantly below the 1,000-step truncation threshold established for training.

This performance plateau suggests a potential limitation in the model's capacity to represent the high-dimensional dynamics inherent in musculoskeletal control. The complexity of the observation and action spaces in this embodiment is substantially higher than that of the rigid-body G1 humanoid. Consequently, we hypothesize that the current actor architecture may be insufficient to

fully capture the intricate coordination required for long-term stability. Future work will investigate whether increasing the depth of the neural network or utilizing wider hidden layers can provide the necessary representational power to escape this local optimum and achieve full-length episode completion.

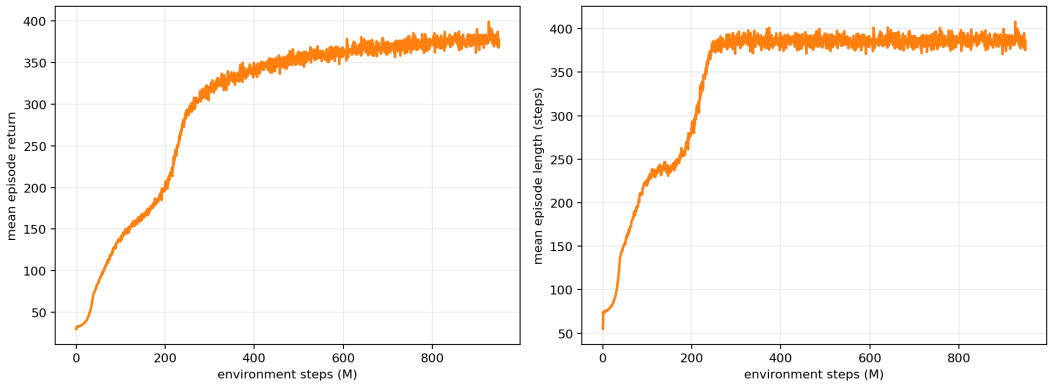

Figure 4: Training Curve for Musculoskeletal Humanoid Policy (PPO)

5 Conclusion

In this project, we evaluated the performance of PPO, SAC, and TD-MPC2 on a whole-body tracking task using the `mjlab` framework. By leveraging massive GPU parallelization, we trained controllers to track complex motion clips on the Unitree G1 robot. Our results show that PPO remains the most robust choice for stable locomotion in this setting, largely due to its established reliability in high-dimensional control and the availability of proven training configurations. In contrast, SAC and TD-MPC2 proved significantly more sensitive to the reward landscape, which was primarily optimized for PPO's on-policy updates.

We identified that the training instabilities in vanilla SAC were driven by critical Q-value misestimation. To mitigate this, we implemented MeanSAC, which utilizes an averaged Q-target and layer normalization to stabilize the critic. MeanSAC successfully eliminated mid-training reward collapses and provided a more stable learning signal for the tracking task. Furthermore, we demonstrated that the motion-imitation pipeline generalizes to extreme embodiments by training a musculoskeletal humanoid with 354 muscles. While the musculoskeletal policy successfully learned to walk, its performance plateaued at 400 steps, indicating that escaping local optima in muscle-driven systems requires higher model capacity and specialized reward density. Ultimately, this project demonstrates that while Deep RL can solve complex whole-body tracking across diverse humanoids, algorithmic success is critically dependent on the alignment between the reward structure and the specific update dynamics of the agent.

5.1 Future Work

The motion imitation framework established in this project can be extended to train musculoskeletal humanoids modeled after specific amputee subjects. Once a representative virtual human is established, it becomes possible to train prosthesis or exoskeleton controllers directly in simulation. This strategy provides two primary advantages: (1) it captures the personalized gait dynamics of the specific wearer, and (2) it removes the burden of exhaustive online tuning on the physical subject. However, the successful implementation of this pipeline requires minimizing the *Sim2Real2Sim* gap to ensure that the co-simulation dynamics are high-fidelity enough for safe and effective deployment on real-world hardware.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL https://arxiv.org/abs/1801.01290.

[3] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control, 2024.

[4] V. Kalva and D. Varun. Locomimic project website. https://sites.google.com/andrew.cmu.edu/locomimic/home, 2026.

