Mind the Phase: Effective Rank and Representation Health in Legged Locomotion

Felipe Andrade G. Tommaselli, Thiago Segreto, Juliano Negri, Ricardo V. Godoy, Marcelo Becker
Department of Mechanical Engineering, University of São Paulo, São Carlos, Brazil

Abstract: Reinforcement learning has become the leading paradigm in legged locomotion, enabling complex behaviors from backflips to parkour through massively parallel simulation. Under PPO's non-stationarity, shallow networks remain the de facto architecture, supported by carefully staged curricula and environments, yet the representations these policies learn stay poorly understood, leaving no training-time signal of how they will behave on hardware. In this work, we empirically study locomotion policies through the effective rank of the policy Jacobian and show that conditioning rank on the gait phase exposes architectural structure that global rank averages away. In particular, we find that standard architectural choices, namely layer normalization and residual connections, allocate roughly two more dimensions of effective rank to swing than to stance, which is fully absent in vanilla MLPs. Building on this, we propose a simple recipe that turns these representational signatures into smoother, more reliable sim-to-real transfer. In practice, this results in roughly $3\times$ lower joint jitter that holds from simulation onto a physical Spot, suggesting that representation health is an effective training-time lens to track sim-to-real smoothness.

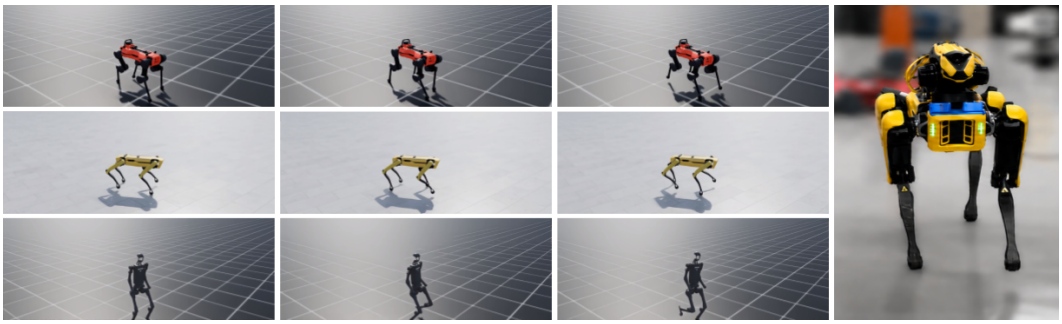

Figure 1: Left: locomotion rollouts for the three simulated embodiments studied here, two quadrupeds (top, middle) and a humanoid (bottom), with motion frames ordered left to right within each row. Right: our deeper, normalized-residual policy deployed on a physical Boston Dynamics Spot, walking on flat terrain.

Keywords: Reinforcement Learning, Representation Learning, Legged Locomotion

1 Introduction

Deep RL has been the backbone of current progress in robotics agility, unlocking previously unreachable acrobatic and athletic behaviors [1, 2]. The combination of scalable on-policy RL with a fast, parallelizable simulator has enabled zero-shot sim-to-real policies with super-natural capabilities [3]. This trend can be effectively understood through advances in domain randomization, curriculum learning, and policy distillation, which empirically push the neural network's plasticity bandwidth to its limit to enable the desired behavior.

Under this regime, preserving representational health is a persistent challenge. PPO's non-stationary objective erodes the network's capacity to fit new targets, and catastrophic forgetting compounds across curriculum stages [4, 5]. To contain this, the field has converged on shallow MLPs supervised almost entirely by reward curves, leaving the induced representations poorly characterized and offering no training-time signal predictive of hardware behavior.

Recent architectural advances reshape this picture. Layer normalization and residual connections are now standard, scaling depth and parameters while preserving optimization stability [6, 7, 8]. They implicitly raise the plasticity ceiling and improve asymptotic performance, yet their mechanism in legged locomotion stays opaque, with no diagnostic linking the representations they induce to motion on hardware.

In this work, we take a mechanistic view of these questions and study locomotion policies through the effective rank of the policy Jacobian, treating rank as a diagnostic of training health. Because the Jacobian relates small changes in the observation to changes in the action, it directly connects to action smoothness, a quantity that is easy to measure and strongly shapes how a policy behaves once deployed. This gives us a single lens that ties the network's internal state at training time to the smoothness we ultimately care about at inference time.

Crucially, we ground this analysis in classical principles of legged locomotion. A globally averaged rank turns out to be misleading, since it folds the distinct demands of the gait cycle into a single number. Once we condition the Jacobian rank on the gait phase and separate swing from stance, a clear architectural structure emerges that a global average washes out. Layer normalization and residual connections allocate roughly two extra dimensions of effective rank to swing relative to stance, a separation that is entirely absent in vanilla MLPs. Building on this signature, we propose a simple recipe that turns it into a smoother, more reliable sim-to-real transfer, cutting joint jitter by roughly $3\times$ as the policy moves from simulation to a physical Spot.

Concretely, we make three contributions.

- **A phase-conditioned rank diagnostic.** We analyze locomotion policies through the effective rank of the policy Jacobian and show that conditioning this rank on the gait phase exposes architectural structure that a global, phase-agnostic average obscures.
- **A characterization of architectural inductive biases.** We find that layer normalization and residual connections devote roughly two more dimensions of effective rank to swing than to stance, a gap that is fully absent in vanilla MLPs, tying these now-standard components to a concrete representational signature.
- **A recipe for smoother sim-to-real transfer.** We convert this signature into a simple training recipe that yields roughly $3\times$ lower joint jitter, holding from simulation onto a physical Spot, and we show that representation health serves as an effective training-time lens on sim-to-real smoothness, a quantity that covaries with deployed performance without being a target to maximize.

2 Related Works

Deep RL for legged locomotion has become the standard tool for legged control, often surpassing the capabilities of model-based controllers. Hwangbo et al. [1] trained a neural policy in simulation and transferred it to ANYmal, and Lee et al. [9] extended this to blind locomotion over challenging natural terrain through a teacher-student scheme. With massively parallel simulation [3, 10], the same approach now produces athletic behavior, including the diverse parkour skills of Zhuang et al. [2] and the extreme terrain traversal of Cheng et al. [11]. A recurring challenge in this progression is that policies trained purely on task rewards tend to exploit high-frequency control signals that are unachievable on hardware, potentially damaging actuators. The common fix is to penalize the rate of change of actions through reward shaping, as in Mysore et al. [12], which requires per-platform tuning of the reward weights. More recent work makes the smoothness objective differentiable and architectural. Chen et al. [13] imposes a Lipschitz constraint on the policy through a gradient

penalty on the action with respect to the observation, and Xie et al. [14] penalizes the full action Jacobian directly, observing that the former is equivalent to constraining only a single direction of that Jacobian. These methods act on the policy's sensitivity to its inputs, which is what the Jacobian rank we track captures.

Deep networks under PPO's non-stationarity. PPO's clipped probability ratio [15] limits how far each update can move the action distribution, but places no comparable limit on the underlying representation. In practice, the probability ratio routinely escapes the clip interval [16], while the unconstrained representation drifts under non-stationarity and does not recover [17]. The value-based literature traces this degradation to the spectrum of the representation itself, where features become progressively rank-deficient under bootstrapping [18], plasticity is lost over continual training [19], and the network loses the capacity to fit new targets [20]. Moalla et al. [21] extends this to PPO, showing that feature rank collapse and capacity loss accompany a degradation of the trust region. The right architectural priors keep high-capacity networks trainable, through residual blocks and normalization in SimBa [6], hyperspherical normalization in SimBaV2 [22], and bounded norms in FlashSAC [23].