[5] Y. Seo, C. Sferrazza, J. Chen, G. Shi, R. Duan, and P. Abbeel. Learning sim-to-real humanoid locomotion in 15 minutes, 2025. URL https://arxiv.org/abs/2512.01996.

[6] K. Zakka, Q. Liao, B. Yi, L. L. Lay, K. Sreenath, and P. Abbeel. mjlab: A lightweight framework for gpu-accelerated robot learning, 2026. URL https://arxiv.org/abs/2601.22074.

[7] Unitreerobotics. unitree_rl_mjlab: A reinforcement learning project built upon the mjlab, using mujoco as its physics simulation backend. URL https://github.com/unitreerobotics/unitree_rl_mjlab.

[8] Q. Liao, T. E. Truong, X. Huang, Y. Gao, G. Tevet, K. Sreenath, and C. K. Liu. Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion, 2025. URL https://arxiv.org/abs/2508.08241.

[9] Lvhaidong. LAFAN1_Retargeting_Dataset. https://huggingface.co/datasets/lvhaidong/LAFAN1_Retargeting_Dataset, 2024. Hugging Face dataset.

[10] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

[11] Y. Seo, C. Sferrazza, H. Geng, M. Nauman, Z.-H. Yin, and P. Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control, 2025. URL https://arxiv.org/abs/2505.22642.

[12] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

[13] C. Li, C. Wang, B. Ziliotto, M. Simos, J. Kovecses, G. Durandau, and A. Mathis. Towards embodied ai with musclemimic: Unlocking full-body musculoskeletal motor learning at scale. *arXiv preprint arXiv:2603.25544*, 2026.

[14] 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*, 37 (4):143:1–143:14, 2018. doi:10.1145/3197517.3201311. URL https://doi.org/10.1145/3197517.3201311.

[15] C. Schwarke, M. Mittal, N. Rudin, D. Hoeller, and M. Hutter. Rsl-rl: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025.

Appendix

A Observation and Action Spaces

Observation Space

The actor's observation vector is 160-dimensional and concatenates the current robot state, reference motion data, and an anchor pose error. Specifically, the observation includes a 58-dimensional command vector that provides reference joint positions and velocities as progress cues. Following the BeyondMimic framework, these are used for timing rather than as explicit kinematic targets. The proprioceptive state consists of the base linear and angular velocities (6D), joint positions and velocities (58D total), and the previous actions (29D). To assist with global balance, a 9-dimensional anchor pose error—comprising position and 6D orientation errors—is also included. For training, the critic utilizes an augmented 286-dimensional observation space that includes additional privileged information, such as the full body positions and orientations, to improve value function estimation. The observations are detailed in Table 5.

Action Space

The action space is a 29-dimensional continuous space where each dimension is normalized within the range of [-1,1], corresponding to each actuated joint. These actions are scaled and added to the default joint configuration to produce position setpoints for the low-level PD controllers. Rather than serving as high-precision tracking targets, these setpoints act as intermediate variables that shape the desired motor torques and are intentionally not clipped by joint kinematic limits to allow for greater compliance.

Table 5: Actor and Critic Observation Space details. Dimensions indicate the flattened vector size for each observation term.

Index	Observation Term	Actor (160)	Critic (286)
0	Command vector	58	58
1	Motion anchor position	3	3
2	Motion anchor orientation	6	6
3	Body position	–	42
4	Body orientation	–	84
5	Base linear velocity	3	3
6	Base angular velocity	3	3
7	Joint positions	29	29
8	Joint velocities	29	29
9	Previous actions	29	29

B Reward Function

The training objective is defined by a unified reward formulation consisting of Gaussian-shaped tracking scores and linear regularization penalties [8]. The tracking rewards for the humanoid body parts $\mathcal{B}_{target}$ are defined as follows:

Table 6: Reward formulation and weights for whole-body locomotion tracking.

Reward Term	**Equation**	**Weight**
Task (Tracking)		
Body Position	$\exp\left(-\frac{1}{\lvert\mathcal{B}\rvert}\sum\lVert\mathbf{p}_b^{des}-\mathbf{p}_b\rVert^2/0.3^2\right)$	1.0
Body Orientation	$\exp\left(-\frac{1}{\lvert\mathcal{B}\rvert}\sum\lVert\log(R_b^{des}R_b^\top)\rVert^2/0.4^2\right)$	1.0
Body Linear Velocity	$\exp\left(-\frac{1}{\lvert\mathcal{B}\rvert}\sum\lVert\mathbf{v}_b^{des}-\mathbf{v}_b\rVert^2/1.0^2\right)$	1.0
Body Angular Velocity	$\exp\left(-\frac{1}{\lvert\mathcal{B}\rvert}\sum\lVert\omega_b^{des}-\omega_b\rVert^2/3.14^2\right)$	1.0
Anchor Position	$\exp\left(-\lVert\mathbf{p}_{anchor}^{des}-\mathbf{p}_{anchor}\rVert^2/0.3^2\right)$	0.5
Anchor Orientation	$\exp\left(-\lVert\log(R_{anchor}^{des}R_{anchor}^\top)\rVert^2/0.4^2\right)$	0.5
Regularization		
Action Smoothness	$\lVert\mathbf{a}_t-\mathbf{a}_{t-1}\rVert^2$	-0.1
Joint Position Limit	$\sum_{j=1}^{N}[\max(l_j-\theta_j,0)+\max(\theta_j-u_j,0)]$	-10.0
Undesired Self-Contacts	$\sum_{b\in\mathcal{B}_{sc}}\mathbf{1}[\lVert f_b^{self}\rVert>1\text{N}]$	-0.1

The total reward r is the weighted sum of these components, as detailed in Table 6.

C PPO

C.1 Hyperparameters

Table 7: PPO training hyperparameters.

Hyperparameter	Value
Learning rate	1×10^{-3}
Learning rate schedule	Adaptive
Target KL	0.01
Discount γ	0.99
GAE λ	0.95
Clip range ϵ	0.2
Value loss coefficient c_v	1.0
Entropy coefficient c_e	0.005
Max gradient norm	1.0
Epochs per update	5
Mini-batches per epoch	4
Number of environments	4096
Rollout steps per environment	24
Total iterations	30,000

C.2 Network Architecture

Table 8: PPO network architecture.

Component	Input dim	Output dim
Actor layer 1	160	512
Actor layer 2	512	256
Actor layer 3	256	128
Actor mean head	128	29
Actor log std	–	29
Critic layer 1	286	512
Critic layer 2	512	256
Critic layer 3	256	128
Critic output	128	1

Both the actor and critic networks use `ELU` activations and incorporate observation normalization to handle varying state scales. The policy variance is parameterized by a learned state-independent log standard deviation vector, initialized to 1.0.

C.3 Implementation Details

The PPO algorithm is executed using the `rsl_rl` [15] library, which is optimized for GPU-accelerated synchronous training. During the data collection phase, on-policy trajectories are stored in a pre-allocated tensor buffer of shape $(T, N, \cdot)$, where $T = 24$ represents the rollout horizon and $N = 4096$ is the number of parallel environments.

To stabilize the policy gradient updates, advantages computed via Generalized Advantage Estimation (GAE) are standardized to have zero mean and unit variance within each minibatch prior to the network update. Furthermore, we apply a clipped value function loss to prevent the critic from diverging during aggressive policy steps.

Observations are normalized online using an exponential moving average (running mean and variance), which is updated continuously during the rollout phase but frozen during deterministic evaluation. Finally, the learning rate is dynamically adjusted at the end of each iteration; if the approximate KL divergence between the old and new policies exceeds 0.01, the learning rate is scaled down to enforce the trust-region constraint.