Health metrics for legged locomotion. Before deep RL, gaits were described by a small set of geometric and temporal quantities. Raibert [24] decomposed dynamic running into the alternation of stance and swing and the placement of the foot within each cycle, a decomposition still implicit in how reward functions are written nowadays, for instance, the periodic stance-swing costs of Siekmann et al. [25].

3 Background

3.1 Reinforcement Learning

We formalize our RL setting as a discounted Markov decision process with states $\mathcal{S}$, actions $\mathcal{A}$, reward $r(s, a, s')$, and discount $\gamma \in (0, 1)$ [26]. At each step t the agent observes s_t, samples $a_t \sim \pi_\theta(\cdot \mid s_t)$ from a stochastic policy, transitions to s_{t+1}, and receives r_{t+1}, seeking to maximize the expected discounted return $J(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1}\right]$ over rollouts of length T.

Following the on-policy actor–critic pipeline of Rudin et al. [3], a policy network π_θ and a value network $\hat{v}_w$ are optimized jointly by a single Adam step over their concatenated parameters. Each iteration collects a rollout with the current policy π_{old}, computes advantages $\hat{A}_t$ with the Generalized Advantage Estimator [27] from γ and $\lambda \in [0, 1]$, and forms return targets $\hat{R}_t = \hat{A}_t + \hat{v}_{w_{\text{old}}}(s_t)$. PPO-Clip [15] then maximizes

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_{\pi_{\text{old}}}\left[\sum_{t=0}^{T-1} \min\Big(\rho_t \hat{A}_t,\ \text{clip}(\rho_t,\, 1-\varepsilon,\, 1+\varepsilon)\, \hat{A}_t\Big)\right], \qquad \rho_t = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}, \quad (1)$$

while the critic regresses on $\hat{R}_t$ with the update clipped within ε of $\hat{v}_{w_{\text{old}}}(s_t)$. The agent runs K epochs of minibatch gradient steps on the joint loss

$$\mathcal{L}(\theta, w) = -L^{\text{CLIP}}(\theta) + c_v\, \mathcal{L}^V(w) - c_H\, \mathcal{H}[\pi_\theta(\cdot \mid s_t)]\,, \quad (2)$$

weighting the value loss by c_v and the entropy bonus by c_H, and clipping actor and critic gradients to a fixed global norm. The learning rate is then adapted from the mean KL divergence between π_{old} and π_θ, shrinking by a constant factor when this KL exceeds twice a target d_{KL} and growing by the same factor when it falls below half of it.

3.2 Effective Rank

In this work, we transition among many expressive rank formulations in the broader deep learning community. The most widely used formulation in deep RL is termed Feature Rank, which measures the activations of the penultimate layer of the network. This represents the most abstract representation before the task-specific compression of the final layer and, therefore, exposes the very pulsating,

sensitive core of feature dynamics in this system. In all our calculations, we extract the entropy effective rank of the matrix, the penultimate layer, for instance, with the following procedure:

$$\mathrm{erank}(A) = \exp\left(-\sum_i p_i \log(p_i + \xi)\right), \tag{3}$$

where, for any matrix A with singular values $\{\sigma_i\}_{i=1}^r$, we discard numerically negligible modes $\sigma_i \leq \xi$, with $\xi = 10^{-10}$, and normalize the remaining spectrum as $p_i = \frac{\sigma_i}{\sum_j \sigma_j}$. This quantity can be interpreted as the entropy-equivalent number of active spectral modes [28]. A flat spectrum over k modes gives $\mathrm{erank}(A) \approx k$, whereas a spectrum dominated by a single singular value gives $\mathrm{erank}(A) \approx 1$.

Feature rank. The shared actor and critic penultimate-layer activations produce a feature matrix $F_\theta(X) \in \mathbb{R}^{N\times d}$ for a batch of N observations of dimension d. At each logging interval, we compute the SVD and apply 3, computing the feature rank. In addition, we also compute the termed PCA-99 rank with $r_{99}(A) = \min\left\{k : \frac{\sum_{i=1}^k \sigma_i^2}{\sum_j \sigma_j^2} \geq 0.99\right\}$, as the minimum k such that the top k singular directions explain at least 99% of the total variance (Frobenius energy). These procedures are often used in combination to understand the impact of the singular values alone (feature rank), and the impact of the smallest top k singular directions (PCA-99 rank), which can account for variance concentration.

Gram rank. For our qualitative representation, we showcase the Gram effective rank, calculated as the cosine-similarity Gram matrix of the fixed actor features. We define $G = \hat{F}\hat{F}^\top \in \mathbb{R}^{N\times N}$, where $\hat{F}_i = \frac{F_i}{\|F_i\|_2}$ is the L2 normalized F. Note that G is always square and therefore, can show structured proportions between different systems qualitatively. The rank is again computed with SVD.

Weight rank. We compute effective ranks of the actor's linear weight matrices $W_\ell \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$, termed Weight Rank. This can be interpreted as a non-representational parametric rank, which is especially useful for checking available bandwidth for downstream representations. For instance, a high representation rank ought to be uninformative if the weight rank has collapsed. Conversely, feature rank can collapse or not, even with a high weight rank.

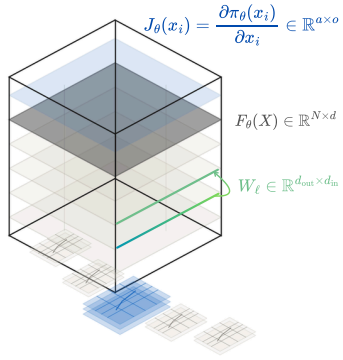

Figure 2: Overview of the rank objects studied here, namely the actor weight matrices W_ℓ, the penultimate feature representation $F_\theta(X)$, and the policy Jacobian $J_\theta(x_i)$ measuring local input-output sensitivity.

Policy Jacobian effective rank. We additionally consider the policy Jacobian effective rank, which measures the dimensionality of the policy's local input-output sensitivity in legged locomotion. For a deterministic actor mean $\pi_\theta(x) \in \mathbb{R}^a$ and observation $x_i \in \mathbb{R}^o$, the per-state Jacobian is $J_\theta(x_i) = \partial\pi_\theta(x_i)/\partial x_i \in \mathbb{R}^{a\times o}$. Given a batch of B evaluation states, we summarize the average sensitivity map as $\bar{J}_\theta = \frac{1}{B}\sum_{i=1}^B J_\theta(x_i)$ and define its effective rank as $r_{\text{Jac}} = \mathrm{erank}(\bar{J}_\theta)$. A low value indicates that the policy actions depend on only a few dominant observation directions, while a higher value indicates broader sensitivity across observation dimensions and action coordinates.

We additionally consider the local ranks $\mathrm{erank}(J_\theta(x_i))$ for individual states and report their aggregate behavior over the batch. This distinction is useful because $\mathrm{erank}(\bar{J}_\theta)$ measures the dimensionality of the average policy sensitivity, whereas the distribution of $\mathrm{erank}(J_\theta(x_i))$ measures how locally rich or collapsed the policy response is at each state. When phase labels are available, the same construction can be applied within each phase p using $\bar{J}_{\theta,p} = |\mathcal{I}_p|^{-1}\sum_{i\in\mathcal{I}_p} J_\theta(x_i)$, yielding a phase-conditioned sensitivity rank $\mathrm{erank}(\bar{J}_{\theta,p})$.

3.3 Gait Patterns

Each step in legged locomotion alternates between a *swing* phase, when a foot is off the ground, and a *stance* phase, when it bears load. Classical controllers built gaits around this cycle by hand, explicitly scheduling foot placement and contact timing. A reinforcement learning policy is given no such schedule and must recover the phase structure on its own, mapping joint proprioception, action history, and base orientation to joint-target adjustments.

The policy's input-output map changes during training, which is the object that the Policy Jacobian effective rank r_{Jac} summarizes. A trot coordinates the four legs through a small number of independent control directions, so we expect a well-trained gait to hold r_{Jac} in the range set by the actuated degrees of freedom, with the phase-conditioned rank $\mathrm{erank}(\bar{J}_{\theta,p})$ shifting across the cycle as legs load and unload. A degraded or reward-hacked gait should look different: the policy responds along a few dominant observation directions, r_{Jac} falls, and the lost control directions surface on hardware as phase misalignment. We test whether r_{Jac} tracks gait quality in this way and whether it does so consistently enough to act as a training-time signal.

The same reasoning should extend to systems with more degrees of freedom, such as humanoids, where balance is underactuated and upper-body momentum couples tightly to the stepping sequence.

4 Experiments

We begin our experiments by evaluating PPO configurations using RSL_RL [3] within IsaacLab [29], a standard platform for legged locomotion learning (Fig. 1). Proximal Policy Optimization (PPO) [15] remains the most widely used algorithm for on-policy RL, despite its well-known vulnerability to non-stationarity and trust-region collapses; we primarily focus on runs that remained stable (healthy) unless explicitly stated otherwise. We follow the experimental procedures of [21], where all healthy runs use five optimization epochs per rollout, which allows the policy to learn efficiently across all scenarios. For intentionally collapsed runs, increasing this value to 10 provides the agent with significantly more information per rollout, ultimately inducing a trust-region collapse.

We performed a learning-rate sweep across all architectures; however, in many cases, the same value proved optimal for models with similar parameter counts. Our improved architecture utilizes an in-house SimBa v1 [6] variant adapted for PPO, incorporating observation normalization and Pre-LN residual blocks. Additionally, we implement per-configuration branch scaling in the residual path to improve training stability and mitigate learning-rate sensitivity [30, 31]. Both architectures were spanned across 5 depths, and the performance remained comparable for our analysis (Fig. 10).

Our real-world experiments were conducted on the Boston Dynamics Spot platform [32] (Fig. 1), utilizing sim-to-real optimizations built upon the vanilla IsaacLab setup described in the appendix. Our generalization efforts extend to evaluations on the ANYmal-D [33] and Unitree H1 [34] platforms, with a primary focus on perceptive locomotion over rough terrain.

We run five independent seeds per hyperparameter configuration and report mean curves along with min/max shaded regions, unless specified otherwise.

Conditioning rank on gait phase exposes a swing-dominant split. To better understand the policy's underlying dynamics, we first examine the gait pattern immediately after training. Intuitively, legged locomotion relies on position control, while the Jacobian rank captures a quantity related to velocity. Therefore, the Jacobian rank also encodes information along this dimension.

During training, the Jacobian matrix evolves toward a more structured pattern, consistent with prior observations on representation matrices [28]. Figure 4 illustrates this transition from an untrained to a trained state, where a primitive form of input-output mapping emerges. We present this example from one of our deployed Simba policies as a qualitative visualization.

Simba-like policies devote about two more dimensions of effective Jacobian rank to swing than to stance, a consistently positive gap, whereas vanilla MLPs show no such preference and sit below

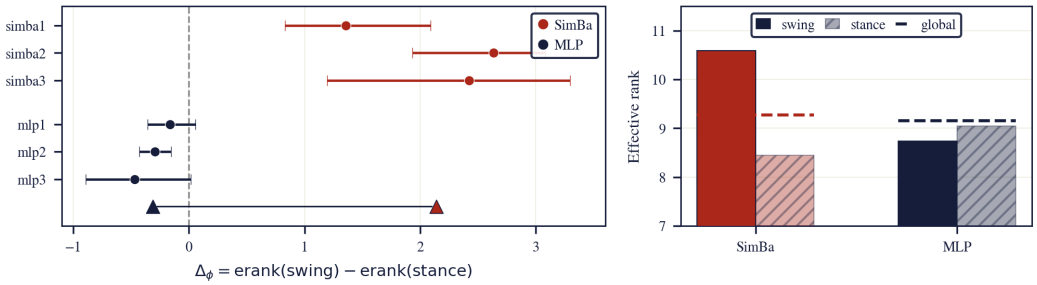

Figure 3: Phase-conditioned Jacobian rank separates the architecture families. (a) Phase gap $\Delta_\phi = \mathrm{erank}(J_{\mathrm{swing}}) - \mathrm{erank}(J_{\mathrm{stance}})$ on flat terrain. Every SimBa configuration is swing-dominant, and every MLP is not. (b) Pooled swing, stance, and global (dashed) Jacobian erank per family. The global values are nearly equal (9.3 vs 9.2), masking the split that phase conditioning exposes.

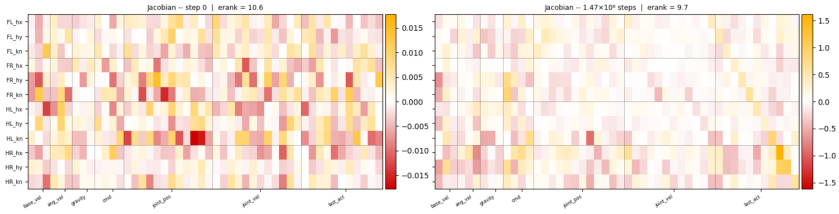

Figure 4: Jacobian matrix for the Simba policy. (a) Untrained matrix exhibiting near-random structure; (b) Trained matrix showing emergent primitive relationships.

zero, as shown in Fig. 3. The contrast is large in the pooled comparison ($\Delta\phi = +2.14$ vs. -0.31, Mann–Whitney $p = 2.5\mathrm{e}{-6}$, Cliff's $d = 0.98$) and unanimous across configurations, every SimBa run positive and every MLP run negative. A globally computed effective rank cannot see any of this, since averaging over the gait cycle aliases the two phases into a single number and washes out the split, which is exactly why we condition on phase throughout. The same swing-dominant structure reappears on the sim-to-real platform, confirming that it is a property of the learned policy rather than of a single simulator or terrain.

We observe that even on multiple platforms, the swing-dominant split persists. Measured as a global feature effective rank, the normalized-residual family carries a large and consistent edge over MLPs, roughly twenty extra dimensions on every robot we test (Fig. 5), and the same gap survives on perceptive rough terrain.

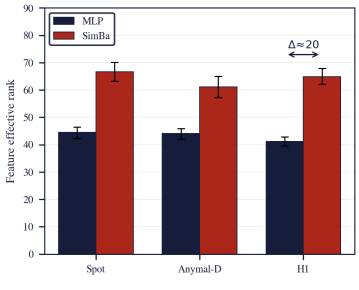

Figure 5: Global feature effective rank for MLP versus SimBa across all three robots, with SimBa holding a consistent $\Delta \approx 20$ gap. Bars show mean $\pm$ spread over runs.

Higher rank policies deploy more smoothly from simulation onto hardware. The representational gap separating the two families also tracks how the resulting policies behave once deployed (Figure 6). In simulation, the higher-rank (SimBa) family is roughly three times smoother than the MLP family, with substantially lower high-frequency joint jitter while matching or improving velocity tracking. The advantage survives the sim-to-real transfer. On the physical Spot, the deep MLP's jitter and tracking both collapse, while the SimBa policy stays smooth and accurate. We observed that the same smoothness advantage holds across robots, including the 19-DoF H1 biped.