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize actor network π_θ and critic network V_ϕ
Initialize rollout storage $\mathcal{R}$ and adaptive learning rate $\eta = 1 \times 10^{-3}$
for iteration $= 1, 2, \ldots, 30000$ **do**
    Phase 1: Rollout Collection
    for $t = 1, 2, \ldots, 24$ (Steps per environment) **do**
        Run policy $\pi_{\theta_{old}}$ in 4096 parallel environments
        Collect states s_t, actions a_t, rewards r_t, and termination flags
        Store transitions in rollout storage $\mathcal{R}$
    end for
    Compute advantage estimates $\hat{A}_t$ using GAE ($\gamma = 0.99, \lambda = 0.95$)
    Compute value targets $V_t^{target} = \hat{A}_t + V_\phi(s_t)$
    Phase 2: Network Updates
    for epoch $= 1, \ldots, 5$ **do**
        for minibatch $\mathcal{B} \subset \mathcal{R}$ (4 mini-batches per epoch) **do**
            Calculate ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
            Compute surrogate loss:
            $L^{\text{clip}}(\theta) = \mathbb{E}\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$
            Compute value loss (clipped):
            $L_V(\phi) = \mathbb{E}\left[(V_\phi(s_t) - V_t^{target})^2\right]$
            Update θ, ϕ by maximizing $L^{\text{clip}}(\theta) - c_v L_V(\phi) + c_e \mathcal{H}(\pi_\theta)$
        end for
    end for
    Phase 3: Adaptive Schedule
    Compute KL divergence between π_θ and $\pi_{\theta_{old}}$
    if KL $>$ target_KL **then**
        Decrease learning rate η
    else if KL $<$ target_KL **then**
        Increase learning rate η
    end if
    $\pi_{\theta_{old}} \leftarrow \pi_\theta$
    Clear rollout storage $\mathcal{R}$
end for

D SAC

D.1 Hyperparameters

The hyperparameters for SAC are detailed in Table 9. These settings are optimized for high-throughput data collection in a massively parallelized GPU environment.

Table 9: SAC training hyperparameters.

Hyperparameter	Value
Hidden dimension (actor)	[512, 512]
Hidden dimension (critic)	[768, 768]
Learning rate	3×10^{-4}
Discount γ	0.99
Soft update rate τ	0.005
Batch size	8,192
Buffer size (per environment)	1,024
Initial temperature α	0.01
Target entropy ratio	0.5
Gradient steps per iteration	4
Policy update frequency	2
Learning starts (iterations)	10
Total learning iterations	200,000

D.2 Network Architecture

The actor and critic networks are implemented as multi-layer perceptrons (MLPs) with SiLU activations. The actor receives a 160-dimensional observation vector and outputs the mean and log standard deviation for the 29 action dimensions. The log standard deviation is clamped between -5.0 and 0.0 for stability. The critic architecture utilizes larger hidden layers to better approximate the value function in high-dimensional locomotion.

Table 10: SAC network architecture.

Component	**Input dim**	**Output dim**
Actor MLP (Hidden)	160	512×512
Actor Mean Head	512	29
Actor Log Std Head	512	29
Critic Q1 / Q2 MLP (Hidden)	315	768×768
Critic Output Head	768	1

D.3 Implementation Details

We began with a baseline custom implementation of SAC based on [2], originally designed for single-environment execution. To leverage the massive parallelization of mjlab, we modified the algorithm to efficiently process and store high-throughput batched data in the replay buffer. Inspired by TD3 [12], we implemented delayed actor updates, which significantly stabilized training by preventing the actor from prematurely exploiting critic mispredictions. Furthermore, we integrated a distributional critic in place of a scalar Q-function; this change notably improved training stability and minimized the severe reward dips observed in earlier iterations. We also identified the entropy temperature factor (α) as a critical tuning parameter for balancing exploration in this high-dimensional space.

During the evaluation of our vanilla SAC baseline, we observed a characteristic failure mode: a sharp decline in Q-values and mean rewards, even as the actor loss decreased monotonically. This divergence confirmed that the actor was exploiting a poorly estimated critic by learning sub-optimal actions. We hypothesized that this instability was driven by a distributional shift caused by the adaptive sampling mechanism in the BeyondMimic framework. While adaptive sampling focuses training on high-failure frames to create a curriculum, it corrupts the off-policy replay buffer with stale transitions. By reverting to a uniform sampling mode, we eliminated this shift, which resulted in significantly more stable training curves and better convergence.