We hypothesize that the implicit architectural nature, already well-known due to Simba's simplicity bias, leads to a strong correlation with greater smoothness. There is no proper artifact to link

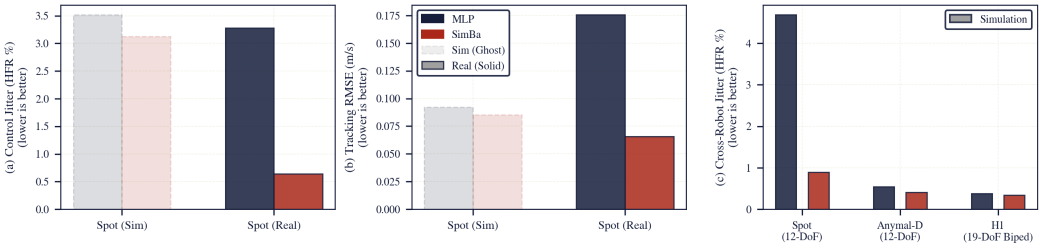

Figure 6: Sim-to-Real Transfer and Cross-Robot Generalization. (a) Spot joint control jitter % and (b) center-of-mass linear velocity tracking RMSE (m/s) in simulation (faded, ghost bars) and physical hardware deployment (solid bars) across MLP and SimBa architectures. (c) Simulation-only cross-robot control jitter across Boston Dynamics Spot (12-DoF), Anymal-D (12-DoF), and H1 (19-DoF biped). In physical hardware deployment, classical MLPs suffer severe velocity-tracking degradation and chattering, while SimBa maintains robust, low-jitter real-world transfer and generalizes consistently across quadrupeds and bipeds.

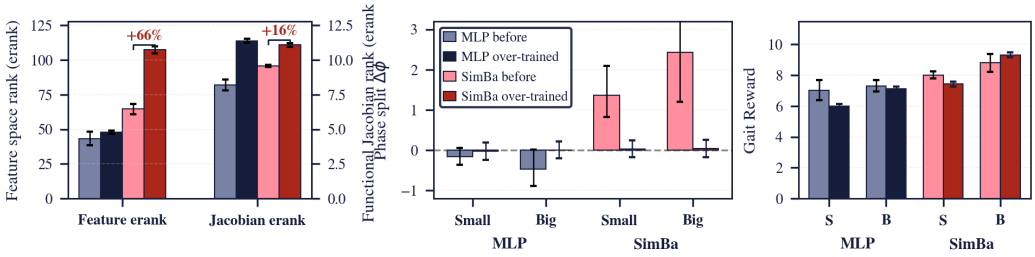

Figure 7: Representation collapse under over-training (healthy baseline desaturated, over-trained saturated). (a) Inflation: feature space erank (left axis) and functional Jacobian erank (right axis) inflate under over-training. SimBa feature rank explodes (+66%) while functional rank saturates. (b) Erasure: the specialized phase split $\Delta\phi$ collapses directly to the zero line for SimBa, while MLP remains flat near zero. (c) Blind reward: standard gait reward is insensitive to this representation collapse, with the Big SimBa config even showing a deceptive reward increase.

rank and smoothness directly, but we can sufficiently state that the co-occurrence phenomenon is constant, which is why we read rank as a diagnostic of representation health that covaries with deployment quality.

Collapse inflates effective rank, so a higher rank is not better. Our last experiment turns the diagnostic into a controlled perturbation, the closest thing to an intervention in the study. Over-training the same Spot-flat policies, by doubling the PPO epochs per update (recall: forcing collapse by construction), pushes the effective rank in two directions at once (Figure 7). The rank climbs, with the Jacobian erank saturating to a uniform ceiling across both swing and stance and the feature rank rising sharply, yet the swing-dominant split that characterized the normalized-residual policies collapses to essentially zero, while the MLPs, flat from the start, stay flat.

The architecture that once allocated rank by gait phase now spends it indiscriminately, so the higher rank coincides with a more degenerate representation, not a healthier one. This is our cleanest evidence that effective rank reports representation health as not a quantity to maximize, and it doubles as confirmation that the phase split of our first experiment is genuine structure, since a single training knob can erase it. We observed that the reward, meanwhile, is slow to register the change (Fig. 7). In-distribution tracking and reward stay largely unmoved, and one configuration's reward even rises, so the rank diagnostic catches an over-training regime that reward alone would miss.

Following [28], successful training regimes can often be interpreted as forming structured subsets in the Gram matrix. We illustrate this in Fig. 8: under healthy training (a), clear geometric structure emerges, whereas collapsed training (b) exhibits smooth, non-linear patterns.

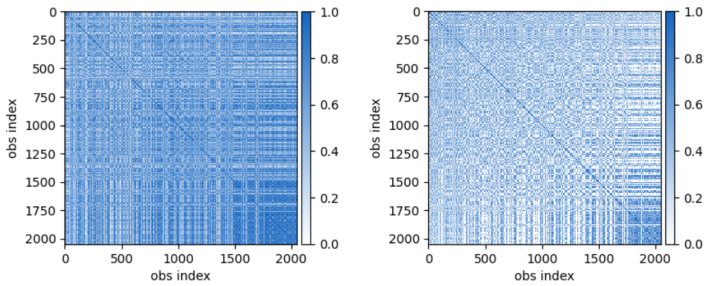

Figure 8: Gram matrix visualization under different training regimes. (a) Healthy training produces well-defined geometric structure, indicative of organized representation learning. (b) Collapsed training yields smooth, non-linear patterns, reflecting a loss of structure.

5 Limitations

As with many other empirical machine learning works, the generalization of our results and the isolation of our method are admittedly not fully characterized. In the results section, we used the best sim-to-real artifacts available to validate the generalizability of our findings. Still, we consistently observed scaling trends through architectures and embodiments, and given the current quality of physical simulation, we see a clear path for future iterations.

In methods isolation, our experimental design and the preference for flat blind locomotion were specifically designed to create an isolated testbed in which our results would be more expressive. However, many variables still remain. In this work, we did not include a variety of architectural choices (e.g., RNNs, Transformers), normalizations, or hyper-parameter changes. We couldn't explore actor–critic decoupling either, which, even with the shared trunk being a clear trend, this asymmetry could be further explored in future iterations.

6 Conclusion

We presented an analysis of PPO locomotion policies based on the effective rank of the policy Jacobian, and showed that the rank becomes a useful diagnostic only when conditioned on the gait phase. A phase-agnostic average is nearly identical across architectures, while separating swing from stance reveals that layer normalization and residual connections allocate roughly two extra dimensions of rank to swing, a structure absent in vanilla MLPs. We showed that this signature is a property of the learned policy: it survives sim-to-real transfer onto a physical Spot, and the same architectural edge appears in simulation on ANYmal-D and H1.

We further showed that the higher-rank family transfers roughly three times more smoothly from simulation to hardware, and that effective rank should not be maximized. Over-training inflates the rank while erasing the phase split, a regime that standard gait reward fails to detect, so we read effective rank as a diagnostic of representation health that covaries with deployment quality rather than as an objective.

Our results remain limited in generalization, highlighting clear directions for future work. We hope this study provides a strong signal for further exploration of mechanistic approaches to legged locomotion.

Acknowledgments

If a paper is accepted, the final camera-ready version will (and probably should) include acknowledgments. All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

References

[1] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019. doi:10.1126/scirobotics.aau5872. URL https://www.science.org/doi/abs/10.1126/scirobotics.aau5872.

[2] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot parkour learning, 2023. URL https://arxiv.org/abs/2309.05665.

[3] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on robot learning*, pages 91–100. PMLR, 2022.

[4] Z. Abbas, R. Zhao, J. Modayil, A. White, and M. C. Machado. Loss of plasticity in continual deep reinforcement learning. In S. Chandar, R. Pascanu, H. Sedghi, and D. Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 620–636. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/abbas23a.html.

[5] S. Dohare, A. R. Mahmood, and R. S. Sutton. Continual backprop: Stochastic gradient descent with persistent randomness. *CoRR*, abs/2108.06325, 2021. URL https://arxiv.org/abs/2108.06325.

[6] H. Lee, D. Hwang, D. Kim, H. Kim, J. J. Tai, K. Subramanian, P. R. Wurman, J. Choo, P. Stone, and T. Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. In *13th International Conference on Learning Representations, ICLR 2025*, pages 50050–50082. International Conference on Learning Representations, ICLR, 2025.

[7] K. Wang, I. Javali, M. Bortkiewicz, T. Trzciński, and B. Eysenbach. 1000 layer networks for self-supervised rl: Scaling depth can enable new goal-reaching capabilities, 2026. URL https://arxiv.org/abs/2503.14858.

[8] E. Daneshmand, S. Omar, G. Berseth, M. Khadiv, and H.-C. Lin. Slowrl: Safe low-rank adaptation reinforcement learning for locomotion. *arXiv preprint arXiv:2603.17092*, 2026.

[9] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020. doi:10.1126/scirobotics.abc5986. URL https://www.science.org/doi/abs/10.1126/scirobotics.abc5986.

[10] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

[11] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme parkour with legged robots, 2023. URL https://arxiv.org/abs/2309.14341.

[12] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko. Regularizing action policies for smooth control with reinforcement learning, 2021. URL https://arxiv.org/abs/2012.06644.

[13] Z. Chen, X. He, Y.-J. Wang, Q. Liao, Y. Ze, Z. Li, S. S. Sastry, J. Wu, K. Sreenath, S. Gupta, et al. Learning smooth humanoid locomotion through lipschitz-constrained policies. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4743–4750. IEEE, 2025.

[14] Z. Xie, K. Karol, and J. Hodgins. Learning smooth time-varying linear policies with an action jacobian penalty. *arXiv preprint arXiv:2602.18312*, 2026.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[16] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[17] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

[18] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *arXiv preprint arXiv:2010.14498*, 2020.

[19] S. Dohare, R. S. Sutton, and A. R. Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

[20] C. Lyle, Z. Zheng, E. Nikishin, B. Avila Pires, R. Pascanu, and W. Dabney. Understanding plasticity in neural networks. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 23190–23211. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/lyle23b.html.

[21] S. Moalla, A. Miele, D. Pyatko, R. Pascanu, and C. Gulcehre. No representation, no trust: Connecting representation, collapse, and trust issues in ppo. *Advances in Neural Information Processing Systems*, 37:69652–69699, 2024.

[22] H. Lee, Y. Lee, T. Seno, D. Kim, P. Stone, and J. Choo. Hyperspherical normalization for scalable deep reinforcement learning. In *International Conference on Machine Learning*, pages 33352–33403. PMLR, 2025.

[23] D. Kim, Y. Lee, M. Park, K. Kim, I. Nahendra, T. Seno, S. Min, D. Palenicek, F. Vogt, D. Kragic, et al. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. *arXiv preprint arXiv:2604.04539*, 2026.

[24] M. H. Raibert. *Legged robots that balance*. MIT press, 1986.

[25] J. Siekmann, Y. Godse, A. Fern, and J. Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 7309–7315. IEEE, 2021.

[26] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[28] M. Huh, H. Mobahi, R. Zhang, B. Cheung, P. Agrawal, and P. Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

[29] M. Mittal, P. Roth, J. Tigue, A. Richard, O. Zhang, P. Du, A. Serrano-Munoz, X. Yao, R. Zurbrügg, N. Rudin, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

[30] H. Zhang, D. Yu, M. Yi, W. Chen, and T.-Y. Liu. Stabilize deep resnet with a sharp scaling factor τ. *Machine Learning*, 111(9):3359–3392, 2022.

[31] B. Bordelon, L. Noci, M. Li, B. Hanin, and C. Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. In *International Conference on Learning Representations*, volume 2024, pages 22088–22127, 2024.

[32] Boston Dynamics. The spot robot platform. https://www.bostondynamics.com/products/spot, 2026. Accessed: May 29, 2026.

[33] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 38–44. IEEE, 2016.

[34] Unitree Robotics. Unitree h1: Full-size humanoid robot. https://www.unitree.com/h1, 2026. Accessed: May 29, 2026.

[35] A. Miller, F. Yu, M. Brauckmann, and F. Farshidian. High-performance reinforcement learning on spot: Optimizing simulation parameters with distributional measures, 2025. URL https://arxiv.org/abs/2504.17857.

A Additional Background

A.1 SimBa Implementation Details

Our implementation follows the SimBa architecture proposed by Lee et al. [6], comprising a linear embedding layer, N pre-layer-normalization (Pre-LN) residual blocks, a post-LN, and a linear output head. Each residual block applies $\mathbf{x} \leftarrow \mathbf{x} + \alpha\, W_2\, \mathrm{ReLU}(W_1\, \mathrm{LayerNorm}(\mathbf{x}))$, where $W_1 \in \mathbb{R}^{4d\times d}$ and $W_2 \in \mathbb{R}^{d\times 4d}$ expand and project the hidden dimension d.

The embedding and head weights are initialized orthogonally; residual branch weights use Kaiming normal initialization. To counteract gradient magnitude growth with depth, each block's residual contribution is scaled by $\alpha = 1/\sqrt{N}$, where N is the number of blocks. This depth-stable scaling is controlled by a `depth_scale` flag and is absent from the original SimBa formulation, which uses $\alpha = 1$.

We provide two integration modes. The first (`SimbaModel`) subclasses the RSL-RL `MLPModel` and replaces the internal MLP body (`self.mlp`) with the SimBa trunk, preserving the full observation-grouping, normalization, and distribution infrastructure of the base class. The second (`SimbaActorCritic`) subclasses `ActorCritic` directly, replacing `self.actor` and `self.critic` with separate SimBa trunks of configurable width and depth after the parent `__init__` has executed; asymmetric sizing is used (actor: $d = 256$, $N = 1$; critic: $d = 512$, $N = 2$). Both variants expose their hyperparameters through drop-in `@configclass` wrappers (`RslRlSimbaModelCfg`, `SpotLocomotionSimbaPPORunnerCfg`) compatible with the Isaac Lab configuration system.

The original SimBa paper prepends an RMSNorm to the trunk to handle heterogeneous observation scales. In our integration, this role is filled by RSL-RL's `empirical_normalization` flag, which maintains running statistics over observations and normalizes inputs at runtime, making an explicit first-layer RMSNorm redundant.

B Experiment Details

B.1 Code

We are making all training code open-source, including the infrastructure to run multi-thread IsaacLab across workers used.

The first version can be found at: `https://anonymous.4open.science/r/IsaacLab-F6E2`.

B.2 Hyper-parameters

All hyperparameters were set as close as possible to those of the original tasks. We varied only the learning rate for Simba, using 1.5×10^{-3}. For the MLPs, we used 1×10^{-3} for the three smallest models and 6×10^{-4} for the two largest.