Algorithm 2 Soft Actor-Critic

Initialize actor π_θ, critics Q_{ϕ_1}, Q_{ϕ_2}, target critics $Q_{\phi_1^-}$, $Q_{\phi_2^-}$, replay buffer $\mathcal{D}$
Initialize temperature $\alpha_0 = 0.001$
for each environment step **do**
    Sample action $a \sim \pi_\theta(\cdot|s)$
    Observe s', r, done
    Store $(s, a, r, s'$, done$)$ in $\mathcal{D}$
    for $j = 1$ to `gradient_steps` **do**
        Sample minibatch $\mathcal{B}$ from $\mathcal{D}$
        $\tilde{a}' \sim \pi_\theta(\cdot|s')$
        $y = r + \gamma \left(\min_i Q_{\phi_i^-}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right) (1 - \text{done})$
        Update critics: minimize $\sum_i (Q_{\phi_i}(s, a) - y)^2$
        $\tilde{a} \sim \pi_\theta(\cdot|s)$
        Update actor: minimize $\alpha \log \pi_\theta(\tilde{a}|s) - \min_i Q_{\phi_i}(s, \tilde{a})$
    end for
    Soft update: $\phi_i^- \leftarrow \tau \phi_i + (1 - \tau)\phi_i^-$
    Update α: minimize $-\alpha(\log \pi_\theta(\tilde{a}|s) + \mathcal{H}_{target})$
end for

E TD-MPC2

E.1 Hyperparameters

Table 11: TD-MPC2 training and planning hyperparameters.

Hyperparameter	Value
Training	
Learning rate (General)	3×10^{-4}
Learning rate (Encoder)	1×10^{-4}
Batch size	256
Replay buffer size	1×10^{6}
Updates per collect	16
Discount γ	0.99
Target network update rate τ	0.01
Temporal loss coefficient	0.5
Consistency loss coefficient c_1	20.0
Reward loss coefficient c_2	0.1
Value loss coefficient c_3	0.1
Termination loss coefficient c_4	1.0
Policy entropy coefficient β	1×10^{-4}
Max gradient norm	20.0
Number of environments	512
MPPI Planning	
Planning horizon H	5
MPC iterations	6
Total sampled sequences	512
Policy prior samples	24
Elite samples	64
MPC temperature	0.5

E.2 Network Architecture

Table 12: TD-MPC2 network architecture based on the 5M parameter configuration.

Component	Input dim	Output dim
Encoder (h_θ)	160 (State)	512
Latent Dynamics (d_θ)	512 (Latent) + 29 (Action)	512
Reward Predictor (R_θ)	512 + 29	101 (Bins)
Value Predictor (Q_θ) x5	512 + 29	101 (Bins)
Policy Prior (π_θ)	512	58 (2×29)
Termination Predictor (T_θ)	512 + 29	1

All fully connected layers across the components use `Mish` activations and Layer Normalization. The policy prior outputs both the mean and log standard deviation for the 29-dimensional action space. To enforce representation sparsity, the encoder and latent dynamics networks apply a SimNorm layer, which projects the 512-dimensional output into 64 partitions of 8-dimensional simplices with a temperature of $\tau = 1.0$. The value predictor utilizes an ensemble of 5 identical Q-networks with 1% dropout applied to the first linear layer.

E.3 Implementation Details

Our implementation is based on the original TD-MPC2 framework [3]. While the core world model design, SimNorm regularization, and discrete regression losses are preserved from the original paper, several modifications were made to adapt the algorithm to high-dimensional humanoid reference tracking.

Importantly, the original TD-MPC2 framework was designed primarily for single-threaded or small-scale vectorized environments (e.g., DMControl, Meta-World). To overcome the sample complexity required for humanoid motion tracking, we integrated the algorithm with `mjlab`[6], enabling synchronized data collection across 512 parallel GPU environments. To support this massive throughput, the replay buffer and data loaders were modified to ingest and sample contiguous trajectory sequences from batched tensors rather than individual environment transitions.

Because our problem is formulated as a single-task tracking environment, we omit the learnable task embeddings utilized in the multi-task generalist TD-MPC2 architecture. We also introduce an explicit binary cross-entropy termination head (L_{term}) into the world model to handle early episode cutoffs caused by the robot falling, an aspect less emphasized in the original TD-MPC2 benchmarks.