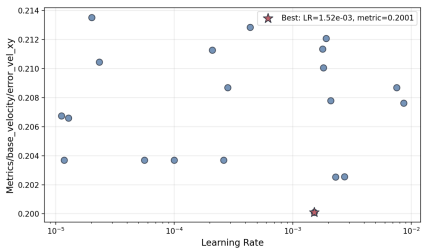

Figure 9: Bayesian optimization search for learning rate on Simba Architecture.

Figure 9 shows the learning rate sweep for Simba, with the top 10 sampled points displayed; in practice, we explored a broader range iteratively.

Both Simba and the MLPs were primarily optimized to maintain healthy representations across multiple depths, as illustrated in Fig. 10.

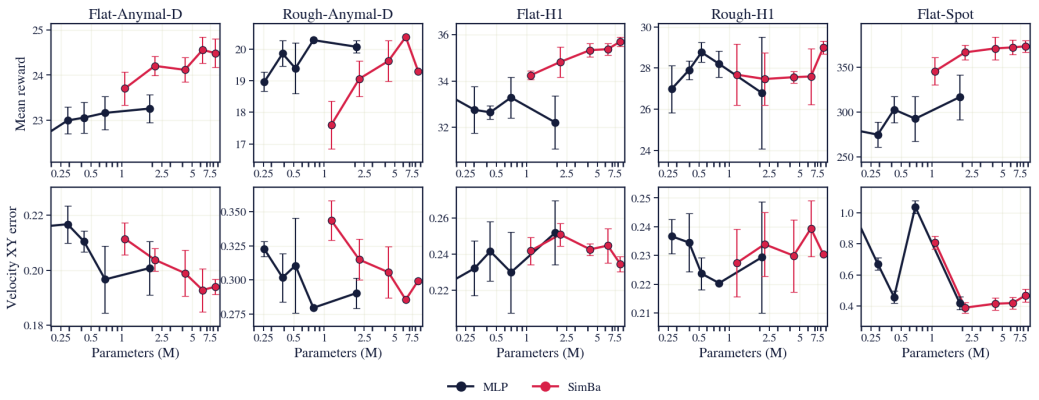

Figure 10: Overall performance across task depths. We compare the classical **MLP** baseline and **SimBa** architecture. Markers denote model capacity (parameters, M) in logarithmic scale. Both in their actuation range show similar performance, which is the necessary condition for our analysis.

C Real-world Experiments

C.1 Deployment Details

We deploy the policy on a Boston Dynamics Spot robot using the ONNX runtime for inference. Robot states are streamed from Spot's proprietary C++ SDK over gRPC, and actions are sent back through the same interface. Inference runs on an external desktop equipped with an Intel i7 CPU, connected to the robot via Ethernet. Under this setup, the policy operates with 50 Hz as a lower bound and end-to-end latency of 2–5,ms.

Figure 11: Boston Dynamics Spot used for the real-world experiments.

Spot does not have explicit binary contact switches in its rubber feet. Instead, standard pipelines (such as the Spot RL Researcher Kit [35]) reconstruct contact states physically:

$$\text{Contact State} = \begin{cases} 1 \text{ (Stance)} & \text{if } |\tau_{\text{knee}}| > 8.0 \text{ Nm} \\ 0 \text{ (Swing)} & \text{if } |\tau_{\text{knee}}| \leq 8.0 \text{ Nm} \end{cases}$$

Because the knee joint torque (τ_{knee}) is directly proportional to the vertical contact force F_z via the leg Jacobian, thresholding the knee torque provides a robust, physics-grounded contact sensor.

The policy is evaluated without command resampling under a scripted sagittal velocity profile:

$$v_{\text{cmd},x}(t) = \begin{cases} 0.0 \text{ m/s}, & \text{stance phase}, \\ +0.5 \text{ m/s}, & \text{forward walking phase}, \\ -0.5 \text{ m/s}, & \text{backward walking phase}. \end{cases}$$

The lateral and yaw commands are held at zero throughout the rollout,

$$v_{\text{cmd},y}(t) = 0, \qquad \omega_{\text{cmd},z}(t) = 0.$$

We use the same phase convention for visualization and downstream analysis as in the real-robot experiments. Timesteps with

$$|v_{\text{cmd},x}| \leq 0.1 \text{ m/s}$$

are labeled as standing, timesteps with

$$v_{\text{cmd},x} > 0.1 \text{ m/s}$$

as forward walking, and timesteps with

$$v_{\mathrm{cmd},x} < -0.1\ \mathrm{m/s}$$

as backward walking.

During each rollout, we log the commanded velocity, measured base velocity, projected gravity, joint positions, joint velocities, joint torques, foot contact states, foot heights, and phase identifiers. These trajectories are used to produce the visualizations and to compute phase-conditioned locomotion metrics such as velocity tracking, torso stability, cost of transport, and swing/stance gait statistics.

We deployed 10 policy configurations recording 81,313 steps of high-resolution telemetry logged at 50 Hz and isolating 193 steady-state forward gait strides. Tight 95% bootstrap confidence intervals confirm that our architectural scaling advantages transfer robustly to hardware.

Under steady-state forward locomotion, the 8.45M parameter SimBa-XXL policy achieves a velocity tracking RMSE of 0.066 m/s (95% CI: $[0.064, 0.067]$), outperforming the collapsed 1.89M MLP-XXL baseline (0.176 m/s, 95% CI: $[0.086, 0.265]$) by 62.7%. Furthermore, SimBa-XXL successfully mitigates joint-level jitter, restricting high-frequency ratio (HFR) to 0.64% (95% CI: $[0.61\%, 0.66\%]$) compared to 3.28% (95% CI: $[0.56\%, 6.00\%]$) for MLP-XXL, which exhibits severe physical leg chattering. This results could be understood as the lower-bound in efficiency and latency across all tested configurations.

C.2 Training Details

Our experimental setup closely follows the Isaac Lab defaults, with specific modifications to improve simulation-to-real transfer regarding style, posture, and stability. The default locomotion training environment uses flat-plane terrain; while a rough terrain generator is present in the configuration file, it remains disabled in the baseline setup. The simulation executes at 200 Hz, while the policy control loop operates at 50 Hz. Each training episode has a maximum duration of 20 s and terminates early if a timeout occurs or if any illegal contact involving the robot's main body or leg links is detected.

In addition to the reward tuning detailed in the following section, we apply a symmetry-based data augmentation strategy to ensure sample efficiency and preserve real hardware. With this setup, each collected transition is mirrored along the robot's sagittal plane via a left–right mapping of observations and actions [3], effectively doubling the usable batch size without requiring additional simulation time. This regularizes the policy, helping prevent overfitting or reward-hacking behaviors whose minor physical misalignments could potentially damage the hardware platform.

Furthermore, we designed an environmental curriculum spanning three registered environments. Progressing through these variants introduces an intentional weight curriculum that scales up selected penalty terms, most notably the base-orientation and contact-force penalties.

Observations are not dynamically normalized during training in the standard `rsl_rl` sense. Instead, normalization bounds are fixed based on prior empirical knowledge of the expected physical observation ranges. The complete set of architectural and algorithmic hyperparameters is reported in Table 1.