Finally, several hyperparameters were tuned to stabilize learning in this domain. To encourage robust representation learning across the 512 parallel environments, we use a hybrid acting scheme where a subset of environments (128) inject uniformly sampled noise into the planned MPPI actions to drive exploration. Furthermore, to stabilize the world model against the complex, dense tracking rewards defined in Section 1, we heavily weight the latent consistency loss ($c_1 = 20.0$) relative to the reward and value losses ($c_2 = 0.1, c_3 = 0.1$) and hypothesize that this will force the encoder to prioritize accurate physical dynamics over chasing raw rewards.

Algorithm 3 TD-MPC2: Model-Based RL with Latent Trajectory Optimization

Initialize world model $\theta \leftarrow \{h, d, R, Q, \pi, T\}$ and target networks Q^-
Initialize replay buffer $\mathcal{D}$ and action sequence parameters μ, σ
for iteration $= 1, 2, \ldots, N$ **do**
    Phase 1: MPPI Planning & Environment Interaction
    Encode current state: $z_t = h_\theta(s_t)$
    for MPPI iteration $= 1, \ldots, 6$ **do**
        Sample 24 action sequences $a_{t:t+H} \sim \pi_\theta(\cdot|z)$
        Sample 488 action sequences $a_{t:t+H} \sim \mathcal{N}(\mu, \sigma^2)$
        Rollout latent states $z_{t+1:t+H}$ using dynamics $d_\theta(z, a)$
        Estimate returns $J = \sum_{h=t}^{t+H-1} \gamma^{h-t} R_\theta(z_h, a_h) + \gamma^H Q_\theta(z_{t+H}, a_{t+H})$
        Select top 64 elite action sequences based on returns J
        Update μ, σ using the elite sequences
    end for
    Sample and execute first action $a_t \sim \mathcal{N}(\mu_0, \sigma_0^2)$
    Observe next state s_{t+1}, reward r_t, and termination flag
    Store sequence transition in replay buffer $\mathcal{D}$
    Phase 2: World Model & Policy Updates
    for update step $= 1, \ldots, 16$ **do**
        Sample sequence minibatch $\mathcal{B}$ from $\mathcal{D}$
        Encode states to compute targets: $z^{\text{target}}_{t:t+H} = h_{\theta^-}(s_{t:t+H})$
        Compute TD-targets y_t using reward and target value Q^-
        Rollout latents $\hat{z}_{t:t+H}$ using d_θ
        Compute Joint Loss:
            $L(\theta) = c_1 L_{\text{consist}}(\hat{z}, z^{\text{target}}) + c_2 L_{\text{reward}} + c_3 L_{\text{value}} + c_4 L_{\text{term}}$
        Update world model $\theta \setminus \{\pi_\theta\}$ to minimize $L(\theta)$
        Compute Policy Loss:
            $L_p(\phi) = -\mathbb{E}\left[\alpha Q_\theta(\hat{z}, \pi_\phi(\hat{z})) - \beta \mathcal{H}(\pi_\phi(\cdot|\hat{z}))\right]$
        Update policy prior π_ϕ to minimize $L_p(\phi)$
    end for
    Soft update target value networks $Q^- \leftarrow \tau Q_\theta + (1 - \tau) Q^-$
end for

F Musculoskeletal Humanoid

Table 13: PPO training hyperparameters for the musculoskeletal humanoid.

Hyperparameter	**Value**
Actor hidden layers	[1024, 1024, 1024, 1024, 1024] (5 layers)
Critic hidden layers	[1024, 1024, 1024, 1024, 1024] (5 layers)
Activation function	SiLU
LayerNorm	Enabled ($\epsilon = 10^{-5}$)
Number of parallel environments	8,192
Rollout length (n_{steps})	80
Total batch size	655,360
Number of minibatches	128
Minibatch size	5,120
Update epochs	1
Discount factor γ	0.99
GAE parameter λ	0.95
PPO clip range (ϵ_{clip})	0.2
Value function clip	0.2
Entropy coefficient	0.0
Initial action std	3.0
Observation normalization	Running mean/std