Reward Function The total reward $\mathcal{R}$ received at each timestep is a weighted sum of individual reward terms r_i, each scaled by a coefficient w_i:

$$\mathcal{R} = \sum_i w_i\, r_i. \tag{4}$$

The reward structure is designed to optimize stable, command-driven locomotion and mechanical efficiency while mitigating structural stress and unnatural postures. Crucially, the objective optimizes purely for locomotion tracking performance and explicitly excludes any arm reachability indices, end-effector trajectory tracking, or force-position manipulation targets. Each constituent term is detailed below.

Table 1: Network architecture and PPO hyperparameters.

Category	Parameter	Value
Rollout	Steps per environment	24
	Parallel environments	4,096
	Buffer size	$\approx 49{,}152$
PPO	Clipping parameter ϵ	0.2
	Value loss coefficient	1.0
	Entropy coefficient	0.005
	Learning rate schedule	Adaptive
	Target KL divergence	0.01
	Learning epochs per iteration	5
	Mini-batches per epoch	4
	Max. gradient norm	1.0
GAE	Discount factor γ	0.99
	GAE parameter λ	0.95

Base Linear Velocity Tracking: Rewards the agent for tracking commanded linear velocities along the x and y axes using an exponential kernel, with an additional scaling factor that increases the reward magnitude for higher commanded speeds:

$$r_{\text{lin\_vel}} = \exp\left(-\frac{\|\mathbf{v}_{xy}^{\text{cmd}} - \mathbf{v}_{xy}^{\text{base}}\|_2}{\sigma_{\text{lin}}}\right) \cdot \max\left(1,\ 1 + \gamma\left(\|\mathbf{v}_{xy}^{\text{cmd}}\|_2 - v_{\text{ramp}}\right)\right), \tag{5}$$

where $\mathbf{v}_{xy}^{\text{cmd}}$ is the commanded planar velocity, $\mathbf{v}_{xy}^{\text{base}}$ is the robot's measured base planar velocity in the body frame, σ_{lin} is the exponential kernel width, v_{ramp} is the velocity threshold above which the scaling activates, and γ is the ramp rate. This term has weight $w = 5.0$.

Base Angular Velocity Tracking: Rewards yaw rate tracking. The first uses a wider standard deviation for general tracking, and the second (finer) uses a narrower standard deviation for high-precision tracking:

$$r_{\text{ang\_vel}} = \exp\left(-\frac{|\omega_z^{\text{cmd}} - \omega_z^{\text{base}}|}{\sigma_{\text{ang}}}\right), \tag{6}$$

where ω_z^{cmd} is the commanded yaw rate and ω_z^{base} is the measured base yaw rate. Both terms carry weight $w = 5.0$, with different values of σ_{ang}.

Gait Reward: Enforces a trotting gait by synchronizing diagonal foot pairs (FL/HR and FR/HL). It is composed of two synchronization sub-rewards and four anti-synchronization sub-rewards:

$$r_{\text{gait}} = R_{\text{sync}} \cdot R_{\text{async}}, \tag{7}$$

where the synchronization reward for a pair (f_0, f_1) is:

$$R_{\text{sync}}(f_0, f_1) = \exp\left(-\frac{\text{clip}\left((t_{\text{air},f_0} - t_{\text{air},f_1})^2,\ \epsilon^2\right) + \text{clip}\left((t_{\text{con},f_0} - t_{\text{con},f_1})^2,\ \epsilon^2\right)}{\sigma_{\text{gait}}}\right), \tag{8}$$

and the anti-synchronization reward for an asynchronous pair (f_0, f_1) is:

$$R_{\text{async}}(f_0, f_1) = \exp\left(-\frac{\text{clip}\left((t_{\text{air},f_0} - t_{\text{con},f_1})^2,\ \epsilon^2\right) + \text{clip}\left((t_{\text{con},f_0} - t_{\text{air},f_1})^2,\ \epsilon^2\right)}{\sigma_{\text{gait}}}\right), \tag{9}$$

where $t_{\text{air},f}$ and $t_{\text{con},f}$ are the current air time and contact time of foot f, ϵ is the maximum clipping error, and σ_{gait} is the kernel width. The full gait reward is:

$$r_{\text{gait}} = R_{\text{sync}}^{(0)} \cdot R_{\text{sync}}^{(1)} \cdot \prod_{k=0}^{3} R_{\text{async}}^{(k)}. \tag{10}$$

This term is only applied when either the commanded velocity is nonzero or the body's planar speed exceeds a threshold v_{thr}. It has weight $w = 10.0$.

Foot Clearance Reward: Incentivizes the swinging feet to reach a target height h^* above the ground, weighted by the foot's planar velocity to avoid rewarding stationary feet:

$$r_{\text{clearance}} = \exp\left(-\frac{1}{\sigma_c} \sum_{f \in \mathcal{F}} \left(z_f - h^*\right)^2 \cdot \tanh\big(\alpha \|\mathbf{v}_{xy,f}\|_2\big) \right), \tag{11}$$

where z_f is the world-frame height of foot f, $h^* = 0.1\,\text{m}$ is the target swing height, α is the tanh scaling factor, σ_c is the kernel width, and $\mathcal{F}$ is the set of feet. This term has weight $w = 0.5$.

Action Smoothness Penalty: Discourages large instantaneous changes in the policy's output action vector $\mathbf{a}$:

$$p_{\text{smooth}} = \|\mathbf{a}_t - \mathbf{a}_{t-1}\|_2, \tag{12}$$

where $\mathbf{a}_t$ and $\mathbf{a}_{t-1}$ are the current and previous network actions. Weight: $w = -1.0$.

Air Time Variance Penalty: Discourages asymmetric foot timing by penalizing the variance in last air and contact times across feet:

$$p_{\text{air\_var}} = \text{Var}(\text{clip}(\mathbf{t}_{\text{air}},\, 0.5)) + \text{Var}(\text{clip}(\mathbf{t}_{\text{con}},\, 0.5)), \tag{13}$$

where $\mathbf{t}_{\text{air}}$ and $\mathbf{t}_{\text{con}}$ are vectors of the last air and contact times for all feet. Weight: $w = -1.0$.

Base Motion Penalty: Discourages excessive vertical linear and roll/pitch angular acceleration of the base:

$$p_{\text{motion}} = 0.8\,\dot{v}_z^2 + 0.2\left(|\dot{\omega}_x| + |\dot{\omega}_y|\right), \tag{14}$$

where $\dot{v}_z$ is the base vertical linear velocity and $\dot{\omega}_x$, $\dot{\omega}_y$ are the base roll and pitch angular velocities. Weight: $w = -2.0$.

Base Orientation Penalty: Discourages tilting of the robot's base by measuring the deviation from the gravity vector:

$$p_{\text{orient}} = \|\mathbf{g}_\perp\|_2, \tag{15}$$

where $\mathbf{g}_\perp \in \mathbb{R}^2$ are the x- and y-components of the projected gravity vector in the base frame (which are zero when the base is level). Weight: $w = -3.0$.

Foot Slip Penalty: Discourages feet from sliding while in ground contact:

$$p_{\text{slip}} = \sum_{f \in \mathcal{F}} \mathbb{1}[f \text{ in contact}] \cdot \|\mathbf{v}_{xy,f}\|_2, \tag{16}$$

where contact is determined by comparing the net contact force magnitude against a threshold. Weight: $w = -0.5$.

Joint Position Penalty: Encourages joints to remain near their nominal configuration. When the robot is at a standstill (low command and low body velocity), the penalty is scaled up by a factor κ:

$$p_{\text{jpos}} = \begin{cases} \|\mathbf{q}_{\text{leg}} - \mathbf{q}_{0,\text{leg}}\|_2 & \text{if } \|\mathbf{v}^{\text{cmd}}\| > 0 \text{ or } v_{\text{body}} > v_{\text{thr}}, \\ \kappa\,\|\mathbf{q}_{\text{leg}} - \mathbf{q}_{0,\text{leg}}\|_2 & \text{otherwise}, \end{cases} \tag{17}$$

where $\mathbf{q}_{\text{leg}}$ is the current joint position vector for the legs and $\mathbf{q}_{0,\text{leg}}$ is their default nominal configuration. Weight: $w = -0.7$.

Joint Velocity, Acceleration, and Torque Penalties: Reduces mechanical stress and promotes smooth motion, joint velocities, accelerations, and torques are penalized by their ℓ_2 norms:

$$p_{\text{jvel}} = \|\dot{\mathbf{q}}\|_2, \quad w = -0.01, \tag{18}$$

$$p_{\text{jacc}} = \|\ddot{\mathbf{q}}\|_2, \quad w = -10^{-4}, \tag{19}$$

$$p_{\text{jtor}} = \|\boldsymbol{\tau}\|_2, \quad w = -5 \times 10^{-4}, \tag{20}$$

where $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\boldsymbol{\tau}$ are the joint velocity, acceleration, and applied torque vectors across the active joints.

Contact Force Penalty: To actively protect structural elements from excessive impact loads during fast transitions or falls, vertical ground contact forces exceeding a safe nominal threshold are penalized:

$$p_{\text{contact}} = \sum_{f \in \mathcal{F}} \max\big(0, \|\mathbf{F}_f\|_2 - F_{\max}\big), \tag{21}$$

where $\mathbf{F}_f$ represents the measured contact force vector at foot f and $F_{\max}$ denotes the safety limit. This term is heavily scaled within environmental curriculum stages.

Reward Summary

Table 2 summarizes all reward and penalty terms with their respective weights.

Table 2: Reward and penalty terms of the locomotion training objective.

Term	Equation	Weight	Description
Base linear velocity	Eq. (5)	$+5.0$	XY velocity tracking
Base angular velocity	Eq. (6)	$+5.0$	Yaw rate tracking
Base angular vel. (fine)	Eq. (6)	$+5.0$	Yaw rate tracking (high precision)
Gait	Eq. (7)	$+10.0$	Enforces trotting via foot sync
Foot clearance	Eq. (11)	$+0.5$	Encourages 0.1 m swing height
Action smoothness	Eq. (12)	-1.0	Penalizes sudden action changes
Air time variance	Eq. (13)	-1.0	Penalizes uneven foot timing
Base motion	Eq. (14)	-2.0	Penalizes base undesired motion
Base orientation	Eq. (15)	-3.0	Penalizes base tilt
Foot slip	Eq. (16)	-0.5	Penalizes sliding feet
Joint position	Eq. (17)	-0.7	Encourages nominal leg joint configuration
Joint velocity	Eq. (18)	-0.01	Penalizes high joint speeds
Joint acceleration	Eq. (19)	-10^{-4}	Penalizes high joint accelerations
Joint torques	Eq. (20)	-5×10^{-4}	Penalizes high joint torques
Contact force	Eq. (21)	-10^{-3}	Penalizes excessive foot impact forces

Domain Randomization To narrow the sim-to-real gap, several domain randomization schemes are applied at different phases of the simulation runtime: at startup initialization, at individual episode resets, and periodically inside active environments.

At startup, four properties alter the structural dynamics of the robot asset and terrain interface. The ground surface friction coefficient is randomized, and the physical masses of the robot base, the attached arm, and the leg links are perturbed. Base and arm mass updates receive additive deviations, while leg segments undergo multiplicative scaling.

At each episode reset, three additional operations diversify the initial boundary state conditions. Random impulsive forces and torques are briefly applied to the robot base, the initial base yaw heading and planar coordinate placement are offset, and initial joint configurations are initialized with random noise distributed around their default nominal values.

Finally, while an episode is active, external push disturbances are periodically applied to the robot base at intervals ranging between 7 and 10 seconds. These force/torque perturbations and external pushes are fully phased into training using curriculum terms as policy optimization advances. Table 3 details the complete parameter sets.

Table 3: Domain randomization events applied during training.

Event	Stage	Range / Parameter
Ground Friction	Startup	Friction $\in [0.3, 1.0]$
Base Mass Perturbation	Startup	$\Delta m \in [-2.5, 2.5]$ kg
Arm Mass Perturbation	Startup	$\Delta m \in [-0.5, 0.5]$ kg
Leg Mass Scaling	Startup	Scale $\in [0.8\times, 1.5\times]$
Impulsive Wrench	Reset	$F = 10$ N, $\tau = 2$ Nm
Base Pose Reset	Reset	Yaw $\in [-\pi, \pi]$, Pos $\in [-0.5, 0.5]$ m
Joint Initialization	Reset	Sampled around nominal configuration
Base Velocity Push	Interval	Every 7–10 s (via curriculum)

Observation Space At each control step, the policy receives a concatenated observation vector $\mathbf{o}_t \in \mathbb{R}^{65}$. To regularize the policy against sensor delays and estimation errors, observation corruption via additive zero-mean Gaussian noise is enabled throughout training. The full observation vector decomposes into the following modalities:

- **Base state**: Linear velocity $\mathbf{v}^{\text{base}} \in \mathbb{R}^3$, angular velocity $\boldsymbol{\omega}^{\text{base}} \in \mathbb{R}^3$, and the gravity vector projected into the body frame $\mathbf{g}_b \in \mathbb{R}^3$.
- **Command**: Planar base velocity targets $(v_x^{\text{cmd}}, v_y^{\text{cmd}}, \omega_z^{\text{cmd}}) \in \mathbb{R}^3$, commanded base height target $h^{\text{cmd}} \in \mathbb{R}^1$, and target base orientation components (roll and pitch) $\boldsymbol{\theta}^{\text{cmd}} \in \mathbb{R}^2$.
- **Joint state**: Relative joint positions $\mathbf{q} - \mathbf{q}_0 \in \mathbb{R}^{19}$ and joint velocities $\dot{\mathbf{q}} \in \mathbb{R}^{19}$, which map across the 12 leg joints and 7 arm joints.
- **History**: The previous execution step's 12-dimensional policy action command $\mathbf{a}_{t-1} \in \mathbb{R}^{12}$.

The specific multi-command environment generates target updates every 10 s. Forward velocity is sampled within $[-1.0, 1.0]$ m/s, lateral velocity within $[-0.5, 0.5]$ m/s, yaw rate within $[-1.0, 1.0]$ rad/s, and heading angle targets within $[-\pi, \pi]$ rad, with explicit heading-tracking mode enabled for exactly half of the active environments. Base orientation commands sample roll and pitch ranges within $[-0.5, 0.5]$ rad, enforcing a level flat-orientation constraint (0 rad) for 40% of the parallel tracks. Base height targets are uniformly sampled within the range $[0.5, 0.7]$ m.

Action Space The policy action space $\mathbf{a}_t \in \mathbb{R}^{12}$ is defined as a 12-dimensional vector corresponding to joint-position targets for the active Spot leg joints. These targets are scaled by a factor of 0.2 and added as angular displacement deltas around the nominal leg joint offsets $\mathbf{q}_{0,\text{leg}}$. Although the simulated physical asset represents the full Spot quadruped complete with its attached arm system, the locomotion policy does not command arm motion. Instead, the 7 arm joints are maintained at a rigid, fixed default reset configuration using a constant joint-position control loop with zero action dimension, ensuring they do not consume policy outputs or modify the action space.